PsychoPy Builder で作る心理学実験

リリース *5.1*

十河宏行

2024 年 04 月 22 日

目次

はじめに

前回の改定 (2021 年 3 月) から 1 年半近く経ちました。その間に初版から 第 6 章 で使い続けてきた RatingScale コンポーネントが廃止されるという大きな変更がありました。厳密に言うと互換性のために残されていはいるのですが、標準の設定では Builder のコンポーネントペインに表示されなくなっています。とりあえずこの変更に対応しなければと思って原稿を読み返してみると、他にも対応していない変更点が思いのほかたくさん見つかったため、全体的に修正を加えることにしました。主な変更点は以下の通りです。

- 図に含まれるスクリーンショットのうち、2022.2.4 との違いが大きいものを更新しました (全般)。
- プロパティダイアログのタブ名の変更に対応しました (全般)。
- 単位として pix を用いていたところを現バージョンの事実上の標準単位である height へ変更しました (全般)。
- 第 3 章の xlsx 形式データファイルの確認部分を除いて、xlsx 形式のデータファイルを使用しないようにしました (全般)。
- **[キーボードバックエンド]** を ioHub に設定しているときに、Mouse オブジェクトの setVisible() メソッドでカーソルを非表示に設定できない場合があることを注記しました (2 章,8 章)。
- コンポーネントペインに追加されたカテゴリについての記述を追加しました (2 章)。
- 実験情報ダイアログの participant にランダムな数値が自動入力されるようになったことに対応しました (2 章,3 章)。
- ルーチンペインに配置したコンポーネント上で右クリックした時のメニューに対象のコンポーネント名が表示されるようになったことに対応しました (2 章)。
- 視覚刺激の **位置揃え** に対応しました (2 章)。
- Mac 上で Text コンポーネントの日本語フォントがおかしくなる現象について追記しました (2 章)。
- カラーピッカーの仕様変更に対応しました (2 章)。
- 2022.1.1 以降、色の指定で RGB 値を書くときに Python の式とせず-1.0,0.0,1.0 のようにカンマ区切りで三つの値を書いても認識するようになったことに言及しました (2 章)。
- 実験設定ダイアログの「スクリーン番号の表示」ボタンについての記述を追加しました (2 章)。
- 実験設定ダイアログに新設された「アイトラッキング」、「入力」タブについての説明を追加しました (2 章)。
- Polygon コンポーネントの **[形状]** の円に対応しました (3 章)。
- ルーチン新規作成時のダイアログに追加された **[テンプレート]** について補足しました (3 章)。
- 小さな画面の PC で実行する場合に実際より寸法が大きいモニターを設定して回避する方法を紹介しました (4 章)。
- 右クリックで表示されるコンテキストメニューからのコンポーネントの貼り付けに対応しました (4 章)。

- RatingScale コンポーネントは以前から非推奨でしたが、ついに Builder のコンポーネントペインに表示されないようになってしまったので RatingScale を使う部分を削除しました (6 章)。
- Image コンポーネントの **[サイズ [w, h] $]** を空欄したときの動作についての解説を追加しました (6 章)。
- Image コンポーネントで単色の長方形を描く機能について、解説していた方法が現行のバージョンでは使えなくなっていること、長方形を描くだけなら Polygon コンポーネントを使うべきこと、マスクと組み合わせるテクニックはあまり必要とする人がいないと思われることなどから解説を削除しました。
- ミュラーリヤー錯視実験の条件ファイルの列名を旧版から変更した点について、解説はもう不要だろうといことで削除しました (7 章)。
- Button コンポーネントの **[フォント]** に関する注意を追記しました (9 章)。
- Button コンポーネントの **[クリック毎に 1 回実行]** は **[Routine を終了]** がチェックされていると無効になることを追記しました (9 章)。
- RatingScale コンポーネントの廃止に伴い、Slider コンポーネントの解説を書き直しました (9 章)。
- サンプルプログラムにおいて Slider コンポーネントによる色の変更ができない場合の原因と解決例について補足しました (9 章)。
- TextBox コンポーネントを使った文字入力についての解説を追加しました (9 章)。
- Form コンポーネントの仕様変更に伴い Form コンポーネントの解説を書き直しました (9 章)。
- form オブジェクトの reset() メソッドを使ってループ内で Form コンポーネントを繰り返し使用できることに言及しました (9 章)。
- Movie コンポーネントのバックエンドについての記述を更新しました (10 章)。
- Microphone コンポーネントについての解説を追加しました (10 章)。
- Camera コンポーネントについての解説を追加しました (10 章)。
- 未解説のコンポーネント一覧を更新しました (付録)。
- PsychoPy の設定ダイアログの解説を更新しました (付録)。

本書の実験で使用している画像ファイルや実験ファイルは、筆者の web ページ (http://www.s12600.net/psy/python/ppb/) でダウンロードできますのでご利用ください。誤字や内容の誤りの指摘、その他内容についての要望などございましたら、十河までご連絡いただけましたら幸いです。

2022 年 9 月 6 日 十河 宏行

2023 年 7 月に以下の変更を行いました。

- 第 4 章の実験において、標準の単位を cm としていましたが height に変更しました。これに伴い刺激のサイズなどのパラメータを見直しました (4 章)。

2023 年 7 月 14 日 十河 宏行

2024 年 4 月 5 日に PsychoPy バージョン 2024.1.1 がリリースされました。まだ動作が安定ですが、今後の Builder による実験作成を考えるうえで非常に重要な機能が追加されました。本来なら本文全体を見直して第

6 版を作成すべき状況なのですが、そのための時間を確保することが当分できそうにないので、第 5 版への付録として 2024.1.1 の新機能を解説することにしました。2024.1 系を使用する人はぜひ付録にも目を通してください。

2024 年 4 月 22 日 十河 宏行

第 1 章

PsychoPy の準備

1.1 PsychoPy ってなに？

PsychoPy とは、パーソナルコンピューター (PC) を使って心理学実験を行うためのツールです。心理学実験では刺激画像をぴったり 0.2 秒間提示するとか、刺激音声が提示されてからボタンを押すまでの反応時間を記録するとか、数十種類の刺激を無作為な順番で 10 回ずつ提示するといった作業が求められることがありますが、こういった作業を手作業で行うのは困難です。PsychoPy を使うと、PC を用いて画像刺激や音声刺激を提示したり、反応時間を計測したり、刺激の提示順序を自動的に制御したりすることができます。

PsychoPy では、Python というプログラミング言語を使って PC に指示を出します。本来 PC に指示を出すには機械語と呼ばれる言語を使わないといけないのですが、機械語は一見ただの数値の羅列で普通の人が読んだり書いたりすることはできません。プログラミング言語は、人間の言語 (たいてい英語なのですが) にちょっと似た雰囲気で読み書きできる言語で、インタプリタやコンパイラと呼ばれるソフトウェアを使って機械語に翻訳することができます。プログラミング言語や機械語で書かれた PC への指示をプログラムと呼びますが、Python のプログラムはスクリプトと呼ばれることもあります。スクリプトって何？という疑問へのお答えはちょっと難しい話になるので「1.3.1: コンパイラとインタプリタ、そしてスクリプト 」を参照してください。

本題に戻りまして、PsychoPy は Python 用のライブラリをまとめたパッケージです。ライブラリとは、PC にいろいろな指示を出すうえでよく使われる手続きなどをまとめたものです。例えば、皆さんが使っている PC のソフトウェアでは「OK」とか「キャンセル」とか書かれたボタンが表示されて、その上にマウスカーソルを動かしてマウスのボタンをクリックするとボタンに応じた動作が行われるでしょう。この時、ソフトウェアを作る人は「マウスカーソルがボタンの上に乗っているか」、「マウスのボタンは押されているか」、「ボタンが押されたらどのような動作を実行するか」といった事を PC に処理させるプログラムを書かないといけません。しかし、これらの処理はどのようなソフトウェアのボタンでも共通しているので、この共通する動作をまとめて「ボタンに関する処理」という名前で呼び出せるようにしておけば、ソフトウェアのプログラムを書くときに「ボタンに関する処理を行う」と書くだけでまとめられた処理を全て PC に行わせることができます。「ボタンに関する処理」の他にもたくさんの共通機能をまとめたものライブラリであり、ライブラリを用いることによって定番の処理を書くことに煩わされずに効率よく自分のプログラムを書くことができるのです。Python では、ライブラリの個々のパーツをモジュールと呼び、モジュールの集合体をパッケージと呼んでいます。ライブラリ、モジュール、パッケージ、と耳慣れない用語が続きますが、とりあえずこれらは同じような意味だと思っておいていただいて問題ないと思います。

PsychoPy は心理学実験で使われる定番の処理、すなわち画面に刺激を描いたり、音声を鳴らしたり、反応時

間を記録したりするための処理などをまとめた Python のパッケージです。Python のパッケージですから、利用するためには Python のプログラムを書けなければいけません。Python 以外の言語でプログラムを書いた経験がある人が Python に乗り換えるのはそれほど難しくありませんが、プログラムを書いた経験がほとんどない人にはなかなか厳しいハードルです。そこで、PsychoPy にはウィンドウ内に刺激提示や反応計測を行うためのアイコンを並べることによってプログラムを書かずに実験を作成するアプリケーションが用意されています。このアプリケーションを *PsychoPy Builder* と呼びます。PsychoPy Builder はアイコンを並べて作成した実験を Python のスクリプトに変換してくれます。自分で書いた Python のプログラムを実行する場合は *PsychoPy Coder* と呼ばれるアプリケーションを使用します。さらに、Builder や Coder で作成した実験を実行するための *Runner* と呼ばれるアプリケーションが用意されています。以下、PsychoPy Builder のことを単に Builder、PsychoPy Coder のことを Coder と表記します。

本書では、Builder を使って心理学実験を作成する方法を解説します。非常に複雑な手続きの実験を行うには Coder を使って直接プログラムを書く必要があるかも知れませんが、工夫をすればさまざまな実験を Builder で行うことができます。

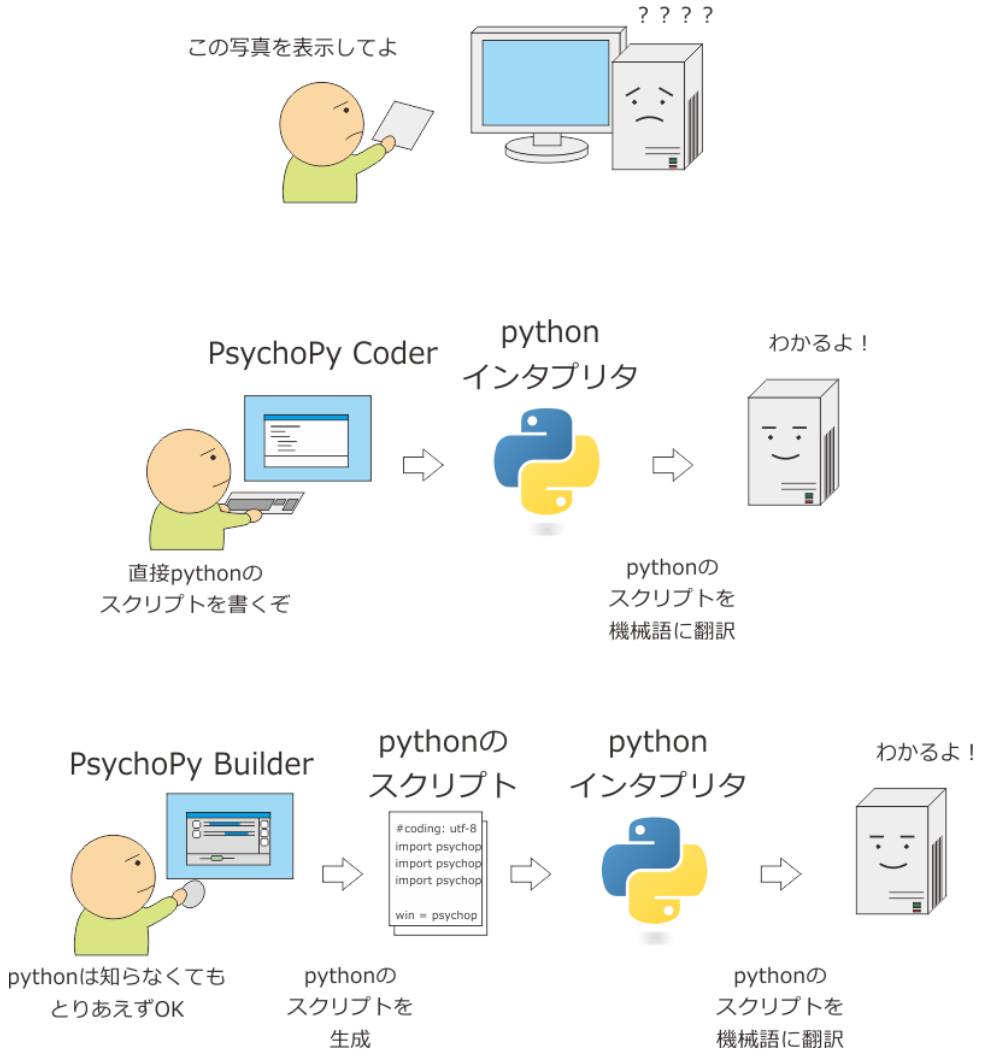


図 1.1　PsychoPy は Python という言語を用いて PC に指示を出すためのソフトウェアです。Python のスクリプトを直接書いて使う PsychoPy Coder と、グラフィカルなインターフェースで利用できる PsychoPy Builder があります。

1.2 PsychoPy をインストールしよう

PsychoPy をインストールする方法は複数ありますが、初心者向けで最も簡単なのは Standalone インストーラーを用いる方法です。PsychoPy を動かすには数多くの Python パッケージが必要で、本来であれば PsychoPy をインストールする前に Python 本体とそれらのパッケージのインストーラーをすべて web 上からダウンロードするなどして集めてひとつひとつインストールしていかなければいけません。Python に慣れていない人にとってこれは大変な作業です。Standalone インストーラーは、これひとつで Python 本体と PsychoPy の動作に必要なパッケージをまとめて一度にインストールしてくれるという優れものです。Standalone インストーラーは GitHub という web サイトでダウンロードすることができます。URL は以下の通りです。

- https://www.psychopy.org/download.html

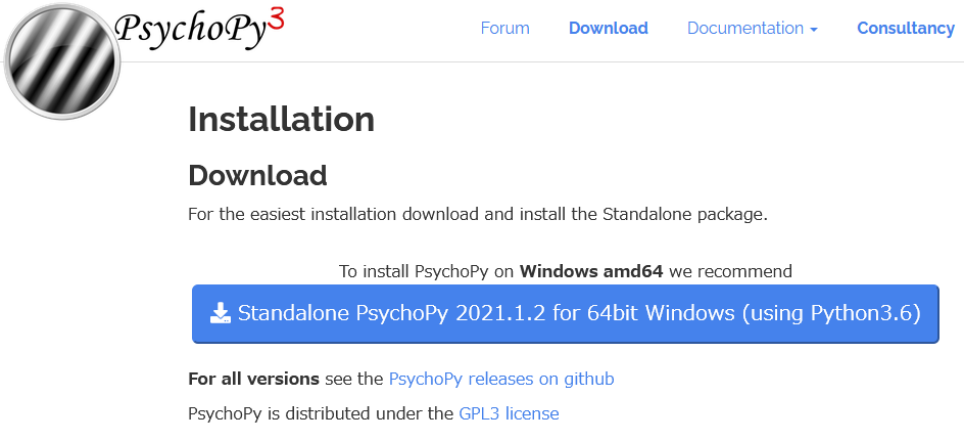


図 1.2　PsychoPy のダウンロード画面。使用している PC に合わせてお勧めのバージョンをダウンロードするボタンが表示されます。

図 1.2 はダウンロードページを開いた様子です。大きく青色で表示されているボタンをクリックすると、使用している PC に最適なバージョンのインストーラーのダウンロードが始まります。通常はこの「お勧め」のバージョンを使用すれば問題ありませんが、「大学の研究室では古いバージョンを使用していて、それと同じものを使用したい」とか「実験に使用する装置が古い Python でなければ動かない」といった事情がある場合は、青いボタンのすぐ下にある For all versions see the PsychoPy releases on github と書かれているリンクをクリックすると異なるバージョンのインストーラーをダウンロードすることができます。

Windows では、ダウンロードしたファイルをダブルクリックするとインストーラーを起動することができます。インストールには管理者の権限が必要ですので注意してください。Windows の保護機能によって 図 1.3 のような警告が出ることがありますが、「詳細情報」をクリックすると「実行」ボタンが出現して実行することができるようになります。

インストールが無事に終了したら、スタート画面を表示してみましょう。PsychoPy3 という項目ができているはずです (図 1.4)。このアイコンをクリックすると PsychoPy が起動します。

MacOS X の方は、ダウンロードした dmg ファイルを開くとディスクイメージが開きますので、PsychoPy のアイコンをアプリケーションフォルダにドラッグ＆ドロップしてください。これでインストールは終了です。Ubuntu を使用している方はパッケージを利用してインストールできます。詳しくは「1.3.2:Linux で PsychoPy を使う 」をご覧ください。PsychoPy のパッケージが用意されていない Linux 系 OS を利用している方や、すでに Windows 上で python.org 版の Python を利用している等の理由で Windows の Standalone 版 PsychoPy を

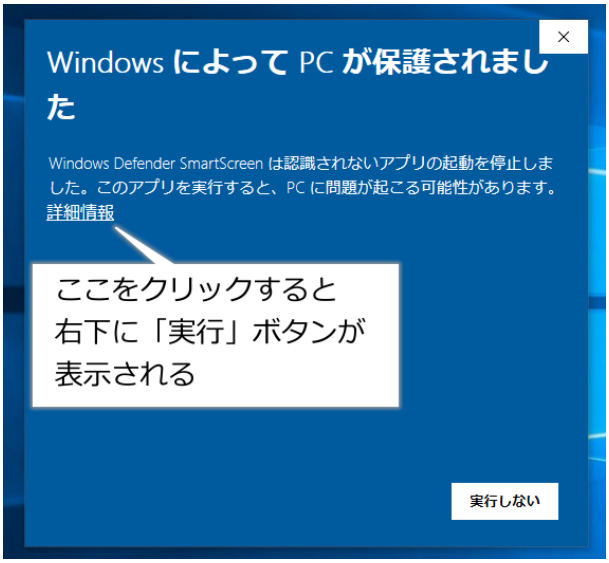


図 1.3　Windows ではインストーラーの実行時に警告が表示されることがあります。

図 1.4　インストールが終了するとスタートメニューの「すべてのアプリ」に PsychoPy3 というグループができています。この中に PsychoPy3 というアイコン (Builder とか Coder とか後ろについていないもの) がありますのでこれを選択して起動します。

使いたくない方は「1.3.3:Standalone インストーラーを使用しないインストール (上級) 」を参照してください。

さて、PsychoPy を実行すると、図 1.5 に示すウィンドウが画面に出現します。図 1.5 左のように、ウィンドウ内が三分割されていて右側にアイコンが表示されているウィンドウが Builder です。Coder と Runner は似ていますが、Runner はウィンドウ右端にアイコンが縦に並んでいるのが特徴です。本書では主に Builder と Runner を使用します。なお、スタートメニューに作られたアイコンのうち、PsychoPy Builder、PsychoPy Coder、PsychoPy Runner と書かれたものは、それぞれ Builder、Coder、Runner のウィンドウのみを開いて起動します。後から他のウィンドウを開かなければ行けなくなった場合は、ウィンドウ上部のメニューの「ビュー」から「Builder を開く」「Coder を開く」「Runner を開く」から開くことができます。

使用しているパソコンが日本語環境のものであれば、メニューなどは自動的に日本語で表示されますが、万一英語で表示されてしまう場合には、ロケールを設定する必要があります。「1.3.4: ロケールの変更 」を見て設定してください。日本語環境のパソコンを使用しているけどあえて英語でメニューなどを表示させたいという場合にもロケールの設定は有効です。

インストールしている PsychoPy より新しいバージョンの PsychoPy が公開されると、PsychoPy を起動してしばらくした後に、図 1.6 のようにアップデート通知のダイアログが表示されます。アップデートを行うと作成した実験が実行できなくなったり動作が変わったりすることがありますので、不用意にアップデートしないことをお勧めします。アップデートによる変更が小さい場合 (図の左のダイアログ) は、「このバージョンをス

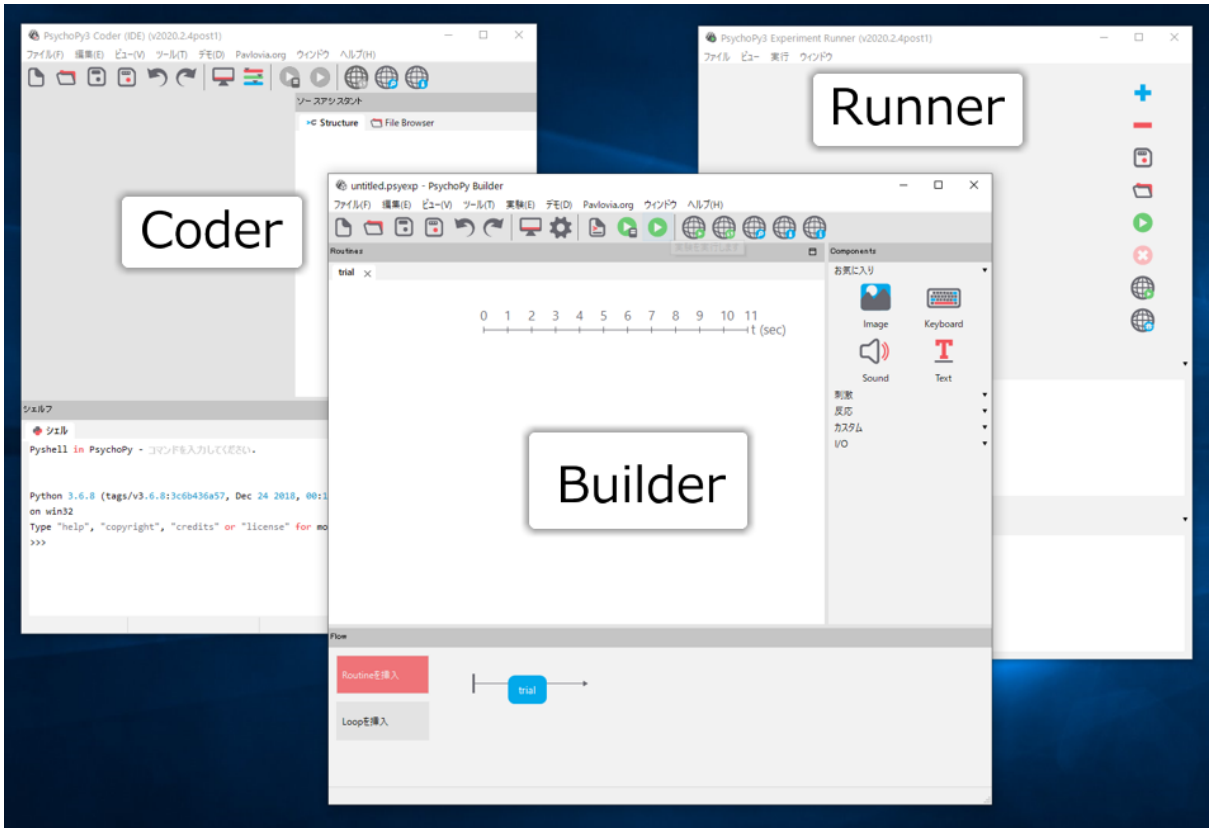


図 1.5　PsychoPy Builder(中央)、Coder(左)、Runner(右)。Builder はウィンドウ内が三分割されていて、右側にアイコンが並んでいます。Coder と Runner は似ていますが、Runner はウィンドウ右端にアイコンが並んでいるのが特徴です。

キップする」というボタンを押すことによってさらに新しいバージョンが公開されるまで通知ダイアログを表示させないようにすることができます。

アップデートの内容を確認して、アップデートを行うことに決めた場合は、インストール済みのファイルを変更できる権限を持つユーザーで PsychoPy を起動する必要があります。詳しくは「1.3.5:PsychoPy のバージョンアップ」をご覧ください。

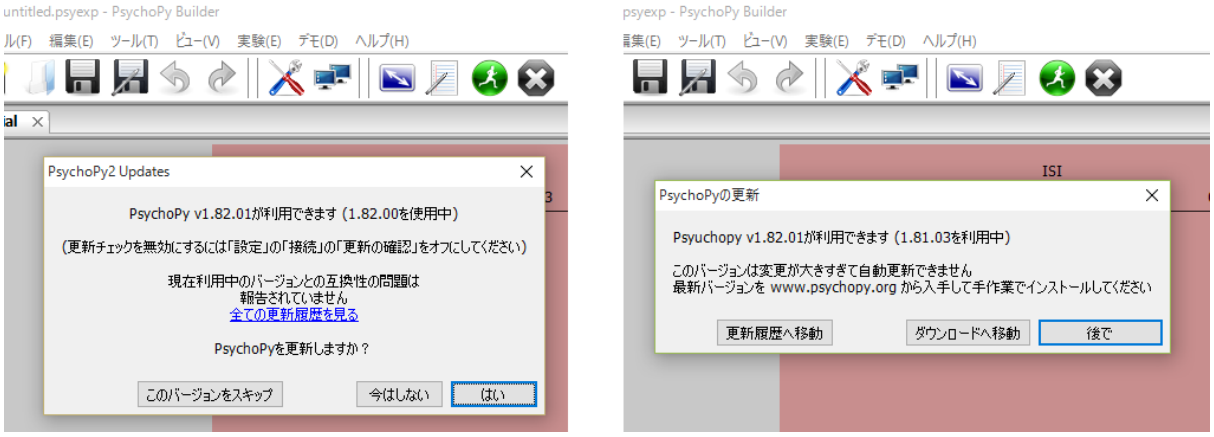


図 1.6　アップデート通知のダイアログ。通常は左のダイアログが表示されますが、アップデートによる変更が大きい場合は右のダイアログが表示されます

従来のバージョンの PsychoPy では、初めて起動した時に、使用中の PC の性能を確認する「設定ウィザード」というものが自動的に実行されたのですが、一部の PC で設定ウィザードの実行がうまくいかないために PsychoPy が強制的に終了されてしまうという問題が生じたため、現在のバージョンでは実行されなくなってしまいました。設定ウィザードは実行しなくても問題ないのですが、自分の PC の性能が十分か確認したい場合は代わりの「ベンチマークウィザード」というものを実行することができます。詳細については「1.3.6: ベンチマークウィザードによる PC 性能のチェック 」をご覧ください。

さて、これで Builder が使用できる状態になりました。さっそく解説を始めたいところですが、その前に二つ確認してください。まず、皆さんが Builder を使って実験を作成する PC には Microsoft Excel (以下 Excel) がインストールされていますか？ Builder では Excel 2007 以降でサポートされた xlsx 形式の Excel ファイルを利用して、実験条件の設定を簡単に行うことができます。このテキストでも 第 3 章 以降で Excel を利用します。

Excel を持っていない場合は、フリーソフトウェアの LibreOffice Calc を利用することもできます。LibreOffice は以下の URL でダウンロードすることができます。

- http://www.libreoffice.org/
- http://ja.libreoffice.org/home/ (日本語)

日本語版の web ページをブラウザで開いた様子を 図 1.7 に示します。「LibreOffice をダウンロード」をクリックしてダウンロードページへ移動し、メインインストールパッケージとヘルプパッケージをダウンロードします。ダウンロードが終了したらメインインストールパッケージ、ヘルプパッケージの順にダウンロードしたインストーラーをダブルクリックしてインストールを済ませてください。LibreOffice Calc は Excel と比べてメニューやツールバーのレイアウトが異なるほか、機能面でもいろいろな違いがありますが、本書で解説している用途の範囲では十分に Excel の代わりとして使えます。

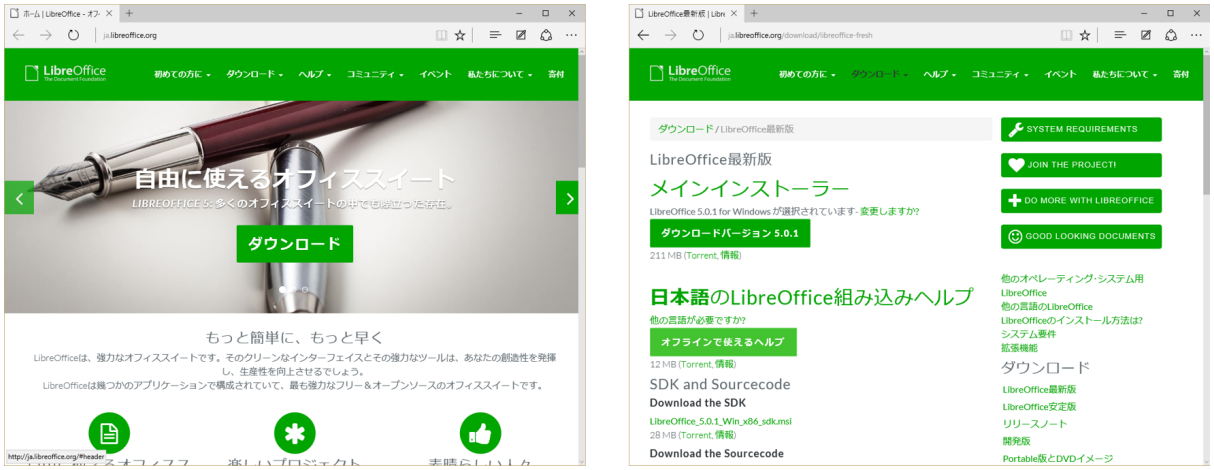


図 1.7　LibreOffice の日本語版 web ページ (左)。「LibreOffice をダウンロード」をクリックするとダウンロードページ (右) へ移動するのでメインインストールパッケージとヘルプパッケージをダウンロードします。

Windows ユーザーの方は、もう一つの確認していただくポイントがあります。Excel のファイルをデスクトップに新規作成してみてください。Excel がインストールされていない場合はテキストドキュメントなどでも構いません。ファイル名の最後に.xlsx(テキストドキュメントなら.txt) という文字列が表示されているでしょうか、それとも表示されていないでしょうか (図 1.8)。このファイル名の最後についている「ピリオド + 英数文字」を拡張子と呼びます。拡張子はファイル名の一部で、ファイルの内容を示す記号として用いられます。

Windows は拡張子に基づいてファイルを編集するアプリケーションを決定するため、拡張子をうっかり変更してしまうとどのアプリケーションで編集すればいいのかわからなくなってしまいます。このようなトラブルを防ぐために、Windows の標準設定では拡張子を表示されないようになっています。ところが、拡張子が表示されていないと次の章からの解説が非常にわかりにくくなってしまいますので、次の章へ進む前に拡張子が表示されるように設定しておいてください。Windows10 の場合は、エクスプローラーのウィンドウの上部の「表示」タブをクリックすると「ファイル名拡張子」という項目がありますので、そこにチェックをつけてください

以上の作業が終了したら、次の章へ進んで実際に Builder を使ってみましょう。

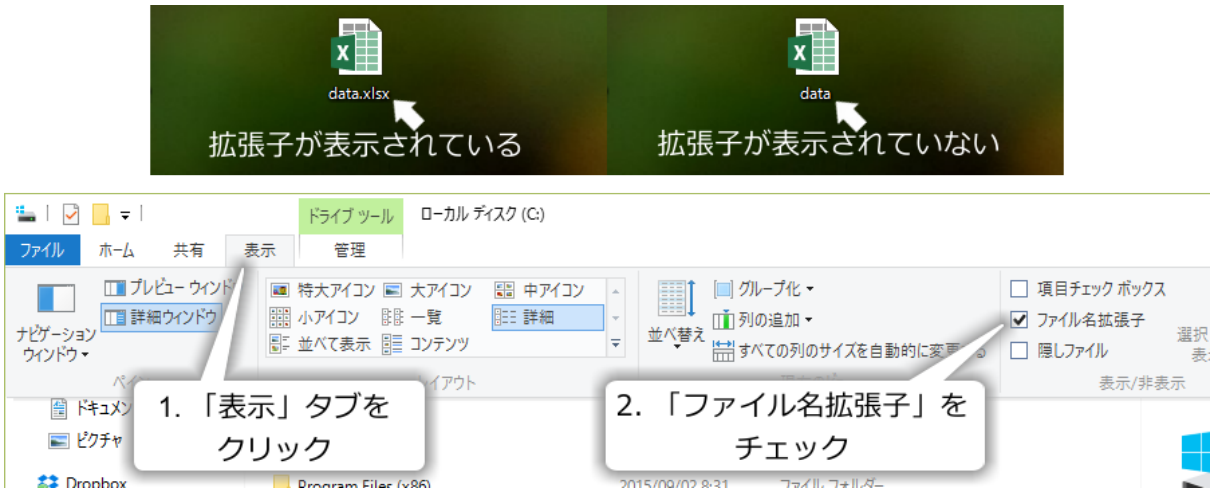


図 1.8　Windows10 で拡張子が表示されていない時の表示方法。

チェックリスト

- PsychoPy を起動できる。
- Coder や Runner のウィンドウから Builder のウィンドウを開くことができる。
- Excel または LibreOffice Calc のどちらかを起動できる。
- ファイルの拡張子を表示できる。

1.3 この章のトピックス

1.3.1 コンパイラとインタプリタ、そしてスクリプト

プログラミング言語で書かれたプログラムを実行する際に、毎回プログラミング言語を機械語に翻訳しながら実行する方法と、事前に機械語に翻訳したプログラムを作成しておいて、実行時には翻訳済みのプログラムを実行する方法があります。前者の毎回翻訳を行う翻訳プログラムをインタプリタと呼び、後者の事前に翻訳済みプログラムを作成する翻訳プログラムをコンパイラと呼びます。

インタプリタの利点は、プログラムを書いたら直ちに PC に実行させることができる点です。少し書いては書き直すといった試行錯誤をする時にはとても楽です。その代り、実行する度に PC は翻訳作業を行いますので、実行速度がやや遅いという欠点があります。コンパイラは逆に、プログラムを書いたらコンパイラを使っ

て翻訳する作業を毎回行う必要があります。また、人間が書いたプログラムのファイルの他に翻訳後のプログラムのファイルができるのでファイルの整理が面倒です。その代り、実行する時にはすでに翻訳済みなので高速に実行することができます。

多くのプログラミング言語では、インタプリタとコンパイラのどちらの方法が用いられるかが決まっています。例えば C 言語という言語ではコンパイラを使って事前に翻訳済みプログラムを作成することがほとんどです。一方、Python は一般にインタプリタを使って実行します。

インタプリタを使う言語のように、人間が書いたプログラムを（コンパイルなどの作業なしに）直接実行できる言語をスクリプト言語と呼ぶことがあります。そして、スクリプト言語で書かれたプログラムをスクリプトと呼びます。この意味では Python はスクリプト言語であり、Python のプログラムはスクリプトです。

1.3.2 Linux で PsychoPy を使う

Linux で PsychoPy を使うことを検討している方向けの発展的な話題です。使用している Linux のディストリビューションが Debian や Ubuntu であれば、PsychoPy のパッケージが用意されているので、通常のパッケージと同じ手順でインストールすることができます。例えば apt コマンドを使用する場合は、sudo を利用できるユーザーでログインして端末から以下のように入力します。

```
sudo apt-get install psychopy
```

PsychoPy のパッケージが用意されていないディストリビューションで利用する場合は、「1.3.3:Standalone インストーラーを使用しないインストール (上級) 」で述べる方法で Python のパッケージとしてインストールすることができます。

1.3.3 Standalone インストーラーを使用しないインストール (上級)

すでに Python を利用している人向けの発展的な話題です。Standalone インストーラーはとても便利なのですが、Python の実行環境を丸ごとインストールするので、Python をインストール済みの PC に Standalone インストーラーで PsychoPy をインストールすると Python の実行環境が PC の中に二重に存在することになってしまいます。このような事態を避けたい場合は、インストール済みの Python へパッケージとして PsychoPy を導入することもできます。

例えばパッケージの管理に pip を使用しているのであれば、以下のコマンドを実行すれば PsychoPy をパッケージとしてインストールすることができます。

```
pip install psychopy
```

ただし、PsychoPy の機能をすべて利用するためにはオプショナルな（上記コマンドで自動インストールされない）パッケージも必要であり、実行時のエラーメッセージを確認しながらパッケージを導入するなどしなければいけません。

1.3.4 ロケールの変更

Builder のメニュー等の表示に使用される言語を変更したい場合は、ロケールを設定してください。Coder でも Builder でも、ウィンドウの上部に 図 1.9 に示す工具のアイコンのボタンがあります。これをクリックすると PsychoPy の設定ダイアログが開きます。「アプリケーション」というページに「ロケール」という項目があります。

初期状態ではロケールは空白となっています。この場合、PsychoPy は使用中のパソコンの言語設定を利用して表示に用いる言語を決定しようとします。PsychoPy によって選ばれた言語と異なる言語で表示させたい場合は、ロケールをクリックして表示される言語の一覧から希望のものを選んでください。

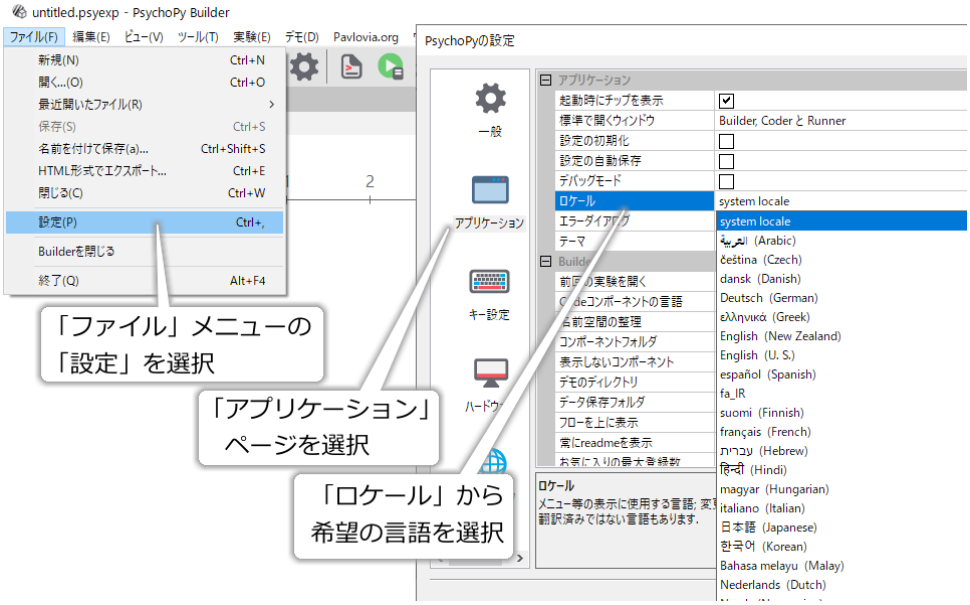


図 1.9　ロケールの設定。

1.3.5 PsychoPy のバージョンアップ

本文中で述べたとおり、PsychoPy の新しいバージョンが公開されると起動時にアップデート通知ダイアログ (図 1.10) が表示されます。このダイアログで「はい」ボタンをクリックすると、図 1.10 のダイアログが表示されます。「自動更新」を選択すると、PsychoPy はインターネット上から最新バージョンを自動的にダウンロードしてアップデートを試みます。しかし、筆者の経験上、「自動更新」は失敗することがあるので次の「以下の Zip ファイルを使用」を利用することをお勧めします。「以下の Zip ファイルを使用」では、ユーザーが自分で PsychoPy の配布サイトから最新版の zip ファイルをダウンロードしてアップデートを行います。Standalone インストーラーではなく、拡張子が zip のアップデートを選択するのがポイントです。Browse ボタンを押してダウンロードした zip ファイルを選択すると、アップデートが行われます。古いバージョンの zip ファイルをダウンロードして利用することによって、以前のバージョンに戻すことも可能です。ただし、バージョン間でアップデート不能な大きな変更がなかった場合に限ります。

どちらの方法でアップデートを行うにせよ、注意すべき点がひとつあります。Windows で Standalone インストーラーを使った場合や、Ubuntu のパッケージを使ってインストールした場合は、管理者の権限があるユーザーでなければアップデートを行えません。Windows の場合は PsychoPy のアイコンを右クリックして「管理

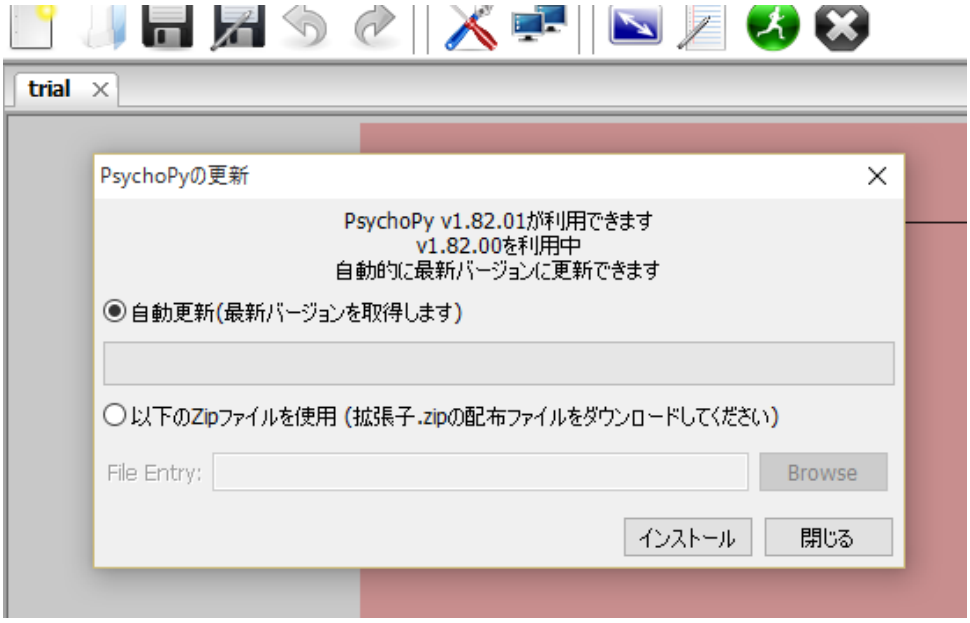


図 1.10　PsychoPy のアップデートのダイアログ。

者として実行」して PsychoPy を起動してください。Ubuntu の場合はターミナルを起動して sudo psychopy と入力すると管理者の権限で PsychoPy を起動できます。

図 1.6 右に示したダイアログが表示された場合は、この方法でのアップデートが困難なほど大きな変更が行われます。多くの場合、PsychoPy 以外のパッケージもバージョンアップが必要だったり、さらに追加のパッケージが必要だったりしますので、配布サイトへアクセスして Standalone 版インストーラーをダウンロードしてインストールするのがお勧めです。

1.3.6 ベンチマークウィザードによる PC 性能のチェック

Builder または Coder のメニューの「ツール」にある「ベンチマークウィザード」(図 1.11 左上) を実行すると、使用中の PC の性能を確認するためのテストプログラムが実行されます。ベンチマークウィザードの実行は PsychoPy を使うにあたって必須ではありませんが、本番の実験を実行する PC では一度は実行しておいて問題がないことを確かめておきたいところです。

「ツール」の「ベンチマークウィザード」を選択すると、20-30 秒ほど時間がかかるが実行してもよいかという確認のダイアログが表示されます。OK をクリックすると、PC の性能を確認するために画面上に様々なパターンが描かれますので、しばらく **なにも操作せずに** 待ってください。余計な操作をすると正確な結果が得られない可能性があります。終了すると 図 1.11 左下のように「全結果を確認するには OK をクリックします」というダイアログが表示されるので、OK をクリックしてください。ブラウザが起動して、図 1.11 右のように結果が表示されます。この例では DropBox が動作していると警告されているほか、テスト中に描画タイミングが (標準偏差で)1.5 ミリ秒程度ばらついていたことや、スピーカーの出力に 0.2 秒ほど要したなどが指摘されています。

PC を使用した実験について十分な知識がない方は、このような警告が表示されると不安になると思います。一般にどのくらいの値であれば問題ないという基準を示すのは難しいのですが、多くの実験では 1.5 ミリ秒程度の画面更新間隔の標準偏差は問題にならないはずです。スピーカーの出力が 0.2 秒遅れるのは、視覚刺激と

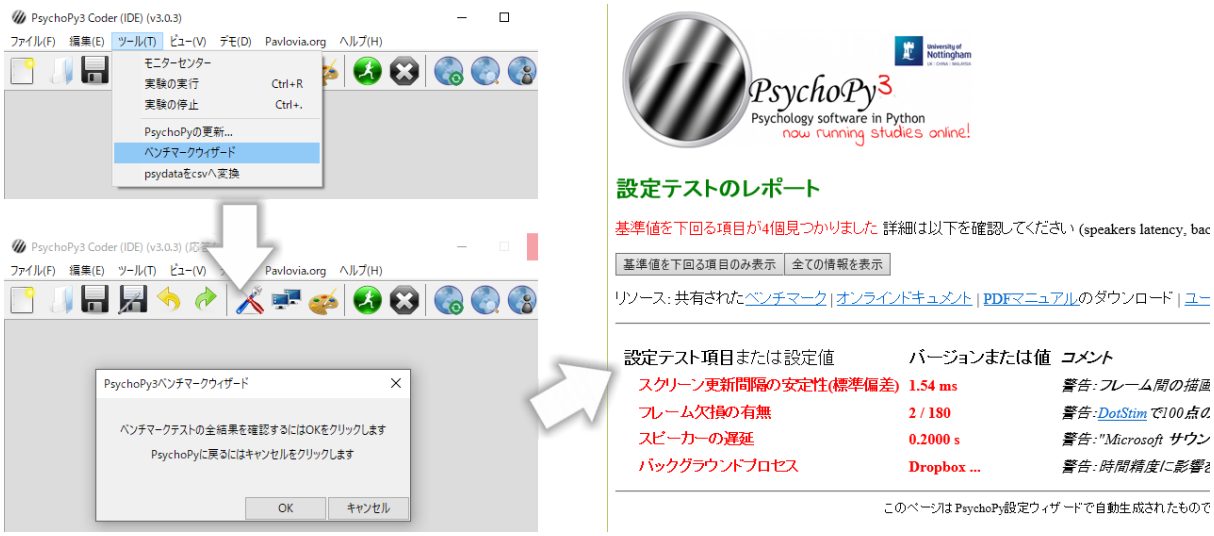


図 1.11　ベンチマークウィザードによる動作環境のテスト

聴覚刺激のずれが問題になる実験をしている人にとってはまずいですが、聴覚刺激を使わない人には問題にならないでしょう。「どのような実験をするか具体的な計画があるわけではないけど、とりあえず PsychoPy Builder の使い方を学んでみたい」という人は、あまり警告を気にせず使ってみるのが良いと思います。具体的な実験計画ができてきたら、先行研究の手続きや結果をよく読んで、どの程度の精度が必要なのか、自分が使っている PC はその精度を達成できるのかを判断しましょう。

第 2 章

刺激の位置や提示時間の指定方法を覚えよう

2.1 まずは表示してみよう

なにはともあれ、まずは刺激を PC の画面に表示してみましょう。Builder を起動するとウィンドウ内が三つに分割されています。この分割されたひとつひとつの部分をペインと呼びます。痛みのペイン (pain) ではなく、窓枠にはめる一枚一枚のガラスのペイン (pane) です。左のペインをルーチンペイン、右のペインをコンポーネントペイン、下のペインをフローペインと呼びます (図 2.1)。コンポーネントとは実験を作るための部品のようなもので、コンポーネントペインには Builder で使用できるコンポーネントを表すアイコンが並んでいます。ここから刺激を表示するためのコンポーネントなどを選んでルーチンペインに配置し、フローペインで実験の流れを指定するという手順で実験を作成します。

この章では、コンポーネントペインとルーチンペインの使い方を覚えましょう。まず、コンポーネントペインの「刺激」と書いてある部分を何度かクリックしてみてください。コンポーネントのアイコンが現れたり消えたりするはずです。コンポーネントはよく使う「お気に入り」、刺激描画に使う「刺激」、反応計測に使う「反応」、高度な処理を行うための「カスタム」、脳波測定のための「EEG」、視線測定のための「アイトラッキング」、外部機器との入出力に使う「I/O」のカテゴリに分類されており、それぞれカテゴリ名をクリックするとそのカテゴリに含まれるコンポーネントのアイコンが表示されたり隠されたりします。

それでは「刺激」カテゴリに含まれている Polygon コンポーネントを使って実際に刺激を PC の画面に表示してみましょう。Polygon コンポーネントはの右下に示されている楕円、三角、十字が描かれたアイコンです (図 2.2)。このアイコンをクリックすると、描画する刺激の大きさや色といった特性（プロパティと呼びます）を設定するダイアログが表示されます (図 2.3)。ダイアログの左上の辺りに「基本」、「レイアウト」、「外観」、… と書かれている点に注目してください。これは「タブ」と呼ばれて、タブをクリックすることによって表示される内容が切り替わります。タブをクリックして表示内容を変更することを「ページを切り替える」などということもあります。

まず一切プロパティを変更せずに右下の OK をクリックしてみましょう。すると、ルーチンペインに polygon コンポーネントのアイコンが表示され、その右側に青い棒が表示されます。これで「コンポーネントをルーチンに配置する」という作業ができました。Builder では、このようにルーチンにコンポーネントを配置していくことによって刺激を作成します。

では、この Polygon コンポーネントがひとつ配置されただけのシンプルな「実験」を実行してみましょう。

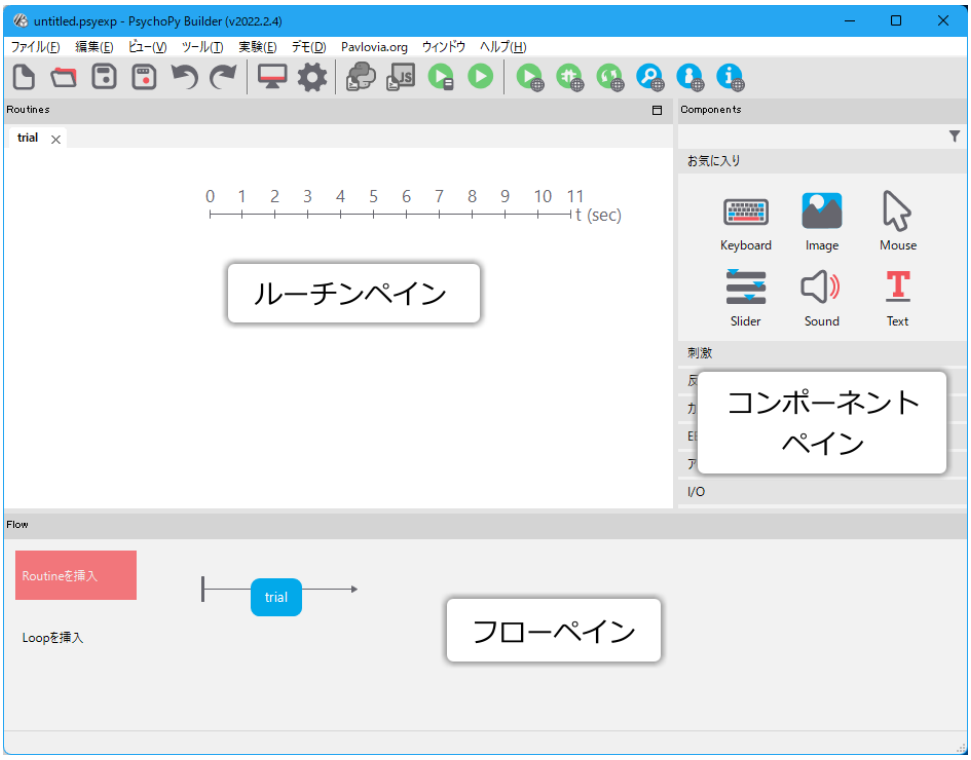


図 2.1　Builder の 3 つのペイン。コンポーネントペインから刺激等を選んでルーチンペインに配置し、フローペインでルーチンの実行順序を指定します。

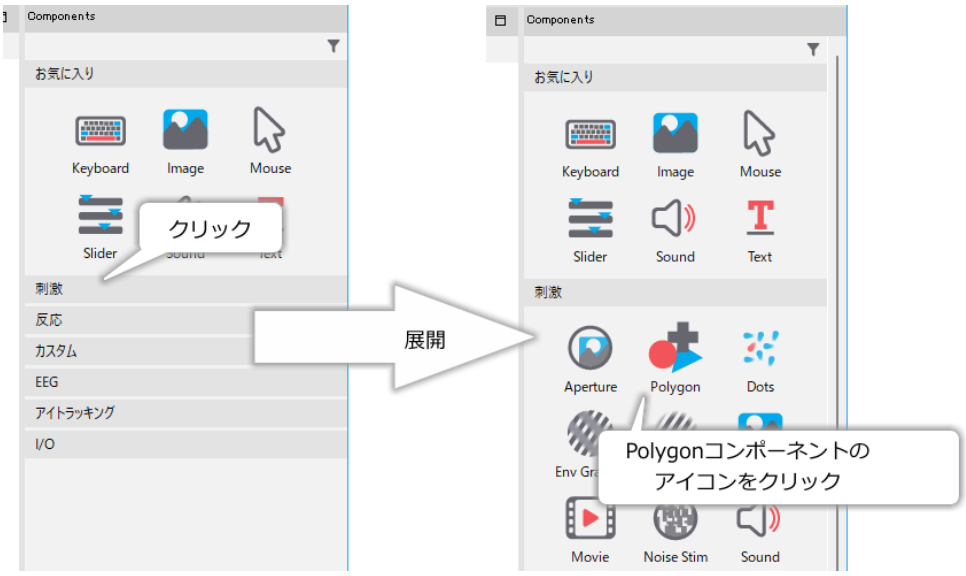


図 2.2　「刺激」カテゴリを開いて Polygon コンポーネントのアイコン (円、三角、十字が描かれたアイコン) をクリックします。

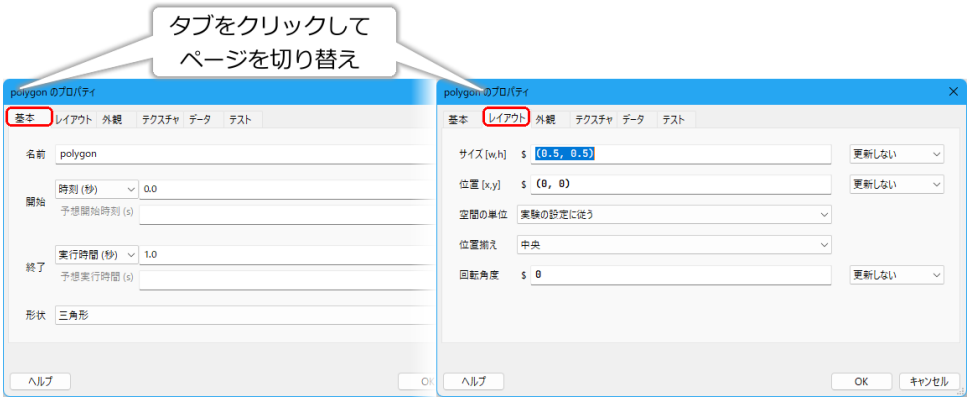


図 2.3　Polygon コンポーネントのプロパティを設定するダイアログ。左上のタブをクリックすることでページを切り替えられます。

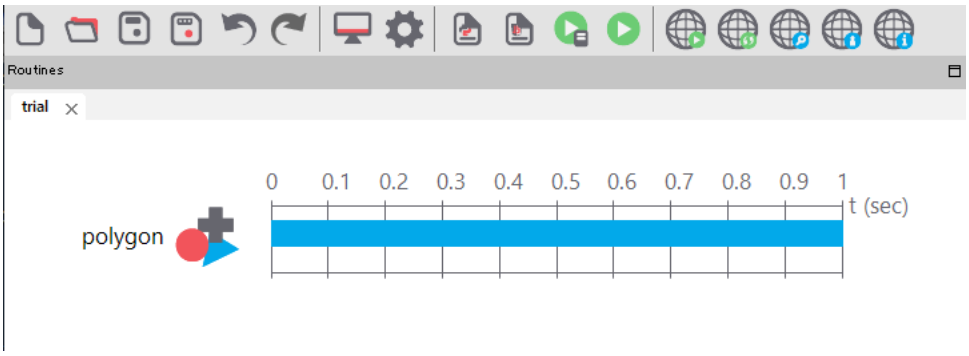


図 2.4　コンポーネントをルーチンに配置した状態。

Builder で実験を実行する方法は「直接実行する方法」と「Runner に登録して実行する方法」の 2 種類がありますが、プログラミング初心者の人は Runner の出力を確認する習慣をつけておいた方が勉強になってよいと思いますので、ここでは「Runner に登録して実行する方法」を紹介します。Runner に実験を登録するには、ウィンドウ上部の 図 2.5 に示したボタンをクリックします (以後、このボタンを Runner ボタンと呼びます)。実験を作り始めてまだ一回も実験を保存していない場合、Runner ボタンをクリックするとまず実験を保存するダイアログが表示されます。ここでは exp01.psyexp という名前で保存しておきます。拡張子が.psyexp のファイルには、Builder で作成した実験の情報が保存されています。以後、このファイルのことを psyexp ファイルと呼びます。psyexp ファイルの他にも Builder が作成するファイルがありますが、それらについては「 2.7:Builder が作成するファイルを確認しよう 」で解説します。

図 2.5　Runner ボタンをクリックして Runner に実験を登録します。まだ実験を保存していなければ保存ダイアログが表示されますので exp01.psyexp という名前で保存してください。

Runner には、図 2.6 のように今保存した実験が登録されているはずです。Runner ウィンドウの右側には、実

験実行ボタンと実験中断ボタンがあります。今登録した実験を選択した状態 (Windows の標準の配色では 図 2.6 のように青色で強調表示されます) で実験実行ボタンをクリックしてください。

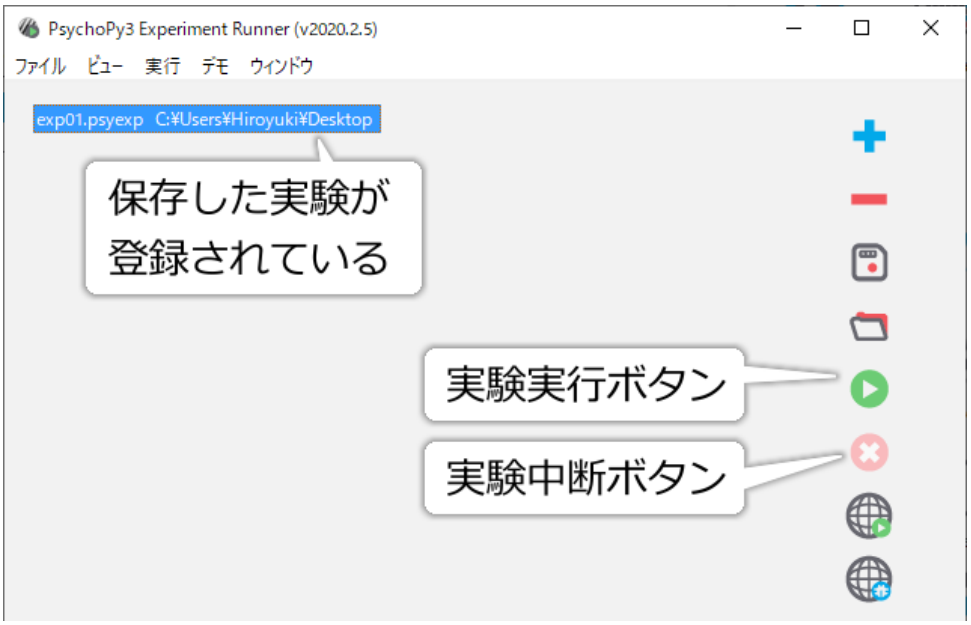


図 2.6　Runner に保存した実験が登録されます。右側の実験実行ボタンで実験を実行することができます。

実験実行ボタンをクリックしてしばらく待つと、 図 2.7 のような小さなダイアログが表示されます。これを実験情報ダイアログと呼びます。少し古めの PC やノート PC を電源につながず実行している場合などには実験ダイアログが表示されるまで少し時間がかかりますのでご注意ください。同時に、Runner の実験中断ボタンが有効になります。実験中断ボタンをクリックすると実行が中断されます。Runner の一番下の「標準出力」という欄には、実験の実行状況についての情報が表示されますが、ここはとりあえず無視して構いません。

実験情報ダイアログは、実験参加者の氏名や実験の条件など、実験の実行に必要な情報や、データと一緒に保存しておきたい情報などを入力するためのものです。どのような情報を入力できるようにするかはもちろん設定できるのですが、ここではまだ何も設定していないので標準で設定されている session と participant という項目が表示されています。この章では実験情報ダイアログを利用するところまで進みませんので、とりあえず変更せずに OK をクリックしてください。すると画面全体が灰色一色になり、少し間をおいて白い三角形が表示されます (図 2.8)。この三角形は Builder の設定によって少し形状が変わるのですが、インストール直後の状態の場合、バージョン 3.0.3 以降ならやや縦長で大きめ、それ以前ならやや横長で小さめになります。

三角形が出現してから 1 秒後に、灰色の画面から元の Runner のウィンドウが表示されている画面に戻ります。これで私たちが作った「最初の実験」が無事に終了しました。

続いて、コンポーネントの設定を変更してみましょう。Builder のウィンドウに戻って、ルーチンペインに表示されているコンポーネントのアイコンにマウスカーソルを動かして左ボタンをクリックしてみてください。すると先ほどコンポーネントを配置した時に表示されたプロパティ設定ダイアログが再び表示されます。ここで 図 2.9 のようにダイアログの「基本」タブにある「形状」を長方形に変更し、「外観」タブの「塗りつぶしの色」という項目に black と入力して、ダイアログ右下の OK をクリックしてみましょう (black と入力する時には日本語入力を OFF にしてください)。これで形状が長方形、刺激の塗りつぶし色が黒色に設定されました。もう一度実験を実行すると、今度はのように黒色に塗りつぶされた正方形が表示されるはずです（三角形の時と同様、3.0.3 より前のバージョンでは設定を変更していなければ横長の長方形になります)。

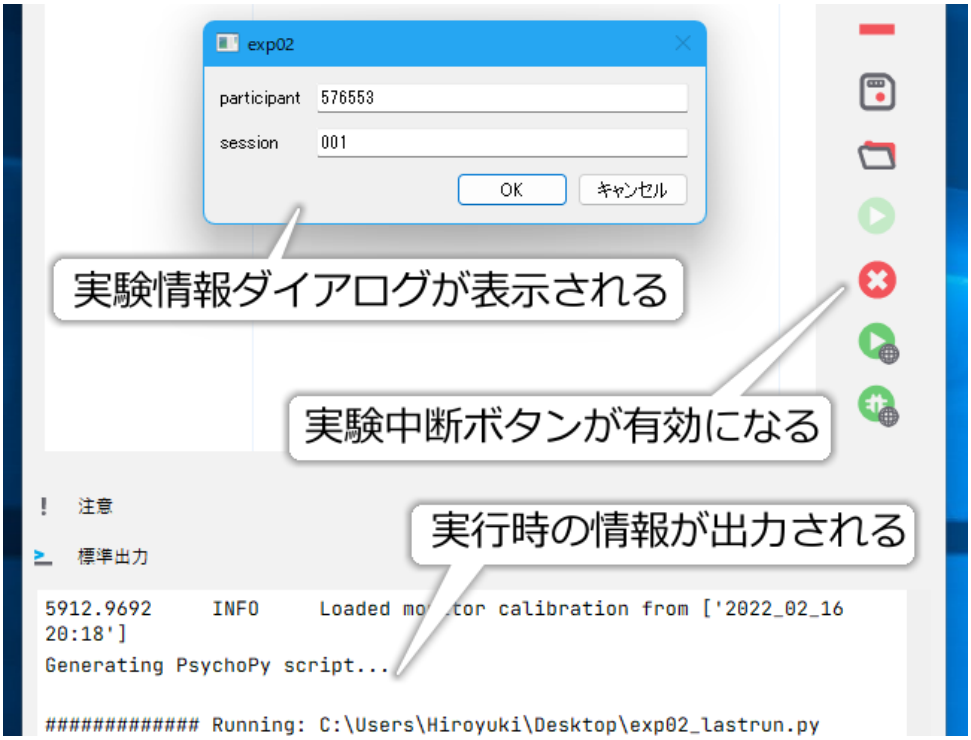


図 2.7　実行ボタンをクリックした後、少し待つと実験情報ダイアログが表示されます。同時に実験中断ボタンが有効になります。

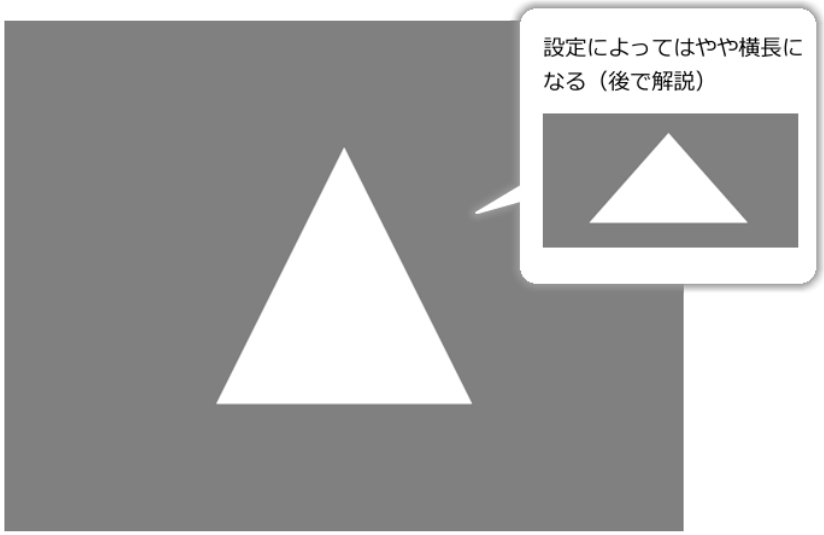


図 2.8　実行すると上図のように灰色の画面に 1 秒間白い三角形が表示されます。終了すると下図のようなログが表示されます。

図 2.9　「形状」を長方形、「塗りつぶしの色」を black に設定して実行してみましょう。

このように、配置済みコンポーネントのプロパティは何度も設定し直すことができます。この操作は今後繰り返しおこなうことになるので覚えておきましょう。本書ではこれ以後、プロパティ名を **[塗りつぶしの色]** のように [ ] で囲って表記します。

次の話題に進む前にもうひとつ覚えておきたい基本操作は、ルーチンペインに設置したコンポーネントの削除です。ルーチンペインに配置された Polygon コンポーネントの上にマウスカーソルを動かして今度は右クリックしてみてください。図 2.10 のようにポップアップメニューが表示されますので、「削除」を選択してください。コンポーネントがルーチンペインから取り除かれます。「削除」の項目に「(polygon)」とついているのは、削除しようとしているコンポーネントの名前を表しています。次節以降、ひとつのルーチンに複数のコンポーネントを配置しますが、そのような場合にどのコンポーネントを削除しようとしているのか一目でわかるように表示されています。

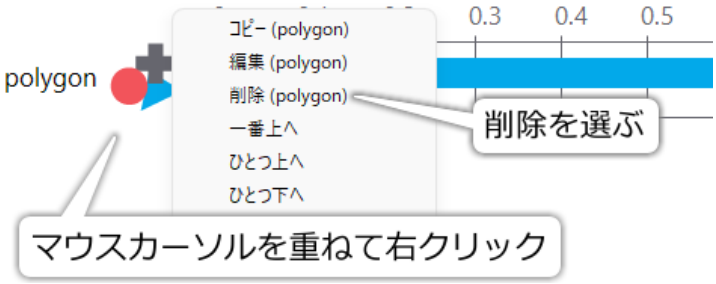


図 2.10　ルーチンペインのコンポーネントのアイコン上でマウスを右クリックするとメニューが表示されます。「削除」を選択するとコンポーネントを削除できます。

さて、これでコンポーネントをルーチンペインに配置し、プロパティを編集し、不必要なコンポーネントを削除し、実験を実行することができるようになりました。続いてコンポーネントのプロパティを編集して刺激の色や大きさを調節する方法を学びましょう。

チェックリスト

- ルーチンペインにコンポーネントを配置できる。
- 作成した実験を実行できる。
- コンポーネントのプロパティ編集ダイアログを開くことができる。
- Polygon コンポーネントを用いて三角形と長方形を表示できる。

- コンポーネントをルーチンから削除することができる。
- 実験を保存したファイルの拡張子を答えられる。

2.2 位置と大きさを指定しよう

再び Polygon コンポーネントを用いて、PsychoPy における視覚刺激の位置と大きさ、向きを指定する方法を習得しましょう。ルーチンペインに Polygon コンポーネントをひとつ配置して、 **[形状]** を長方形にしてください。

Polygon コンポーネントのプロパティ設定ダイアログの「レイアウト」タブを表示すると、 **[位置 [x, y] $]** 、**[サイズ [w, h] $]** という項目があります。それぞれ刺激の位置と大きさの指定に対応しています。これらの項目に 1.0 とか 150 とかいった値を入力することによって位置や大きさを指定するのですが、実際にこういった値を入力した時にどのような結果が得られるかを理解するためには、PsychoPy における位置と大きさの単位を理解する必要があります。

位置や大きさの単位は、同じタブにある **[空間の単位]** という項目で指定します。 表 2.1 に PsychoPy で使用できる単位を示します。スクリーン上の画素 (ピクセル) で指定する pix、センチメートルで指定する cm、スクリーンの大きさに対する比で指定する norm と height、視角で指定する deg, degFlat, degFlatPos があります。これらの単位の関係を、スクリーンの解像度が横 1920 ピクセル、縦 1080 ピクセル、寸法が幅 51.0cm、高さ 28.6cm のモニターを例として示したのが 図 2.11 です。いずれの単位でもスクリーンの中心は常に原点 (0, 0) で、水平方向は右、垂直方向は上が正の方向です。スクリーンの右上の位置を pix で示す場合、スクリーンの横方向に 1920 ピクセルあるのですからスクリーン中心を基準にすればスクリーンの左端は 960 ピクセル (1920 ピクセルの半分) 進まなければいけません。同様に垂直方向に 1080 ピクセルありますからスクリーンの上端は中心から 540 ピクセル進まなければいけません。ですから、スクリーン右上の座標は (960, 540) です。スクリーン左下の座標は水平垂直共に負の方向に進まないといけないので、(-960, -540) です (厳密にな議論は「 2.10.1: スクリーン左下の座標についての厳密な議論 (上級) 」参照)。単位が cm の場合は同様の計算でスクリーン右上が (25.5, 14.3)、左下が (-25.5, -14.3) です。

norm と height は、Builder で実験を作成するときの（設定変更してい場合の）単位の初期値として使われる重要な単位です。これらの単位はスクリーンの解像度に対する比で位置や長さを指定します。norm ではスクリーンの解像度に関わらず必ず右上の座標は (1.0, 1.0)、左下の座標は (-1.0, -1.0) になります。一般的に PC のモニターは水平方向の方が解像度は高いので、垂直方向の 1.0 よりも水平方向の 1.0 の方が画面上の長さは長くなります。そのため、正方形や円を表示したり図形を回転したりするときに注意が必要です（「 2.3: 刺激を回転させよう 」参照)。前節で「設定によっては三角形が横長になる」、「正方形ではなく長方形が描かれる」と書いたのは、単位が norm の場合に起こる現象です。PsychoPy 3.0.3 より前のバージョンでは、この norm が単位の初期値でした。

一方、PsychoPy 3.0.3 から単位の初期値となった height は、スクリーンの高さが 1.0 になるように水平方向の幅を決めます。 図 2.11 の例の場合、垂直方向 1080 ピクセルに対して水平方向に 1920 ピクセルありますので、スクリーンの幅は 1920 ÷ 1080 ≒ 1.78 です。スクリーンの幅が 1.78、高さが 1.0 なのですから、スクリーンの右上と左下の座標は (0.89, 0.5) と (-0.89, -0.5) になることに注意してください。norm と比べて正方形を表示したり図形を回転させたりするのが容易なのが特徴ですが、縦横比の異なるモニター間で右上や左下の座標が異なる点が面倒です。norm や height には、実験用と学会発表用で異なる解像度のモニターを使っている時に、学会発表用に実験プログラムを書きなおさなくてもモニターの解像度に合わせて刺激を調整できる

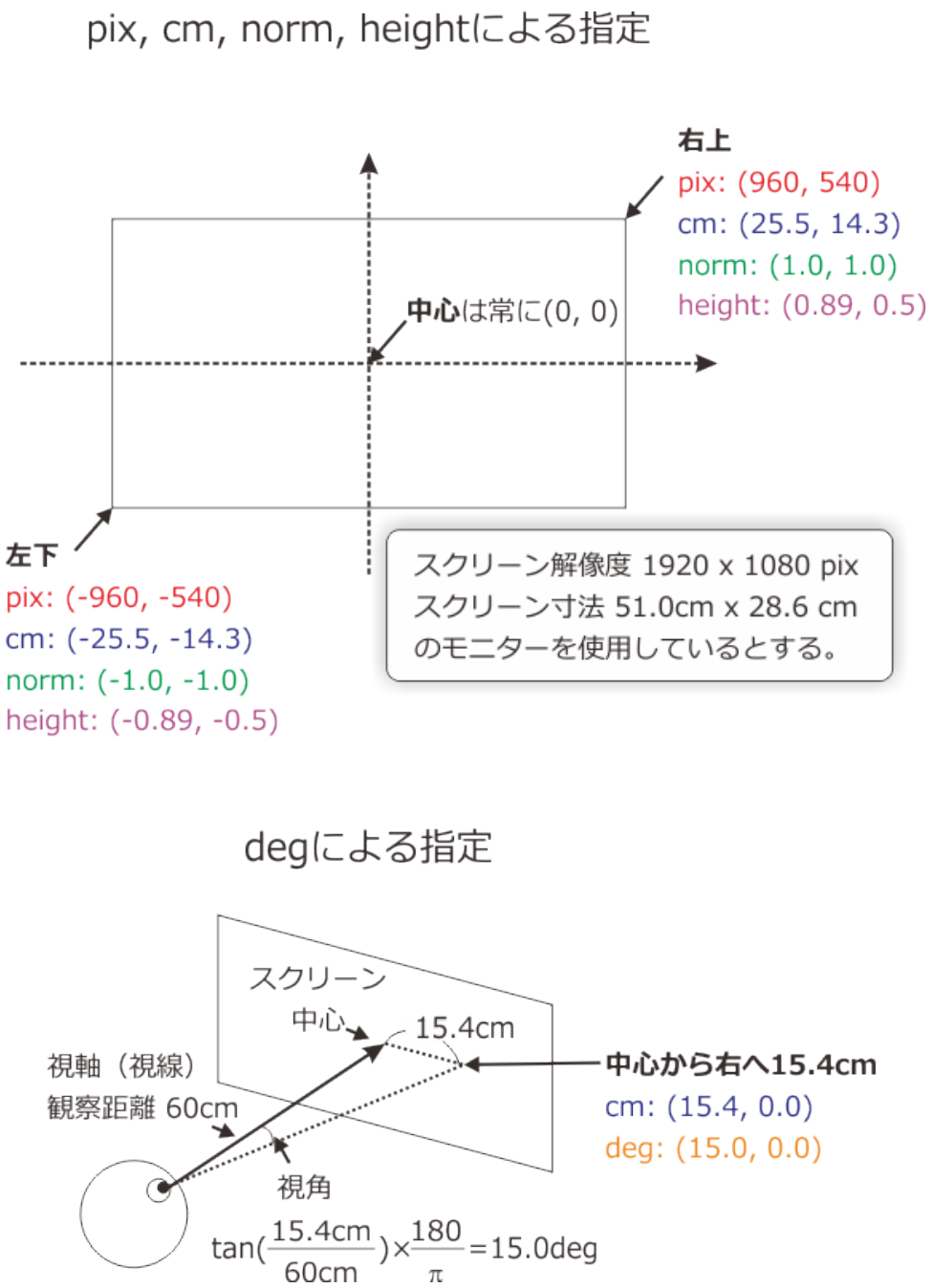


図 2.11　PsychoPy で使用できる単位。

というメリットがあります。

残るは deg, degFlat, degFlatPos ですが、まず deg から説明します。 図 2.11 下の図のように被験者が 60cm 離れた位置にあるスクリーンの中心に真っ直ぐ視線を向けているとします。眼から視線を向けている対象に引いた直線を視軸と呼びます。さて、スクリーン中心から右へ 15.4cm の位置に刺激があるとして、眼からこの刺激の位置まで引いた直線と視軸が成す角度を考えましょう。三角関数を思い出していただければ 15.4cm を観察距離 60cm で割った値の正接 (tan) を求めれば角度が得られます。この角度の単位はラジアンなので分かりやすいように 180/π を掛けて単位を度 (deg) にすると 15.0deg です。この角度を視角と呼びます。視覚を研究する時には、刺激がスクリーン上で中心から何 cm 離れていたかよりも、網膜の中心から何 deg 離れていたかの方が重要な意味を持つことがよくあるので、単位として視角が頻繁に用いられます。PsychoPy では、あら

かじめスクリーンの寸法と観察距離を登録しておくことで、deg を単位として刺激の位置や大きさを指定できます。寸法と観察距離の登録方法はこの章の「 2.8: 実験の設定を変更しよう 」で触れますので、ひとまずは「deg という単位が使える」ということを覚えておいてください。なお、PsychoPy の deg の計算は恐らく実行速度を速めるために近似的な方法を用いています。より正確な値を必要とする人のために用意されているのが degFlat, degFlatPos という単位です。詳しくは「 2.10.2:PsychoPy における視角の計算について 」をご覧ください。

表 2.1　PsychoPy で使用できる位置と大きさの単位

単位	説明
pix	モニター上の画素に対応します。例えば 100pix であればモニターの 100 画素分に対応します。
cm	モニター上での 1cm に対応します。使用しているモニターの画面の寸法と縦横の画素数を Monitor Center に登録しておく必要があります。
deg	視角 1 度に対応します。例えば 2.5deg であれば視角 2.5 度に対応します。使用しているモニターの画面の寸法と縦横の画素数、モニターと参加者の距離を Monitor Center に登録しておく必要があります。
norm	モニターの中心から上下左右の端までの距離が 1.0 となるように正規化された単位です。一般的に PC 用のモニターは縦方向より横方向の方が長いので、norm の単位で幅と高さに同じ値を指定すると横長の長方形になります。
height	モニターの上端から下端の距離が 1.0 になるように正規化された単位です。norm と異なり、一般的な PC 用モニターで幅と高さに同じ値を指定するとほぼ正方形となります。「ほぼ」というのはモニターによっては画素の縦横の長さがわずかに異なる場合があり、そのようなモニターでは正確に正方形にならないからです。
degFlat	deg と同様ですが、deg よりも正確に計算します (「2.10.2:PsychoPy における視角の計算について 」参照)。
degFlatPos	deg と同様ですが、deg よりも正確に計算します (「2.10.2:PsychoPy における視角の計算について 」参照)。
実験の設定に従う	実験設定ダイアログで指定された単位に従います。実験設定ダイアログで「PsychoPy の設定に従う」に設定されている場合は PsychoPy 設定ダイアログの **[単位]** に従います。

では、実際に刺激の大きさを変化させてみましょう。Polygon コンポーネントをひとつルーチンペインに置いて、 図 2.12 上のように **[サイズ [w, h] $]** の値を (0.2, 0.1) にしてください。数値の両脇の括弧やカンマを忘れずに入力してください（ **括弧は丸括弧でも角括弧でも前後で一致していれば構いません** ）。 **[サイズ [w, h] $]** の w と h はそれぞれ width と height ですから、(0.2, 0.1) と入力すれば、スクリーンの高さを基準として幅 0.2、高さ 0.1 の長方形を表示するように指定したことになります。入力を終えたら実験を実行すると、 図 2.12 下のようにスクリーン中央に横幅が高さの 2 倍の長方形が表示されるはずです。三角形が表示された人は **[形状]** を長方形にし忘れているので変更してください。ものすごく大きな長方形が表示されたり、何も表示されなかったりした場合は、標準の単位の設定が何らかの理由で height になっていない可能性が考えられます (研究室の共用コンピュータを使用していて設定が変更されている場合など)。その場合は **[空間の単位]** を height に変更してみてください。毎回手作業で変更するのが面倒な場合は「 2.8.2:「スクリーン」タブ 」を参考に PsychoPy の設定を変更してください。

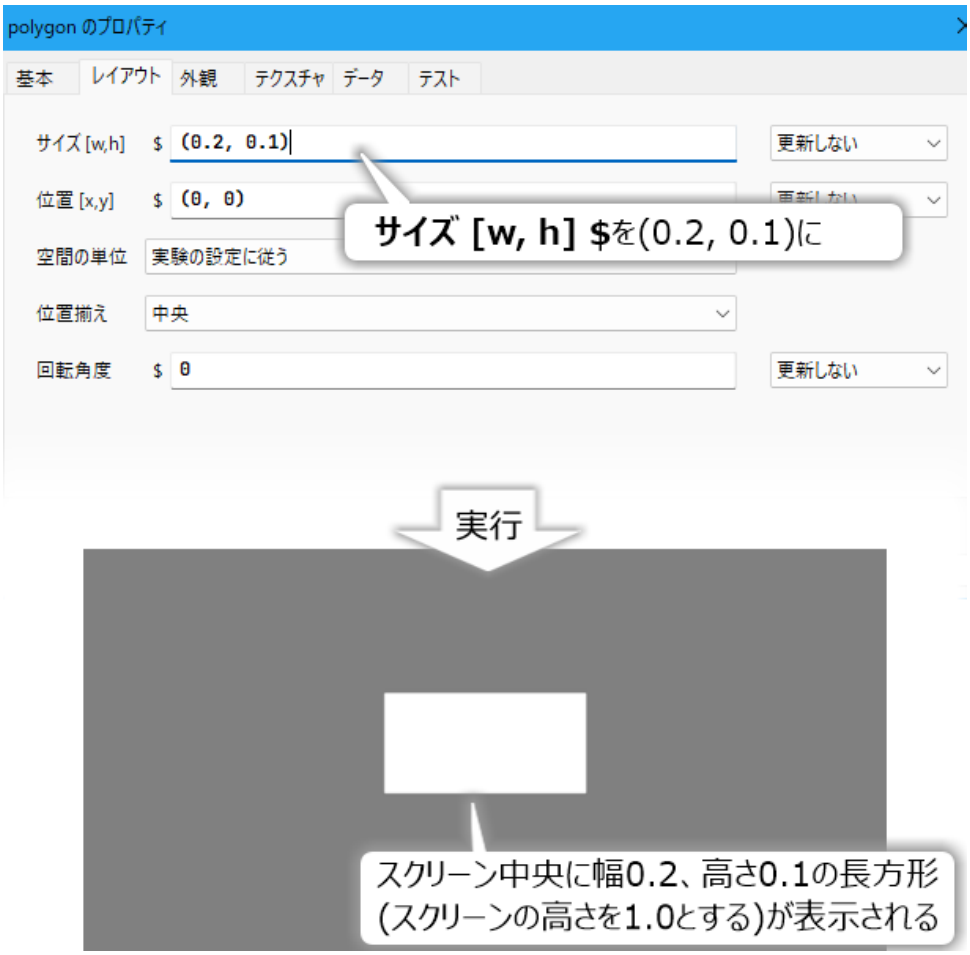


図 2.12　サイズの設定。Polygon コンポーネントの形状を長方形にして、横幅 0.2、高さ 0.1 に設定しています。

使用している環境によっては、 **[サイズ [w, h] $]** や **[位置 [x, y] $]** などに入力されたカンマ (,) と小数点 (.) が非常に区別しにくい場合があります。フォントの設定を変更すると改善される場合がありますので、気になる方は「 2.10.3:Builder の設定ダイアログで用いられるフォント 」をご覧ください。

また、使用している PC のグラフィック機能によっては長方形の対角線上の灰色の線が見える事があります (図 2.13)。PsychoPy では長方形を表示する時に実際には二つの直角三角形を並べているのですが、うまく並べられずに隙間ができてしまった時に生じる現象です。灰色の線は隙間から灰色の背景が見えてしまっているために生じています。多くの場合、Polygon コンポーネントのプロパティ設定ダイアログの「テクスチャ」タブにある **[補間]** というプロパティを変更するとこの問題は解消されます。

続いて刺激の位置を変更してみましょう。Builder の画面に戻ったら、先ほどの Polygon コンポーネントのプロパティ設定ダイアログを開いて、 図 2.14 上のように **[位置 [x, y] $]** に (0.1, 0) と入力して実行してみましょう。 **[位置 [x, y] $]** の x と y はそれぞれ水平 (X 軸) 方向、垂直 (Y 軸) 方向を表していますので、右と上が正の方向であることに注意すれば、(0.1, 0) はスクリーン中央から右へ 0.1 移動した位置を示しているはずです。実際に実行して確認すると、 図 2.14 下のように確かにスクリーン中央より右寄りに長方形が表示されます。

でも、 図 2.14 を見ただけでは長方形が右寄りに表示されていることがわかりますが、目視しただけでは本当に 0.1 右に寄っているのかの判断は困難です。そもそも、この (0.1, 0) という位置指定は長方形のどこを指しているのでしょうか？ 答えは「レイアウト」タブの **[位置揃え]** にあります。標準では **位置揃え** が「中央」に設定されているので、(0.1, 0) は長方形の中央の座標に対応します。そうすると、長方形の左下の頂点は中央の

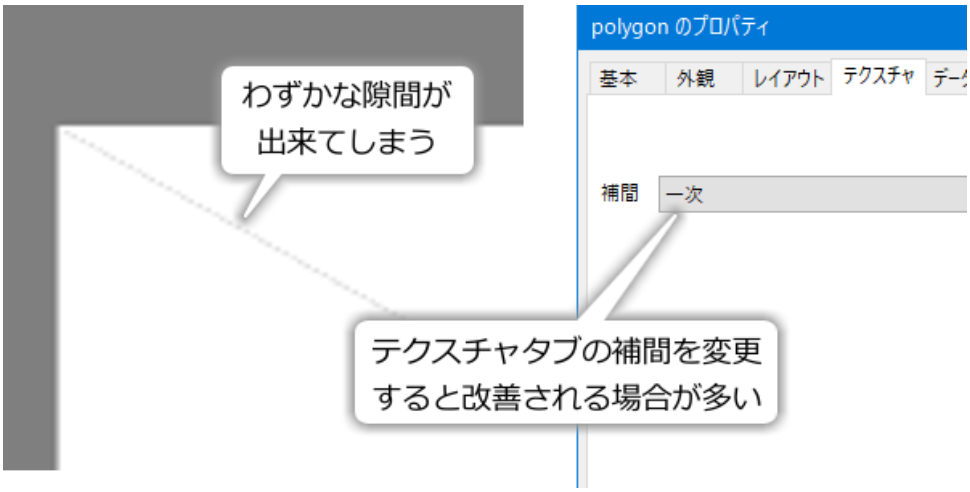


図 2.13　Polygon コンポーネントで長方形を表示すると細い線が見える事があります。「テクスチャ」タブの **[補間]** を変更すると多くの場合問題が解消されます。

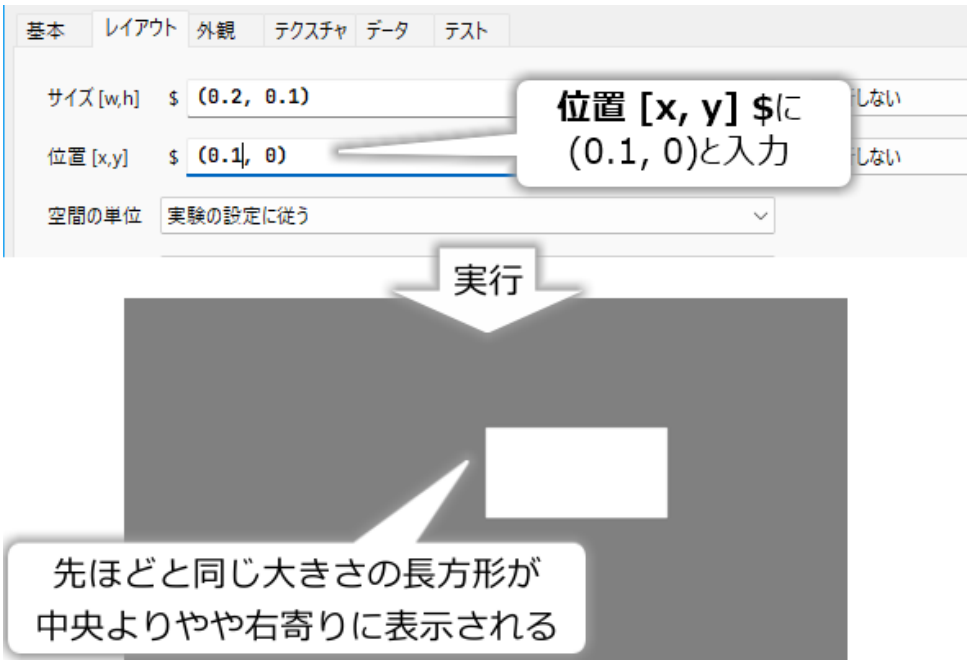


図 2.14　**[位置 [x, y] $]** に (0.1, 0) を設定すると長方形が右寄りに表示されます。

座標 (0.1, 0) から左へ横幅の半分、下へ高さの半分だけ移動した位置にあるはずです。確認するために、ルーチンペインにもうひとつ Polygon コンポーネントを配置してみましょう。

ルーチンにもうひとつ Polygon コンポーネントを配置するには、最初に Polygon コンポーネントを配置した時と同様に、コンポーネントペインの Polygon コンポーネントのアイコンをクリックします。そうするとやはり最初のコンポーネントを配置した時と同様にプロパティ設定ダイアログが表示されます。今回は、Polygon コンポーネントを使用する練習も兼ねて三角形を描画させてみましょう。まず「基本」タブの **形状** が三角形になっていることを確認し、「レイアウト」タブへ移動して 図 2.15 のように **[サイズ [w, h] $]** に (0.2, 0.2)、 **[位置 [x, y] $]** に (0, -0.05) と同じ値を入力してください。そして、 **[位置揃え]** を「中央上」にしましょう。入力ができたら実験を実行してください。 図 2.15 左のように、三角形の上側の頂点が長方形の左下の頂点と一致するはずです。

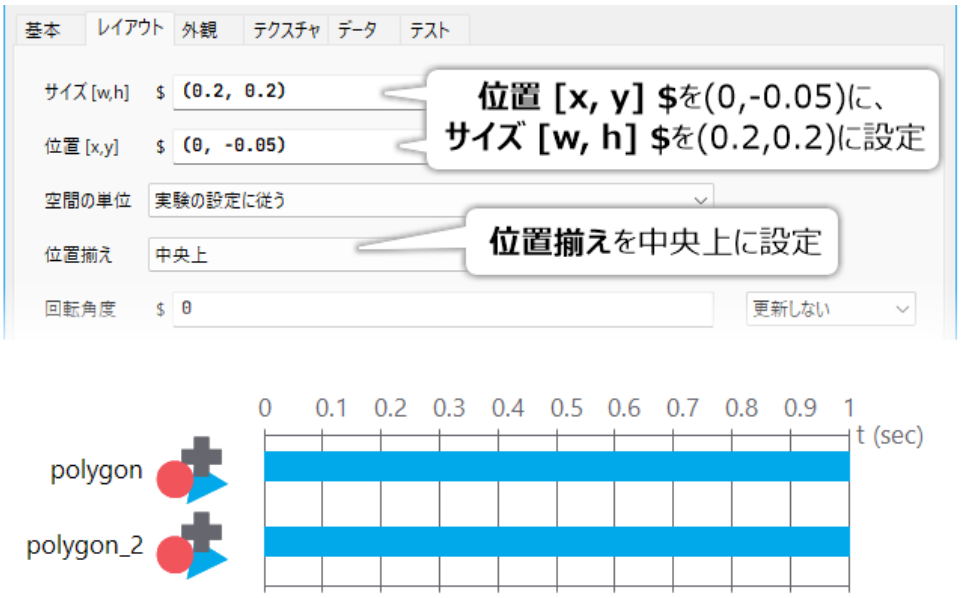


図 2.15　Polygon コンポーネントを追加し、頂点数を 3 にして位置と大きさを設定します。コンポーネントが追加されると下の図のようにルーチンペイン上に複数のアイコンが並びます。

なぜこのようになるのかを解説したのが 図 2.15 の右です。先ほど述べたように、長方形の左下の頂点の座標は (0, -0.05) になるはずです。一方、三角形は 図 2.15 左の結果からおわかりのように底辺が水平な二等辺三角形として描かれます。三角形の **[位置 [x, y] $]** を (0.-0.05) にしたうえで **[位置揃え]** を中央上にしましたので、三角形の上の頂点が (0.-0.05) の位置になるように描かれます。この座標は長方形の左下の頂点の座標と一致しているので、 図 2.15 左のような出力が得られるというわけです。

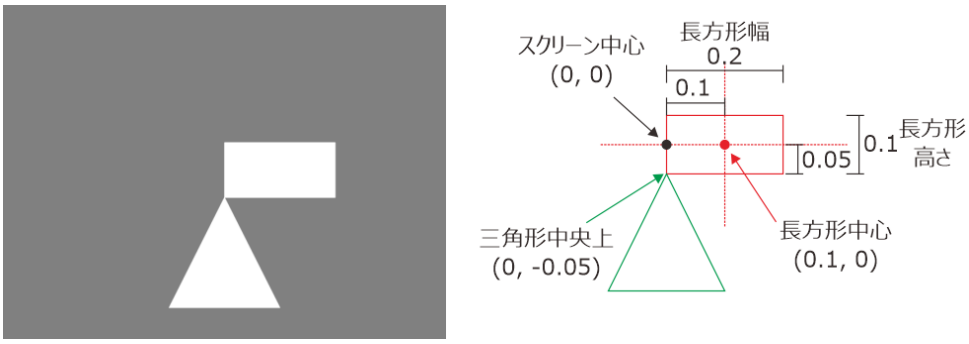


図 2.16　三角形と長方形の Polygon を描画した結果。長方形の左下の頂点と三角形の上の頂点の座標が一致することがわかります。

ここで 図 2.15 右に示されてる三角形が正三角形よりやや縦長である点に注意してください。高さ 0.2 の正三角形の幅は $0.2/\sqrt{3} \times 2 =$ 約 0.2309 ですから、正三角形を表示するには **[サイズ [w, h] $]** を (0.2309, 0.2) としなければいけません。 **[位置揃え]** を中央にした時に **[位置 [x, y] $]** が指し示す位置が三角形の重心と一致しない事にも注意する必要があります。

なお、「 **[位置揃え]** が中央の時に **[位置 [x, y] $]** は 図形の左右端の中点、上下端の中点に対応する」という原則は本書で取り上げる他の視覚刺激に対しても成り立つのですが、Polygon コンポーネントで五角形以上の正多角形を描画した時のみは例外的に、 **[位置 [x, y] $]** が図形の左右端の中点、上下端の中点ではなく、外接する楕円の中心の座標に一致します。

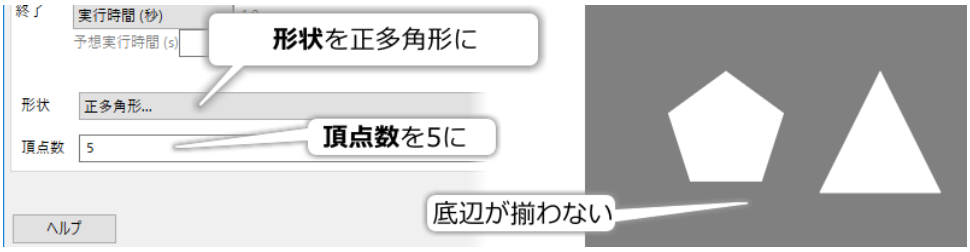


図 2.17　正五角形を描くための設定 (左) と正五角形を描いた結果 (右)。 比較のために同じ **[サイズ [w, h] $]** で描いた三角形を並べて描画しています。

試しに 図 2.17 左のように Polygon コンポーネントの **[形状]** を正多角形にしてください。すると **[頂点数]** の値が変更できるようになりますので 5 を指定してみましょう。指定を終えて実行すると画面上に正五角形が描かれますが、三角形や長方形を描いた時よりやや小さく描かれるはずです。 図 2.17 右には **[サイズ [w, h] $]** が (0.1, 0.1) の五角形と三角形を描画した結果を示していますが、底辺の位置が揃っていないことがわかります。

本節では PsychoPy の標準の単位である height の場合を例に解説してきましたが、他の単位でも考え方は同じです。単位はそれぞれのコンポーネントで独立して設定できるので、ひとつのポリゴンは height、もうひとつのポリゴンは deg を使うといったことも可能です。位置と大きさの指定についての解説はこのくらいにしておいて、次は図形を回転させてみましょう。

チェックリスト

- PsychoPy における座標系の原点と水平、垂直軸の正の方向を答えられる。
- height が単位の状態で **[位置 [x, y] $]** 、 **[サイズ [w, h] $]** 、 **[位置揃え]** を使って任意の大きさの多角形を任意の位置に表示させることができる。
- Polygon コンポーネントで正五角形以上の正多角形を描画できる。
- Polygon コンポーネントで **[頂点数]** が 5 以上の時に **[位置 [x, y] $]** が例外的に図形のどの位置に対応するかを答えることができる。
- cm、deg、norm、height という単位を説明できる。
- 複数のコンポーネントをルーチンペインに配置できる。

2.3 刺激を回転させよう

再び Polygon コンポーネントのプロパティ設定ダイアログを開いてください。 **[位置 [x, y] $]** の上に **[回転角度 $]** という項目があります。この項目に回転量を数値で入力することによって、刺激を回転させることができます。入力する数値の単位は「度」で、時計回りが正の回転方向です。つまり、 **[回転角度 $]** を 30 に設定すると時計回りに 30 度、90 に設定すると 90 度回転します。負の値も指定できますので、-60 を指定すると反時計回りに 60 度回転します。回転の中心は、 **[位置 [x, y] $]** によって指定されている位置です。 図 2.18 に縦長の三角形を 30 度、90 度、-60 度回転させた例を示します。

刺激の回転について学んだついでに、先ほど「norm を単位にすると図形の回転が難しい」と述べた点について確認しておきましょう。Polygon コンポーネントをルーチンペインにひとつ配置して、 **[空間の単位]** を norm に、 **[サイズ [w, h] $]** を [0.5, 0.5] にしてください。そして、 **[回転角度 $]** に 0 を入力した場合と 30 した場合

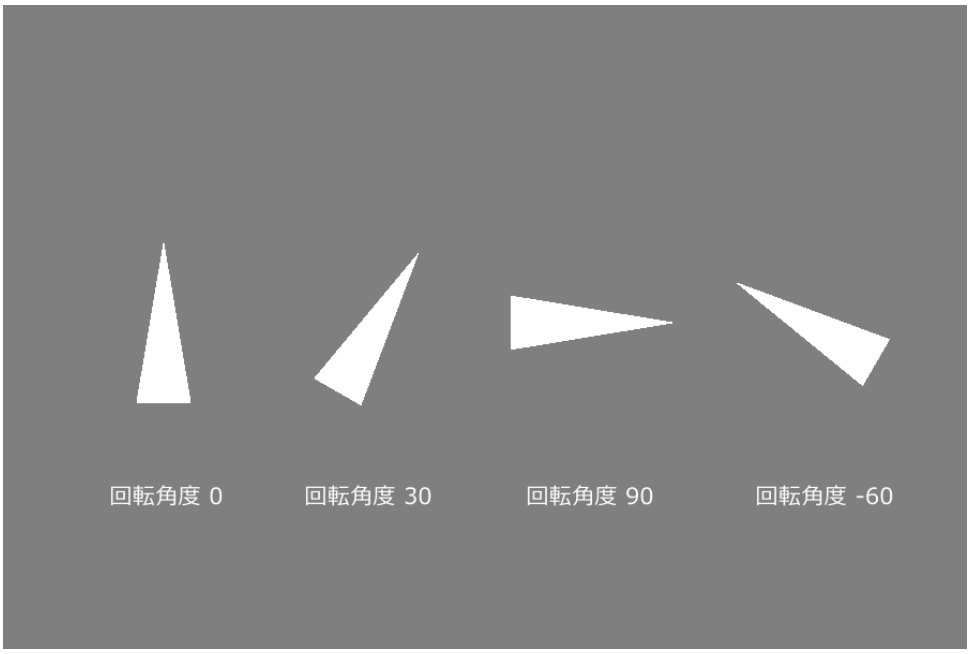


図 2.18　**[回転角度 $]** の指定による図形の回転。

の結果を比較してみてください。他のコンポーネントを置いていても構いませんが、他のコンポーネントと重なるとわかりにくいので削除しておいた方がよいと思います。実行すると、 図 2.19 左のように **[回転角度 $]** が 0 であれば長方形が表示され、30 であれば傾いた平行四辺形が表示されたはずです。これは、norm を単位に使用した時に、スクリーンの右上の座標が (1, 1)、左下の座標が (-1, -1) になるように変換を行うために生じる現象です。通常、PC に接続されているモニターのスクリーンは横に長いので、変換の際に横方向に引き伸ばされてしまうのです。そのため、水平軸や垂直軸に平行な辺しか含まない図形は norm を使用しても単に横長に見えるだけですが、平行ではない辺を含む図形では辺が交わる角度が変わってしまうのです。横長のスクリーンを使用時に norm を使う限り、この問題は回避できません。height を使った方が良いでしょう。

さて、以上で図形の大きさ、位置、回転方向の指定方法の解説が終わりました。これらの設定に用いるプロパティ、 **[サイズ [w, h] $]** 、 **[位置 [x, y] $]**、**[回転角度 $]** 、 **[空間の単位]** の使い方は、視覚刺激を表示するコンポーネントでほぼ共通していますので、使い方をしっかりマスターしておきましょう。

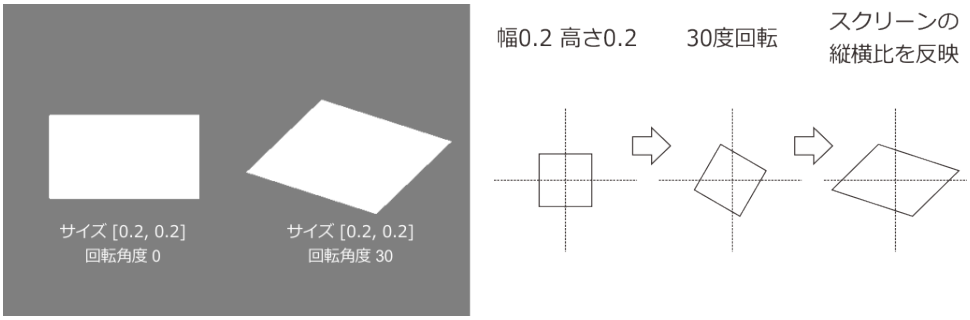


図 2.19　**[空間の単位]** に norm を指定した場合の回転。回転してからスクリーンの縦横比を反映させるため図形が歪みます。

チェックリスト

- **[回転角度 $]** に適切な値を設定して図形を回転させて表示させることができる。

- 図形の正の回転方向を答えられる。
- 単位が norm の時に図形を回転させた時に生じる図形のひずみを説明できる。

2.4 色の指定方法を理解しよう

今度は Text コンポーネントを使う練習をしながら、色の指定方法をマスターしましょう。ルーチンに Text コンポーネント (図 2.20 のアイコン) をひとつ配置してください。Polygon コンポーネントなど他のコンポーネントを配置している人は削除しておいてください。

配置した Text コンポーネントのプロパティ設定ダイアログを開いてください。 **[位置 [x, y] $]** 、 **[回転角度 $]** 、 **[回転角度 $]** 、 **[空間の単位]** は Polygon コンポーネントの同名のプロパティと同じ働きをします。ここではこれらに加えて「基本」タブの **[文字列]** と「書式」タブの **[文字の高さ $]** 、「レイアウト」タブの **[反転]** 、 **[折り返し幅 $]** を使ってみましょう。

図 2.20　Text コンポーネントのアイコン

まず、**[文字列]** に適当な文字列を入力してください。日本語でも英語でも構いませんし、改行しても構いません。そして「書式」タブの **[文字の高さ $]** に 0.02 や 0.04 といった数値を指定して実行してみましょう。スクリーン上に **[文字列]** に入力した文字列が表示されるはずです。 **[文字の高さ $]** を変更して実行し、文字の大きさが変わることを確認してください (MacOS で日本語の文字が欠けてしまう場合は「 2.10.4:Mac 上で日本語の文字が欠けてしまう場合の対策 」をご覧ください)。

[空間の単位] を pix にして **[文字の高さ $]** を 0.1 などにして実行すると、文字が 1 ピクセルより小さくなって何も表示されません。同様に **[空間の単位]** に height が設定されている時に **[文字の高さ $]** を 24 などにしてしまうとスクリーンよりも文字がはるかに大きくなってしまい正常に表示されません。特に数値が大きすぎる場合はエラーダイアログが出て実験自体が実行できない場合があります。ありがちなミスなので注意してください。

なお、「書式」タブの **[言語スタイル]** は文字の書き方を指定します。左から右 (LTR: left to right)、右から左 (RTL: right to left)、Arabic のいずれかです。初期値は LTR で、通常は変更する必要はないでしょう。

続いて「レイアウト」タブの **[反転]** と **[折り返し幅 $]** です。 **[反転]** は None(または空白) にしておくと通常の文字列が表示されますが、vert と入力すると上下反転、horiz と入力すると左右反転して文字列が表示されま

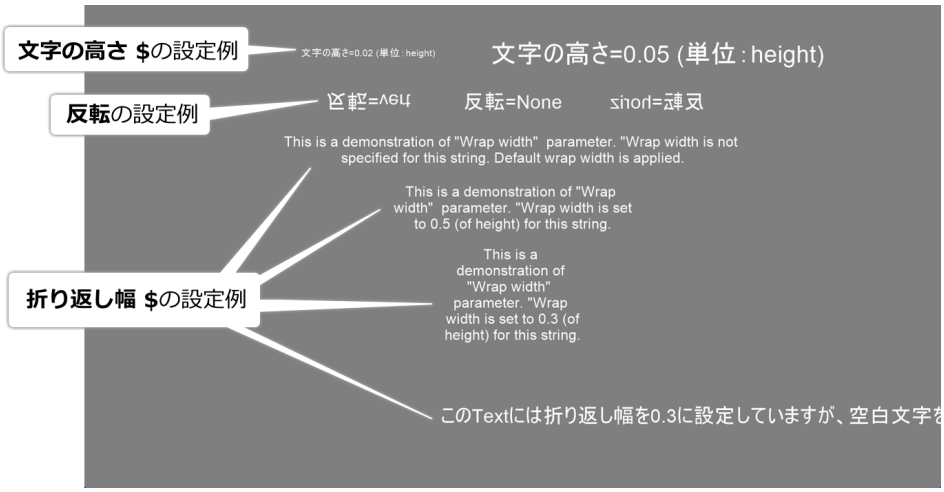


図 2.21　Text コンポーネントにおける文字の高さ、反転、折り返し幅の設定例。

す (図 2.21)。 **[折り返し幅 $]** は、 **[文字列]** に改行を含まない長い文字列が入力されたときに自動的に折り返す幅を指定します。折り返し幅の単位は **[空間の単位]** プロパティに従います。 図 2.21 では **[折り返し幅 $]** を指定しなかった場合、0.5 を指定した場合、0..3 を指定した場合を示しています。いずれも **[文字列]** には改行を含めずに文を入力してあるのですが、適切に折り返しが行われていることがわかります。ただし残念なことに、文字列の自動折り返しは日本語ではうまく機能しません。 図 2.21 の一番下の日本語の文字列は **[折り返し幅 $]** に上の英文と同じ 0.3 を指定しているのですが、折り返されずに画面からはみ出してしまっています。

Text コンポーネントに慣れたところで、次は色の指定をしてみましょう。色を指定するには「外観」タブの **[前景色]** に値を設定します。 **[前景色]** の入力欄の右側に小さなボタンがありますが (図 2.22)、これをクリックすると「カラーピッカー」という色選択ツールを開くことができます。カラーピッカーは便利なのですが、PsychoPy における色指定の仕組みを理解していないとわかりにくい内容もありますので、まずは色指定の仕組みを学びましょう。

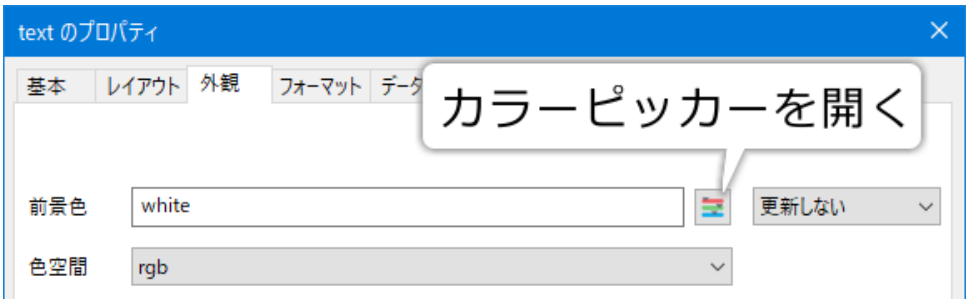


図 2.22　色を指定する項目には右側に「カラーピッカー」を開くボタンがあります

PsychoPy では、色を表す値として web/X11 Color name と呼ばれる色名と、16 進数表記の web カラーと、色空間における座標値を利用することができます。図 2.23 は web/X11 Color name の一覧を示しています。先ほどの Polygon コンポーネントの表示で **[塗りつぶしの色]** に black と書くことで黒く塗りつぶすことができたのは、この web/X11 Color name による指定が利用できるからです。図 2.23 を見ながら、Text コンポーネントのプロパティ選定ダイアログの **[前景色]** に色名を入力して実行してみましょう。

16 進数表記の web カラーというのは、0xFD087A や#FAF のように、「0x または#」+「0 から 9 の数字および A から F のアルファベット文字を 3 文字または 6 文字」で色を表す方法です。web ページを作成するときによく用いられる色指定なので、そちらですでにこの指定方法をご存じの方には使いやすいでしょう。しかし、ご

black	aliceblue	darkcyan	lightyellow	coral
dimgray	lavender	teal	lightgoldenrodyellow	tomato
gray	lightsteelblue	darkslategray	lemonchiffon	orangered
darkgray	lightslategray	darkgreen	wheat	red
silver	slategray	green	burlywood	crimson
lightgray	steelblue	forestgreen	tan	mediumvioletred
gainsboro	royalblue	seagreen	khaki	deeppink
whitesmoke	midnightblue	mediumseagreen	yellow	hotpink
white	navy	mediumaquamarine	gold	palevioletred
snow	darkblue	darkseagreen	orange	pink
ghostwhite	mediumblue	aquamarine	sandybrown	lightpink
floralwhite	blue	palegreen	darkorange	thistle
linen	dodgerblue	lightgreen	goldenrod	magenta
antiquewhite	cornflowerblue	springgreen	peru	fuchsia
papayawhip	deepskyblue	mediumspringgreen	darkgoldenrod	violet
blanchedalmond	lightskyblue	lawngreen	chocolate	plum
bisque	skyblue	chartreuse	sienna	orchid
moccasin	lightblue	greenyellow	saddlebrown	mediumorchid
navajowhite	powderblue	lime	maroon	darkorchid
peachpuff	paleturquoise	limegreen	darkred	darkviolet
mistyrose	lightcyan	yellowgreen	brown	darkmagenta
lavenderblush	cyan	darkolivegreen	firebrick	purple
seashell	aqua	olivedrab	indianred	indigo
oldlace	turquoise	olive	rosybrown	darkslateblue
ivory	mediumturquoise	darkkhaki	darksalmon	blueviolet
honeydew	darkturquoise	palegoldenrod	lightcoral	mediumpurple
mintcream	lightseagreen	cornsilk	salmon	slateblue
azure	cadetblue	beige	lightsalmon	mediumslateblue

図 2.23　PsychoPy で使用できる色名 (web/x11 color name)

存じでない方は次に紹介する色空間における座標値を指定する方法を覚えた方が良いと思います。web カラーによる色指定については「2.10.5:16 進数と色表現」で解説していますので詳しくはそちらをご覧ください。

さて、色空間における座標値を指定する方法ですが、これは人間が知覚できる色が三次元空間の点として表現できることを利用しています。ちょっと数学的な話になりますが、空間の位置を表現する方法は何通りもあります。例えば二次元平面の水平方向に X 軸、垂直方向に Y 軸を引いて「原点から X 軸の方向に 10、Y 軸の方向に 10 進む」といった具合に平面上の位置を表現することができますが、同じ位置を「原点から 45 度の方向に $10\sqrt{2}$ 進む」と表現することもできます。前者を直交座標、後者を極座標と呼びますが、同じ位置でも直交座標と極座標では異なる数値で表されるわけです。これと同様に、色の表現も座標軸の取り方によって同一の色に対して複数の方法で表現することができます。PsychoPy では、RGB、HSV、LMS、DKL という 4 種類の表現をサポートしています。しかし、HSV は Builder からは使用できず、LMS と DKL は専用の装置を用いて実験に使用するモニターをキャリブレーション (調整) しないと使えませんので、本書では RGB による

表現を使用します。この色表現を切り替えるのがプロパティ設定ダイアログの **[色空間]** です。 **[色空間]** の値が rgb に設定されていることを確認しておきましょう。なお、 **[色空間]** の値は web/X11 color name や web カラーで色を指定する時には無視されます。

ようやく色空間における座標値で色を指定する方法を説明する準備ができました。 **[色空間]** を rgb に設定している場合、赤 (R)、緑 (G)、青 (B) の三種類の光の強度の組み合わせで色を指定することができます。RGB のそれぞれの成分の強度は-1.0 から 1.0 の実数で指定します。Text コンポーネントのプロパティ設定ダイアログを開いて **[前景色]** に

```
$[-1, -1, -1]
```

と記入して実行してみましょう。カンマや括弧、$記号もこの通りに入力してください（括弧は丸括弧でも構いません)。黒色で文字が表示されるはずです。続いて以下の三つを順番に試してみましょう。

```
$[ 1, -1, -1]
$[-1,  1, -1]
$[-1, -1,  1]
```

上から順番に赤色、緑色、青色で文字が表示されたはずです。三つの数字が左から順番に R、G、B に対応しているのが理解していただけたでしょうか。さらに以下の値も試してみましょう。これらがどのような色になるかは実際に皆さんが確認してみてください。

```
$[-0.3, -0.3, -0.3]
$[0.2, 0.2, 0.2]
$[-0.92, -0.46, 0.05]
$[0.09, 0.63, 0.13]
```

なお、一般的なグラフィックソフトウェアでは RGB それぞれ 256 段階の整数で指定する表現方法が用いられていますが、この 256 段階表現の色を PsychoPy で使用するには-1.0 から 1.0 の実数に換算する必要があります。256 段階表現の場合、RGB 各成分の最小値は 0 で最大値は 255 ですから、値を 255.0 で割れば 0.0 から 1.0 の値が得られます。これを-1.0 から 1.0 に変換すればいいのですから、2 倍して 1.0 を引けば目的が達成されます。式で書けば以下の通りです。

```
2 × (256 段階表現の値÷ 255.0)-1.0
```

慣れないうちは狙った色を指定するのは難しいですが、 図 2.22 で触れたカラーピッカーがここで役に立ちます。 図 2.22 のボタンをクリックすると、 図 2.24 に示すカラーピッカーダイアログが表示されます。ダイアログ中央の RGB Channels という枠内にある **[R]** 、 **[G]** 、 **[B]** のスライダーを調整すると、それに対応する色でダイアログ左側の領域が塗りつぶされます。文章で説明するより触ってみた方が早いと思いますので、ぜひ自分で操作してみてください。

ダイアログ右側に並んでいる色見本をクリックすると、その色の RGB 値がスライダーに反映されます。色名と RGB 値がどのように対応しているかを確認したり、色名で表される色をほんの少し変更した色を指定したりするときに便利です。「この色を使いたい！」という色ができあがったら、ダイアログ左下の **[出力空間]** が PsychoPy RGB (rgb) になっていることを確認したうえで、右下にある OK ボタンをクリックしましょう。するとダイアログが閉じると同時に、 **[前景色]** の欄に作成した色の RGB 値が自動的に挿入されます。-1.0, 0.0,

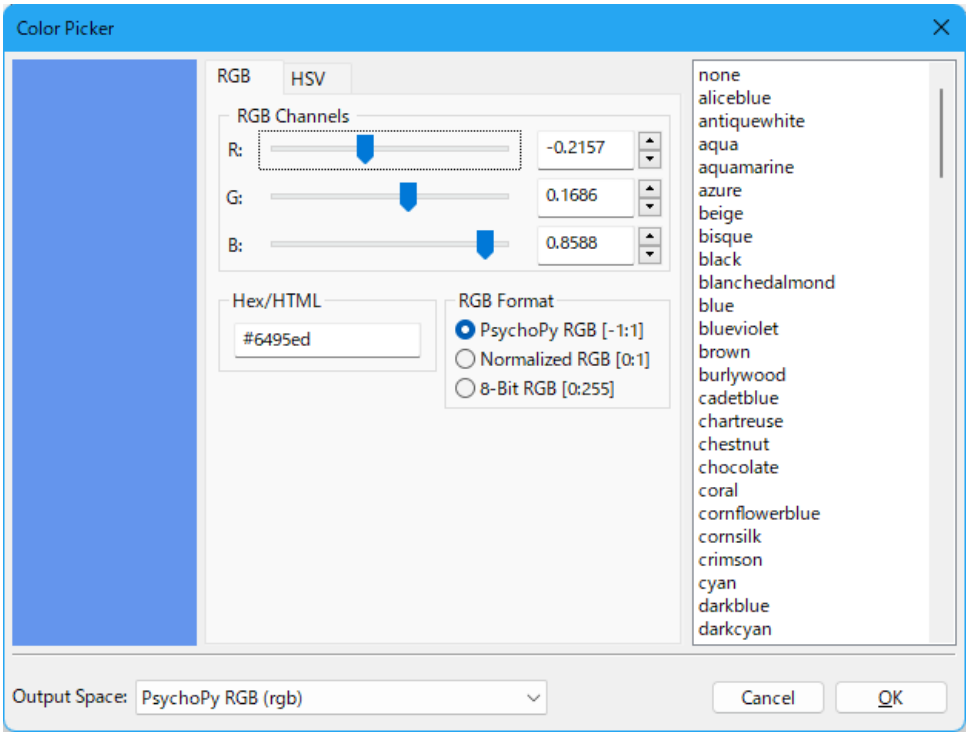


図 2.24　カラーピッカーダイアログ

0.0 といった具合に$記号や括弧がついていませんので、手作業で **$**(-1.0, 0.0, 0.0) といった具合に$記号と括弧を追加する必要があります (**バージョン 2022.1.1 以降の PsychoPy では$と括弧を追加しなくても認識するようになりました**)。色見本の色名を自動挿入することはできませんので、色名で指定したい場合は色名を覚えるかメモしたうえでダイアログを閉じ、手作業で色名を入力する必要があります。

最後に、Polygon コンポーネントの色指定について補足しておきます。Polygon コンポーネントには色指定に関して **[塗りつぶしの色]** と **[枠線の色]** という 2 つのプロパティがあり、それぞれ塗りつぶしと枠線の色に対応しています。これらの色に None という値を指定することによって、内部が塗りつぶされていない枠線だけの図形や、枠線がない図形を描画することができます。 **[塗りつぶしの色]** を None にすると内部が塗りつぶされていない枠線だけ、**[枠線の色]** を None にすると枠線がない図形になります。ぜひ覚えておいてください。

チェックリスト

- Text コンポーネントを用いて文字列を表示できる。
- 文字列を指定された位置に表示できる。
- 文字列を指定された大きさで表示できる。
- 文字列を上下反転、左右反転表示することができる。
- 文字列の自動折り返し幅を設定できる。どのような文字列では自動折り返しが起きないか説明できる。
- web/X11 color name による色指定で文字列の色を白、灰色、黒、赤、オレンジ色、黄色、黄緑色、緑、水色、青、ピンク、紫にすることができる。
- **[色空間]** を rgb に設定して、数値指定によって文字列の色を白、灰色、黒、赤、黄色、緑色、青色

にすることができる。

- Polygon コンポーネントを用いて内部が塗りつぶされていない枠線だけの多角形を描画することができる。
- Polygon コンポーネントを用いて枠線がない多角形を描画することができる。

2.5 刺激の重ね順と透明度を理解しよう

刺激を色分けできるようになりましたので、刺激が重なってしまった時にどのような結果が得られるのかを解説できるようになりました。さっそく、刺激の重ねあわせについて解説しましょう。

Polygon コンポーネントひとつと Text コンポーネントひとつをルーチンペインに配置して、以下のように設定します。

- Polygon コンポーネント
  - **「基本」タブ**
    - **[形状]** を長方形にする
  - **「レイアウト」タブ**
    - **[サイズ [w, h] $]** を [0.2, 0.2] にする
  - **「外観」タブ**
    - **[塗りつぶしの色]** を red にする
  - 他のプロパティは初期値のままにする
- Text コンポーネント
  - **「基本」タブ**
    - **[文字列]** を「PsychoPy Builder による心理学実験」にする
  - **「書式」タブ**
    - **[文字の高さ $]** を 0.02 にする
  - 他のプロパティは初期値のままにする

どちらのコンポーネントを先にルーチンペインに配置したかによって、各コンポーネントのアイコンが並ぶ順番が異なります。先に配置したコンポーネントが上にあって、その下に配置した順番にアイコンが並びます。今までルーチンペイン上におけるアイコンの順番については触れませんでしたが、実はこの順番には大きな意味があります。 図 2.25 をご覧ください。Builder では、ルーチンペインで上に配置されているコンポーネントから順にスクリーン上に表示します。ですから、ルーチンペイン上で下に配置されているコンポーネントほど重ね順は上になります。重ね順で上にあることを「手前にある」、下にあることを「奥にある」という言い方をすることもあります。

ルーチンペイン上での配置順を変更するには、変更したいコンポーネントのアイコン上へマウスカーソルを動かして、右クリックをしてメニューを表示させます。ここまではコンポーネントを削除する時の操作と同じです。削除する時にはメニューの「削除」という項目を選択しましたが、配置順を変更する時には「ひとつ上へ」、「ひとつ下へ」、「一番上へ」、「一番下へ」を選択します。

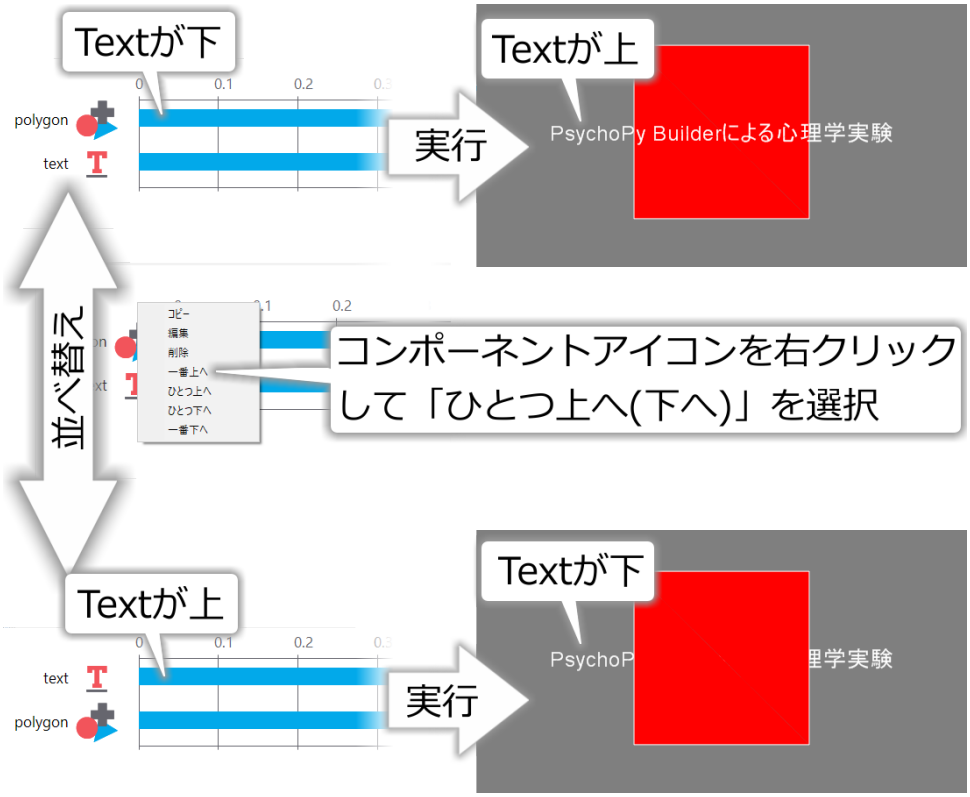


図 2.25　ルーチンペインにおける順序と刺激の重ねあわせの関係。ルーチンペインで上の方に配置されている刺激から順に表示されますので、スクリーン上での刺激の重ね順では上に配置されている刺激ほど下になります。

刺激の重ね順を解説したついでに、 **[位置 [x, y] $]** などと同様に多くの視覚刺激用コンポーネントで使用できる **[不透明度 $]** を紹介します。 **[不透明度 $]** は刺激の透明度を指定するプロパティで、0.0 から 1.0 の値をとります。0.0 は完全な透明で、スクリーン上では見えなくなってしまいます。1.0 は完全な不透明で、重ね順で下にある刺激は見えません。 図 2.26 では、文字列の上に赤い正方形を重ねて、正方形の透明度を 1.0、0.75、0.5、0.25、0.0 と変化させています。簡単に試すことができると思いますので、ぜひ各自でいろいろな値を試してみてください。

チェックリスト

- ルーチンペイン上における視覚刺激コンポーネントの順番とスクリーン上での重ね順の関係を説明できる。
- ルーチンペイン上におけるコンポーネントの順番を変更できる。
- 視覚刺激コンポーネントの透明度を設定して完全な透明、完全な不透明とその中間の透明度で刺激を表示させることができる。

図 2.26　**[不透明度 $]** による透明度の指定。赤い正方形が文字列の上に重ねて、その赤い正方形の **[不透明度 $]** を段階的に変化させています

2.6 刺激の提示開始と終了時刻の指定方法を理解しよう

この節では、刺激がいつ画面上に表示されて、いつ消えるかという時間的な側面を設定する方法について解説します。ここまで使用してきた Polygon コンポーネントと Text コンポーネントのプロパティ設定ダイアログの「基本」タブを見比べてみると、どちらにも **[開始]** と **[終了]** というプロパティが存在しているのがわかります (図 2.27)。これらが刺激の提示開始および終了を決めるプロパティです。解説に入る前に、ちょっと **[開始]** の上にある **[名前]** プロパティの使い方にも触れておきましょう（ **[名前]** については次章で詳しく解説する予定です)。

まず、Polygon コンポーネントを二つルーチンペイン上に配置して、二つの正方形が左右に隙間なく並ぶようにしてみましょう。ここでは以下のように設定したとします。

- Polygon コンポーネントその 1 (赤)
  - **「基本」タブ**

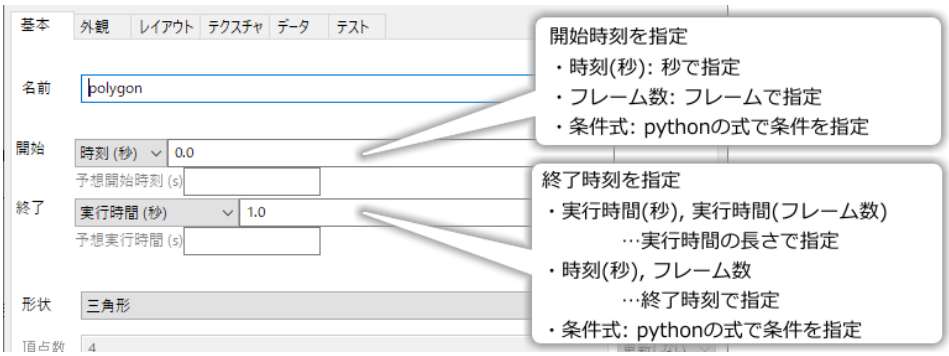


図 2.27　コンポーネントの開始、終了時刻を指定するプロパティ。「条件式」の使い方については 第 9 章 で触れます。

    - **[名前]** に red と入力

  - **「レイアウト」タブ**
    - **[サイズ [w, h] $]** を (0.2, 0.2)、 **[位置 [x, y] $]** を (-0.1, 0) にする
  - **「外観」タブ**
    - **[塗りつぶしの色]** を red にする
  - 他のプロパティは初期値のままにする

- Polygon コンポーネントその 2 (緑)
  - **「基本」タブ**
    - **[名前]** に green と入力
  - **「レイアウト」タブ**
    - **[サイズ [w, h] $]** を (0.2, 0.2)、 **[位置 [x, y] $]** を (0.1, 0) にする
  - **「外観」タブ**
    - **[塗りつぶしの色]** を green にする
  - 他のプロパティは初期値のままにする
- 緑色の長方形が上に描画されるようにルーチンペイン上でのアイコンの順番を並べる。

[名前] を設定すると、 図 2.28 のようにルーチンペイン上で **[名前]** に設定した文字列が各コンポーネントのアイコンの左側に表示されます。同じ種類のコンポーネントが複数配置されている場合に区別しやすくてとても便利です。

さて、この状態で実験を実行すると、赤と緑の正方形がスクリーンに 1 秒間表示されて終了するはずです。Builder の画面に戻ったら、ルーチンペイン上に配置した赤い正方形のプロパティ設定ダイアログを開き、 図 2.29 左のように **[開始]** を「時刻 (秒)」にして 0.5 と入力し、 **[終了]** を「実行時間 (秒)」にして 2 と入力してください。 **[開始]** と **[終了]** はそれぞれ初期状態で「時刻 (秒)」、「実行時間 (秒)」になっているはずですので、変更していないのであればそれぞれ 0.5 と 2 を入力すれば大丈夫です。プロパティ設定ダイアログの OK をクリックしてダイアログを閉じると、ルーチンペイン上の表示が 図 2.29 右のように変化しているはずです。アイコンの横の青い横棒はコンポーネントが有効になる時間帯、視覚刺激の場合は画面上に表示されている時間

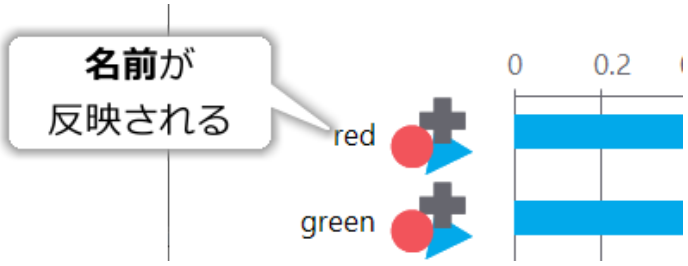


図 2.28　**[名前]** プロパティに文字列を入力すると、ルーチンペイン上でコンポーネントのアイコンの左側に入力した文字列が表示されます。

帯を示しています。開始時刻に 0.5 秒を指定したので、青棒の左端は 0.5 の位置にあります。青棒の右端は終了時刻に対応していますが、こちらは少し説明が必要でしょう。 **[終了]** は「実行時間 (秒)」を指定して 2 と入力してありますので、刺激が画面上に表示されている時間は 2 秒です。刺激の表示開始時刻が 0.5 秒なのですから、終了時刻は 0.5 秒から 2 秒後の 2.5 秒でなければいけません。ルーチンペインの青棒の右端を確認すると、確かに右端は 2.5 秒の位置にあります。実験を実行してみると、最初に緑色の正方形のみが表示された後、一瞬 (0.5 秒) 遅れて赤い正方形が出現し、さらにすぐ後に緑色の正方形がスクリーンから消えます。赤い正方形は 2 秒間スクリーンに表示された後に消えて、その直後に実験が終了します。

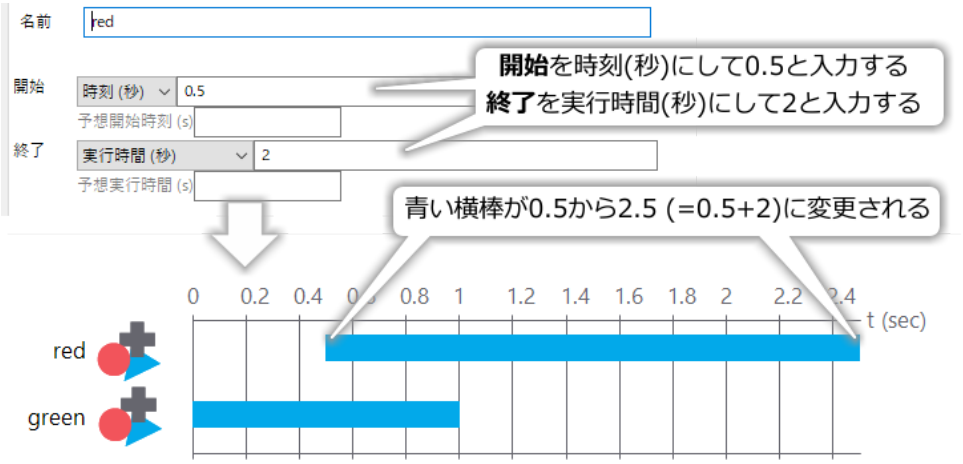


図 2.29　**[開始]** と **[終了]** の値を変更すると、ルーチンペイン上で青いバーの長さが変更されます。青いバーは実験実行時に、そのコンポーネントが有効となる (視覚刺激の場合は表示される) 時間帯を示しています。

基本的にこれだけで刺激の表示開始時刻と終了時刻を制御できますが、Builder では他の方法も提供されています。まず、 **[終了]** で「実行時間 (秒)」の他に「時刻 (秒)」を選択することができます。こちらを選択すると、終了時刻を直接入力して指定することができます。 図 2.29 の例で **[終了]** を「時刻 (秒)」に変更し、値に 2.5 を入力してみてください。ルーチンペインの青棒は 図 2.29 と同じになり、実行結果も同じになるはずです。ぜひ皆さん自身で手を動かして確認してみてください。

他には、刺激の表示開始、終了時刻を秒ではなくフレーム数で指定する方法があります。フレーム数で指定する場合も秒と同様に **[終了]** の項目で表示する時間の長さを指定するか、終了する時刻を直接指定するかを選択できます。「『フレーム数で指定』と言われてもよくわからない」という方は、「 2.10.6: 時刻指定における frame について 」をご覧ください。 図 2.27 で灰色の文字で描かれている **[予想開始時刻 (s)]** 、 **[予想実行時間 (s)]** という項目はフレーム数で指定する際に使う項目なので、そちらで合わせて説明してあります。

[開始] 、 **[終了]** ともに、秒による指定、フレーム数による指定に加えて「条件式」という選択肢がありますが、

これは Python の条件式を直接記入する方法です。使いこなすには Python の文法を知っていなければいけませんので、ここでは一旦無視して 第 9 章 であらためて取り上げます。

最後に、非常に重要なテクニックをひとつ紹介しておきましょう。 **[終了]** の項目の数値を入力する欄を空白 (入力済みの数値を削除) してみてください。ルーチンペイン上でコンポーネントの有効時間帯を示す青い横棒が右側へ突き抜けてしまったはずです (図 2.30)。この状態になると、何らかの方法でルーチンが強制終了されない限り、この刺激は画面上に表示され続けます。この章で今まで製作してきたシンプルな「実験」では、すべてのコンポーネントの終了時刻が決められていました (1.0 秒)。Builder の実験を実行した時には、ルーチン内に含まれるすべてのコンポーネントの終了時刻を経過したらルーチンが自動的に終了し、すべてのルーチンを実行すれば実験は自動的に終了します。 図 2.30 のように終了時刻が定められていないコンポーネントが存在すると、ルーチンが「永遠に」終了しません。現実には OS が再起動したり PC の電源が切れたりしていずれは終了してしまうでしょうが、そういう事態でもない限り刺激が表示され続けます。誤ってルーチンが終了しない状態に陥ってしまった時には、焦らずにキーボードの ESC キーを押してください。Builder の標準設定では、ESC キーが押されると直ちに処理中のルーチンを中断して実験を終了します。

終了時刻を定めないコンポーネントが定義できるようになっているのは、「実験参加者が反応するまで刺激を提示し続ける」といった実験手続を実現する為です。次章ではキーボードからの反応を取得する方法を学びますが、「キーボードが押されたらルーチンを終了する」という設定と、「ルーチンが終了するまで刺激を提示し続ける」という設定を組み合わせれば「実験参加者が反応するまで刺激を提示し続ける」ことが実現できるのです。詳しくは 第 3 章 で説明します。

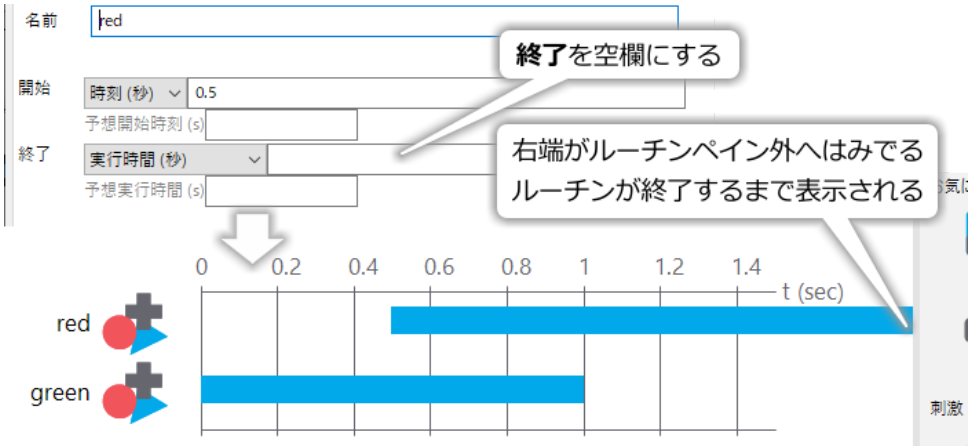


図 2.30 **[終了]** を空白にしておくと、ルーチンの終了までコンポーネントが有効になります。何らかの方法でルーチンを終了させない限り刺激は表示され続けます。標準設定では ESC キーを押すと強制的に実験を終了させることができます (ルーチンも強制終了されます)。

チェックリスト

- 刺激の表示開始時刻と表示時間を指定して表示させることができる。
- 刺激の表示開始時刻と表示終了時刻を指定して表示させることができる。
- 刺激の表示終了時刻を定めずに表示させることができる。
- 実行中の実験を強制的に終了させることができる。

2.7 Builder が作成するファイルを確認しよう

「刺激の位置や提示時間を指定する方法を覚える」というこの章の内容はほぼ終わりました。最後に実験の基本設定を行う方法を解説したいのですが、その前に Builder が作成するファイルとフォルダについて簡単に触れておきます。

ここまで作業の作業を進めた後で exp01.psyexp を保存したフォルダを確認すると、data というフォルダと exp01_lastrun.py というファイルができているはず。もし psyexp ファイルを exp01 以外の名前で保存したのでしたら、_lastlan.py の前の部分が保存したファイル名に対応した文字列になっているはずです。

exp01_lastrun.py は Builder が psyexp ファイルを「翻訳」して作成した Python のスクリプトです。メモ帳などのテキストエディタで開いてみると内容を確認することができます。Python を用いて実験をするとは元々このようなファイルを自分で書くということであり、その作業を人の代わりに行ってくれるのが PsychoPy Builder だというわけです。ただし、Builder が生成するスクリプトは人が書く場合に比べて少々冗長ですので、人が書けばもっと短いスクリプトで実現することも可能です。このファイルは実験を実行する度に自動的に作成されるので、実験終了後に削除してしまっても問題ありません。

data フォルダは、実験結果を記録したファイルが保存されるフォルダです。実験を一回実行する度に複数のファイルが作成されるので、もしここまで一気に作業してこられたのでしたら非常にたくさんのファイルが作成されているはずです。このフォルダ内のファイルには実験結果が記録されています。この章で作業した内容は特に記録する必要はありませんので、data フォルダごと削除してしまって構いません。ファイルの内容については 第 3 章 以降で詳しく見ていきます。

最後に、psyexp ファイルそのものについて少し補足しておきましょう。Builder で作成した実験の内容はすべてこのファイルに保存されていますので、実験が不要にならない限りこのファイルを削除してはいけません。psyexp ファイルの他に、第 3 章 で解説する条件ファイルも psyexp ファイルと一緒に保存しておく必要があります。刺激として画像ファイルや音声ファイルを使用する場合は、それらのファイルも忘れずに保存しておかなければいけません。なお、psyexp ファイルは XML 形式と呼ばれるデータ形式で保存されたファイルなので、メモ帳などのテキストエディタを使って開くと中身を見ることができます。この方法を使って Builder を使わずに直接実験を編集することも可能です。第 13 章 でそういったテクニックも紹介します。

保存した psyexp ファイルは、 図 2.31 の「psyexp ファイルを開く」ボタンをクリックすると Builder で開くことができます。作成途中で保存した psyexp ファイルを開いたり、完成した psyexp ファイルを使って実験したりする時に使います。OS によっては psyexp ファイルのアイコンをダブルクリックするだけで自動的に Builder を起動してファイルを開くこともできます。作業を保存する時は「上書き保存」ボタン、別の名前で保存したいときには「名前を付けて保存」ボタンを使います。現在作成中の実験を置いておいて新たに実験を作成したい場合は「実験の新規作成」ボタンを使います。他にも「元に戻す」と「やり直す」ボタンも便利ですので一緒に覚えておくとよいでしょう。

チェックリスト

- foo_lastrun.py (foo は psyexp 実験ファイル名) の役割を説明することができる。
- data フォルダの役割を説明することができる。
- 実験結果を保存する必要がない場合、どのファイルを削除しても問題ないかを判断できる。
- 作製済みの psyexp ファイルを Builder で開くことができる。

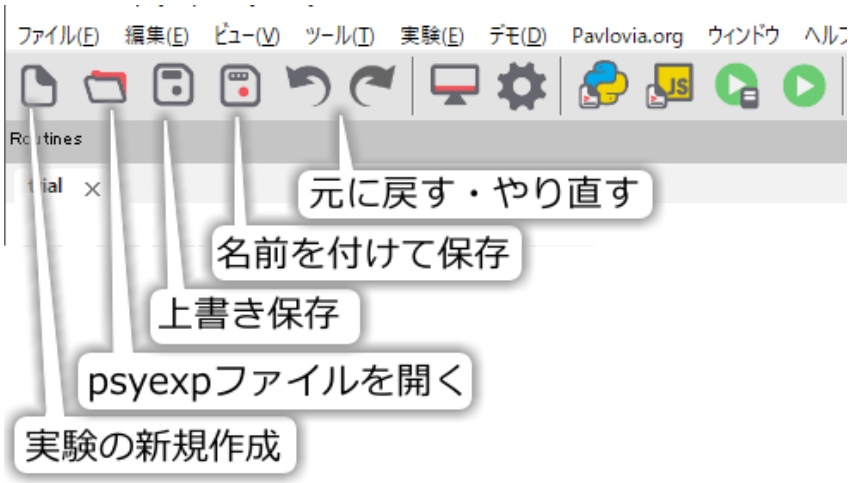


図 2.31　ファイル操作に関するボタンと元に戻す・やり直すボタン。

- psyexp ファイルを別の名前で保存することができる。

2.8 実験の設定を変更しよう

これで刺激の描画方法の基礎を一通り解説しました。本格的な実験の作成に入る前に、実験の設定について解説しておきます。Builder ウィンドウ上部のツールバーの 図 2.32 に示したアイコンをクリックすると、実験設定ダイアログが開きます。このダイアログには「基本」、「データ」、「スクリーン」の三つのページがあり、非常に多くの項目が含まれています。「基本」から順番に見ていきましょう。

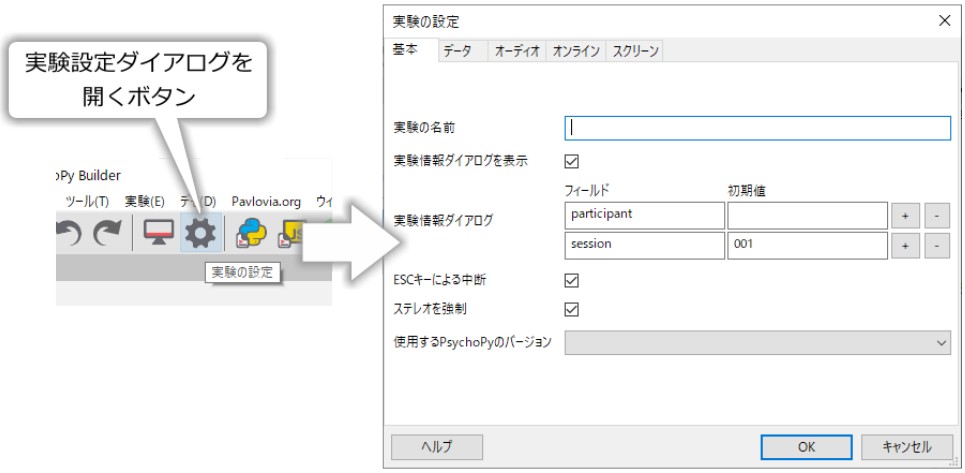


図 2.32　Builder ウィンドウ上部のツールバーのボタンから実験設定ダイアログを開くことができます。

2.8.1 「基本」タブ

[実験の名前] 実験の名前を入力します。実験結果の記録ファイルに反映されるため、データ整理の際に便利でしょう。ファイル名として使用できる文字列でなければいけません。日本語の文字の使用は避けた方が無難です。

[使用する PsychoPy のバージョン] 旧バージョンの PsychoPy で作成した実験が現在使用中のバージョンで動かないときに、旧バージョンで実行するように指定することができます。この機能を利用するためには、git というコマンドが使用できるように設定しておく必要があります。利用のためには他にもいろいろと条件があるので、初心者の方には利用をお勧めしません。

[実験情報ダイアログを表示] 実験実行時に表示される実験情報ダイアログの表示、非表示を指定します。チェックを外しておくと実験情報ダイアログが表示されません。

[ESC キーによる中断] 「2.6: 刺激の提示開始と終了時刻の指定方法を理解しよう」で触れた、ESC キーによる実験の強制終了を有効にするか無効にするかを指定します。実験中に実験参加者が誤って ESC キーを押してしまう恐れがある場合はチェックを外しておくべきですが、チェックを外してしまうと強制終了ができなくなりますので注意してください。実験が完成して、十分に動作確認をして問題がないことを確認してからチェックをはずすとよいでしょう。なお、フルスクリーンモードを使用していない場合は、実験実行中に Runner のウィンドウを選択して実験中断ボタンをクリックしても強制終了することができます。

[実験情報ダイアログ] 実験情報ダイアログに表示する項目を設定します。詳しくは 第 4 章 を参照してください。

2.8.2 「スクリーン」タブ

[モニター] 使用するモニターを指定します。PsychoPy ではモニターの設定に名前をつけて保存しておくことができますが、ここでは使用するモニター設定の名前を入力します。モニター設定の作成方法はこの章の最後に触れます。

[スクリーン] 複数台のモニターが接続された PC を使用する場合、どのモニターを視覚刺激提示に使用するかを番号で指定します。モニター番号がわからない場合は、「スクリーン番号の表示」をクリックして各モニター上に番号を表示させて調べることができます。

[フルスクリーンウィンドウ] 刺激提示にモニターのスクリーンいっぱいに広がったウィンドウを用いるか否かを指定します。チェックが入っていると、スクリーン全体が Builder の実験ウィンドウで覆われて、他のアプリケーションやデスクトップは見えなくなります。この状態をフルスクリーンモードと呼びます。チェックを外すと、視覚刺激提示用に通常のアプリケーションのようなウィンドウが開いて、そのウィンドウ内に刺激が提示されます。一般論として、フルスクリーンモードの方が実験実行時の時間的な精度が高い傾向にあります。何らかの理由があって通常のアプリケーションウィンドウで実行したい場合を除いて、この項目はチェックしておくべきです。

[ウィンドウの大きさ (pix) $] フルスクリーンウィンドウを使用しない時に、刺激提示用ウィンドウの幅と高さを指定します。書き方は視覚刺激の大きさの指定と同様 [1920, 1080] といった具合に幅と高さの値をカンマで区切って書き、角括弧で囲みます。単位は pix です。フルスクリーンウィンドウ使用時にはこ

の項目は灰色に表示されていて編集できません。フルスクリーンウィンドウ使用時のスクリーンの解像度は OS による解像度の設定に従います。

[マウスカーソルを表示] この項目をチェックしておくと、フルスクリーンモードでの実験実行時にもマウスカーソルが表示されます。標準ではチェックされていません。マウスを用いて参加者の反応を記録する実験を実施する場合などに使います。

[単位] 視覚刺激コンポーネントで用いられる標準の単位を指定します。具体的には、 表 2.1 に示した単位のうち「実験の設定に従う」を選択した際に使用される単位を指定します。個々のコンポーネントで「実験の設定に従う」以外の単位を選択した場合は、そちらが優先されます。単位の選択肢の中に「PsychoPy の設定に従う」という項目がありますが、これは PsychoPy 設定ダイアログで定義されている標準の単位に従うことを意味しています。PsychoPy 設定ダイアログは、「ファイル」メニューの「設定」を選択して開きます。ダイアログの「一般」というページの **[単位]** を設定することで標準の単位を設定することができます (図 2.33)。ただし筆者の個人的な意見としては、この設定に頼ってしまうと実験を作成した PC のとは別の PC で動かそうとしたときに、両 PC で標準に設定している単位が一致していないと正常に動かなくなってしまうので、実験設定ダイアログで個々の実験に対して単位を指定するべきです。

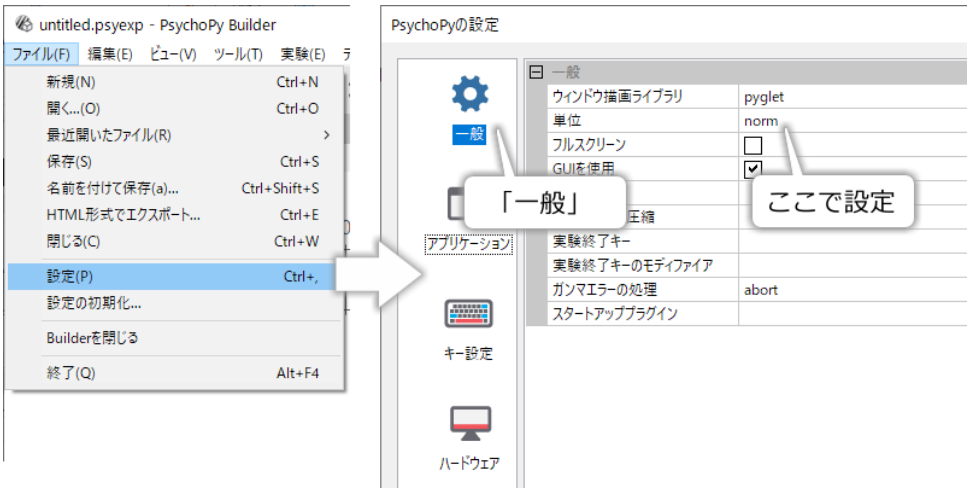


図 2.33 PsychoPy 設定ダイアログを開いて標準の単位を設定することができます。PsychoPy 設定ダイアログはメニューの「ファイル」から「設定」を選択して開きます。

[色] 視覚刺激提示画面の背景色を指定します。視覚刺激の色の指定方法と同様に、web/X11 color name や web カラー、色空間を指定した数値表現を使用することができます。

[色空間] 背景色の指定に使用する色空間を指定します。

[ブレンドモード] 刺激を重ね書きした時の挙動を指定します。標準値は「平均」で、「 2.5: 刺激の重ね順と透明度を理解しよう 」で解説した通りに描画されます。「加算」にすると色が足し合わされます。「足し合わされる」といってもわかりにくいと思いますので、 **[不透明度 $]** 0.3 の赤、緑、青の円をブレンドモード「平均」と「加算」で重ね合わせた出力を 図 2.34 に示します。

「加算」の重ね合わせのほうが光の加法混色に近いですが、重ね合わせの結果、色が PsychoPy(正確には PsychoPy が描画に使用している OpenGL というライブラリ) が表現できる範囲を超えてしまった時には描画が破綻してしまいますので、実験製作者がよく考えて刺激の色を決定する必要があります。「平均」ではそのような破綻が起きることはありません。

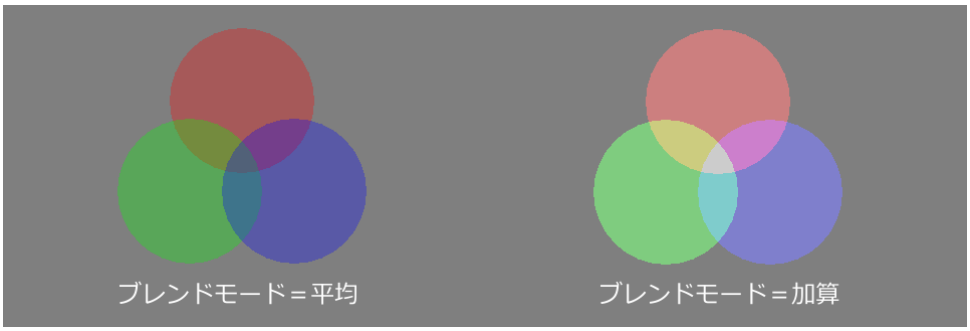


図 2.34　ブレンドモードの比較。

2.8.3 「オーディオ」タブ

オーディオの再生に関する設定を行います。詳しくは 第 10 章 を参照してください。

[オーディオライブラリ]　オーディオの再生に使用するライブラリを選択します。

[オーディオ遅延の優先度]　PsychToolbox のオーディオを使用する場合の再生遅延の設定をおこないます。0 から 4 まで 5 段階あり、数字が小さいほど幅広いハードウェアで再生できますが遅延が長くなります。

[ステレオを強制]　Sound コンポーネント (第 10 章) でステレオ再生するように設定します。とりあえずそのままにしておいて問題ないでしょう。

2.8.4 「オンライン」タブ

インターネット上でオンライン実験を行うための設定を行います。とりあえずこのページは無視してください。

[出力パス]　サーバーへアップロードする HTML 形式の実験ファイルを出力するフォルダ名を指定します。**通常、ここは空欄にしておいてください。**

[HTML 形式でエクスポート]　HTML 形式の実験ファイルを出力するタイミングを指定します。「同期時」ならサーバーと同期するとき、「保存時」なら保存するとき、「手作業で」なら手動で出力します。

[正常終了時の URL]　実験が正常に終了した時に表示するページの URL を指定します。

[中断時の URL]　実験が中断された時に表示するページの URL を指定します。

[追加リソース]　実験で使用する画像ファイルや音声ファイルのうち、Builder が自動で見つけられないものをここに追加しておくと Builder が認識してくれます (第 4 章 のテクニックで実行時に読み込むファイルを決定する場合などに便利です)。

2.8.5 「アイトラッキング」タブ

[アイトラッカーデバイス] 使用するアイトラッカーを選択します。デフォルトは None(使用しない) です。選択したアイトラッカーに応じて必要な設定項目が表示されます。ここではマウスによってアイトラッカーの動作をシミュレートする MouseGaze を選択した場合の項目を解説します。

[Move ボタン] マウスカーソルの動きをどのように視線の動きに変換するかを指定します。CONTINUOUS ならマウスカーソルの位置がそのまま視線位置となります。LEFT_BUTTON、MID_BUTTON、RIGHT_BUTTON のいずれかを選択すると、選択したボタンをクリックしたときにその位置へ視線が移動します。

[Blink ボタン] 瞬目をシミュレートするボタンを選択します。

[サッカード閾値] サッカード検出のための閾値を視線の移動量 (単位:deg) で指定します。deg 単位が有効になるためにはモニターの設定でモニターの幅と観察距離が設定されていないといけない点に注意してください。

2.8.6 「入力」タブ

[キーボードバックエンド] キーボードのキー押し検出に使うライブラリを選択します。PsychToolbox は時間精度の高さが特徴です。ioHub は PsychToolbox と比べると機能的にやや劣りますが、キーボードやマウス以外の入出力デバイスを用いるのなら ioHub の方がよいかもしれません。Pyglet は PsychToolbox、iohub には劣りますがこれらのライブラリを利用できない環境でも使うことができます。

なお、この項目を ioHub に設定している場合、「8.7: カーソルの位置を設定し、カーソルの表示 ON/OFF を制御しよう 」で紹介する方法でマウスカーソルを非表示にできない場合があります (2022.2.4/Win11 で確認)。

2.8.7 「データ」タブ

[データファイル名 $] 実験結果を記録したファイルの名前を決定する規則を Python の式で入力します。標準で入力されている式では % 演算子を使った文字列への値の埋め込みが利用されていますが、現在は format メソッド (第 12 章) を使うのが一般的です。通常の用途では変更する必要はないはずです。

[データファイルの区切り文字] データファイルで列を区切る文字を指定します。auto にするとファイル名から自動的に区切り文字を決定します。他にはコンマ、セミコロン、タブを指定できます。

[xlsx 形式のデータを保存] Excel の xlsx 形式で実験結果を記録します。詳しくは 第 3 章 を参照してください。

[CSV 形式のデータを保存 (summaries)] CSV 形式で実験結果の要約を記録します。詳しくは 第 3 章 を参照してください。

[CSV 形式のデータを保存 (trial-by-trial)] CSV 形式で実験の全試行の結果を記録します。詳しくは 第 3 章 を参照してください。

[pydat 形式のデータを保存] Python の pydat 形式で実験結果を記録します。チェックを外すことはできません。

[hdf5 形式のデータを保存] アイトラッカーなどの、iohub を介して連携する一部の装置の測定データは hdf5 形式で記録されます。ここでは詳しい解説を省略します。

[ログの保存] PsychoPy の動作状況をログファイルに記録します。

[ログレベル $] ログファイルに出力される内容を指定します。レベルには error から debug まで 6 段階あり、error が最も簡潔、debug が最も詳細です。標準設定は exp です。通常は変更する必要がありません。

以上で実験設定ダイアログの概要の説明は終了です。

- 「基本」タブの **[実験情報ダイアログを表示]** をオフにする
- 「スクリーン」タブのに **[色]** を指定して背景色を変える

の二つは各自で実際に試してみてください。

チェックリスト

- 実験設定ダイアログを開くことができる。
- 実験開始時に実験情報ダイアログを表示させるか否かを設定することができる。
- 登録済みのモニターのうちどれを実験に使用するかを実験設定ダイアログで設定できる。
- 実験をフルスクリーンモードで実行するか否かを設定することができる。
- フルスクリーンモードを使用しない時に、視覚刺激提示ウィンドウの幅と高さを指定できる。
- 視覚提示ウィンドウの背景色を指定できる。
- 「実験の設定に従う」で参照される単位を指定することができる。
- ESC キーによる実験の強制終了を有効にするか無効にするかを指定することができる。
- 実験記録のファイルを保存するフォルダ名を指定することができる。
- フルスクリーンモード時にマウスカーソルを表示するか否かを指定することができる。

2.9 モニターを設定しよう

長くなりましたが、これが本章の最後の話題です。deg や cm を単位として使用できるように、あなたが使用しているモニターを PsychoPy に登録しておきましょう。

モニターを登録するには、ツールバーの 図 2.35 に示したモニターセンターダイアログを開くボタンをクリックします。メニューの「ツール」の「モニターセンター」からも開くことができます。開いたダイアログの左上に登録されたモニターの一覧が表示されており、その横の「新規…」ボタンで新たなモニターを登録、「保存」ボタンで変更の保存、「削除」ボタンで登録の削除を行います。登録モニター一覧の下にある日付のようなリストは、選択中のモニターに対するキャリブレーションデータの一覧を示しています。簡単に言えばモニターのキャリブレーションとは、PC 上では数値によって表されている色を、モニターが正確に表現できるよ

うに調整することです。キャリブレーションには専用のセンサーが必要なので、この本では扱いません。この本で作成する実験は、ブラウザでインターネット上のニュースの写真などを閲覧して、特に違和感を感じない程度に色が表示できていれば問題なく実行できます。

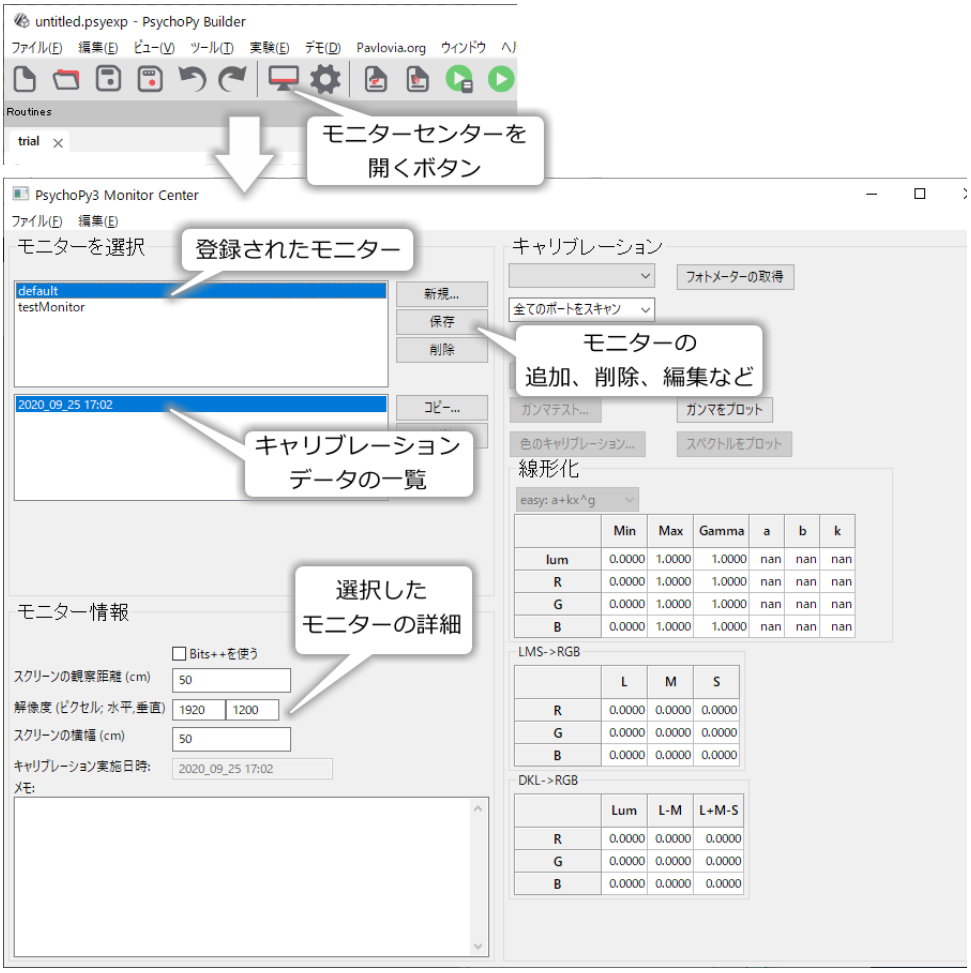


図 2.35　モニターセンターのダイアログ。モニターの登録や削除、設定の変更などができます。右半分はセンサーを用いたキャリブレーション時に使用します。

モニターの寸法や観察距離を設定するには、モニターを選択して左下の「モニター情報」と書かれた枠内に数値を入力します。ここでは、新しいモニターを登録して設定を行うことにしましょう (図 2.36)。登録モニター一覧の右の「新規…」をクリックしてください。モニターの名前を登録するダイアログが表示されるので、My Monitor と入力しておきます。OK をクリックすると、モニター一覧に My Monitor という項目が追加されているはずです。My Monitor が選択されていることを確認して、左下の「モニター情報」という枠内の「スクリーンの観察距離 (cm)」、「解像度 (ピクセル; 水平, 垂直)」、「スクリーンの横幅 (cm)」に適切な値を入力します。 図 2.36 では 図 2.11 と同じ 1920 × 1080pix、幅 51.0cm のスクリーンを持つモニターを入力しています。皆さんは各自が使用しておられるモニターの数値を入力してください。PsychoPy では画素の縦横の長さは同一として計算しているので、幅だけを入力すれば高さは解像度と幅から自動的に計算されます。観察距離は 図 2.36 の例では 57.3cm としておきました。終了したら登録モニター一覧の「保存」をクリックして保存して、モニターセンターのダイアログを閉じてください。保存せずに閉じようとすると変更を保存するか尋ねられるので、保存しておきましょう。

モニターの登録が終わったら、実験設定ダイアログを開いて「スクリーン」のページの「モニター」に My Monitor と入力しましょう。そして、ルーチンペインに Polygon コンポーネントを配置し、 **[サイズ [w, h] $]**

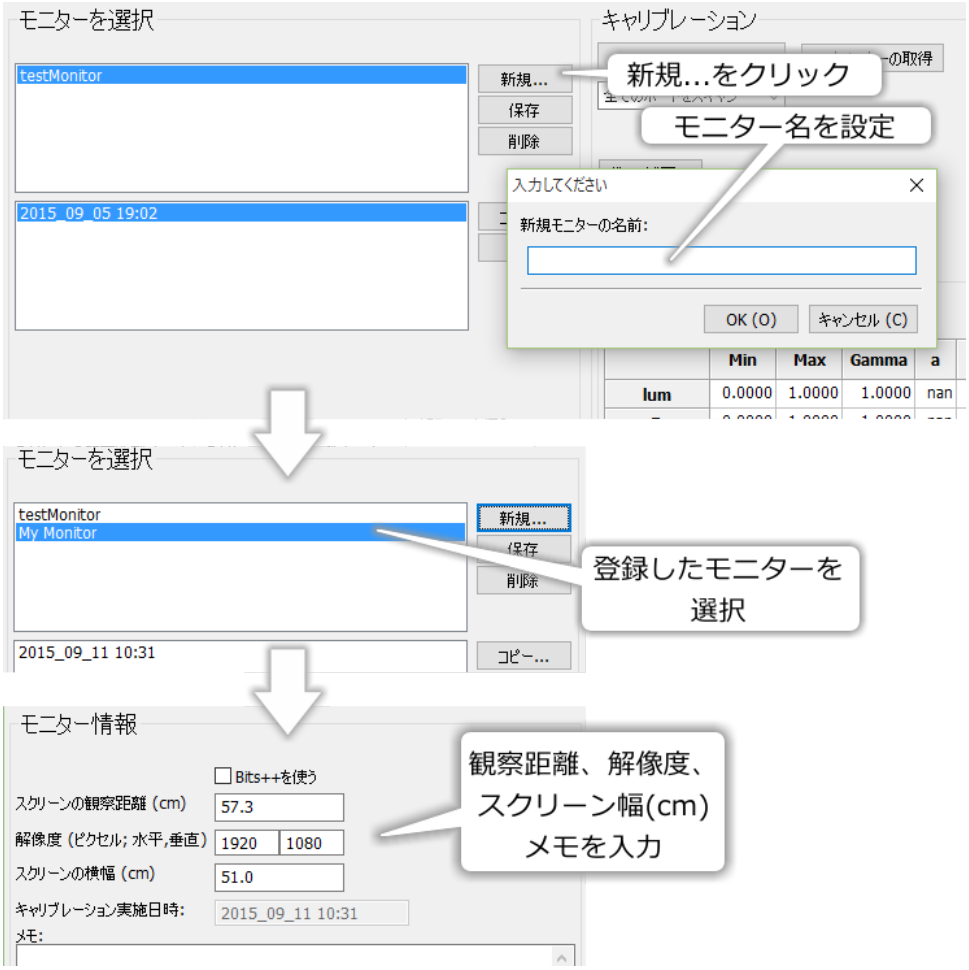


図 2.36　モニターの登録手順。

を [5, 5]、 **[空間の単位]** を cm にして実行してみましょう。正しく設定されていれば、一辺の長さ 5cm の正方形のスクリーン上に表示されます。ぜひ定規で測って確認してください。刺激がすぐ消えてしまって測れないという方は、刺激の表示時間を長くしましょう。

確認ができたら、Polygon コンポーネントのプロパティ設定ダイアログを開いて **[空間の単位]** を deg にして実行しましょう。観察距離 57.3cm (≒ 180/π) の時には、1deg がほぼ 1cm となりますので、画面上ではやはり一辺約 5cm の正方形が表示されているはずです。それを確認したらモニターセンターへ戻って、My Monitor の観察距離を 30cm に変更してから実験を実行してみましょう。そうするとスクリーンに表示される正方形の一辺は 5cm より短くなったはずです。観察距離が短くなると刺激は網膜に大きく映るので、視角 5deg にするためには刺激を縮小しなければいけません。この縮小作業を PsychoPy が自動的に行ってくれたのです。さらに観察距離 30cm のままで Polygon コンポーネントを編集して **[空間の単位]** を cm に戻してみましょう。単位が cm の場合には、観察距離に関わらず常に一辺 5cm の正方形が表示されるはずです。

これで準備は完了です。次章ではいよいよ最初の実験を作成してみましょう。

チェックリスト

- モニターセンターを開くことができる。
- モニターセンターに新しいモニターを登録することができる。

- モニターの観察距離、解像度、スクリーン幅を登録することができる。

2.10 この章のトピックス

2.10.1 スクリーン左下の座標についての厳密な議論 (上級)

本文では、(1920, 1080) の解像度を持つモニターで右上の座標は (960, 540)、左下の座標は (-960, -540) と述べました。しかし、じっくり考えてみれば変です。水平方向の解像度が 10 しかない小さな小さなモニターを考えてみましょう。10 を半分にすると 5 ですから左端の座標を-5 として、-4、-3…と数えていくと、10 ピクセル目は 5 ではなく 4 です。したがって、このモニターの左端の X 座標が-5 なら右端の X 座標は 4 でなければいけません。同じように、水平解像度が 1920 のモニターで左端の X 座標を-960 にしたのなら、右端は 960 ではなく 959 でなければならないのです。

コンピュータ上で符号付きの整数を扱う場合、一般的には負の数の方が絶対値が 1 大きくなるように範囲を定めます。しかし、筆者の実行環境 (Windows10, Python x64, PsychoPy 3.0.5) で画面上に 1 × 1 ピクセルの刺激を描いて確認したところ、右上が (960, 540)、左下が (-959, -539) となり、正の方向に絶対値が 1 大きくなっていました。ただし、この結果はすべての実行環境、すべてのバージョンで保証されるとは限りませんので、参考程度にとどめてください。一般的な実験ではこの 1 ピクセルの差が問題となることはないと思われますので、本文ではこれ以降もスクリーンの中心を (0, 0)、解像度を 2 で割った値をスクリーン端の座標として記します。

2.10.2 PsychoPy における視角の計算について

視角による視覚刺激の位置や大きさの表現は、視知覚の実験などでは欠かせないものです。本文中で述べた通り、PsychoPy では視角による表現のために deg という単位が用意されていますが、その実装は近似計算です。PsychoPy Coder を使うと PsychoPy に刺激の位置などの単位を deg、pix、cm の間で相互に変換することができるのですが、それを利用して観察距離 55cm のモニターで 1.0deg を cm に変換すると 0.96cm という結果が得られます。これは 55cm × tan(π/180) の計算結果と等しいです。同様に PsychoPy に 10.0deg を cm に変換させると、9.60cm という結果が得られます。一方、55cm × tan(10 π/180) の計算結果は 9.70cm となり、9.60cm と一致しません。

これは、PsychoPy が deg を cm に変換する時に毎回三角関数の計算を行わずに、スクリーン中央から 1.0deg の位置を cm に変換した時の値を C として、X deg を cm に変換する時には C × X で近似計算しているために起きる現象です。X が小さいときはとてもよい近似値が得られるのですが、画面の中央から遠ざかるほど誤差が大きくなります (図 2.37)。先の例では画面中央から 10deg 離れた位置で 0.1cm しか異なりませんので、多くの実験では問題になることはないと思われます。

しかし、極めて正確な刺激の描写が必要な実験や、最近増えてきた大型モニターを両端いっぱいまで使って刺激を描画するような実験では、この誤差が問題となる可能性があります。この問題に対応するために、PsychoPy 1.80 では deg より厳密な計算を行う degFlat、degFlatPos という単位が追加されました。degFlat では、図形の頂点座標を視角の定義通りに計算します。先ほどの観察距離 55cm で 10.0deg の例でも正しく 9.70cm という換算値が得られます。

defFlatPos は deg と degFlat の中間のような計算で、図形の頂点座標は deg と同様に計算し、刺激を配置する

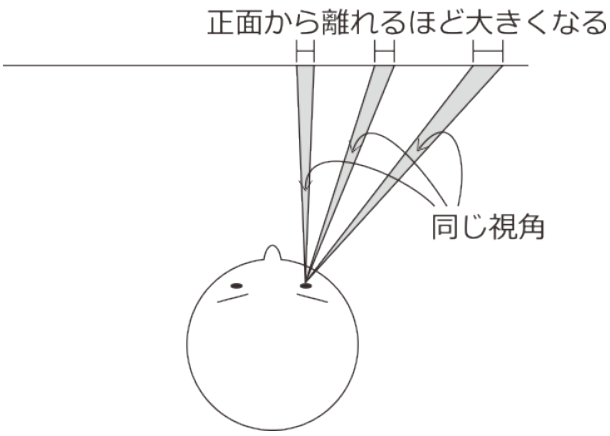


図 2.37　平面モニターを使う場合、視角を一定のまま刺激を端に移動させるとモニター画面上では刺激を大きくする必要があります。

ときの位置のみを正確に計算します。degFlat と degFlatPos の違いを図で示したのが 図 2.38 です。この例では deg, degFlat, degFlatPos を用いて傾いた正三角形を (0.0, 0.0) の位置から (1.0, 0.5) ずつ動かしながら (24.0, 12.0) の位置まで描画しています。白が deg、赤が degFlat、青が degFlatPos で描いたものです。deg で描いた白い三角形はすべて同じ形で等間隔に並んでいます。一方、degFlat で描いた赤い三角形は、右上に向かうにつれて間隔が広がり、三角形は大きくなり形が歪んでいます。三角形の頂点座標をすべて視角の定義に基づいて計算しているのでこのような結果となります。それに対して青で描かれた degFlatPos では、三角形の間隔こそ degFlat と同様に右上に向かうにつれて広がっていますが、三角形の大きさや形状は一定のままです。これが「頂点座標は deg と同様に計算し、配置するときの位置のみを正確に計算する」という意味です。

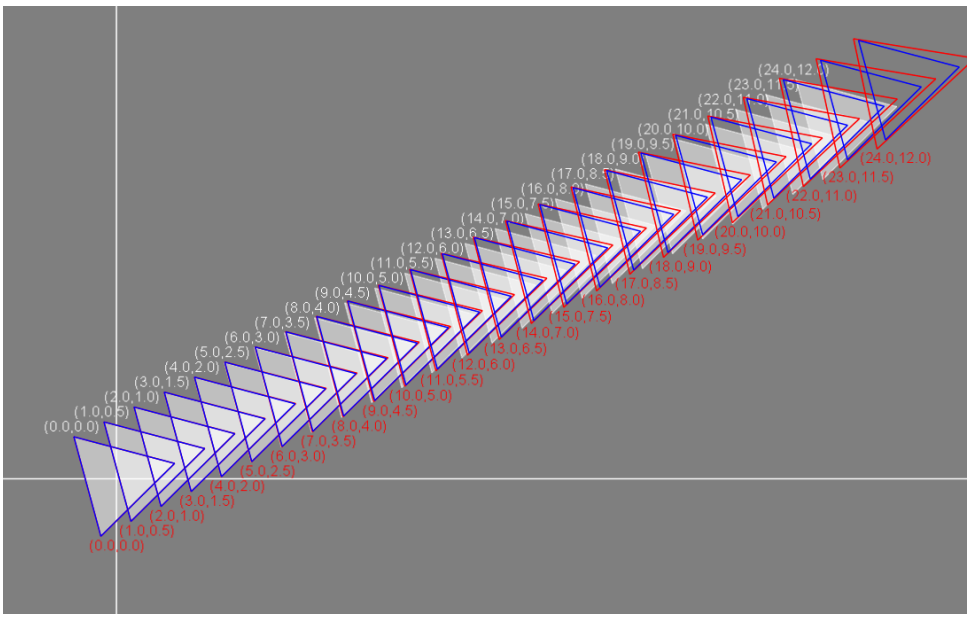


図 2.38　deg(白)、degFlat(赤)、degFlatPos(青) の違い。

2.10.3 Builder の設定ダイアログで用いられるフォント

PsychoPy Builder の標準のフォント設定では、実行環境によっては **[位置 [x, y] $]** のような $ が付くプロパティ値の小数点とカンマが非常に見分けにくいです。例えば Windows10 では 図 2.39 のようにカンマが非常に小さく表示されてしまい、小数点と見間違えることがあります。PsychoPy のウィンドウ上部のメニュー「ファイル」の「設定」を選び、表示されるダイアログの「Coder」タブをクリックして、「コード用フォント」や「コード用フォントサイズ」を変更すると見やすくなります。見分けにくくて苦労している人はぜひ試してください。

なお、この設定ダイアログはツールバーの 図 2.33 に示すボタンをクリックしても表示することができます。プロパティ名についている $ 記号の意味については 第 3 章 で解説します。

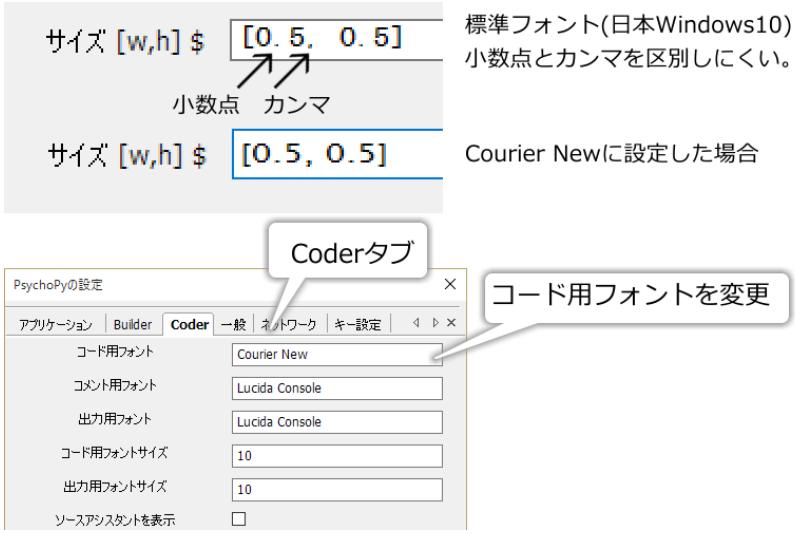


図 2.39　コード用フォントの変更。

2.10.4 Mac 上で日本語の文字が欠けてしまう場合の対策

MacOS では日本語フォントの仕様がしばしば変更されるようで、Text コンポーネントでの日本語の表示が乱れてしまう現象が度々生じています。ほとんどの場合、「書式」タブの **[フォント]** に適切なフォント名を指定すると解決します。残念ながら「こう書いたら必ず解決する」という表記方法はないのですが、「游ゴシック体」のような日本語でのフォント名か、「Hiragino Sans」のような英語でのフォント名を指定すると解決するケースが多いです。

2.10.5 16 進数と色表現

私たちが普段使い慣れている数は 10 進数でしょう。10 進数では 0、1、2、3、…と値が増加して、9 の次の整数は桁がひとつ上がり 1 の位は 0 に戻って「10（イチゼロ）」と表記されます。同様に考えると、例えば 8 進数とは 7 の次の整数になるときに桁がひとつ上がって 1 の位が 0 に戻る表記だということになります。0 から 10 までの整数を 8 進数で書くと 0、1、2、3、4、5、6、7、10（イチゼロ）、11（イチイチ）、12（イチニ）…となります。最後の 10（イチゼロ）、11（イチイチ）、12（イチニ）は 10 進数の 8、9、10 に対応しています。

頭がこんがらがってきますが、同様に 16 進数の表記を考えてみると困ったことが起きます。0、1、2、…、9

と来て、その次の整数を示す文字がアラビア数字には無いのです。仕方がないので、多くのプログラミング言語では 9 の次の整数を示す文字としてアルファベットの A、さらにその次の整数を表す整数として B、という具合にアルファベットを割り当てます。この調子でアルファベットを使っていくと、10 進数の 15 が F となり、10 進数の 16 が 10 となって 16 進数表記が完成します。Python では、「10」と書かれた時に 10 進数の 10 を表しているのか 16 進数の 16 を指しているのかを区別するために、16 進数表記の数には先頭に「0x」を付けることになっています。つまり、ただ「10」と書いてあれば 10 進数の 10 であり、「0x10」と書いてあれば 16 進数の 16 というわけです。

さて、なぜ web カラーによる色の表現に 16 進数が用いられるかということですが、web カラーでは赤、緑、青 (RGB: Red, Green, Blue) の光の強度をそれぞれ 0 から 255 の 256 段階で指定するからです。例えば RGB をそれぞれ 100 段階 (0 から 99) で変化させられる機械があるとすると、6 桁の 10 進数を用いて左から 1 桁目と 2 桁目を R、3 桁目と 4 桁目を G、5 桁目と 6 桁目に B を割り当てれば、この機械で使用できる色を 6 桁の 10 進数の数値としてわかりやすく表現することができます。この割り当て方法を 6 桁の 16 進数に適用すると、左側から 2 桁ずつ 16 進数の値を R、G、B に割り当てる形になります。2 桁の 16 進数というと 0x00 から 0xFF ですが、この範囲を 10 進数に換算すると 0 から 255 になりますので、RGB の各成分が 256 段階となる機械にとって 16 進数による表現は非常に相性がよいはずです。

具体的な数値で考えてみましょう。赤色は左の 2 桁の数値が 10 進数の 255 に対応する 16 進数 0xFF で、残りの桁は 0 となります。すなわち、0xFF0000 です。同様に、緑色は 0x00FF00、青色は 0x0000FF です。黄色の場合は、赤と緑の混色ですから赤に対応する桁と緑に対応する桁が 0xFF で青に対応する桁が 0 である値、すなわち 0xFFFF00 です。これらの色に対応する値を 10 進数で表記すると、赤が 16711680、緑は 65280、青は 255、黄色は 16776960 です。RGB の各成分を 256 段階で表現する機械においては、10 進数表記より 16 進数表記の方がはるかに直感的に RGB の強度が把握しやすいことがお分かりいただけるのではないかと思います。

なお、Python では数値が 16 進数表記であることを示すために 0x を先頭に付けますが、web ページを記述するのに用いられる HTML という言語では#FF0088 のように先頭に「#」を付けて 16 進数を示します。PsychoPy では「0x」も「#」も両方使用することができます。HTML では RGB のそれぞれに 2 桁ではなく 1 桁の 16 進数を割り当てて、#7FA のように 3 桁の 16 進数で色を表現することもできます。PsychoPy はこの表現もサポートしています。3 桁の 16 進数が与えられた場合、PsychoPy は内部で例えば#7FA を#70F0A0 にするという具合に 0 を挿入することで 6 桁の表現に変換します。

2.10.6 時刻指定における frame について

ご存知の方も多いと思いますが、PC のモニターは 1 秒間に数十回も静止画をスクリーンに表示することによって滑らかな動きを表現しています。モニターがスクリーンを 1 秒間に書き換える回数をリフレッシュレートと呼びます。リフレッシュレートの単位は Hz です。一般的な PC 用モニターのリフレッシュレートは 60Hz 前後です。60Hz のモニターであれば、1 秒間に 60 回の書き換えを行います。高速に書き換えられる個々の静止画をフレームと呼びますが、このフレームという用語を用いると 60Hz のモニターは 1 秒間に 60 フレームを表示するという事もできます。フレームの書き換え間隔は一定ですので、1 フレームの表示時間は 1 秒÷ 60 回=0.0167 秒、すなわち 16.7 ミリ秒です。

16.7 ミリ秒に 1 度しか書き換えが行われないということは、例えば Builder で刺激の表示時間を設定する際に **[終了]** で「実行時間 (秒)」を選択して 0.04 (=40 ミリ秒) を入力しても、実際に刺激が表示されている時間は 40 ミリ秒にはならないということです。Builder と 60Hz のモニターを用いて実際に動作確認してみると、「実

行時間 (秒)」が 0.04 の時には 33.3 ミリ秒しか刺激は提示されていません (PsychoPy 1.79.01 で確認)。33.3 ミリ秒は 2 フレームに相当します。刺激提示開始時刻や終了時刻、提示時間は 1 ミリ秒 (もしくはそれ以下の) 単位で指定できますが、実質的にはフレーム単位でしか制御できていないのです (図 2.40)。言い換えると、刺激の提示時間としてフレームの表示時間の倍数を指定した時以外は、設定した時間と実際に表示されている時間の間には必ずズレが生じているのです。

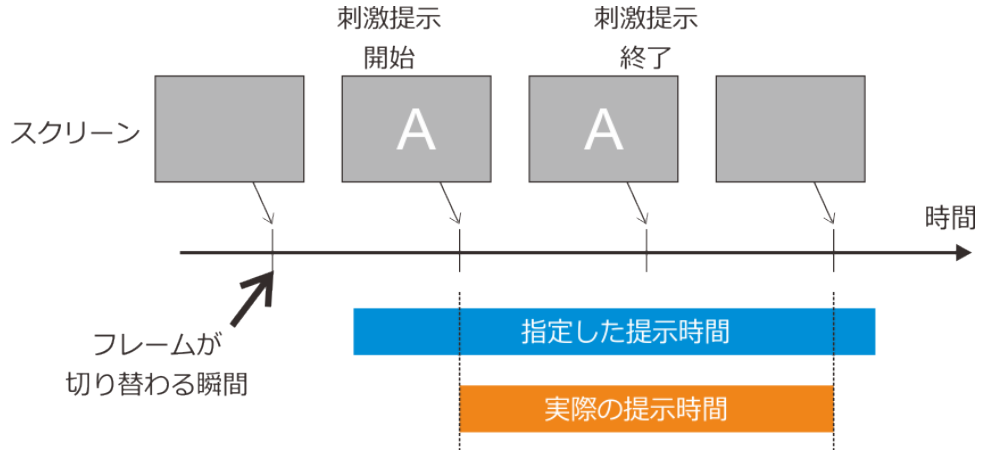


図 2.40　提示時間がフレーム表示時間の倍数になっていなければ指定した時間と実際の提示時間の間にずれが生じます。

どうせフレーム表示時間の倍数でしか正確に刺激提示時間を指定できないのでしたら、いっそのこと秒単位ではなくフレーム数で刺激提示時間を指定した方がわかりやすいという考え方もあるでしょう。フレーム数による指定を可能にするのがプロパティ設定ダイアログの **[終了]** で選択できる「実行時間 (フレーム数)」という選択肢です。「実行時間 (フレーム数)」を選択すると、指定された数のフレームを表示する間刺激を提示します。当然、正の整数を指定しなければ意味がありません。同様に **[開始]** 、 **[終了]** でともに選択できる「フレーム数」を用いるとルーチンが開始されてから何フレーム目に刺激提示を開始、終了するかを指定することができます。100 ミリ秒未満の短時間の刺激を提示する場合や、特に正確な提示時間の制御が必要な実験をする場合はフレームによる指定が有効です。

なお、フレームで刺激提示時間を指定すると、ルーチンペインの青い棒が表示されなくなってしまいます。Builder は実験に使用されるモニターのリフレッシュレートを知らないので、何秒から何秒まで刺激が提示されているかを計算することができないのです。このままではルーチンペインを見たときに刺激がどのような順番に提示されていくのか非常にわかりづらいので、Builder には「この時刻からこの時刻まで提示される」という目安を表示させる機能があります。Polygon コンポーネントや Text コンポーネントを配置してプロパティ設定ダイアログを確認してください。灰色の文字で **[開始]** の下に **[予想開始時刻 (s)]** 、 **[終了]** の下に **[予想実行時間 (s)]** と書かれた項目があります。ここに開始時刻と提示時間の見積もりをそれぞれ入力すると、見積もりに従って青い棒が表示されます (図 2.41)。見積もり値は、各自で使用しているモニターのリフレッシュレートと指定したフレーム数から計算する必要があります。ちなみに「時刻 (秒)」や「実行時間 (秒)」を選択している時にもこれらの見積もりを入力することができますが、混乱を招くだけで意味はありません。

蛇足ですが、リフレッシュレートと非常によく似た用語でフレームレートというものがあります。フレームレートの単位は frames per second の略で FPS です。「1 秒あたりのフレーム数」という意味ですから、リフレッシュレートと同じ意味のように思えます。しかし、フレームレートという用語は通常「PC がモニターに対して 1 秒間に表示するように要求したフレーム数」を指します。フレームレートが 100FPS に達しても、使用しているモニターのリフレッシュレートが 60Hz であれば 1 秒間に実際に表示されるフレーム数は 60 枚です。間違えやすいのでご注意ください。

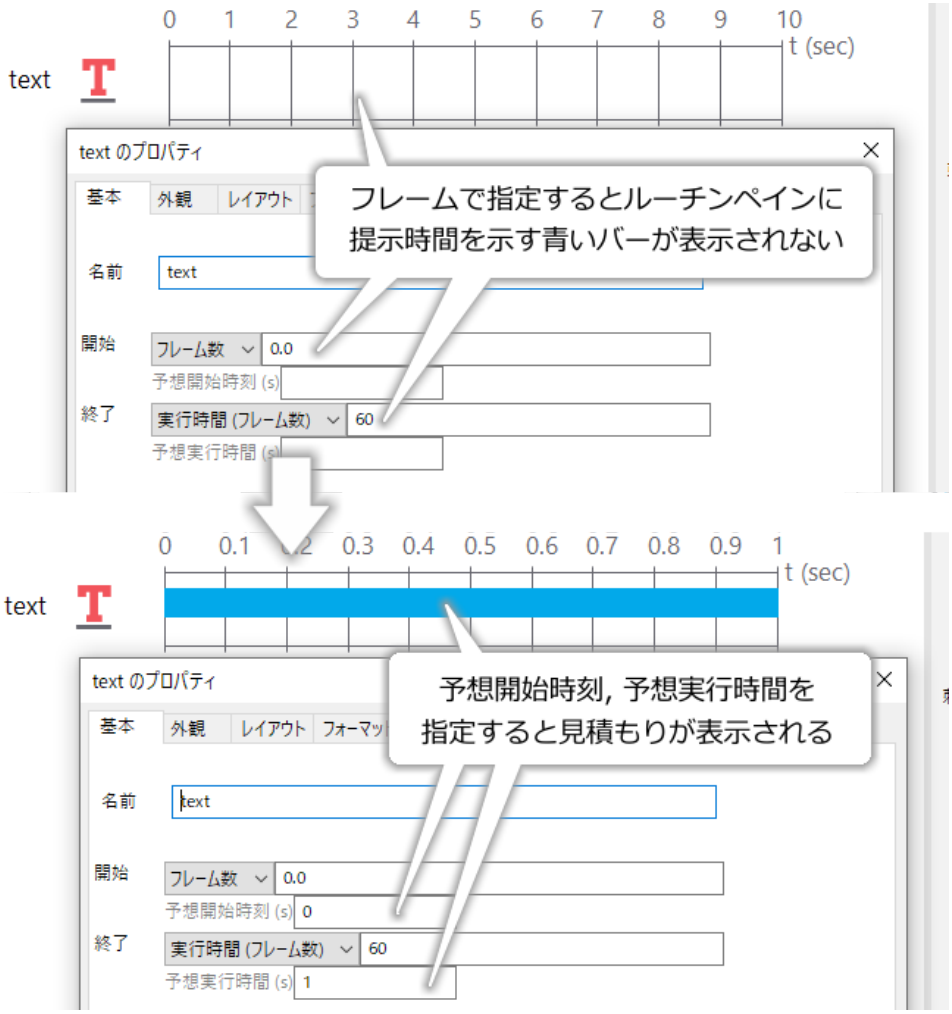


図 2.41　フレームで刺激の提示時間を指定するとルーチンペイン上で提示時間を示す青い棒が表示されません。 **[予想開始時刻 (s)]** 、 **[予想実行時間 (s)]** に見積もり値を入力することでルーチンペイン上に提示時間を表示することができます。

第 3 章

最初の実験を作ってみよう—サイモン効果

3.1 実験の手続きを決めよう

この章では、最初の実験としてサイモン効果の実験を作ってみましょう。サイモン効果について詳しい解説は認知心理学の教科書などを参考にしていただくとして、ここでは実験の手続きを説明します。

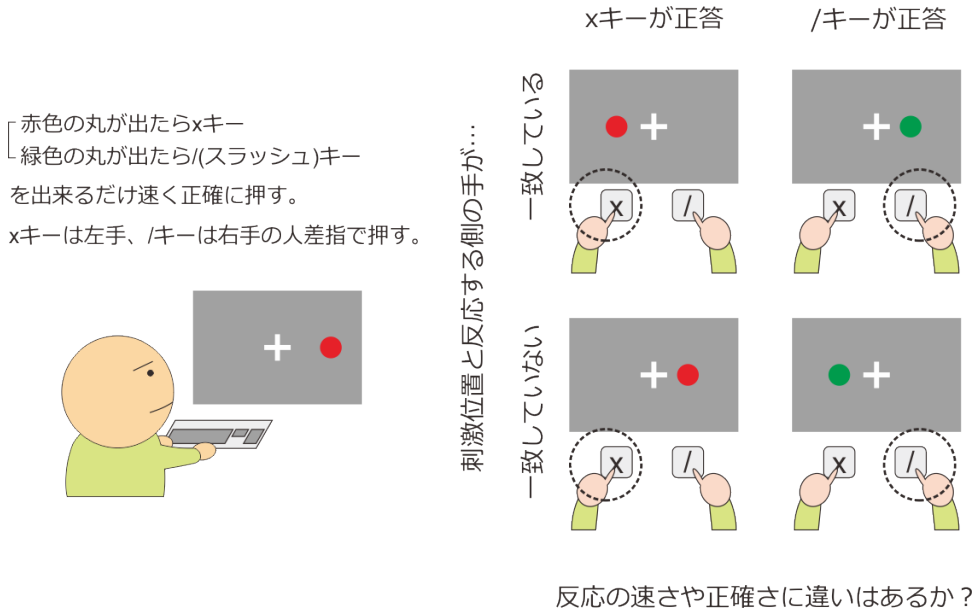


図 3.1　サイモン効果の実験概要。実験参加者は刺激の色に対応するキーをできるだけ速く正確に押すように求められます。スクリーン上の刺激の位置と反応するキーの位置関係が反応の速さや正確さに及ぼす影響を検討します。

図 3.1 をご覧ください。実験参加者に与えられた課題は、PC のスクリーン上に提示された刺激の色に応じて、PC のキーボードのキーを指定された指で押すことです。参加者はスクリーン中央の十字 (固視点) に視線を向けて、刺激が出現するのを待ちます。刺激は赤色または緑色の円で、固視点の右側または左側のどちらかに出現します。参加者は刺激の色が赤色であれば左手の人差し指で x のキー、緑色であれば右手人差し指で / (スラッシュ) のキーをできるだけ速く、間違わないように押さないといけません。左側に赤色の刺激が出現した場合と右側に緑色の刺激が出現した場合には刺激と反応する指が同じ側になりますが、それ以外の場合は刺激が出現した側と反対側の指で反応しないといけません。刺激が提示される位置と反応する指の関係が反応の速さや正確さにどのように影響するかを確認しようというわけです。

さて、実際に実験を作成するためには、図 3.1 の内容よりもっと詳細に手続きを決定する必要があります。図 3.2 は刺激提示スクリーンの時間的な変化を図示したものですが、このように書いてみると 図 3.1 の内容では x1 から x5 の値が決まっていないことがわかります。何かの文献に基づいて実験を作成しているのであれば文献に書かれている手続きを熟読して値を決めることになるでしょう。この章では、あまり内容を高度にし過ぎないためにあらかじめ以下のように値を決めることにします。

- x1 固視点の大きさは縦横 0.05(位置と大きさの単位はすべて height)。
- x2 試行が始まってから刺激が出現するまでの時間は 1.0 秒。
- x3 固視点の中心から刺激の中心までの距離は 0.7。
- x4 刺激の直径は 0.1。
- x5 刺激の位置 (2 か所) と色 (2 種類) の組み合わせで合計 4 種類の刺激に対して 25 試行ずつ、合計 100 試行。

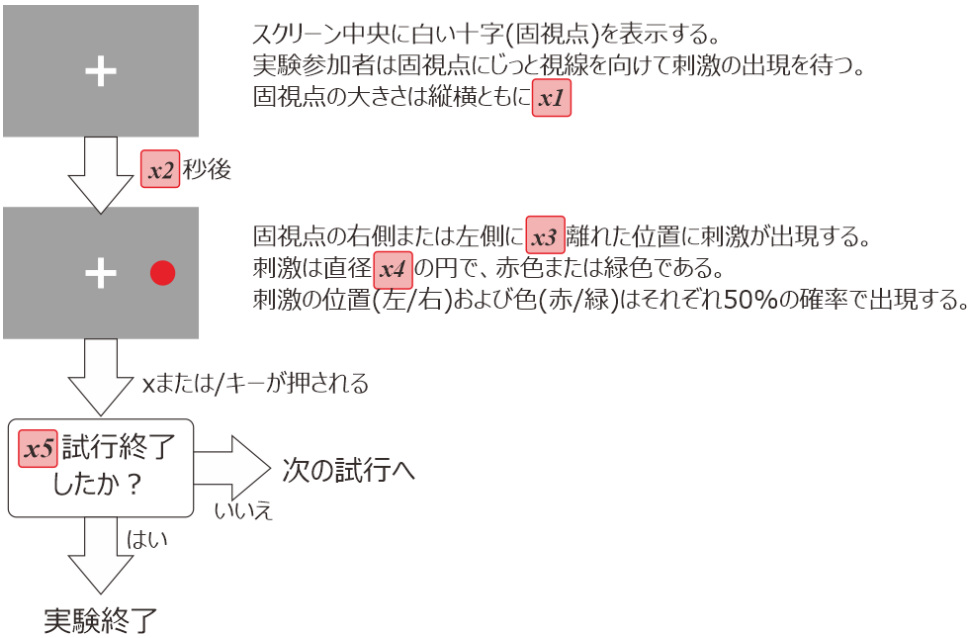


図 3.2　サイモン効果実験の手続き。実際に実験を作成するためには x1 から x5 の値を決定する必要があります。

刺激はスクリーンの左右端付近に出したいのですが、使用するスクリーンの縦横比によって左右端の X 座標が異なります。ここでは縦横比 3:2 のスクリーンでも刺激がおさまることを考えて固視店の中心から刺激の中心までの距離 (x3) を 0.7、刺激の直径 (x4) を 0.1 としました。この辺りの数値は各自の実行環境に応じて変更してください。

実験を作成する前にもうひとつ確認して置きたいのは、この実験において「試行毎に変化する値」は何かという点です。図 3.1 から明らかなように、刺激の色 (赤 or 緑) と刺激の位置 (右 or 左) は試行毎に変化します。刺激の色が変化するという事は、それに対応して正解となるキー (x or /) も変化することに注意しましょう。

さて、それではいよいよ実験の作成を始めましょう。なお、今回作成する実験の psyexp ファイル名は、第 3 章の実験ということで exp03.psyexp とすることにします。

3.2 視覚刺激を配置しよう

第 2 章 では視覚刺激の大きさや位置、色の指定などについて学んだのですから、まずは刺激の作成から始めましょう。まず、刺激の円は Polygon コンポーネントで **[形状]** を円すれば描画できます。刺激の位置と色は試行毎に変化しますが、まだその方法は解説していませんのでひとまず (0.7,0) の位置に赤色で表示することにしましょう。Polygon コンポーネントをひとつルーチンに配置し、以下のように設定してください。

- **「基本」タブ**
  - **[名前]** に stimulus と入力する。
  - **[形状]** を円にする。
- **「外観」タブ**
  - **[塗りつぶしの色]** に red、**[枠線の色]** に None と入力する。
- **「レイアウト」タブ**
  - **[位置 [x, y] $]** に (0.7, 0) と入力する。
  - **[サイズ [w, h] $]** に (0.1, 0.1) と入力する。
- PsychoPy の設定で height 以外の単位を標準に設定している場合は、実験設定ダイアログを開いて **[単位]** を height にしておく。

続いて固視点ですが、幸いなことに Polygon コンポーネントは十字を描くこともできます。新たに Polygon コンポーネントを 1 つルーチンに配置し、以下のように設定してください。

- **「基本」タブ**
  - **[名前]** に cross と入力する。
  - **[形状]** を十字にする。
- **「レイアウト」タブ**
  - **[サイズ [w, h] $]** に (0.05, 0.05) と入力する。

続いて提示時間の設定について考えましょう。試行開始時には固視点のみがスクリーン上に提示されていて、試行開始 1.0 秒後に刺激が提示されるのですから、固視点 (cross) の **[開始]** は「時刻 (秒)」で 0.0、刺激 (stimulus) の **[開始]** は「時刻 (秒)」で 1.0 にすればよいでしょう。固視点、刺激ともにキーが押されるまでずっと提示されるのですから、 **[終了]** はどちらも空白にしておきましょう。

さて、ここまでの作業を終えると、ルーチンペインには 図 3.3 上のように 2 つの Polygon コンポーネントが並んでいるはずです。コンポーネントの順番はこの通りでなくても構いません。そして実験を実行すると 図 3.3 下のようにスクリーンに刺激が提示されることを確認してください。ここまでの記述で分からないことがあったら 第 2 章 を復習してください。ここまで確認できればいよいよキーボードを用いて実験参加者の反応を検出する方法を学びましょう。

チェックリスト

- Polygon コンポーネントを用いて円をスクリーン上に提示することができる。

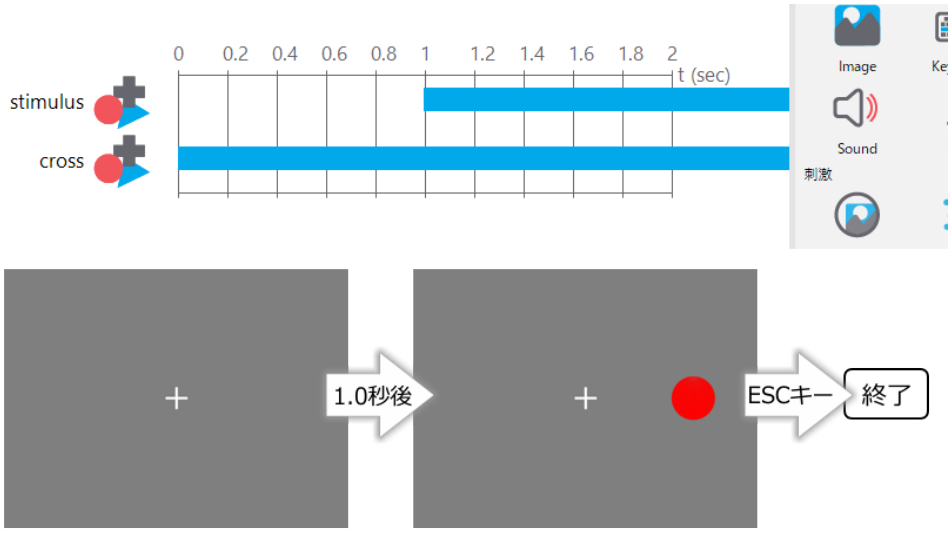


図 3.3　視覚刺激用の Polygon コンポーネントの配置と実行確認。

- Polygon コンポーネントを用いて十字をスクリーン上に提示することができる。

3.3 キーボードで反応を検出しよう

Builder からキーボードを利用するには、コンポーネントペインの Response というカテゴリに含まれる Keyboard コンポーネントをルーチンペインに配置します。Polygon や Text コンポーネントを配置する時と同じように、コンポーネントペインの Keyboard コンポーネントのアイコン (図 3.4) をクリックしましょう。すると Keyboard コンポーネントのプロパティ設定ダイアログが開きます。

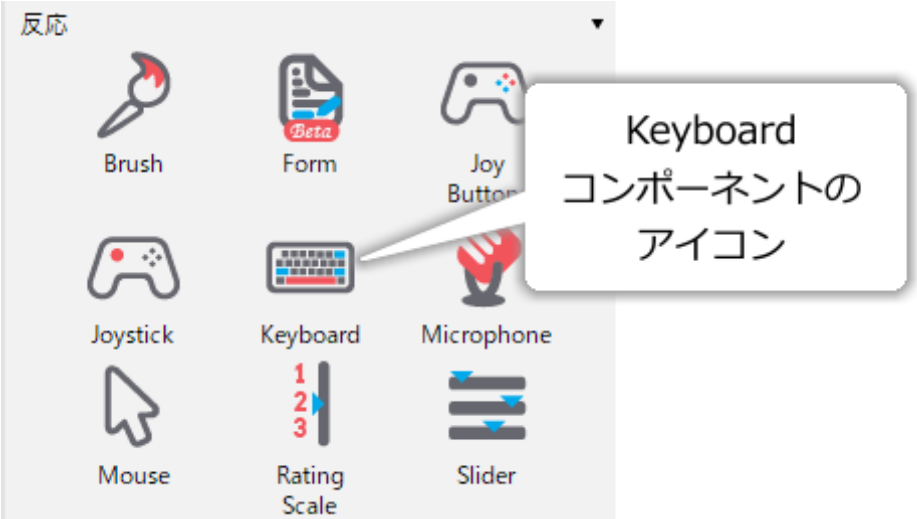


図 3.4　Keyboard コンポーネントのアイコン

「基本」タブの **[名前]** 、 **[開始]** や **[終了]** は、 第 2 章 で扱ったコンポーネントと同様に、名前をつけたりコンポーネントが有効となる時間を指定したりするために使用します。視覚刺激の場合、「有効となる時間」はスクリーン上に表示されている時間に対応しましたが、Keyboard コンポーネントの場合はキー押し反応を受け付ける時間に対応します。例えば今回の実験の場合、刺激がスクリーン上に出現する前に押されたキーは無視

するべきですから、試行開始 1.0 秒経過するまで Keyboard コンポーネントは無効にするべきです。また、今回は参加者が反応するまで待ち続けますが、実験によっては 3 秒以内に反応できなかったら強制的に次の試行に移るといったこともあるでしょう。 **[開始]** と **[終了]** を適切に設定することによってこういった実験を実現することができます。

[終了] の下に **[Routine を終了]** という項目がありますが、ここにチェックが入っていれば、キーが押されると他のコンポーネントがまだ終了していなくても強制的にルーチンを終了します。今回はこの機能を利用して「参加者がキーを押すまで刺激を提示し続ける」という動作を実現します。仮に「30 秒間スクリーンを注視し、刺激が無作為な時間間隔で複数回スクリーン上に出現する度にできるだけ速くキーを押す」といった実験の場合はキーが押されてもルーチンを中断してはいけません。このような実験の場合は **[Routine を終了]** のチェックを外しておきます。

[検出するキー $] には、Builder がキー押しを監視するキーの名前を列挙します。初期状態では'y', 'n', 'left', 'right', 'space' と入力されています。これは順番にアルファベットの Y キー、N キー、カーソルキーの「←」キー、「→」キー、スペースキーを表すキー名です。必ず半角文字で入力すること、キー名の前後にシングルクォーテーションまたはダブルクォーテーション記号でキー名を囲むこと、キー名の間はカンマで区切ることが重要なポイントです。ここに書かれたキー以外が押された場合は、Keyboard コンポーネントが有効な時間帯であっても無視されます。参加者が反応に使用するキーのみを書くことによって、参加者が誤って無効なキーを押してしまってルーチンが終了してしまうことを防止できます。心理学実験ではあまり無いことだと思いますが、すべてのキーを有効する必要がある場合は **[検出するキー $]** を空白にしてください。

問題は自分が使いたいキーのキー名を何と書けばよいのかですが、一般的な日本語キーボードであれば 表 3.1 に示すキー名が使えるはずです。一般的な日本語キーボードでは Shift キーを押しながら 1 キーを押すと！記号を入力できますが、PsychoPy は！というキーが押されたとは判断せず、あくまで Shift と 1 キーが押されたと判断しますので注意してください。

もうひとつ注意しないといけない点は、キーボードによっては特殊キーの左右を区別しないなどのクセがある点です。例えば 表 3.1 では左シフトキーと右シフトキーにそれぞれ lshift、rshift という名前が割り当てられていますが、使用しているキーボードが左右どちらのキーを押しても lshift というキー名を返すように作られている場合があります。自分が使用する予定のキーボードがどのようなキー名を返すのか確認したい、Windows 用キーボードの Windows キーや Mac 用キーボードの Command キーがどのような名前を返すのか確認したいといった場合は、キー名を表示させる Builder の「実験」を作成すると便利です。作成にはずっと先の章で取り上げる予定のテクニックが必要になりますので、ここでは扱わず「 3.11.1: 自分のキーボードで使えるキー名を確かめる 」に記しておきました。

表 3.1: 一般的な日本語キーボードで使用できるキー名

キー名	対応するキー	キー名	対応するキー
f1	F1	**minus**	-
f2	F2	**asciicircum**	^
f3	F3	**backslash**	\
f4	F4	**bracketleft**	[
f5	F5	**bracketright**	]
f6	F6	**semicolon**	;
f7	F7	**colon**	:

次のページに続く

表 3.1 – 前のページからの続き

キー名	対応するキー	キー名	対応するキー
f8	F8	**comma**	,
f9	F9	**period**	.
f10	F10	**slash**	/
f11	F11	**at**	@
f12	F12	**return**	Enter キー
num_0	0 (テンキー)	**1**	1
num_1	1 (テンキー)	**2**	2
num_2	2 (テンキー)	**3**	3
num_3	3 (テンキー)	**4**	4
num_4	4 (テンキー)	**5**	5
num_5	5 (テンキー)	**6**	6
num_6	6 (テンキー)	**7**	7
num_7	7 (テンキー)	**8**	8
num_8	8 (テンキー)	**9**	9
num_9	9 (テンキー)	**0**	0
num_add	+ (テンキー)	**a**	A
num_subtract	- (テンキー)	**b**	B
num_multiply	* (テンキー)	**c**	C
num_divide	/ (テンキー)	**d**	D
num_decimal	. (テンキー)	**e**	E
left	←	**f**	F
down	↓	**g**	G
right	→	**h**	H
up	↑	**i**	I
escape	ESC キー	**j**	J
pageup	PageUp キー	**k**	K
pagedown	PageDown キー	**l**	L
end	End キー	**m**	M
home	Home キー	**n**	N
delete	Del キー	**o**	O
insert	Ins キー	**p**	P
backspace	バックスペースキー	**q**	Q
tab	タブキー	**r**	R
lshift	左シフトキー	**s**	S
rshift	右シフトキー	**t**	T
lctrl	左 Ctrl キー	**u**	U
rctrl	右 Ctrl キー	**v**	V
lalt	左 Alt キー	**w**	W
ralt	右 Alt キー	**x**	X
		y	Y

次のページに続く

表 3.1 – 前のページからの続き

キー名	対応するキー	キー名	対応するキー
		z	Z

説明も長くなってきましたので、その他の項目は後で解説することにして、まずは動作を確認してみましょう。Keyboard コンポーネントのプロパティ設定ダイアログで以下のように設定してください。

- **「基本」タブ**
  - **[開始]** を「時刻 (秒)」にして 1.0 と入力する。
  - **[終了]** を空白にする。
  - **[Routine を終了]** にチェックが入っていることを確認する。
  - **[検出するキー $]** に 'x', 'slash' と入力する。
- その他の項目は初期設定のままにする。

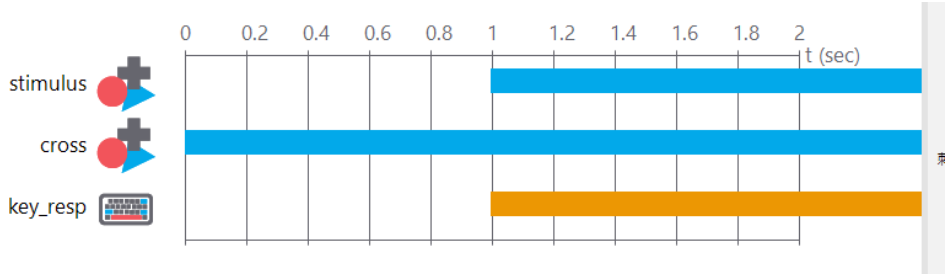


図 3.5　Keyboard コンポーネント配置後のルーチン。反応に関するコンポーネントの有効な時間はオレンジ色のバーで表示されます。

これらの設定が済めばルーチンペインは 図 3.5 のようになるはずです。Keyboard コンポーネントの有効な時間を示すバーが青ではなくオレンジである点に注意してください。反応計測に用いるコンポーネントはこのようにオレンジ色で表示されます。これで、x または / キーが押されたことを検出してルーチンを終了させることができるようになりました。以下のことを必ず確認しておいてください。

- 刺激 (赤い円) が出現した後に x キーまたは / キーを押すと刺激提示が終了すること。
- 刺激 (赤い円) が出現する前に x キーまたは / キーを押しても刺激提示が終了しないこと。
- x キー、 / キー、ESC キー以外のキーを押しても効果がないこと（ただし Windows キーなど OS により特殊な役割を持つキーは除く）。

キー押しが検出できたら次はそれをファイルに記録する方法をマスターしたいところですが、それはひとまず置いておいて、次節では刺激の位置や色を変更しながら試行を繰り返す方法を学びます。

チェックリスト

- ルーチン実行中にキー押しが有効となる時間帯を指定できる。
- キーが押されると直ちにルーチンを中断するように設定することができる。
- 有効とするキーを指定できる。

- すべてのキーを有効にするように設定することができる。

3.4 条件ファイルを作成しよう

今までは「実験」と言いつつただ静止画像が一回スクリーンに表示されて終了するだけでしたが、本節からいよいよ実験らしくなってきます。Builder では、条件ファイルと呼ばれるファイルから試行毎の刺激の色や反応に使用できるキーなどの値を読み込んで設定する機能があります。この機能を用いることによって、今までは不可能だった「刺激の色や位置を変えながら試行を繰り返す」ことが可能になります。条件ファイルの作成方法は何通りかあるのですが、ここでは Excel を用いた方法を解説します。第 1 章で述べたとおり、Excel をお持ちでない方は LibreOffice Calc を代わりに用いることができます。

では、条件ファイルの作成を始めましょう。Builder を一旦離れて Excel を起動してください。Builder で利用できる条件ファイルを Excel で作成するには、Excel のシートの各列に刺激の色や位置等のパラメータを割り当てて、実験で使用するパラメータの組み合わせを 1 条件 1 行で列挙します。 図 3.6 を参考にしながら具体的に手順を追っていきましょう。

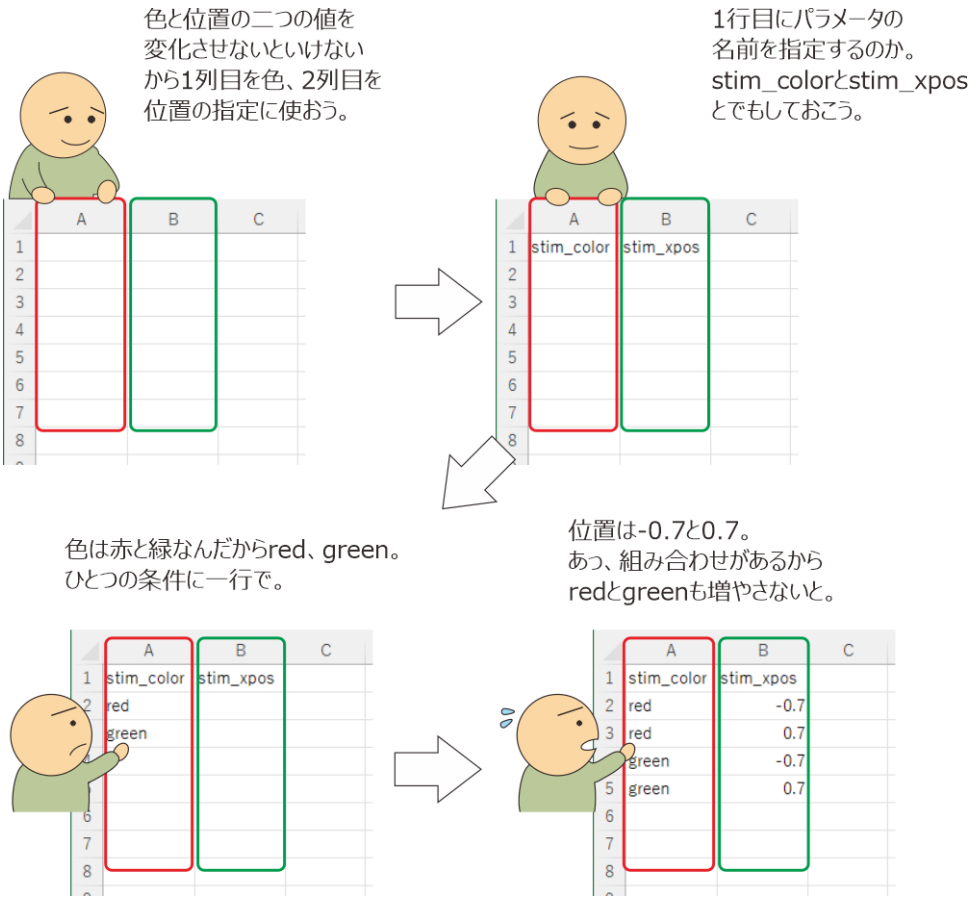


図 3.6　条件ファイルの作成。

今回の実験では試行毎に刺激の「色」と「位置」が変化します。このように変化していくものそれぞれに Excel の列を割り当てていきます。例えば「色」を 1 列目、「位置」を 2 列目に割り当てるとしましょう。このように、条件ファイルを通じて変化させていく値をパラメータと呼びます。この用語は以後頻出しますので覚えておいてください。続いて決めないといけないのは、それぞれのパラメータの「名前」です。人にものを頼む時には「コレをアレに載せといてよ」と言っても通じるかも知れませんが、コンピュータに作業を頼む

ときには「コレ」や「アレ」を曖昧さなしに示す必要があります。そこで用いられるのが名前です。「刺激の色」と言った時に「刺激」とは具体的にどのコンポーネントを指しているのか、「値を設定する」と言った時に「値」とはどこに記されているものか、といったことをコンピュータにもわかるように示すために、私たちがコンピュータに指示を出す時には、操作や参照の対象となるものすべてに固有の名前をつけなければいけません。「固有の名前」とは、言い換えると異なるものに同じ名前を割り当ててはいけないということです。第 2 章 で Polygon コンポーネントを複数配置した時に、一つ目の Polygon コンポーネントの **[名前]** の初期値が「polygon」、二つ目の Polygon コンポーネントの **[名前]** の初期値が「polygon_2」になっていたのに気付かれた方がおられると思いますが、これは Builder が二つの Polygon コンポーネントを明瞭に **[名前]** で区別できるように自動的に異なる名前を割り振ってくれた結果なのです。

Builder で使用できる名前は、「Python の変数として使用できる名前」のうち、PsyhcoPy や Builder でまだ使用されていないものに限られます。現在の PsychoPy のベースとなっている Python3 では日本語の文字や一部の記号も名前に使用できるのですが、Python の文法規則の詳細を知らなければ理解しがたい制限も多いので、以下のルールを満たすものを使用するのが無難でしょう。

- 1 文字目が英文字 (A-Z、a-z) であること。アンダーバー (_) も使用できるがアンダーバーから始まる名前は避けた方がよい。英文字の大文字と小文字は区別される。
- 2 文字目以降が英文字 (A-Z、a-z) またはアンダーバー (_) であること。英文字の大文字と小文字は区別される。
- Python の正常な動作のために確保されているキーワード (if, else, while, for, from, global など) や PsychoPy のモジュール名 (core, data, visual, event など)、Builder の予約語 (expInfo, expName, buttons など) は使用できない。以後、記述の簡略化のためにこれらを合わせて「予約語」と呼ぶ。

一番目と二番目の条件を満たすのはそれほど難しくありませんが、最後の「予約語を使用してはいけない」という条件は厄介です。予約語は Python や Builder が正常に動作するようにすでに使用している名前のことであり、かなりの数の名前が予約済みなので覚えるのはあまり現実的ではありません (「 14.2: 予約語 」参照)。予約語を避けるのに有効なのが、英小文字から始まる接頭語 + アンダーバーという名前を用いるという方法です。例えば自分で決める名前すべて先頭に my_ を付けたり、刺激には s_、反応には r_を付けたりするといったルールを決めておけば、ほぼ確実に予約語を避けることができます。Builder を起動中であれば「 3.11.2:Builder で使用できない名前を判別する 」で述べる方法を用いて Builder で使用できる名前を判別することができます。

本題に戻りましょう。今回は刺激の色と位置を変化させる必要がありますので、二つの名前を自分で決める必要があります。上記の名前に使える文字列の制限に加えて、作成中のルーチンペインにすでに配置済みのコンポーネントの **[名前]** と重複しないように名前を決める必要があります。ここでは色に対応する名前として stim_color、位置に対応する名前として stim_xpos という名前を使うことにしましょう。Excel の 1 行目にこれらの名前を入力してください。

名前を決めたら、次はそれぞれのパラメータの値を列挙していきます。値は Builder が理解できるものでなければいけません。例えば「赤」とか「緑」とか日本語で入力しても Builder は理解できませんので、第 2 章 で覚えたように red や green とか#FF0000、#00AA00 などと書かねばなりません。ややこしいことに$[1, 0, 0] などの表記は$を付けずに [1, 0, 0] と書かないといけないのですが、その理由は本章の「 3.6: パラメータを利用して刺激を変化させよう 」で説明します。刺激の位置も「右」、「左」ではなく刺激の **[位置 [x, y] $]** にそのまま記入できる値を書く必要があります。

これらの注意点を念頭に置いて、まず刺激の色を入力しましょう。刺激の色は赤または緑ですから、stim_color

	A	B	C
1	stim_color	stim_xpos	
2	red	-0.7	
3	red	0.7	
4	green	-0.7	
5	green	0.7	
6			

(赤, 緑)と(-0.7, 0.7)のすべての
組み合わせを呈示する場合

	A	B	C
1	stim_color	stim_xpos	
2	red	0.7	
3	green	-0.7	
4			
5			
6			

赤と0.7、緑と-0.7の
組み合わせだけを呈示する場合

図 3.7　パラメータの組み合わせを全て提示するか、特定の組み合わせだけを提示するかによって条件ファイルの書き方が異なります。

には red と green を挙げる必要があります。これらの値を、stim_color と入力したセルの下へ、空白セルを間に挟まずに並べていきます。続いて刺激の位置ですが、今回の実験では位置の Y 座標は変化しないので X 座標の値だけを列挙することにしましょう。固視点から刺激への距離を 0.7 に決めているのですから、左は-0.7、右は 0.7 です。stim_color と同じように stim_xpos の列に-0.7、0.7 を縦に並べればよいのですが、今回の実験では「赤」の刺激が「右」と「左」、「緑」の刺激が「右」と「左」で合計 4 通りの条件があることに注意しなければいけません。条件ファイルでは 1 行が一つの実験条件に対応しますので、これら 4 通りの条件がすべて列挙されなければいけません。どうすればよいかというと、stim_color の列を red, red, green, green といった具合に red と green が 2 回ずつ出現するように書き換えて、stim_xpos に-0.7、0.7、-0.7、0.7 と書けばよいのです (図 3.7 左)。これですべての条件が挙げられたことになります。この並べ方は Builder で実験を作成する際には非常に頻繁に用いますのでよく覚えておいてください。なお、「赤色の時は必ず右に、緑色の時は必ず左に刺激が出現する」という実験の場合には 2 通りしか組み合わせがありませんので、 図 3.7 右のように red と 0.7、green と-0.7 がそれぞれ同一の行に並ぶように書きます。

今回の実験のパラメータには、刺激の位置と色の他にも「正解となる反応」がありますが、まずは実際に刺激が次々と提示される様子を確認することにしましょう。他のセルに不必要な値が入力されていないのを確認してから、作成したシートを Excel の「ブック (*.xlsx)」形式で保存してください。以後、この形式で保存されたファイルを xlsx ファイルと呼びます。保存するファイル名は、第 3 章の実験用の条件ファイルという事で exp03_cnd.xlsx としておきます。LibreOffice Calc を使っている場合は標準では xlsx ファイルとして保存されませんので保存ダイアログの「ファイルの種類」から xlsx 形式を選択することを忘れないようにしてください (図 3.8)。

これで条件ファイルの準備ができました。Builder に戻ってこの条件ファイルを読み込むように実験を拡張しましょう。

チェックリスト

- Excel または LibreOffice Calc を用いて、実験に用いるパラメータを列挙した条件ファイルを作成することができる。
- 実験のパラメータを表現するために PsychoPy で使用できる名前を決めることができる。

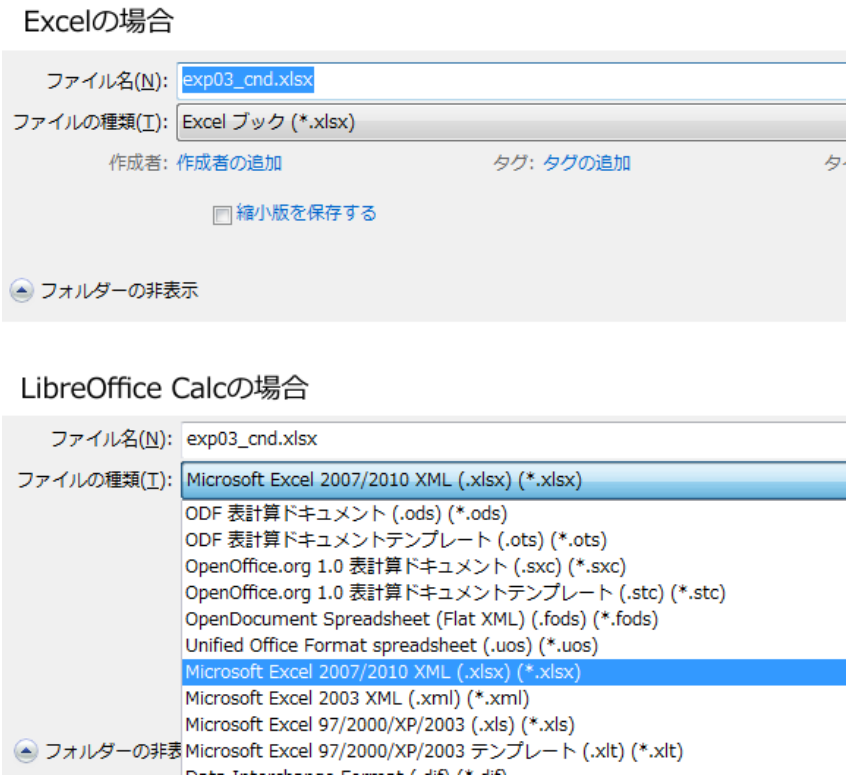


図 3.8　条件ファイルを Excel のブック (*.xlsx) 形式で保存します。Excel の場合は標準で xlsx ファイルとして保存されますが、LibreOffice Calc の場合は [ファイルの種類] から xlsx 形式を選択する必要があります。

3.5 繰り返しを設定しよう

Excel は一旦終了して、Builder に戻ってください。一度 Builder を終了してしまった人は exp03.psyexp を開きましょう。開いたら、ルーチンペインの下のフローペインを見てください。図 3.9 の左上のように、左から右に矢印が描かれていて、矢印の上に trial と書かれた四角いアイコンが描かれています。フローペインは実験の流れ (フロー) を示しており、左から右へ向かって実験が進行します。trial と描かれた四角はルーチンを示しており、trial ルーチンは実験を作成すると自動的に作成されるルーチンです。ここまでの作業で「ルーチンにコンポーネントを配置する」という作業を何度も繰り返しましたが、これらの作業はいずれも trial ルーチンに対して行っていたのです。

さて、この trial ルーチンを繰り返し実行するためには、フローにループを挿入する必要があります。ループを挿入するには、フローペインの左端の「ループを挿入」を左クリックします。クリックした後にフロー上にマウスカーソルを動かすと、ループの始点または終点を挿入できる点にカーソルが重なった時にその場所に黒い円が表示されます (図 3.9 左下)。黒い円が表示されている時にその位置をクリックすると、始点または終点として指定することができます。通常は始点と終点の二カ所を指定する必要がありますが、今回はルーチンが一つしかなくて始点が決まると終点は自動的に決まってしまうため、一カ所をクリックするだけで始点と終点が確定します。

ループの始点と終点が確定すると、ループの設定ダイアログが表示されます。ここではループの名前や種類、ループ回数、条件ファイルなどを指定します。名前はコンポーネントの名前と同様に、Builder 上での表示をわかりやすくするためなどに使えます。問題は **[Loop の種類]** ですが、本章では指定できる種類の内 sequential、

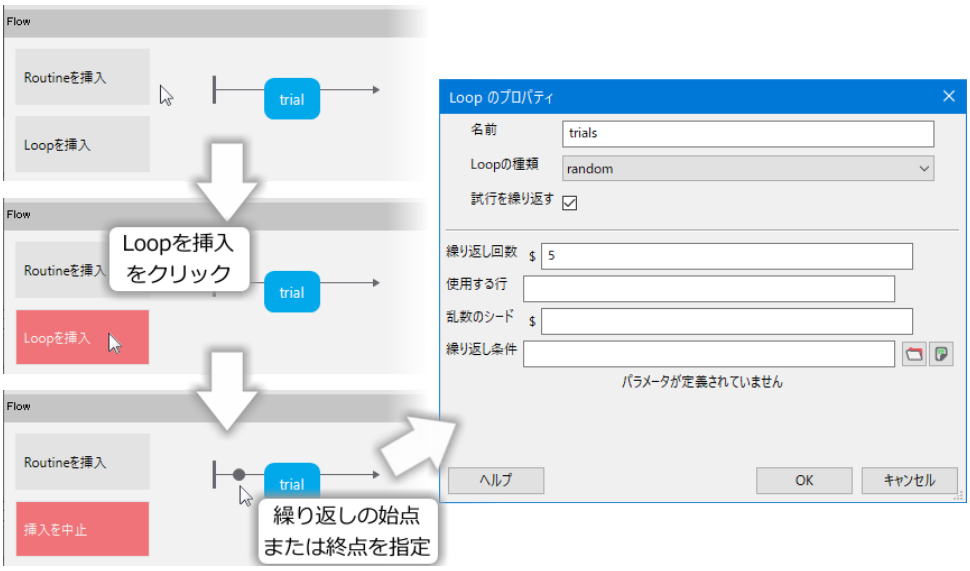


図 3.9　繰り返しの挿入。

random、fullRandom の三種類を解説します。残りの 2 つは 第 11 章 を参照してください。

図 3.10 は、exp03_cnd.xlsx で指定した 4 条件の条件ファイルを sequential、random、fullRandom で繰り返した場合の実行例を示しています。ループ回数を指定する **[繰り返し回数 $]** には 3 が入力されていると仮定しています。sequential では、条件ファイルに書かれた条件が、そのままの順番で 2 行目 (1 行目はパラメータ名) から実行されます。最後の行まで進むとまた 2 行目に戻って繰り返し、 **[繰り返し回数 $]** で指定された回数の繰り返しが終了すると終了します。random と fullRandom では無作為に順番が並べ替えられますが、random では「条件ファイルに書かれた条件を無作為に並び替えて実行する」という作業を **[繰り返し回数 $]** 回繰り返します。従って、それぞれのループでは条件ファイルに書かれた条件が必ず 1 回ずつ実行されます。 図 3.10 の random の例で 1 回目、2 回目、3 回目の繰り返しの全てにおいて条件ファイルに書かれた 4 つの条件が 1 回ずつ実行されていることを確認してください。これに対して、fullRandom では全試行を終えた時点で条件ファイルに書かれた条件がそれぞれ **[繰り返し回数 $]** 回繰り返されます。 図 3.10 の random と fullRandom をよく見比べてその違いを理解してください。一般論として、random では同一条件の試行が続くことはあまりありません (n 回目の繰り返しの最後の試行と n+1 回目の繰り返しの最初の試行が一致した場合のみ) が、fullRandom では同一条件の試行が続くことがしばしばあります。これらのループタイプを使い分けることが Builder を使いこなすコツです。

[乱数のシード $] は、繰り返し順序の無作為にするための処理を操作するための項目です。 **[乱数のシード $]** を空白にしておくと、PsychoPy が実験を実行する度に異なる繰り返し順序を決定してくれます。しかし、例えば 498202 という整数を指定しておくと、この値を変更しない限り、一見無作為な並び替えに見えますが毎回必ず同じ順序の繰り返しが行われます。この整数のことを乱数のシードと呼びます。シードを実験者が自分で指定してその記録をとっておけば、以前行った実験と全く同一の繰り返し順序を再開できます。研究の目的によってはこのような再現性が必要になるかも知れませんので、各自の判断で選択してください。なお、乱数のシードについてもう少し詳しく知りたい方は「 3.11.3: 無作為化と疑似乱数 」をご覧ください。

[繰り返し条件] には、条件ファイル名を入力します。 図 3.11 に示すように入力欄の右側にある 2 つのボタンがあります。左側のボタンをクリックするとファイル選択ダイアログが開くので、先ほど作成した条件ファイルを選択しましょう。特に難しい操作ではないと思いますが、ひとつだけ注意してほしい事があります。 **[繰り返し条件] を入力する前に少なくとも一度、実験 (psyexp ファイル) を保存してください** 。Builder は psyexp

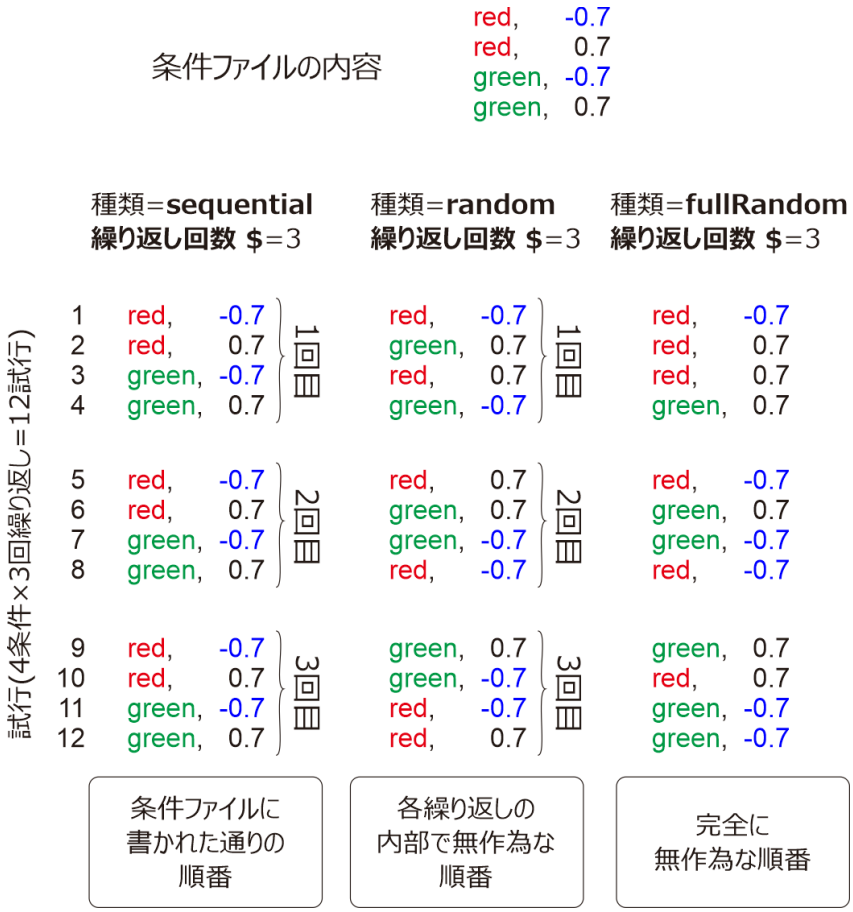


図 3.10　3 種類のループタイプの違い。

ファイルが保存された場所を基準に条件ファイルを探しますので、一度も psyexp ファイルが保存されていない状態で条件ファイルを指定してしまうと、入力結果がおかしくなってしまう場合があります。この章を読みながら作業している方はすでに exp03.psyexp という名前で実験を一度保存しているはずですので問題は生じませんが、今後自分で実験を作成する時には気を付けてください。

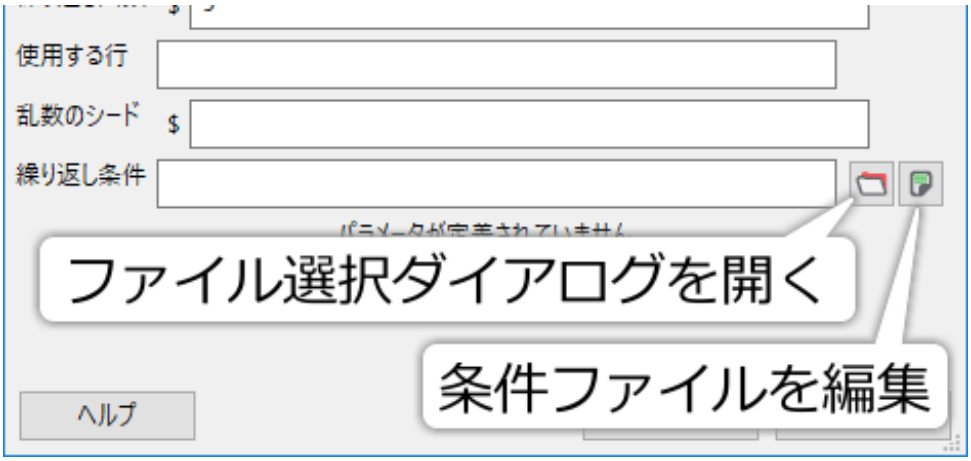


図 3.11　**[繰り返し条件]** の入力欄の右側のボタン。ファイルを設定済みの状態で右端のボタンをクリックすると Excel など xlsx 編集アプリケーションが起動して条件ファイルの内容を確認、修正できます。

[繰り返し条件] に条件ファイルを設定済みの状態で 図 3.11 の右側のボタンをクリックすると、Excel などの

アプリケーションで条件ファイルを開いてくれます。条件ファイルの詳細を確認したい時や、内容を修正したい場合に便利なので覚えておいてください。 **[繰り返し条件]** が空欄の時にこのボタンをクリックすると、内容が空っぽの xlsx ファイルが作成されます。内容を入力して保存しても自動的に **[繰り返し条件]** に設定してくれないので、手作業で設定する必要があります。ご注意ください。

あと **[試行を繰り返す]** というチェックボックスと **[使用する行 $]** という項目がありますが、 **[試行を繰り返す]** については次章で解説します。 **[使用する行 $]** については「 3.11.4:Loop のプロパティ設定ダイアログの **[使用する行 $]** について 」をご覧ください。

さて、解説はこのくらいにして実験の作成を進めましょう。今回の実験では **[Loop の種類]** に fullRandom を用いることにして、 **[繰り返し回数 $]** には 25 を入力しましょう。 **[乱数のシード $]** は空白で良いでしょう。条件ファイルに 4 つの条件が定義されているので、それぞれを 25 回ずつ繰り返すと 25 × 4=100 試行となり、最初の計画と一致します。そして **[繰り返し条件]** の項目の右端の「選択…」ボタンを押して条件ファイル (exp03_cnd.xlsx) を選択してください。正常に読み込むことができれば、 図 3.12 左のように **[繰り返し条件]** の上に条件ファイルの概要が表示されます。 図 3.12 では「2 パラメータ, 4 条件」と書かれていますが、「4 条件」が条件ファイルの行数 (先頭行を除く)、「2 パラメータ」が列数に対応しています。次の行に [stim_color, stim_xpos] と書かれているのは条件ファイル先頭行に入力したパラメータ名です。パラメータ名の順番は Builder が扱いやすいように自動的に並べ替えるので、パラメータ名に過不足がないことだけを確認してください。exp03_cnd.xlsx で定義したパラメータ名と一致していることが確認できたら、ダイアログの OK をクリックしてダイアログを閉じましょう。すると 図 3.12 右のようにフローペインにループ (左側へ戻る矢印) が表示されます。ループの矢印上に trials と書かれた白い四角が表示されていますが、これはループのプロパティ設定ダイアログの **[名前]** で設定したループの名前を示しています。

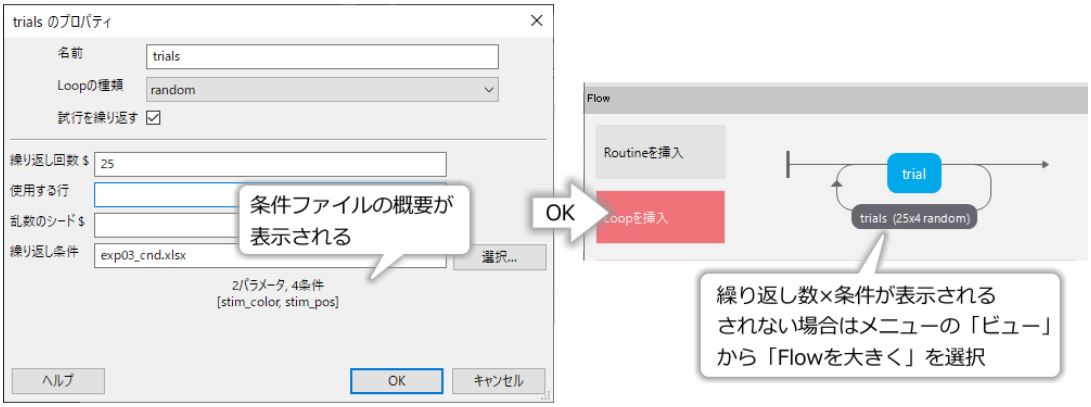


図 3.12 **[繰り返し条件]** に条件ファイルを指定すると、条件ファイルの概要が表示されます。概要が表示されない場合は条件ファイルの読み込みに失敗していますので条件ファイルの内容を確認しましょう。

一度設定したループを再設定したい場合は、ループの名前が書かれた白い四角にマウスカーソルを重ねて左ボタンをダブルクリックしてください。削除したい場合は、同じくマウスカーソルを重ねた状態で右クリックをしてください。「削除」という項目だけがあるメニューが表示されるので、「削除」を選択しましょう (図 3.13)。

さて、これでいよいよ赤や緑の刺激が左右に表示されるようになりました、と言いたいところですが、まだもう一つ作業が残っています。この節の作業によって、Builder は条件ファイルから条件を読み取って、パラメータを変化させながらルーチンを繰り返し実行できるようになりました。しかし、肝心のルーチンが条件ファイルから読み取られたパラメータを利用できるようになっていません。次の節では、読み取ったパラメータを利用する方法を解説します。

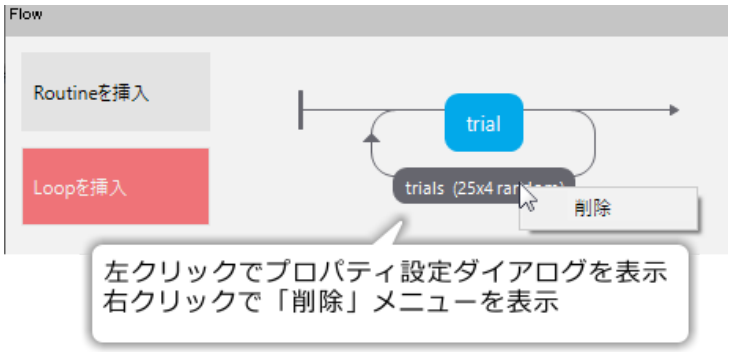


図 3.13　ループの再設定と削除

チェックリスト

- フローペインでループの挿入を開始できる。
- ループを挿入する際、フローペインの矢印上に表示される黒い円の意味を説明できる。
- ループのプロパティ設定ダイアログで条件ファイルを指定し、表示された条件ファイルの概要が適切か確認できる。
- **[Loop の種類]** の sequential、random、fullRandom の違いを説明できる。
- **[繰り返し回数 $]** の値、条件ファイルに含まれる条件数、ルーチンが繰り返される回数の関係を説明できる。
- **[乱数のシード $]** に値を設定するとどのような効果が得られるか説明できる。
- 条件ファイルを **[繰り返し条件]** に指定する前に psyexp ファイルを保存すべき理由を説明できる。

3.6 パラメータを利用して刺激を変化させよう

手始めに、条件ファイルから読み取ったパラメータを使って刺激の位置を左右に変化させてみましょう。ルーチンペインに戻って、stimulus のプロパティ設定ダイアログを開いてください。 **[位置 [x, y] $]** の欄には (0.7, 0) と入力されているはずですが、これを [stim_xpos, 0] に書き換えてください。そして、入力欄の右側にある「更新しない」と書かれたプルダウンメニューをクリックして「繰り返し毎に更新」を選択しましょう (図 3.14)。

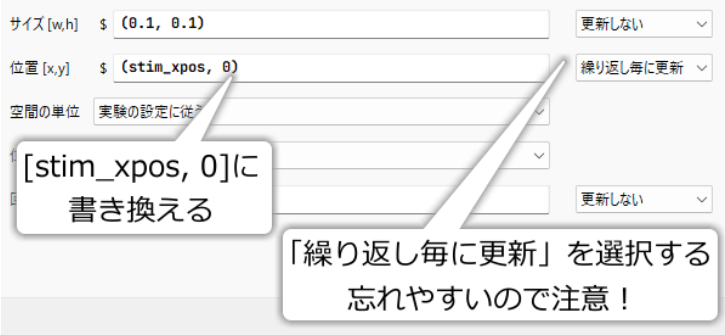


図 3.14　条件ファイルからパラメータを読み込んで自動的にプロパティを更新するように設定します。

このようにプロパティ設定ダイアログの項目に、条件ファイル先頭行のパラメータ名を書くと、Builder が実験を実行する時にパラメータの値を自動的に更新してくれます。右側のメニューを「繰り返し毎に更新」にし忘れるという失敗は、ある程度慣れてきたユーザーでも時々うっかりしてしまいがちなので、疑わしい時には警告を表示してくれる機能が Builder には備わっています。 図 3.15 のように入力内容が赤色で表示され、ダイアログの下部に「更新されるように設定すべき変数を含んでいるようです」と表示されている場合は、入力内容が間違っていないか、「更新しない」から変更する必要がないかよく確認しましょう。

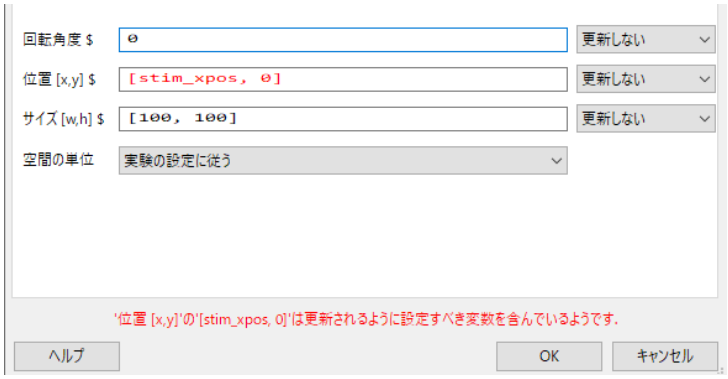


図 3.15　「繰り返し毎に更新」を選択し忘れると警告が表示されます。

Builder の警告機能は万全ではありませんので、「繰り返し毎に更新」に設定すべきところを忘れたまま実験を実行してしまうことも起こり得ます。そのようなときは、実験が突然終了して Runner の画面に戻ってしまいます。この時、Runner の「標準出力」の欄を見ると、 図 3.16 のように「stim_xpos なんて値は知らないよ」というメッセージが表示されているはずです。「繰り返し毎に更新」に設定していないと Builder が条件ファイルから読み込んだ値を stim_xpos に代入するという作業が行われないので、stim_xpos という名前の変数が存在していないというのがこのメッセージの意味なのですが、かなり分かりにくいですね。Builder がもう少しわかりやすいメッセージを出してくれたらと思うのですが、技術的にとても難しいことなので人間ががんばって解釈してあげる必要があります。

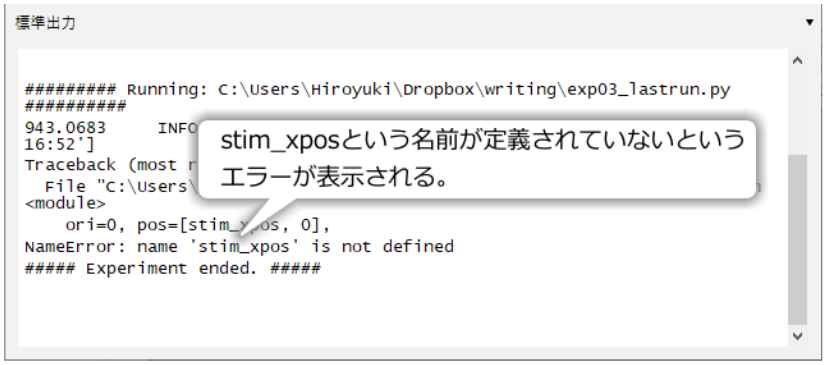


図 3.16　条件ファイルからパラメータを読み込むつもりなのに「更新しない」のまま実行してしまった時に表示されるエラー。

変更作業が終わったら、実験を実行して刺激の位置が左右に変化することを確認してください。これでようやく実験らしくなってきました。そして、エラーで実験が止まることを体験しておくために、ぜひ一度「更新しない」に戻して実行してみてください (体験した後はまた「繰り返し毎に更新」に戻すことを忘れずに！)。

続いて刺激の色も条件に応じて更新するようにしましょう。基本的には位置の更新と同じなのですが、ここにも一つ注意しないといけない点があります。今まで敢えて説明していなかったのですが、プロパティ設定ダイアログの項目には$が最後についているものとついていないものがあることに皆さんは気付いておられるで

しょうか。例えば **[位置 [x, y] $]** や **[回転角度 $]** には$が付いていますし、 **[塗りつぶしの色]** や **[文字列]** には$が付いていません。この$記号は、難しい言い方をしますと「入力した値が文字列として評価されるか ($なし)、Python の式として評価されるか ($あり)」を表しています。

具体的な例を挙げましょう。Text コンポーネントの **[文字列]** に Target と入力されているとします。これは画面上に Target という文字列を表示したいということでしょうか。それとも、条件ファイルに Target という名前のパラメータが定義されていて、そのパラメータの値を代入したいということでしょうか。条件ファイルという便利なものを導入した代償に、「各コンポーネントのプロパティに文字列が入力されている場合、それはデータとしての文字列なのか、パラメータの名前なのか」を区別する方法を考えなければいけなくなってしまいました。ここで登場するのが$です。Builder では、プロパティに入力された値に$が含まれていなければ、その値は文字列であるとみなされます (図 3.17)。従って、先ほど挙げた「 **[文字列]** に Target と入力されている」例では Target という文字列をスクリーン上に提示したいのだと解釈されます。もし$Target と書かれていれば、Target という名前のパラメータの値を条件ファイルから読み込むのだと解釈されます。当然、条件ファイル内で Target という名前のパラメータが定義されていなければエラーになります。

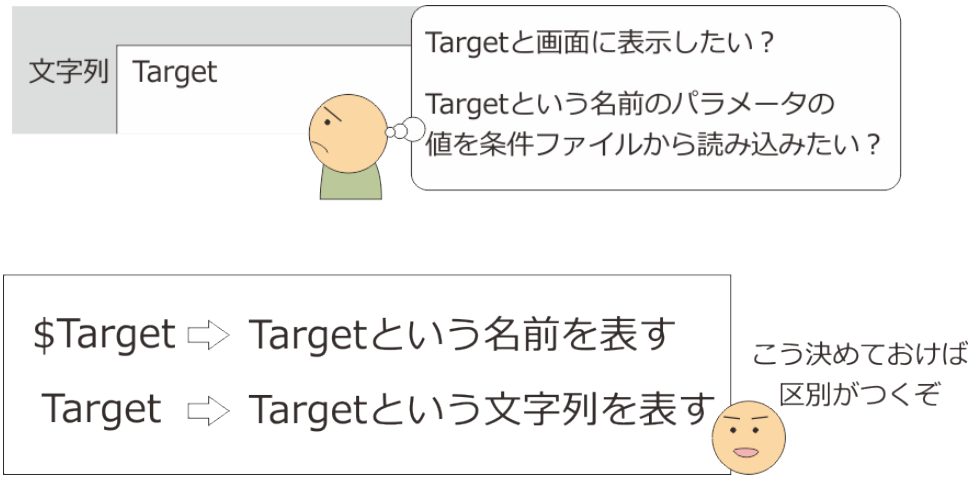


図 3.17　$を付けることによって入力した値が名前を表しているのか文字列を表しているのかを区別します。

ここで、読者の皆さんの中には「ちょっと待てよ、じゃあ 図 3.14 の時はなんで$[stim_xpos, 0] じゃなくて [stim_xpos, 0] でパラメータの値を読み込めたの？」と疑問を持った方もいるかも知れません。確かに 図 3.17 の規則に従うのであれば、 図 3.14 は$[stim_xpos, 0] と書かないといけないはずです。ここで注目してほしいのが **[位置 [x, y] $]** というプロパティ名の最後の$記号です。この$は本来入力欄に書くべきだった$がプロパティ名に引っ越してきたと思えばよいでしょう。考えてみれば、 **[位置 [x, y] $]** や **[回転角度 $]** と言ったプロパティには文字列が指定される可能性はありません。例えば論文の手続きに「刺激を X 度回転した」と書いてあったら、常識で考えて X は変数であって文中のどこかで「X は 30 または 60」とかいう具合に数値が指定されているはずでしょう。それと同じことで、位置や回転角度のように数値を指定すべきプロパティに文字列が書いてあったら、常識的に考えてそれは名前であるはずです。だから、最初からプロパティ名に$を含ませてしまって、いちいち$を入力しなくてよいようになっているのです (図 3.18)。

以上の解説が済んだところで、ようやく条件ファイルを使って刺激の色を変化させる方法を説明することができるようになりました。刺激の色は **[塗りつぶしの色]** と **[枠線の色]** で指定します。これらのプロパティに条件ファイルで定義したパラメータ stim_color の値を代入したいのですが、 **[塗りつぶしの色]** にも **[枠線の色]** にも$はついていません。ではどのように書いたら良いか、もうお分かりですね。正解を 図 3.19 に示します。「繰り返し毎に更新」を選択するのを忘れないようにしてください。設定ができたら、実験を実行してみましょう。今度は刺激の位置も色もきちんと条件ファイルに従って変化するはずです。

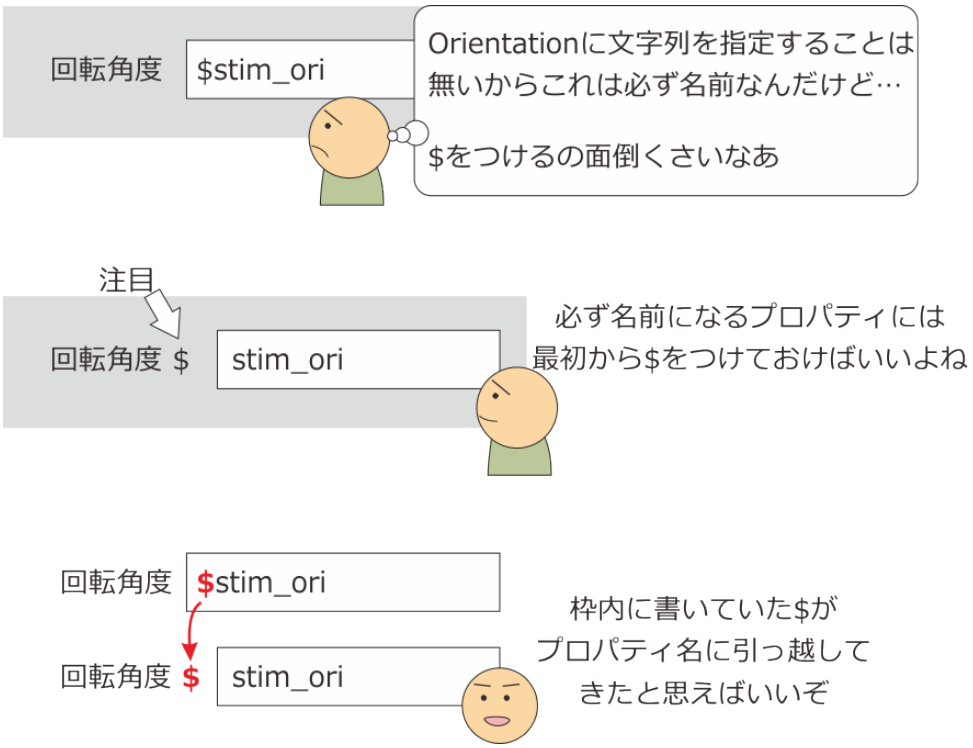


図 3.18　プロパティ名に$が付いているものは、文字列が指定されることがあり得ないプロパティなので、$を入力しなくても名前として解釈されます。

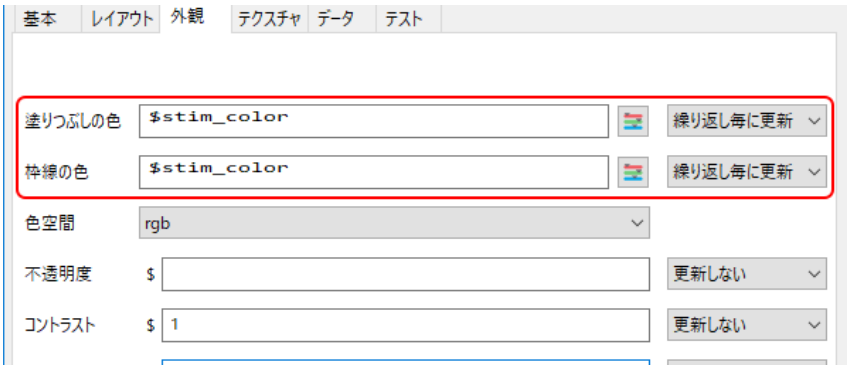


図 3.19　条件ファイルを用いて刺激の色を繰り返し毎に更新する時の設定例。「高度」タブですので注意してください。

これでようやく刺激が完成しました。次は実験記録を保存したファイルを確認しますが、その前に色の指定に関してここまで説明を後回しにしていた問題を取り上げましょう。まず、第 2 章 で色の指定方法について学んだ時に、色空間内の座標による指定では$[1.0, 0.0, 0.75] のように$を必ず付けていました。この節の内容を理解した方は、なぜ$が必要であったか想像がついているのではないかと思います。 **[塗りつぶしの色]** 、 **[枠線の色]** 、 **[色]** にはいずれも$が付いていないので、$なしで [1.0, 0.0, 0.75] と書かれていると Builder には「[1.0, 0.0, 0.75] という名前の色」に見えてしまうのです。当然こんなヘンテコな名前は web/X11 Color name にはありませんので、Builder は「こんな名前の色は知らない」というエラーを出して止まってしまいます。一方、本章で条件ファイルの作成方法を解説した際に「条件ファイル内において色空間座標値で色を指定する場合は$なしで書かねばならない」と述べましたが、こちらはちょっと事情が複雑です。ごく簡単に説明しますと、Builder は条件ファイルから読み込んだ値に関しては「名前ではありえない」と考えます。ですから、条件ファイル内に書かれた$記号は「名前か、文字列か？」を判断するための記号としてではなく、単なる文字として解釈されます。ですから、条件ファイル内に$付きで$[1.0, 0.0, 0.75] と書かれていると、余計な$が付い

ていることになって色座標値として解釈されなくなってしまうのです。まあ、この辺りはよくわからなければとりあえず置いておいて、「条件ファイル関連でうまく値が読み込まれない時は$記号に問題があるかも知れない」と疑うことを覚えておいていただければ十分かもしれません。では$記号そのものを文字列として表示させたい場合はどうしたらいいの？という点については「3.11.5:$を含む文字列を提示する」に書いておきましたので、興味がある方はご覧ください。

チェックリスト

- 条件ファイルで定義したパラメータを用いてコンポーネントのプロパティ値を更新できるように設定できる。
- コンポーネントのプロパティ名の最後に$が付いているものと付いてないものの違いを説明できる。

3.7 実験記録ファイルの内容を確認しよう

製作中の実験もかなり体裁が整ってきたので、ここで一度実験記録ファイルの内容について解説しておきましょう。以下の手順 (図 3.20) で設定を変更してから実験を実行してください。ちょっと面倒ですが、100 試行すべて行ってください。

- data フォルダ内に作成されているファイルをすべて削除する。
- 実験設定ダイアログを開いて、hdf5 形式以外のすべての実験記録ファイルを出力するように設定する。すなわち、「xlsx 形式のデータを保存」、「CSV 形式のデータを保存 (summaries)」、「CSV 形式のデータを保存 (trial-by-trial)」にチェックを付ける (「pydat 形式のデータを保存」のチェックはチェックが付いている状態から変更できない)。
- 実験を実行し、実験情報ダイアログの participant に foo と入力して実行する。

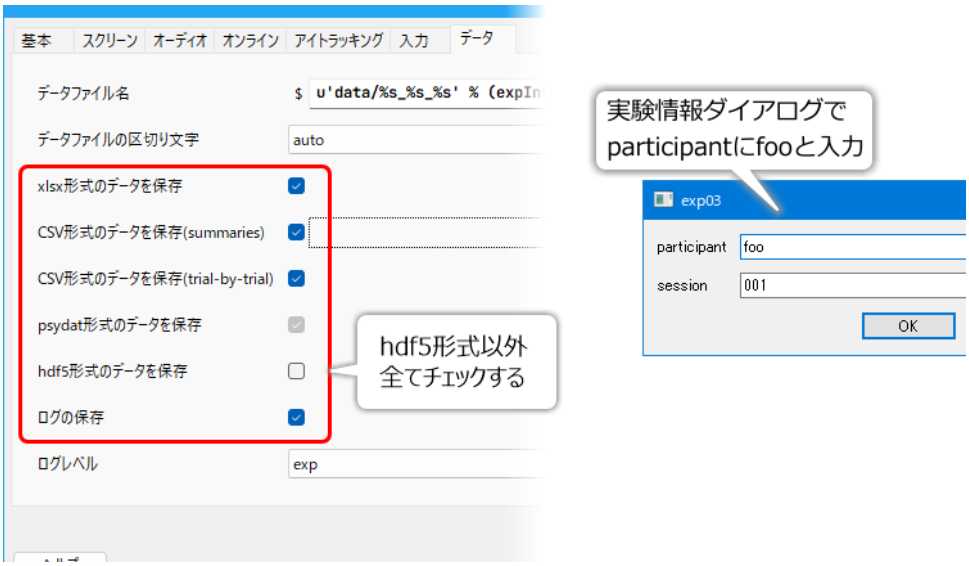


図 3.20　実験設定ダイアログを開いてすべての記録ファイルを保存するようにチェックして実験を最後まで実行してください。実験開始時の実験情報ダイアログの participant に foo と入力してください。

実験が終了したら、data フォルダの内容を確認してください。 表 3.2 に示したファイルができているはずです。ファイル名の先頭部分には participant に入力した文字列が代入されており、続いて実験を実行した時刻

が続きます。participant に入力する文字列を工夫すると、データの整理が非常に楽になります。participant を空白のまま実験を実行すると、アンダースコア (_) の後に実行時刻が続くファイル名になります。participant にファイル名として使用できない文字列を指定しないように注意してください。

表 3.2　Builder が作成する実験記録ファイル (foo は実験情報ダイアログの participant に入力した文字列、yyyy_mm_dd_hhMM は年 (西暦)、月、日、時、分)

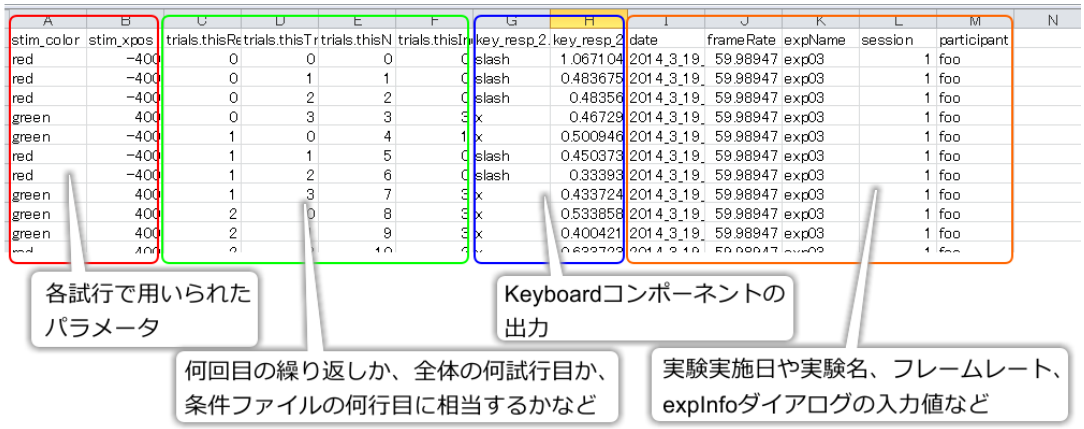
ファイル名	概要
foo_yyyy_mn_dd_hhMM.xlsx	条件毎に結果をまとめた Excel ファイルを保存する。実験設定ダイアログの「xlsx 形式のデータを保存」をチェックすると作成される。
foo_yyyy_mn_dd_hhMMtrials.csv	条件毎に結果をまとめた CSV ファイルを保存する。実験設定ダイアログの「CSV 形式のデータを保存 (summaries)」をチェックすると作成される。
foo_yyyy_mn_dd_hhMM.csv	試行毎に結果をまとめた CSV ファイルを保存する。実験設定ダイアログの「CSV 形式のデータを保存 (trial-by-trial)」をチェックすると作成される。
foo_yyyy_mn_dd_hhMM.psydat	Python の cPickle というモジュールを用いて出力された実験記録。自分で分析用 Python スクリプトを書く場合に用いる。実験設定ダイアログの「pydat 形式のデータを保存」をチェックすると作成される。
foo_yyyy_mn_dd_hhMM.log	実験中の PsychoPy の動作記録が保存されている。実験設定ダイアログの「ログレベル $」の選択肢によって内容が変化する。PsychoPy 自体の動作解析や開発などに用いる。通常のデータ分析向きではない。実験設定ダイアログの「ログの保存」をチェックすると作成される。

作成されるファイルのうち、拡張子が.log と.psydat のファイルはとりあえず無視してよいでしょう。.psydat ファイルは「データ分析は自作の Python スクリプトをでする」という人しか使わないでしょうし、.log ファイルは通常のデータ分析では必要のないファイルです。ただ、「実験が正常に動作していたのか？」という事が問題になった時には.log ファイルがあると役に立ちますので、学会や論文で公表する可能性がある実験の.log ファイルは保存しておいた方がよいでしょう。

データ分析の際に基本となるファイルは、foo_yyyy_mn_dd_hhMM.csv です。拡張子の.csv は、このファイルの保存形式がカンマ区切りのテキストファイル (comma-separated values; CSV) であることを意味しています。Excel も LibreOffice Calc も CSV ファイルを開くことができますので、以下の解説ではこれらのアプリケーションで CSV ファイルを開くという前提で進めます。

foo_yyyy_mn_dd_hhMM.csv を Excel で開くと (LibreOffice Calc でも同様、以下省略)、シートの 1 行が 1 回の試行に対応する形で実験記録が保存されていることを確認できます。このファイルは、実験設定ダイアログで「CSV 形式のデータを保存 (trial-by-trial)」をチェックしていると作成されるので、以下 trial-by-trial 記録ファイルと呼ぶことにします。1 行目は各列の見出しで、2 行目が第 1 試行、3 行目が第 2 試行という具合に 101 行目 (第 100 試行) までデータがあることを確認してください。列数は条件ファイル数に含まれるパラメータ数やルーチンに配置されたコンポーネントによって変化しますが、一般的に以下の内容が含まれます (

図 3.21)。



stim_color	stim_xpos	trials.thisRe	trials.thisTr	trials.thisN	trials.thisIn	key_resp_2.	key_resp_2	date	frameRate	expName	session	participant
red	-400	0	0	0	0	slash	1.067104	2014_3_19_	59.98947	exp03	1	foo
red	-400	0	1	1	0	slash	0.483675	2014_3_19_	59.98947	exp03	1	foo
red	-400	0	2	2	0	slash	0.48356	2014_3_19_	59.98947	exp03	1	foo
green	400	0	3	3	3	x	0.46729	2014_3_19_	59.98947	exp03	1	foo
green	-400	1	0	4	1	x	0.500946	2014_3_19_	59.98947	exp03	1	foo
red	-400	1	1	5	0	slash	0.450373	2014_3_19_	59.98947	exp03	1	foo
red	-400	1	2	6	0	slash	0.33393	2014_3_19_	59.98947	exp03	1	foo
green	400	1	3	7	3	x	0.433724	2014_3_19_	59.98947	exp03	1	foo
green	400	2	0	8	3	x	0.533858	2014_3_19_	59.98947	exp03	1	foo
green	400	2		9	3	x	0.400421	2014_3_19_	59.98947	exp03	1	foo

図 3.21　trial-by-trial 記録ファイルの内容。「各試行で用いられたパラメータ」は条件ファイルを使用した場合のみ、「Keyboard コンポーネントの出力」は Keyborad コンポーネントを使用した場合のみ出力されます。

各試行で用いられたパラメータ　条件ファイル内で定義されているパラメータの内、どの値が使用されたかが示されています。この情報を含む列の数は条件ファイルの列数に対応しています。実行した実験で条件ファイルが使用されていない場合は出力されません。

繰り返しに関する情報　その行の試行が何回目の繰り返しの何試行目にあたるか、全体を通して数えて何試行目にあたるかといった情報が示されています。 図 3.21 の trials.thisRepN は何回目の繰り返しか、trials.thisTrialN はその繰り返しにおける何試行目か、trials.thisN は全体を通して何試行目か、trials.thisIndex はその試行で用いられたパラメータが条件ファイルの何個目の条件に相当するかを示しています。Python では「順番は 0 番目から数える」という規則がありますので、第 1 試行が 0 と表記されていることに注意してください。また、列見出しの trials の部分は Builder におけるループの名前に対応しています。

刺激の開始・終了に関する情報　各試行で刺激が (描画) 開始・終了された時刻が、実験開始を 0 秒として秒単位で出力されています。何か実験中に異常があって描画タイミングがおかしくなったりしていなかったかをこれらの値で確認することができます。終了時刻が設定されていない刺激では終了時刻が None となります。この情報が不要な場合は「 3.11.7: コンポーネントの開始・終了時刻をデータファイルに出力しないようにする 」を参照してください。

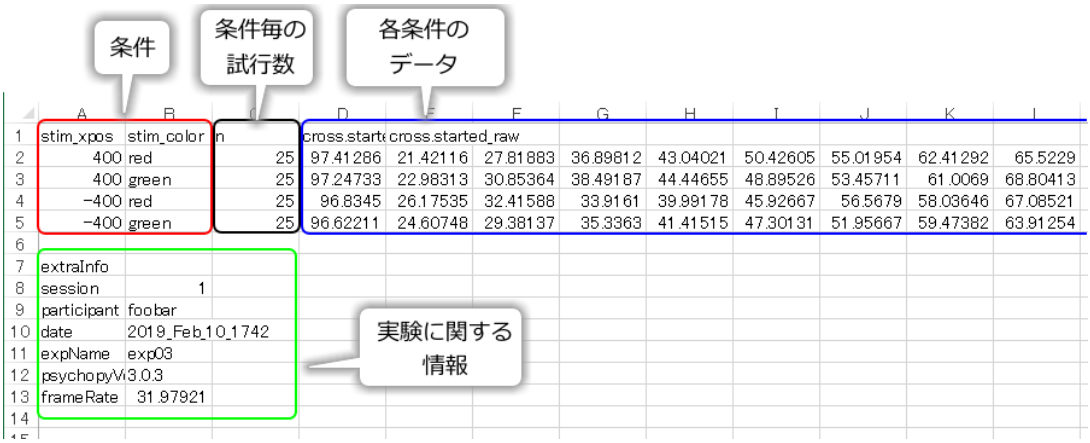
キー押しに関する情報　実験に Keyboard コンポーネントが含まれていなければ出力されません。key_resp_2.keys は押されたキーのキー名、key_resp_2.rt は Keyboard コンポーネントが有効になってからキーが押されるまでの時間を示しています。単位は秒です。列見出しの key_resp_2 の部分は Builder における Keyboard コンポーネントの名前に対応しています。なお、時間が小数点以下ものすごい桁数まで出力されていますが、計測に詳しい方は「一体この値はどの程度精度があるのか」と不安になるかもしれません。PsychoPy Builder が出力する時間の精度については「 3.11.6:PsychoPy の時間計測の精度について (上級) 」をご覧ください。

キー押し検出開始・終了に関する情報　実験に Keyboard コンポーネントが含まれていなければ出力されません。Keyboard コンポーネントがキー押し検出を開始・終了した時刻が、実験開始を 0 秒として秒単位で出力されています。終了時刻が設定されていない場合は終了時刻が None となります。その他の実験に関する情報実験実施日や実験名、実験情報ダイアログで入力した値、使用した PsychoPy のバージョ

ン、フレームレートなどが示されています。フレームレートとは、1 秒あたりに描画したフレーム数のことで、モニターのリフレッシュレートと大きく異なる場合はスクリーンの描画に問題が生じている可能性があります。リフレッシュレートとフレームについては「2.10.6: 時刻指定における frame について」をご覧ください。特にこの値がモニターのリフレッシュレートより大幅に小さい場合は、スクリーンの描画に遅延が生じていて刺激が適切な時刻に表示されていない恐れがあります。

今回の実験では、刺激が提示される位置と反応する指の関係が反応の速さや正確さにどのように影響するかを調べるのが目的ですから、刺激の位置と色の組み合わせ別に、正答率と反応時間を計算すればよいでしょう。すでに自分自身でいろいろなアプリケーションを用いて実験を行っている方は各自で好みの方法で分析していただければよいですが、こういった分析が初めての方はとりあえず stim_color と stim_xpos でデータを並べ替えしてみましょう。するとそれぞれの条件に対応する試行が集まりますので、ワークシート関数を用いて平均値を求めたり、条件毎に x キーと slash キーが押された回数を数えれば計算することができますが、平均値の計算と比べれば手間がかかります。実は、Builder には参加者が押したキーが正答であったか否かを判定して記録する機能があり、これを使うと正答率の計算は楽になります。次節ではこの方法を学びます。

続いて「xlsx 形式のデータを保存」をチェックすることで出力される拡張子が xlsx のファイルと、「CSV 形式のデータを保存 (summaries)」をチェックすることで出力される拡張子が csv のファイルについても確認しておきましょう。これらはいずれも条件ファイルで定義された実験条件ごとに結果を 1 行に要約したもので、内容はほぼ同じです。拡張子が xlsx のファイルを開いてみてください。 図 3.22 のような内容になっているはずです。



	A	B	C	D	E	F	G	H	I	J	K	L
1	stim_xpos	stim_color	n	cross.starte	cross.started_raw							
2	400	red	25	97.41286	21.42116	27.81883	36.89812	43.04021	50.42605	55.01954	62.41292	65.5229
3	400	green	25	97.24733	22.98313	30.85364	38.49187	44.44655	48.89526	53.45711	61.0069	68.80413
4	−400	red	25	96.8345	26.17535	32.41588	33.9161	39.99178	45.92667	56.5679	58.03646	67.08521
5	−400	green	25	96.62211	24.60748	29.38137	35.3363	41.41515	47.30131	51.95667	59.47382	63.91254
6												
7	extraInfo											
8	session	1										
9	participant	foobar										
10	date	2019_Feb_10_1742										
11	expName	exp03										
12	psychopyVi	3.0.3										
13	frameRate	31.97921										
14												

図 3.22　xlsx 記録ファイルの内容。データが 1 条件 1 行でまとめられています。

ひとつの実験条件が 1 行にまとめられるため、本章の例では 4 行となります。条件ファイルの行数と同じですね。各行の最初に条件ファイルに記述されているパラメータが示されています。パラメータの次には n という見出しの列があり、各条件に対応する試行が何試行あったかが記されています。 図 3.22 ではすべて 25 となっています。一般に、n の値はループの **[繰り返し回数 $]** に一致します。さらに続いて、各条件のデータが数十列にわたって出力されています。これらの行の下には、実験の実施時刻やフレームレート、実験情報ダイアログの内容といった実験に関する情報が記されています。この記録形式は **[繰り返し回数 $]** が増えると非常に列数が多くなるという問題があるのですが、条件毎の正答率や平均反応時間といった情報が出力されるので、trial-by-trial 記録ファイルから自分で計算する手間が省けるという利点があります。章末の「3.11.8:「CSV 形式のデータを保存 (summaries)」・「xlsx 形式のデータを保存」で作成される記録ファイルの形式」に詳しく解説しておきますので、興味がある方はそちらをご覧ください。どちらを使うかは好みの問題ですが、本書では trial-by-trial 記録ファイルを使用します。

チェックリスト

- expInfo の participant に設定した文字列が記録ファイル名にどのように反映されるかを説明できる。
- data フォルダに作成される拡張子 log と psydat のファイルに何が記録されているか、概要を説明することができる。
- CSV ファイルの CSV とは何の略であり、何を意味しているか説明することができる。
- trial-by-trial 記録ファイルから、各試行で用いられたパラメータの値を読み取ることができる。
- trial-by-trial 記録ファイルから、各試行において押されたキーのキー名と反応時間を読み取ることができる。
- trial-by-trial 記録ファイルに記録されている反応時間の計測開始時点 (0.0 秒になる時点) がどのように決まるか答えることができる。
- trial-by-trial 記録ファイルから実験情報ダイアログで入力した値を読み取ることができる。
- trial-by-trial 記録ファイルから実験実行時のフレームレートを読み取ることができる。
- 分析時に未変更の記録ファイルを保存しておいた方がよい理由を説明できる。

3.8 反応の正誤を記録しよう

Builder の画面に戻って、exp03.psyexp を開いてください。そして Keyboard コンポーネントのプロパティ設定ダイアログを開きましょう。 図 3.23 に示す通り **[正答を記録]** という項目がありますが、この項目をチェックすると **[正答]** という項目に入力できるようになります。ここに正答となるキー名を入力しておくと、Builder が押されたキーが正答キーと一致するか否かを判定して記録ファイルに保存してくれます。x キーが正解ならば'x'、 / キーが正解であれば'slash' です。GO/NOGO 課題のように「押さないのが正解」の場合は None と入力します。None の前後にクォーテーションマークが付いていないことに気を付けてください。

多くの心理学実験では、条件によって正答となる反応が変化します。従って、 **[正答]** に入力するキー名は条件毎に変化させる必要があります。「条件毎に変化する」となると、条件ファイルの出番です。刺激が赤色の時に x キー、緑色の時に / キーを押すのが正答ですが、この関係は作成済みの条件ファイルに含まれている値からはわからないので、正答のキー名を示すパラメータ列を条件ファイルに追加する必要があります。刺激の位置と色を変化させた時にはまず条件ファイルの作成から始めましたが、どのようなパラメータ名を使用するかすでに決めているのであれば、Builder 側の作業から始めてしまっても問題はありません。correct_ans という列を条件ファイルに追加して、そこへ正答キー名を入力することにしましょう。

Keyboard コンポーネントのプロパティ設定ダイアログの **[正答]** に「correct_ans から値を代入して繰り返し毎に更新する」ように設定しないといけないのですが、ここで条件ファイルの参照について皆さんがきちんと理解しているかテストです。 **[正答]** に入力する値は correct_ans ですか、それとも$correct_ans ですか？ 答えは$付きの$correct_ans です。なぜ$が必要かをきちんと説明できない方は、もう一度$記号について復習しておきましょう。繰り返し毎の更新でもう一つ注意すべき点として「「繰り返し毎に更新」を忘れずに選択する」というものがありましたが、 **[正答]** の右側にはそもそもプルダウンメニューがありません。この項目に関しては、「繰り返し毎に更新」を選ばなくても更新が行われます。

図 3.23　反応の正誤を記録するよう設定する手順。

[正答] の設定が終わったら、今度は Excel を開いて条件ファイルを編集しましょう。correct_ans と 1 行目に書かれた列を追加して、正答キー名を入力してください。正答キーは刺激の色と対応しているのですから、stim_color が red の行は x、green の行は slash が正答キーです (図 3.24)。入力したら条件ファイルを保存しましょう。条件ファイルの名前や保存場所を変更していなければ、Builder で条件ファイル名を設定し直す必要はありません。試しに Builder に戻ってフローペインのループを左クリックしてループのプロパティ設定ダイアログを表示させると、相変わらずパラメータは stim_color と stim_xpos の二つと表示されています。しかし、実行するときちんと新しく追加したパラメータも読み込まれません。正しく表示されないと気持ち悪いという方は、もう一度条件ファイルを設定し直してください。正しく stim_color、stim_xpos と correct_ans の 3 パラメータが表示されるはずです。

	A	B	C
1	stim_color	stim_xpos	correct_ans
2	red	-0.7	x
3	green	-0.7	slash
4	red	0.7	x
5	green	0.7	slash
6			

図 3.24　条件ファイルに correct_ans の列を追加し、正答キー名を入力します。

Keyboard コンポーネントと条件ファイルの修正が終わったら、実験を実行してみましょう。きちんと 100 試行終わるまで実行してください。終了したら、trial-by-trial 記録ファイルの内容を確認してみましょう (図 3.25)。trial-by-trial 記録ファイルには key_resp_2.corr という列が追加されています。この列の値が 1 の試行は正答で、0 の試行は誤答です。ということは、その平均値を計算すれば正答率 (0.0=全問不正解、1.0=全問正解) を計算できるというわけです。

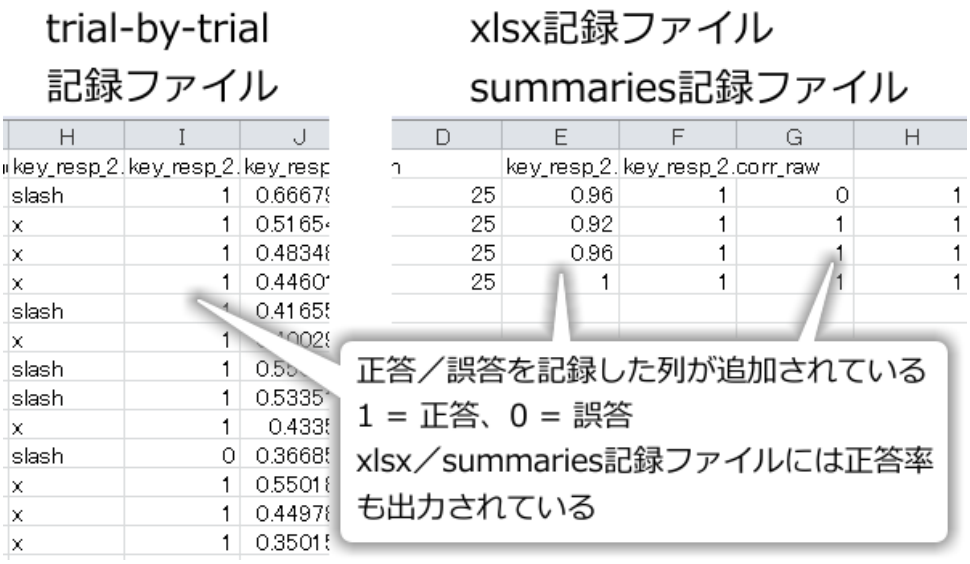


図 3.25　**[正答]** を設定すると記録ファイルに正答／誤答が記録されます。

以上で、サイモン効果を題材としたこの章の実験は「一応」完成しました。「一応」というのは、確かに最初に計画した目標は一通り達成しているのですが、実際にこの psyexp ファイルを使って参加者を募って実験するとなるといくつか不都合な点があるからです。例えば、実験を実行して実験情報ダイアログに値を入力すると、直ちに最初の試行が始まってしまいます。実験者が実験情報ダイアログに入力する場合、入力したらすぐに実験参加者にキーボードを渡さないと最初の試行の刺激に反応できないでしょう。また、実験参加者に対する教示も実験開始前にスクリーン上に表示しておきたいものです。終了時もいきなり Builder のウィンドウに戻るのではなく、何か一言実験参加者へのお礼や指示があると気が利いていて良いでしょう。次の節では、こういった気配りを実験に組み込む方法を解説します。

チェックリスト

- 正答／誤答を記録するように Keyboard コンポーネントを設定することができる。
- キーを押さないことが正答となるように設定することができる。
- 正答となるキー名を条件ファイルから読み込んで繰り返し毎に更新することができる。
- trial-by-trial 記録ファイルから、各試行の正答／誤答を読み取ることができる。

3.9 ルーチンを追加して教示などを表示しよう

この節では、exp03.psyexp の実験開始時と終了時に教示などを表示させます。一口で教示を表示すると言っても「イラストで教示するのか、文章で表示するのかなど」いろいろと選択肢がありますが、ここでは文章で表示することにしましょう。

文章を表示するとなると Text コンポーネントの出番ですが、ここまで解説してきた方法だけではうまく表示することができません。というのも、今まで通りルーチンペイン上に Text コンポーネントを配置してしまうと、繰り返しの度に表示されてしまうので「実験開始時と終了時」ではなくて試行毎の表示になってしまうのです。繰り返しが始まる前や後に何かを表示したいときには、新しいルーチンを追加する必要があります。図 3.26 はフローペインを使って新しいルーチンを作成してフローに挿入する手順を示しています。まず、フ

ローペイン左端の Insert Routine をクリックします。すると (new) と trial という二つの項目があるポップアップメニューが表示されます。「繰り返しを設定しよう」の節で少しだけ触れましたが、PsychoPy で実験を新規作成した時に最初から表示されているルーチンは trial という名前がついています。ポップアップメニューの trial という項目を選択すると、trial ルーチンがもう一つフローに挿入されます。同じルーチンを何個も挿入してどうするのかと思われるかもしれませんが、その例は練習問題で触れることにしましょう。

新しくルーチンを作成してフローに挿入する場合は、(new) を選択します。すると挿入するルーチン名を入力するダイアログが表示されますので、わかりやすい名前を決めて入力しましょう。名前はパラメータ名に使用できる文字列と同様の規則を満たすものでなければいけません。また、すでにパラメータ名やコンポーネントの **[名前]** プロパティで使用している名前とも重複してはいけません。ここでは実験開始前に教示を表示するためのルーチンということで instruction という名前を付けておきます。なお、PsychoPy 2022.1 からはルーチンの名前を決めるダイアログに「テンプレート」という機能が追加されています。これについては「3.11.9: ルーチン新規作成時のテンプレート機能について」をご覧ください。

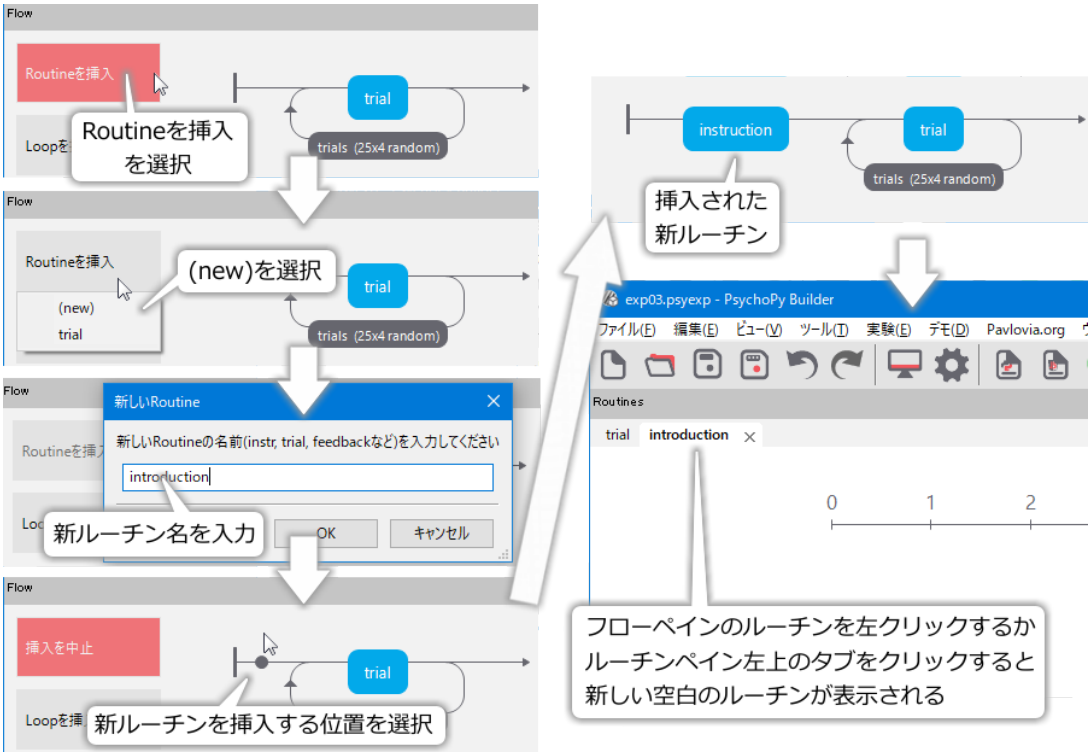


図 3.26　フローペインから新しいルーチンの追加を追加します。

名前を決定すると、新しいルーチンをフローのどの位置に挿入するかを指定しなければいけません。フロー上でマウスカーソルを動かすと、マウスカーソルが挿入可能な位置に重なった時に黒い丸が表示されますので、ループの前を選んで挿入してください。すると 図 3.26 の右上のように instruction と書かれた四角いアイコンがループの前に挿入されます。同時に、ルーチンペインの左上に instruction と書かれたタブが出現しているはずです。フローの instruction と書かれたアイコンか、ルーチンペイン左上の instruction と書かれたタブをクリックすれば、まだコンポーネントが配置されていない空白のルーチンが表示されるはずです。この空白のルーチンが新たに挿入された instruction ルーチンです。

instruction ルーチンの編集を始める前に、操作方法の確認を兼ねてもうひとつ実験終了時のメッセージを表示するためのルーチンを追加しておきましょう。ルーチン名は thanks とします。ここでは三つの点を確認しておいてください (図 3.27)。

- フローペインの左端の Insert Routine を使ってフローにルーチンを挿入する時にポップアップメニューに instruction ルーチンが含まれていること。
- フロー上の挿入済みルーチンの上にマウスカーソルを重ねて右クリックするとポップアップメニューが表示され、とルーチンをフローから削除したり名前を変更したりできること。
- 上記の方法でフローからルーチンを削除しても、該当するルーチンはルーチンペインに残ったままであること。

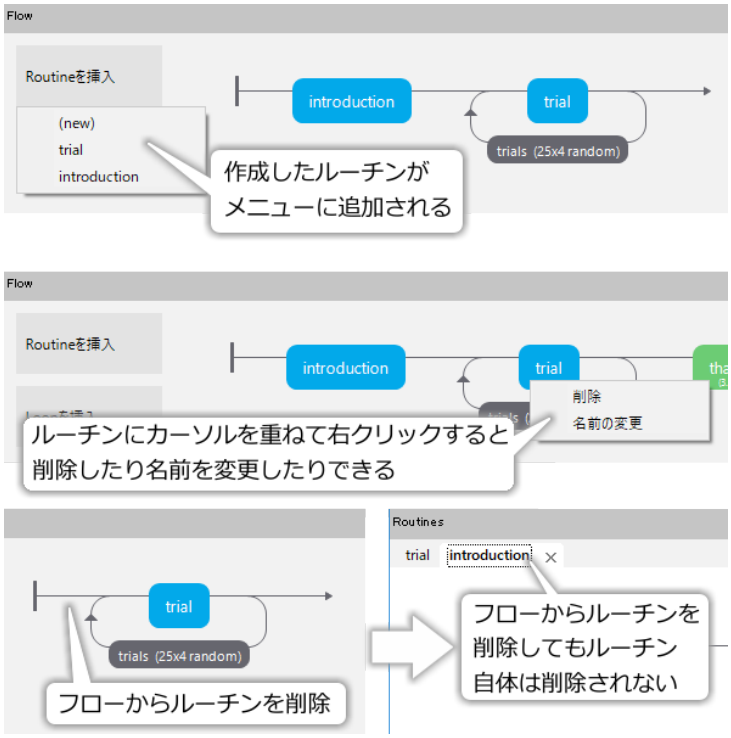


図 3.27　フローペインにおけるさまざまな操作。フローからルーチンを削除してもルーチンそのものは削除されません。

特に、フローからルーチンを削除してもルーチンそのものはルーチンペインに残っているという点は覚えておいてください。バージョン 2020.2.6 現在で Builder はフローにおけるルーチンの並び替えをサポートしていませんので、ルーチンの順番を変えたいときには一旦ルーチンをフローから削除して再挿入する必要があります。その際、一旦フロー上から目的のルーチンがなくなってしまっても全く問題ありません。

さて、instruction と thanks のルーチンを追加してそれぞれループの前と後ろに挿入ら、ルーチンの編集に入りましょう。まず instruction ルーチンでは、画面中央に以下のような教示を画面上に提示することにしましょう。

画面の中央に白い十字が表示されたあと、すこし遅れて赤色または緑色の円が表示されます。 **刺激の色を見て、以下のキーをできるだけ速く間違わずに押してください。** **赤色：左手人差し指でキーボードの x キー** **緑色：右手人差し指でスラッシュ（ / ）キー** **スラッシュキーを押すと課題が始まります。** **100 問終了するまで休憩できないので注意してください。**

実験参加者の中には「スラッシュキー」と言われてもどのキーかわからない方もいるでしょうから、キーの確認も兼ねてスラッシュキーを押して実験を開始することにしてみましたが、スペースキーなどの実際の反応で使用しないキーを割り当てるのもよいでしょう。上記の文章を表示するための Text コンポーネントの設定については、ここまで作業してきた皆さんでしたらヒントなしでできると期待します。

ひとつ注意すべき点は、ルーチン終了時刻の指定です。教示に「スラッシュキーを押すと課題が始まります。」と書いてあるのですから、Keyboard コンポーネントを使用して、スラッシュキーが押されたらルーチンを終了するように設定すればよいでしょう。もちろん Text コンポーネントと Keyboard コンポーネントの **[終了]** は空白にしておきます。ここまではすでに解説済みのテクニックですが、このままだと記録ファイルに「課題を開始する時に押したスラッシュキー」まで保存されてしまいます (ただし 第 7 章 参照)。分析に必要な反応が記録されなくなってしまうわけではないので致命的な問題ではありませんが、分析と無関係なキー押しが記録されていると分析の際に思わぬ手間がかかる恐れがあります。

キー押しを記録したくない場合には、Keyboard コンポーネントのプロパティ設定ダイアログを開いて、 **[記録]** を「なし」に設定しましょう (図 3.28)。その Keyboard コンポーネントで検出されたキー押しは一切記録されません。「その Keyboard コンポーネントで」と断ったとおり、ひとつの実験に複数の Keyboard コンポーネントを配置している場合、個々の Keyboard コンポーネントで独立に **[記録]** の設定を行うことができます。今回の実験の場合、instruction ルーチンに配置する Keyboard コンポーネント **[記録]** を「なし」にして、trial ルーチンに配置する Keyboard コンポーネントは「最後のキー」のままにしておけばよいでしょう。これで instruction ルーチンは完成です。

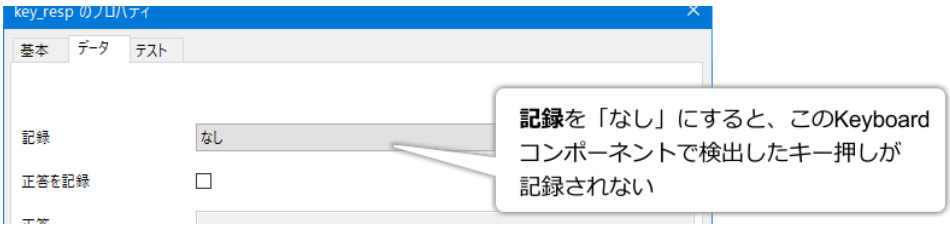


図 3.28　Keyboard コンポーネントで **[記録]** を「なし」に設定すると、その Keyboard コンポーネントで検出したキー押しが記録ファイルに出力されません。

続いて thanks ルーチンですが、ここでは Text コンポーネントを使用して「終了しました。ご協力ありがとうございました。」というメッセージを 3 秒間表示することにしましょう。Text コンポーネント以外のコンポーネントを配置する必要はありません。特に難しい点はないはずですが、ひとつ解説しておきたい点があります。Text コンポーネントの **[終了]** の設定を終えた後にフローペインを見ると、thanks ルーチンだけ緑色で表示されていて、小さく 3.00s と表示されているはずです (表示されていない場合はメニューの「ビュー」から「Flow を大きく」を選択してみてください)。対して、trial や instruction ルーチンは青色で表示されています (図 3.29)。緑色のルーチンは、中に含まれるすべてのコンポーネントの終了時刻が確定していて、ルーチン開始から終了までに要する時間が計算できることを示しています。青いルーチンは、キーが押されるまで終了しないルーチンのように、ルーチン開始から終了までの時間が実行時にならないと確定しないことを示しています。

以上でこの節の目標は達成しましたが、ルーチンの新規作成やフローへの挿入について解説したついでに、Builder ウィンドウ上部のメニューの「実験」という項目について補足しておきます (図 3.30 上)。メニューの「実験」からは、ルーチンの新規作成やコピー＆ペースト、フローへのルーチンとループの挿入の操作ができます。これらの操作のうち、ルーチンの新規作成とフローへのルーチンとループの挿入はフローペイン左端の「Routine を挿入」や「Loop を挿入」を用いる方法とほぼ同じです。ルーチンのコピー＆ペーストは、内容が

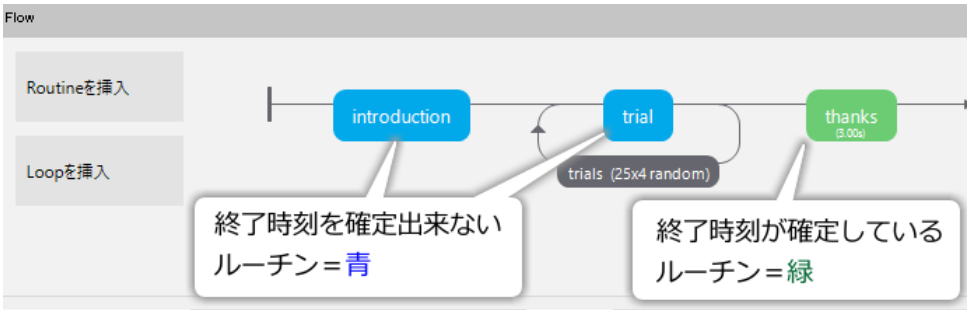


図 3.29　青いルーチンは終了時刻が確定していないことを、緑色のルーチンは終了時刻が確定していることを表しています。緑色のルーチンにはルーチンの実行時間の見積もりが表示されています。

わずかに異なるルーチンを複数作る必要がある時に便利な機能です。

試しにルーチンペインに trial ルーチンを表示している状態で、「実験」メニューから「Routine のコピー」を選択してください。続いてもう一度「実験」メニューを開いて「Routine の貼り付け」を選択しましょう。すると新しいルーチンの名前を入力するダイアログが表示されるので、他のルーチンやコンポーネントなどと名前が重複しないように新しい名前を入力してください。すると、trial と同じコンポーネントが配置された新しいルーチンが作成されます。ただし、同じコンポーネントが配置されていると言っても、各コンポーネントの名前は Builder がコピー元のルーチンに含まれるコンポーネントの名前と重複しないように自動的にアンダーバー + 数字が付けられるので注意してください (図 3.30 下)。

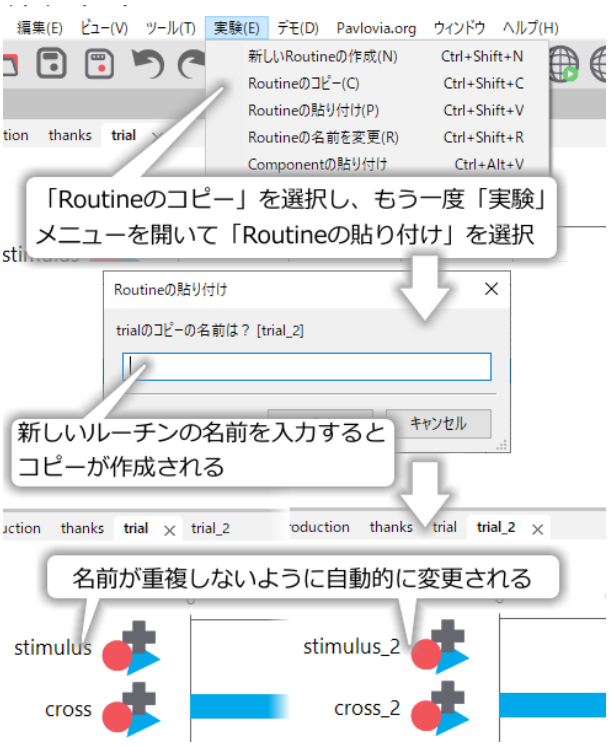


図 3.30　ルーチンのコピー＆ペースト。コンポーネントの名前が重複しないように、ペーストされたルーチン内のコンポーネント名には元のコンポーネント名にアンダーバー + 数字が自動的につけられます。

ルーチンの新規作成、コピー＆ペーストときたら、削除の方法についても知りたいところです。削除はルーチンペイン左上のタブから行います。現在選択中のタブにはルーチン名の右に×マークが表示されています。この×を左クリックすると、そのタブのルーチンが削除されます (図 3.31)。誤って削除してしまった場合はす

ぐにツールバーの元に戻すボタンをクリックしましょう。

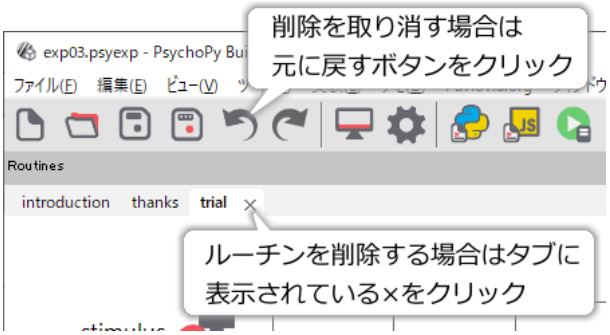


図 3.31　ルーチンを削除するにはタブに表示されている×をクリックします。誤って削除してしまったなどの理由で削除を取り消したい場合は元に戻るボタンを使います。

以上で本章の解説を終えたいとと思います。簡単な実験であれば本章の内容だけで十分に実現できるはずです。締めくくりとして、本章で解説したテクニックを応用する練習問題を出しておきます。本章の内容をマスターできたと思った方は是非挑戦してください。

チェックリスト

- ルーチンを新たに作成してフローに追加することができる。
- 既存のルーチンをフローに追加することができる。
- フローからルーチンを削除することができる。
- フローペインの赤いルーチンと緑のルーチンが何に対応しているか説明することができる。
- 特定の Keyboard コンポーネントが検出したキー押しを記録しないように設定することができる。
- 既存のルーチンと同一の内容を持つ新しいルーチンをコピー＆ペーストの機能を使って作成することができる。

3.10 練習問題：練習試行を追加しよう

先ほどは instruction ルーチンを終えて実験を開始する際に、実験参加者にスラッシュキーを確認させることも兼ねてスラッシュキーで instruction ルーチンを終了するようにしました。しかし、本来であれば本試行と同様の手続きの練習試行を数回体験させてから実験を開始するべきです。練習と本試行を別々の psyexp ファイルとして作成するのも良いのですが、ここでは練習問題ということで単一の psyexp ファイルで練習試行と本試行を実施するように改造してみましょう。

図 3.32 はこの練習問題で作成する実験の流れです。本試行と終了メッセージは本章で作成した実験から変更する必要はありません。教示はもうひとつルーチンを追加する必要があるでしょう。この練習問題のポイントは、いかに手間をかけずに練習試行を作成するかです。ヒントは「フローの中に同一のルーチンを複数回挿入できる」という点と、「練習試行と本試行はループ回数が違うだけで条件は同じ」という点の二点です。これでもまだピンと来なければ、 図 3.32 の一番下のフローも参考にしてください。

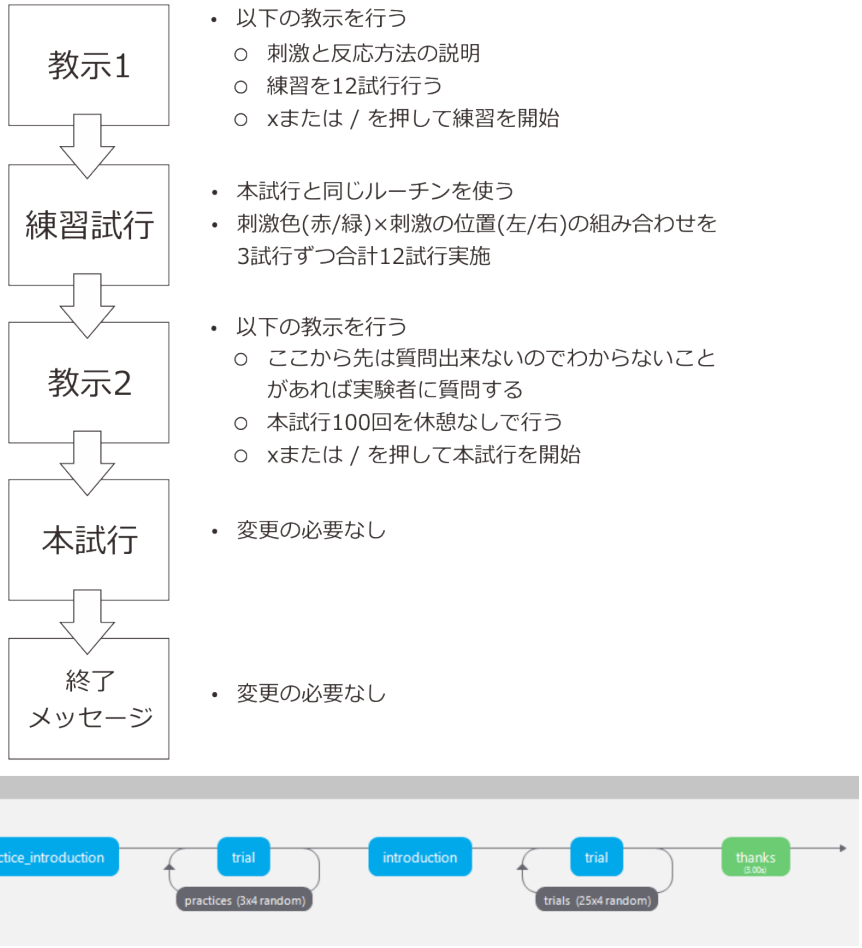


図 3.32　練習問題。本試行を開始する前に 12 試行の練習を行うように改造してみましょう。一番下は作成例のフローを示しています。

3.11 この章のトピックス

3.11.1 自分のキーボードで使えるキー名を確かめる

Builder を起動し、ルーチンペインに Keyboard コンポーネントと Text コンポーネントを配置して以下のように設定してください。

- Keyboard コンポーネント
  - **[名前]** を test_keyboard とする。
  - **[終了]** を空白にする。
  - **[Routine を終了]** のチェックを外す。
  - **[検出するキー $]** を空白にする。
- Text コンポーネント
  - **[終了]** を空白にする。

  - **[文字列]** に$test_keyboard.keys と入力する。$記号やピリオドには重要な意味があるので忘れずにつける。間に空白文字を含んではいけない (第 4 章 、第 6 章 参照)。
  - **[文字列]** の横にある「更新しない」と書かれたプルダウンメニューを開いて「フレーム毎に更新する」を選択する。
  - 一度実行してみて、字が小さすぎたなら **[文字の高さ $]** を調節する。

- 実験設定ダイアログ
  - Full-screen Window のチェックを外す。このチェックを外しておかないと ESC による強制終了が効かなかった時に強制終了させるのが面倒なので忘れずにチェックを外しておく。

以上の設定を行ったら実験を保存して実行し、適当にキーボードのキーを押してみてください。押したキーに応じてスクリーン中央に文字列が表示されるはずです。これが押したキーに対応するキー名です。通常は ESC キーを押すと強制終了できますが、この実験では画面上に escape と ESC キーのキー名が表示されて実験が終了しない場合があります。その場合はあわてずに Runner のウィンドウを選んで実験中断ボタンを押してください。実験が終了します。

3.11.2 Builder で使用できない名前を判別する

Builder を起動し、適当なコンポーネントをルーチンに配置して、 **[名前]** に名前として使用したい文字列を入力してみましょう。文字列が Python の予約語や PsychoPy のモジュール名、Builder の予約語に一致する場合は、 図 3.33 のようにプロパティ設定ダイアログの下部に赤色でエラーメッセージが表示されます。ダイアログ左下の OK ボタンをクリックできなくなりますので、意図せずに予約語を名前に使用してしまった場合でもまず気づくでしょう。

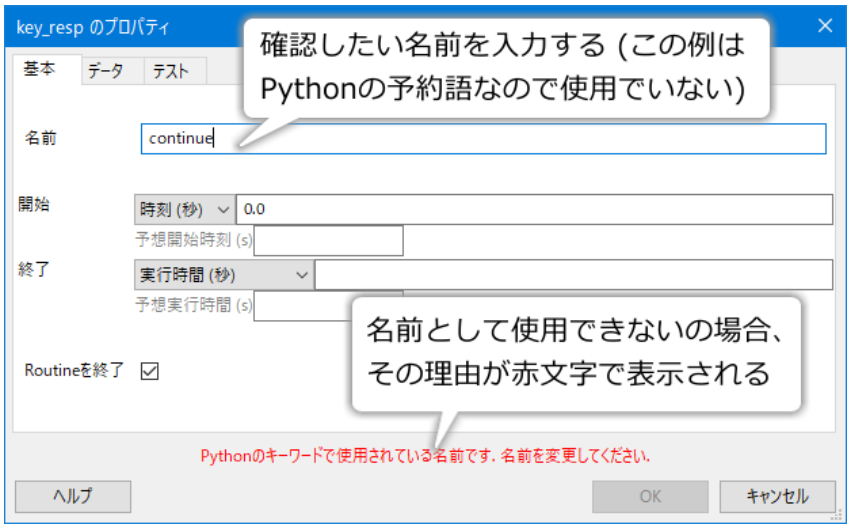


図 3.33　Builder で適当なコンポーネントをルーチンに配置して **[名前]** に文字列を入力することによって、その文字列が予約されていないかを確認することができます。

この方法でチェックできるのは予約語などと一致する場合と、自分で定義した他のコンポーネントの **[名前]** や条件ファイルのパラメータ名などと一致している場合です。ただし、 第 4 章 で紹介する、読み込む条件ファイルを実行時に切り替える方法を用いている場合は、Builder は名前の重複を判断できません。実験作成者が管理する必要があります。

3.11.3 無作為化と疑似乱数

予測不可能な順序に並べられた数の列のことを乱数 (または乱数列) と呼びます。乱数には、一様乱数や正規乱数など、値の出現確率の違いによってさまざまな種類があります。コンピュータゲームなどにおいて人間にとって予測困難な動作を実現する際には、乱数が非常によく用いられます。例えば「地図上を歩き回っていると予測困難なタイミングで敵と遭遇する」というゲームの場合、「地図上を一歩歩くごとに乱数の先頭から順番に値をひとつ取り出し、その値を 20 で割った余りが 0 であれば遭遇する」といった処理を行うと上手くいきます。10 で割ると遭遇しやすくなり、100 で割るとめったに遭遇しないといった具合に遭遇頻度の調節もできます。

このように便利な乱数ですが、実は完全な乱数を発生させるのは難しいことで、代用のために「乱数っぽく見える」数列を作成する方法がいろいろと考案されてきました。「乱数っぽく見えるけれども、実は確定的な計算によって生成されている数列」のことを疑似乱数と呼びます。疑似乱数はシード (seed) と呼ばれるパラメータを持っており、シードの値が一定であれば毎回同じ数列が得られます。あまり良い例ではないかも知れませんが、関数 y=sin kx (k は定数) において、k の値が一定であれば x=1,x=2,x=3,…の時の y の値は何度計算しても同じです。しかし、k の値を変えれば x=1,x=2,x=3,…の時の y の値は変化します。y=sin kx 程度の式であれば生成される数列に規則性があることはすぐにわかりますが、工夫して疑似的に乱数としての性質を持つ数列を生成するように考えられた式が疑似乱数というわけです。

Builder では、**[乱数のシード $]** が空白であれば、実験実行時に Builder が適当なシードを決定して疑似乱数を発生させます。 **[乱数のシード $]** に整数が与えられていれば、それをシードとして疑似乱数を発生させます。そして、生成した疑似乱数を用いて条件の実行順を並び替えます。ですから **[乱数のシード $]** を指定すれば必ず同じ順番で繰り返しが行われるのです。

3.11.4 Loop のプロパティ設定ダイアログの [使用する行 $] について

この機能を使いこなすには 第 5 章 以降で学ぶ知識が必要なので、Python の文法をご存じない方はひとまずこの機能は無視して先へ進むことをお勧めします。

[使用する行 $] は、条件ファイルで定義されている条件のうち、一部分だけを利用して実行したい場合に利用します。1 行に 1 条件が定義されているので、使用したい条件が書かれている行番号をカンマ区切りなどで列挙します。注意が必要なのは、Python は順番を表現するときは最初を「0 番目」と書く点です (第 8 章)。

具体的な使用方法ですが、例えば今回の実験で条件ファイルの 1 行目と 3 行目に書かれた条件のみを実施する場合には

```
"0,2"
```

と書きます。" "を忘れないでください。他にもリストやタプル (第 5 章 、第 12 章) を用いて

```
(0,2)
[0,2]
```

という書き方も受け付けます。この場合は" "が不要です。連続する多数の行を指定する場合はスライス (第 13 章) を使用します。

```
[:20]
```

筆者は普段この機能を利用しないのですが、第 5 章 で学ぶ「実験ダイアログからの情報の取得」を利用して、実行時に利用する行を指定するといった使い方が便利そうです。また、数か月、数年後に実験を再確認する必要が出てきた場合に備えて、条件ファイルにコメントを書いておいて、この機能を用いてコメント行を読み飛ばすといった使い方もできるでしょう。

3.11.5 $を含む文字列を提示する

本文中で、Text コンポーネントの **[文字列]** プロパティに$を入力すると条件ファイル内で定義されたプロパティの値を代入することができると述べました。では、Text コンポーネントで「$を含む文字列を表示させたい」場合はどうすればいいのでしょうか。ただ$を書くと Builder に「名前」として判断されてしまうため、目的を達成できません。一番シンプルな解決策は、「半角ではなく全角の$を使う」というものです。コンピュータにとって半角と全角の$は全く別の文字なので、問題なく表示することができます。

どうしても半角の$を使いたい、という場合はエスケープ文字を利用する方法があります。エスケープ文字とは半角のバックスラッシュ (\ ) のことで、日本語フォントでは円記号 (¥) として表示されます。こう書くと初心者の方は「エスケープ文字って何？」と「日本語フォントでは円記号で表示されるってどういう意味？」という二つの疑問を持たれることと思いますが、まずエスケープ文字の説明から始めましょう。

エスケープ文字の役割は、Builder における$の役割と似ています。例えば Python のスクリプトにおいては、Target という文字列がデータとして文字列を表すのか、データに対してつけられた名前 (変数) を指すのかを区別するために、文字列の前後を半角のシングルクォーテーションまたはダブルクォーテーションで囲みます。

表 3.3　変数と文字列の区別

Target	Target という変数
"Target"	Target という文字列

言語によってダブルクォーテーションのみしか使えないなどの違いがありますが、このような表記は多くのプログラミング言語で用いられています。さて、Python の場合、シングルクォーテーションやダブルクォーテーション自体を含む文字列、例えば

> He said, "Please respond as quickly and precisely as possible."

という文字列はどう表記すればよいのでしょうか。何も考えずに前後にダブルクォーテーションを付けてしまうと

> **"He said, "** Please respond as quickly and precisely as possible. **""**

となってしまい、"He said, "までがダブルクォーテーションで囲まれていて、Please から possible までは囲まれてないことになってしまいます。最後に""が残りますが、これは「何も文字がない文字列」と解釈されます。数字の 0 のようなもので、空文字列と呼ばれます。

この問題の回避方法として、「文字列中に含まれるダブルクォーテーションの前には\を付ける」という記法が用意されています。これを使うと、先の文字列は

"He said, \" Please respond as quickly and precisely as possible. \" "

と表記することができます。多くのプログラミング言語において、\には「その直後に続く文字 (列) を通常とは異なる方法で解釈させる」という役割があります。\などの特別な文字によって、後続の文字を通常とは異なる方法で解釈させることをエスケープと呼び、エスケープによって異なる意味を示す文字列をエスケープシーケンスと呼びます。エスケープシーケンスを開始する記号 (ここでは\) はエスケープシーケンスプレフィックスやエスケープ文字などと呼びます。He said,…の例文では、\"と書くことによって\に続く " が「文字列の終了を表す」という通常の意味を失って、単なる " という文字だと解釈されています。もし皆さんが将来プログラミング言語を本格的に学習するのであれば、他にもさまざまなエスケープの例に出会うことになるでしょう。なお、Python に限って言えば、文字列をシングルクォーテーションで囲んでも良いことと、シングルクォーテーションで囲まれた文字列の中ではダブルクォーテーションを自由に使うことができるということも付け加えておきたいと思います。

以上を踏まえて Builder の Text コンポーネントの話に戻りましょう。Text コンポーネントで$を含む文字列を表示したい場合にも、\によるエスケープが利用できます。つまり、\$と書けば$を表示することができます。ぜひ試してみてください。なお、本文中で述べた「Builder は条件ファイル内に書かれた$記号を単なる文字として解釈する」というルールはここでも有効ですので、Text コンポーネントの **[文字列]** プロパティの値を条件ファイルから読み込んで表示する場合にはエスケープをする必要はありません。$を含む文字列をそのまま表示することができます。

最後に蛇足ですが、「日本語フォントでは円記号で表示されるってどういう意味？」という疑問にもお答えしておきましょう。コンピュータでは、文字を管理するためにすべての文字に数値を割り当てています。例えばアルファベットの大文字の A は 0x41、B は 0x42、小文字の a は 0x61 が割り当てられています。この数値を文字コードと呼びます。困ったことに、現在のコンピュータでは歴史的な経緯から複数の文字コードが使用されており、ある文字コードで書かれた文書を別の文字コードで解釈すると、本来意図されていた文字とは全く別の文字として解釈されてしまいます。昔の電子メールで時々生じていた「文字化け」の原因のひとつが送信側と受信側で使用している文字コードの違いでした。今回問題になっているバックスラッシュ (\ ) は、初期のコンピュータからずっと利用され続けている ASCII コードという文字コードで 0x5C に割り当てられています。一方、日本のコンピュータで最初に用いられた JIS 規格の文字コードでは、大半が ASCII と共通のコードが割り当てられていたのですが 0x5C には円記号 (￥) が割り当てられていました。結果として、同じ 0x5C というコードの文字が、解釈に用いるコードによってバックスラッシュになったり円記号になったりするようになってしまったのです。現在のコンピュータでは、欧文フォントを用いて表示するとバックスラッシュで表示され、日本語フォントを用いて表示しなおすと円記号になるといった具合に使用するフォントによって表示が切り替わるという現象が見られます。

3.11.6 PsychoPy の時間計測の精度について (上級)

本文中でも述べた通り、PsychoPy の記録ファイルを確認すると、キーが押された時刻が 0.766847908126 秒のように小数点以下かなりの桁数が出力されています。この数値は一体何桁目まで信頼できるのでしょうか。

残念ながらこの疑問の答えは PsychoPy を実行しているハードウェアや OS、デバイスドライバに依存するので一概には答えられません。筆者が主にプログラムの開発等に使用している PC では、約 2.6MHz のタイマーカウンターが使用されているようです。分解能は約 260 万分の 1 秒ですから 3.8×10^{-7} 、0.38 マイクロ秒です。Web 上でいろいろな資料を確認する限り、この周波数はあまり高くない部類で、10MHz を超えるタイマーカウンターが利用できる場合もあるようです。反応時間を指標とした心理学実験の論文では「ミリ秒」の

小数点 1 桁まで報告されることがしばしばありますが、その用途には十分な分解能があると言えます。

むしろ注意しないといけないのは、実験参加者がキー押しの動作をはじめてから実際に「キーが押された」と PsychoPy のプログラムまで到達する時間や、PsychoPy が「刺激を画面に描画せよ」という命令を実行してから、実際にそれがモニターに伝わって画面に表示されるまでの時間です。これらの値もやはりハードウェアや OS、デバイスドライバに依存するので一概に言えないのですが、「ゲーム用」などと謳われている高性能なキーボードやモニターでなければ、数十ミリ秒もの時間を要することも珍しくありません。

- 先行研究の結果と比較して平均反応時間が十数ミリ秒にわたってずれている場合は、機材が原因である可能性も考慮する。可能であれば先行研究と同一条件の追試を行っておき、そこで大きなずれがないか確認しておくことが望ましい。
- 複数の実験参加者を呼んで並行して実験を行う場合、実験に使用するハードウェアやソフトウェアは可能な限り統一する。

といった点を守っておけば、多くの実験において時間計測の精度が大きな問題となることはないと思います。

3.11.7 コンポーネントの開始・終了時刻をデータファイルに出力しないようにする

刺激やキーボードといったコンポーネントの開始・終了時刻をデータファイルに出力する機能は、何か実験中に問題が生じた場合に、どの時点で生じたのかを知る重要な手がかりを与えてくれます。しかし、ルーチン内に配置されているがコンポーネントが多い場合、コンポーネント数に応じてデータファイルの列数が増えてしまうため、非常にデータファイルが見難くなる場合があります。

開始・終了時刻の出力を抑制するには、各コンポーネントのプロパティ設定ダイアログの「データ」タブにある **[開始・終了時刻の保存]** のチェックを外します (図 3.34)。チェックを外したコンポーネントだけが出力されなくなりますので、実験の流れを把握する上で重要なコンポーネントだけチェックを残して他はチェックを外すとよいでしょう。

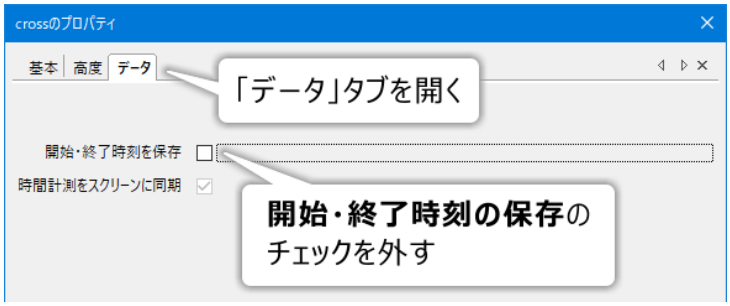


図 3.34 **[開始・終了時刻の保存]** のチェックを外すとそのコンポーネントの開始・終了時刻が出力されなくなります

3.11.8 「CSV 形式のデータを保存 (summaries)」・「xlsx 形式のデータを保存」で作成される記録ファイルの形式

本文で述べた通り、これらの記録ファイルでは条件ファイルで記述された条件ひとつにつき 1 行に結果が要約されています。最初にパラメータが出力され、続いて繰り返し数が出力されていることも本文で述べました。ここではさらにその右側に数百列にわたって出力されている内容について解説します。

これらの情報は、実験内で用いられているコンポーネント毎にまとまっています。本章の実験では (教示などの追加前であれば)Polygon コンポーネントが 2 つ (cross, stimulus) と Keyboard コンポーネント 1 つ (key_resp_2) が使用されているので、これらのコンポーネント毎にまとまっているはずです。出力される情報はコンポーネントにより異なります。表 表 3.4 に Polygon コンポーネントが出力する情報を示します。

表 3.4　Polygon コンポーネントの出力 (**[名前]** は cross とする)

列名	概要
cross.started_mean	刺激の開始時刻の平均値を計算したもの。単位は秒。
cross.started_raw	刺激の開始時刻を試行順に並べたもの。単位は秒。列数は繰り返し数に一致し、1 列目にのみ列名が付く。
cross.started_std	刺激の開始時刻の標準偏差を計算したもの。単位は秒。
cross.stopped_mean	刺激の終了時刻の平均値を計算したもの。単位は秒。ただし、刺激の終了時刻が指定されていない場合は列自体が出力されない。
cross.stopped_raw	刺激の終了時刻を試行順に並べたもの。単位は秒。列数は繰り返し数に一致し、1 列目にのみ列名が付く。刺激の終了時刻が指定されていない場合は空白となる。

Polygon コンポーネントからは、刺激の開始・終了時刻の raw データ、つまり統計処理などをする前の「生」のデータと、平均値と標準偏差が出力されています。raw データならともかく、平均値や標準偏差にはあまり意味がないので、利用することは少ないと思います。Polygon コンポーネント以外の刺激提示用のコンポーネントはほぼ同様の情報を出力します。

続いて Keyboard コンポーネントが出力する情報を表 表 3.5 に示します。Polygon コンポーネントの出力と似たようなレイアウトですが、最初に正答率と各試行の正誤に関する情報 (**[正答を記録]** を選択した場合のみ)、平均反応時間、各試行の反応時間、反応時間の標準偏差が出力されています。これらの情報が出力されていることに魅力を感じるなら、trial-by-trial 記録ファイルではなく xlsx 形式 (または CSV 形式の summaries) の出力ファイルを使うとよいでしょう。

表 3.5　Keyboard コンポーネントの出力 ([名前] は key_resp_2 とする)

列名	概要
key_resp_2.corr_mean	正答率を計算したもの。 **[正答を記録]** を選択した場合のみ出力される。
key_resp_2.corr_raw	試行順に正答なら 1、誤答なら 0 を並べたもの。列数は繰り返し数に一致する。1 列目にのみ列名が付く。 **[正答を記録]** を選択した場合のみ出力される。
key_resp_2.keys_raw	押されたキーを試行順に並べたもの。列数は繰り返し数に一致する。1 列目にのみ列名が付く。
key_resp_2.rt_mean	キー押し時刻の平均値を計算したもの。単位は秒。
key_resp_2.rt_raw	キー押し時刻を試行順に並べたもの。列数は繰り返し数に一致する。1 列目にのみ列名が付く。単位は秒。
key_resp_2.rt_std	キー押し時刻の標準偏差を計算したもの。単位は秒。
key_resp_2.started_mean	キー押し検出開始時刻の平均値を計算したもの。単位は秒。
key_resp_2.started_raw	キー押し検出開始時刻を試行順に並べたもの。単位は秒。列数は繰り返し数に一致し、1 列目にのみ列名が付く。キー押し検出開始時刻が指定されていない場合は空白となる。
key_resp_2.started_std	キー押し検出開始時刻の標準偏差を計算したもの。単位は秒。
key_resp_2.stopped_mean	キー押し検出終了時刻の平均値を計算したもの。単位は秒。ただし、キー押し検出終了時刻が指定されていない場合は列自体が出力されない。
key_resp_2.stopped_raw	キー押し検出終了時刻を試行順に並べたもの。単位は秒。列数は繰り返し数に一致し、1 列目にのみ列名が付く。キー押し検出終了時刻が指定されていない場合は空白となる。
key_resp_2.stopped_std	キー押し検出終了時刻の標準偏差を計算したもの。単位は秒。ただし、キー押し検出終了時刻が指定されていない場合は列自体が出力されない。

すべてのコンポーネントについて情報が出力された後には order という見出しがついた列があり、その列を 1 行目と数えて繰り返し数と同じだけの列数の出力があります。これは各条件の 1 試行め、2 試行め…が実験全体の中で何試行めであったかを示しています。最初の試行が 0 番目と出力される点にご注意ください。以上で出力は終了です。

コンポーネントひとつにつき開始時刻・終了時刻の情報だけで繰り返し数× 2 ＋ α の列数が出力されるので、繰り返し数が 100 回以上、コンポーネント数が十数個もあるような実験を作ると軽く 1000 列を越えるファイルが出力されてしまいます。これだけ列数が多いと必要な情報を探すのが大変なので、あまり実用的ではないというのが筆者の考えです。ただ、「3.11.7: コンポーネントの開始・終了時刻をデータファイルに出力しないようにする」で述べた方法で開始・終了時刻を保存しないようにすると、開始・終了時刻に関する情報がごっそりなくなって Keyboard コンポーネントの反応の正誤、反応時間の出力と order の情報だけになります。これならかなり実用的になるかと思います。

最後に「xlsx 形式のデータを保存」と「CSV 形式のデータを保存 (summaries)」の使い分けですが、これは分析に使用するアプリケーションが対応できる形式のものを選ぶとよいでしょう。一般論として、CSV 形式のデータはただのテキストファイルなので数多くのアプリケーションで開くことができますが、条件ファイルのパラメータに日本語が含まれていると正常に表示されない場合があります。xlsx 記録ファイルは開くために Excel や LibreOffice Calc 等が必要ですが、日本語を含んでいても正常に表示できます。各自の都合に合わせ

て選んでください。

3.11.9 ルーチン新規作成時のテンプレート機能について

PsychoPy 2022.1 より、ルーチンの新規作成時にあらかじめ「心理学実験にありがちな」画面を作成するためのコンポーネントが配置されたルーチンを作成する「テンプレート機能」が追加されました。 図 3.35 のように、ルーチンの新しい名前を決定するダイアログに **[Routine のテンプレート]** という項目が表示され、新しいルーチンに適用するテンプレートを選ぶことができます。

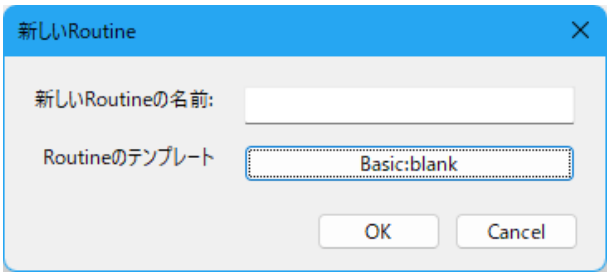


図 3.35　新規ルーチン作成時のテンプレート選択。

便利な機能だとは思いますが、テンプレートを適用した後に手作業で設定しないといけない項目があったり、テンプレートの機能を活かすためにその前のルーチンで適切な前処理をしておかないといけなかったりするなど、Builder の初心者の方がいきなり使いこなせるものではないと思います。そのため、本書ではコンポーネントが何も配置されていない「Basic:blank」というテンプレートを使うという前提で解説していきます。**[Routine のテンプレート]** は初期状態で「Basic:blank」となっているので、変更せずに **[新しい Routine の名前]** だけ入力して OK ボタンをクリックして作業を進めてください。

第 4 章

繰り返し方法を工夫しよう—傾きの対比と同化

4.1 この章の実験の概要

この章では、傾きの対比の実験を題材として、実験を実施する度に条件ファイルを変更したり、繰り返しを多重に用いることによってやや複雑な計画の実験を実現したりする方法を学びます。

さっそく実験の内容を確認しましょう。 図 4.1 に縞模様の刺激が描かれていますが、このように正弦波上に明るさが変化する縞模様の刺激をグレーティングと呼びます。今回の実験では、大小 2 個の円形のグレーティング刺激を実験に使用します。スクリーン中央に 図 4.1 のように二つの刺激の中心が一致するように重ねます。グレーティングの模様がスクリーンの垂直方向にぴったり一致しているのを 0 度の方向として、大きいグレーティング刺激の向きを反時計回り、または時計回りに 20 度傾けます。そして小さいグレーティング刺激の向きを 0 度 ±5 度の範囲で変化させて、小さいグレーティング刺激が 0 度より反時計回り、時計回りのどちらに傾いているかを実験参加者に判断させます。「小さい方のグレーティング刺激」などといちいち書くのは面倒ですので、小さい方のグレーティング刺激を「テスト刺激」、大きい方のグレーティング刺激を「コンテクスト刺激」と呼ぶことにしましょう。コンテクスト刺激がテスト刺激の傾きの知覚にどのような影響を与えるかを調べるのが実験の目的です。

実験の作成を始める前に刺激の大きさや反応方法などの詳細を決める必要がありますが、それ以前にこの刺激は前章までに学んだ知識で作成できそうにありません。PsychoPy Builder にはこの刺激を描くためにぴったりな Grating コンポーネントというコンポーネントが用意されているので、まずはその使い方を学びましょう。

4.2 Grating コンポーネント

Grating コンポーネントは、簡単に言うと縞模様の視覚刺激を描くためのコンポーネントです。視知覚の研究においては基本刺激とでも言うべき重要な刺激ですし、視覚認知の研究でもまた頻繁に用いられます。 **[開始]** や **[終了]** で刺激の有効な時間を指定したり、 **[位置 [x, y] $]** で位置を指定したりするなど、大半のプロパティは Polygon コンポーネントや Text コンポーネントと共通です。Grating コンポーネントが Polygon および Text コンポーネントと大きく異なるのは「テクスチャ」タブです。本節ではこのタブの項目について解説します (**[テクスチャ]** は複雑なので「 4.11.1:Grating コンポーネントの **[テクスチャ]** プロパティについて 」で詳しく取り上げます)。

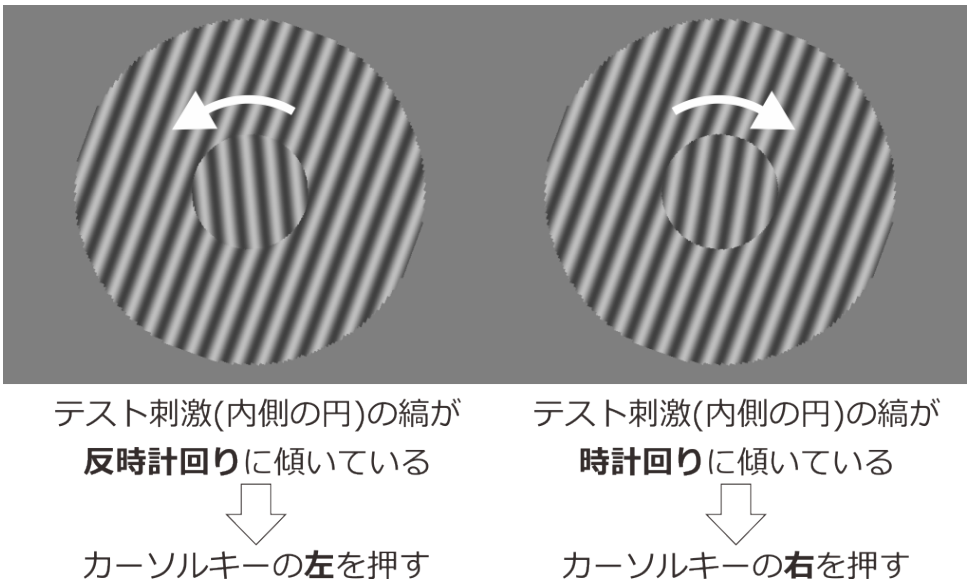


図 4.1　この章の実験。実験参加者は、テスト刺激の縞が垂直方向より反時計回りに傾いているか、時計回りに傾いているかを判断します。テスト刺激の周囲のコンテクスト刺激が判断に与える影響を明らかにするのが目的です。

図 4.2 は **[マスク]** 、 **[位相 (周期に対する比) $]** 、 **[空間周波数 $]** を変化させるとどのような刺激が提示されるかを示したものです。 **[マスク]** は刺激を切り抜くプロパティで、None、circle、gauss を指定することができます。空白にしておくと長方形の刺激が描画され、circle にすると楕円状に切り抜かれた刺激が描画されます。gauss は 2 次元 Gauss 関数の値に従ってコントラストが変調された縞模様を描きます。Gauss 関数と言われてもピンと来ない方は正規分布の密度関数を思い出してください。正規分布の密度関数は Gauss 関数の一種であり、「Gauss 関数の値に従ってコントラストが変調される」とはあの正規分布の密度関数のように中心から周辺に向かってなだらかに縞模様の薄れていくということです。正弦波を正規分布で変調した刺激は Gabor パッチと呼ばれ、視知覚の実験では非常によく用いられます。

続いて **[空間周波数 $]** ですが、この値を大きくすると縞模様の密度が高くなります。より厳密な表現を用いれば、空間周波数 (spatial frequency) とは一定の空間範囲に描かれる模様の繰り返し回数のことです。 **[空間の単位]** が cm と deg の場合には、刺激の幅 1.0 に含まれる縞模様の数に対応します。例えば刺激の幅が 5.0cm で **[空間周波数 $]** が 0.4 ならば 5.0 × 0.4=2.0、つまり刺激には 2 周期分の縞模様が描かれます (図 4.2 中段左)。 **[空間の単位]** が norm と height の場合は、刺激に描かれる縞模様の数に直接対応します。つまり、 **[空間周波数 $]** が 2.0 なら刺激の幅の値がいくらであろうと常に 2 周期分の縞が描かれます。 **[空間の単位]** が pix の場合は複雑で、「 **[空間周波数 $]** の値×刺激の幅」の周期分の縞が描かれます。刺激の幅が 200pix であれば、 **[空間周波数 $]** に 0.01 を指定すれば 200 × 0.01 で 2 周期分の縞になります。

[位相 (周期に対する比) $] は、縞模様の位相を指定するパラメータです。Grating コンポーネントは初期設定では刺激の中心で明るさが最大になるように縞模様が描かれますが、位相を指定すると明るさが最大になる位置をずらすことができます。単位は 1 周期に対する比であり、0.5 ずらすとちょうど明暗が反転します (図 4.2 下段の左端と右端)。

[テクスチャの解像度 $] と **[補間]** の効果を示したのが 図 4.3 です。グレーティングコンポーネントを大きく拡大した図が描かれていますが、まず左側の二つをご覧ください。上が **[テクスチャの解像度 $]** が初期値の 128、下が 512 の時の結果です。512 の時の結果と比べると、128 の時には刺激の境界も縞模様もぼけています。この「ぼけ」は、PsychoPy がグレーティングを描くときには内部で縞模様の画像データ (テクスチャ) を

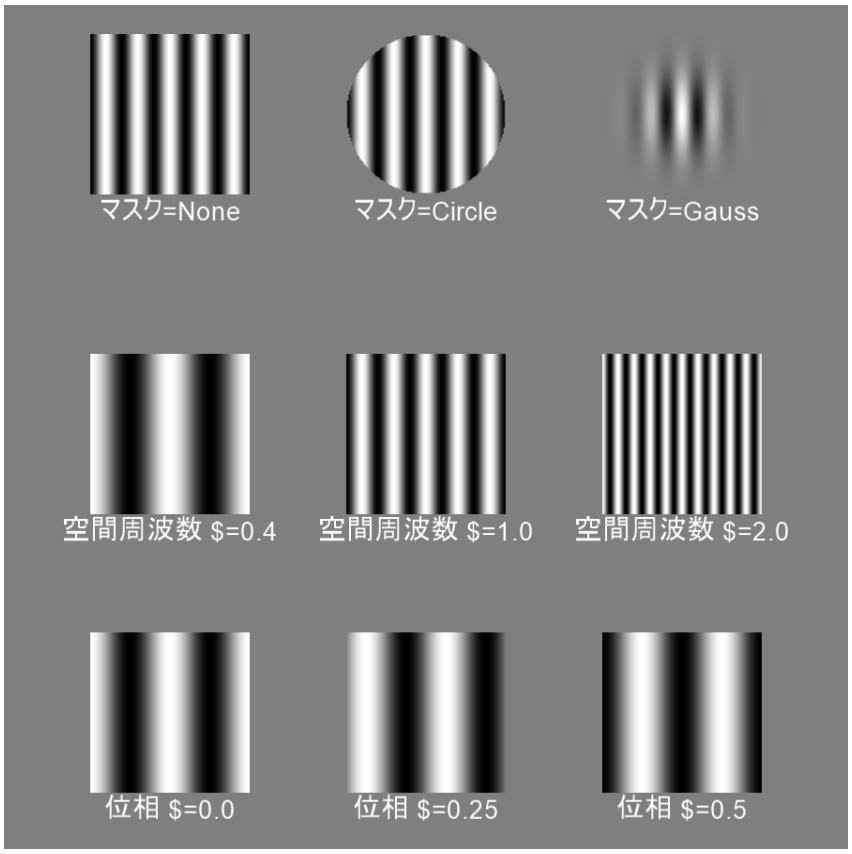


図 4.2　Grating コンポーネントのプロパティのうち、 **[マスク]** (上段)、 **[空間周波数 $]** (中段)、 **[位相 (周期に対する比) $]** を変化させた例。いずれも **[サイズ [w, h] $]** は [5.0,5.0] で、 **[単位]** は cm を指定しています。

作成し、それを **[サイズ [w, h] $]** で指定された大きさに拡大するために生じます。 **[テクスチャの解像度 $]** はテクスチャの解像度を指定するプロパティで、128 であれば 128 × 128 ピクセル、512 であれば 512 × 512 ピクセルのテクスチャを生成します。値が高い方が大きく拡大した時のぼけが小さく済みますが、その代わりに PC のグラフィック描画機能に負担をかけます。小さなグレーティング刺激を多数描画する場合は、描画の負担を小さくするために **[テクスチャの解像度 $]** を低く設定するべきです。逆に今回の実験のように大きなグレーティング刺激をせいぜい 2、3 個描画する場合は高い値を設定するとよいでしょう。描画負担は使用している PC のグラフィック性能に大きく依存しますので、グラフィック性能が高い PC であれば 512 に設定して多数のグレーティングを描いても問題は生じません。

[補間] は、グレーティングを拡大した時の補間方法を指定します。補間方法と言われてもピンと来ないかもしれませんが、 図 4.3 右の二つの拡大図を比べてください。上は刺激の境界がぼやけていますが、下は境界がくっきりしていてカクカクしています。上が **[補間]** に「一次」を指定した例で、拡大した時に足りない色情報を周囲のピクセルと滑らかにつながるように補います。結果として、このように大きく拡大した場合はぼけが目立ってしまいます。一方「最近傍」を指定した場合は、元のテクスチャで最も近いピクセルの色を使用しますので、ぼけが生じない代わりにカクカクになってしまいます。滑らかにする方法は使用する PC のグラフィック機能によって異なりますので、自分が使用する PC でどちらの方がよい出力が得られるか各自で確認してください。

なお、「外観」タブの **[前景色]** はすでに Text コンポーネントで解説しましたが、Grating コンポーネントの場合は縞模様を描画しますので少々複雑です。Grating コンポーネントの色は、 **[前景色]** の値を **[テクスチャ**

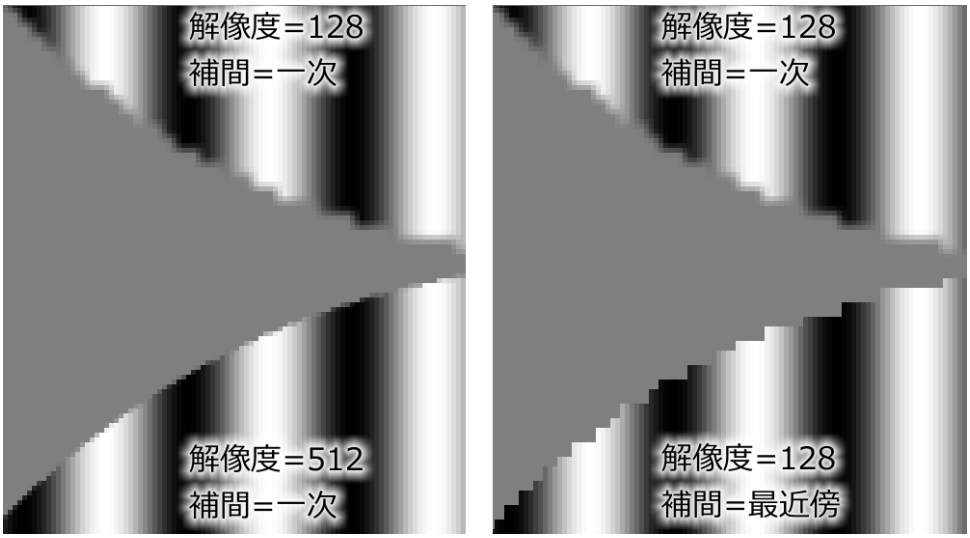


図 4.3　Grating コンポーネントの **[テクスチャの解像度 $]** および **[補間]** オプションの効果。

] で指定された波形に掛け算することによって決まります。つまり、 **[テクスチャ]** が sin で **[前景色]** の値が $[1,1,1] であれば-1 から +1 まで変化します。 **[前景色]** が$[0.5,0.5,0.5] であれば-0.5 から +0.5 まで変化します。$[-0.5,-0.5,-0.5] であれば、$[0.5,0.5,0.5] のときと同様に-0.5 から +0.5 まで変化しますが、負の値を掛け算していますので明暗が$[0.5,0.5,0.5] の時と反転します。 $[1,0,0] のように RGB 各成分の値が異なる場合も、それぞれの成分に波形が掛け算されます。$[1,0,0] の場合は$[-1,0,0] から$[1,0,0] まで変化するということです。 **[前景色]** に red や green といった色名が指定され場合は、PsychoPy の内部でこれらの色名を RGB に変換したうえで波形との掛け算が行われます。

ずいぶん脱線が長くなってしまいました。以上の解説を踏まえて、実験手続きの詳細を決定して実験を作成しましょう。

チェックリスト

- Grating コンポーネントを用いて長方形、または楕円形に縞模様が描かれた刺激を提示することができる。
- Grating コンポーネントで **[空間の単位]** が cm、deg、pix、norm、height のいずれの場合においても、「幅 x に対して y 周期分の縞模様」を描けるように **[空間周波数 $]** の値を決定できる (x、y は正の数値)。
- Grating コンポーネントで描かれる縞模様を初期設定の状態からずらして描画することができる。
- Grating コンポーネントで描画処理の負担を軽くするためにテクスチャの解像度を下げることができる。
- Grating コンポーネントを大きく表示するときに画質を高めるためにテクスチャの解像度を上げることができる。
- Grating コンポーネントの色を指定したときに、何色の縞模様が描かれるのかをこたえることができる。

4.3 パラメータを決定しよう

それでは実験に用いる刺激のパラメータを決定しましょう。まず、テスト刺激とコンテクスト刺激の大きさと縞模様を 図 4.4 のように設定することにします。 コンテクスト刺激の**[サイズ [w, h] $]** と **[空間周波数 $]** はそれぞれテスト刺激の 3 倍の値が設定されていて、同じ幅の縞が描かれるようにしてあります。いずれの刺激も **[マスク]** に circle を指定して円形にします。 **[前景色]** はいずれも$[0.5, 0.5, 0.5] にしておきましょう。

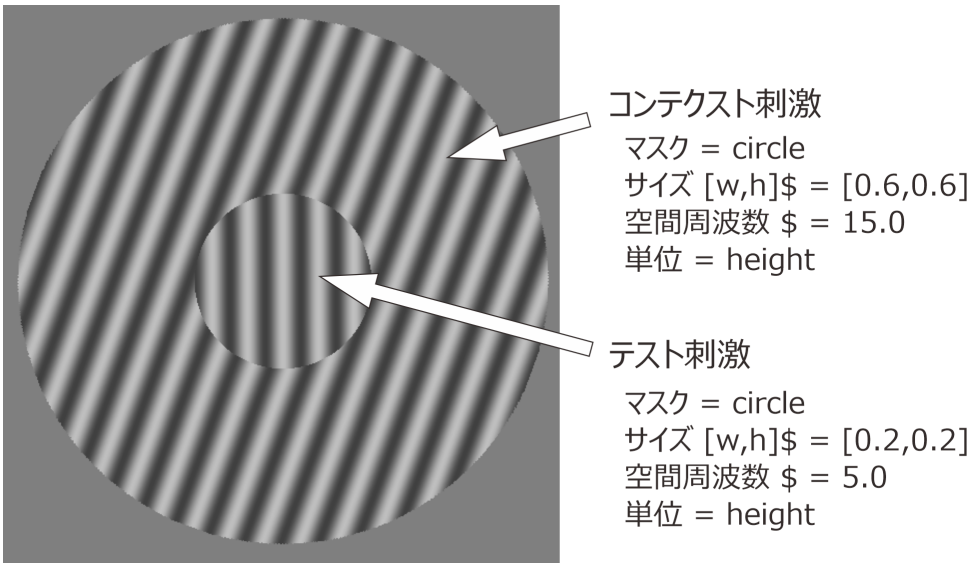


図 4.4　刺激のパラメータ

今回の実験で一番重要なパラメータはテスト刺激とコンテクスト刺激の方向です。すでに概要で述べたとおり、コンテクスト刺激には反時計回りに 20 度 (**[回転角度 $]** =20) 傾いたものと、時計回りに 20 度 **[回転角度 $]** =-20 傾いたものの 2 種類を用います。それぞれのコンテクスト刺激に対して、-5 度から 1 度間隔で 5 度までの計 11 種類のテスト刺激を組み合わせることにしましょう。それぞれの組み合わせに対して 5 回ずつ、無作為な順序に実験参加者に提示して、判断させることにします。試行数はコンテクスト刺激 2 種類×テスト刺激 11 種類×繰り返し 5 回=110 試行です。

続いて 図 4.5 に実験の手続きを示します。まず実験が始まったら、反応方法の教示をスクリーンに提示します。反応方法は、テスト刺激が時計回りに傾いているように見えたらカーソルキーの右、反時計回りに傾いているように見えたらカーソルキーの左を押すものとします。教示画面は各自で自由に作成していただいてよいと思いますが、刺激の例を示しながら反応するキーを示すとよいと思います。

教示画面でカーソルキーの左右いずれかを押すと実験が始まります。各試行はまず 0.5 秒間の空白のスクリーンから始まり、続けて刺激を提示します。実験参加者がカーソルキーの左右いずれかを押して反応するまで刺激を提示し続け、キーが押されたら直ちに次の試行に進みます。全試行終了すれば実験は終了です。 図 4.5 では示していませんが、最後に「実験は終了しました」などのメッセージを表示するのも良いでしょう。

以上が手続きです。いよいよ実験を作成しますが、第 3 章 の実験と比べて使用するコンポーネントが多いので便利なテクニックを紹介しながら作業を進めましょう。

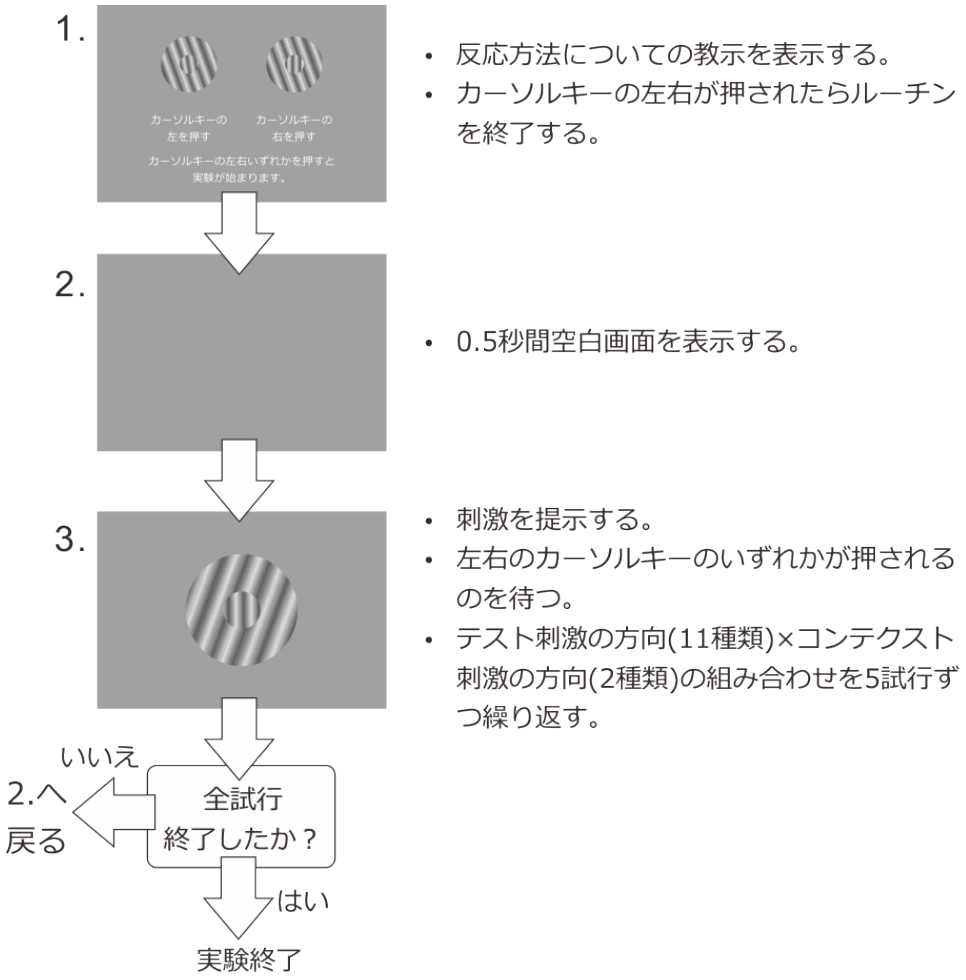


図 4.5　実験の手続き

4.4 コンポーネントのコピーを活用して実験を作成しよう

では、Builder を開いて作業を開始します。実験は exp04a.psyexp というファイル名で保存するものとします。まず trial ルーチンで以下の作業をしてください。

- trial **ルーチン**
  - Grating **コンポーネントを 1 つ配置し、以下の設定をする。**
    - 「基本」タブの **[名前]** を testStim とし、**[開始]** を「時刻 (秒)」で 0.5 に、 **[終了]** を空白にする
    - 「外観」タブの **[前景色]** を$[0.5,0.5,0.5] にする。
    - 「テクスチャ」タブの **[マスク]** を circle に、**[空間周波数 $]** を 5.0 にする。 **[テクスチャの解像度 $]** を 512 にする。
    - 「レイアウト」タブの **[サイズ [w, h] $]** を [0.2,0.2] にする。 **[回転角度 $]** を testDir にして、「繰り返し毎に更新」にする。

これでテスト刺激が完成しました。続いてコンテキスト刺激を作成しなければいけませんが、**[名前]** と **[サイ**

ズ [w, h] $]、**[回転角度 $]** 以外はテスト刺激と設定が同じです。このような場合には、コンポーネントのコピー機能を使用すると作業が楽になります。

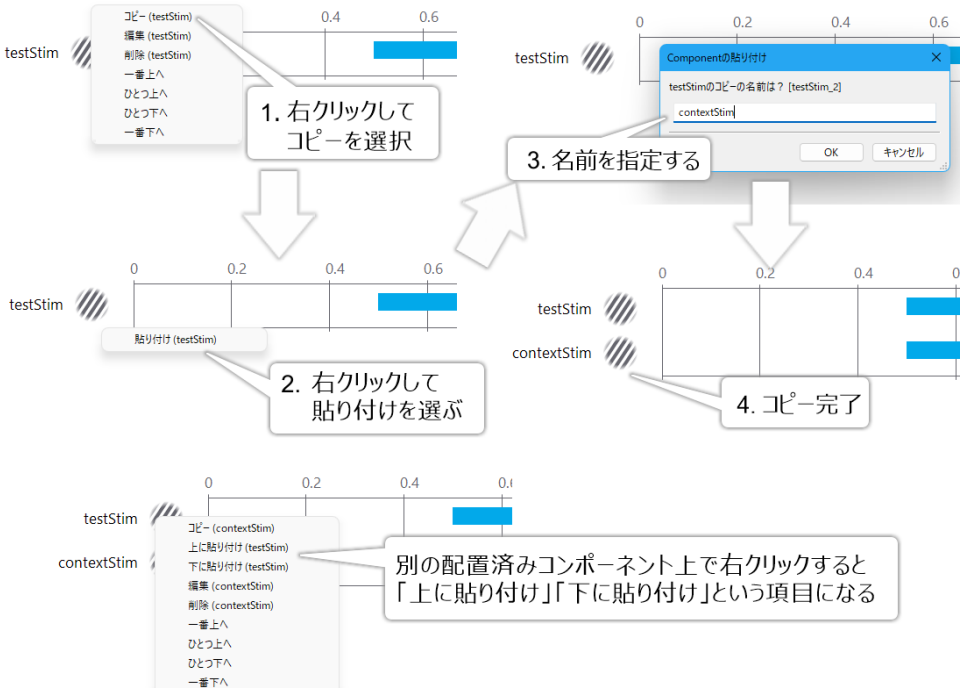


図 4.6　コンポーネントのコピー

図 4.6 にコピーの手順を示します。まず図の左上のように、コピーしたいコンポーネントにマウスカーソルを重ねて右クリックし、メニューから「コピー」を選択します。その後、コピーを作成したいルーチンを表示します。今は trial ルーチンに作成したいので、そのままで結構です。ルーチンの余白部分でマウスを右クリックすると、「貼り付け (testStim)」というメニューが表示されます (図の中段左)。選択すると図の右上のようにコンポーネントの名前をたずねるダイアログが表示されます。Builder では同一の名前を持つコンポーネントを複数作ることができませんので、新しい名前をつけなければいけません。名前を空欄のまま OK をクリックすると testStim_2 のように元のコンポーネントの名前の後ろに数字を添えた名前となりますが、ここは contextStim という名前にしておきましょう。新しい名前を入力して OK をクリックすれば、図の中段右のようにコンポーネントのコピーが完了します。

なお、「貼り付け」のメニューを表示するために右クリックをする際に、配置済みの他のコンポーネント上で右クリックすると、図の下段のように削除や順番変更といった項目の中に「上に貼り付け ()」「下に貼り付け ()」という項目が表示されます (括弧内はコピー中のコンポーネント名)。これを利用すると、たくさんのコンポーネントを配置しているルーチンで、他のコンポーネントの順番を考慮して狙った位置に貼り付けることができます。複雑な実験を作るときにはとても便利なテクニックなので覚えておきましょう。

コンポーネントのコピーが終了したら、contextStim のパラメータを変更します。

- trial ルーチン
  - contextStim の **[サイズ [w, h] $]** を [0.6,0.6] に、**[空間周波数 $]** を 15.0 にする。**[回転角度 $]** を contextDir にする。
  - contextStim の上に testStim が表示されるようにコンポーネントの順番を並び替える。

これで刺激は完成です。あとは参加者の反応を計測するために Keyboard コンポーネントを置いておきましょう。

- trial ルーチン
  - Keyboard コンポーネントをひとつ配置し、「基本」タブの **[開始]** を「時刻 (秒)」で 0.5 に、 **[終了]** を空白にする。「データ」タブの **[検出するキー $]** を'left', 'right' にする。 **[正答を記録]** をチェックして、 **[正答]** に$correctAns と入力する。

続いて実験の開始時に表示する教示画面を作成します。 図 4.1 のように時計回り、反時計回りの刺激の実例を表示するとわかりやすいでしょう。 図 4.1 のような画面を作成するとなると、Grating コンポーネントを 4 個配置しなければいけません。たった今覚えたコンポーネントのコピーを使えば楽ができますが、前章で紹介したルーチンのコピーを使うとさらに手順を省略できます。 図 4.7 のように trial ルーチンを instruction という名前でコピーしましょう。

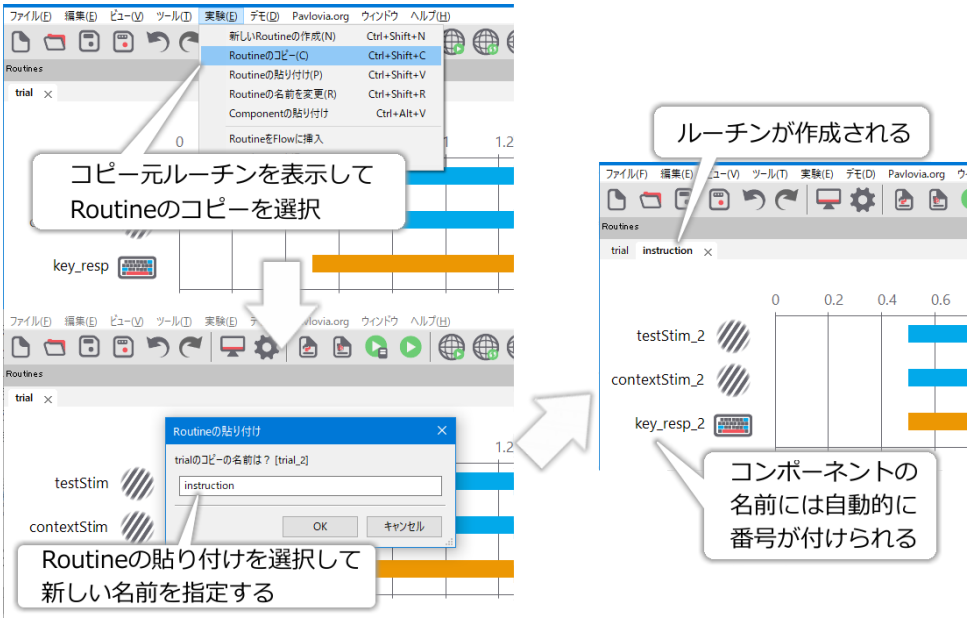


図 4.7　ルーチンのコピー。

コピー元の trials ルーチンと同じコンポーネントが含まれていること、コンポーネントの名前には (名前が重複しないように) 番号が自動的につけられていることを確認してください。これでかなり作業を楽することができました。後は instruction ルーチンで以下のように作業してください。

- instruction ルーチン
  - **フローの先頭に挿入する** （忘れがちなので注意！)。
  - contextStim_2 の **[名前]** を left_large とし、 **[開始]** を「時刻 (秒)」で 0.0 にする。 **[位置 [x, y] $]** を [-0.4, 0.0] とする。 **[回転角度 $]** を 20.0 にし、「更新しない」にする。
  - testStim_2 の **[名前]** を left_small とし、 **[位置 [x, y] $]** を [-0.4, 0.0] とする。 **[回転角度 $]** を-10.0 にし、「更新しない」にする。 **[開始]** を「時刻 (秒)」で 0.0 にする。
  - left_large をコピーして **[名前]** を right_large にする。 **[位置 [x, y] $]** を [0.4, 0.0] とする。

  - left_small をコピーして **[名前]** を right_small にする。 **[位置 [x, y] $]** を [0.4, 0.0]、 **[回転角度 $]** を 10.0 にする。
  - left_large の上に left_small、right_large の上に right_small が描かれるように各コンポーネントが並んでいることを確認する。

これで刺激は完成です。さらにテキストを追加しましょう。

- instruction ルーチン
  - **Text コンポーネントを 1 個配置して、 [名前] を TextInst とする。**
    - **[文字列]** に「カーソルキーの左右いずれかを押すと始まります」等のメッセージを入力する。画面の大きさに応じて文を短くするなり改行するなりするとよい。
    - **[文字の高さ $]** を 0.04 にする (各自の PC 画面の大きさに応じて調節すると良い)。
    - **[位置 [x, y] $]** に [0, -0.4] と入力する。
  - **TextInst をコピーして textCC の名前で貼り付ける。**
    - **[位置 [x, y] $]** に [-0.4, -0.35]、 **[文字列]** に「反時計回り」と入力する。
  - **TextInst をコピーして textC の名前で貼り付ける。**
    - TextC の **[位置 [x, y] $]** に [0.4, -0.35]、 **[文字列]** に「時計回り」と入力する。

最後に Keyboard コンポーネントの調整をします。

- instruction ルーチン
  - 配置済みの Keyboard コンポーネント (標準では key_resp_2 という名前になっているはず) の **[記録]** を「なし」にして、 **[正答を記録]** のチェックを外す。 **[開始]** を「時刻 (秒)」で 0.0 にする。

あとはループを挿入して、条件ファイルを作成しましょう。

- trials ループ (作成する)
  - trial ルーチンのみを繰り返すように挿入する。
  - **[繰り返し回数 $]** が 5(初期値) になっていることを確認する。
  - **[繰り返し条件]** に exp04_20.xlsx と入力する。
- exp04_20.xlsx (条件ファイル)
  - testDir、contextDir、correctAns の 3 パラメータを設定する。
  - 実験手続きの説明を満たすように testDir に 11 種類の、contextDir に 2 種類の値を入力する。
  - すべての行の correctAns に right を入力する。

さて、これで各ブロックの手続きは完成です。最後の項目「すべての行の correctAns に right を入力する」に注目してください。テスト刺激が時計回りに傾いている時にカーソルキーの右を押すのですから、testDir が正の値の時のみ right が正答のはずです。しかし、この実験では敢えてすべての試行の正答を right とした方が楽にデータ処理できるのです。次節でこの点について補足します。

4.5 反応の記録方法を工夫しよう

この章の実験の手続きは、心理物理学的測定法のひとつである恒常法の手続きです。恒常法を用いた実験のデータ分析でよく用いられる方法が心理物理曲線の作成です。 図 4.8 に今回の実験で得られる心理物理曲線の例を示します。横軸はテスト刺激の傾き、縦軸に時計回りに傾いているという反応の頻度です。物理的な刺激と反応が一致していれば、横軸の 0 度を境界として左側では縦軸の値は 0.0、右側では 1.0 となりますが、グラフは 0.0 から 1.0 へなだらかに上昇する曲線を描いています。グラフが 0.0 から 1.0 へ変化する範囲が左右に寄っていれば、実験参加者の判断が全体的に偏っていたことがわかります。また、この範囲が左右に広がっていれば、反時計回りか時計回りかの判断が難しい課題であったことがわかります。

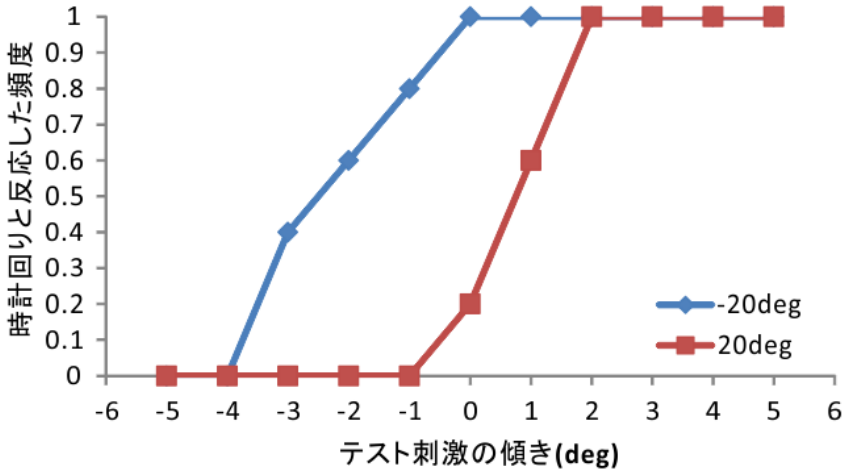


図 4.8　心理物理曲線の例。横軸にテスト刺激の傾き、縦軸に時計回りに傾いているという反応の頻度をプロットしています。折れ線グラフはそれぞれコンテクスト刺激の傾きが-20 度と 20 度の条件に対応しています。

心理物理曲線を描く時には、実験参加者の反応が刺激の物理的特性と一致していたか否かは関係がありません。ですから、PsychoPy で実験参加者の反応を記録する時に、両者が一致していたかを記録しても何の役にも立ちません。前節のように、「right(=時計回りに傾いている)」という反応をしていたかを記録するようにしておけば、正答率の値がそのまま心理物理曲線の縦軸の値として利用できます。分析の手間が大幅に省けます。便利なテクニックですので、ぜひ覚えておいてください。

チェックリスト

- 恒常法の実験において、正答率がそのまま心理物理曲線の縦軸の値として使用できるように正答を定義できる。

4.6 実験情報ダイアログで条件ファイルを指定しよう

この節からがいよいよこの章の本題です。exp04a.psyexp ではコンテクスト刺激の傾きとして-20 度と 20 度を用いましたが、これらの条件に加えて-70 度と 70 度傾いた条件のデータも取りたいとします。さらに、-20 度／ 20 度のコンテクストを使う試行と、-70 度／ 70 度のコンテクスト刺激を使う試行はそれぞれまとめて実施することにします。つまり、実験を二つのブロックに分割し、一方のブロックではコンテクスト刺激はすべて-20 度／ 20 度、もう一方のブロックではすべて-70 度／ 70 度とします。

コンテクスト刺激の種類で実験をブロック化せず、全部無作為な順番で実施するのであれば、条件ファイルに-70 度／ 70 度の条件に対応する行を追加するだけで対応できます。しかし、ブロック化するのであればこの方法は使えません。一番簡単な方法は、-70 度／ 70 度条件に対応する新たな条件ファイル (exp04_70.xlsx) を作り、この exp04_70.xlsx を条件ファイルとして使用する実験をもう一個作成するというものでしょう。まあ別にこの方法で乗り切っても構わないのですが、せっかくですのでひとつの実験ファイルで二つの条件ファイルを切り替える方法を習得しましょう。

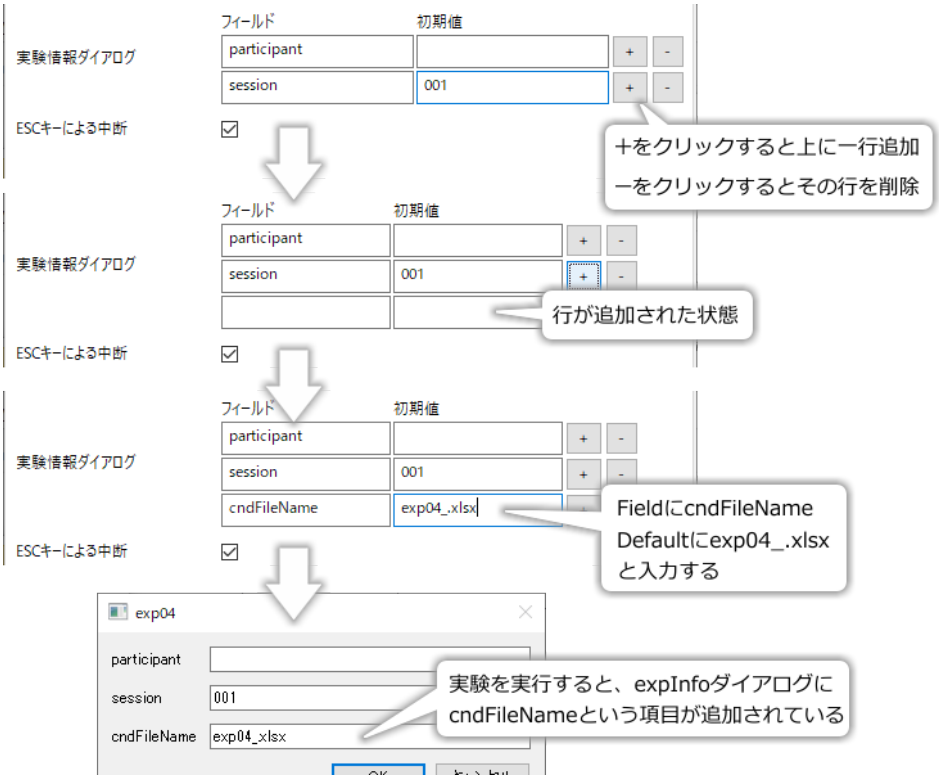


図 4.9　実験情報ダイアログに項目を追加する手順。

使用する条件ファイル名をはじめ、実験のパラメータを実験開始時に指定するには実験情報ダイアログを使用します。初期状態では実験情報ダイアログには session と participant の 2 項目しかありませんが、実験設定ダイアログから項目を追加することができます。 図 4.9 は実験情報ダイアログに項目を追加する手順を示しています。exp04a.psyexp を開いて設定ダイアログを開いてみましょう。 **[実験情報ダイアログを表示]** のチェックを外している人はチェックしなおしておいてください。初期状態では実験設定ダイアログの **[実験情報ダイアログ]** に session と participant という項目が表示されていて、その右側に [+]、[-] が書かれたボタンが表示されています。[+] ボタンを押すと、その行の下に新しい行が追加されます。[-] を押すと、その行が削除されます。participant と session のどちらの行の下に追加しても、動作には違いがありません。

では、[+] を押して新しい行を追加して、「フィールド」に cndFileName、「初期値」に exp04_.xlxs と入力してください。入力したら実験を実行してみましょう。すると、 図 4.9 の一番下の図のように実験情報ダイアログに cndFileName という項目が追加されていて、exp04_.xlsx という文字列が最初から入力されているはずです。「フィールド」は実験情報ダイアログに表示する項目の名前、「初期値」はその項目に最初から入力されている文字列 (初期値) に対応しています。participant のように「初期値」が空白の場合は、実験情報ダイアログでも空白になります。

実験情報ダイアログの項目名は、条件ファイルのパラメータ名と異なり、空白文字 (スペース) や#、$といった記号を含むことができます。日本語の文字列でも指定できますが、表示が乱れることがありますのであまりお

勧めしません。項目が実験情報ダイアログに表示される順番は、従来のバージョンでは PsychoPy が自動的に決定されてしまって自由に指定できなかったのですが、現在のバージョンでは実験設定ダイアログで入力した通りの順番となります (2020.2.6 で確認)。

実験情報ダイアログで入力した値は、条件ファイルに記述されたパラメータのように Builder 内部で参照することができます。ただし、条件ファイルの場合とは書き方が異なっていて、expInfo[' パラメータ名'] という具合に書きます。この書き方の意味を理解するためには Python の文法を理解する必要がありますので、ここではとりあえず「こう書くんだ」と覚えておいてください。

それでは、実験情報ダイアログを利用して「ひとつの実験ファイルで二つの条件ファイルを切り替える」という問題に挑戦してみましょう。先ほどの cndFileName という項目を exp04a.psyexp に追加した状態から作業を続けます。trials ループのプロパティを開いて、 **[繰り返し条件]** を$expInfo['cndFileName'] に変更してください。 **[繰り返し条件]** には$が付いていないので、条件ファイルからパラメータを読む時と同様に$を付けないといけない点に注意してください (図 4.10)。これで、実験実行時に実験情報ダイアログの cndFileName の項目に入力された名前の条件ファイルを開くようになりました。実験を exp04b.psyexp の名前で保存しておきます。

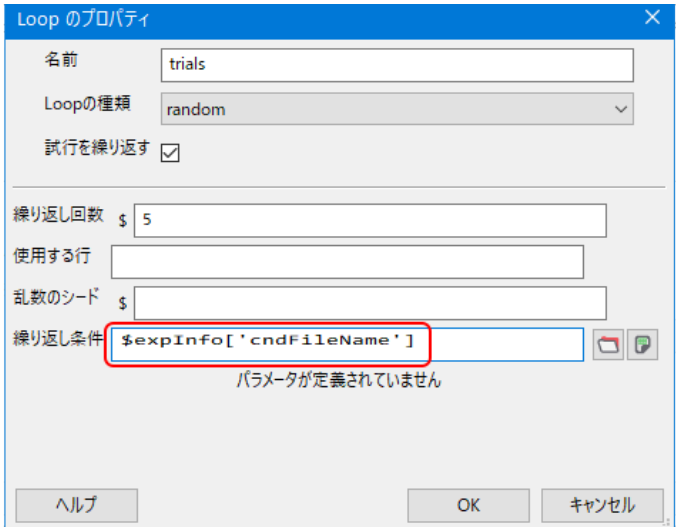


図 4.10 **[繰り返し条件]** に実験情報ダイアログの項目を指定します。先頭の$と項目名の前後の' を忘れないように注意してください。

続いて、Builder をいったん離れてコンテクスト刺激が-70 度／ 70 度傾いている条件の条件ファイルを作成しましょう。これは単に exp04_20.xlsx を開いて、contextDir の列の-20 を-70 に、20 を 70 に書き換えて別名で保存するだけです。ここでは exp04_70.xlsx という名前で保存しておきましょう。元の exp04_20.xlsx も引き続き使用しますので、上書き保存しないように注意してください。exp04_20.xlsx、exp04_70.xlsx の両方とも exp04b.psyexp と同じフォルダに置いてください。

以上で実行時に条件ファイルを切り替えられる実験の作成は終了です。PsychoPy に戻って exp04b.psyexp を実行しましょう。実験情報ダイアログの cndFileName に exp04_20.xlsx と入力すれば、-20 度／ 20 度、exp04_70.xlsx と入力すれば-70 度／ 70 度の条件の試行が始まります。最初から exp04_.xlsx という文字列が入力されているので、_の後に 20 または 70 と入力するだけでよいはずです。ぜひ、-70 度／ 70 度の実験を最後まで行って、-20 度／ 20 度の実験結果と比較してみてください。

チェックリスト

- 実験情報ダイアログの項目を追加、削除することができる。
- 実験情報ダイアログの項目の初期値を設定することができる。
- 実験情報ダイアログの項目名から、その値を利用するための Builder 内における表記に変換することができる。
- 条件ファイル名を実験情報ダイアログの項目から取り出してループのプロパティに設定することができる。

4.7 実験情報ダイアログで項目を選択できるようにしよう

前節で実験情報ダイアログを使って条件ファイルを切り替えできるようになりましたが、実際に使用してみた感想はいかがでしょうか？ 確かに便利なのですが、条件ファイル名を入力するのが面倒ではないでしょうか。今回の例のように入力する内容がいくつかの決まったパターンしかない場合、いちいちキーボードから入力するのではなくドロップダウンメニューから選択できるようにする機能が Builder には用意されています。

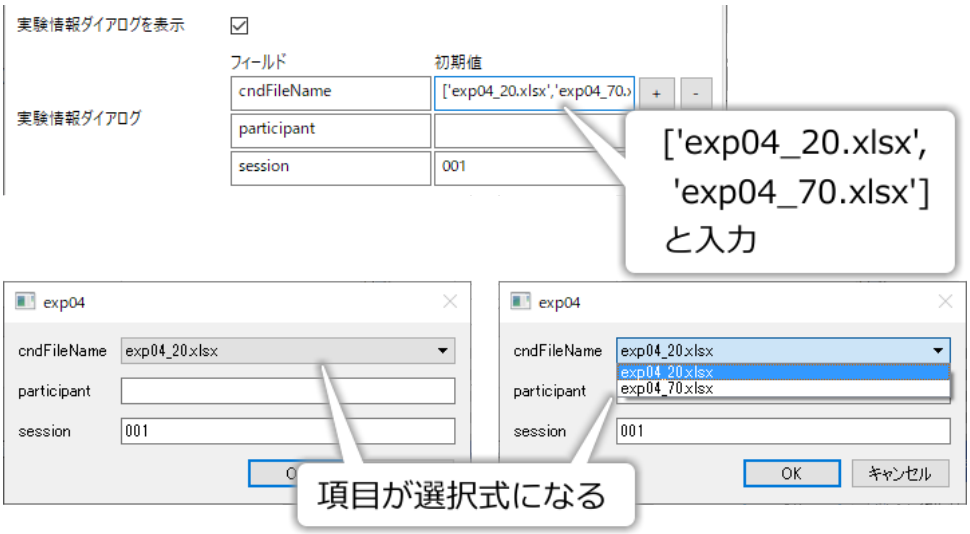


図 4.11　実験情報ダイアログの項目を選択式にする

さっそく使ってみましょう。exp04b.psyexp を開いて設定ダイアログを開き、先ほど編集した **[実験情報ダイアログ]** の cndFileName の項目を以下のように変更します。[ ] や' '、, などの記号に注意して入力してください。

```
['exp04_20.xlsx', 'exp04_70.xlsx']
```

入力できたら実験を実行してみましょう。 図 4.11 に示すように、cndFileName の項目が exp04_20.xlsx と exp04_70.xlsx のどちらかを選べるようになりました。この形式をドロップダウンメニューと呼びます。これで条件ファイルの切り替えが少し楽になったのではないでしょうか。

さて、動作を確認したところで、この機能を使うためのポイントを確認しておきましょう。 **[実験情報ダイアログ]** の項目の「初期値」が以下の条件を満たすとき、その項目はドロップダウンメニューと解釈されます (上級者向け：厳密には「初期値」に記入されている内容が Python の list オブジェクトとして解釈可能であることが条件)。

1. メニューの各項目が ' ' または " " で囲まれていて、かつ , で区切られている。

2. 1. の内容が [と] で囲まれている。 [の前や] の後ろには文字があってはいけない。

1 番目の条件は、Keyboard コンポーネントの **[検出するキー $]** で'left', 'right' という具合にキー名を ' ' または " " で囲ってカンマ区切りで並べたとの同じように書くということです。いろいろな値を追加して練習してみてください。

チェックリスト

- 実験情報ダイアログの項目をドロップダウンメニューにできる。

4.8 多重繰り返しを活用しよう

今回の実験のようにある要因 (ここではコンテクスト刺激の傾き角度) で試行がブロック化されている場合、一人の実験参加者が両方のブロックへ参加する実験計画を用いることもあれば (参加者内計画)、一人の実験参加者はいずれか一方のブロックしか参加しない計画を用いることもあります (参加者間計画)。前節の実験情報ダイアログで実験条件を指定する方法は、参加者内計画でも参加者間計画でも柔軟に対応できますが、実験者がいちいち条件ファイル名を設定しないといけないので面倒です。面倒なだけならいいのですが、条件ファイル名を間違えてしまうかもしれません。参加者内計画の場合、一方の条件を実行したら次のブロックは必ず残りの一方の条件です。Builder に自動的に二つの条件ファイルを読み込んで実行させるように、exp04b.psyexp を改造してみましょう。

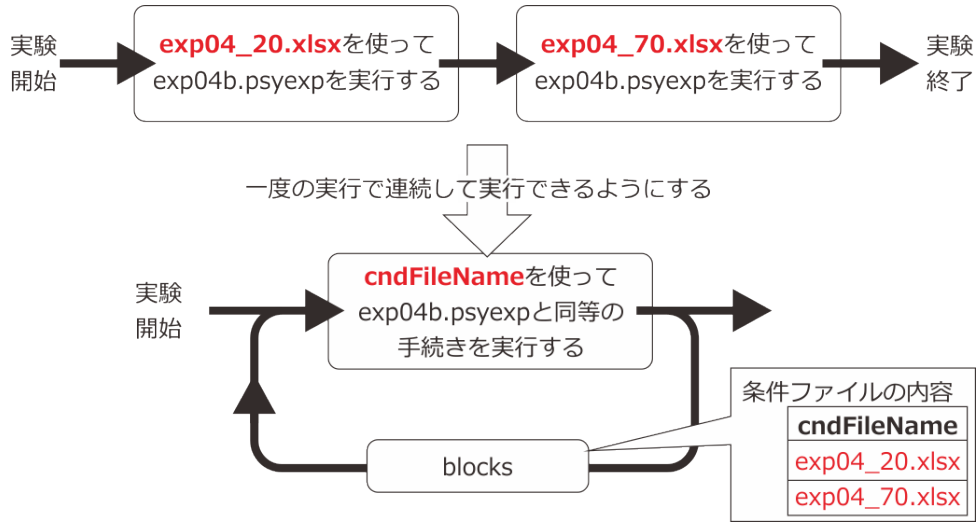


図 4.12　読み込む条件ファイルを変更しながら繰り返すことによって、-20 度／ 20 度と-70 度／ 70 度の条件を連続して実行する実験を作成します。

図 4.12 に改造の方針を示します。exp04b.psyexp を使う場合は、 図 4.12 の上の図のように、条件ファイル名を変更しながら 2 回 exp04b.psyexp を実行しなければいけません。これは、今まで作ってきた実験における「刺激の色や傾きを変更しながら刺激提示を繰り返す」という作業とよく似ています。ということは、刺激の色や傾きを条件ファイルから読み込んで代入しながら繰り返すことができたように、 図 4.12 下のようにすれば条件ファイルの名前を別の条件ファイルから読み込んで代入しながら繰り返すことができるはずです。

さっそく改造してみましょう。Builder で exp04b.psyexp を開いて、念のため別名で保存しておきましょう

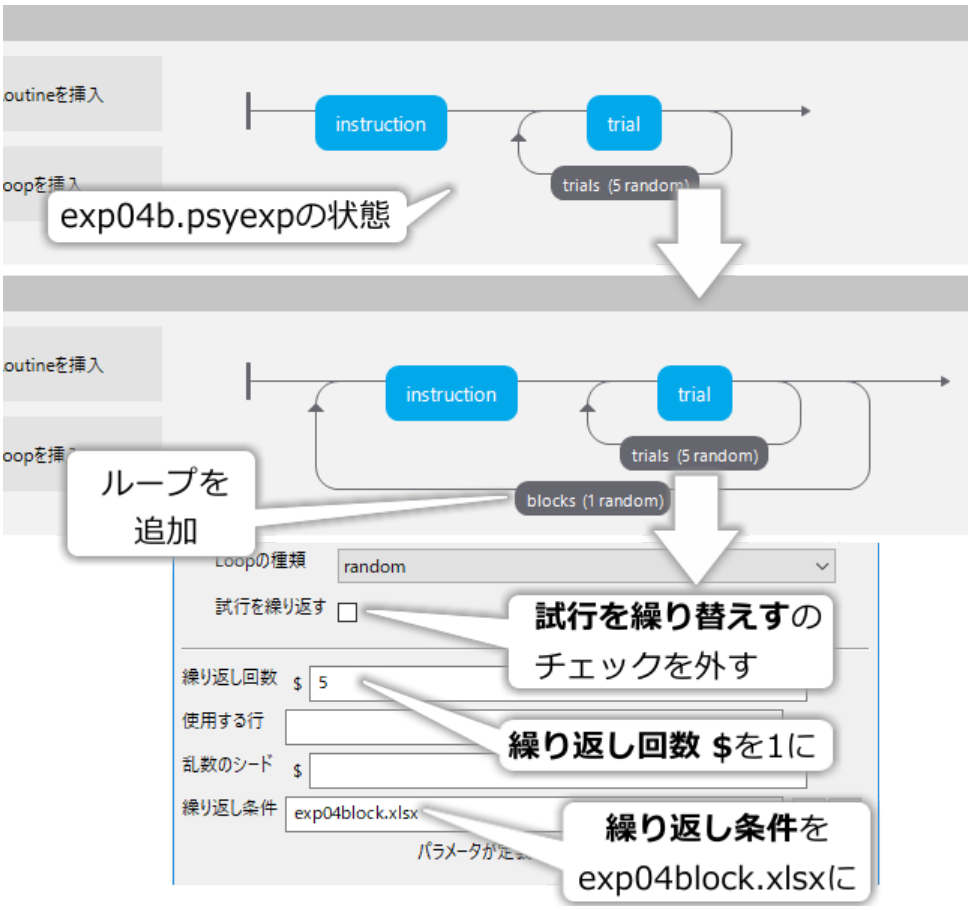


図 4.13　多重に繰り返しを挿入します。

(exp04c.psyexp とします)。そして、 図 4.13 のように blocks というループを挿入してください。Blocks ループの始点は instruction ルーチンの後ろでも構わないのですが、後ろに挿入してしまうと 1 回目の trials ループが終わった直後に 2 回目の trials ループが始まってしまうため、実験参加者に対する予告なしにいきなり-20 度／ 20 度条件と-70 度／ 70 度条件が切り替わってしまいます。instruction ルーチンを bolorks ループに含むようにしておけば、条件が切り替わる前に教示画面が再び表示されますので、切り替わりがわかりやすいでしょう。さらにわかりやすくするためには、trial ルーチンの前にこれから始まる条件を表示するためのルーチンを挿入すると良いでしょうが、これは練習問題とします。bocks ループのプロパティでは、 **[繰り返し回数 $]** を 1 に設定して、 **[繰り返し条件]** に exp04blocks.xlsx と入力しておきます。 **[Loop の種類]** は random のままでよいでしょう。これで新しいループの追加は完了です。

続いて、blocks ループのための条件ファイル、exp04blocks.xlsx を用意しましょう。この条件ファイルでは 図 4.14 左上のように exp04_20.xlsx と exp04_70.xlsx の二つの値を持つ cndFileName というパラメータを設定します。続いて trials ループで cndFileName に基づいて条件ファイルを読み込むように設定するために、trials ループの **[繰り返し条件]** を$cndFileName に変更します (図 4.14 右上)。実験情報ダイアログで条件ファイル名を入力する必要はなくなったので、実験設定ダイアログを開いて実験情報ダイアログから cndFileName の行を削除しておきましょう (図 4.14 下)。

以上で改造は終了です。exp04c.psyexp を実行して、自動的に-20 度／ 20 度条件と-70 度／ 70 度条件が続けて実行されることを確認してください。blocks ループの **[Loop の種類]** を random のままにしたので、-20 度／ 20 度条件と-70 度／ 70 度条件のどちらが先に実行されるかは毎回無作為に決定される点に注意してください。

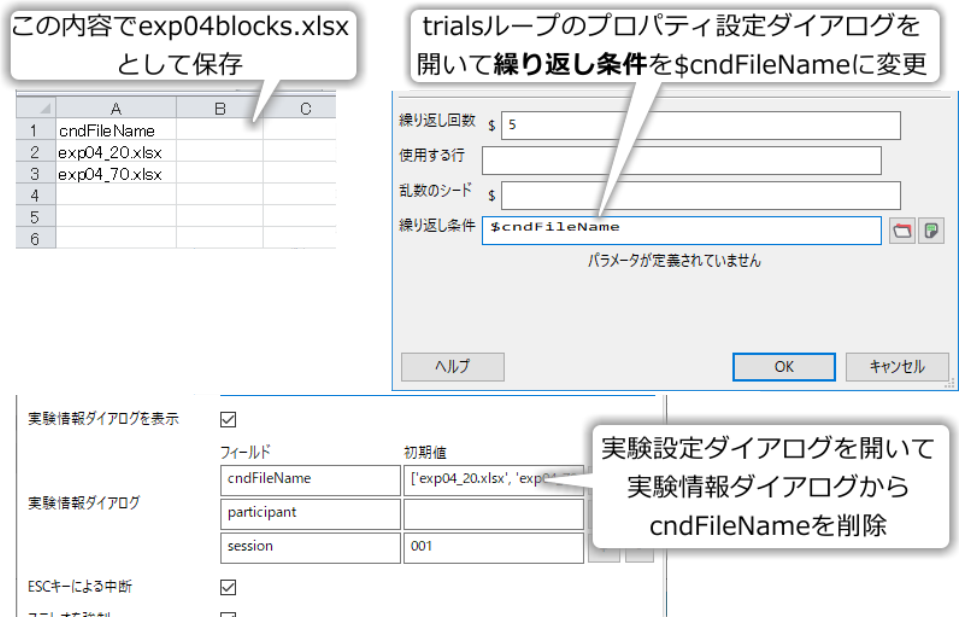


図 4.14　blocks ループを使って条件ファイルを切り替えるように trials ループなどを修正します。

なお、ループのプロパティには **[試行を繰り返す]** という項目がありますが、この項目は多重繰り返しを使ったときのデータ保存形式に関係があります。詳しくは「4.11.2: ループの **[試行を繰り返す]** プロパティについて」を参照してください。

チェックリスト

- 多重繰り返しを挿入できる。
- 多重繰り返しの内側のループで条件ファイル名を外側のループの条件ファイルから読み込んで設定することができる。

4.9 複雑な実験を作るときにちょっと役立つテクニック

本章の実験とは関係ありませんが、多重繰り返しを応用すると、複雑な実験の一部分だけを実行することができます。複雑な実験を作成する際に知っておくと便利なので、ここで紹介しておきましょう。図 4.15 上段は教示などをを含む複雑なフローを持つ実験の例です。この実験の終盤にある test というルーチンに問題があって、少し修正したとしましょう。修正がうまくいったか実際に実行して確認したいところですが、test ルーチンにたどり着くまでには practices や trials といったループをすべて実行する必要があります。これらのループ内でおこなわれる課題によっては、数十分かかってしまうかも知れません。

こういったときに役に立つのが「**[繰り返し回数 $]** 0 回のループ」です。図 4.15 下段のように、test ループより前の部分をすっぽり包むループを追加して **[繰り返し回数 $]** 0 を設定しましょう。**[Loop の種類]** などはどれでも構いませんし、**[繰り返し条件]** も空欄で構いません。**[名前]** もなんでも結構です。これで実験を実行すると、0 回のループで囲まれた部分は「0 回実行する＝全く実行されない」ので、すぐに test ルーチンが実行されます。確認が済んだら、0 回のループを削除するだけで元の状態に戻ります。

なお、図 4.15 の例で「trials ループで囲まれている部分だけを飛ばしたい」場合は、新たにループを追加しな

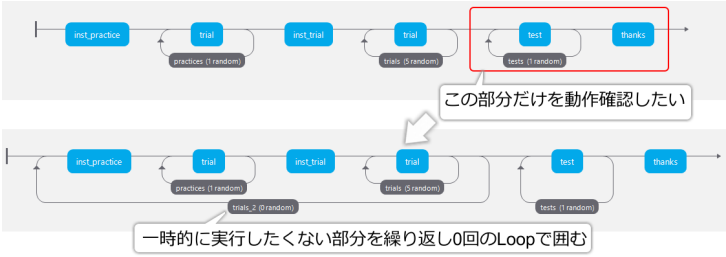


図 4.15　繰り返し 0 回のループを挿入すると、ループで囲んだ部分の実行を飛ばすことができます。

くても trails ループの **[繰り返し回数]** を 0 にするだけで OK です。0 回ループは複雑な実験を作成するときに非常に便利なテクニックなので、覚えておいてください。

関連テクニックとして、「あるルーチンに含まれる一部のコンポーネントだけ実行したくない」場合についても触れておきます。これはちょっと具体例を挙げるのが難しいのですが、「実験を実行してみるとエラーで停止してしまうのだけれど、どのコンポーネントが原因なのかわからない」場合などに便利です。各コンポーネントのプロパティダイアログには「テスト」というタブがありますが、このタブにある **[コンポーネントの無効化]** という項目をチェックすると、あたかもこのコンポーネントが配置されていないかのように全く実行されなくなります (図 4.16)。原因不明のエラーで困っている時に、あるコンポーネントを無効化して実行するとエラーが発生しないのなら、きっとそのコンポーネントがエラーに関与しているはずです。中級者から上級者向けの機能かも知れませんが、こんな機能があるということを頭の片隅に置いておくと役に立つかもしれません。

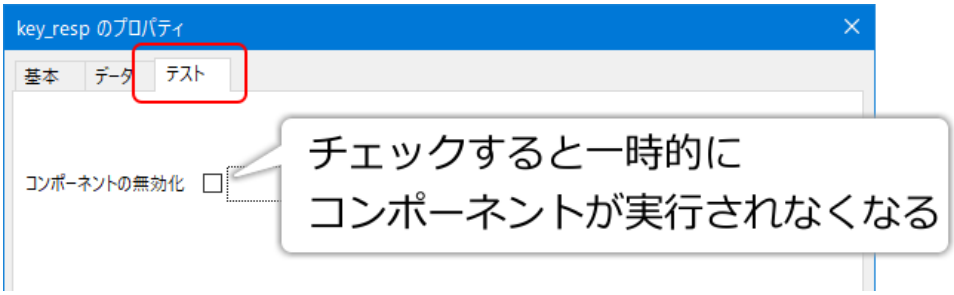


図 4.16　「テスト」タブの **[コンポーネントの無効化]** をチェックするとコンポーネントを一時的に無効化できます。

4.10 練習問題：条件ファイルを使う順番を指定できるようにしよう

exp04c.psyexp では blocks ループの **[Loop の種類]** に random を指定したので、実行の度にどちらの条件が先に実行されるかが無作為に変化します。ところが、この章の実験のようにブロックの順序を実験参加者間で変更する場合は、各順番に同数の参加者が割り振られるようにバランスを取ることがよく行われます。つまり、実験参加者が 20 人いるとしたら、10 人は-20 度／ 20 度条件を先に、残り 10 人は-70 度／ 70 度条件を先に行うようにするということです。実行順序を無作為に決定してしまうと、9 人と 11 人といった具合にバランスが崩れてしまう可能性があります。確実にバランスが取れるようにするには、-20 度／ 20 度条件が先になるように書かれた条件ファイルと、-70 度／ 70 度条件が先になるように書かれた条件ファイルを用意して、どちらの条件ファイルを使用するかを実験情報ダイアログで指定するという方法があります。本章のまとめとして、exp04c.psyexp を改造してこの方法で実験参加者数のバランスを取ることができる実験を作成してください。さらに加えて、trials ループを開始する前に、これから行われるブロックについての情報を表示するルー

チンも追加してください。

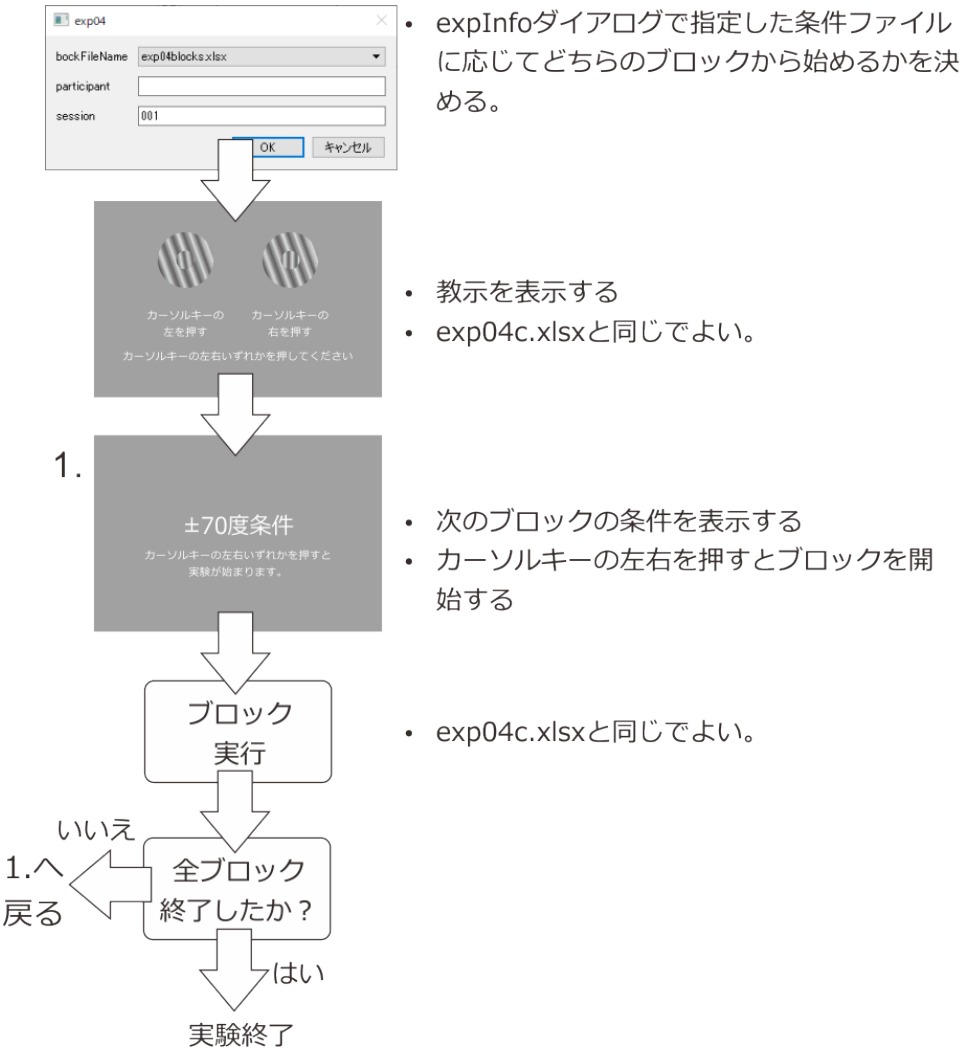


図 4.17　練習問題で作成する実験の流れ

文章だけではわかりにくいので、図 4.17 をご覧ください。まず、exp04c.psyexp をベースとして、実験情報ダイアログにブロックの順番を指定するための項目を設けてください。そしてこの項目に例えば exp04blocks.xlsx を指定したら-20 度／ 20 度条件から始まって、exp04blocks2.xlsx を指定したら-70 度／ 70 度条件から始まるようにしてください。そして、教示画面を表示した後に、これから始まるブロックが-20 度／ 20 度条件と-70 度／ 70 度条件のどちらなのかを示す画面を表示してください。最初のブロックが終了した後は、教示画面ではなくこれから始まるブロックを示す画面に戻るようにしてください。以上が練習問題です。

ちょっとこれだけでは難しいという方のために、 図 4.18 にヒントを示します。条件ファイルでブロックの順番を指定するためのポイントは、blocks ループの **[Loop の種類]** の値です。ブロックの順番を指定するための条件ファイルは、 図 4.18 下のようなファイルと、2 行目と 3 行目を入れ替えたファイルを用意すればよいでしょう。なぜこれでうまくいくかわからない方は、第 3 章 の **[Loop の種類]** の解説を読み直してください。健闘を祈ります。

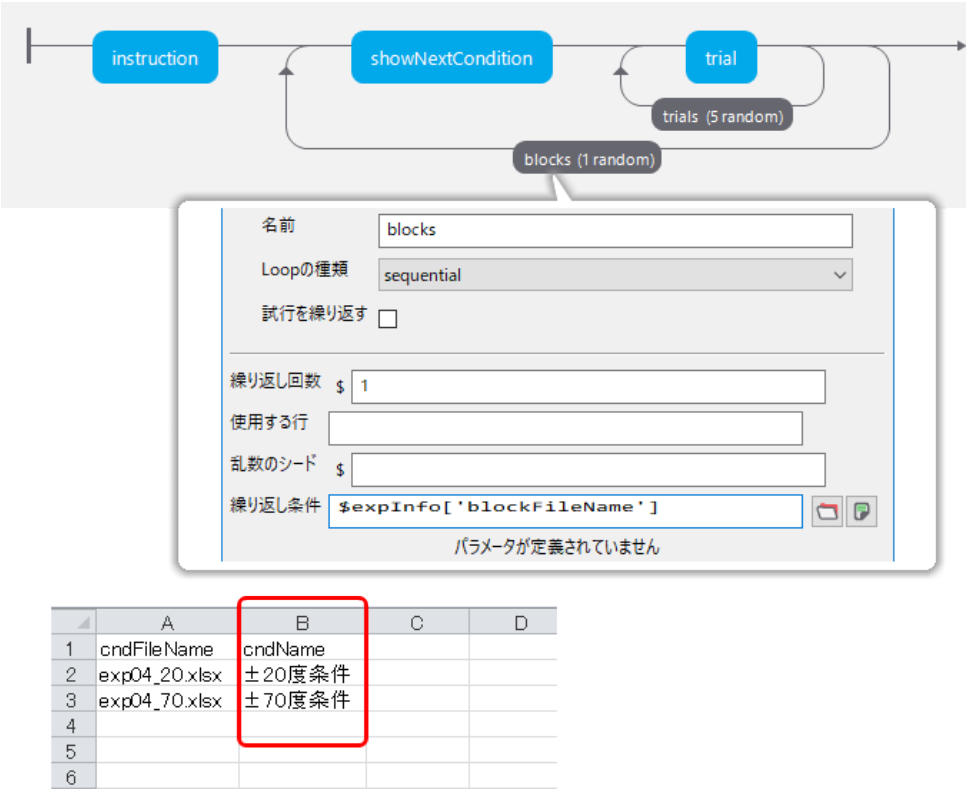


図 4.18　練習問題のヒント。

4.11 この章のトピックス

4.11.1 Grating コンポーネントの [テクスチャ] プロパティについて

視知覚の研究でグレーティングと言うと一般的に正弦波状に明るさが変調された刺激を指すのですが、PsychoPy の Grating コンポーネントはより一般的な「同一のパターンが 2 次元に繰り返される刺激」を描く機能を持っています。 図 4.19 は **[テクスチャ]** に指定できる値を示しています。sin は正弦波、saw は鋸波、sqr は矩形波、tri は三角波を描きます。None を指定すると、 **[前景色]** で指定された色で塗りつぶします。これらの波は 1 次元、すなわち一方向にのみ明るさが変化しますが、sinXsin では垂直方向と水平方向に変化する 2 次元の波を描きます。同様に sqrXsqr、gauss、radRamp、raisedCos といった波形を指定できます。垂直方向と水平方向の空間周波数を別々に指定するためには、 **[空間周波数 $]** に刺激の大きさを指定する時と同様の記法を用います。 図 4.20 左上は sqrXsqr の波形に **[空間周波数 $]** として [2,3]、[3,2] を指定した時の出力を示しています (単位は height)。図からわかるように、第 1 の値が水平方向、第 2 の値が垂直方向に対応しています。

[テクスチャ] および **[マスク]** には、画像ファイルを指定することもできます。 図 4.20 右上は、PsychoPy のアイコン画像を保存した icon.png というファイルを **[テクスチャ]** に指定した例を示しています。 **[テクスチャの解像度 $]** の値が 2 の冪乗 (64 や 128 など) であったように、PsychoPy 内部ではテクスチャは縦横の解像度が 2 の冪乗の正方形でなければいけません。それ以外の解像度の画像を指定すると PsychoPy が内部的に拡大縮小を行いますので、正確さを期する場合は最初から 2 の冪乗の正方形の画像ファイルを作成することが大切です。 **[マスク]** にモノクロの画像ファイルを指定すると、 **[マスク]** 画像のピクセルの明るさがテクスチャの透明度となります。文章では説明しにくいので 図 4.20 下段をご覧ください。mask1.png、mask2.png とい

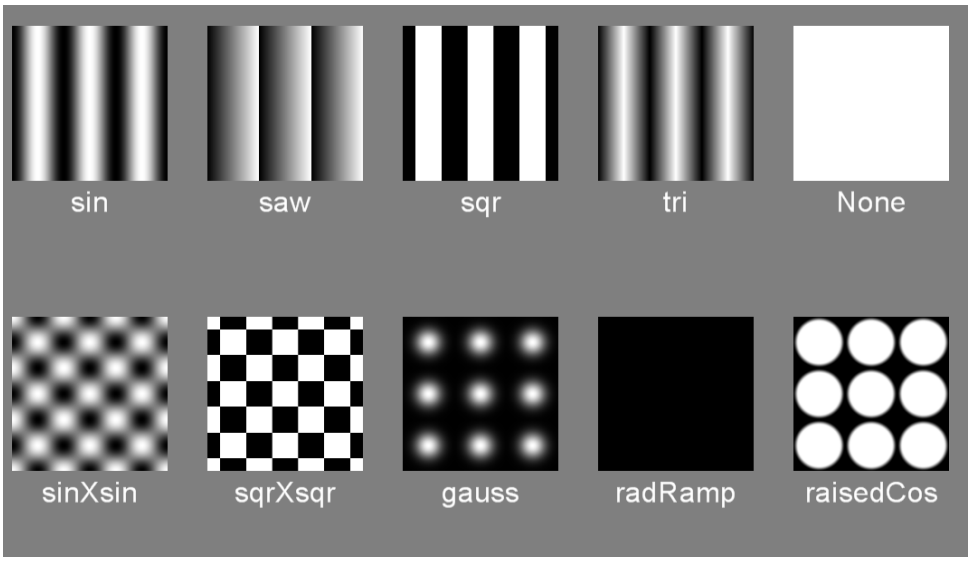


図 4.19　Grating コンポーネントの **[テクスチャ]** として指定できる値。これらの他に画像ファイル名を指定することができます。いずれも **[空間の単位]** は height で **[空間周波数 $]** は 3.0 を指定しています。

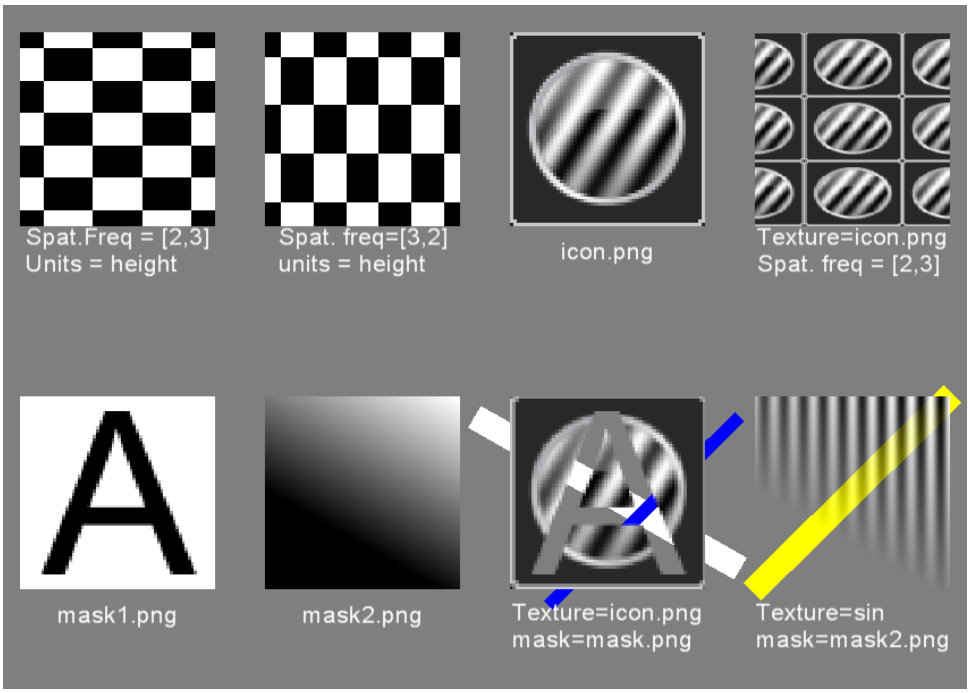


図 4.20　**[空間周波数 $]** に 2 つの値を指定する例と (上段の左側 2 つ)、 **[テクスチャ]** に画像ファイルを指定する例 (上段の右側 2 つ)、 **[マスク]** に画像ファイルを指定する例 (下段)。

う 2 種類のモノクロ画像をマスクとして用意しました。 **[マスク]** に mask1.png を、 **[テクスチャ]** に先ほどの icon.png を指定した例が 図 4.20 下段左から 3 番目です。mask1.png の黒色の部分が透明になって、背景の灰色が見えています。透明になっていることがわかりやすいように、白色と青色の Polygon コンポーネントを背後に配置しています。 図 4.20 下段の右端は、 **[テクスチャ]** を sin にして、 **[マスク]** に mask2.png を指定した例を示しています。mask2.png が黒色から白色に向かって滑らかに変化しているので、縞模様が滑らかに透明から不透明へと変化しています。

4.11.2 ループの [試行を繰り返す] プロパティについて

ループの **[試行を繰り返す]** プロパティは、実験設定ダイアログで「xlsx 形式のデータを保存」または「CSV 形式のデータを保存 (summaries)」をチェックして出力されるデータファイルの形式を設定するものです。

「xlsx 形式のデータを保存」をチェックして exp04c.psyexp を実行することによって作成される xlsx ファイルの例を 図 4.21 に示します。ただし、 **[試行を繰り返す]** はチェックしていないものとします。

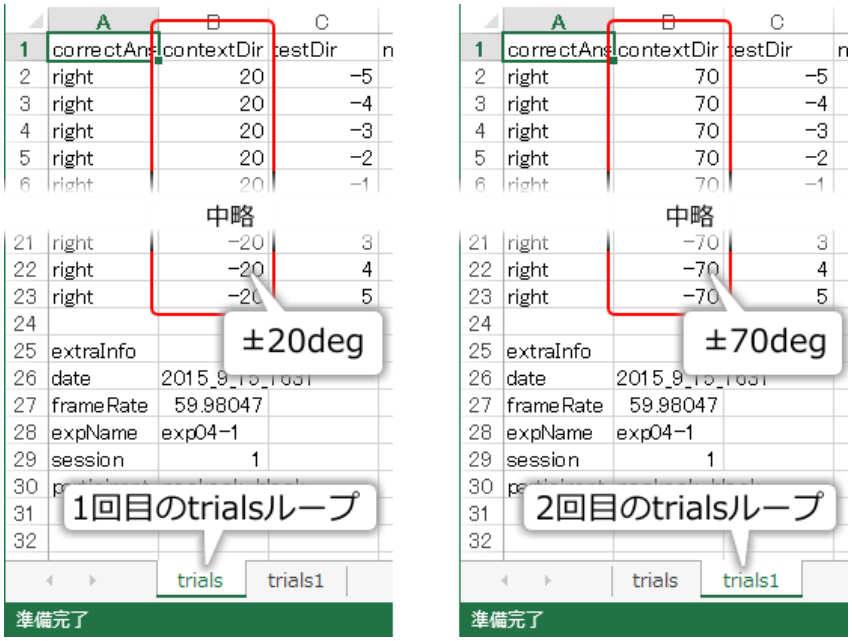


図 4.21　多重ループを用いた実験によって作成される xlsx 記録ファイルの例 (**[試行を繰り返す]** をチェックしない場合)。

表 4.1　多重ループによって作成される xlsx 記録ファイルのシート名 (内側のループ名が trials の場合)

trials	最初に実行された trials ループ
trials1	2 回目に実行された trials ループ
…	…
trialsN (N は自然数)	N+1 回目に実行された trials ループ

trials と trials1 というシートが作成されていますが、これは 表 4.1 に示すようにそれぞれ trials ループの 1 回目、2 回目に対応しています。シートの内容を確認すると、 図 4.21 の例の場合は trial シートの contextDir の列が 20 または-20 なので、第 1 ブロックは-20 度／ 20 度条件であったことがわかります。同様に、trial1 シートの contextDir の列を確認すると大 2 ブロックは-70 度／ 70 度条件であったことがわかります。

今回の実験では blocks ループによる trials ループの繰り返しは 2 回のみでしたが、仮に blocks ループによる繰り返しが 20 回に及ぶ場合は trials、trials1、…、trials19 と 20 枚もシートが作成されます。あまりシート数が多くなると Excel 上での作業が大変ですので、繰り返し回数が多くなる場合は実験を分割することも検討すべきです。

続いて blocks ループの **[試行を繰り返す]** をチェックした場合に出力される xlsx 記録ファイルの例を 図 4.22 に示します。trials、trials1 というシートができるのはチェックを外した場合と同じですが、それに加えて

blocks というシートが含まれています。このシートには、blocks ループの繰り返しにおいて、exp04blocks.xlsx から読み込んだ条件のどの行が使われたかが記録されています。別にこのシートが存在していてはいけないわけではないのですが、trials、trials1 のシートから読み取ることができる情報しか含まれていないので、無駄なシートといえます。今回のように blocks ループが一番外側のループだったら 1 枚余分なシートが増えるだけですが、さらに外側にループが組まれている実験では何枚も無駄なシートが作成されてしまいます。このような場合は **[試行を繰り返す]** のチェックを外してシート数を抑えることが有効です。

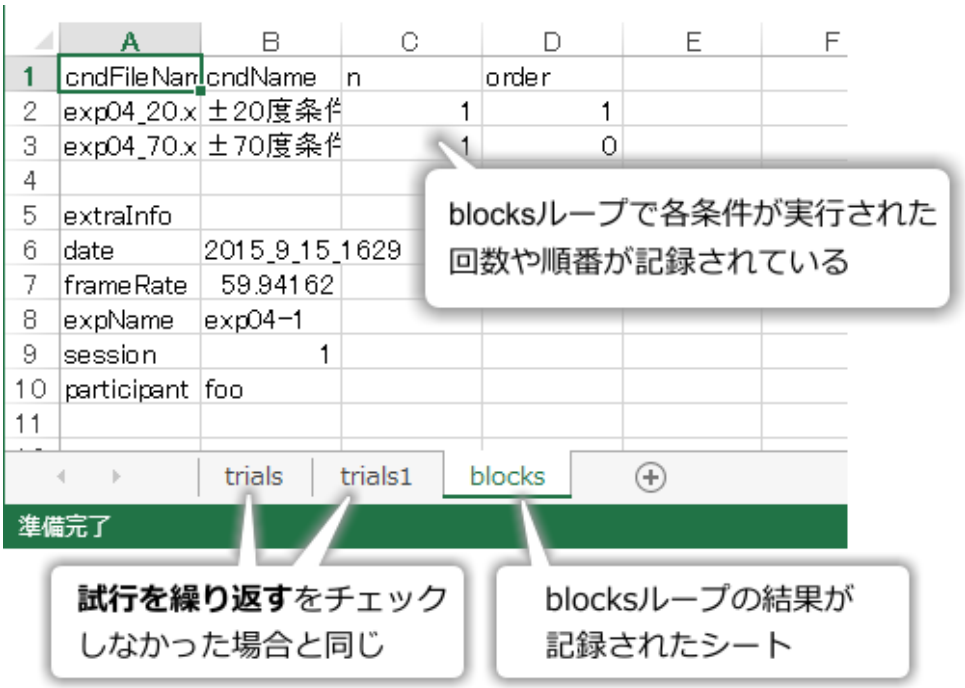


図 4.22　多重ループを用いた実験によって作成される xlsx 記録ファイルの例 (**[試行を繰り返す]** をチェックした場合)。

なお、外側のループ内にループに含まれないルーチンが存在していて、そのルーチンでの刺激や反応を記録する必要がある場合は、 **[試行を繰り返す]** のチェックを外してはいけません。具体的には、 図 4.23 の test ルーチンのようなものが blocks ループ内にある場合です。このような場合に blocks ループの **[試行を繰り返す]** のチェックを外してしまうと、test ルーチンで用いられた条件や、test ルーチンにおける参加者の反応などが記録されません。チェックを外していいか自信がない場合、xlsx 記録ファイルのシート数を気にしないのであれば **[試行を繰り返す]** のチェックはつけたままにしておくというのも一つの手だと思います。

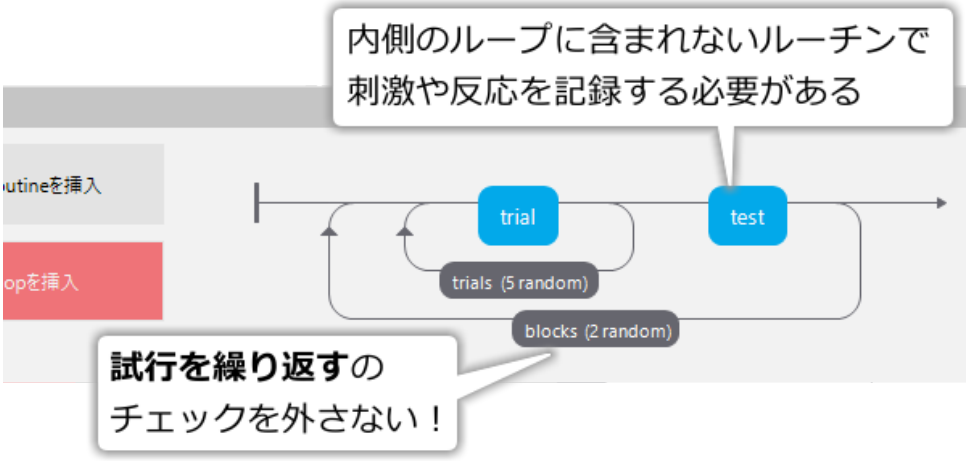


図 4.23　**[試行を繰り返す]** のチェックを外してはいけない例

第 5 章

Python コードを書いてみよう—視覚の空間周波数特性

5.1 この章の実験の概要

この章では、視覚の時空間特性の計測実験を題材として、Builder による実験に Python コードを組み込む方法を解説します。グラフィカルユーザーインターフェース (GUI) で実験を作成できる事が Builder の長所ですが、正直なところ本格的な実験をしようとすると「あれができない、これができない」という事だらけであまり役に立ちません。しかし、Builder に Python のコードを組み合わせることによって、飛躍的に「できる事」の幅が広がります。プログラミングの経験がない方には最初かなり難しく感じられるかもしれませんが、実際に実験を動かしながら少しずつ理解を深めてください。

この章の実験では、視覚の空間周波数特性を取り上げます。「空間周波数」という用語は 第 4 章 で紹介しましたね。皆さんは、視力検査であまりにも小さな視標は知覚できない事を経験していると思います。視標が小さいということは狭い範囲内で明暗が大きく変化するということですから、「空間周波数」という用語を使えば「空間周波数が高すぎる視覚刺激は私たちには知覚できない」と表現できます。視力検査では高空間周波数の刺激がどこまで知覚できるかしか調べませんが、実は空間周波数が低すぎる刺激に対しても私たちの視覚の感度が低下することがわかっています。この章では、空間周波数の変化に応じて私たちの視覚の感度が変化することを確認する実験を作成します。便宜上「実験」と呼んでいますが、ベースになっているのは筆者が以前に知人から紹介してもらったゲーム風のデモです。正確な感度の計測に適した手続きではありませんが、Python のコードを Builder に組み込む最初の一歩として適しているので取り上げました。

なお、この実験では刺激の視角が非常に重要な意味を持ちますので、 **[空間の単位]** に deg を使用します。モニターの観察距離の設定などをしっかりおこなっておいてください (第 2 章 参照)。

図 5.1 に使用する刺激を示します。スクリーン中央に直径 0.1deg の白色の小さな円を提示します。実験中、実験参加者はこの刺激を固視し続けないといけません。この刺激を固視点と呼ぶことにします。固視点から視角で 5.0deg 離れた位置に、直径 4.0deg の時計回りまたは反時計回りに 15 度傾いたグレーティング刺激を 1.0 秒提示します。この刺激をターゲットと呼ぶことにします。ターゲットの出現方向は固視点の右を 0 度として反時計回りに 0 度、45 度、90 度、135 度、180 度、225 度、270 度、315 度の 8 方向の中から無作為に選びます。グレーティングの空間周波数は 0.2、0.4、0.8、1.6、3.2、6.4、12.8 cpd (cyecle per degree: 視角 1 度あたりの繰り返し回数) の 7 種類を用います。

図 5.2 に実験の手続きを示します。試行開始時には、スクリーン中央に固視点のみが提示されています。実験

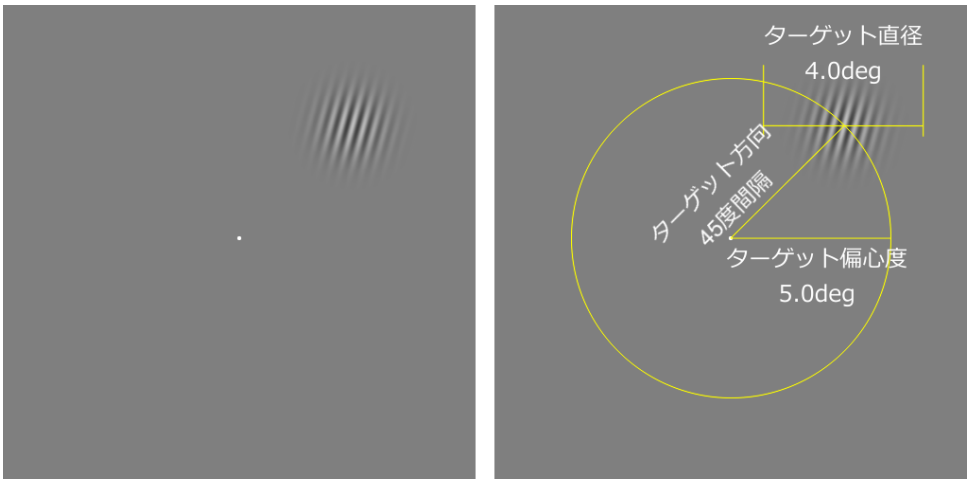


図 5.1　実験に使用する刺激。

参加者がカーソルキーの左右どちらかを押すと、1 秒後にターゲットがいずれかの位置に出現します。ターゲットの色はキーが押された直後には$[0,0,0]、すなわち背景と全く同一で知覚することができませんが、6 秒間かけて一定の速度で$[1,1,1] まで変化します。視覚刺激の明暗差のことをコントラストと言いますが、この用語を使えばターゲットのコントラストが上昇していくと表現できます。実験参加者は、ターゲットがどちらに傾いているかを知覚できたらできるだけ速く、反時計回りであればカーソルキーの左、時計回りであれば右のキーを押します。キーが押されるとターゲットは消えて、直ちに次の試行に進みます。ターゲット出現後 6 秒経過しても反応がなかった試行はエラーとして記録して、やはり直ちに次の試行に進みます。時間と共にターゲットのコントラストが上昇するので、実験参加者の反応時間が早いほど低いコントラストでターゲットを知覚できたと考えられます。以上の手続きを、すべてのターゲット方向 (8 種類) とターゲット空間周波数 (7 種類) の組み合わせに対して反時計回りを 1 回、時計回りを 1 回、合計 $8 \times 7 \times 2 \times 2 = 112$ 試行行います。途中に休憩を挟まずに一気に実行し、すべての試行が終了すれば実験は終わりです。

最初に述べたとおり、この手続きは正確な感度を計測することよりも、簡単な手続きで空間周波数による感度の違いを感じるためのデモと言った方が適切です。この手続きでは試行数が少なすぎますし、なにより参加者の反応時間からターゲットを検出できる閾値のコントラストを測定しようという発想自体が正確な測定に適していません。というのも、反応時間には実際にターゲットが知覚できてからキーを押すまでの時間などが含まれて、その間にもターゲットのコントラストは上昇し続けているからです。正確さを期するのであれば 第 4 章 のような恒常法の手続きを用いるべきです。この章の実験は、あくまで Builder の実験に Python のコードを組み込む方法を学習するための例題だと考えてください。

さて、まずは 第 4 章 までに解説済みの作業でできるところまで実験を作成しましょう。以下のように実験を作成して exp05a.psyexp の名前で保存するものとします。

- 実験設定ダイアログ
  - **[単位]** を deg にする。モニターの設定 (観察距離、モニターの幅長および解像度) をまだ設定していない場合は 第 2 章 を参考にモニターセンターを開いて設定する。
- trial ルーチン
  - Grating コンポーネントをひとつ配置し、以下のように設定する。
    - 「基本」タブの **[名前]** を targetStim に、 **[終了]** を実行時間 (秒) で 6.0 にする。

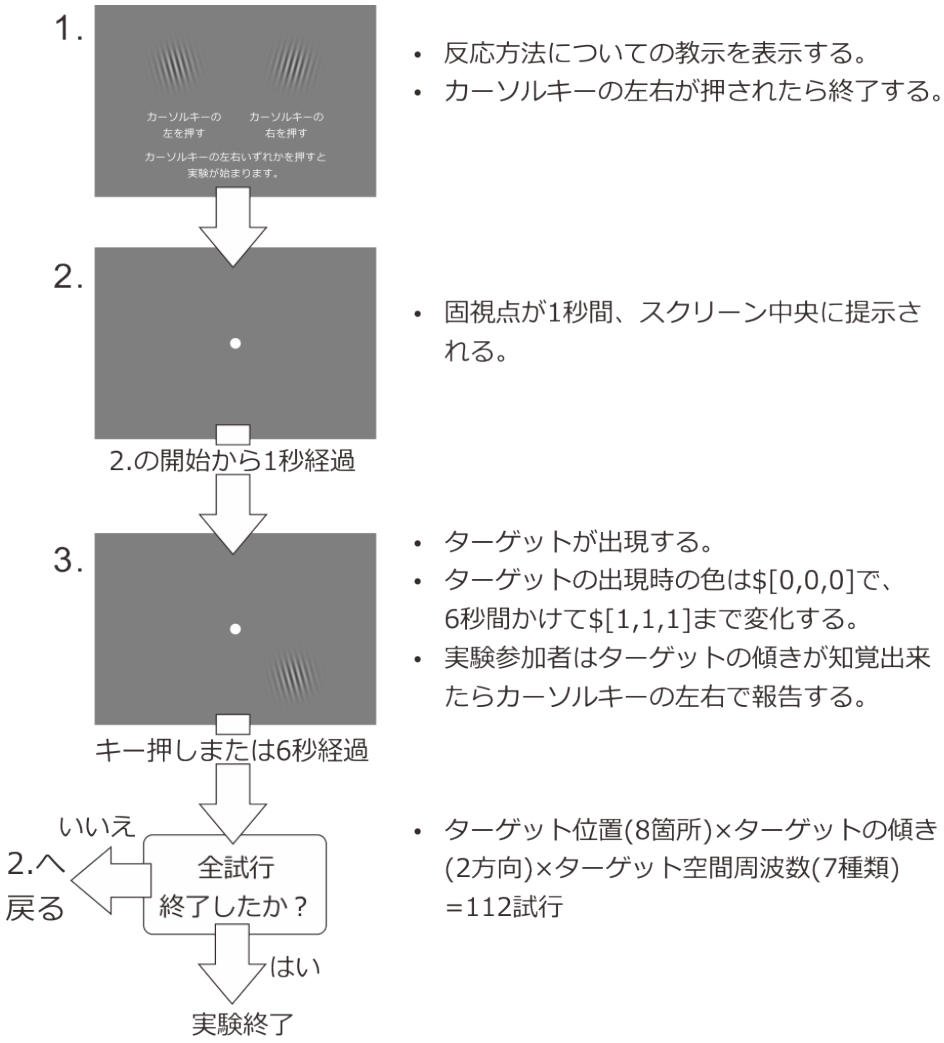


図 5.2　実験の手続き。

- 「テクスチャ」タブの **[マスク]** を gauss に、 **[テクスチャの解像度 $]** を 512 にする。 **[空間周波数 $]** を targetSF にして、「繰り返し毎に更新」にする。
- 「レイアウト」タブの **[回転角度 $]** を targetDir にして、「繰り返し毎に更新」にする。 **[サイズ [w, h] $]** を [4.0, 4.0] にする。

– Keyboard コンポーネントをひとつ配置し、以下のように設定する。

- 「基本」タブの **[終了]** を実行時間 (秒) で 6.0 にする
- 「データ」タブの **[検出するキー $]** を'left', 'right' にする。 **[正答を記録]** をチェックして、 **[正答]** に$correctAns と入力する。

– Polygon コンポーネントをひとつ配置し、以下のように設定する。

- 「基本」タブの **[名前]** を fixation、 **[終了]** を実行時間 (秒) で 6.0 にする。 **[形状]** を正多角形、 **[頂点数]** を 12 にする。
- 「レイアウト」タブの **[サイズ [w, h] $]** を [0.1, 0.1] にする。

• instruction ルーチン (作成する)

  - フローの先頭に挿入する。
  - Grating コンポーネントを 2 個配置してそれぞれ **[名前]** を right_sample、left_sample とする。以下のように設定する。
    * 「基本」タブの **[終了]** を空白する。
    * 「テクスチャ」タブの **[マスク]** を gauss、 **[空間周波数 $]** を 1.0、 **[テクスチャの解像度 $]** を 512 にする。
    * 「レイアウト」タブの **[サイズ [w, h] $]** を [4.0, 4.0] にし、さらに以下のように設定する。
      · left_sample の **[位置 [x, y] $]** を [-5.0, 0.0] とする。right_sample の **[位置 [x, y] $]** を [5.0, 0.0] とする。
      · left_sample の **[回転角度 $]** を-10.0 にする。right_sample の **[回転角度 $]** を 10.0 にする。
  - Text コンポーネントを 3 個配置して、以下のように設定する。
    * 「基本」タブの **[終了]** を空白する。
    * 「書式」タブの **[文字の高さ $]** を適当な値 (0.5 など) にする。
    * ひとつを left_sample の下に位置するようにして「基本」タブの **[文字列]** に「反時計回り」と入力する。別のひとつを right_sample の下に位置するようにして **[文字列]** に「時計回り」と入力する。最後に残った Text コンポーネントの **[文字列]** に、反時計回りならばカーソルキーの左、時計回りならば右をできるだけ速く押して反応するように教示するメッセージを入力する。
  - Keyboard コンポーネントをひとつ配置し、「データ」タブの **[検出するキー $]** を'left', 'right' にして **[記録]** を「なし」にする。「基本」タブの **[終了]** を空白する。

- blank ルーチン (作成する)
  - フローの instruction ルーチンと trial ルーチンの間に挿入する。
  - Polygon コンポーネントをひとつ配置し、以下のように設定する。
    * 「基本」タブの **[名前]** を fixation_2、 **[終了]** を「実行時間 (秒)」で 1.0 にする。 **[形状]** を正多角形、 **[頂点数]** を 12 にする。
    * 「レイアウト」タブの **[サイズ [w, h] $]** を [0.1, 0.1] にする。
- trials ループ (作成する)
  - blank ルーチンと trial ルーチンをまとめて繰り返すように挿入する。
  - **[繰り返し回数 $]** の値を 1 にする。
  - **[繰り返し条件]** に exp05cnd.xlsx と入力する。
- exp05cnd.xlsx (条件ファイル)
  - targetSF、targetDir、correctAns、targetPos の 4 パラメータを設定する。

– 実験手続きの説明を満たすように targetSF に 7 種類の空間周波数、targetDir に 2 種類の傾きの値を入力する。targetDir と対応するように correctAns に値を入力する (targetDir が-15 の行は'left'、15 の列は'right')。targetPos はとりあえず空白にしておく。この時点でパラメータ名の行を除いて 14 行となる。

blank というルーチンではただ 1 秒間固視点を提示しているだけです。これは実験手続きの「1 秒間固視点が提示される」という点を実現するために挿入されているのですが、blank ルーチンが無くても trial ルーチンで targetStim の **[開始]** を fixation の **[開始]** から 1 秒遅らせれば実現できます (第 3 章 参照)。しかし、このテクニックを使うと後でコントラストを時間の経過とともに上昇させる処理の解説がかなり複雑になってしまいますので、今回は固視点の提示のために独立したルーチンを使用しています。

これで準備が整いました。いよいよ Python のコードを Builder に追加しますが、その前に用語の説明をしておきましょう。退屈かもしれませんが、用語を覚えておかないと後の解説がわからないのでしっかり覚えてください。

5.2 変数、データ型、関数といった用語を覚えよう

第 3 章 で Python における「名前」の話が出てきたことを思い出してください。刺激に stimulus、刺激色を表すパラメータに stim_color という「名前」を付けて、「stimulus の **[前景色]** プロパティに stim_color の値を代入する」とかいったことをしたのでした。この時に敢えて触れなかったのですが、この「名前」は一体何の名前なのでしょうか。「えっ、今『刺激』や『パラメータ』に名前を付けたって言ったじゃないか」と言われるかも知れません。確かにその通りなのですが、より正確に言うと、これらのものを収納しておく「箱」に名前をつけたのです。stimulus という名前の「箱」に Polygon コンポーネントを収納したり、stim_color という名前の「箱」に色の値を収納したりしていたのです。この名前は「箱」の名前なのですから、当然中身を入れ替えることもできます。第 3 章 の実験でループの繰り返しの度に刺激色が変化していたのは、Builder が毎回 stim_color という箱の中におさめられた値を変更してくれていたからです (図 5.3)。この箱のことを変数と呼びます。今まで「名前」と呼んでいたものは変数の名前、すなわち変数名です。

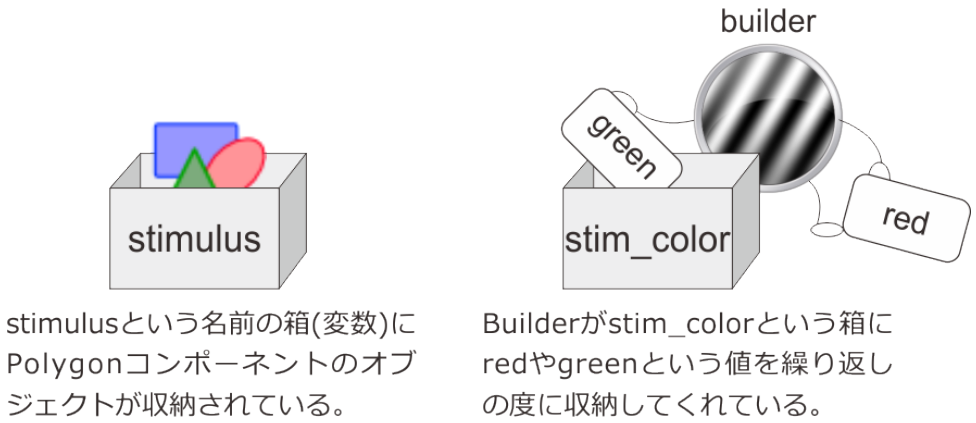


図 5.3　変数とはいろいろなものを収納しておける箱のようなもの。

Python では、変数の中にさまざまなものを収納することができます。C 言語のように変数に「型」がある言語では、整数値を入れる変数には整数値のみ収納できるといった制限がありますが、Python にはそのような制限がありません。Python で扱える型のデータであればすべて収納できます。プログラミングを学んだ経験がない方は「データの型ってなんだ？」と思われるかもしれませんね。ほとんどのプログラミング言語では、扱

うことができるデータにいくつかの「型」があって、型によって適用できる処理が異なります。 表 5.1 に基本的な Python の型を示します。ここでそれぞれの型を詳しく説明し出すとなかなか実験の作成にたどり着きませんので、ごく簡単な説明に止めています。ここで注目していただきたいのは「シーケンス」という型です。他のプログラミング言語を学んだことがある方は「配列」という名前の方がピンと来るかもしれません。第 2 章 で刺激の色を RGB 値で指定したり、刺激の位置や大きさを指定したりする時に「両端の角括弧は意味があるので忘れずに入力してください」と書きましたが、実はこれはシーケンスの一種である「リスト」を書くときの Python の文法なのです。[1.0, 0.0, 0.5] と書かれていたら、Python はこれをリストとして解釈して「最初の要素は 1.0、その次は 0.0、最後は 0.5」という具合にそれぞれの値を取り出すことができます。[ ] が欠けているとリストとして解釈することができないのでエラーとなるわけです。

表 5.1　Python の基本的なデータ型。ここでは最小限の説明に止めます。

型	概要
数値	単一の数値。Python 内部では整数のみを扱う型 (整数型) と小数点つきの数値を扱う型 (浮動小数点型) がある。
文字列	アルファベットやかな文字、漢字、各種記号等の文字が連なったもの。
シーケンス (リスト、タプル)	複数のデータを順番に並べてひとまとめにしたもの。「2 番目の要素を取り出す」といった具合に位置を指定して内容を取り出すことができる。カンマで区切られた要素を [ ] で囲んだものをリスト、( ) で囲んだものをタプルと呼ぶ (両者の違いについては「 8.15.3: リストとタプル 」 参照)。
辞書	複数のデータをひとまとめにして、キーと呼ばれる値を用いて内容を取り出せるようにしたもの。第 4 章 で実験情報ダイアログの入力値を保持していた expInfo という変数に格納されているデータがこれに該当する。

データ型の詳細については必要に応じて解説することにして、最後に関数という用語に触れておきましょう。関数と聞くと、数学の授業で習った「y は x の関数」とか「2 次関数のグラフ」といった内容を思い出される方も多いと思います。例えば「z は x と y の関数」と言った場合、x と y の値が与えられたら、その関数で定められた計算を行えば対応する z の値を得ることができます (図 5.4 左)。Python の関数とはこの関係を一般化したようなものです。例えば、「ファイルを保存する」という関数があるとします。この関数にファイル名と保存したいデータを与えると、ファイルを作成したりデータを書きこんだりといった処理が行われて、保存が成功したか否かを示す値が得られるとします (図 5.4 右)。ずいぶん数学で習った関数と違うような感じがするかもしれませんが、「値を入力すると定められた処理が行われて結果が出力される」という構図はよく似ています。このような働きを持つものを Python では関数と呼びます。関数に入力する値のことを引数、出力のことを戻り値と呼びます。実は関数にもいろいろな種類があるのですが、それについても今後必要に応じて説明することにして、関数、引数、戻り値という用語を覚えておいてください。

チェックリスト

- 変数の役割を説明できる。
- 関数の役割を、引数、戻り値という用語を用いながら説明できる。

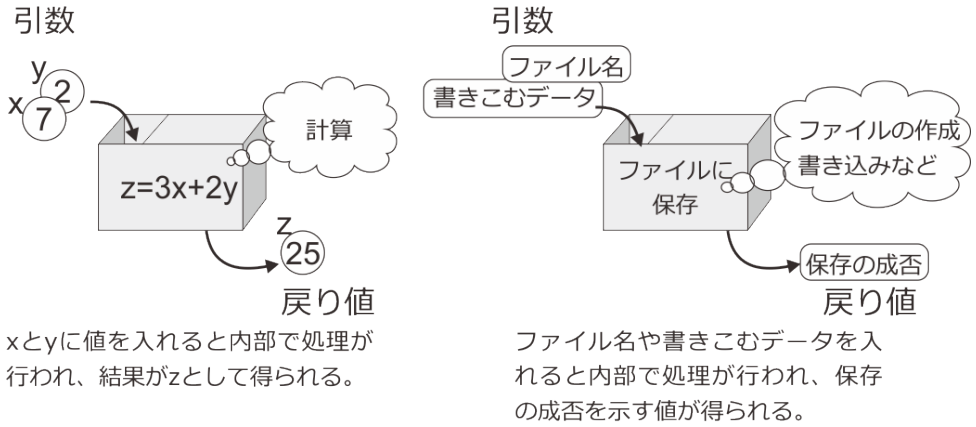


図 5.4　数学で習った関数と Python における関数。

5.3 数学関数を利用して刺激の位置を指定しよう

退屈な用語の説明はいったん切り上げて実験の作成に戻りましょう。まず、Builder で exp05a.psyexp を開いて、trial ルーチンの targetStim のプロパティ設定ウィンドウを開いてください。先ほどの作業では、まだ targetStim の **[位置 [x, y] $]** が初期値のままに残されていました。刺激の位置は固視点から半径 5.0deg の円周上で、右方向を 0 度として 0 度から 45 度間隔で 8 方向の中から無作為に選ぶのでした。試行毎に変化するのですから条件ファイルにこれら 8 種類の位置を書きこまなければいけないのですが、45 度や 135 度の方向の x 座標、y 座標の値を書きこむのが少々面倒です。半径が 5.0 ですから、45 度の方向は x 座標、y 座標とも 5.0 × 1/ $\sqrt{2}$ =3.53553…です。幸い 135 度や 225 度の方向は符号を変えたらいいだけですのでこの程度なら手入力してもいいのですが、後々半径を変更したくなったり 45 度間隔の代わりに 60 度間隔にしたくなったりした時に面倒です。なにより実験記録ファイルを見たときに刺激位置のパラメータが 3.53553, 3.53553 と書かれているよりも 45 と書かれていた方が圧倒的にわかりやすいでしょう。そこで、さっそく関数を用いて条件ファイルに 45 とか 135 とか書けば Builder が座標値を計算してくれるように改造してみましょう。

表 5.2　Builder で使用できる数学関数。min と max は任意の個数の引数を受け取ることができます。

sin(x)	正弦関数	min(x,y,z…)	x, y, z,…の最小値
cos(x)	余弦関数	max(x,y,z…)	x, y, z,…の最大値
tan(x)	正接関数	average(x)	x の平均値
log(x)	自然対数関数	std(x)	x の標準偏差
log10(x)	常用対数関数	deg2rad(x)	x を度からラジアンに変換
pi	円周率	rad2deg(x)	x をラジアンから度に変換
sqrt(x)	平方根		

視覚刺激を用いた知覚や認知の実験では、三角関数や指数関数の数学関数や円周率などの定数はしばしば用いられるので、Builder では 表 5.2 に示す関数が用意されています。今「三角関数や指数関数」と書いたばかりなのに指数関数がないじゃないか、と思われるかも知れませんが、これ以外の数学関数を使用する方法については「 5.8.1: 他の数学関数を使用する方法 」を参照してください。さて今回の目的の場合、条件ファイルに書かれた角度から座標値を計算するのですから、deg2rad( ) と sin( )、cos( ) が役に立ちそうです。まずは 45 度の sin を計算する式を作るところから始めてみましょう。数学で x の関数 f(x) に対して x に 3 を代入する

時に f(3) と書いたように、deg2rad( ) を用いて 45 度をラジアンに変換するには deg2rad(45) と書きます。座標値を計算するにはラジアンに変換した値を sin( ) および cos( ) に代入する必要がありますが、Python を含む多くのプログラミング言語では、関数の引数に他の関数 (の戻り値) を用いることが許されています。このテクニックを使うと、sin(deg2rad(45)) と書けば「まず deg2rad(45) を計算して、その結果を sin( ) に代入する」という計算ができます (図 5.5)。同様に、45 度の cos は cos(deg2rad(45)) という式で計算ができます。

図 5.5　関数の引数に関数を書くと Python が順番に計算をします。

これで 45 度の sin と cos を計算する式ができましたが、目的の座標値を得るためにはこれらの式に半径の値、5.0 を掛けなければいけません。数値同士の足し算や掛け算を行うには、算術演算子と呼ばれる記号を用います (表 5.3)。また難しそうな用語が出てきたと思われるかもしれませんが、要するに + とか-とかいった記号のことです。昔のタイプライターで×や÷といった記号を使えなかった名残で、掛け算は * 、割り算は / と書くことに決まっています。この算術演算子を利用すれば、ようやく 45 度の時の座標値を計算する式を完成させることができます。 **[位置 [x, y] $]** に書くときには x 座標、y 座標の値をこの順番に並べたリストにしなければいけないことに注意すると、以下の式が得られます。

```
[5*cos(deg2rad(45)), 5*sin(deg2rad(45))]
```

表 5.3　Python の算術演算子

x + y	x 足す y	-x	x の符号を反転
x - y	x 引く y	x % y	X を y で割った余り
x * y	x 掛ける y	x ** y	X の y 乗
x / y	x 割る y	x // y	x 割る y(割り切れない場合は切り上げ)

これでターゲットの方向が 45 度の時の式が完成しましたが、今回の実験では条件ファイルから角度を読み込

んで座標値を計算しないといけません。幸い、関数の引数には変数を書くことができるので、単に先ほどの式の 45 を変数名に書き換えるだけでこの目標は達成できます。先ほど条件ファイルを作成した時に空白のまま残しておいた targetPos を利用することにしましょう。Builder を開いて trial ルーチンの targetStim の **[位置 [x, y] $]** に以下のように入力し、「繰り返し毎に更新」にしてください。

```
[5*cos(deg2rad(targetPos)), 5*sin(deg2rad(targetPos))]
```

Builder での変更を終えたら条件ファイル exp05cnd.xlsx を開いてください。すでに 14 行分の条件が入力されていますが、これらの 14 条件を 8 種類 (0, 45, 90, 135, 180, 225, 270, 315) の targetPos すべてに対して実行するように変更してください。最終的に 14 × 8=112 行の条件ファイル (パラメータ名の行を除く) になるはずです。

以上で刺激の位置を試行毎に決定することができるようになりました。あとは時間の経過とともにだんだんコントラストが高くなる刺激を実現できれば実験は完成します。また新しい話が出て来るので、ここでいったん解説を区切ることにしましょう。

チェックリスト

- Builder で使用できる数学関数を挙げることができる
- 関数の引数に別の関数の戻り値を使うことができる。
- 条件ファイルから読み込んだパラメータ値を関数の引数に使うことができる。
- Python の算術演算子 8 つ挙げてその働きを説明することができる。

5.4 ルーチン開始後の時刻を取得して刺激を変化させよう

時間の経過とともに刺激のコントラストを上昇させるには、「現在の時刻に応じて **[前景色]** の値を変更する」ことと、「 **[前景色]** の変化を刺激に反映させる」ことが必要です。第一の点については、t という Builder の内部変数を利用します。内部変数とは Builder が正常に動作するためにユーザーから見えない内部で自動的に作成する変数で、t には現在のルーチンが開始されてから何秒経過したかが保持されています。第二の点については、各プロパティの値を入力する欄の右側のプルダウンメニューで「フレーム毎に更新する」を選択することで実現できます。ごくシンプルな実験を作成してこれらのことを確認してみましょう。

一旦 exp05a.psyexp を保存して、新規に実験を作成してください。trials ルーチンに Text コンポーネントをひとつ配置して、Text コンポーネントの **[終了]** を「実行時間 (秒)」にして値に 10.0 と入力して **[文字列]** に $t と入力してください。そして、 **[文字列]** の右端のプルダウンメニューを「フレーム毎に更新」にしましょう (図 5.6)。これだけで準備は OK ですので、適当な名前で保存して実行してみてください。スクリーン中央に、ものすごい速さで 0.0 から 10.0 に向かって上昇していく数値が表示されるはずです。試しに「フレーム毎に更新する」を今までの章で使ってきた「繰り返し毎に更新」に変更して、0 から全く値が変化しないことを確認しておいてください。なお、小数点以下 2 桁目まで表示したいなど、小数の表示を変更する方法については第 12 章 で扱います。

一般論として、リアルタイムに刺激の色や位置といったプロパティを変更すると、PC のグラフィック機能への負担が大きくなります。ですから「更新しない」や「繰り返し毎に更新」に設定された刺激については、Builder はルーチン開始時に刺激を作成した後、一切プロパティ値を変更しません。近年の高性能な PC なら

ば、刺激数が数百、数千個でもない限り全て「フレーム毎に更新する」にしてもほとんど問題なく処理できるでしょうが、実験中は少しでも PC に無用な処理はさせないようにして PC の処理遅れなどを防ぎたいところです。実験の性質上どうしてもリアルタイムに値を変化させないといけないプロパティに関してだけ「フレーム毎に更新する」を設定するようにしましょう。

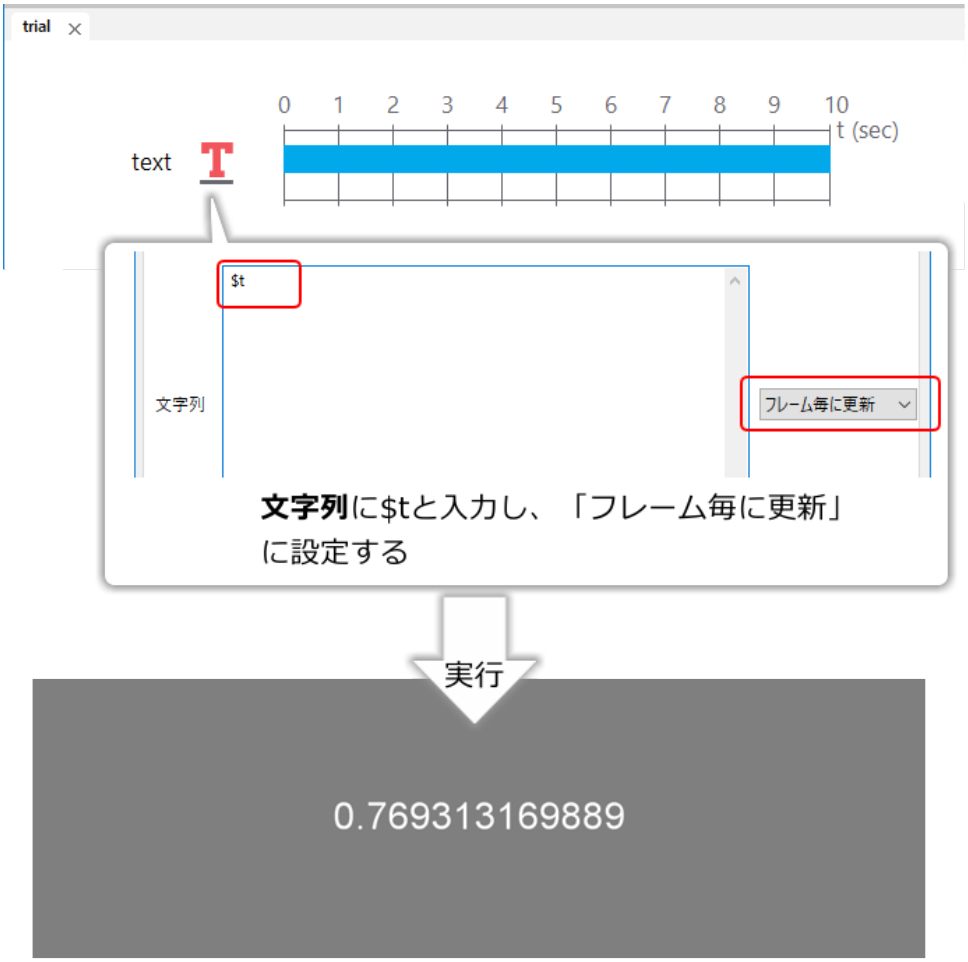


図 5.6　Builder の内部変数 t のテスト。画面上にルーチン開始から経過した時間が表示されます。

Builder の内部変数 t と「フレーム毎に更新する」を用いれば、時間と共にコントラストを変化させることは難しくありません。Builder で exp05a.psyexp を開いて trial ルーチンの targetStim のプロパティ設定ダイアログを開いてください。ルーチン開始時に **[前景色]** が$[0.0, 0.0, 0.0] で、一定の速度で上昇して 6.0 秒経過した時点で$[1.0, 1.0, 1.0] になればよいのですから、$[t/6.0, t/6.0, t/6.0] とすれば目的を達成できるはずです。忘れずに「フレーム毎に更新する」の設定もしてください。これで実験が完成しました。

実験を実行したら、targetSF の値別に反応時間の平均値を計算してみてください。ひとつの targetSF の値に対して targetDir が 2 種類、targetPos が 8 種類で 16 試行あるはずです。平均値を計算したら、横軸に targetSF、縦軸に反応時間の平均値をプロットしてみましょう。実験に使用したモニターの平均輝度や部屋の照明などによって結果は変化しますが、targetSF が 1.6 か 3.2 辺りで反応が最も速く、最小値の 0.2 や最大値の 12.8 では反応が遅くなります (図 5.7 左)。反応時間が遅いという事はそれだけターゲットのコントラストが高くならないと刺激の傾きが判断できないということですから、ターゲットの空間周波数が高すぎても低すぎても視覚の感度が低下することがわかります。縦軸を反応時間の逆数にすると、一般的な空間周波数別の感度のグラフの形状に近くなります (図 5.7 右)。

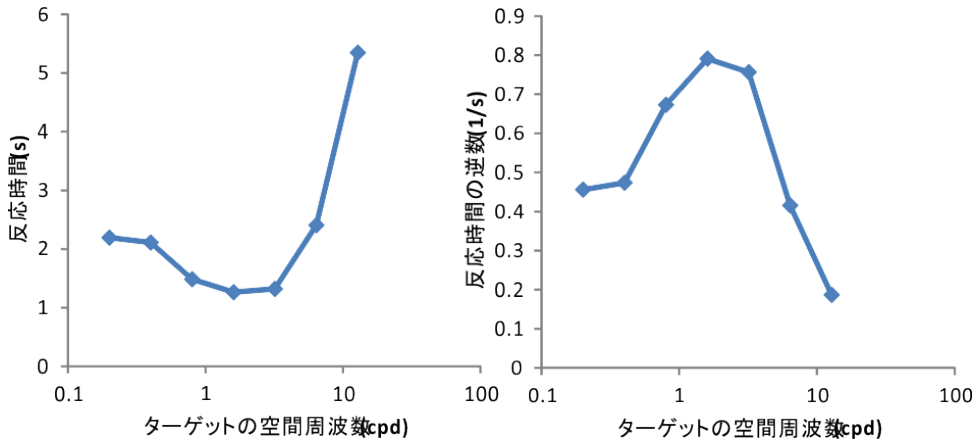


図 5.7　実験結果の例。左は反応時間、右は反応時間の逆数を縦軸にプロットしています。横軸が対数軸になっている点に注意してください。

ここでこの章の内容は一区切りですが、次の話題に移る前にひとつ補足します。Builder では現在のルーチンが開始してから経過した時間を t という変数に保持していると述べましたが、同時にルーチンが開始してからのフレーム数を frameN という変数に保持しています。 **[開始]** や **[終了]** をフレーム数で指定している場合は、t よりも frameN を使用する方が便利でしょう。フレーム数による **[開始]** や **[終了]** の指定については「2.10.6: 時刻指定における frame について 」を参考にしてください。

チェックリスト

- 現在のルーチンが開始してから経過した時間を保持している Builder の内部変数を答えられる。
- 現在のルーチンが開始してから描画したフレーム数を保持している Builder の内部変数を答えられる。
- 現在のルーチンが開始してから経過した時間の値を用いて、時間の経過とともに色や位置が変化する実験を作成することができる。

5.5 実験情報ダイアログから実験のパラメータを取得しよう

この章で目指した実験はすでに完成していますが、応用問題ということで少し改造してみましょう。 第 4 章で実験情報ダイアログから条件ファイル名を取得しましたが、同様に刺激の位置や色といったプロパティ値を取得することが可能です。ただ、実験情報ダイアログで数値を入力してプロパティ値として使う場合は少し工夫が必要です。この点を解説するために、exp05a.psyexp を改造して taregtStim の大きさと、固視点からの距離を実験情報ダイアログで変更できるようにしてみましょう。exp05a.psyexp を開いて exp05b.psyexp という別名で保存し直してください。

図 5.8 に作業の概要を示します。実験情報ダイアログに追加する項目のうち、targetStim の大きさを入力する項目を eccentricity、固視点からの距離を入力する項目を target size という名前にすることにしましょう。ちなみに eccentricty とは、視覚刺激が視野中心からどれだけ外れた位置にあるかという意味です。また、target size という項目名にはスペースが入っていますが、実験情報ダイアログの項目名は Python の変数名でなくてもかまわないのでこのようにスペースが入っていてもエラーになりません。方法がわからない人は 第 4 章 を復習しましょう。

続いて targetStim のプロパティ設定ダイアログを開いて、 **[位置 [x, y] $]** と **[サイズ [w, h] $]** で実験情報ダイアログの値を参照するように変更します。第 4 章 の内容を思い出すと、expInfo['eccentricity'] と書けばその値が取り出せるはずです。取り出した値は **[位置 [x, y] $]** に入力済みの式の 5 に相当しますから、5 を expInfo['eccentricity'] に書き換えればよいはずです。

```
[expInfo['eccentricity'] * cos(deg2rad(targetPos)),
 expInfo['eccentricity'] * sin(deg2rad(targetPos))]
```

同様に、target size の値は expInfo['target size'] と書けば取り出せます。この値は **[サイズ [w, h] $]** の 4.0 に対応しますから、 **[サイズ [w, h] $]** を以下のように書き換えます。

```
[expInfo['target size'], expInfo['target size']]
```

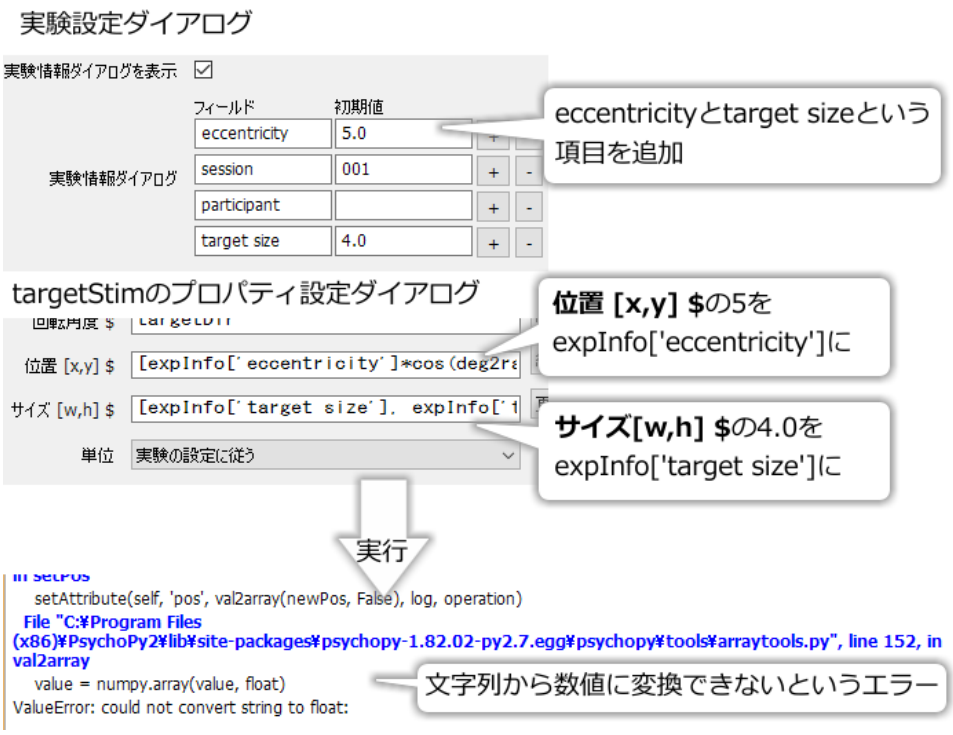


図 5.8　実験情報ダイアログから数値を読み込んでプロパティに設定するとエラーが表示されます。実験情報ダイアログに入力されたものが数値ではなく文字列となっているのが原因です。

これで改造は終了のはずなのですが、いざ実験を実行してみると、教示画面が終わってターゲットが提示される直前にエラーで停止してしまいます (図 5.8 下)。英語のエラーメッセージで難しそうですが、一番下の行を読むと could not convert string to float と書かれています。string とは文字列、float とは浮動小数点数 (表 5.1) のことですから、文字列を浮動小数点数に変換できなかったということです。どういう事かといいますと、例えば eccentricity に 4.0 と入力されていた場合、Builder には 4.0 という数値ではなく「4」、「.」、「0」という三文字の文字列に見えているということです。人間としては「数値を入力したいに決まっているだろう、そのぐらい察してくれよ」と言いたくなりますが、Builder はそのような配慮はしてくれません。仕方がないので、明示的にこれは数値として解釈しなさいと Builder に教えてなければいけません。そのために使用できるのが Python のデータ型変換関数 (表 5.4) です。float( ) 関数は、引数に与えられたデータを浮動小数点数に変換して返します。データは整数型のように浮動小数点数ではない数値でも構いませんし、浮動小数点数として解釈できる文字列でも構いません。浮動小数点数として解釈不可能なデータが与えられるとエラーになります。こ

の float( ) 関数を用いると、eccentricity の項目に入力された値を数値に変換することができます。

表 5.4　Python におけるデータ型変換関数。Builder で必要と思われるものだけを抜粋しています。

関数	概要
int(x)	x を整数に変換します。x が小数だった場合、小数点以下の値は切り捨てられます。
float(x)	x を浮動小数点数に変換します。
str(x)	x を文字列に変換します。
list(x)	x をリストに変換します。
tuple(x)	x をタプルに変換します

```
float(expInfo['eccentricity'])
```

これを **[位置 [x, y] $]** の式に当てはめると以下のようになります。長くて読みづらいですが頑張って先ほどエラーとなった式と見比べてください。

```
[float(expInfo['eccentricity']) * cos(deg2rad(targetPos)),
 float(expInfo['eccentricity']) * sin(deg2rad(targetPos))]
```

同様に、 **[サイズ [w, h] $]** の式も以下のように訂正しましょう。

```
[float(expInfo['target size']), float(expInfo['target size'])]
```

訂正が終わったら実行してください。今度こそ、実験情報ダイアログに入力した値でターゲットの大きさや固視点からの距離を変更できるはずです。

これで改造は完了ですが、実験情報ダイアログについて触れたついでに xlxs 記録ファイルへの出力についてひとつ補足しておきます。exp05b.psyexp を実行してできた xlsx 記録ファイルを見ると、ワークシートの最後に 図 5.9 のように実験情報ダイアログに入力した値が記録されているのがわかります。分析の際に役に立つことも多いので覚えておいてください。

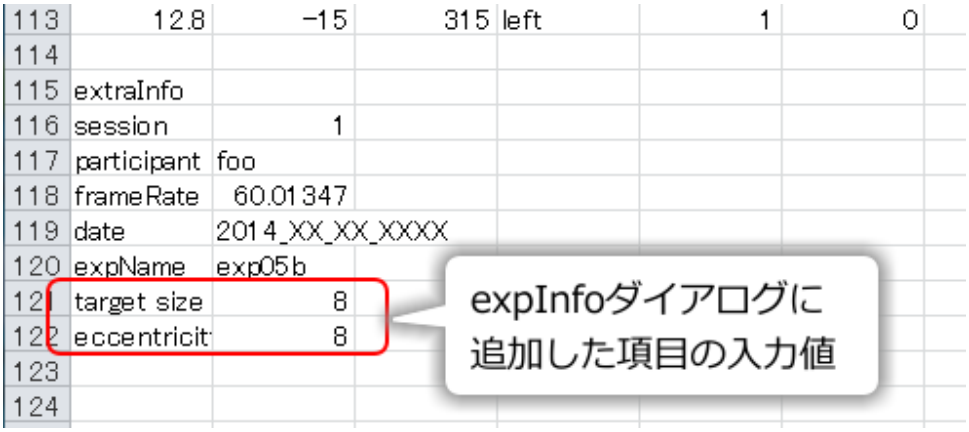


図 5.9　xlsx 記録ファイルを開くと、ワークシートの最後に実験情報ダイアログに入力された値が記録されています。

チェックリスト

- 実験情報ダイアログに入力されたデータの型を答えることができる。
- データを整数型、浮動小数点型、文字列型、リストに変換することができる。
- 実験情報ダイアログを用いて刺激の大きさや位置などの値を実験開始時に指定するように実験を作ることができる。

5.6 複雑な式には Code コンポーネントを使ってみよう

本章を終える前にあと一つ、今このタイミングで話しておきたいことがあります。本章で **[位置 [x, y] $]** に複雑な式を入力しましたが、入力欄が狭くて非常に作業しにくかったのではないでしょうか？ 特に、正しく入力したつもりなのに「Python の構文エラーがあります」と表示された場合、間違っている場所を探すのがとてもやりくかったのではないかと思います。このような複雑な式を扱う時に便利なのが Code コンポーネントです。Code コンポーネントはコンポーネントペインの「カスタム」というカテゴリにあります (図 5.10)。

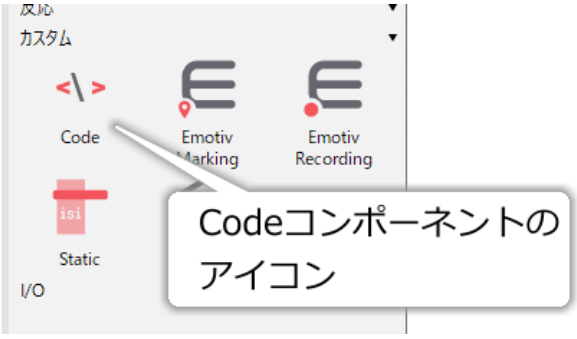


図 5.10　Code コンポーネントのアイコン。「カスタム」カテゴリにあるので注意してください。

Code コンポーネントは、実験が始まる時やルーチンが始まる時、各フレームの描画を行う時など、さまざまなタイミングで Python のコードを実行させることができるコンポーネントです。Code コンポーネントをルーチンに配置すると今までのコンポーネントと同様にプロパティ設定ダイアログが開きますが、その内容は大きく異なります (図 5.11)。まず、ダイアログ上部におなじみの **[名前]** があり、その右に **[コードタイプ]** という項目があります。さらにその右に **[無効化]** という項目がありますが、これは他コンポーネントの **[コンポーネントの無効化]** と同じ働きをするものです (「 4.9: 複雑な実験を作るときにちょっと役立つテクニック 」参照)。

[コードタイプ] は、PsychoPy 標準の設定から変更していなければ「Auto->JS」となっているはずです。もし他の値になっていれば、ダイアログのレイアウトが 図 5.11 と一致しない可能性がありますので、ひとまず「Auto->JS」してください。「Auto->JS」設定されていれば、ダイアログ中央には左右に分割された大きな入力欄があるはずです。そして入力欄の上には **[実験初期化中]** 、 **[実験開始時]** 、 **[Routine 開始時]** 、 **[フレーム毎]** 、 **[Routine 終了時]** 、 **[実験終了時]** というタブがあります。実はこのタブのそれぞれが Code コンポーネントのプロパティであり、他のコンポーネントと同様にプロパティ毎に入力欄を並べるとダイアログが大きくなりすぎてしまうので、このようにタブで切り替えるようになっています。以後、Code コンポーネントについて「 **[実験開始時]** に…と入力する」と書いた場合、「 **[実験開始時]** タブを選んでその下の入力欄に入力する」ことを表すとします。

入力欄が大きく左右に分割されているのは、 **オンライン実験** に対応するためです。Builder はもともとローカルな（つまり目の前にある）PC 上で動作する Python の心理学実験プログラムを人間の代わりに書いてくれるアプリケーションなのですが、現在のバージョンの Builder には **Pavlovia** (https://pavlovia.org/) というオン

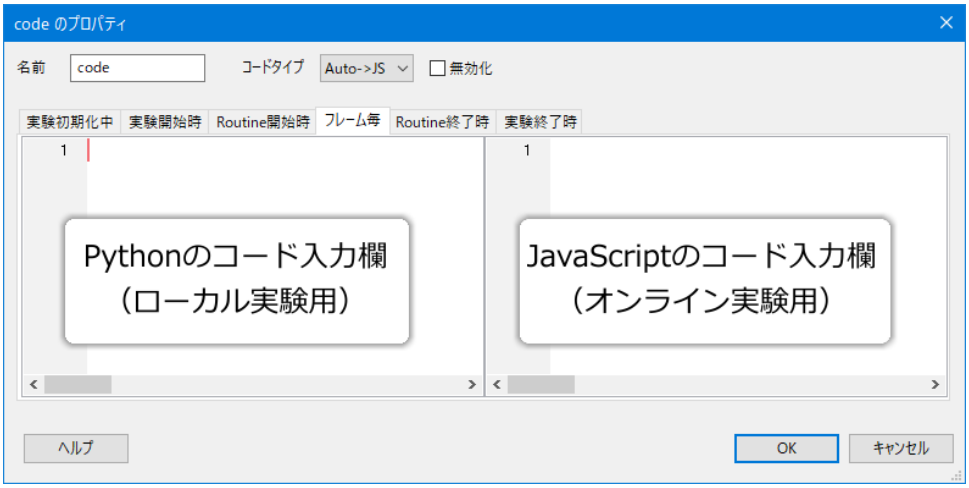


図 5.11　Code コンポーネントのプロパティ設定ダイアログ。

ライン実験サービス用のプログラムを出力する機能も備わっています。オンライン実験用プログラムというのはインターネット上のページを閲覧するために用いる（Google Chrome や Mozilla FireFox などの）ブラウザで動作するのですが、ブラウザ上では Python よりも JavaScript という言語を使う方が有利なので Builder は JavaScript でプログラムを出力します。ユーザーが Code コンポーネントに Python で書きこんだプログラムが Builder によってどのように JavaScript に翻訳するかを確認したり、必要に応じて JavaScript のコードも手作業で編集したりできるように、入力欄が左右 2 つ用意されています。左側が Python コード用、右側が JavaScript コード用です。

本書では、オンライン実験のことはひとまず忘れてローカル PC 上で動作する実験の作成方法を解説しますので、左側の入力欄を使います。右側の入力欄は無視しても構わないのですが、小さな画面のノート PC で作業する場合などは邪魔に感じるかもしれません。そのような場合、ダイアログ上部の **[コードタイプ]** を「Py」にすると、Python 用の入力欄だけをダイアログの横幅いっぱいに表示させることができます (図 5.12 上)。この作業は Code コンポーネントを新たに置くたびに行わないといけないのですが、それが面倒な場合はメニューの「ファイル」から「設定」を選んで PsychoPy の設定ダイアログを表示し、「アプリケーション」ページの「Code コンポーネントの言語」を「Py」に変更してください。これで Code コンポーネントを配置する時の標準のコードタイプが Py になります。

Code コンポーネントの解説に戻りましょう。 **[実験初期化中]** から **[実験終了時]** までの各プロパティは、コードを実験中のどのタイミングで実行したいかによって使い分けます。 表 5.5 にコードが実行されるタイミングを示します。現時点でこの表を見てもよく意味が分からないと思いますが、本書ではこれからさまざまな目的で Code コンポーネントを使っていきますので、少しずつイメージをつかんでいただければと思います。

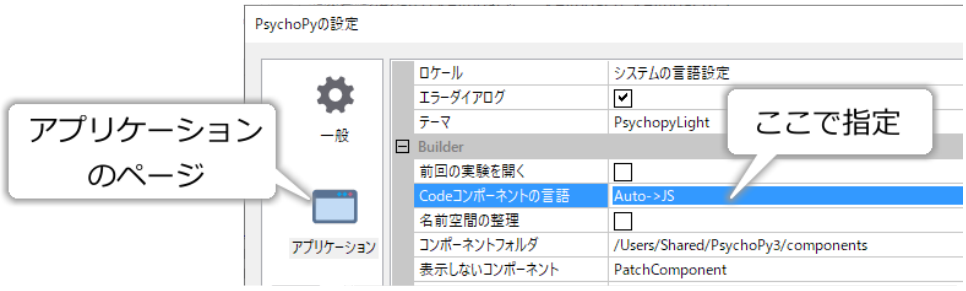


図 5.12　Code コンポーネントのプロパティ設定ダイアログ。

表 5.5　Code コンポーネントの各プロパティに記入したコードが実行されるタイミング

プロパティ	概要
[実験初期化中]	実験の初期化中、実験情報ダイアログを開く直前に実行されます。
[実験開始時]	実験情報ダイアログの OK がクリックされて刺激描画用ウィンドウが作成された直後、実験が実際に開始される前に実行されます。
[Routine 開始時]	Code コンポーネントを配置したルーチンが開始される直前に実行されます。ルーチンがループ内にある場合、繰り返しのたびに実行されます。
[フレーム毎]	Code コンポーネントを配置したルーチンの実行中、フレーム描画のたびに実行されます。つまり、毎秒 60 フレームで実験が実行されている場合、1 秒間に 60 回実行されます。
[Routine 終了時]	Code コンポーネントを配置したルーチンが終了した直後に実行されます。ルーチンがループ内にある場合、繰り返しのたびに実行されます。
[実験終了時]	フローのすべてのルーチンやループの実行を終えて、実験が終了する直前に実行されます。

それではさっそく、本章の「**[位置 [x, y] $]** の複雑で読みにくい式」を Code コンポーネントですっきりさせてみましょう。exp05b.psyexp を開いて、trial ルーチンに Code コンポーネントを配置してください。そして **[実験開始時]** に以下のように入力します。

```
ecc = float(expInfo['eccentricity'])
```

おおよそ想像できると思うのですが、この式は ecc という変数に float(expInfo['eccentricity']) の値を割り当てることを意味しています。この操作を「代入する」といい、=記号を **代入演算子** と呼びます。 **[実験開始時]** にこのコードを実行することによって、実験の以後の部分では ecc という名前で float(expInfo['eccentricity']) の値を利用できます。したがって **[位置 [x, y] $]** の式は以下のように短く書くことができるようになりました。

```
[ecc * cos(deg2rad(targetPos)), ecc * sin(deg2rad(targetPos))]
```

代入演算子にはいくつか注意してほしい点があるのですが、ここに書くと脱線が長くなるので「5.8.2: 代入演算子に関する注意点」をご覧ください。

さて、同じ要領で deg2rad(targetPos) という部分もすっきりさせてしまいましょう。

```
q = deg2rad(targetPos)
```

としてやれば、 **[位置 [x, y] $]** の式は以下のようにシンプルにできます。余談ですが、変数を q としているのは角度を表す時によく用いられるギリシャ文字 θ に対応するアルファベットが q だからです。

```
[ecc * cos(q), ecc * sin(q)]
```

問題は、この式を実験中のどの時点で実行すればよいかです。先ほどの ecc と同じ **[実験開始時]** に追加すると、taregetPos の値がまだ Excel ファイルから読み込まれていないのでエラーとなってしまいます。targetPos は trials ループによる繰り返しのたびに値が更新されるのですから、trials ループの内部で式を実行しなければいけません。さらに、刺激を提示する前の時点で値が決まっていなければいけませんので、 **[Routine 開始時]** が最も適しています。作業手順と解説が混じってわかりにくかったと思うので、ここまでの手順を整理しておくと以下のようになります。

- trials ルーチンに Code コンポーネントを配置する。
- **[実験開始時]** に ecc = float(expInfo['eccentricity']) と入力する。
- **[Routine 開始時]** に q = deg2rad(targetPos) と入力する。
- targetStim (Grating コンポーネント) の **[位置 [x, y] $]** を [ecc * cos(q), ecc * sin(q)] に変更する。

これで挿入するコードは完成ですが、あともう一つ大切なポイントがあります。ecc や q の値は、targetStim が描画される前に決定されていなければいけません。言い換えると、targetStim を描く処理を実行する前に Code コンポーネントのコードが実行されなければならないということです。この実行順序を決めているのは、ルーチンペインに並んでいるコンポーネントの順番です。「2.5: 刺激の重ね順と透明度を理解しよう」でルーチンペイン上でのコンポーネントの順番が刺激の重ね順に対応していることを学びましたが、正確にはこの順序は **コンポーネントの実行順序に対応しています** 。視覚刺激の場合は「実行する＝画面に描画する」ということであり、先に描いたものの上に後から描いたものが上描きするので、下の方にあるものほど上に描かれるのです。したがって、targetStim の描画前に Code コンポーネントを描画するには、 図 5.2 のように Code コンポーネントがルーチンペイン上で targetStim より上に置けばよいということです。

コンポーネントを並び替えたら実行してみましょう。Code コンポーネント導入前と同じように動作したはずです。以上で、動作を同じに保ったまま **[位置 [x, y] $]** の式をかなりシンプルに、意味を理解しやすくすることが出来ました。プログラムのコードというものは、書いている時は意味が分かっていても、数か月経ってから見直すとさっぱり意味がわからなくなりがちなものです。いま作成中の実験を将来利用する人のために (それは自分自身かも知れませんし、研究室の後輩や同じ分野の研究者かも知れません)、できる限り読みやすいコードを書くことをお勧めします。

以上、Code コンポーネントを使って複雑なコードを読みやすくしましたが、実はここでこのテクニックを解説しておいたことにはもうひとつ理由があります。それは、 **将来的に Builder でオンライン実験を作成する**

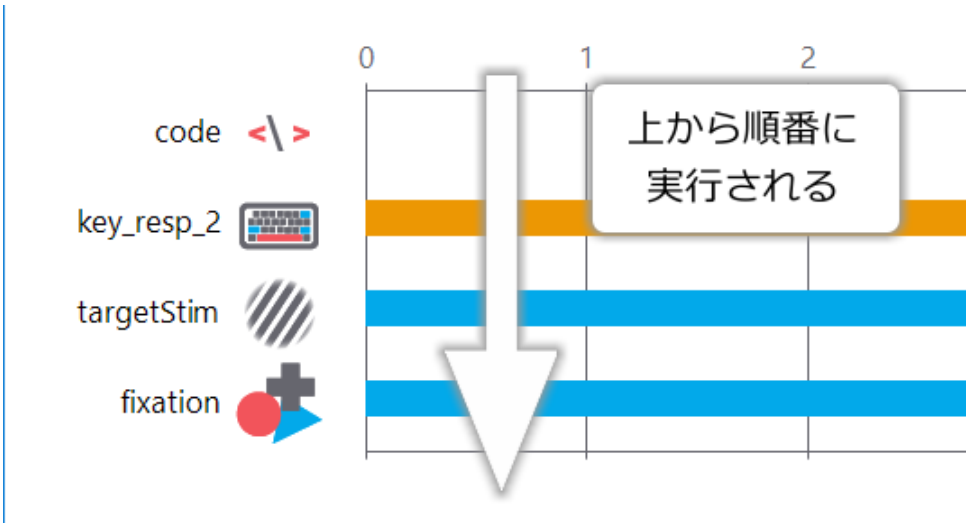


図 5.13　ルーチン上のコンポーネントは上から下の順に実行されます。刺激のパラメータを Code コンポーネントで設定するなら、Code コンポーネントは刺激描画用のコンポーネントより上になければいけません。

ことを考えているなら、単純な数値の四則演算以上の複雑な式はコンポーネントのプロパティに書くべきではない からです。先ほど述べたように、Builder が作成するオンライン実験は JavaScript という言語で書かれています。もしコンポーネントのプロパティに Python の式が書かれていると、オンライン実験として出力した時に JavaScript にうまく翻訳できない可能性があります。例えば今回の式の場合、現時点では cos() や sin() をうまく翻訳できません。 **[Routine 開始時]** のコードを

```
q = deg2rad(targetPos)
targPos_xy = [ecc * cos(q), ecc * sin(q)]
```

としておいて、targetStim の **[位置 [x, y] $]** を targPos_xy と書くようにしておけば、Code コンポーネントで JavaScript 用のコードを手作業で書くことで対応できます。Builder でのオンライン実験作成に興味がある人は覚えておいてください。

チェックリスト

- Code コンポーネントを用いて Python のコードをルーチンに配置することができる。
- Code コンポーネントのプロパティである **[実験初期化中]**、**[実験開始時]**、**[Routine 開始時]**、**[フレーム毎]**、 **[Routine 終了時]**、 **[実験終了時]** の違いを説明できる。
- Code コンポーネントに Python 用のコード入力欄だけを表示するように設定できる。
- ルーチンにおける Code コンポーネントと他のコンポーネントの実行順序を説明できる。
- 変数に値を代入する式を書くことができる。

5.7 練習問題：パラメータが適切な範囲を超えないようにしよう

この章では Builder の Python のコードを書いてみました。まだまだ解説したいことはたくさんありますが、ここで一区切りとします。最後に練習問題として、exp05a.psyexp を改造して、trial ルーチンでキーが押されるまで刺激を提示し続けるようにしてください。ターゲット出現後 6 秒以上経過した後はキーが押されるまで、targetStim の **[前景色]** はずっと$[1.0, 1.0, 1.0] の値を保ち続けるものとします。

キーが押されるまで trial ルーチンを継続し、刺激を提示し続けることはこの章まで進んだ皆さんなら問題はないはずです。この問題のポイントは、exp05a.psyexp において、trial ルーチンが 6 秒以上継続してしまうと targetStim の **[前景色]** に入力した式 ($[t/6.0, t/6.0, t/6.0]) の値が 1.0 を超えてしまう点です。一度皆さんも試してみていただければと思いますが、 **[前景色]** に入力した式を exp05a.psyexp のまま変更せずに 6 秒以上刺激を提示すると、モノクロのグレーティング刺激を提示していたはずなのに突然カラーグレーティングが提示されてしまいます。そのまま放置していると実験が正常に動作しなくなる場合もありますので、現象を確認したらすぐにエスケープキーを押して実験を終了しましょう。 **[前景色]** に入力した式の RGB 各成分の値が 1.0 を超えないようにする方法を考えてください。練習ついでに、targetStim の **[位相 (周期に対する比) $]** の式を t にして「フレーム毎に更新する」にしてみましょう。グレーティング刺激を運動させることができます。運動知覚の実験ではよく使われる刺激ですので覚えておく価値があると思います。

どうしても答えがわからないという方向けに少しだけヒント。 表 5.2 のいずれかの関数を使えば式の値が一定値を超えないように制限することができます。

5.8 この章のトピックス

5.8.1 他の数学関数を使用する方法

本文中で述べたとおり、指数関数 e^x は標準の状態では Builder に読み込まれません。Python の膨大な数学関数を利用するためには Code コンポーネントを用いてそれらの関数を Builder に読み込む必要があります。Code コンポーネントを用いて実験の最初に以下のコードを実行すると、exp(x) という x の指数関数の値を得る関数が利用できるようになります。

```
from numpy import exp
```

一般に、foo というパッケージ（またはモジュール）に含まれる bar という名前の関数等を参照する時には

```
from foo import bar
```

と書きます。Code コンポーネントを用いて読み込んだ場合は使用済み名前のチェックが効かないので十分に注意してください。もっと踏み込んだ話をしますと、異なるパッケージで同一の名前の関数が存在する場合がありますので、上記の from を用いる方法よりも

```
import foo
```

と書いて foo パッケージを読み込み、

```
foo.bar(x)
```

という具合にドット演算子を使ってパッケージ名を明示する方が安全です。

5.8.2 代入演算子に関する注意点

Python における代入はかならず「右辺の値を左辺の変数へ」という方向で行われるので、以下の式はエラーとなります。

```
x+7 = 3
```

ちょっと注意が必要なのは、以下のような式です。これは右辺の「x+7 を計算して、その結果をまた x に代入する」という意味で、刺激の位置などを一定量だけ増減させるときなどに非常によく使われます。

```
x = x+7
```

数学では=演算子を「左辺と右辺が等しい」という意味でも使うので、この文をそちらの意味で解釈しようとして「意味が分からない」という方が時々おられます。 **Python では=演算子を「左辺と右辺が等しい」という意味で使うことはありません** 。このことはよく覚えておいてください。「左辺と右辺が等しい」ことを表す演算子は次章で登場します。なお、この「計算結果を元の変数に格納する」という式は非常に良く用いられるので、Python には二項算術演算子 (x+y のように二つの値をとる算術演算子) と代入演算子を組みあわせた演算子が用意されています (図 5.14 左)。x=x+7 を x+=7 に書き換えるといった具合に二項算術演算子と代入演算子を続けて書くと、右辺に変数名を書く必要がなくなります。変数名が x などのように短い時にはあまり意味がないのですが、 図 5.14 右のように変数名が長くなった場合に「target_leftside_length から 3 を引く」ということがとても読み取りやすくなります。

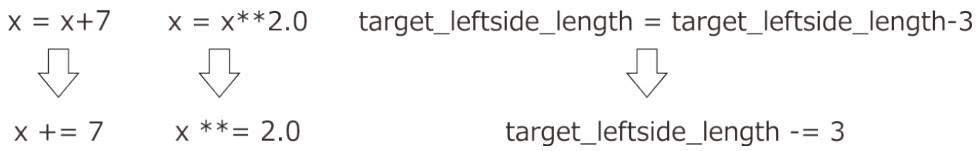


図 5.14　二項算術演算子と代入演算子の組み合わせ。変数名が非常に長い時に有効です。

第 6 章

反応にフィードバックしよう—概念識別

6.1 この章の実験の概要

ずっと視知覚の実験ばかりが続いたので、この章では概念識別の実験を取り上げましょう。私たちが新しい概念を学ぶときには講義を聞いたり書籍を読んだりといった言語を通じた手段を用いることが多いでしょう。その他にも、ある概念について「この事例は当てはまる」、「あの事例は当てはまらない」といった事例を経験することによって、概念を獲得することがあります。例えば、幼児が言語を獲得する時に「これは『くるま』じゃないよ」、「いま『くるま』がみえたね」といった事例を通じて「くるま」という概念を獲得するといった具合です。ある概念が適用される事例を正事例、適用されない事例を負事例と呼びます。この章で取り上げる概念識別の実験は、この事例を通じた概念の獲得過程を単純化したものです。

実験では 図 6.1 に示す画像を刺激として使用します。眼鏡の有無 (かけている／かけていない)、顔の形 (丸い／四角い)、目の大きさ (大きい／小さい)、眉毛の形 (上がっている／下がっている)、口の形 (口角が上がっている／下がっている) の 5 種類の次元の組み合わせで合計 2 の 5 乗=32 種類の顔画像を用います。これらの画像ファイルは 図 6.1 下に示すように、'Face'+5 桁の数値 +'.png' という名前で保存され、数値の各桁の値が 5 種類の次元の値に対応しています。

実験の最初に無意味な単語をひとつ決定します。以後、この単語を「ターゲット語」と呼びます。例えば今、ターゲット語として「リニ」という無意味な単語を選んだとしましょう。この「リニ」の正事例として、5 種類の次元のいずれを選んでその値のどちらかを割り当てます。例えば「眼鏡の有無」の「かけていない」を選んだとしましょう。そうすると、これから実施する実験では「リニ」は刺激の顔が「眼鏡をかけていない」時に「当てはまる」、それ以外の時に「当てはまらない」ということになります。

以上のことを決めたうえで、実験に入ります。実験は 1 回から複数回のセッションから成っています (図 6.2)。各セッションの最初には、先ほど決定したターゲット語と、反応方法の教示が表示されます。反応方法は「当てはまる」ならばカーソルキーの左、「当てはまらない」ならカーソルキーの右を押すことにしましょう。実験参加者が教示画面でカーソルキーの左右いずれかを押すと最初の試行が始まります。各試行では、スクリーン中央にターゲット語と顔画像が提示され、スクリーン左下に「当てはまる」、右下に「当てはまらない」と提示されます。実験参加者はこの顔画像がターゲット語に「当てはまる」か「当てはまらない」か判断して、キーを押して反応します。反応に制限時間は設けず、刺激は反応があるまで提示し続けます。反応の検出後に、反応が正解であれば「正解です」、誤答であれば「不正解です」とスクリーンに表示して正誤を実験参加者にフィードバックします。32 種類の全ての画像に対して一回ずつ判断するまで試行を繰り返し、終了したら実験参加者に正事例の条件を口頭で回答させ、実験者は筆記します。以上の手続きを 1 セッションとしま

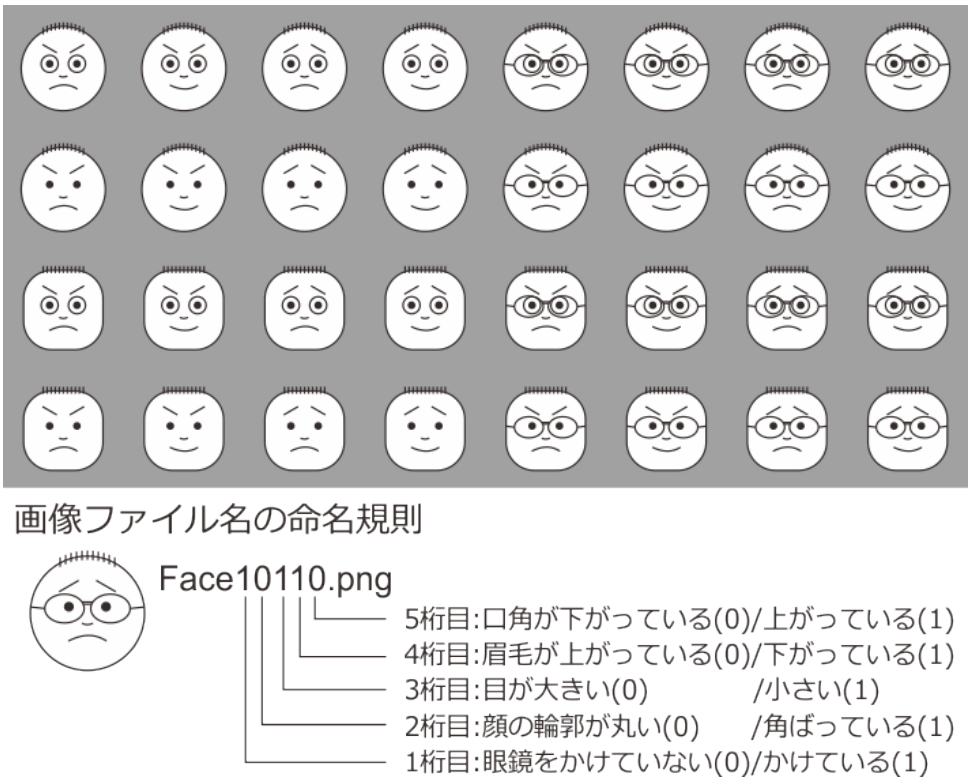


図 6.1　実験に用いる刺激。5 次元の特徴にそれぞれ 2 種類の値があり、合計 2 の 5 乗=32 種類の画像あります。

す。本来なら参加者の回答をキーボードで文章として入力してもらいたいところですが、PsychoPy で文章入力をするのはちょっと難しいので今回はこういう形を取ります ((「9.5:TextBox コンポーネントで文字を入力してみよう」も参照のこと)。実際の実験では複数セッション実施するでしょうが、複数セッションは 第 4 章 の多重ループで簡単に実現できますので、この章では 1 セッションのみを完成させることを目指しましょう。

以上が実験の概要です。実験の作成に入る前に、実験の作成に必要となる新しいコンポーネントの紹介をします。

6.2 Image コンポーネント

Image コンポーネントは **[画像]** で指定した画像ファイルをスクリーンに描画するコンポーネントです (図 6.3)。Grating コンポーネントと共通する部分が多く、Grating コンポーネントと同様に **[マスク]** 、 **[テクスチャの解像度 $]** や **[補間]** というプロパティがあります。これらのプロパティについては 第 4 章 を参考にしてください。

他コンポーネントと共通のプロパティのうち、注意が必要なのが **[サイズ [w, h] $]** と **[前景色]** です。まず **[サイズ [w, h] $]** ですが、ここを空欄にすると画像ファイルの本来の大きさで描画します。つまり、幅 640pix、高さ 480pix の画像であればそのまま幅 640pix、高さ 480pix で描かれるということです。これがどういう時に役に立つかというと、ループを使って縦横比が異なる画像ファイルを次々と表示するようなケースです。この時 **[サイズ [w, h] $]** を指定してしまうと、常に指定された縦横比で描かれるので、画像によって横方向か縦方向に引き延ばされてしまいます。ただ、この方法は height 単位のようにスクリーンの解像度に応じて刺激の大きさを調整してくれる機能と相性がよくありません。フル HD(解像度 1920 × 1080) のモニターではいい感じの大きさで表示されるのに、4K(解像度 3840 × 2160) のモニターだと小さすぎて刺激がよく見えないと

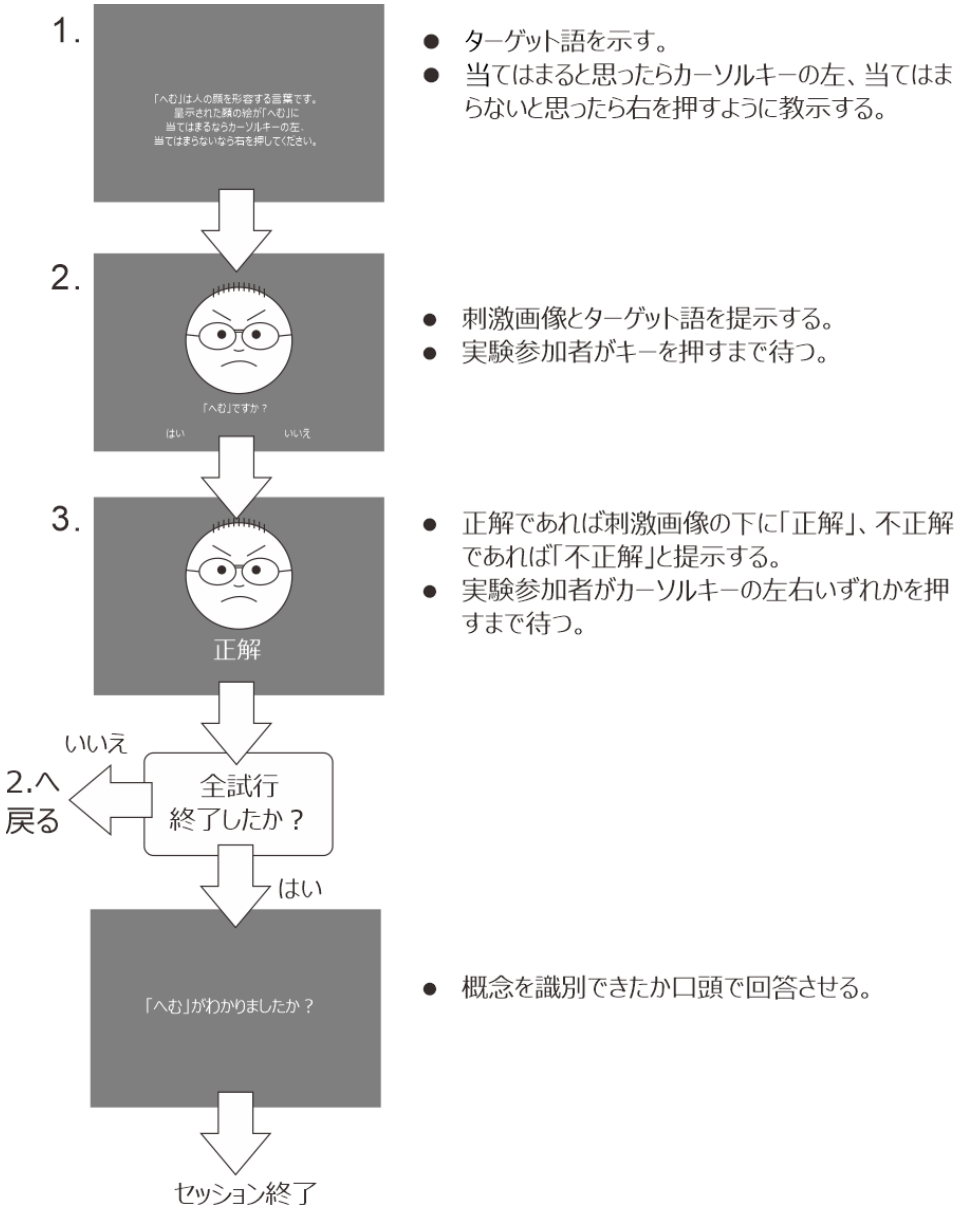


図 6.2　実験の概要。

図 6.3　Image コンポーネント

いったことが起こり得ます。実験室で、常に同じ解像度のモニターで実行するのでしたら問題にならないでしょうが、デモ用に公開する実験やオンライン実験のように、どのような解像度のモニターで実行されるかわからない実験には不向きです。

height 単位の場合にスクリーンとの相対値で画像の大きさを指定しつつ、画像ごとに縦横比を変更したい場合は、条件ファイルを使って画像ごとに **[サイズ [w, h] $]** を指定する必要があります (実行中に画像の縦横比を計算することも可能ですが、本書で想定しているレベルを超えます)。あるいは、そういう面倒なことにならないように画像の縦横比がすべて同じになるように画像を用意するのもよいでしょう。

もうひとつの要注意プロパティである **[前景色]** ですが、画像を描画する際に、ここに指定した色と画像の色の乗算がおこなわれます。つまり、画像上のある位置の色が PsychoPy 風の表現で (-0.3, 0.5, 0.7) であるときに **[前景色]** に$(-1.0, 0.0, 0.3) と指定されていたら、RGB 各成分をそれぞれ乗算して (0.3, 0.0, 0.21) として描かれるということです。これを使うと、R 成分だけを描画するとか G 成分だけを反転するといった簡単な画像処理だったら、前もって処理をおこなった画像ファイルを用意しなくても実行時に処理できるということです。 **[前景色]** の初期値は$(1, 1, 1) ですので、乗算しても何も起こらずに画像の色そのままに描画されます。

Image コンポーネントで初登場のプロパティは **[水平に反転]** 、 **[垂直に反転]** 、 **[画像]** です。 **[水平に反転]** と **[垂直に反転]** にチェックを入れておくと、それぞれ画像が左右反転、上下反転された状態で提示されます。 **[画像]** には画像ファイル名を指定します。条件ファイルと同様に、ただファイル名だけが与えられた場合には、実行している psyexp ファイルと同じフォルダからファイルを探します。当然ファイルが見つからなければエラーとなり実験は停止してしまいます。psyexp と異なるフォルダにある画像ファイルを参照する場合は、絶対パスおよび相対パスによる指定が使えます。絶対パス、相対パスと言われてもピンと来ない方もおられると思いますので、節を改めて解説しておきましょう。

チェックリスト

- 画像ファイルをスクリーン上に提示することができる。
- 画像ファイルを上下、または左右に反転させて提示することができる。

6.3 絶対パスと相対パス

パス (path) とは「小道」のことで、PC ではプログラムを実行するときに開いたり保存したりしたいファイルへたどり着くための経路を表します。私たちの身の回りのものに例えるとすれば、「住所」によく似ています。「X 県 Y 市 Z 町 1 丁目 1-1」という住所があるとして、手紙の宛先にこの住所を書いておけば、全国どこから発送しても同じところへ配送されます。PC の場合でも同様に、PC のファイルシステム（ハードディスクや USB メモリ等）でファイルの位置を特定できる「住所」があります。この住所を絶対パスと呼びます。

Microsoft Windows の場合、USB メモリを差し込むと「ドライブ F:」などのようにアルファベットが割り当てられるのは御存知のことと思います。このアルファベットをドライブレターと呼びます。この USB メモリに「psychology」というフォルダがあって、さらにその中に「report」というフォルダがあって、その中の「report01.docx」というファイルがあるとします。この report01.docx を絶対パスで表すには、以下のようにドライブレターの後にコロンとバックスラッシュを書き、以後フォルダ名をバックスラッシュで区切って記述します。

```
F:\psychology\report\report01.docx
```

日本語 Windows ではバックスラッシュは半角の円記号 (￥) で表示されますのでご注意ください (「3.11.5:$を含む文字列を提示する」を参照)。日本での住所の表記が都道府県、市町村、と大きな区分から小さな区分に向かって書かれるのと似ています。

Ubuntu などの Linux 系 OS では、バックスラッシュではなく以下のようにスラッシュでフォルダを区切ります。また、ドライブレターは使用されず、絶対パスの先頭はスラッシュです。

```
/home/user/Documents/Report/report01.txt
```

言うなれば先頭のスラッシュの前に「名前がない」フォルダがあることになりますが、この名無しフォルダの事を root と呼びます。ファイルシステムがこの名無しフォルダを根として枝が広がっていくように見えるところに由来する名称です。

絶対パスの良いところは、どの位置に目的のファイルがあるかを曖昧さなしに特定できることです。学期末のレポートの時期に「○○概論」や「△△特殊講義」といったあちこちのフォルダに report.doc という名前のファイルが散らかっていて訳が分からなくなることがありますが、絶対パスであればどのフォルダの report.doc であるかを見失うことがありません。しかし、この性質が逆に「融通の利かなさ」という欠点になる場面もあります。例えば自宅の PC で USB メモリが F: ドライブとして接続されていて、その中にある F:\Exp07\stim01.jpg という刺激画像を Builder から絶対パスで参照するようにしたとします。そして psyexp ファイルを保存して大学の実験室へ入り、実験室の PC に USB メモリを接続したら、E: ドライブとして認識されてしまったとしましょう。そうすると psyexp ファイルで参照している F:\Exp07\stim01.jpg の絶対パスは今や E:\Exp07\stim01.jpg に変わっていますので、それに合わせて psyexp ファイルを書き換えないといけません。これはあまりにも面倒です。このようなときに便利なのが相対パスによる指定です。

相対パスとは、住所の例えで言うならば、何県の話をしているのか文脈から明らかなときに県を省略して「Y 市 Z 町 1 丁目 1-1」のように書くことに似ています。しかし、住所の例えでは PC の相対パスは上手く説明できません。PC の相対パスでは、まず基準となる位置を決めて、そこから住所をどのように辿っていくかを記述します。基準となる位置のことをカレントフォルダと呼びます。今、F:\experiment\exp01 というフォルダに exp01.psyexp という Builder の実験ファイルがあるとしましょう (図 6.4)。この exp01.psyexp を実行する時には、F:\experiment\exp01 がカレントフォルダとなります。exp01.psyexp から他のファイルを探す時、絶対パスではないパス (ドライブレターや root から始まっていない書き方) が与えられると、カレントフォルダからファイルを探します。第 3 章 以降で条件ファイルを指定する時に exp01cnd.xlsx という具合にファイル名だけを書いたことを思い出してください。これは絶対パスではないので、カレントフォルダである F:\experiment\exp01 の exp01cnd.xlsx を指すことになります (図 6.4 (1))。image\stim01.png と指定されたら、カレントフォルダの中に含まれる image というフォルダの中にある stim01.png を指していると解釈されます (図 6.4 (2))。このようなカレントフォルダを基準にした指定方法を相対パスと呼びます。

普通の住所の表記と相対パスが大きく異なるのは、相対パスには階層をさかのぼる記法が用意されている点です。相対パスの中にピリオドが 2 文字だけのフォルダ (..\ ) が含まれていると、それは現在のフォルダを含んでいる上位のフォルダを指します。 図 6.4 の例では、..\が F:\experiment を指します。この記法は重ねて使用することができるので、..\..\と書くと F:\experiment のさらに上位のフォルダである F:\を指します。..\で指し示される上位フォルダのことを親フォルダと呼びます。この記法を用いることによって、カレントフォルダより上位のフォルダに含まれるファイルでも自在に指定することができます。 図 6.4 の (3) の

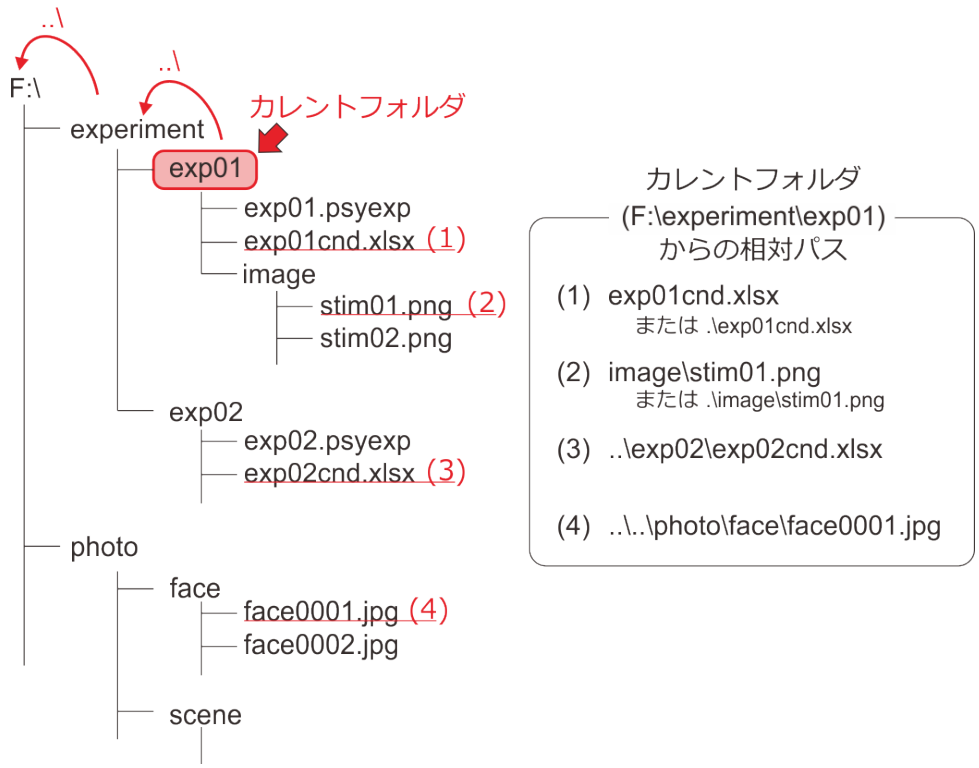


図 6.4　相対パスによるファイルの指定。

exp02cnd.xlsx へ到達するためにはまず..\で親フォルダに移動し、そこから exp02 フォルダへ下ればたどり着けますから..\exp02\exp02cnd.xlsx と書きます。同様に (4) の face001.jpg にたどり着くためには、..\..\と書いて親フォルダを二つ遡って F:\まで移動し、そこから photo フォルダ、face フォルダと下ればたどり着きますから..\..\photo\face\face001.jpg と書けばよいということです。Builder で大量の画像ファイルを刺激として使う場合や、複数の実験で同じ画像を使いまわす場合などには、この相対パスの表記法を覚えておくと必ず役に立ちます。しっかり理解しておきましょう。

相対パスのもう一つの利点は、「相対パスを使えば Python が OS によるパスの区切り文字の違いを吸収してくれる」という点です。先ほどから繰り返し例を示しているように、Microsoft Windows ではフォルダ名を区切る文字としてバックスラッシュを用います。ですから、Windows 上で動作するプログラムで相対パスを記述する時には image\stim01.png という具合に書かなければいけません。しかし、先述のように Linux ではスラッシュで区切りますので、Windows 上で動いたプログラムをそのまま持ってきても動作しません。この問題は「個人用 PC は Windows だけど大学の実験室では Linux」といった状況では困りものなのですが、ありがたいことに Windows 版の Python にはパスを Linux 流に image/stim01.png と書いても適切に解釈してくれる機能があります。ですから、Linux 流の相対パスを書いておけば Windows でも Linux でも動作する Builder の実験を作ることができるのです。「Windows 上でもスラッシュをパスの区切り文字と見なしてくれる機能は絶対パスの時にも有効なんじゃないの？」と思われる方もいるかも知れませんが、絶対パスの場合はドライブレターの問題が立ちはだかります。さすがの Python でもドライブレターを自動的に補ったり削除したりはできませんので、絶対パスで書くと OS 依存になってしまいます。

最後に、親フォルダの記法を紹介したついでに、カレントフォルダの記法も一応紹介しておきます。ピリオド 1 文字だけのフォルダ名はカレントフォルダとして解釈されます。ですから、図 6.4 の (1) は.\exp01cnd.xlsx とも書くことができます。(2) も同様に.\image\stim01.png と書くことができます。Builder を使うだけでしたら覚える必要はないのですが、今後本格的なプログラミングを学習したりする時には知っていると役に立つか

もしれません。

チェックリスト

- 画像ファイルや条件ファイル等の位置を絶対パスで指定することができる。
- 画像ファイルや条件ファイル等の位置を相対パスで指定することができる。
- OS によるパスの書き方の違いを説明できる。
- 複数の OS で実行できる Builder の実験を作成するためにはどの記法でパスを記述したらよいか答えられる。

6.4 実験の作成

コンポーネントの解説が終わったので、実験の作成に入りましょう。この章の解説では、Builder で実験を新規作成し、以下の作業を行って exp06.psyexp の名前で保存したものとします。

- 実験設定ダイアログ
  - **[実験情報ダイアログ]** に word という項目と、condition という項目を追加する。
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
- trial ルーチン
  - Image コンポーネントをひとつ配置して、「基本」タブの **[名前]** を faceImage に、 **[終了]** を空白にする。「レイアウト」タブの **[サイズ [w, h] $]** を (0.5, 0.5) に、 **[位置 [x, y] $]** を (0.0,0.1) に設定する。
  - Text コンポーネントをひとつ配置して、以下のように設定する。最終的に 3 つの Text コンポーネントが配置される。
    - 「基本」タブの **[名前]** を textYes、**[終了]** を空白にし、 **[文字列]** に「はい」と入力する。
    - 「書式」タブの **[文字の高さ $]** を 0.05 にする。
    - スクリーン左下に提示されるように、「レイアウト」タブの **[位置 [x, y] $]** を (-0.2,-0.3) に設定する。
    - textYes をコピーして、textNo、textQuestion という名前で貼り付ける。
    - textNo がスクリーン右下に提示されるように **[位置 [x, y] $]** を (0.2,-0.3) に設定する。 **[文字列]** に「いいえ」と入力する。
    - textQuestion が faceImage の下に提示されるように **[位置 [x, y] $]** を (0,-0.2) に設定する。textQuestion の **[文字列]** は後で変更するのでとりあえずそのままでよい。
  - Keyboard コンポーネントをひとつ配置し、以下のように設定する。
    - 「基本」タブの **[名前]** を key_choice とし、 **[終了]** が空白となっていることを確認する。

    * 「データ」タブの **[検出するキー $]** を'left', 'right' にする。 **[正答を記録]** をチェックして、 **[正答]** に$correctAns と入力する。

- instruction ルーチン (作成する)
  - フローの先頭に挿入する。
  - Text コンポーネントをひとつ配置し、「基本」タブの **[名前]** を textInst に、 **[終了]** を空白にする。「書式」タブの **[文字の高さ $]** に 0.05 を入力する。
  - Keyboard コンポーネントをひとつ配置し、「基本」タブ **[終了]** を空白であることを確認する。「データ」タブの **[検出するキー $]** を'left','right' にし、 **[記録]** を「なし」にする。
- feedback ルーチン (作成する)
  - フローの trial ルーチンの直後に挿入する。
  - Image コンポーネントをひとつ配置して、「基本」タブの **[名前]** を faceImage_2 に、 **[終了]** を空白にする。 **[サイズ [w, h] $]** を (0.5, 0.5) に、 **[位置 [x, y] $]** を (0.0,0.1) にする。
  - Text コンポーネントをひとつ配置し、「基本」タブの **[名前]** を textFeedback に、 **[終了]** を空白にする。「書式」タブの **[文字の高さ $]** を 0.08 にする。faceImage_2 の下に提示されるように「レイアウト」タブの **[位置 [x, y] $]** を (0,-0.28) に設定する。
  - Keyboard コンポーネントをひとつ配置し、「基本」タブの **[終了]** が空白であることを確認する。「データ」タブの **[検出するキー $]** を'left','right' にし、 **[記録]** を「なし」にする。
- report ルーチン (作成する)
  - フローの末尾に挿入する。
  - Text コンポーネントをひとつ配置し、以下のように設定する。
    * 「基本」タブの **[名前]** を textReport に、 **[終了]** を空白にする。
    * 「書式」タブの **[文字の高さ $]** に 0.05 を入力する。
  - Keyboard コンポーネントをひとつ配置し、「基本」タブの **[終了]** が空白であることを確認する。「データ」タブの **[検出するキー $]** を'space' にし、 **[記録]** を「なし」にする。
- trials ループ (作成する)
  - trial ルーチンと feedback ルーチンを繰り返すように挿入する。
  - **[繰り返し回数 $]** を 1 にする。
- 刺激画像
  - こちらのリンク (http://www.s12600.net/psy/python/ppb/chapter06/image.zip) からダウンロードする。
  - exp06.psyexp の保存フォルダに image というフォルダを作成し、その中に保存しておく。
- exp06cnd01.xlsx(条件ファイル)

  - imageFile、correctAns の 2 パラメータを設定する。
  - imageFile に 32 種類の刺激画像のファイル名を設定する。ファイル名 (FileXXXXX.png) のみでパスは入力しないこと。
  - imageFile の列でファイル名の 5 桁の数字の左から 1 桁目が 1 である行の correctAns を left に、それ以外の行の correctAns を right にする。これで眼鏡をかけている画像ファイルに対応する correctAns が left となる。
- exp06cnd02.xlsx(条件ファイル)
  - exp06cnd01.xlsx をコピーし、correctAns の列の right を left に、left を right に書き換える。これで眼鏡をかけていない画像ファイルに対応する correctAns が left となる。

以上で準備完了です。Image コンポーネントの **[画像]** をはじめ、いくつかの Text コンポーネントの **[文字列]** やループの **[繰り返し条件]** などが未設定のまま残っています。これから、以下の作業に取り組みたいと思います。

- Image コンポーネントの **[画像]** に、条件ファイルから読み込んだ刺激画像ファイル名にフォルダ名を付け足して相対パスを完成させる式を入力する。
- 実験実行時に実験情報ダイアログの word からターゲット語を取得し、instruction や trial ルーチンの Text コンポーネントの **[文字列]** 、および report ルーチンの textReport で提示する教示文に組み込む。
- 実験実行時に実験情報ダイアログの condition から条件ファイル名の番号 (01、02) を取得し、前に exp06cnd、後ろに.xlsx を結合して条件ファイル名を得る。条件ファイル名を全て入力する手間を省ける。

6.5 文字列を結合して画像ファイルのパスや教示文を作成しよう

まず、Image コンポーネントの **[画像]** プロパティを設定しましょう。 **[画像]** には読み込む画像ファイル名を指定します。ファイル名は条件ファイルの imageFile というパラメータから読み込まれますが、単に$imageFile と書くと画像ファイルが exp06.psyexp と同じディレクトリにあると見なされてしまいます。そこでフォルダ名の image/をファイル名の前に補いたいのですが、$image/imageFile とか$"image/"imageFile とか書いてもエラーになります。ではどう書けばいいかと言いますと、+ 演算子を使います。数値と数値の間に + 演算子を書けば足し算になりますが、文字列と文字列の間に + 演算子を書くと両者を連結した文字列が得られます。今回の場合は、以下のように **[画像]** に記入すれば目的を達成できます。image/を文字列として認識させるためにシングルクォーテーションまたはダブルクォーテーションで囲むことを忘れないでください。

```
$'image/' + imageFile
```

trials ルーチンの faceImage と feedback ルーチンの faceImage_2 の両方の **[画像]** にこの式を入力してください。そして、忘れずに「繰り返し毎に更新」を設定しておきましょう。

教示文へのターゲット語の組み込みも同様の方法で実現できます。expInfo['word'] という式で実験情報ダイアログから文字列を取得できるのですから、以下のような式を用いれば「これは「〇〇」ですか？」という文字列が得られます。

```
$' これは「' + expInfo['word'] + '」ですか？'
```

trial ルーチンを開いて textQuestion の **[文字列]** にこの式を入力してください。実験の実行中には expInfo['word'] の値は変化しませんので、「繰り返し毎に更新」に設定する必要はありません。この式は少々複雑なので、+ 演算子の前後にスペースを入れて見易くしてみましょう。

```
$ ' これは「'     +     expInfo['word']     +     '」ですか？'
```

三つの文字列を 2 つの + 演算して結合していることがわかります。数値演算における + 演算子が 5+3+8 という具合に 3 つ以上の値に次々と適用できるのと同じことです。ただし、数値演算では 5+3+8 でも 8+3+5 でも同じですが、文字列の + 演算では常に演算子の左側の文字列の最後尾に右側の文字列の先頭が結合されます。

最後の教示も同様に設定してしまいましょう。report ルーチンを開いて、textReport の **[文字列]** に以下の式を入力してください。 **紙面の都合上 2 行になっていますが、入力する時は途中で改行せずに 1 行で入力してください。**

```
$'「' + expInfo['word'] + '」にあてはまる画像の条件を考えて、
 実験者に口頭で答えてください。
```

続いて instruction ルーチンの textInst の **[文字列]** ですが、ここは少々解説が必要です。目標として、以下のような複数行にわたる教示文をひとつの Text コンポーネントで表示するものとします。〇〇には実験情報ダイアログの word から取得した文字列が入るものとします。

```
「〇〇」は人の顔を形容する言葉です。
提示された顔の絵が当てはまるならカーソルキーの左、
当てはまらないなら右を押してください。
```

〇〇にあたる部分を expInfo['word'] にして前後を + でつなげばよいだけのように思われるかもしれません。しかし実際に試してみると、 図 6.5 のように Builder に「構文エラーがある」と怒られて OK ボタンをクリックすることができません。

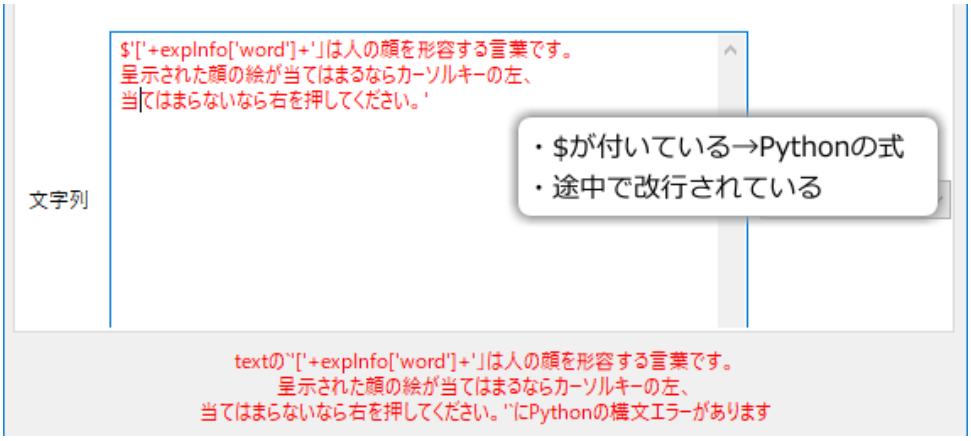


図 6.5　改行がある文と $の指定は両立しない

第 2 章 で述べたとおり、Text コンポーネントの **[文字列]** には改行を含むことができます。それなのになぜ 図 6.5 はエラーになってしまうのでしょうか。この問題の鍵は、Builder が **[文字列]** に入力されている値をどう解

釈するかにあります。 **[前景色]** の設定を思い出してください (第 3 章)。 **[前景色]** には$が付いていないから、red と書けば red という文字列が入力されていると Builder は判断するのでした。一方、$red と書けば、変数 red に格納された値が指定されていると Builder は判断すると述べました。実は後者は不正確な記述で、正確には$が書かれていると、Builder は$を除いた部分が Python の式である判断するのです。$red から$を除くと red が残り、red という式を「評価」すると、変数 red に格納された値が得られるのです。ここで言う「評価」とは、式の中に + などの演算子が含まれていたらその計算をしたり、関数が含まれていたらその戻り値を計算したりといった作業を行って、最終的な式の計算結果を得ることです。この「評価」という考え方は Builder を理解する上でとても大切です。例えば 第 5 章 で出てきた時刻に応じて色を変化させる式 $[t/6.0, t/6.0, t/6.0] では、t/6.0 が評価されて t を 6.0 で割った値に置き換えられることによって、RGB 値として解釈できるリストとなるのです。

これを踏まえて 図 6.5 に戻ります。ここには$が入力されているので Builder はこれを Python の式と見なします。Python の式としてこれを解釈すると、1 行目の最後が文字列の途中で終わってしまっています。このような書き方は Python の文法で許されていません。同様に 2 行目、3 行目も文字列が適切にクォーテーションで囲まれていませんので Python の文法を満たしていません。これでは Builder に拒否されるのは当然です。

では目指している出力を得るにはどうすればよいでしょうか。ふたつ方法がありますが、簡単な方法を紹介しましょう。Python では、シングルクォーテーションまたはダブルクォーテーションを 3 個連ねることにより、複数行にわたる文字列を表記することができます。例えば以下のように書くと 2 行に表示されます。

```
'''必要に応じて休憩を取ってください。
準備ができたらスペースキーを押して実験を再開してください。'''
```

このテクニックを使うと、textInst の **[文字列]** は以下のように書くことができます。

```
'「'+expInfo['word']+'''」は人の顔を形容する言葉です。
提示された顔の絵が当てはまるならカーソルキーの左、
当てはまらないなら右を押してください。'''
```

以上で **[文字列]** に$記号を使って Python の式を記述した場合でも複数行にわたる文字列を表示できました。この方法を知っていれば Builder を使う分には困ることはないと思いますが、今後さらにステップアップすることを考えるのならばもうひとつの方法も知っておくと役に立つかもしれません。「6.9.1: 改行文字を使った複数行の文字列の表現 (上級) 」に解説しておきますので、興味がある人は読んでください。

最後に実験情報ダイアログの condition へ入力された値から条件ファイル名を得る問題が残っていますが、ここまでの解説を理解していればもうこれ以上の解説は不要でしょう。trials ループの設定ダイアログを開き、**[繰り返し条件]** に以下の式を入力してください。これで 01 とだけ入力すれば exp06cnd01.xlsx を指定することができます。

```
$'exp06cnd' + expInfo['conditionFile'] + '.xlsx'
```

チェックリスト

- 複数の文字列を結合した文字列を得る式を書くことができる。
- 条件ファイルや実験情報ダイアログから読み込んだ文字列が組み込まれた文を提示することができる。

- Text コンポーネントの **[文字列]** に Python の式を書いた時に、表示する文字列を改行させることができる。

6.6 Python における比較演算子、論理演算子、条件分岐を学ぼう

ここまでの作業でとりあえず実験を実行できるところまでたどり着きましたが、最後の難題である「判断の正誤をフィードバックする」が残っています。フィードバックの提示は feedback ルーチンの textFeedback を用いて行います。trial ルーチンでの判断が正しければ textFeedback の **[文字列]** に「正解」、誤っていれば「不正解」と提示したいのですが、当然実験参加者が正解するか否かは実験を実行する前にはわからないので、条件ファイルでは実現できません。実はこの種の参加者の反応に応じた刺激や課題の変化は Builder が苦手とするところで、現状の Builder ではどうしても Python の文法知識、Python コードの記述が必要になります。

さて、これから「反応が正しければ『正解』、誤っていれば『不正解』と提示する」という作業を Python のコードへ変換するわけですが、このように条件に応じて行う処理を変更することを条件分岐と言います。条件分岐は

もし A が成り立つならば B を行う。さもなければ C をおこなう。

という文で表現できます。一般にプログラミング言語では「成り立つ」ことを「真 (True) である」と言い、成り立たないことを「偽 (False) である」と言います。この用語を用いると、先の文は

A が真であれば B を行う。偽であれば C をおこなう。

と書き直すことができます。Python では、この文を if, else という語を使って 図 6.6 のように書きます。if と else の後ろにコロン (:) がある点と、B、C が if や else より「字下げ」されている (行頭に空白文字がある) 点に注意してください。空白文字を何文字入れるかについての Python での文法上の取り決めは少々複雑なのですが、Python Enhancement Proposals (PEP) と呼ばれる Python の公式文書において「字下げには半角スペース 4 文字を用いる」ことが推奨されていますので、この文書では半角スペース 4 文字で統一します。第 7 章で詳しく触れますが、Python の文法では字下げが重要な意味を持っています。

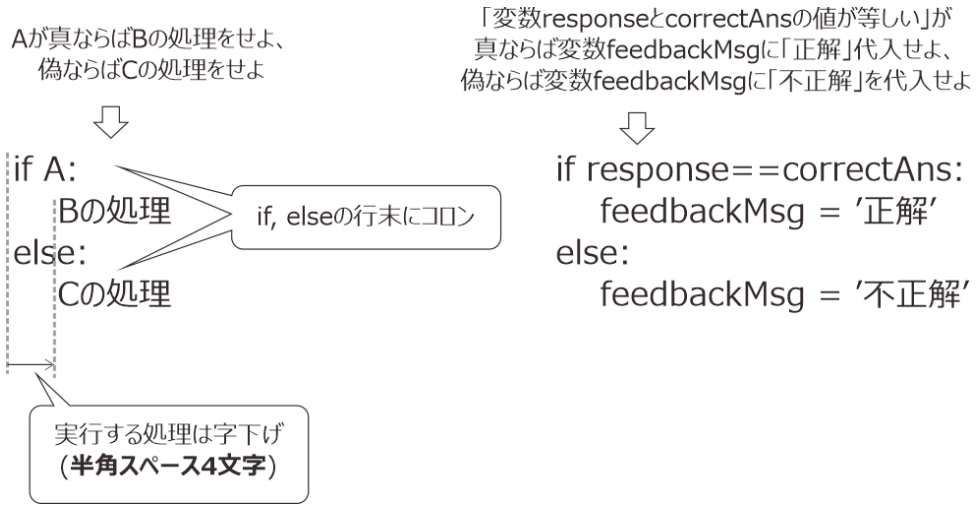


図 6.6　条件分岐 (if 文) の書式。右は実際のコードの例です。

反応が正しければ『正解』、誤っていれば『不正解』と提示する」という目標をこの if 文の形式に当てはめることができれば目的は達成されますが、どう当てはめたらよいでしょうか。この目標のままでは PC にとってはまだ抽象的すぎますので、もう少し書き直してみましょう。

・反応が正しければ
↓
・「押されたキーの名前が変数 `correctAns` の値と一致している」が真であれば

「押されたキーの名前」を Python のコードとして表現する方法はまだ解説していないので、とりあえず変数 response に押されたキーの名前が格納されているものとして書き換えを進めますと、以下の文が得られます。

・「押されたキーの名前が変数 `correctAns` の値と一致している」が真であれば
↓
・「変数 `response` の値が変数 `correctAns` の値と一致している」が真であれば

続いて「『正解』と提示する」という部分についても書き直してみましょう。提示には Text コンポーネントを使うのですから、以下のように書き直すことができます。

・『正解』と提示する
↓
・`Text` コンポーネントの「文字列」に `'正解'` と設定する

第 3 章 以降の解説では、Builder のコンポーネントのプロパティ値を実行中に変更する時には変数を用いてきました。今回もこの方法が有効でしょう。feedbackMsg という変数を用いることにしましょう。

・`Text` コンポーネントの「文字列」に `'正解'` と設定する
↓
・変数 `feedbackMsg` に `'正解'` を代入する

「『不正解』と提示する」は「『正解』と提示する」と同様ですので省略します。ここまで書き直すことができれば、Python のコードへ変換することができます。 図 6.6 の右側が実際に Python のコードに置き換えてみた結果です。 図 6.6 左側と見比べて、 図 6.6 左側の A、B、C に対応する右側のコードを見てください。右側のコードの意味が何となく分かると思うのですが、ここで「なんとなく」で済ませると後で躓くのでしっかり理解しておきましょう。

まず if の後に続く response == correctAns ですが、response と correctAns はすでに何度も出てきている Python の変数であり、その中に値が保持されています。両者の間にある ==という記号ですが、これは比較演算子と呼ばれる演算子です。==の前後に置かれた値を比較して、両者が一致していれば True、一致していなければ False という「値」を返します。True や False を「値」と言われると奇妙に感じるかも知れませんが、そういうものだと思ってください。10+5 を評価すると 15 という値が得られるのと同様に、比較演算子を含む式を評価すると True や False という「値」が得られるのです (正確に知りたい方は「 6.9.2:True と False の「値」 」参照)。比較演算子には==の他にも 表 6.1 に示すものがあります。表中の X と Y が両方とも数値である場合は特に難しいことはないと思うのですが、どちらか一方に文字列やリストが含まれている場合は話が厄介です。詳しく知りたい方は「 6.9.3: 文字列、シーケンスに対する比較演算子 」を読んでいただきたいのですが、慣れないうちは以下の点を守って使用することをお勧めします。

- 文字列やシーケンス型の比較の場合は==(等しい) と!=(等しくない) 以外使用しない
- 異なる種類のデータ型の比較 (数値と文字列の比較) はしない

表 6.1　Python の比較演算子

X == Y	X と Y は等しい	X != Y	X と Y は等しくない
X < Y	X は Y より小さい	X <= Y	X は Y 以下
X > Y	X は Y より大きい	X >= Y	X は Y 以上

比較演算子は、二つ以上同時に使うこともできます。例えば以下の例では x が 1 以上で 5 未満の時に True、それ以外の時は False になります。数値がある範囲に収まっているか否かで処理を分岐させる時に便利です。

```
1 <= x < 5
```

比較演算子について学んだついでに、もうひとつ演算子を学んでおきましょう。「刺激の位置がスクリーンの左上だった場合」といった条件で分岐させたい場合には、X 座標が負の値であること、Y 座標が正の値であることの二つの条件を同時に満たす必要があります。このような場合は論理演算子 (表 6.2) を用います。X 座標と Y 座標の値がそれぞれ X、Y という変数に格納されているのであれば、この条件は以下のように記述できます。

```
X<0 and Y>0
```

逆に、「刺激の位置がスクリーンの左上ではなかった場合」とうい条件を指定したい場合は、X 座標が 0 以上、または Y 座標が 0 以下のどちらか一方が成立すればいいのですから、以下のように記述できます。

```
X>=0 or Y<=0
```

否定演算子を使うと以下のように書くこともできます。演算子を適用する順序を指定するために ( ) を使用していることに注意してください。まず ( ) の中が評価されて、その後に not が適用されます。

```
not ( X<0 and Y>0 )
```

表 6.2　Python における論理演算子

X and Y	X かつ Y (論理積)	X と Y がともに True の時に True
X or Y	X または Y (論理和)	X と Y のいずれかが True の時に True
not X	X の否定	X が True ならば False、False ならば True

比較演算子と論理演算子についての解説はこのくらいにして、 図 6.8 の B と C に対応するコードも見ておきましょう (#以下の部分は Python によって無視されるのでコメントを書く時に用いられます)。

```
# B にあたるコード
feedbackMsg = '正解'
```

(次のページに続く)

(前のページからの続き)

```
# C にあたるコード
feedbackMsg = '不正解'
```

すでに前章で代入演算子を学んだ皆さんには、もうおわかりですね。feedbackMsg という変数に表示したい文字列を代入しています。

これで参加者の反応に応じて表示するメッセージを変更するための Python のコードの書き方がわかりました。残された問題は、この節の解説では「変数 response に格納されているものとする」と仮定した反応キー名をどうやって取得するかです。ここでまた新たな概念が登場するので節を改めましょう。

チェックリスト

- 条件に応じて処理を分岐させる Python コードを書くことができる。
- 数値の大小や一致・不一致に応じて処理を分岐させることができる。
- 文字列の一致・不一致に応じて処理を分岐させることができる。
- Python の比較演算子を 6 つ挙げてそれらの働きを説明することができる。
- Python の論理演算子を 3 つ挙げてそれらの働きを説明することができる。

6.7 オブジェクトのデータ属性を利用して反応にフィードバックしよう

いよいよこの章の実験の完成が近づいてきました。反応キー名を得るコードですが、これは知っていないといくら考えてもわからないことなので、解答から始めます。key_choice という名前の Keyboard コンポーネントから反応キー名を得るには以下のように書きます。

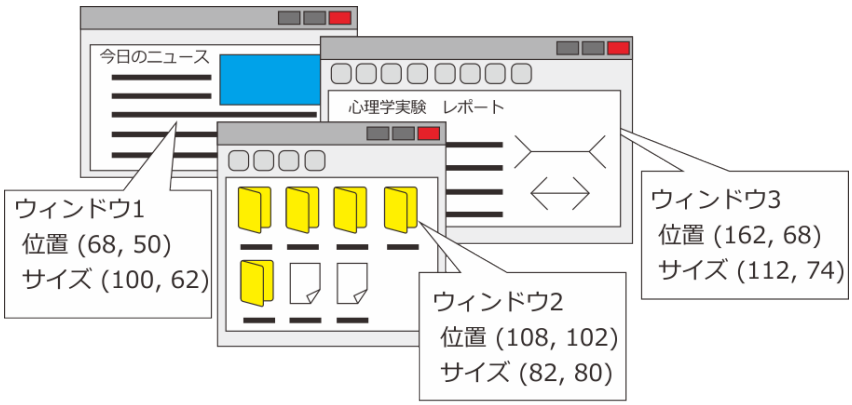
```
key_choice.keys
```

プログラミングの経験がない方には難しい話が続きますが、この部分を理解していないと次章以降の内容を理解できません。じっくり解説しますので、頑張ってついてきてください。

この key_choice.keys という式ですが、Python の文法では名前にピリオドを含むことはできませんので、key_choice と keys という名前がピリオドで結ばれているはずです。key_choice というのはルーチンに配置している Keyboard コンポーネントと同じ名前なので。恐らく Keyboard コンポーネントと関係がある「何か」だというのは容易に想像できるのではないかと思います。この key_choice というのは変数で、その中には Builder のキーボード反応計測用オブジェクト (psychopy.event.BuilderKeyResponse：以下 BuilderKeyResponse) というモノが入っています。オブジェクトとは、コンピュータ上で扱うさまざまな対象、キーボードやマウスといった物理的なものから各種ソフトウェアの動作のために用いられるデータなどをプログラミング言語上から扱いやすくするための仕組みです。

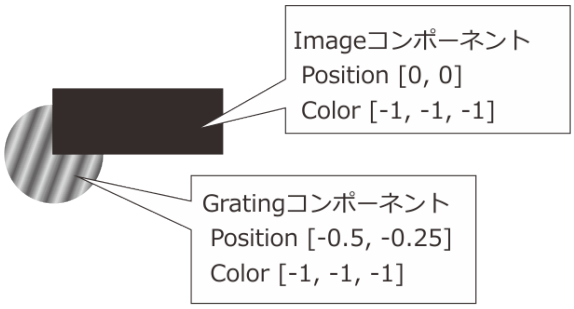
とても抽象的な概念でイメージしにくいと思いますので、具体的な例を挙げましょう。図 6.7 は web ブラウザと文書作成ソフト、フォルダ内容表示の 3 つのウィンドウを開いている様子を示しています。これらのウィンドウは右上のボタンをクリックしてウィンドウを閉じたり最大化、最小化したりといった操作や、ウィンドウの枠をドラッグして位置や大きさを変更する動作は共通しています。しかし、ウィンドウを動かしてしま

うと今まで別ウィンドウに隠されていた部分が見えるようになるので OS はウィンドウの描き直しを行うのですが、描き直しの方法は web ブラウザや文書作成ソフトなど、個々のウィンドウで異なるでしょう。このように、コンピュータ上で扱う対象には共通化できる部分と、固有の部分があります。この共通化できる部分を共通化するための仕組みがオブジェクトです。実は、オブジェクトという考え方はここまで解説してきた Builder のコンポーネントにも利用されています。例えば刺激に対応するコンポーネントにはいずれも **[位置 n [x, y] $]** や **[サイズ [w, h] $]** というプロパティがあり、これらを使ってスクリーン上のどの位置にどのくらいの大きさで描画されるかを指定することができました。 **[前景色]** プロパティで色を指定することもできましたが、Image コンポーネントと Grating コンポーネントでは同じ値を **[前景色]** に指定してもスクリーンに描かれる刺激の色は全く異なりました (図 6.7)。こういった刺激間の共通点や相違点を効率よくするために、Builder の内部ではオブジェクトが使用されています。



共通点　ウィンドウの右上のボタンで閉じたり隠したりする動作
ウィンドウの枠をドラッグして位置や大きさを変える動作

相違点　位置やサイズの値
ウィンドウの内容を描き直す動作
ウィンドウ内部でマウスをクリックしたりキーを押したときの動作



共通点　Positionで指定した位置が刺激の中心となるように描画する

相違点　PositionやColorの値
Colorの値に応じて塗りつぶす方法

図 6.7　オブジェクトの例。個々のウィンドウはそれぞれ固有の位置や大きさを持ちますが、閉じたり移動させたりといった操作方法は共通しています。これまでに扱った Builder の刺激にも、それぞれに共通する点や異なる点があります。

Python におけるオブジェクトでは、こういったさまざまな対象を表現するために、データ属性とメソッドと呼ばれるものを用います。Builder の刺激オブジェクトの例を用いると、データ属性とは **[位置 [x, y] $]** や **[サイズ [w, h] $]** のように、それぞれの刺激で固有の値をとるデータのことです。メソッドについては詳しくは

次章で触れますが、それぞれの刺激をスクリーンに刺激を描画する手続きのように、そのオブジェクトに対する操作をおこなうものです。それぞれのオブジェクトのデータ属性やメソッドの定義をクラスと呼び、クラスに従って生成されたオブジェクトを該当するクラスのインスタンスと呼びます。 図 6.8 はデータ属性、メソッド、クラス、インスタンスの関係を示しています。Grating クラスはグレーティング刺激を描画するためのクラスで、**[位置 [x, y] $]**、**[回転角度 $]**、**[テクスチャ]** に対応する Position、Orientation、Texture といったデータ属性を持っています。また、スクリーンに描画を行うための draw というメソッドを持っています。グレーティング刺激をスクリーンに 2 個描画するために、Grating クラスのインスタンスを 2 個生成して、それぞれ gratingPatch と stripePatch という変数に格納しました。各インスタンスのデータ属性に値を代入する時には、gratingPatch.Position = (0,0) という具合に、変数名とデータ属性名をドット演算子 (.) で結合して記述します。ドット演算子を用いることによって、どちらのインスタンスに代入するのか混乱することがありません。draw メソッドを用いて刺激を描画する時には、gratingPatch.draw( ) という具合にやはりドット演算子を使って変数名とメソッド名を結合して記述します。メソッドは関数と同様に引数をとることができますので、( ) が draw の後に付きます。draw メソッドでは各インスタンスのデータ属性の値を用いて刺激の位置や波形が決定されて刺激が描画されます。

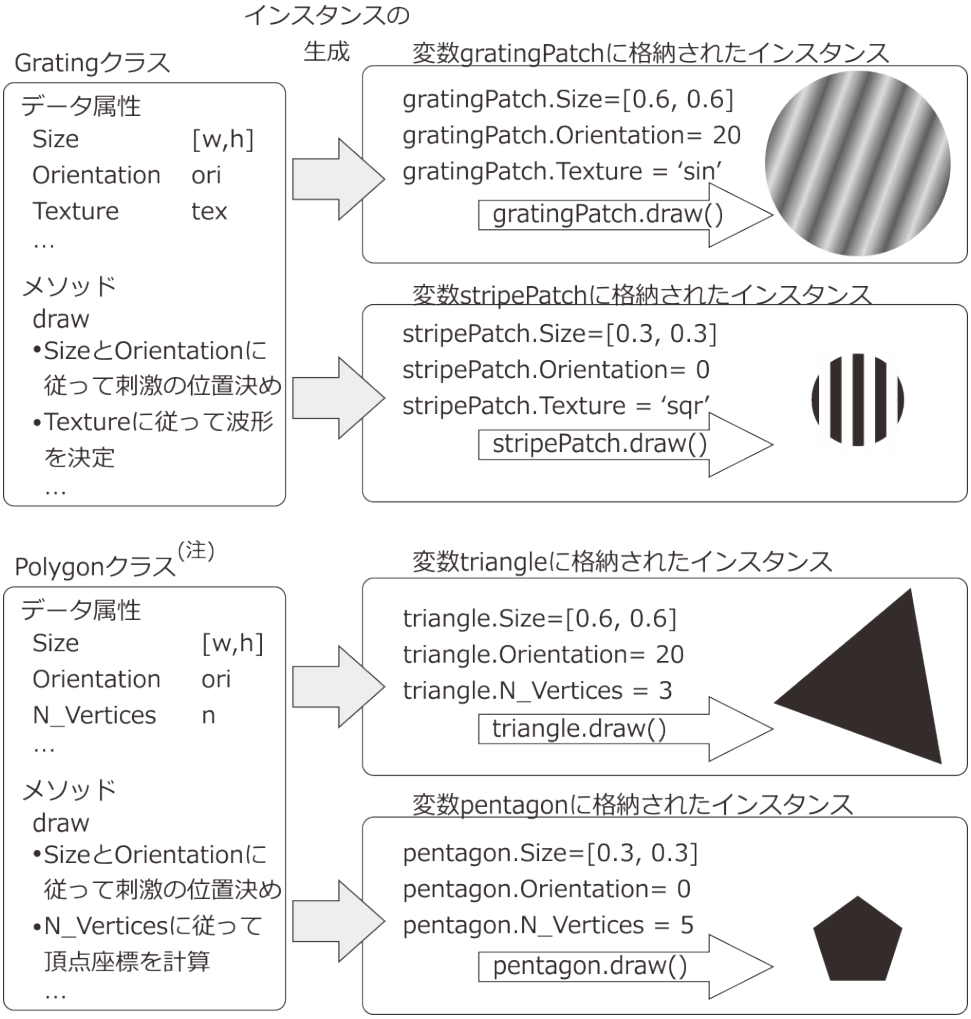


図 6.8　データ属性とメソッド、クラスとインスタンス。データ属性の値は個々のインスタンスで異なります。メソッドを呼ぶと自分のクラスで定義されているメソッドが呼び出されるので、異なるクラス間で同名のメソッドがあっても混乱しません。

グレーティング刺激に追加して、多角形の刺激を描画する Polygon クラスを用いて三角形と五角形を 1 個ずつ

描画するとします。Polygon クラスは Grating クラスと似ていますが Grating クラスには無い N_Vertices というデータ属性を持っています (**[頂点数]** に対応)。また、Grating クラスと同様に draw というメソッドを持っていますが、その処理内容は異なります。多角形を 2 個描画するのですから、Polygon クラスのインスタンスを 2 個生成して triangle と pentagon という変数に格納します。triangle と pentagon のデータ属性に適切な値を設定して、draw メソッドを呼びます。triangle に格納された Polygon を描画する時には trialgle.draw() と記述しますが、このように変数名とメソッド名をドット演算子で結合して記述することによって、この draw( ) は Polygon クラスのオブジェクトの draw メソッドであることが Python にはわかります。ですから、Grating クラスに同名のメソッドがあっても Python が両者を混同することはありません。

なお、ここで想定した Grating クラスや Polygon クラスおよびそのデータ属性は、実際の Builder で使用されているものと異なります。クラスおよびデータ属性の正確な名称および Builder との対応関係を「 6.9.4:Builder のコンポーネントと PsychoPy のクラスの対応 」に記しておきますので、詳しく学びたい方はご覧ください。Builder の通常の用途ではそこまで知っておく必要はありません。

ここまで説明して、ようやく key_choice.keys という式の意味を解説できます。key_choice には BuilderKeyResponse というクラスのインスタンスが格納されています。BuilderKeyResponse はキーボードの状態を記録するためのクラスで、 表 6.3 に示すデータ属性を持っています。この表によると、keys には押されたキー名が保持されています。key_choice という変数名は Builder の trial ルーチンに配置した Keyboard コンポーネントの **[名前]** に対応していますから、key_choice.keys という式は trial ルーチンに配置した Keyboard コンポーネントで記録したキー名です。具体的に言えば'left' か'right' のいずれかの文字列が格納されています。ですから、Code コンポーネントに入力した key_choice.keys == correctAns は実験参加者が押したキーのキー名が correctAns と一致すれば True、一致しなければ False が得られます。

表 6.3　BuilderKeyResponse クラスのデータ属性

データ属性	概要
keys	ルーチンで記録されたキー名が格納されています。キーが押されていなければ空のリスト、「最後のキー」または「最初のキー」を記録している場合は該当するキー名、「全てのキー」を記録している場合は押されたキーのキー名が順番に並んだリストが格納されています。
corr	反応が正答であれば 1、誤答であれば 0 が格納されています。キーが押される前は 0 です。
rt	キーが押された時刻が格納されています。キーが押されていなければ空のリスト、「最後のキー」または「最初のキー」を記録している場合は該当するキーが押された時刻、「全てのキー」を記録している場合は各キーが押された時刻が順番に並んだリストが格納されています。
clock	Builder が時刻を計測するための時計オブジェクトのインスタンスが格納されています。直接操作すべきではありません。

これで必要な要素はすべてそろいました。Builder で exp06.psyexp を開いて、trial ルーチンを開いて Code コンポーネントを配置してください。そして **[Routine 終了時]** に以下のコードを入力しましょう。

```
if key_choice.keys == correctAns:
    feedbackMsg = '正解'
else:
```

(次のページに続く)

(前のページからの続き)

```
    feedbackMsg = '不正解'
```

ついに exp06.psyexp の完成です。さっそく exp06.psyexp を実行してください。実験情報ダイアログの condition には 01 とだけ入力し、word に「リニ」などの無意味語をターゲット語として入力して OK をクリックしてください。条件ファイルとして exp06cnd01.xlsx が読み込まれて、最初の教示文にはターゲット語が埋め込まれているはずです。そして、刺激が出てきたら適当に反応して、眼鏡をかけている顔画像に対して「はい」を選択する (カーソルキーの左を押す) と正解、それ以外の顔画像に対して「はい」を選択すると不正解になることを確認しましょう。実験が終了したらもう一度実験を実行して、今度は実験情報ダイアログの condition に 02 と入力して OK をクリックしてみましょう。眼鏡をかけている顔画像に「はい」を選択すると不正解、それ以外の画像に対して「はい」を選択すると正解になるはずです。

以上でこの章の解説はおしまいです。if 文についてはまだまだ説明したいことがたくさんありますが、この章もずいぶん長くなってしまったので、ここで一区切りにして次の章で取り扱うことにしましょう。

チェックリスト

- クラスとインスタンスの違いを説明することができる。
- データ属性とは何かを説明することができる。
- 変数 x に格納されたインスタンスのデータ属性 foo に値を代入したり値を参照したりするときの Python の式を書くことができる。
- BuilderKeyResponse のデータ属性を参照して押されたキー名を知ることができる。

6.8 練習問題：データ属性 corr で正誤を判定しよう

この章の解説では、条件ファイルとして眼鏡をかけている顔画像が正事例となる exp06cnd01.xlsx と、眼鏡をかけていない顔画像が正事例となる exp06cnd02.xlsx の二種類しか作成しませんでした。しかし、実際に exp06.psyexp を使って概念識別の実験をするためには、他の特徴が正事例となる条件ファイルを準備する必要があります。練習問題として、「眼鏡をかけていて眉が上がっている (論理積)」顔画像が正事例となる条件ファイルと、「顔が丸い、または眉毛が下がっている (論理和)」顔画像が正事例となる条件ファイルを作成してください。

もうひとつ、if 文と BuilderKeyResponse の復習問題として、trial ルーチンの反応が正答であったか誤答であったかに応じて feedbackMsg に代入する文字列を変更する部分で、データ属性 keys を使わずにデータ属性 corr を使うように Code コンポーネントの内容を書き換えてください。 表 6.3 に示すように、データ属性 corr には反応が正答であれば 1、誤答であれば 0 が格納されています。これはほぼ答えを言っているようなものなので、これ以上はノーヒントで挑戦してください。

6.9 この章のトピックス

6.9.1 改行文字を使った複数行の文字列の表現 (上級)

Microsoft Word などの文書作成アプリケーションを使ったことがある方は、この種のアプリケーションでは図 6.9 文の末尾に矢印のような記号が (しばしば本文とは異なる色で) 描かれていることに気付いておられると思います。実は、アプリケーションの内部では、この「矢印」もひらがなや漢字、英数字と同じ「文字」の一種で、ここで「行が変わる」ことを表しています。文字として扱われている証拠（？）に、普通の文字と同様に BackSpace キーや Del キーで削除できますし、削除したら前後の文が 1 行につながりますよね。この「行が変わる」ことを表す文字を **改行文字** と呼びます。「3.11.5:$を含む文字列を提示する」で「コンピュータにおいて文字は『文字コード』と呼ばれる数値で表される」と書きましたが、改行文字にも数値が割り当てられています。例えば Windows では 0x0D0A、Linux では 0x0A です。

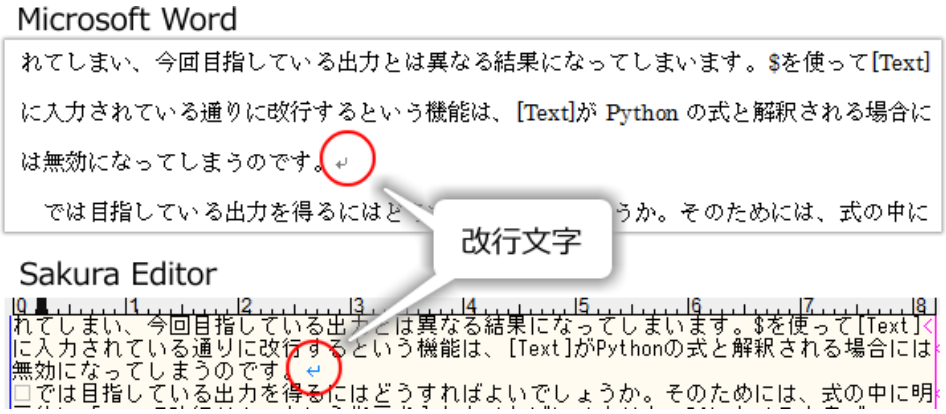


図 6.9　改行文字の表示例。行末の矢印のような記号が改行文字です。

Python を含む多くのプログラミング言語では、この改行文字をエスケープシーケンス (「3.11.5:$を含む文字列を提示する」参照) を使って\n と表すことができます。つまり

```
' 当てはまるならカーソルキーの左、\n 当てはまらないなら右を押してください。'
```

と書くと、

> 当てはまるならカーソルキーの左、
> 当てはまらないなら右を押してください

のように\n の位置で改行されて出力されます。\n を使う方法を知っていると、インターネット上で誰かが書いたプログラムを参考にして勉強するときなどにきっと役に立ちますのでぜひとも覚えておきたいところです。

さて、本章の教示も\n を使って記述することができますが、一行に書くには長すぎます。このような場合、Python の文法には

- 行末が \ のみで終わる場合は次の行につながっているとみなす
- ' 今日は' ' いい天気' のように二つの文字列の間に空白文字しかない場合は自動的に接続して ' 今日はいい天気' と同じとみなす

という規則があることを利用して

```
'「' + expInfo['word'] + '」は人の顔を形容する言葉です。\n' \
' 提示された顔の絵が当てはまるならカーソルキーの左、\n' \
' 当てはまらないなら右を押してください。'
```

と書くことができます。\は最後の行には不要（というか書いてはいけない）点にご注意ください。\n は各行の末尾以外に置いても構わないので、

```
'「' + expInfo['word'] + '」は人の顔を形容する言葉です。\n 提示された顔の絵が' \
' 当てはまるならカーソルキーの左、\n 当てはまらないなら右を押してください。 '
```

と書くこともできます。上の例とじっくり見比べて、同じ結果になることを確認してください。

6.9.2 True と False の「値」

Python では、if 文に True や False 以外の値を返す式を書くことができます。その場合、式を評価した結果が 0 であれば False、0 以外であれば True として処理されます。ですから、図 6.10 上段に示した例ではいずれの場合も if 文に続く x=7 が実行されます。このことを積極的に利用したプログラムを書くことも可能ですが、わかりにくいのでお勧めしません。

お勧めしないと言えば、Python2 における True や False は、あたかも通常の変数であるかのように値を代入することができます。ですから、True=0 などと書いて True に 0 を代入することも可能です。ただ、このようにしてしまうと True を評価すると 0 が得られて、0 は False として機能するため、図 6.10 下段の例のような非常にややこしい事態が生じてしまいます。True や False への代入は絶対に行うべきではありません。**Python3 では True と False が予約語となったため、このような代入はできなくなっています。**

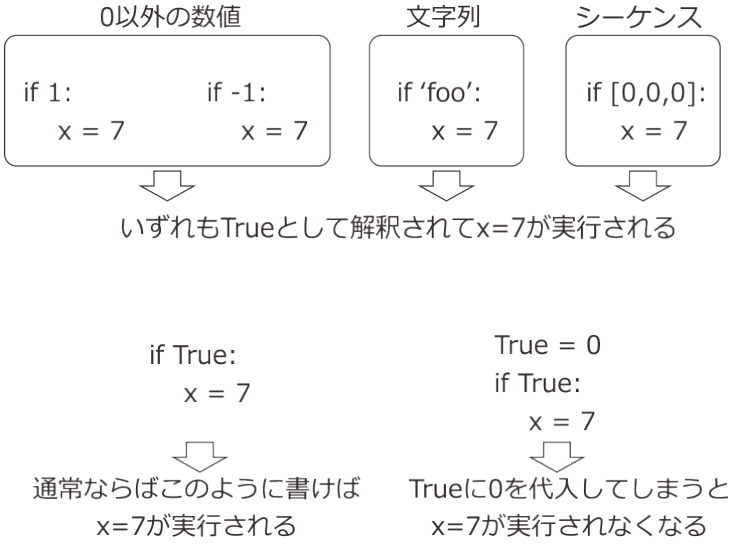


図 6.10　True と False に関する注意事項 (True/False への代入は Python2 のみ)。

6.9.3 文字列、シーケンスに対する比較演算子

本文では<や>=といった比較演算子を使って数値の大小を比較する方法について解説しましたが、Python では文字列やシーケンス型のデータに対しても比較演算子を使用することができます。これらのデータに対して比較演算子が適用された場合は「辞書順」に従って比較が行われます。

例えば memory と mind という文字列を比較するとしましょう。英和辞典の順番では、1 文字目から順番に比較していって、最初に異なる文字のアルファベット順でどちらが先に掲載されるかが決まります。memory と mind でしたら 1 文字目はどちらも m、2 文字目は e と i ですから、e の方が前です。Python では、辞書順で前にくる文字列ほど「小さい」と判定されますので、'memory' < 'mind' は True となります。'memory' > 'mind' は False です。

少し注意が必要なのは、数値や漢字が文字列に含まれる場合です。半角数字はすべてのアルファベットより「小さい」と判断されます。ですから'magical number' < '7' を評価すると False が得られます ('7' は'm' より小さいと判断される)。かな文字や漢字は文字コードに従って解釈されますので、文字コードを理解していなければ結果を予測するのは困難です。例えば Unicode で「心」は「記」より小さい値で表されますので（それぞれ 0x5fc3 と 0x8a18）' 記憶' < ' 心' を評価すると False が得られます。筆者の個人的な意見としては、このような比較は非常にわかりにくいので可能な限り使用するべきではないと思います。

シーケンスの場合は、要素を先頭から順番に比較していきます。[7,8,2] < [7,1,5,9] という比較でしたら、最初の要素の 7 は同一、2 番目の要素は 8 と 1 で左辺の方が大きい値ですので、左辺の方が大きいと判定されます。従って [7,8,2] < [7,1,5,9] は False です。シーケンスの要素に文字列が含まれている場合も、同様に個々の要素を先頭から順番に比較します。['theory', 7, 'mind'] > ['theory', 7, 'memory'] でしたら最初と 2 番目の要素は同一、3 番目の要素は memory より mind の方が「大きい」のでしたから True が得られます。

6.9.4 Builder のコンポーネントと PsychoPy のクラスの対応

図 6.11 に Builder の Grating コンポーネントと、それに対応する PsychoPy のクラスを示します。Grating コンポーネントに対応するクラスは psychopy.visual.GratingStim という名称 (以下 GratingStim) です。Grating コンポーネントの各プロパティは GratingStim クラスのデータ属性と対応しています。Builder において各プロパティに式や値を設定するという作業は、対応する GratingStim クラスのデータ属性にそれらを設定することと同義です。自分で実験用の Python コードを書く場合は、GratingStim クラスのデータ属性名やその設定方法を覚えなければいけません。Builder はプロパティ設定ダイアログという形でプロパティの一覧を見て設定ができるので、コードを書く方法を覚えるよりも手軽に自分の実験を作成できる段階まで学習できます。これが Builder を利用する最大の利点です。

ただし、の下の方に対応するプロパティが存在しないデータ属性があるように、Builder では GratingStim クラスの全てのデータ属性やメソッドを利用することができません。PsychoPy の機能を最大限に活かすためにはやはりコードを書く必要があります。

表 6.4 は、Polygon コンポーネントと PsychoPy のクラスの対応関係を示しています。Polygon コンポーネントの場合は、 **[形状]** に応じて最も効率がよいクラスを Builder が選択します。クラスによってデータ属性が異なりますので、Polygon コンポーネントのプロパティとデータ属性の対応も選択されたクラスに応じて変化します。

Builder は、背後にある PsychoPy のクラスやそのデータ属性についての知識がなくても使用できるように設

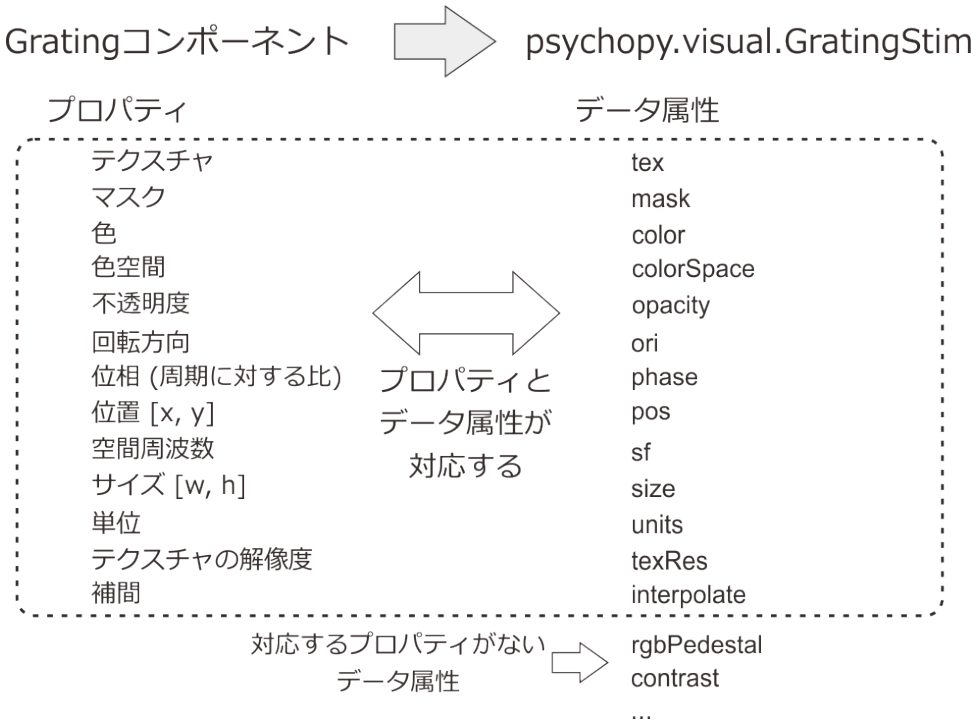


図 6.11　Builder の Grating コンポーネントのプロパティと、それに対応する PsychoPy のクラスのデータ属性。

計されています。実際、第 5 章 までは PsychoPy のクラスについて触れずに解説を進めてくることができました。しかし、この章の実験のように、参加者の選択に応じて刺激などが動的に変化する実験を作成するためには、残念ながら現状の Builder では PsychoPy のクラスについて言及せざるを得ません。

表 6.4　Builder の Polygon コンポーネントに対応する PsychoPy のクラス

形状	対応するクラス
直線	psychopy.visual.Line
三角形	psychopy.visual.ShapeStim
長方形	psychopy.visual.Rect
十字	psychopy.visual.ShapeStim
星	psychopy.visual.ShapeStim
正多角形...	psychopy.visual.Polygon

第 7 章

キーボードで刺激を調整しよう―ミューラー・リヤー錯視

7.1 この章の実験の概要

この章では有名な錯視のひとつであるミューラー・リヤー錯視を題材として、調整法の手続きを Builder で実現する方法を解説します。この章まで進んできた皆さんはそろそろ教示画面の作成は各自でできるでしょうから、重要な部分だけを取り上げましょう。 図 7.1 に実験に用いる刺激を示します。スクリーン上に左右に並んでテスト刺激とプローブが表示されます。テスト刺激はミューラー・リヤー錯視図形で、水平線 (以下主線と呼びます) の長さは 0.2、矢羽の長さは 0.05 です。主線と矢羽のなす角度 (以下夾角と呼びます) として 0 度から 30 度間隔で 150 度まで、6 種類の図形を用います。0 度の時は矢羽と主線がぴったり重なって長さ 0.2 の水平線だけに見える点に注意してください。プローブは水平な線分で、キーボードのカーソルキーの左右を使って長さを調節することができます。実験参加者は主線とプローブの長さが同じに見えるようにプローブの長さを調整して、スペースキーを押して報告します。この時のプローブの長さと主線の長さの差で錯視量を評価しようというのが本実験の狙いです。試行開始時のプローブの長さが毎回同じだと、参加者が何回キーを押せば主線とプローブが同じ長さになるかを学習してしまう恐れがありますので、試行開始時のプローブの長さは 170、190、210、230pix の中から無作為に選択します。テスト刺激、プローブともスクリーンの中央の高さで、水平方向の中心がスクリーン中央から 0.2 離れた位置に提示されるものとします。

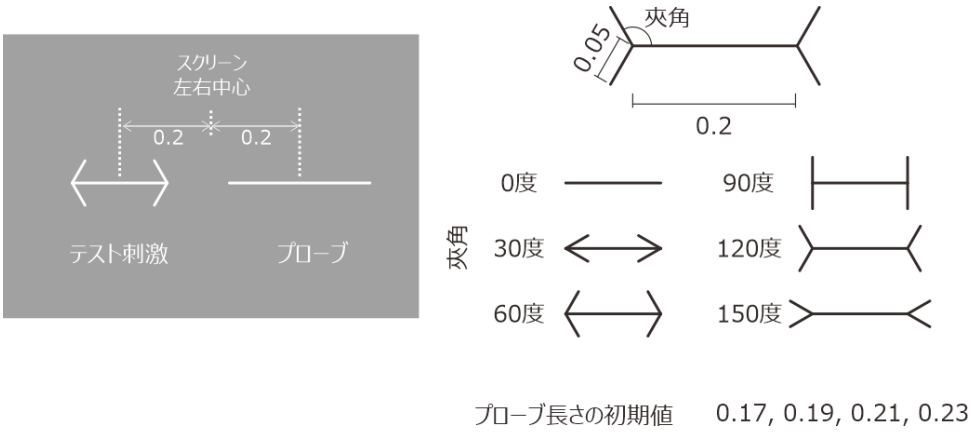


図 7.1　実験に用いる刺激。テスト刺激の水平線 (主線) の長さを 0.2、矢羽の長さを 0.05 で固定し、夾角を 0 度から 150 度まで変化させます。参加者は主線の長さとプローブの長さが等しく見えるようにプローブの長さを調整します。試行開始時のプローブの長さは 170、190、210、230pix のいずれかです。

図 7.2 に実験手続きを示します。教示画面は省略しましたので、いきなり最初の試行から始まります。各試行の最初に 0.5 秒間空白のスクリーンを提示した後、テスト刺激とプローブを提示します。テスト刺激は半数の試行で右側に、残り半数の試行で左側に提示され、順番は無作為に決定します。実験参加者がカーソルキーの右を押したら刺激の長さを 5pix 長く、左を押したら 5pix 短くします。参加者がスペース―を押したら試行は終了です。テスト刺激 (6 種類) ×テスト刺激の位置 (2 種類) ×プローブの初期長さ (4 種類)=48 通りの条件を 3 試行ずつ、合計 144 試行を行ったら実験は終了です。

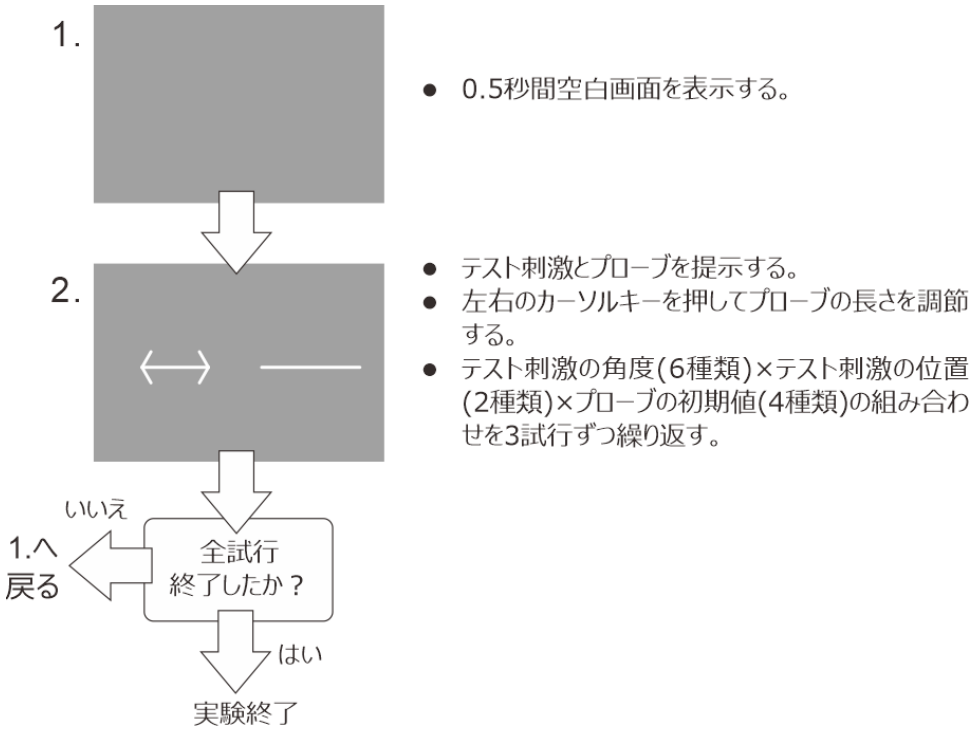


図 7.2　実験の流れ。

実験手続きは単純なので、前章までに解説したテクニックで十分実現できるはずです。問題は、「カーソルキーで刺激を調節してスペースキーで試行を終了する」という手続きをどうやって Builder で実現するかです。Keyboard コンポーネントでは、 **[Routine を終了]** プロパティを用いてキーが押されたときにルーチンを終了させるか否かを指定できます。しかし、特定のキーが押されたときだけ終了させるといった指定はできません。 第 6 章 で学んだ if 文を使うとキーに応じた処理を実行することが出来ます。

それでは実験を作成していきましょう。以下の解説では、Builder で実験を新規作成し、以下の作業を行って exp07a.psyexp の名前で保存したものとします。 **この章から既出のコンポーネントのプロパティについてはタブを省略しますのでご注意ください** 。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
- trial ルーチン
  - Polygon コンポーネントを 6 つ配置し、 **[名前]** を probe, testline, arrowTR, arrowBR, arrorTL, arrorBL とする (BR=Bottom Right、TL=Top Left といった命名規則)。すべて **[開始]** を「時刻 (秒)」の 0.5 とし、 **[終了]** を空白にして **[形状]** を直線にする。「外観」タブの **[枠線の色]** を white にする。どれかひとつを作成してこれらの設定を行い、残りをコンポーネントのコピーで作成すると楽

である。続いて以下の設定を行う。

* testline の **[位置 [x, y] $]** を (testPos, 0) にして、「繰り返し毎に更新」に設定する。
* testline の **[サイズ [w, h] $]** を (0.2,0) にする。
* probe の **[位置 [x, y] $]** を (-testPos, 0) にして、「繰り返し毎に更新」に設定する。
* probe の **[サイズ [w, h] $]** を (probeLen,0) にして、「フレーム毎に更新する」を設定する。
* arrowTR、arrowBR、arrorTL、arrorBL の **[サイズ [w, h] $]** を (0.05,1) にする。

– **Keyboard コンポーネントを 1 つ配置して、以下の設定を行う。**

* **[名前]** を key_response にする。
* **[開始]** を「時刻 (秒)」の 0.5 とし、 **[終了]** を空白にする。
* **[Routine を終了]** のチェックを外す。
* **[検出するキー $]** を'left','right','space' にする。
* **[記録]** を「なし」にする。

– **Code コンポーネントを 1 つ配置して、以下の設定を行う。**

* **[Routine 開始時]** に probeLen = initProbeLen と入力する。
* trial ルーチン内での順序を一番上にする。

- trials ループ (作成する)
  – **[Loop の種類]** を fullRandom にする。
  – **[繰り返し回数 $]** を 3 にする。
  – **[繰り返し条件]** に exp07cnd.xlsx と入力する。
- exp07cnd.xlsx(条件ファイル)
  – testPos、angle、initProbeLen の 3 パラメータを設定する。
  – 実験手続の内容を満たすように、2 種類の testPos(-0.2、0.2)、6 種類の angle(0、30、60、90、120、150)、4 種類の initProbeLen(0.17、0.19、0.21、0.23) の組み合わせを入力する。2 × 6 × 4=48 行の条件ファイルとなる (パラメータ名の列を除く)。testPos はテスト刺激の中心の X 座標を表しており、プローブ刺激の位置は testPos の符号を反転すれば得られるのでパラメータとして用意する必要はない。

教示などを省略したので以上で単純なフローの実験となりました。trial ルーチンに 6 個の Polygon ルーチンを配置しましたが、その内の 5 個 (testline、arrowTR、arrowBR、arrorTL、arrorBL) を使ってミューラー・リヤー図形を描きます。これらの位置を指定するのはちょっと複雑ですので次節でていねいに見ていきましょう。

7.2 Polygon コンポーネントで線分を描画しよう

Polygon コンポーネントで **[形状]** に直線を指定することによって、線分を描画することができます。多くのプログラミング言語でスクリーン上に線分を描くライブラリの多くは両端の座標を指定するのですが、Builder では **[位置 [x, y] $]** で図形の中央の座標を、 **[サイズ [w, h] $]** で大きさを指定することになっているため、他のプログラミング言語に慣れている人ほど戸惑うかもしれません。

Polygon コンポーネントで描く線分の長さは、 **[サイズ [w, h] $]** の幅の値 (1 番目の値) で指定します。2 番目の値は無視されますので、適当な値 (例えば 0) を入れておきましょう。線分の太さは **[枠線の幅 $]** で指定します。線分の色は **[枠線の色]** で指定します。 **[塗りつぶしの色]** は無視されます。

問題は **[位置 [x, y] $]** です。多角形を描画する時と同様に中心の座標を指定しないといけませんが、今回は「矢羽用の線分の端点を主線の端点と一致させる」必要がありますので、端点が意図した位置にくるように線分の中心位置を計算して **[位置 [x, y] $]** に指定しないといけません (図 7.3 上)。線分の中点の座標を (0, 0)、線分の長さを L、回転角度を q とすると、線分の端点の座標は ±(L/2 × cos(q), L/2 × sin(q)) です。± が付いているのは線分には端点が 2 つあるからです。線分の中点を (a, b) の位置へ移動させると、端点の座標は (a, b)±(L/2 × cos(q), L/2 × sin(q)) です。今、端点の一方である (a + L/2 × cos(q), b + L/2 × sin(q)) が (x, y) に一致するように線分の中点の座標を決めたいとしましょう。その時、(a, b) は (a + L/2 × cos(q), b + L/2 × sin(q)) = (x, y) を満たしますので、これを解いて a = x - L/2 × cos(q)、b = y - L/2 × sin(q) を得ます。本来ならばこれで線分の中点をどこへ置けばよいかの計算は終わりなのですが、Builder では正の回転方向が時計回りであり、通常の三角関数の計算の時と回転方向が逆であることを考慮しないといけません。対策は簡単で、回転時の Y 座標の符号を反転するだけです (図 7.3 下)。従って、a = x- L/2 × cos(q)、b = y + L/2 × sin(q) が求める答えです。

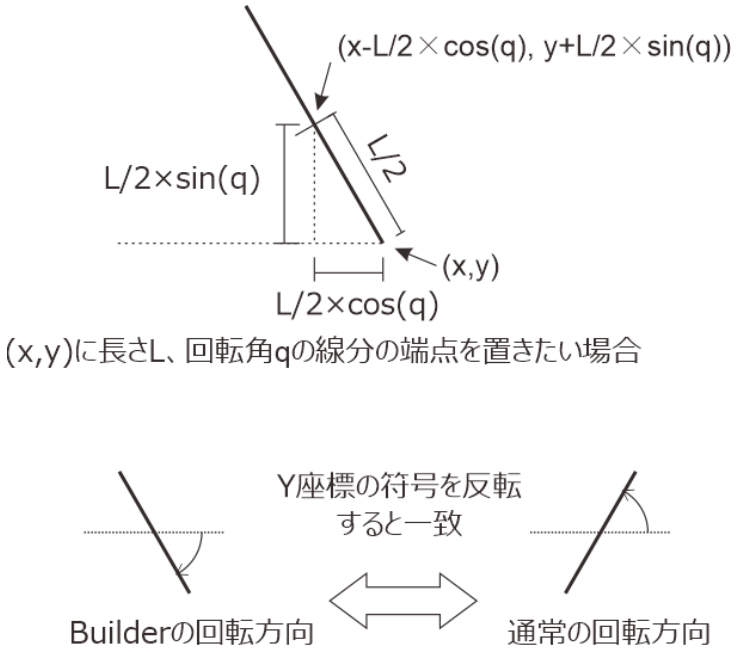


図 7.3　Polygon コンポーネントで斜めの線分を描画する時の位置の計算。Builder の回転方向と通常の三角関数における回転方向が逆なので、Y 座標の符号を反転させる必要があります。

同様の要領で、ミューラー・リヤー図形を描けるように 4 本の線分の中心位置を求めたのが 図 7.4 です。4 本すべての線分の回転角が q で統一できるように符号を決めていますので、確認してください。

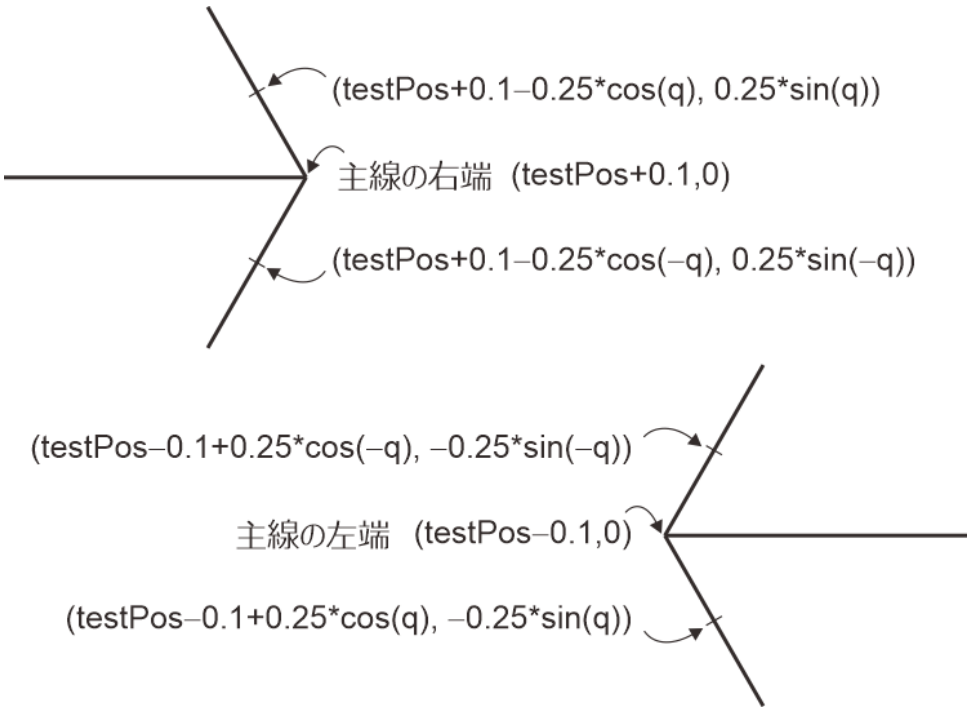


図 7.4　ミューラー・リヤー錯視の矢羽用の Polygon コンポーネントの座標。テスト刺激の位置 (X 座標) を testPos で、ミューラー・リヤー図形の夾角を q で表しています。

それでは exp07a.psyexp を開いてこれらの式を入力していきましょう。一点だけ注意しないといけないのは、図 7.4 の式の回転角 q の単位がラジアンである点です。条件ファイルに入力した夾角 angle は単位が度ですので、q の部分を deg2rad(angle) に書き換える必要があります。Polygon コンポーネントのプロパティに直接入力しても構わないのですが、式が複雑なので Code コンポーネントを使いましょう。矢羽の角度や位置は試行毎に変化しますので、 **[Rooutine 開始時]** に以下のように入力します (c と cc は時計回り (clockwise)、反時計回り (counterclockwise) を表しています)。

```
cx = 25 * cos( deg2rad(angle) )
cy = 25 * sin( deg2rad(angle) )
ccx = 25 * cos( deg2rad(-angle) )
ccy = 25 * sin( deg2rad(-angle) )
```

この cx, cy, ccx, ccy を使って以下の Polygon コンポーネントに式を入力していきます。

- arrowTR
  - **[回転角度 $]** に angle と入力し、「繰り返し毎に更新」を設定する。
  - **[位置 [x, y] $]** に [testPos+0.1-cx, cy] と入力し、「繰り返し毎に更新」を設定する。
- arrowBR
  - **[回転角度 $]** に-angle と入力し、「繰り返し毎に更新」を設定する。
  - **[位置 [x, y] $]** に [testPos+0.1-ccx, ccy] と入力し、「繰り返し毎に更新」を設定する。
- arrorTL

  - **[回転角度 $]** に-angle と入力し、「繰り返し毎に更新」を設定する。
  - **[位置 [x, y] $]** に [testPos-0.1+ccx, -ccy] と入力し、「繰り返し毎に更新」を設定する。
- arrorBL
  - **[回転角度 $]** に angle と入力し、「繰り返し毎に更新」を設定する。
  - **[位置 [x, y] $]** に [testPos-0.1+cx, -cy] と入力し、「繰り返し毎に更新」を設定する。

以上で条件ファイルのパラメータを読み込んでミューラー・リヤー図形を描く準備ができました。次はプローブの長さの変更に挑戦します。

チェックリスト

- Polygon コンポーネントを使って線分を描画することができる。
- Polygon コンポーネントを使った線分の長さ、角度が決まっている時に、その端点が指定された座標に一致するように **[位置 [x, y] $]** を指定することができる。

7.3 Code コンポーネントを使って刺激のパラメータとルーチンの終了を制御しよう

Code コンポーネントでは、押されたキーを判別してカーソルキーの右であればプローブの長さを 5pix 長く、左であれば 5pix 短くし、スペースキーであればルーチンを終了します。第 6 章 を読んだ皆さんであれば「if 文を使うとよい」ということはすぐにわかると思いますが、今回は処理が複雑です。第 6 章 では

1. 押されたキーの名前が変数 correctAns の値と一致している
2. 押されたキーの名前が変数 correctAns の値と一致していない

の 2 通りに分岐しましたが、今回は

1. 押されたキーの名前が'left' である
2. 押されたキーの名前が'right' である
3. 押されたキーの名前が'space' である
4. 押されたキーの名前がいずれにも一致しない

の 4 通りに分岐しなければいけません。このように 3 通り以上の分岐を処理するために、Python の if 文では elif という語を使うことができます。elif を使った if 文の書式は以下の通りです。n 個の式を連ねて書くことができます。最初は if、最後は else で、それ以外はすべて elif でなければいけません。

```
if 式 1:
    式 1 が真の時の処理
elif 式 2:
    式 1 が偽で式 2 が真の時の処理
（中略）
```

(次のページに続く)

(前のページからの続き)

```
elif 式n:
    式1から式n-1が偽で式nが真の時の処理
else:
    式1から式nがすべて偽であった時の処理
```

最初に真になった式に対応する処理だけが実行されますので、例えば式 1 が偽で式 2 が真であれば、「式 1 が偽で式 2 が真の時の処理」だけが実行されます。その後に続く式 3、式 4…が真であっても、対応する処理は一切実行されません。

押されたキーの名前が変数 key に入っているとすれば、今回の処理は以下のように書けます。上記の書式で n=3 の場合に該当します。

```
if key == 'left':
    プローブの長さから0.005を引く
elif key == 'right':
    プローブの長さに0.005を足す
elif key == 'space':
    ルーチンを終了する処理
```

else はどこへいった？と思われるかもしれませんが、else に対応する処理が何もない場合は else を省略することができます。今回はカーソルキーの左、右、スペースキーのいずれも押されていない場合は何もする必要がないので else を省略できます。

プローブの長さを変更する処理については、プローブ刺激に対応する Polygon コンポーネント probe の **[サイズ [w, h] $]** に [probeLen, 1] と書いているのですから、変数 probeLen の値を増減すればいいだけです。0.005 を加えるには probeLen += 0.005、0.005 を引くには probeLen -= 0.005 です。

ルーチンを終了させる処理については、まだ解説していない Builder の機能を使用する必要があります。Builder には、1 フレーム描画する毎に 1 回、continueRoutine という変数を確認し、値が False であれば直ちにルーチンを終了するという機能があります。if 文を用いて、ルーチンを終了させたい条件を満たした時に continueRoutine=False という文を実行すれば、ルーチンを終了させることができるわけです。

以上を踏まえて、if 文の処理内容を記述すると以下のようになります。

```
if key == 'left':
    probeLen -= 0.005
elif key == 'right':
    probeLen += 0.005
elif key == 'space':
    continueRoutine = False
```

残るは押されているキー名の取得です。第 6 章 の内容を踏まえると、Keyborad コンポーネントの **[名前]** が key_response ですから、key_response のデータ属性 keys を利用すればよいだけのような気がします。しかし、表 6.3 をよく読みなおしてほしいのですが、keys には押されたキー名が格納されているのではなく、そのルーチンで Keyboard コンポーネントが保存するキー名が格納されています。今回の実験では key_response の **[記**

録] を「なし」に設定しているのですから、データ属性 keys にはキー名が格納されません。

ではどうにすれば良いかといいますと、Builder が自動的に用意する変数 theseKeys を利用します。theseKeys は最後に実行した Keyboard コンポーネントの結果を格納している変数です。 **[検出するキー $]** に記述されたキーがいずれも押されていなければ空の (要素数ゼロの) リスト、押されていたキーがあれば、それらの名前をすべて列挙したリストが格納されています。「それらの名前」と複数形になっているのは、同時に複数のキーが押される場合があるからです。キーの同時押しについての詳細は「 7.7.1: 複数キーの同時押しの検出について 」を参考にしてください。

theseKeys を利用する場合、theseKeys の中身はリストですから theseKeys == 'left' という具合に直接文字列と等しいか比較しても意味がありません。theseKeys に格納されたリストの中に'left' という文字列があるかを検査しなければいけません。Python にはこの用途にうってつけの in という演算子があります。in 演算子は x in a という形で使用して、a の中に x があれば True、なければ False となります。theseKeys と in 演算子を用いて if 文を書くと以下の通りになります。

```
if 'left' in theseKeys:
    probeLen -= 0.005
elif 'right' in theseKeys:
    probeLen += 0.005
elif 'space' in theseKeys:
    continueRoutine = False
```

これで if 文は完成しましたが、問題はこの if 文を Code コンポーネントのどこへ入力すれば良いのかという点です。ここで覚えておいてほしいのが、「あるルーチンを実行している最中に何かをするのであれば、コードを入力する欄は **[フレーム毎]** でなければいけない」ということです。今回のコードはルーチンの実行中に押されたキーを判別して処理を振り分けるのですから、 **[フレーム毎]** に入力しなければいけません。タイプミスしないように気を付けて **[フレーム毎]** にこの if 文を入力してください。left や right、space の前後のシングルクォーテーションや、if、elif が出て来る最後のコロン、if、elif 以外の行は行頭にスペースが必要な点などが間違えやすいポイントです。

以上で Code コンポーネントによる刺激パラメータとルーチン終了の制御ができるようになりました。作業内容を exp07a.psyexp に保存して実行してみましょう。カーソルキーの左右を押すとプローブの長さが変化し、スペースキーを押すと次の試行へ進むはずです。残念ながらカーソルキーを押しっぱなしにしていても連続的に長さは変化しませんので、何度もカチカチとボタンを押して長さを調節する必要があります。

一見、これで 図 7.1 および 図 7.2 に示した実験が完成したように思えます。しかし、実験を最後まで実行するか Esc キーで中断して trial-by-trial 記録ファイルを確認してみるとわかりますが、probeLen の列に試行開始時の値が出力されていて、参加者が調整後の probeLen の値が記録ファイルのどこにも出力されていません。Builder は、Code コンポーネントで独自に使用した変数や、ルーチン開始後に変更されたパラメータの値を記録ファイルには出力しないのです。自動で出力してくれたらいいのにと思われるかもしれませんが、本当にそんなことをしたら記録ファイルが分析に不要な変数だらけで大変なことになりかねません。次節では、Coder コンポーネントを用いて実験記録ファイルに出力する変数を追加する方法を解説します。

チェックリスト

- 3 通り以上の分岐を処理させる if 文を書くことができる。

- リストの中にある要素が含まれているか否かで処理を分岐させることができる。
- Code コンポーネントからルーチンを終了させることができる。

7.4 Code コンポーネント使って独自の変数の値を記録ファイルに出力しよう

Builder が実験記録ファイルに出力する変数はどのように管理されているのでしょうか。ここで思い出していただきたいのは、条件ファイルで定義した変数はすべて実験記録ファイルに出力されるという点です。条件ファイルはループのプロパティ設定ウィンドウで指定するのですから、実験記録ファイルに出力される値を管理しているのはループであるはずです。

では、Builder におけるループの「実体」とは何でしょうか。その答えは **[Loop の種類]** によって異なるのですが、この本で使用している random、sequential、fullRandom の場合はいずれも psychopy.data.TrialHandler (以下 TrialHandler) というクラスのインスタンスです。TrialHandler はその名が示す通り、試行を制御するためのクラスです。繰り返しの度にパラメータの値を更新したり、パラメータや反応を実験記録ファイルに保存したりする Builder の機能はこのクラスによって実現されています。

表 7.1 に TrailHandler の主なデータ属性を示します。これらのデータ属性を利用すると、画面上に「現在第 n 試行」、「残り n 試行」といったメッセージを提示することができます。例えば trials という名前を付けたループに対応する TrailHandler のインスタンスは Coder 上では変数 trials に格納されているので、trials.thisN と書くと現在 trials ループで何回繰り返しを終えているかが得られます。 表 7.1 に書かれているように 1 回目の繰り返しでは thisN=0 なので、1 を加えて trials.thisN+1 とすれば「第 n 試行」の n に当てはめる数値が得られます。

表 7.1　TrialHandler の主なデータ属性

データ属性	概要
thisIndex	条件ファイルの何行目の条件がこのループで用いられているかを示します。ただしパラメータ名の行は行数に数えず、1 行目を 0 と数えます。Trial-by-trial 記録ファイルの thisIndex と同じです。
nTotal	このループで実行される繰り返しの総数。
nRemaining	このループで実行される残りの繰り返し回数。1 回目の繰り返しの実行中に nTotal-1 で、繰り返しの度に 1 ずつ減少します。
thisN	このループで実行済みの繰り返し回数。1 回目の繰り返しの時に 0 で、繰り返しの度に 1 ずつ増加します。Trial-by-trial 記録ファイルの thisN と同じです。
thisRepN	Trial-by-trial 記録ファイルの thisRepN と同じです。
thisTrialN	Trial-by-trial 記録ファイルの thisTrialN と同じです。

ちょっと脱線なのですが、第 5 章 の復習がてら実際に「第〇試行」と Text コンポーネントを使って提示する場合の注意点を述べておきましょう。trials.thisN+1 は数値ですので、そのまま文字列と結合することができません。ですから Text コンポーネントの **[文字列]** に$' 第' + (trials.thisN+1) + ' 試行' と書くと当然エラーになります。ここで 第 5 章 において実験情報ダイアログからターゲットの大きさや偏心度の値を得たときの処理を

思い出してください。あの時は実験情報ダイアログの値は文字列で、刺激のパラメータとして利用する時には数値に変換しないといけないのでした。今回はこの逆で、数値である trials.thisN+1 を文字列にしないといけません。第 5 章 で紹介したように、この変換には関数 str( ) を用います。$' 第' + str(trials.thisN+1) + ' 試行' とすれば、Text コンポーネントの Text の値として使用できます。

話を元に戻しましょう。TrialHandler クラスのインスタンスに実験記録ファイルへ出力する変数を追加するには、TrialHandler のメソッドを使用する必要があります。 表 7.2 に主な TrialHandler のメソッドを示します。メソッドの引数の書き方が PsychoPy のヘルプドキュメントと異なりますが、この点について解説するには Python の文法に関する詳しい解説が必要です。興味がある方は「 7.7.2: メソッドの第一引数について (上級) 」をご覧ください。 表 7.2 の最初に挙げられている addData( ) が今回の目的を達成するためのメソッドです。第 6 章 でも少しだけ例が出ていたのですが、メソッドはデータ属性と同様にインスタンスが格納されている変数とメソッド名をドット演算子で連結して呼び出します。trials ループに出力する変数を追加する場合は trials.addData( ) と書くわけです。 表 7.2 に書かれている通り、addData( ) には引数が必要です。第 1 引数には、trial-by-trial 記録ファイルや xlsx 記録ファイルに置いてその値が出力される列の名前 (記録ファイルの 1 行目の見出し) を文字列として渡します。第 2 引数には、出力したい変数や値を記述します。今回の例では、response という列名で実験参加者が調整した後の probleLen の値を出力してみることにしましょう。記入すべき文は以下の通りです。

```
trials.addData('response', probeLen)
```

この文を Code コンポーネントに追加すればいいのですが、どの欄に追加すればいいでしょうか。参加者がスペースキーを押して反応を確定した後に保存しないと意味がありませんから、 **[Routine 終了時]** に追加するのが正解です。追加して変更を保存してください。

表 7.2　TrialHandler の主なメソッド。

メソッド	概要
addData(thisType, value)	実験記録ファイルに出力する値を追加します。thisType に実験記録ファイルにおける列名、value に値を指定します。
getEarlierTrial(n)	n 回前に用いられたパラメータを得ます。1 回前であれば n=-1 という具合に負の整数で指定します。n が省略された場合は n=-1 と見なされます。n 回前が存在しない (2 回目で-5 を指定するなど) 場合は None、存在する場合は n 回前のパラメータが辞書オブジェクトとして得られます。
getFutuerTrial(n)	n 回後に用いられるパラメータを得ます。1 回後であれば n=1 という具合に正の整数で指定します。n が省略された場合は n=1 と見なされます。n 回後が存在しない場合は None、存在する場合は n 回後のパラメータが辞書オブジェクトとして得られます。

保存したら実験を実行してみましょう。終了後に trial-by-trial 記録ファイルと xlsx 記録ファイルを開くと、図 7.5 のように response という名前の列が存在していて、そこに調整後の probeLen の値が出力されているのがわかります。xlsx 記録ファイルには Keyboard コンポーネントの出力と同様に平均値や標準偏差も出力されています。

これでこの章の実験は完成です。ですが、せっかく 表 7.2 に addData( ) 以外の TrialHandler のメソッドを紹

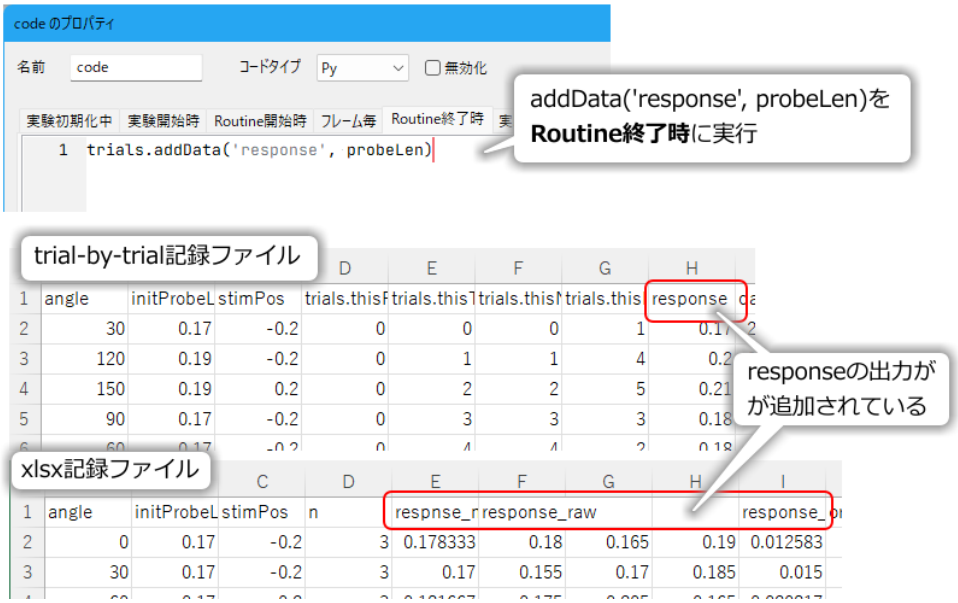


図 7.5　addData( ) メソッドによる変数の出力。

介しましたので、少し触れておきましょう。 表 7.2 に書かれている通り、TrialHandler には getEarlierTrial( ) と getFutuerTrial( ) というメソッドがあります。これらのメソッドを使うと、それぞれ現在のループの n 回前、および n 回後の繰り返しで条件ファイルから読み込まれたパラメータのどの値が用いられた（用いられる）かを知る事ができます。例えば trials ループの内部のルーチンに Code コンポーネントを配置して、 **[Routine 開始時]** 、 **[フレーム毎]** 、 **[Routine 終了時]** のいずれかで以下の文を実行すると、2 回前の繰り返しで用いられたパラメータが変数 prevParam に格納されます。

```
prevParam = trials.getEarlierTrial(-2)
```

prevParam に格納されているのは辞書オブジェクトです。 第 4 章 、第 5 章 でも触れたように、辞書オブジェクトとは実験情報ダイアログの値を保持するのにつかわれているデータ形式です。ですから、実験情報ダイアログと同様に、以下のように書くと angle というパラメータの値を取り出すことができます。

```
prevParam['angle']
```

記憶課題の一種に「左右の選択肢のうち、n 試行前に提示されていた刺激と一致する選択肢を選べば正解」という課題 (n-back 課題) がありますが、 **[Loop の種類]** に random や fullRandom を選んでいる場合、n 試行前に提示した刺激は無作為に決定されているので条件ファイルで正答を定義することができません。そのような場合に getEarlierTrial( ) メソッドは非常に有効です。

チェックリスト

- TrialHandler のインスタンスから現在ループの何回目の繰り返しを実行中かを取得できる。
- TrialHandler のインスタンスから現在ループの繰り返し回数が残り何回かを取得できる。
- TrialHandler のインスタンスから現在ループの総繰り返し回数を取得できる。
- 上記 3 項目の値を使って「現在第 n 試行」、「残り n 試行」、「全 n 試行」といったメッセージをスクリーン上に提示できる。

- Code コンポーネントを用いて、実験記録ファイルに出力するデータを追加することができる。
- 現在実行中の繰り返しの n 回前、n 回後に使われるパラメータを取得するコードを記述することができる。

7.5 プローブの長さが一定範囲に収まるようにしよう

すでにこの章で目的とする実験は完成しているのですが、if 文の練習を兼ねて少し改造してみましょう。exp07a.psyexp では、実際にする人がいるかどうかは別として、テスト刺激に重なったりスクリーンからはみ出してしまったりするくらいプローブを大きくすることができてしまいます。また、プローブをどんどん小さくしていけばいずれ長さは 0 になり、負の値になってしまいます。長さは負の値をとることができませんので、そこで実験はエラーとなり停止してしまいます。このような事態を避けるために、if 文を使ってプローブの長さが 0.05 から 0.35 の範囲を超えて短くしたり長くしたりできないようにしてみましょう。

作成済みの exp07a.psyexp を exp07b.psyexp という別名で保存してください。そして trial ルーチンの Code コンポーネントを開いてください。 **[フレーム毎]** に入力してあるコードによってプローブの長さが変わるのですから、ここのコードを書きかえると長さを一定の範囲に制限することができるはずです。ひとつの問題を解決するための方法を一度に何通りも紹介するのはよくないかも知れませんが、ここでは if 文の練習なので 2 通りの方法を考えます。

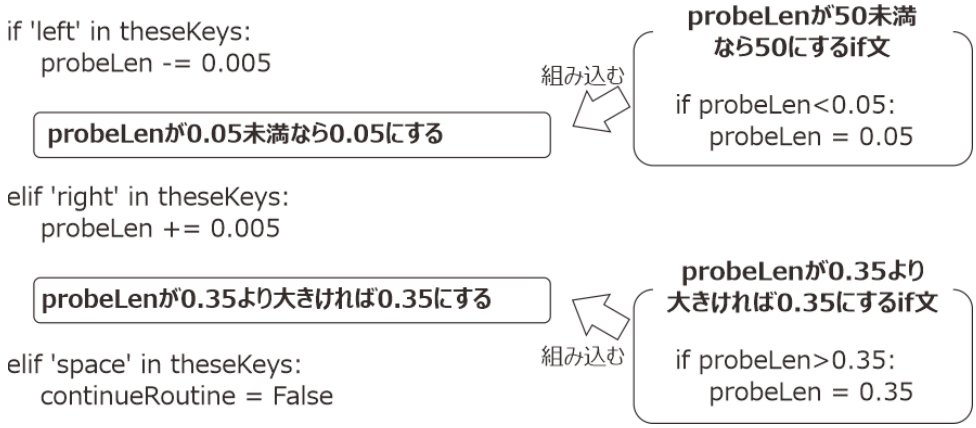


図 7.6　プローブの長さを制限する方法その 1。exp07a.psyexp の Code コンポーネントに書いたコードに組み込む処理を日本語で記入したものを左側に、組み込む処理に対応するコードを右側に示しています。

第一の方法は、長さを 0.005 増加させた時に 0.35 より大きくなっていないか確認して、なっていれば 0.35 に修正し、長さを 0.005 減少させた時に 0.05 未満になっていないか確認して、なっていれば 0.05 に修正するというものです。入力済みのコードに日本語で処理を書きこむと 図 7.6 の左のようになります。ご覧のとおり、if 文で分岐した後にまた「もし〜なら…」という分岐処理が含まれる形になっています。if 文では、このような「入れ子」になった条件分岐も書くことができます。「probeLen が 0.05 未満なら 0.05 にする」という部分だけを考えると、これは 0.05 以上なら何もしないということですから、else は省略できて 図 7.6 右上のように書けます。同様に「probeLen が 0.35 より大きければ 0.35 にする」という処理も 図 7.6 右下のように書けます。 図 7.6 左側のコードの日本語で記入した部分に、 図 7.6 右側の対応するコードを埋め込むと、 図 7.7 左に示すコードが得られます。これだけで完成です。

図 7.7 左のコードをもう少ししっかり見ておきましょう。 図 7.7 左のコードの冒頭部分を拡大したのが 図 7.7 右です。if が及ぶ範囲は字下げの量で決まります。Python は if を発見すると、コードを下へ読み進めていっ

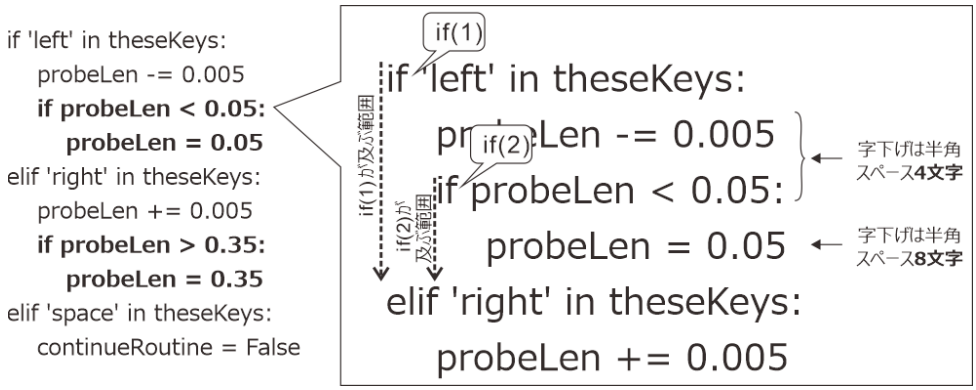


図 7.7　if 文を組み込んで得られたコード。if、elif、else の及ぶ範囲は、下方向に向かってこれらの語が出現した行と字下げ幅が狭いか同じ行に出会うまでです。if (2) が半角スペース 4 文字字下げされているので、if (2) の次の行は if (2) よりさらに 4 文字字下げして 8 文字字下げとなります。

て、これらの語が出現した行と字下げ量が同じか少ない行の直前の行までを if の条件式が及ぶ範囲と見なします。このことを念頭に置いて 図 7.7 右のコードを見ると、1 行目の if 文 (if (1) とします) の及ぶ範囲は 5 行目の elif 文の手前まで、すなわち 4 行目までであることがおわかりいただけると思います。if (1) の条件式が真であれば、4 行目までのコードが実行されます。一方、3 行目の if 文 (if (2) とします) の範囲はどこまでかと言いますと、これも 5 行目の elif 文の手前まで、すなわち 4 行目までです。ですから if (2) の条件式が真であれば、4 行目だけ実行されます。もちろん if (2) は if (1) の範囲に入っているので、if (2) が実行されるためには if(1) が真でなければいけません。if (1) が真で if (2) が偽であれば、2 行目だけが実行されます。elif、else が及ぶ範囲も if と同様に決まります。なお、本書では字下げをすべて 4 文字としているので 図 7.7 の説明で問題ないのですが、web 上で誰かが書いた Python のコードを流用する時にはそのコードが異なる字下げルールを使っているかもしれません。そのようなコードをコピーするときの注意点を「7.7.3:Python コードの字下げについて」に記しておきますので参考にしてください。

では、exp07b.psyexp を開いて、 図 7.7 左のコードを trial ルーチンの Code コンポーネントの **[フレーム毎]** に入力してください。すでに細字の部分は入力済みのはずなので太字部分を入力するだけでいいはずです。入力したら実験を保存して実行し、プローブの長さが一定以上伸びたり縮んだりしないことを確認してください。プローブ長が 0.05 や 0.35 に達するまでカーソルキーを連打するのが面倒くさい！という方は実験をいったん終了して、Code コンポーネントを編集してカーソルキーの左右を押したときに probeLen が増減する量を ±0.005 から ±0.05 などに変更して実行してみましょう。

続いて第二の方法の解説です。exp07b.psyexp を閉じて exp07a.psyexp を開きなおして、exp07c.psyexp という名前で保存して作業しましょう。第一の方法では、probeLen の長さを増減した直後に範囲外に出てしまっていないかを確認しましたが、第二の方法では一連の if-elif-else が終わってから probeLen を確認します。 図 7.8 にこの方法を用いたコードを示します。ここでのポイントは、if に対応する elif、else が置ける範囲です。図 7.8 に記した通り、1 行目の if 文 (if (1) とします) に対応する elif、else が置けるのは、if (1) と字下げが同じで elif、else 以外から始まる行が出てくるか、if (1) より字下げが少ない行が出てくる直前の行までです。 図 7.8 のコードでは、2 つ目の if (if (2) とします) が出現した時点で if (1) が終了していますので、if (1) から続く一連の if、elif の結果がどうであろうと必ず if (2) の条件式は評価されます。elif で条件式を列挙した場合は手前の if や elif で真になった時に全く評価されなかったのと対照的です。

exp07c.psyexp は exp07a.psyexp を別名で保存して作成したので、trial ルーチンの Code コンポーネントの **[フレーム毎]** に 図 7.8 のコードの if (2) の手前まですでに入力済みのはずです。そこへ、 図 7.8 のコードの最後の 4 行 (if (2) に対応する部分) を追加入力してください。continueRoutine = False という行と if (2) の最初の

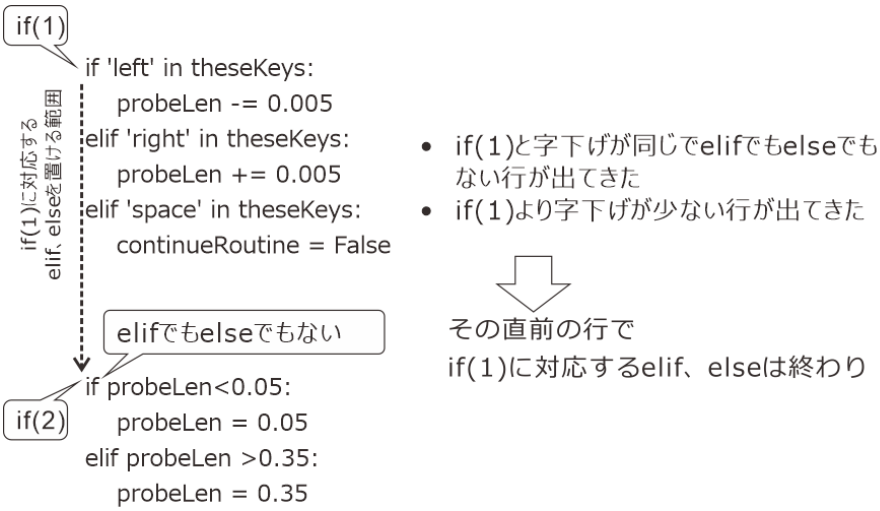


図 7.8　プローブの長さを制限する方法その 2。if に対応する elif、else を置ける範囲に注意。

行の間は空白行を入れても入れなくても動作しますが、1 行空白を入れておいた方が後から見直した時にここで新たな if 文が始まることがわかりやすくてよいでしょう。入力を終えたら、exp07c.psyexp を保存して実験を実行してみてください。exp07b.psyexp の時と同様に、プローブの長さが一定以上伸びたり縮んだりしないはずです。キーを連打するのが面倒な方はやはり exp07b.psyexp の時と同様に、一回のキー押しで probeLen を増減する量を大きくして試してみましょう。

以上で if 文の練習は終わりですが、最後にひとつ補足しておきます。第一の方法は入れ子になった if 文の練習のためにまず「カーソルキーの左が押されたか」を判定して probeLen の長さを変更してから「probeLen が 0.05 未満か」を判定するという二段階の判定を行いましたが、第 6 章 で学んだ論理演算子を使えば一度に判定することができます。probeLen の初期値は 170、190、210、230 の 4 通りしかなくて、±0.005 ずつしか増減しないのですから、0.005 を引いて 0.05 未満になるのは probeLen が 0.05 の時のみです。ということは、「カーソルキーの左が押されていて、なおかつ probeLen が 0.05 より大きい」時には probeLen から 0.005 を引いても 0.05 を下回ることはありません。したがって、論理演算子 and を使って一つの条件式として記述できます。

```
if 'left' in theseKeys and probeLen > 0.05:
    probeLen -= 0.005
```

同様に、probeLen に 0.005 を加えて 0.35 より大きくなるのは probeLen が 0.35 に達しているときだけですから、「カーソルキーの右が押されていて、なおかつ probeLen が 0.35 未満」の時には probeLen に 0.005 を足しても 0.35 を超えません。したがって、この条件は and を使って一つの式として記述できます。

```
elif 'right' in theseKeys and probeLen < 0.35:
    probeLen -= 0.005
```

この方法の弱点は、probeLen の増減量が可変である時にはかえって複雑になってしまうことです。その場合は 図 7.7 や 図 7.8 に示した方法を用いた方がすっきりとしたコードが書けます。この「増減量が可変」な実験を作成することを練習問題として、この章を終えることにしましょう。

チェックリスト

- if 文の中に入れ子上に if 文を組み込んだコードを記述することができる。
- if や elif の条件式が真であった時に実行されるコードがどこまで続いているかを判断することができる。if、elif の条件式が全て偽で else まで進んだときに実行されるコードがどこまで続いているかを判断することができる。
- 一連の if-elif-else の組み合わせがどこまで続いているかを判断することができる。
- 論理演算子を用いて複数の条件式をひとつの式にまとめることができる。

7.6 練習問題：プローブ刺激の伸縮量を切り替えられるようにしよう

exp07a.psyexp をベースにして、プローブの伸縮量 (probeLen の増減量) を切り替えられるようにしてみましょう。二通りの実現方法を挙げますので、ぜひ両方の方法の実現に挑戦してください。

- 実現方法その 1
  - Shift キーを押すと、伸縮量が ±2 と ±10 で切り替わる。余力がある人は Text コンポーネントを使って現在の伸縮量を画面上に提示すること。
    - ヒント 1：伸縮量の絶対値を保持する変数を一つ用意して、Shift キーが押されたら値を切り替える。
    - ヒント 2：伸縮量の絶対値を保持する変数には、ルーチン開始時に初期値を与える必要がある。
- 実現方法その 2
  - X キーを押すと-10、C キーを押すと-2、ピリオド (.) キーを押すと +2、スラッシュ (/) キーを押すと +10 伸縮する。
    - この方法についてはヒントなし。

「どちらもあっさりできてしまった」という方は、以下の問題にも取り組んでみてください。

- 実現方法その 1、その 2 共通
  - 図 7.8 の例のように、キー入力に関する処理を終えた後に probeLen が範囲を超えていないかを確認して必要があれば値を修正すること。ただし、その際 if 文を使わずに、第 5 章 に出てきた関数を使って「1 行で」処理を記述すること。
    - ヒント：二種類の関数を使う必要がある。

7.7 この章のトピックス

7.7.1 複数キーの同時押しの検出について

キーボードは製品によってテンキーと呼ばれる独立した数字や四則演算のキーがあったり、音量を調節するためのキーがあったり、いろいろなものがありますが、特殊なものを除けば 80 個以上のキーがあります。これらのキーは物理的には複数個同時に押すことはできますが、文書を書くなどの一般的な用途では「P と S と Y のキーを同時押しする」といった具合に複数の文字キーを押すことはありません。ですから、市販されているキーボードの中には、Shift や Ctrl といった特殊なキーを除いて、複数キーの同時押しを検出することを前提に設計されていないものがあります。例えば筆者が使用しているキーボードの中には、F、G、H、J のキーを同時に押すと F と J のみ、G と H のみといった具合にいずれか 2 つのみしか同時に認識しないものがあります。一方、これらの 4 個のキーの同時押しを認識させることができるキーボードもあります。

通常の文書入力では 4 つのキーの同時押しを検出できなくても困ることはまず無いのですが、キーボードを使用して操作するアクションゲームの場合は大きな問題になり得ます。ですから、ゲーム用と銘打って販売されているキーボードは多くのキーを同時押しできるように設計されています。中にはすべてのキーの同時押しを検出できる製品もあります。

Builder は、同時押しに対応したキーボード、対応していないキーボードのどちらが接続されていて同じコードで処理できるように、変数 theseKeys に押されたキー名を必ずリストとして保存するように作られています。複数キーを同時押ししているにも関わらず theseKeys に押したキー名が含まれていない場合は、そのキーの組み合わせが使用中のキーボードで検出できない組み合わせである可能性があります。

7.7.2 メソッドの第一引数について (上級)

TrialHandler の主要クラスメソッドの表 (表 7.2) において、getEarlierTrial( ) の引数は n の 1 個だけしか示されていません。しかし、Python インタプリタ上で psychopy.data を import して help(psychopy.data.TrialHandler) を実行して getEarlierTrial( ) のヘルプを見ると、self と n という 2 つの引数が記載されています。 表 7.2 で第一引数 self を省略している理由について簡単に解説します。

公式ヘルプに記載されている第一引数 self ですが、これは TrialHandler に限らずすべてのクラスのメソッドに必ず存在します。これは「インスタンス自身」を指し示す引数です。C 言語や C++ 言語を御存知の方には、「インスタンスへのポインタが渡される」と言えばわかりやすいかも知れません。クラスの定義に関する Python の文法を解説していないのでどうしても不正確な説明にしかならないのですが、この self はインスタンスが自分自身に格納されたデータ属性の値を知るために必要なもの、と思っておいてください。

Python の文法では、メソッドを呼び出す時に self は省略して記述すると定められています。ですから、引数が self のみしかない foo というメソッドを呼び出す場合は foo( ) という具合に括弧の中は空白にします。TrialHandler の getEarlierTrial( ) を呼び出す場合には、このメソッドには self と n という 2 つの引数があるので、self を省略して getEarlierTrial(n) と書きます。 表 7.2 では、実際にコードを書くときの表記と一致させることを重視して getEarlierTrial( ) の引数として n のみを記載しています。

なお、同じくヘルプの addData( ) メソッドの引数を見ると、self、thisType、value に加えてさらに position=None と書かれています。これはデフォルト値付き引数と呼ばれるもので、「引数 position が渡されなかった場合は None が渡されたと解釈する」ということを意味します。ですから、本文中で trials.addData('response',

probeLen) としたように position に相当する引数を渡さなくてもエラーにならなかったのです。getEarlierTrial( ) の引数 n も n=-1 という具合にデフォルト値付き引数として定義されているので、本文中や 表 7.2 で述べたように n を省略することができるのです。

7.7.3 Python コードの字下げについて

第 6 章 において、Python Enhancement Proposals (PEP) という公式文書で半角スペース 4 文字の字下げが推奨されていることを紹介しました。PEP は Python の言語仕様や Python プログラマのコミュニティ向けの情報などを記述した文書の集合で、その中の PEP-8 という Python のコードの書き方を定めた文書に「半角スペース 4 文字を使いなさい」と記されています。

しかし、Python 以外のプログラミング言語では字下げに Tab 文字や 8 文字の半角スペースなど、さまざまな字下げが使用できるためか、Python でも半角スペース 4 文字以外の字下げを使用できるようになっています。図 7.9 は、2 文字や 6 文字の字下げが混在しているコードの例を示しています。 図 7.9 左のように字下げが混在していてもそれぞれのブロック内で字下げが統一されていれば動作します。例えば 図 7.9 の (3) は直前の if に対する字下げが 6 文字、(4) は直前の else に対する字下げが 4 文字であり、一連の if-else 文にも関わらず字下げが一致していません。しかし、(3)、(4) のブロック内でそれぞれ字下げが一貫しているので Python は適切にこのコードを解釈して実行することができます。それに対して 図 7.9 の (5) では、最初の 2 行の字下げが 4 文字であるにもかかわらず最後の i+=1 の字下げは 3 文字であり、(5) のブロック内で一貫していません。従って、 図 7.9 右のコードはエラーとなり実行できません。

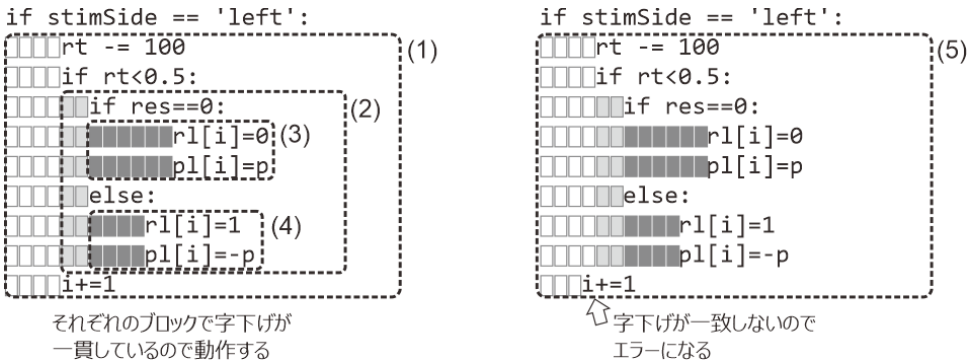


図 7.9　4 文字以外の半角スペースによる字下げ。それぞれのブロック内で字下げが一貫していればエラーにはなりません。

スペースと Tab 文字が混在しているとさらに事態は複雑になります。基本的には、Tab 文字は半角スペース 8 文字に置き換えられます。しかし、半角スペース 8 文字未満に Tab 文字が続く場合は、半角スペースと Tab 文字を合わせて半角スペース 8 文字と解釈されます。 図 7.10 の例をご覧ください。 図 7.10 左のコードの最終行は、4 文字の半角スペースより前に Tab 文字がありますから、Tab 文字が半角スペース 8 文字分に解釈されて合計半角スペース 12 文字と解釈されます。従って、 図 7.10 左のコードはエラーとならず実行できます。一方、 図 7.10 右の最終行は、左と同じ Tab 文字と半角スペース 4 文字の組み合わせなのですが、半角スペースが Tab 文字より前にあります。この場合、半角スペースと Tab 文字を合わせて半角スペース 8 文字と解釈されますので、直前の if 文と字下げ量が同じとなってしまいエラーになります。

以上のように、Python のスクリプトでは字下げは半角スペース 4 文字でなくても動作します。しかし、混乱を避けるためにはやはり PEP-8 に従って半角スペース 4 文字で統一するべきだと思われます。

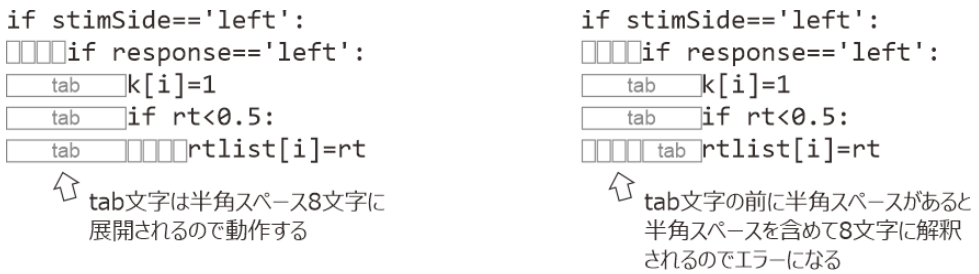


図 7.10　Tab 文字による字下げ。基本的には Tab 文字は半角スペース 8 文字に置換されると考えておけばよいですが、左の例のように半角スペースの後ろに Tab がある場合は半角スペースと Tab をまとめて 8 の倍数個のスペースとして解釈されてしまいます。

第 8 章

マウスで刺激を動かそう—鏡映描写課題

8.1 この章の実験の概要

これまでの章では実験参加者の反応計測のためにキーボード用いてきましたが、この章ではキーボードと並んで普及している PC の入力機器であるマウスを反応に用いる方法を学びます。ただ今までカーソルキーの左右を押していたのをマウスの左クリック、右クリックに置き換えただけでは面白くないので、マウスならではの課題である鏡映描写課題を Builder で作成したいと思います。

鏡映描写課題とは、鏡に映った自分の手の像を見ながら図形をペンでなぞる課題です (図 8.1 左)。ペンでなぞる図形の上には遮蔽版が置かれていて図形を直接見ることができないようになっていて、奥に立てられた鏡に映った像を見ながら手を動かさなければいけません。鏡を見ながら描画するために前後方向（鏡に近づく、離れる）に手を動かしたときの視覚像の動きが逆転してしまうので、うまく図形をなぞるのはかなり困難です。しかし、何度も練習を繰り返していると、次第に手とその鏡像の動きの関係が学習され、素早く間違わずになぞることができるようになってきます。状況にふさわしい知覚と運動の関係を学習することを知覚運動学習といいます。鏡映描写課題は、知覚運動学習の課題として用いられます。

鏡映描写課題と類似の課題を PC で実現するために、この章では 図 8.1 右のような課題を考えます。スクリーン上に何カ所か折れ曲がった白い太線 (パス) と、小さな円 (プローブ) が提示されています。実験参加者がマウスを動かすとプローブが動きますが、スクリーンの上下方向のみ通常のマウスカーソルとは逆向きに動きます。つまり、マウスを奥に向かって動かすとプローブはスクリーンの下方向へ移動し、マウスを手前へ動かすとプローブはスクリーンの上方向へ移動します。マウスを動かすテーブルに箱か何かを置いて手元が見えないようにすれば、鏡映描写に類似した状況が得られるはずです。

もう少し実験の詳細を検討していきましょう。まず、パスをどのような形にするかを決めなければいけません。 図 8.1 は実験のアイディアを記しただけですので適当なパスを描いてありますが、実際に実験を作成するとなるときちんとした形状を決定する必要があります。特に Builder で作成するにはその大きさや中心位置を座標として計算できないといけません。また、パス間で移動すべき距離が異なるのは避けたいところです。そこで、この章では 図 8.2 のように星形の図形の辺を時計回りまたは反時計回りに進んで半周するパスを採用します。星形の中心から鋭角の頂点までの距離は 0.3 で、星形の中心がスクリーンの中心に一致するように配置します。スタート地点は鈍角の頂点から選び、星形の中心をはさんでスタート地点と向かい合う鋭角の頂点がゴール地点となります。スクリーン上でゴール地点をすぐに判別できるように、直径 0.03 のディスク (円) をゴール地点に提示します。パスは幅 0.02 の長方形を用いて描画します。スタート・ゴールの組み合わせが 5 種類、進行方向が反時計回りか時計回りかの 2 種類ありますので、合計 10 種類のパスが得られます。

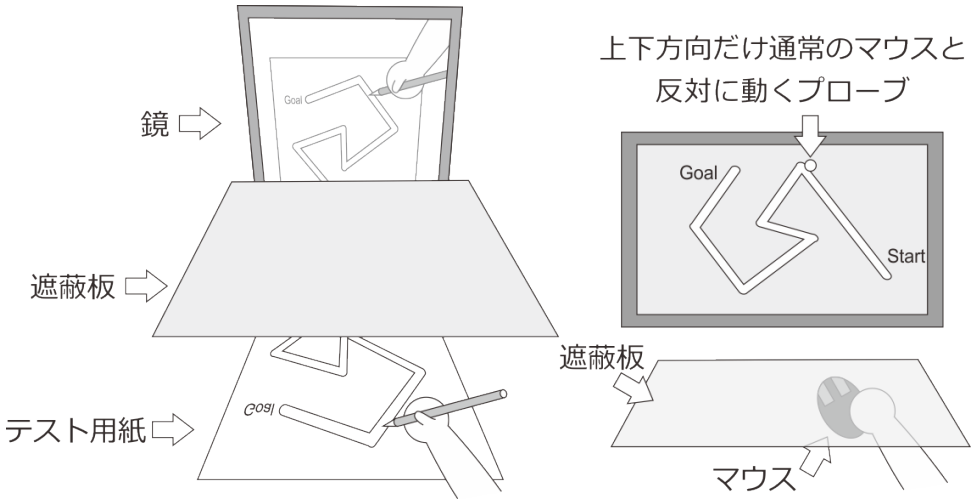


図 8.1　鏡映描写課題。遮蔽版の下に置かれたテスト用紙に描かれた図形を、鏡に映った像を見ながらペンでなぞります。スクリーンとマウスを用いてこの実験装置をシミュレートします。

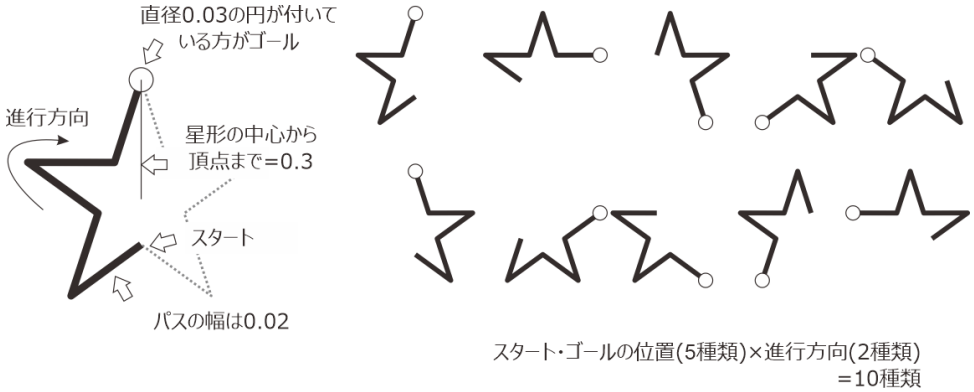


図 8.2　実験で使用するパス。星形の辺を時計回りまたは反時計回りに進んで半周します。5 種類のスタート・ゴールの組み合わせと 2 種類の進行方向で合計 10 種類のパスが得られます。

パスを描画するための座標値を計算してみましょう (図 8.3)。座標値の計算は、Builder でマウスを操作する方法を学ぶという本章の目的とは直接関係ありませんので、よくわからなければ次の段落まで読み流していただいて構いません。まず、長方形を用いてパスを描画するのですから、Builder では Polygon コンポーネントを使うのが適切です。Polygon コンポーネントで星形の辺を描画するには、その辺の長さと中点の座標が必要です。 図 8.3 では、辺 AE の長さと、辺 AE の中点である点 F の座標が必要な値です。点 A と B の座標がすぐに求まること、B と E の Y 座標が等しいこと、辺 AC が y=-tan(72°) x+300 上にあることを利用すれば、F の座標は計算できます。点 F の座標が得られれば Y 軸を挟んで向かい合った点 G の座標が直ちに得られます。点 F と G の座標がわかれば、原点を中心としてこれらの座標を 72 度ずつ回転させればすべての辺の中点が得られます。点 F の座標を求める途中で点 A と E の座標が得られますので、辺の長さも計算可能です。

座標値を計算して、図 8.2 の 10 通りのパスを描けるように Builder の条件ファイルの形で並べたものが 図 8.4 です。pathXpos(X=1 から 5) が各辺の中心の座標で、 **[位置 [x, y] $]** の値として使用します。pathXori は **[回転角度 $]** に使用する回転角です。startPos と goalPos はそれぞれスタート地点とゴール地点の座標です。このファイルを exp08cnd.xlsx という名前で保存して使用します。

パスにおける各辺の長さと幅はすべて等しいので、条件ファイルを利用せずに Builder 上で直接 **[サイズ [w, h] $]** に値を入力することにします。 図 8.3 から計算した辺の長さは 0.218 なのですが、ぴったり 0.218 にする

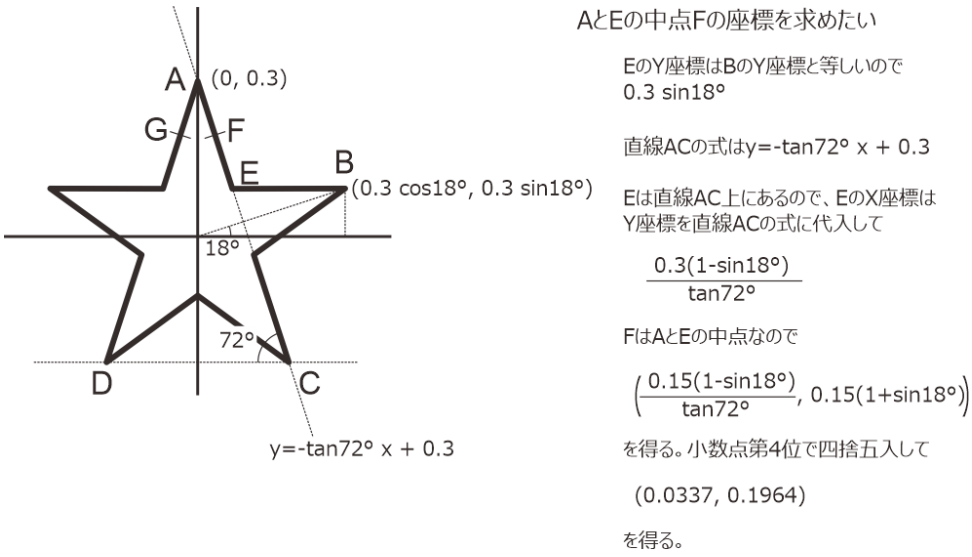


図 8.3　星形を描画するための座標値の計算。F の座標が得られれば G の座標も直ちに得られます。F と G の座標を 72 度ずつ回転させればすべての辺の中点が得られます。

	A	B	C	D	E	F	G	H	I	J	K	L
1	path1pos	path1ori	path2pos	path2ori	path3pos	path3ori	path4pos	path4ori	path5pos	path5ori	startPos	goalPos
2	(0.1763,0.0927)	0	(0.1971,0.0286)	-36	(0.1427,-0.1391)	-108	(0.0882,-0.1786)	-144	(-0.0882,-0.1786)	-36	(0.0674,0.0927)	(-0.1763,-0.2427)
3	(0.1427,-0.1391)	-108	(0.0882,-0.1786)	-144	(-0.0882,-0.1786)	-36	(-0.1427,-0.1391)	-72	(-0.1971,0.0286)	-144	(0.1090,-0.0354)	(-0.2853,0.0927)
4	(-0.0882,-0.1786)	-36	(-0.1427,-0.1391)	-72	(-0.1971,0.0286)	-144	(-0.1763,0.0927)	0	(-0.0337,0.1964)	-72	(0.0000,-0.1146)	(0.0000,0.3000)
5	(-0.1971,0.0286)	-144	(-0.1763,0.0927)	0	(-0.0337,0.1964)	-72	(0.0337,0.1964)	-108	(0.1763,0.0927)	0	(-0.1090,-0.0354)	(0.2853,0.0927)
6	(-0.0337,0.1964)	-72	(0.0337,0.1964)	-108	(0.1763,0.0927)	0	(0.1971,0.0286)	-36	(0.1427,-0.1391)	-108	(-0.0674,0.0927)	(0.1763,-0.2427)
7	(0.0337,0.1964)	-108	(-0.0337,0.1964)	-72	(-0.1763,0.0927)	0	(-0.1971,0.0286)	-144	(-0.1427,-0.1391)	-72	(0.0674,0.0927)	(-0.1763,-0.2427)
8	(-0.1763,0.0927)	0	(-0.1971,0.0286)	-144	(-0.1427,-0.1391)	-72	(-0.0882,-0.1786)	-36	(0.0882,-0.1786)	-144	(-0.0674,0.0927)	(0.1763,-0.2427)
9	(-0.1427,-0.1391)	-72	(-0.0882,-0.1786)	-36	(0.0882,-0.1786)	-144	(0.1427,-0.1391)	-108	(0.1971,0.0286)	-36	(-0.1090,-0.0354)	(0.2853,0.0927)
10	(0.0882,-0.1786)	-144	(0.1427,-0.1391)	-108	(0.1971,0.0286)	-36	(0.1763,0.0927)	0	(0.0337,0.1964)	-108	(0.0000,-0.1146)	(0.0000,0.3000)
11	(0.1971,0.0286)	-36	(0.1763,0.0927)	0	(0.0337,0.1964)	-108	(-0.0337,0.1964)	-72	(-0.1763,0.0927)	0	(0.1090,-0.0354)	(-0.2853,0.0927)

図 8.4　本章で用いる条件ファイル (http://s12600.net/psy/python/ppb/ からダウンロードできます)

と頂点のところでパスが狭くなってしまうので、余裕を持たせて辺の長さは 0.24 にしましょう。パスの幅は 0.02 なので、 **[サイズ [w, h] $]** に指定する値は (0.24, 0.02) です。

これで提示する図形についてはほぼ決まりましたが、まだプローブの大きさと色を決めていません。プローブの直径は 0.01 とします。プローブの色は、パスを白色で描画することを考慮すると、パスと重なっていても区別できるように白以外の色にする必要があります。パスや背景と区別が付けば何色でも構わないのですが、ここでは白い輪郭に赤色の塗りつぶしで描画することにしましょう。ついでに、ゴール地点に提示する円もわかりやすいように、白い輪郭に緑色の塗りつぶしで描画することにしましょう。

続いて実験の手続きを決定しないといけませんが、以下の解説では Builder の使い方を学ぶという目的を最優先して、ごくシンプルな手続きだけを作成します。 図 8.5 をご覧ください。各試行の最初に、プローブと共に、ゴール地点を示す円と色と大きさが等しい円をスタート地点に提示します。実験参加者がマウスの左ボタンをクリックすると、スタート地点の円がゴール地点へ移動すると同時にパスが提示されます。実験参加者はマウスを操作して、できる限りプローブがパスからはみ出ないように注意しながらプローブをゴール地点まで動かします。プローブの中心がゴール地点の円内に入ったことが検出された時点で試行は終了します。試行終了後は直ちに次の試行へ進みます。以上の手続きを、exp08cnd.xlsx に定義されている 10 種類の条件に対して 1 回ずつ、合計 10 試行を無作為な順序で実施したら実験は終了です。

この課題を実験実習などの授業で使用するならば、鏡映描写課題を練習した群としなかった群で比較したり、

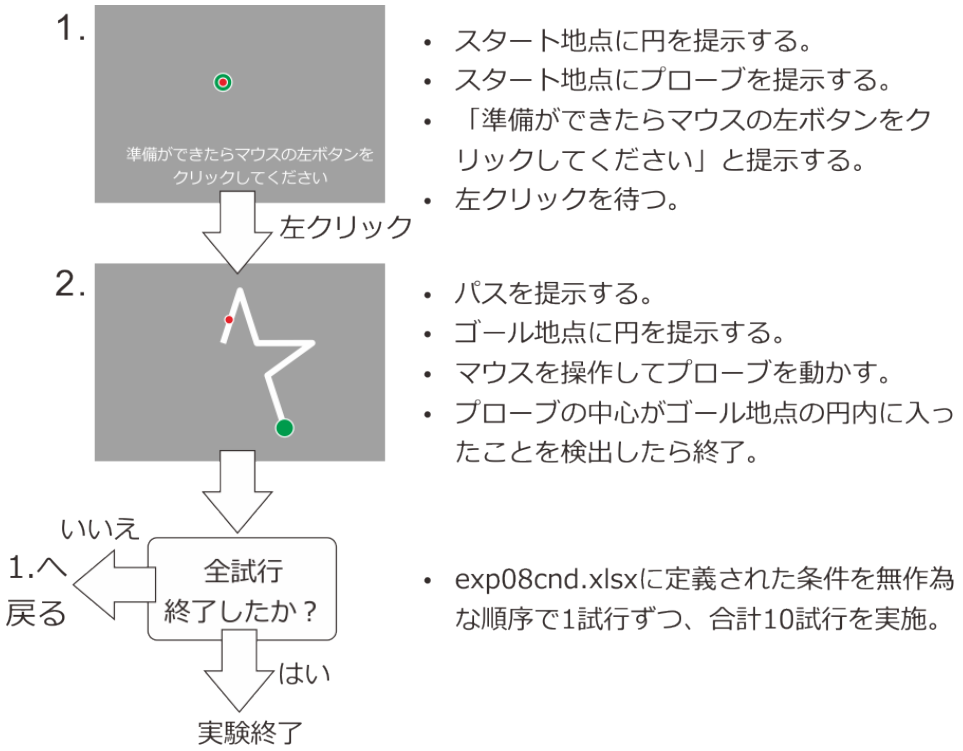


図 8.5　実験の流れ。

非利き手で練習して学習の効果が利き手にどの程度転移するかを調べたりするなどの工夫をする必要があります。この辺りは各自で工夫してみてください。

さて、以上で作成する実験の内容が決まりました。作成方法の解説を始めたいと思いますが、その前に本章で初登場の Mouse コンポーネントについて解説しておきましょう。

8.2 Mouse コンポーネント

Mouse コンポーネントは、その名の通り Builder からマウスを利用するためのコンポーネントです。3 ボタンのマウスを想定しているので、4 つ以上ボタンがあるマウスの場合は押されたことを検出できないボタンが出てきてしまいます。一般的なマウスの場合、3 つのボタンはそれぞれ「左クリック」と「右クリック」に使うボタンと、これらのボタンの間にあるボタンに対応します。 Mouse コンポーネントのアイコンは「反応」カテゴリにあります (図 8.6)。

Mouse コンポーネントのプロパティ設定ダイアログのうち、「基本」タブの **[名前]** 、 **[開始]** 、 **[終了]** は今までのコンポーネントと共通ですので解説は必要ないでしょう。 **[ボタン押しで Routine を終了]** はボタン押しでルーチンを終了するか否かを指定します。「なし」はボタンを押してもルーチンを終了しないということですが、ルーチンを終了させたい場合は「全てのクリック」と「有効なクリック」のどちらかを使用します。「全てのクリック」は文字通り、マウスボタンがクリックされたらどのような状況でも終了します。「有効なクリック」は「データ」タブにある **[クリック可能な視覚刺激]** で指定した刺激上でクリックされた場合のみルーチンを終了します。この機能については次章で詳しく取り上げます。

続いて「データ」タブです。まず、 **[新たなクリックのみ検出]** がチェックされていれば、マウスのボタン押し検出が始まった時点ですでにマウスのボタンが押されていた場合に、いったんボタンを離してから押し直さな

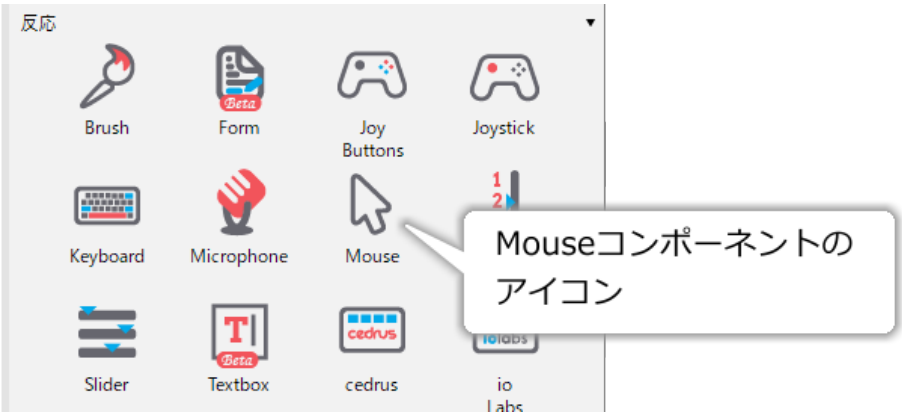


図 8.6　Mouse コンポーネントのアイコン。

いと検出しません。実験参加者がマウスのボタンを押しっぱなしにしていた場合にいきなりルーチンが終了してしまうことを防げます。

[マウスの状態を保存] はマウスのどのような情報が保存されるかを指定します。選択可能な項目については表 8.1 をご覧ください。 **[時刻の基準]** は Mouse コンポーネントで使用される時刻をルーチン開始時 (routine)、実験開始時、Mouse コンポーネント開始時のいずれのタイミングで 0 にするかを指定します。Mouse コンポーネント開始時というのは、その Mouse コンポーネントの **[開始]** で指定されたタイミングで 0 にするということです。ルーチンの途中から Mouse コンポーネントが有効になって、その時点を 0 秒としたい場合に便利です。

表 8.1　Mouse コンポーネントの **[マウスの状態を保存]** で選択可能な値

値	概要
最終	ルーチン終了時のマウスの状態が保存されます。 **[名前]** が foo の場合、実験記録ファイルの foo.x と foo.y にマウスカーソルの X 座標と Y 座標、foo.leftButton、foo.midButton、foo.rightButton にマウスの左ボタン、中央ボタン、右ボタンの状態が保存されます (0=押されていない、1=押されている)。座標の単位は実験設定ダイアログの **[単位]** に従います。
クリック時	マウスのいずれかのボタンが押されるたびに、その時のマウスの状態が保存されます。「最終」の時に出力される項目に加えて、押された時刻を保存する foo.time(foo は **[名前]** に入力した文字列) という項目が実験記録ファイルに出力されます。ボタンを押しっぱなしにしていると、その間 1 フレーム毎に状態が保存されていきます。
フレーム毎	マウスが有効な間 (**[開始]** から **[終了]** の間)、フレーム毎にマウスの状態が保存されます。保存形式は「クリック時」と同じです。60Hz のモニターを使っている場合 1 秒間に 60 件もデータが記録されますのでご注意ください。
なし	マウスの状態を保存しません。

注意する必要があるのは **[マウスの状態を保存]** の値によってどのようなデータが保存されるかです。まず、「最終」ではボタン押しの有無にかかわらず、とにかくルーチン終了時のマウスの状態が保存されます。この点は Keyboard コンポーネントと異なっています。マウスカーソルの座標の単位は実験設定ダイアログの **[単位]** に従います。

「クリック時」では、 **[開始]** と **[終了]** で指定された期間内にマウスのボタンを押すとマウスの状態が記録されます。このように書くと「クリックした回数だけ記録される」と思われてしまうかも知れませんが、そうではなくて「ボタンが押されている間ずっとフレーム毎に」記録され続けます。つまり、リフレッシュレートが 60Hz のモニターを使用していてマウスのボタンを 1 回、1 秒間押し続けると、1 件ではなく 60 件のデータが保存されます。実験記録ファイルには、カーソルの X 座標の値が [0.48, 0.49, 0.51]、Y 座標の値が [0.09,-0.03, -0.11] といった具合に Python のリストの形で保存されたデータが出力されています。この例の場合、ボタンが押されたときのマウスカーソルの座標は記録された順番に [0.48, 0.09]、[0.49, -0.03]、[0.51, -0.11] だったということです。このように 3 件のデータある場合、「素早く 3 回クリックされた」のか「1 回、3 フレーム分の時間ボタンを押した」のか「1 回素早くクリックし、1 回 2 フレーム分の時間ボタンを押した」のかを区別するには、ボタンが押された時刻の出力を確認する必要があります。 表 8.1 に記したとおり、時刻は実験記録ファイルの foo.time という列に同じ書式で出力されています (foo は **[名前]** に指定した文字列)。

また、当然ですが **[ボタン押しで Routine を終了]** がチェックされている状態では、「クリック時」が設定されていていても、ボタン押しが検出された時点でルーチンが終了してしまうので 1 件しかデータが保存されません。

続いて「フレーム毎」ですが、これを選択すると「クリック時」と同様のデータが Mouse コンポーネントが有効な期間中ずっと記録され続けます。最後の「なし」を選択すると、Mouse コンポーネントは実験記録ファイルに何も出力しません。

これから作成する実験では、「最終」でルーチン終了時の状態を記録しても意味がありません。「クリック時」にしたら参加者が操作した軌跡が保存されるので役に立ちそうな気がしますが、実際に保存させるには実験参加者にずっとマウスのボタンを押さえつづけてもらわないといけません。参加者への負担になりますし、何よりちゃんと押してくれなかった場合にデータが取得できないのでは話になりません。「フレーム毎」にすると軌跡は保存されますが、同時にマウスのボタンの状態も保存されて記録ファイルが非常に大きくなってしまいます。操作記録には Code コンポーネントを使うことにして、とりあえず「なし」を使う事にします。

Mouse コンポーネントにはまだ設定項目がありますが、解説はここでいったん区切りとして実験の作成に入りましょう。

チェックリスト

- ルーチン終了時のマウスカーソルの位置およびマウスのボタンの状態を記録することができる。
- ボタンが押された時点のマウスカーソルの位置およびマウスのボタンの状態を記録することができる。
- 実験記録ファイルにマウスカーソルの座標やボタンの状態がリストとして複数件出力されている時に、個々のデータの座標値やその取得時刻を判断できる。

8.3 実験の作成

まずは前章までに解説済みのテクニックで作成できる部分を作成します。Builder で新規に実験を作成して以下の作業を行い、exp08a.psyexp という名前で保存するものとします。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
- trial ルーチン
  - Mouse コンポーネントをひとつ配置して、 **[名前]** を mouseTrial にする。
    - 「基本」タブの **[終了]** を空白にする。
    - 「基本」タブの **[ボタン押しで Routine を終了]** を「なし」にする。
    - 「データ」タブの **[マウスの状態を保存]** を「なし」にする。
  - Code コンポーネントをひとつ配置し、 **[名前]** を codeTrial にする。
  - Polygon コンポーネントを 1 つ配置して以下の作業を行う。 このさぎょうにより path1、path2、path3、path4、path5 という 5 つの Polygon コンポーネントが作成される。
    - **[名前]** を path1 にする。
    - **[終了]** を空白にする。
    - **[形状]** を長方形にする。
    - **[塗りつぶしの色]** を white にする。
    - **[回転角度 $]** を path1ori にして、「繰り返し毎に更新」に設定する。
    - **[位置 [x, y] $]** を path1pos にして、「繰り返し毎に更新」に設定する。
    - **[サイズ [w, h] $]** を (0.24, 0.02) に設定する。
    - path1 をコピーして「Component の貼り付け」で貼り付け、**[名前]** を path2 にする。 **[回転角度 $]** を path2ori、**[位置 [x, y] $]** を path2pos にする。以後、同様の手順で path3, path4, path5 を作成する。 **[回転角度 $]** と **[位置 [x, y] $]** も **[名前]** に合わせて path2ori, path2pos, path3ori, path3pos,... と変更する。
  - Polygon コンポーネントを 2 つ配置して **[終了]** を空白にし、 **[名前]** を goalDisc、probe にする。必ず goalDisc は path1〜path5 より上に描画されるようにし、probe は goalDisc よりも上に描画されるようにする。
  - goalDisc について以下の作業を行う。
    - **[終了]** を空白にする。
    - **[形状]** を円にする。
    - **[塗りつぶしの色]** を green にする。

    * **[位置 [x, y] $]** を goalPos にし、「繰り返し毎に更新」に設定する。
    * **[サイズ [w, h] $]** を (0.03, 0.03) にする。
  - probe について以下の作業を行う。
    * **[終了]** を空白にする。
    * **[形状]** を円にする。
    * **[塗りつぶしの色]** を red にする。
    * **[位置 [x, y] $]** を [px, py] にし、「フレーム毎に更新する」に設定する。
    * **[サイズ [w, h] $]** を (0.01, 0.01) にする。
- ready ルーチン (作成する)
  - フローの trial ルーチンの前に挿入する。
  - Mouse コンポーネントをひとつ配置して、 **[名前]** を mouseReady にする。
    * **[終了]** を空白にする。
    * **[ボタン押しで Routine を終了]** が「すべてのクリック」になっていることを確認する。
    * **[マウスの状態を保存]** を「なし」にする。
  - Polygon コンポーネントを 2 つ配置して **[終了]** を空白にし、 **[名前]** を startDisc、probeReady にする。probeReady は startDisc よりも上に描画されるようにする。
  - startDisc について以下の作業を行う。
    * **[形状]** を円にする。
    * **[塗りつぶしの色]** を green にする。
    * **[位置 [x, y] $]** を startPos にし、「繰り返し毎に更新」に設定する。
    * **[サイズ [w, h] $]** を (0.03, 0.03) にする。
  - probeReady について以下の作業を行う。
    * **[形状]** を円にする。
    * **[塗りつぶしの色]** を red にする。
    * **[位置 [x, y] $]** を startPos にし、「繰り返し毎に更新」に設定する。
    * **[サイズ [w, h] $]** を (0.01, 0.01) にする。
  - Text コンポーネントをひとつ配置する。
    * **[文字の高さ $]** を 0.03 にする。
    * **[位置 [x, y] $]** [0, -0.32] にする。
    * **[文字列]** に「準備ができたらマウスの左ボタンをクリックしてください」と入力する。

- trials ループ (作成する)
  - ready ルーチンと trials ルーチンを繰り返すように挿入する。
  - **[繰り返し回数 $]** を 1 にする。
  - **[繰り返し条件]** に exp08cnd.xlsx を指定する。

ここまで準備できれば、あとは ready ルーチンと trials ルーチンの Code コンポーネントに必要なコードを記入するだけです。まず、trials ルーチンの Code コンポーネントで実現すべき処理を整理して、何が今まで学んできたことではできないのかをはっきりさせましょう。まだ exp08a.psyexp で実現できていない処理は以下の 3 点です。

1. 現在のマウスカーソルの座標を得る。
2. マウスカーソルの動きを上下反転させた場合の座標を計算する。
3. 反転させた座標が goalDisc に含まれているか判定し、含まれていたらルーチンを終了する。
4. 試行開始時にマウスカーソルをスタート地点に移動させる。

1 は未解説の処理です。先ほど解説した Mouse コンポーネントの機能には、ルーチン実行中に座標を得るという機能は含まれていませんでした。2 もちょっと考える必要がありそうです。3 は 第 6 章 、第 7 章 で扱ってきた if 文が利用できそうです。「指定した条件に当てはまればルーチンを終了する」という処理は 第 7 章 ですでに解説しました。ですから、残っているのは「座標が goalDisc に含まれているか否かをどうやって判定すればよいか」という問題だけです。4 も未解説の処理ですね。

1 と 4 の問題を解決するには、PsychoPy のマウス制御用クラスである psychopy.event.Mouse クラス (以下 Mouse クラス) を利用する必要があります。Buildern の Mouse コンポーネントもこのクラスを利用しています。まずは Mouse クラスの解説から始めることにしましょう。

8.4 psychopy.event.Mouse クラスのメソッドを利用してマウスの状態を取得しよう

第 6 章 で解説したように、Builder 上で Grating コンポーネントを配置すると、その **[名前]** に設定した変数に Grating クラスのインスタンスが格納されます。それと同様に、Builder 上で Mouse コンポーネントを配置すると **[名前]** に設定した変数に Mouse クラスのインスタンスが格納されます。現在作成中の exp08a.psyexp では、trial ルーチンに mouseTrial という **[名前]** の Mouse コンポーネントを配置したのですから、Code コンポーネントからは mouseTrial という変数を通して trial ルーチン上の Mouse クラスを利用することができます。

Mouse クラスは、データ属性に直接アクセスするのではなく、メソッドを通してすべての処理を行うように設計されています。ですから、ここではメソッドのみを紹介しましょう 表 8.2 に代表的な Mouse クラスの主なメソッドを示します。

表 8.2　psychopy.event.Mouse クラスの主なメソッド

メソッド	概要
clickReset(buttons=(0,1,2))	マウスの反応時間計測用タイマーをリセットします。buttons にはリセットしたいボタンを並べたリストを指定します。0=左ボタン、1 ＝中央のボタン、2=右ボタンです。
getPos()	現在のマウスカーソルの座標をリストとして取得します。リストの最初の要素は X 座標、次の要素は Y 座標を示しています。
getPressed(getTime=False)	現在のボタンの状態をリストとして取得します。リストの要素は順番に左ボタン、中央のボタン、右ボタンに対応していて、0=押されていない、1=押されている、です。getTime=True とすると、最後に clickReset を呼んでからの時間も返されます。
setPos(newPos=(0,0))	マウスカーソルの位置を newPos に移動させます。
setVisible(visible)	visible に True を指定すると、マウスカーソルが描画されます。False を指定すると、マウスカーソルは描画されません。
isPressedIn(shape, buttons=(0,1,2))	引数 shape で指定したオブジェクトの内部にマウスカーソルがある状態で buttons に指定したボタンが押されていれば True、押されていなければ False を返します。

getPos( ) と getPressed( ) を利用すれば、Mouse コンポーネントで出力される程度の情報を取得することができます。注目してほしいのは、clickReset( ) や getPressed( ) におけるボタンの番号です。clickReset( ) では、マウスの左ボタンが 0、中央のボタンが 1、右ボタンが 2 で表されます。一方、getPressed( ) の戻り値として得られるマウスボタンの状態は [0, 0, 1] といった具合に 0 または 1 が 3 つ並んだリストで、1 番目が左ボタン、2 番目が中央のボタン、3 番目が右ボタンに対応しています。clickReset( ) のボタン番号と getPressed( ) の戻り値のリストにおけるボタンの順番が一致しているのは一目瞭然ですが、なぜボタン番号が 1、2、3 ではなく 0、1、2 なのでしょうか。それは、Python の文法ではリストの要素の順番を数える時には 1 からではなく 0 から始めるからです。ですから、Python にとっては 0、1、2 と番号が割り当てられている方が自然なのです。「0 から数え始める」ということは次節以降の内容を理解する上で非常に重要なので必ず覚えておいてください。

以上を踏まえて Builder の作業に戻りましょう。exp08a.psyexp の trial ルーチンを開いて、codeTrial の **[フレーム毎]** に以下のコードを入力してください。

```
mousePos = mouseTrial.getPos()
```

これで、フレーム毎に mousePos という変数にマウスカーソルの位置が代入されるようになりました。続いて probe の動きをマウスカーソルの動きと上下方向だけ反対にする方法を考えます。

なお、ひとつ補足しておきますと、今回とは違ってマウスカーソルの位置に合わせて刺激の位置を変更するのであれば、Code コンポーネントを使う必要はありません。動かしたい刺激の **[位置 [x, y] $]** に以下のように入力して「フレーム毎に更新する」に設定するだけで目的を達成できます (Mouse コンポーネントの **[名前]** は mouseTrial とする)。

```
mouseTrial.getPos( )
```

なぜこれだけで済むか、おわかりでしょうか。 getPos( ) メソッドの戻り値は X 座標、Y 座標の値を並べたリ

ストであり、 **[位置 [x, y] $]** に記述すべき値とデータの型が一致しているから、というのが理由です。関数やメソッドを見た時に、その戻り値の型を思い浮かべる習慣をつけるようにしてください。

チェックリスト

- Code コンポーネントでマウスカーソルの座標を取得するコードを記述できる。
- Code コンポーネントでマウスのボタンの状態を取得するコードを記述できる。
- getPressed() メソッドを用いて取得したマウスのボタンの状態を示すリストの各要素がどのボタンに対応しているか答えられる。また、リストの各要素の値とボタンの状態の関係を答えられる。
- Code コンポーネントを用いずにマウスカーソルの位置に刺激を描画することができる。

8.5 リストの要素にアクセスしてマウスカーソルと上下反対にプローブを移動させよう

変数 mousePos に現在のマウスカーソルの座標を代入することができたので、次はマウスカーソルと上下反対にプローブを移動させる方法を考えましょう。 図 8.7 の左図をご覧ください。マウスカーソルと上下方向だけ反対にプローブが移動するということは、マウスカーソルが (dx, dy) 移動したときにプローブは (dx, -dy) 移動するということです。

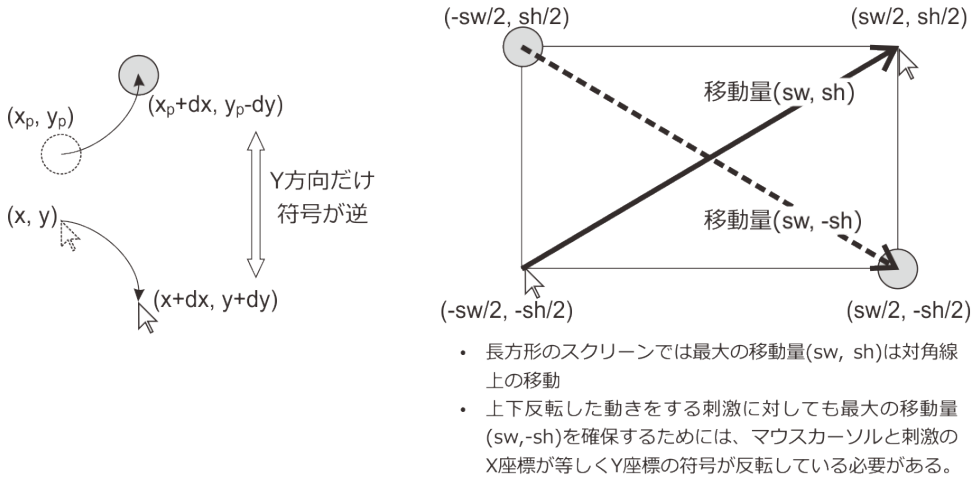


図 8.7　マウスカーソルと上下方向だけ反転して移動するプローブ。マウスカーソルの Y 座標の符号を反転した座標をプローブの座標とすれば、最大の移動量 (対角線上の運動) に対応することができます。ただし sw、sh はスクリーン幅、スクリーン高を表しているとします。

この要件さえ満たせばどのようにプローブの座標を決定しても構わないのですが、ここではマウスカーソルを最大限移動させてもプローブが追従できるように座標を決定する方法を考えます。PsychoPy でサポートしているスクリーンは長方形ですから、マウスカーソルの移動量が最大となるのは対角線上を頂点から頂点まで移動させた時です。マウスカーソルを幅 sw、高さ sh のスクリーンの左下から右上の頂点へ移動させるとすると、マウスカーソルの座標は (-sw/2, -sh/2) から (sw/2, sh/2) まで移動し、移動量は (sw, sh) です。この時、プローブは (sw, -sh) だけ移動しないといけませんが、(sw, -sh) の移動量を確保するためにはプローブは (-sw/2, sh/2) から (sw/2, -sh/2) へ移動する以外あり得ません。これらの座標を比較すると、マウスカーソルの Y 座標の符号を反転した座標をプローブの座標とすれば、最大の移動量に対応できることがわかります (図 8.7 右)。

以上を踏まえたうえで、Code コンポーネントに入力するコードを考えましょう。マウスカーソルの X 座標と Y 座標がそれぞれ mx、my という変数に格納されているのであれば、[mx, -my] と書けば目的を達成できます。しかし、今回は mousePos という変数に X 座標と Y 座標をまとめてひとつのリストとして代入されているので、このような書き方ができません。mousePos に代入されたリストから、X 座標と Y 座標を個別に抜き出す必要があります。

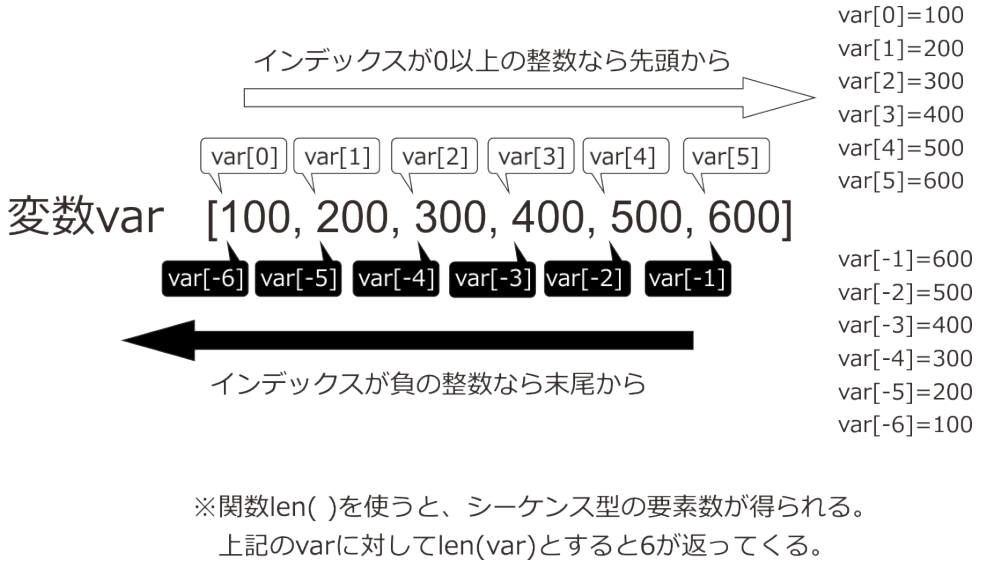


図 8.8　シーケンス型のデータからの要素の取出し。インデックスが 0 以上の整数なら先頭から、負の整数なら末尾から数えます。

Python の文法において、リストをはじめとするシーケンス型のデータから一部の要素を取り出すには、[ ] 演算子を使用します。[ ] 演算子は、mousePos[1] という具合にシーケンス型のデータの後に添えて、[ ] の中に何番目の要素を取り出すかを指定します。この指定のことをインデックスと呼びます。 図 8.8 にインデックスとして整数を指定した時の動作を示します。インデックスが 0 以上の整数の場合、最初の要素を 0 として順番に先頭 (左) から数えた位置にある要素を取り出します。現在変数に格納されているシーケンス型データに何個の要素が含まれているかわからない場合は len( ) という関数を用います。len( ) は、引数に与えられたデータに含まれる要素数を返します。インデックスとして負の整数が指定された場合 (負のインデックス) は、最後の要素を-1 として末尾 (右) から数えた位置にある要素を取り出します。0 から数えるというのは先ほど getPressed( ) メソッドについて解説した時に述べたとおりですね。[ ] 演算子を使用して、以下のようなコードを書くと変数 mousePos から X 座標、Y 座標を取り出してそれぞれ変数 px、py に代入することができます。

```
px = mousePos[0]
py = mousePos[1]
```

今回は Y 座標の符号を反転してプローブの座標として用いたいのですから、以下のように-演算子を組み合わせて代入と符号反転を一度に済ませると便利です。

```
px = mousePos[0]
py = -mousePos[1]
```

これらの文を exp08.py の trial ルーチンの codeTrial の **[フレーム毎]** に追加してください。追加後の **[フレーム毎]** は以下のようになります。

```
mousePos = mouseTrial.getPos( )
px = mousePos[0]
py = -mousePos[1]
```

trial ルーチンの probe の **[位置 [x, y] $]** に [px, py] と設定されておりますので、これでマウスカーソルと上下反対方向に移動するプローブが完成しました。この節の目標は達成されましたが、せっかく [ ] 演算子が出てきましたので、シーケンス型の扱いについてもう少し学んでおきましょう。まず、今回のように符号の反転などの操作が必要ないのであれば、以下の書式で、関数やメソッドの戻り値として渡されたシーケンス型データを要素に分解して別々の変数に代入することができます。

```
px, py = mouseTrial.getPos( )
```

この書式を使用する場合、=演算子の左辺にカンマ区切りで並べられた変数の個数と、戻り値として渡されるシーケンス型データの要素数が一致している必要があります。一致していない場合、エラーでスクリプトの処理が停止してしまいます。

続いて補足しておきたいのは、多重シーケンスからの要素の取出しです。シーケンス型は非常に柔軟なデータ型で、入れ子のようにシーケンスの要素としてシーケンスを含むことができます。このような多重のシーケンスに対して [ ] 演算子を適用した例を 図 8.9 に示します。この例では ['A', 'B', 'C'] というリストと [1, 2, 3, 4] というリストを要素とするリストが変数 var に格納されています。var[0] は ['A', 'B', 'C']、var[1] は [1, 2, 3, 4] です。var[0] はリストなのですから、これに対してさらに [ ] を適用することができて、var[0][0] は'A'、var[0][1] は'B' です。同様に、var[1][3] とすると 4 が得られます。もちろん負のインデックスも組み合わせて使用できますので、var[1][3] は var[1][-1] と書くこともできます。

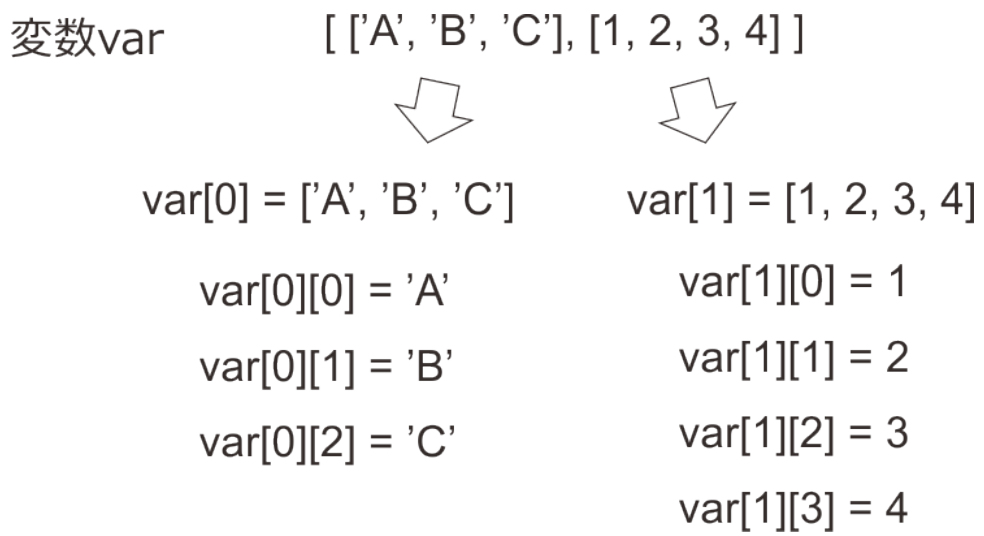


図 8.9　多重シーケンスからの要素の取出し。var[0] とするとリストが得られるので、さらに [ ] 演算子を適用して var[0][0] とすると var[0] の最初の要素を取り出せます。

少しややこしいという注意が必要な点として、リストを作る [ ] とシーケンス型の要素を取り出す [ ] 演算子の違いがあります (図 8.10)。リストを作る [ ] の前には変数も値もありません。一方、要素を取り出す [ ] 演算子は必ずその前にシーケンス型のデータがあります。「シーケンス型のデータがあればよい」ということは、[0.24, 0.02][0] という具合に変数ではなくリスト等のデータが前に直接置かれていても構わないということです。 [0.24, 0.02][0] は「[0.24, 0.02] という要素数 2 のリストの最初の要素」ということですから、その値は

240 です。よろしいでしょうか？

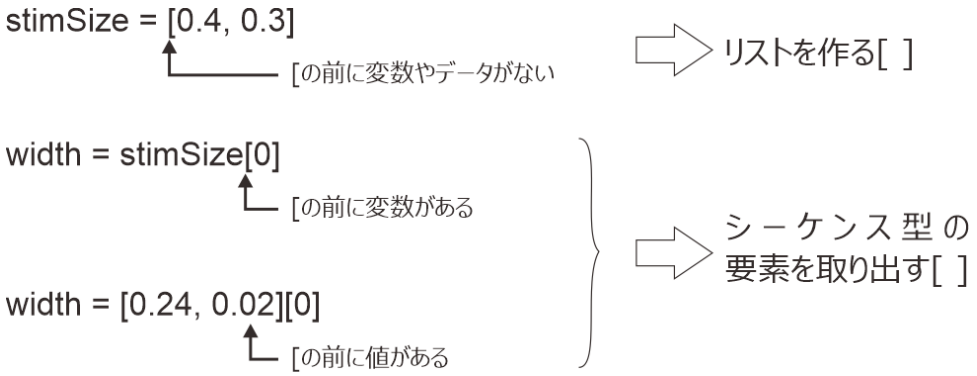


図 8.10　リストを作る [ ] とシーケンス型の要素を取り出す [ ]。

シーケンス型に対する [ ] 演算子の使い方には解説したいテクニックがたくさんあるのですが、どんどんこの章の実験から離れていってしまいますので、ひとまずこのくらいにしておきましょう。ここで述べただけでも Builder を便利に拡張する様々なコードを書くことができます。終わりと言いつつ最後にひとつだけ付け足しておきますと、[ ] 演算子は文字列に対しても同様に使用することができます (というか文字列はシーケンス型の仲間です)。'Psychology' という文字列が格納されている変数 s に s[3] とすると'c' が得られますし、s[-2] とすると'g' が得られます。覚えておいてください。

チェックリスト

- シーケンス型のデータから要素をひとつ取り出して他の変数に代入したり関数の引数に使ったりすることができる。
- シーケンス型データの前から N 番目の要素を取り出す時の式を答えられる。
- シーケンス型データの後ろから N 番目の要素を取り出す時の式を答えられる。
- 関数を用いてシーケンス型データに含まれる要素数を調べることができる。
- シーケンス型のデータが入れ子構造になっている時に、要素であるシーケンス型データや、さらにその要素を取り出すことができる。
- 文字列の中から N 番目の文字を取り出して変数に代入したり関数の引数に使ったりすることができる。
- [1,2,3][0] といった具合に [から始まって、] の後に [ ] 演算子が続く式を評価した時の値を答えられる。

8.6 刺激の重なりを判定しよう

プローブの動きが実現できたので、続いてプローブの中心がゴール地点の円内に入ったことを判定する方法を考えましょう。これは「プローブの座標がゴール地点の座標を中心とした半径 0.015 の円内に入る」ことと等しいので、以下の式で判定することができます。さっそく [ ] 演算子の復習になっているので、よくわからなければ前節を復習してください。

```
(goalPos[0]-px)**2 + (goalPos[1]-py)**2  < 0.015**2
```

念のため補足しておきますが、ゴール地点の座標は変数 goalPos に格納されていて、ゴール地点に置かれている Polygon コンポーネント goalDisc の **[サイズ [w, h] $]** は (0.03, 0.03) です。 **[サイズ [w, h] $]** は刺激の幅と高さ、すなわち直径に相当しますので、半径は 0.015 です。この式を if 文の条件式に用いれば目的は達成できるのですが、この節ではもっと便利な方法を学びましょう。

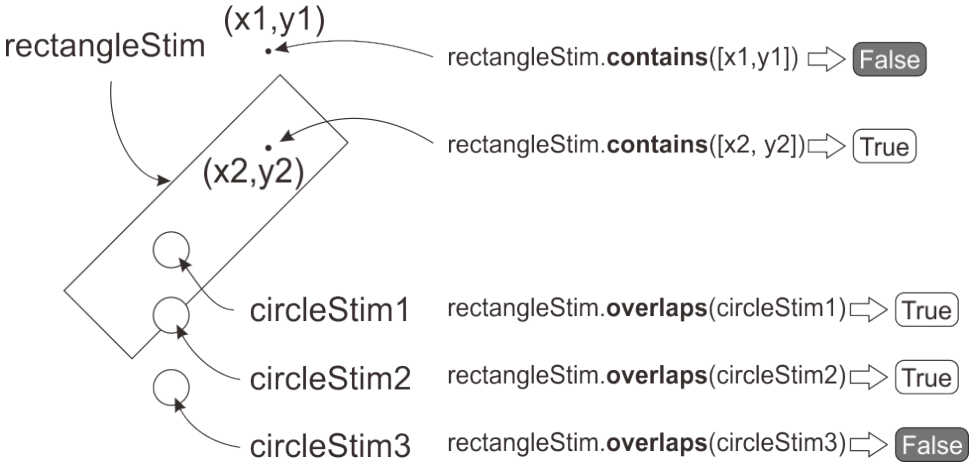


図 8.11　Contains( ) と Overlaps( ) メソッドの例。

Polygon コンポーネントに対応するクラスが複数あるのは 第 6 章 で述べたとおりですが、 **[頂点数]** が 3 以上の時に用いられるクラスでは、contains( ) と overlaps( ) というメソッドが用意されています。これは今回の実験にはうってつけのメソッドで、contains( ) は引数に指定された座標が内側に含まれていれば True、いなければ False を返します。overlaps( ) は、引数に指定された Polygon コンポーネントと重なっていれば True、いなければ False を返します。文章で説明するより図の方がわかりやすいでしょう。 図 8.11 は rectangle という名前の Polygon コンポーネントから contains( ) と overlaps( ) を呼び出した例を示しています。判定の対象となる図形が 図 8.11 のように傾いている場合、先ほどのような簡単な条件式では図形の中に点が含まれているかを判定することはできませんので、contains( ) と overlaps( ) は非常にありがたいメソッドです。もちろん図形が三角形や五角形など、Polygon コンポーネントで描画できる多角形であれば適切に判定が行われます。

ここまで解説が進めば、もう「プローブの中心がゴール地点の円内に入ったらルーチンを終了する」という動作を実現するのは簡単です。円内に入った判定には contains( ) を使用し、ルーチンを終了するには continueRoutine に False を代入すればいいのですから

```
if goalDisc.contains([px, py]):
    continueRoutine = False
```

とすればよいはずです。このコードを、trial ルーチンの codeTrial の **[フレーム毎]** に追加しましょう。念のた

め、追加後のコード全体を示しておきます。これで trial ルーチンは完成しました。

```
mousePos = mouseTrial.getPos( )
px = mousePos[0]
py = -mousePos[1]
if goalDisc.contains([px, py]):
    continueRoutine = False
```

これでマウスカーソルの現在位置の取得、マウスカーソルと上下反転した移動をするプローブ、プローブとゴール地点との重なりを判定してルーチンの終了、の課題をクリアしました。ここで一度、exp08a.psyexp を保存して実行してみてください。狙い通りの動作が実現できているはずです。残るは試行開始時のマウスカーソルの位置を設定するだけです。

チェックリスト

- ある座標が Polygon コンポーネントで描画した多角形の内側に含まれているか判定するコードを書くことができる。
- Polygon コンポーネントで描画した二つの多角形に重なっている部分があるが判定するコードを書くことができる。

8.7 カーソルの位置を設定し、カーソルの表示 ON/OFF を制御しよう

Mouse クラスのメソッドを使用すれば、カーソル位置の設定は何も難しいことはありません。 表 8.2 で紹介済みの setPos( ) メソッドを使用します。

注意しないといけないのは、今回の実験では「スタート位置にプローブを置くためには、マウスカーソルの位置はスタート位置の座標と上下反対の位置になければいけないという点です。前節ではマウスカーソルの位置をプローブ位置に変換しましたが、逆にプローブ位置をマウスカーソルの位置に変換しないといけないわけです。

変換といっても単に Y 座標の符号を反転すればよいだけですから、処理は非常に単純です。startPos にスタート位置の座標が格納されているのですから、[startPos[0], -startPos[1]] とすればよいでしょう。これを setPos( ) の引数にすればよいのですから、以下のコードを実行すればプローブのスタート位置に対応する位置へマウスカーソルを設定することができます。

```
mouseTrial.setPos([startPos[0], -startPos[1]])
```

試行開始時にカーソルがスタート位置にないといけないのですから、このコードは trial ルーチンの開始時に実行するべきです。trial ルーチンに配置してある codeTrial の **[Routine 開始時]** に追加しましょう。

これで実験の基本的な部分は完成ですが、実験の間ずっとマウスカーソルが表示されたままになっているのが気になります。実験設定ダイアログに Show mouse というマウスカーソルを表示させるための項目はあるのですが、このチェックを外してもマウスカーソルは消えません。 Show mouse は Mouse コンポーネントを使っていない時にマウスカーソルを表示したい (ウィンドウモードで他アプリケーションと一緒に使用したいときなど) 場合に使用するためのもので、Mouse コンポーネントを使用するとマウスカーソルは強制的に表示

されます。まあ普通はマウスを使用する時にはカーソルが表示されているのが当然なのでこのような仕様になっているのでしょうが、今回の実験の場合では余計です。マウスカーソルを非表示にするには、 表 8.2 の setVisible( ) メソッドを使用します。以下のコードを trial ルーチンの Code コンポーネントの **[実験開始時]** に入力してください。

```
mouseTrial.setVisible(False)
mouseReady.setVisible(False)
```

入力を終えたら、exp08a.psyexp を保存して実行してください。今度はマウスカーソルが表示されないはずです。このコードを追加してもなおカーソルが表示される場合は、実験設定ダイアログの **[キーボードバックエンド]** の設定を確認してください (筆者の環境では **[キーボードバックエンド]** が ioHub に設定されていると setVisible() が無効になる場合がありました)。

これで 図 8.5 に示した実験の流れは完成しましたが、現在の状態では参加者の反応が一切記録されません。心理学実験の実習であれば、ストップウォッチなどを利用しながら手作業で記録するのも良いのですが、せっかく PC を使っているのですからやはり記録も PC で行いたいところです。次節では、反応の記録方法について考えてみましょう。

チェックリスト

- マウスカーソルの位置を設定するコードを書くことができる
- マウスカーソルの表示 ON/OFF を切り替えるコードを書くことができる。

8.8 for 文を用いて複数の対象に作業を繰り返そう

今回の実験では、実験参加者のどのような反応を記録すればよいでしょうか。真っ先に思い浮かぶのは、課題遂行中の (つまり trial ルーチン実行中の) プローブの軌跡をすべて記録することです。しかし、ただプローブの軌跡を記録しただけでは、プローブがパスの上にきちんと乗っているのか後から計算しないといけません。パスに用いる長方形の中心座標値の計算 (図 8.3) よりもさらに面倒な計算なので、できることならしたくありません。せっかく contains( )、overlaps( ) というメソッドを覚えたのですから、これを利用してプローブがパス上にあるか否かを判定しましょう。

今回の課題では、パスを構成する Polygon コンポーネントが 5 個あり、このいずれかの上にプローブがあれば、パス上にプローブが乗っています。すでに今まで学んだ Python の文法の範囲でも何通りかの方法でこの処理を記述することができますが、ここでは以下のようなコードを考えてみましょう。

```
onPath = False
if path1.contains([px, py]):
    onPath = True
if path2.contains([px, py]):
    onPath = True
if path3.contains([px, py]):
    onPath = True
if path4.contains([px, py]):
```

(次のページに続く)

(前のページからの続き)

```
    onPath = True
if path5.contains([px, py]):
    onPath = True
```

最初に onPath という変数に False を代入しておいて、path1 から path5 まで順番に contains( ) を実行して戻り値が True であれば onPath を True にします。最後の行まで進んだ時点で onPath が True であればいずれかの上にプローブが乗っています。2 つ目の if からは elif のほうが効率がいいじゃないの？と思われた方は良い点に注目しておられますが、以下の解説と対応づけるためにわざとこうしていますのでご理解ください。

さて、このコードは確かに目的とする処理は達成できるのですが、、同じような文がだらだらと続いて読みづらいですし、「やっぱり contains( ) じゃなくて overlaps( ) を使うことにしよう」などと思ったときに書き換えが面倒です。うっかり 1 行だけ書き換え忘れたりすると、間違えた場所を見つけるのは非常に厄介です。この例のように「変数が異なるだけで同じ処理を繰り返す」時に非常に有効な for という文が Python には用意されています。

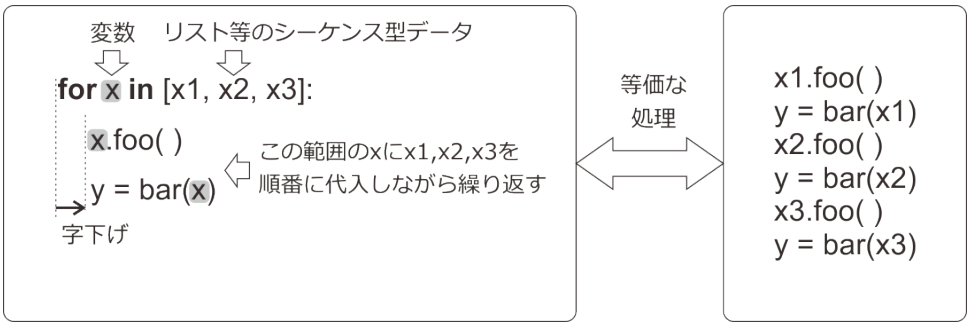


図 8.12　for 文による繰り返し処理。

図 8.12 に for 文の例を示します。for 文は「for 変数 in シーケンス型データ:」という形で使用し、シーケンス型データの先頭から要素をひとつずつ取り出して変数に代入しながら処理を繰り返します。繰り返しの範囲は if 文と同様に、字下げ量が for の出現する行と同じかそれより少ない行の手前まで及びます。また、for が出て来る行の最後にコロンが必要な点も if 文と同じなので忘れないようにしてください。 図 8.12 左の例では、for に続く変数として x を使用していますので、x に x1 を代入して繰り返し処理を行い、続いて x に x2 を代入して繰り返し処理を…という具合に実行されます。結果的に、 図 8.12 左の for 文は、 図 8.12 右に示したコードと等価な処理を行います。for 文を使うと、path1 から path5 まで contains( ) メソッドを実行するコードは以下のように短縮できます。

```
onPath = False
for path in [path1, path2, path3, path4, path5]:
    if path.contains([px, py]):
        onPath = True
```

とりあえずこれで「プローブがパス上にあるか判定する」という目標は達成されましたが、for 文についてもう少し勉強しておきましょう。今回は 5 個ある長方形のどれにプローブが乗っていても構わないので、もし path1 の上にプローブが乗っていれば (すなわち path1.contains([px, py]) の戻り値が True であれば)、path2 以降について contains( ) を実行する必要がありません。今時の PC の性能であれば残り 4 回余分に contains( ) を実行しても大した負担になりませんが、PC の処理遅延による刺激の描画漏れや反応時間計測の遅延が起こ

る可能性を少しでも下げたいと思うのであれば PC に余分な処理をさせたくありません。このように、for 文を最後まで繰り返す前に「これ以上 for 文を繰り返す必要はない」という状況になった場合は、break という文を使います。 図 8.13 に break の使用例を示します。for 文による繰り返し処理を遂行中に break が実行されると、まだ繰り返すべき処理が残っていても直ちに for 文が終了します。 図 8.13 の場合、プローブの中心が path3 の上に乗っていますので、path に path1、path2 を代入して実行している時は if 文の式が True になりません。break は if 文の中にあるので実行されず、次の繰り返しへ進みます。そして path に path3 が代入された時、if 文の式が True になるので break が実行され、直ちに for 文による繰り返しが終了します。したがって、path4、path5 に対する処理は実行されません。

この節最初で if 文をずらずらと並べたコードを示したときに「2 番目以降は elif 文のほうが効率がいいんじゃないのか？」と思った方は、この break による for 文の中断が elif による効率化と対応することがおわかり頂けると思います。

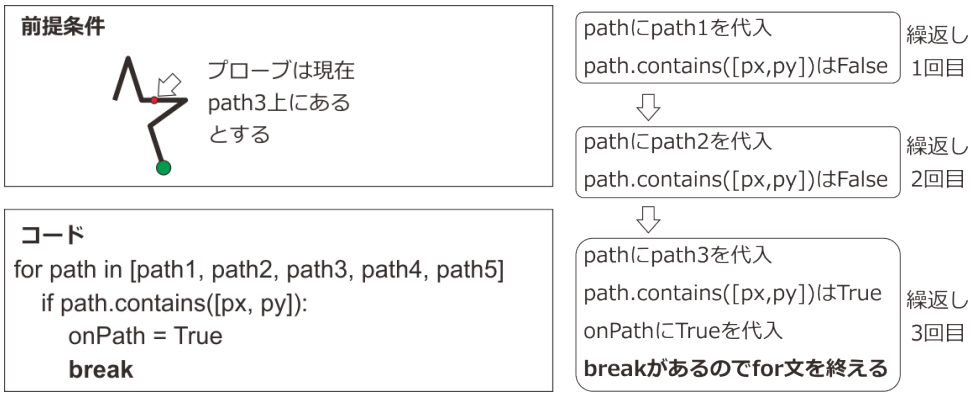


図 8.13　break による for 文の中断。if 文などと組み合わせて使用するのが一般的です。

この章の実験を完成させるためにはここまで学んでいれば十分なのですが、今後皆さんが自分の実験を作成する時のための知識を高める、という観点でもうひとつ for 文について紹介しておきたい点があります。少々高度な内容を例題として取り上げますので、その前にいったん節を区切りましょう。

チェックリスト

- ある変数に格納されている値だけが異なる処理を繰り返し実行しなければいけない時に、for 文を用いて記述することができる。
- for 文を継続する必要がなくなった時に直ちに終了させることができる。

8.9 ルーチンに含まれる全コンポーネントのリストから必要なものを判別して処理しよう

この節で紹介するコードは、この章の実験の完成版では使用しませんので、ここまで作業してきた exp08a.psyexp に手を加えずに読み進めてください。コードを入力して実行させたい場合は、exp08a.psyexp を別名で保存して作業してください。

それでは for 文の勉強を再開しましょう。for 文の実行を制御す文として、break と併せて覚えておきたいのが continue です。continue は、現在実行中の繰り返しを終了させて、次の繰り返しに移行させる時に使用します。 図 8.14 は break と continue の働きを示した図です。break はもう for 文をこれ以上続ける必要がないと

きに、continue は現在の要素に対してはもう処理する必要がないけれども残りの要素に対して処理を続ける必要あるときに使用します。この節では、continue を使用例として、「ルーチンに含まれるコンポーネントから特定のコンポーネントを探して処理をする」という処理をするコードを作成します。この「特定のコンポーネントを探す」という処理が少々難解なので、「とりあえず Builder の基本的な使い方をマスターして自分の実験を作ってみたい」という方はよくわからなくても先に進んでいただいて構いません。

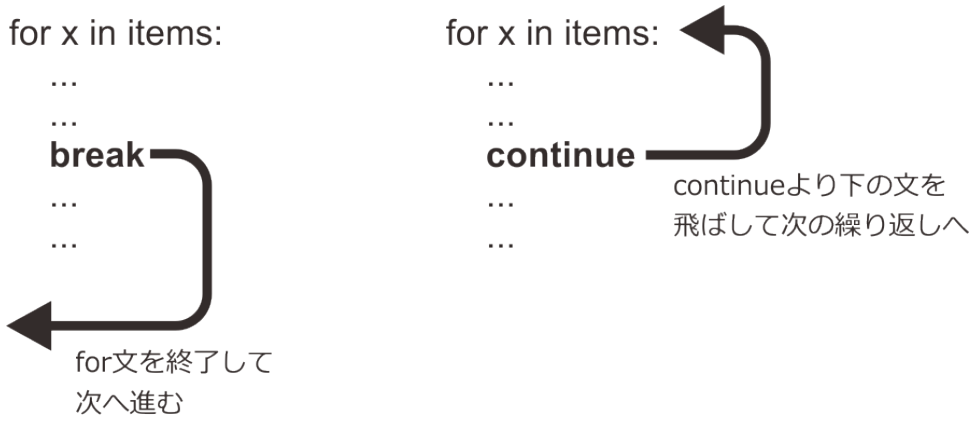


図 8.14　break と continue の違い。残りの要素に対して処理をする必要がない場合は break、必要がある場合は continue を使用します。

例題の題材は前節に引き続き「プローブがパス上にあるか判定する」処理です。前節ではパスを構成する Polygon コンポーネントのオブジェクトを並べたリストを手で入力して for 文へ渡しましたが、オブジェクトの個数が多くなるとリストが長くなって後から読み返す時に非常に読みにくいです。また、コンポーネントの名前を変更したら入力済みのコードを全部修正必要があり、間違いが生じやすいです。

Builder は、この問題を解消するのにうってつけの内部変数を持っています。Builder は各ルーチンの処理を開始する前に、「ルーチン名 +Components」という名前の変数に、ルーチン内に配置されている全てのコンポーネント (正確に言うとコンポーネントに対応するクラスのインスタンス：「8.15.1:Builder の内部変数 trialComponents に含まれるオブジェクトについて 」参照) を列挙したリストを格納します。trial ルーチンでしたら変数名は trialComponents、ready ルーチンでしたら readyComponents です。この変数を for c in trialComponents: という具合に for 文に渡せば、ルーチンに含まれるインスタンスを持つすべてのコンポーネントに対して処理を行うことができるのです。そうすればコンポーネントの数が増えてもコードの長さは変わりませんし、 **[名前]** を変更した時にいちいちコードを修正する必要もありません。

ただ、今回の「プローブがパス上にあるか判定する」という処理に変数 trialComponents を利用するには、少々面倒な問題を解決する必要があります。trialComponents には trial ルーチンに置かれたすべてのコンポーネントが列挙されているので、Mouse コンポーネントやプローブ自身、ゴール地点の円も含まれています。これらに対して contains( ) を実行しようとすると問題が生じるので、パスを構成している Polygon コンポーネントに対してだけ処理をしなければいけません。このような時に、continue は非常に有効です。具体的には以下のように contains( ) を実行する前に、現在処理しようとしているコンポーネントが path1 から path5 ではない時には continue を実行します。このようにすれば、continue 以下の文が実行されずに for 文は次の繰り返しへ進みます。

```
onPath = False
for c in trialComponents:
    if (path1 から path5 ではないことを判別する式)
```

(次のページに続く)

(前のページからの続き)

```
        continue
    if c.contains([px, py]):
        onPath = True
        break
```

残る問題は、現在 c に代入されている要素が path1 から path5 でないかをどうやって判別するかです。これには PsychoPy の刺激描画用クラスが持っている name というデータ属性を使用します。データ属性 name には対応する Builder のコンポーネントの **[名前]** に入力した文字列が格納されているので、name に'path' という文字列が含まれているかを判別すればよいでしょう。この判別にはシーケンス中にある要素が含まれるか否かを判定するときに用いた in 演算子が使用できます (第 7 章 参照)。

```
'path' in c.name
```

c.name に'path' という文字列が含まれていれば、この式は True になります。これで万全、と言いたいところなのですが、残念ながらもう一工夫必要です。データ属性 name は PsychoPy の刺激描画用クラスが持っているものであり、trial ルーチンで使われている Mouse クラスには name がないのです。ですから、c に Mouse クラスのインスタンスが代入されている状態で'path' in c.name を実行すると「name というデータ属性はありません」というエラーが出て実験が停止してしまいます。この問題に対応するためには、まず c に格納されているオブジェクトが name というデータ属性を持っていることを確認する必要があります。確認のためには hasattr( ) という関数を使います。hasattr( ) は 2 個の引数を持ち、第 1 引数に指定されたオブジェクトが第 2 引数に指定された名前のデータ属性を持っていれば True が返されます。以下のように使用すると、c に格納されているオブジェクトが name というデータ属性を持っている時に True の値を得ることができます。

```
hasattr(c, 'name')
```

この二つの条件式を組み合わせれば、c が'path' という文字列を **[名前]** に含むコンポーネント (に対応するインスタンス) であるか判別できます。どちらか一方の条件式を満たさないオブジェクトはパスを構成している Polygon コンポーネントではありませんので、

```
not hasattr(c, 'name') or not 'path' in c.name
```

が True となった時には continue を使って次のオブジェクトへ処理を進めればよいということになります (or の前後はこの順序である必要があります：「 8.15.2: 論理式の評価順序について 」参照)。

```
if not hasattr(c, 'name') or not 'path' in c.name:
    continue
```

このコードを for 文に追加したものを 図 8.15 左に示します。コードがどのように動作するのかを追った流れをその右側に示してあります。continue の働きを確認してください。

for 文にはまだまだ解説したい機能があるのですが、いくらなんでも脱線しすぎですのでそろそろ実験の作成に戻りましょう。次節では、プローブがパス上にあるかを判定した結果とプローブの座標を実験記録ファイルに出力することに取り組みます。

チェックリスト

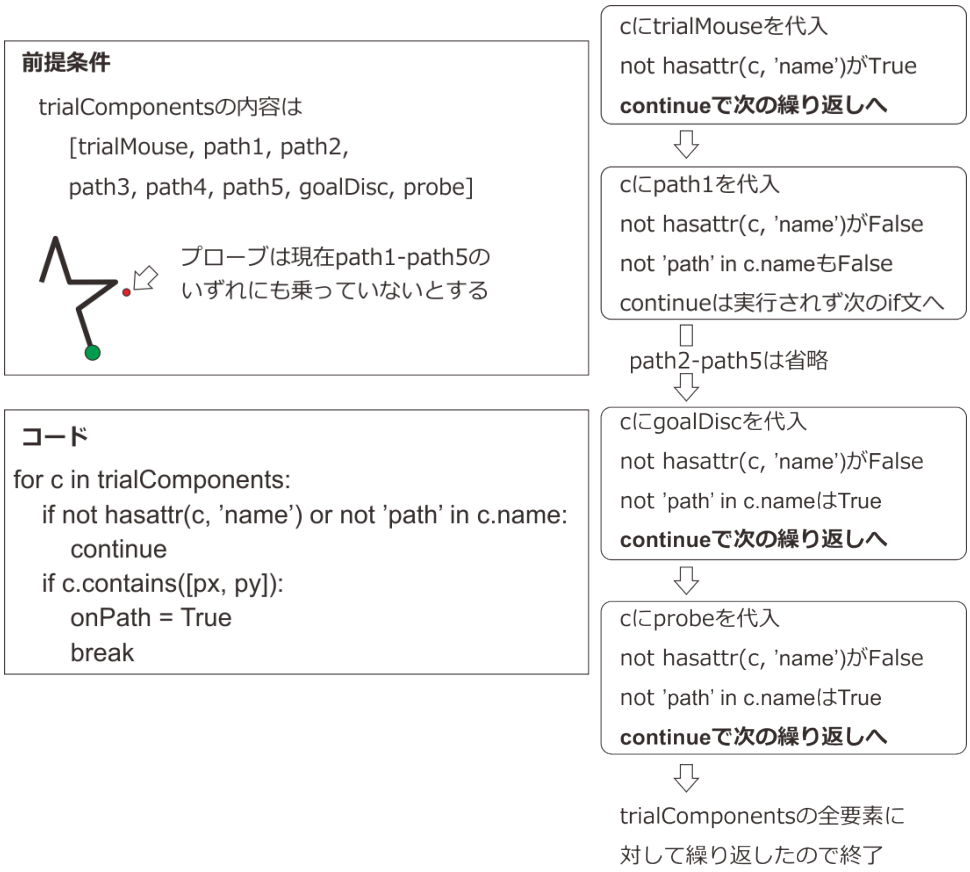


図 8.15　continue による繰り返し処理のスキップ。

- for 文で現在処理中の要素に対してこれ以上処理を続ける必要がなくなった時に、次の要素の処理へ直ちに移行するコードを書くことができる。
- ルーチン内に含まれるコンポーネントの一覧を格納した Builder の内部変数の名前を答えられる。
- オブジェクトがある名前のデータ属性 (たとえば'foo') を持っているか判別するコードを書くことができる。
- ある文字列が別の文字列の中に含まれているか (例えば'psych' が'psychophysics' に含まれるか) 判別するコードを書くことができる。

8.10 リストにデータを追加してマウスの軌跡を保存しよう

この章の実験作成も最終段階です。プローブの軌跡と、プローブがパス上にあるかを判定した結果を実験記録ファイルに出力しましょう。具体的には、Mouse コンポーネントの出力のように、実験記録ファイルに probe_x、probe_y、on_path という列を設けて、そこへそれぞれ X 座標、Y 座標、判定結果の値を並べたリストを出力したいと思います。

この章までにリストは何度も扱ってきましたが、以前の章に出てきたリストはすべて事前に長さが決まっていました。 **[位置 [x, y] $]** に入力する座標でしたら長さは 2 ですし、 **[前景色]** に RGB で色を指定するのでしたら長さは 3 です。それに対して、今回実験記録ファイルに出力しようとしているリストは事前に長さがわかりません。実験参加者が課題遂行に時間がかかればかかるほど、プローブの座標を記録したリストは長くなりま

す。課題遂行に要する時間が予測できないので、記録に必要なリストの長さも予測できないのです。このような場合に有効なのが、リストの要素を追加したり削除したりする機能です。実は今までリストと呼んできたデータは Python に標準で備わっている List というクラスのインスタンスであり、List は PsychoPy の Mouse クラスなどと同様にメソッドを持っています。 表 8.3 に List クラスの主なメソッドを示します。これらのメソッドを活用すれば問題を解決できそうです。

表 8.3　List クラスの主なメソッド

メソッド	概要
append(x)	リストの末尾に新たな要素として x を追加します。リストの長さは 1 増加します。
extend(seq)	リストの末尾に新たな要素として seq の要素を追加します。seq はシーケンス型や文字列型など、長さが定義されるデータ型である必要があります。リストの長さは seq の長さ分増加します。
insert(i, x)	リストの i 番目の要素として x を追加します。i は整数でなければいけません。
remove(x)	リストの先頭からチェックして、最初に現れた x を削除します。x がリストに含まれていない場合はエラーになります。
index(x)	リストの先頭からチェックして、最初に現れた x のインデックスを返します。x がリストに含まれていない場合はエラーになります。
count(x)	リストに x が何個含まれているかを返します。
sort( )	リストの要素を昇順に並び替えます。
reverse( )	リストの要素を現在の順番と逆順に並び替えます。

さて、じっくりと 表 8.3 を眺めると、要素を追加するメソッドとして append( ) と extend( ) の 2 種類があることがわかります。両者の違いをまとめたものが 図 8.16 です。ポイントは、append( ) は引数をそのままひとつの要素として付け加えるのに対して、extend( ) は引数の要素を付け加えます。ですから、append( ) を実行すると必ずリストの要素が 1 個増えます。それに対して extend( ) の場合は引数に含まれる要素数だけリストの要素が増えます。また、extend( ) は長さがないデータ、すなわち [ ] 演算子を付けて要素を取り出すことができないデータを引数にすることはできません。具体的には数値や真偽値 (True/False) などは extend( ) の引数にできません。今回の目的では、保存しようとしているプローブの X 座標、Y 座標は数値、判定結果は真偽値ですから、いずれも extend( ) で追加することができません。append( ) を使用しましょう。

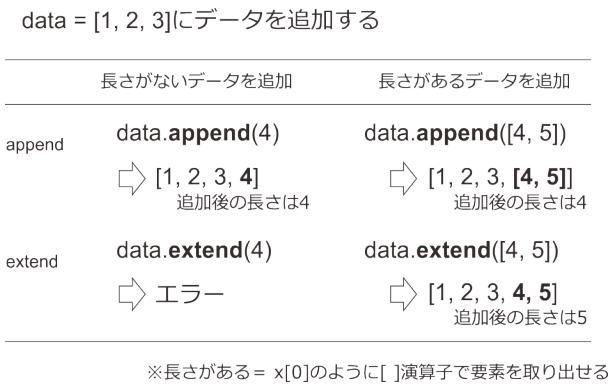


図 8.16　append( ) と extend( ) によるリストへのデータの追加。

それではコードを入力していきます。まず、exp08a.psyexp を開いて trial ルーチンの Code コンポーネントのプロパティ設定ダイアログを開いてください。 **[Routine 開始時]** に以下のコードを追加してください。

```
probeX_list = [ ]
probeY_list = [ ]
onPath_list = [ ]
```

ここでは、trial ルーチン開始時にこれから実行する試行のデータを追加していくためのリストを用意しています。=演算子の右辺に奇妙な式が出てきましたが、 図 8.10 をよく思い出してください。式の中に [ ] が出てきた場合、[の直前が変数やデータであれば、要素を取り出す [ ] です。そうでなければ、リストを作成する [ ] です。上記の式の場合、[の前は=演算子であって変数やデータではありませんので、これはリストを作成する [ ] です。[ ] の中身に何も書いてありませんから、中身が空っぽ (長さは 0) のリストが作成されます。

続いて、 **[フレーム毎]** に入力済みのコードの最後に以下のコードを追加します。

```
probeX_list.append(px)
probeY_list.append(py)
onPath_list.append(onPath)
```

ここでは新しいプローブ座標 (px, py) が得られるたびに、先ほど作成したリストにプローブの X 座標、Y 座標、判定結果を追加しています。そして最後に、 **[Routine 終了時]** でこれらのリストを実験記録ファイルに出力します。この方法については 第 7 章 ですでに解説済みですので、わからない人は 第 7 章 をしっかり復習してください。

```
trials.addData('probe_x', probeX_list)
trials.addData('probe_y', probeY_list)
trials.addData('on_path', onPath_list)
```

probeX_list、probeY_list、onPath_list の内容は、実験記録ファイルに出力さえしてしまえばもう用済みです。ですから、次の試行を開始する時には内容を初期化 (=空っぽに) してしまっても構いません。このように、現在実行中のルーチン内でだけ必要なリストは、 **[Routine 開始時]** で作成するのが定番です。一方、実験を通じてデータを蓄積して、実験終了時に何かの処理をするためのリストを作成するのであれば、 **[実験開始時]** で作成するべきです。

これで作業は完了しました。exp08a.psyexp を保存して実行し、数試行実行してみてください。適当なところで Escape キーを押して実験を終了し、Excel で trial-by-trial 記録ファイルを開いてみましょう。 図 8.17 のように probe_x、probe_y、on_Path という列が追加されていて、Python のリストのように [ ] で囲まれたカンマ区切りの値が並んでいるはずです。Excel にはこれらの値がまとめて文字列として認識されてひとつのセルに収められています。ちょっと面倒ですが、これらのセルの文字列を別の場所にコピーして「データの区切り位置」でカンマを区切り文字に指定すれば、ひとつひとつの値が 1 個のセルを占めるように変換できます。この状態まで変換すれば、後は 図 8.18 のように Excel の機能でグラフをプロットしたり、さまざまな関数を利用して分析したりできます。

これで完成！と言いたいところですが、この節で解説した方式だと一般的な 60fps のモニターを使用している場合 1 秒ごとに 60 件ものデータが追加されてしまうので、参加者がじっくりと課題に取り組む人の場合、膨大な量のデータが出力されてしまいます。軌跡をどの程度詳細に分析するかに寄って適切なサンプリング頻度

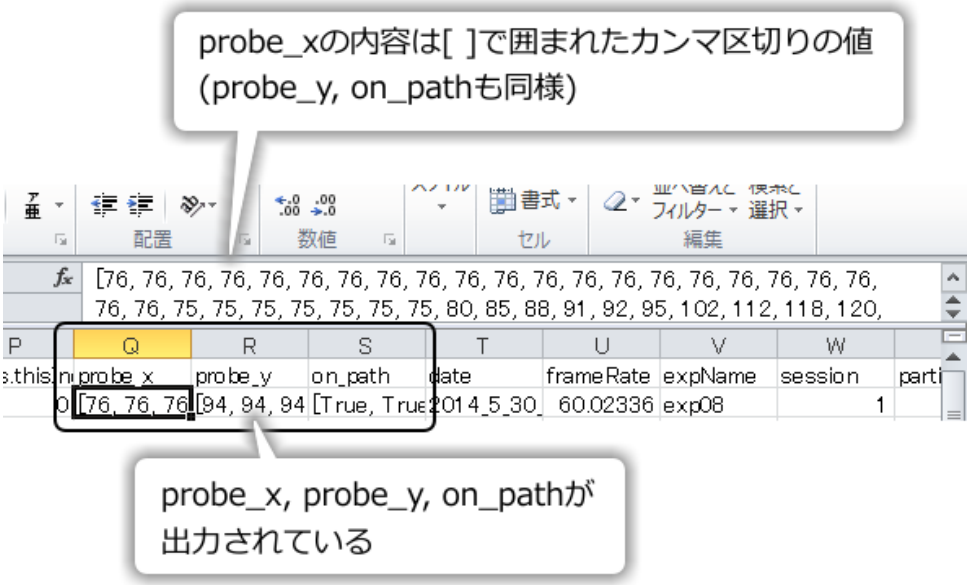


図 8.17　trial-by-trial 記録ファイルにおけるプローブ座標と判定結果の出力。Python のリストと同様に、[ ] で囲まれたカンマ区切りの値として出力されています。

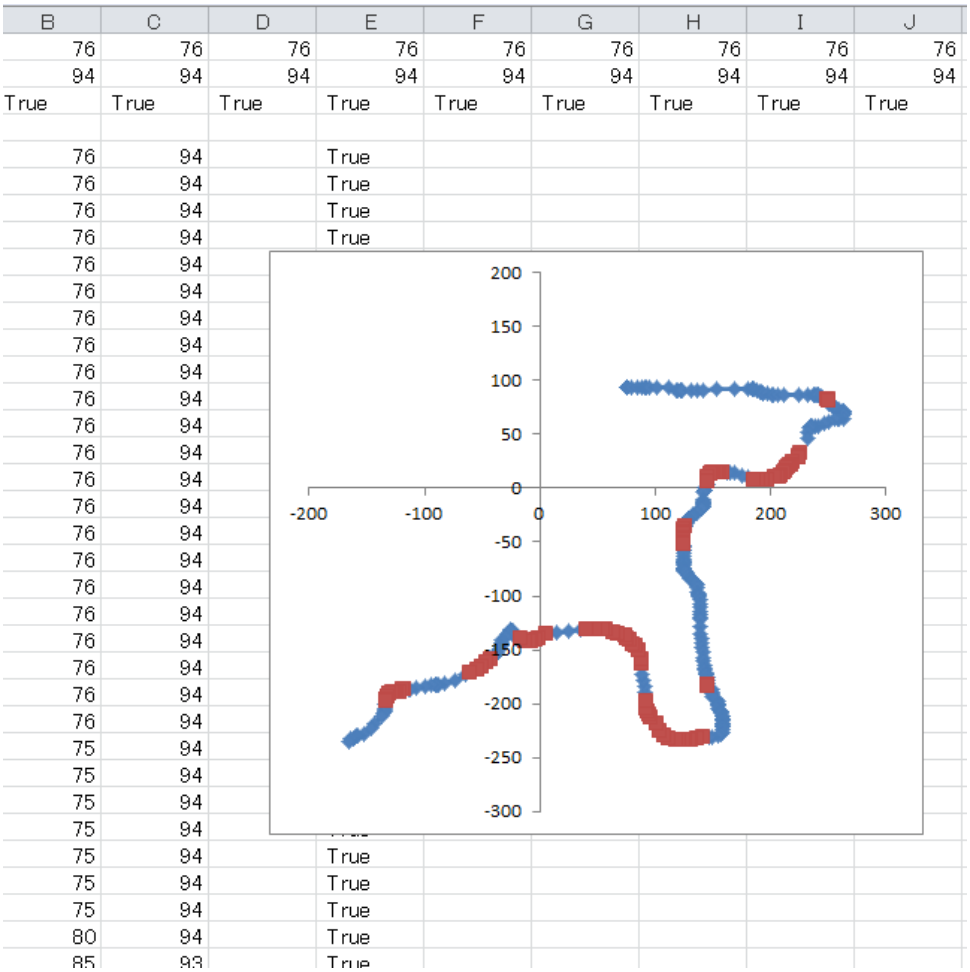


図 8.18　probe_x、probe_y、on_Path を Excel の「データの区切り位置」機能を用いて復元し、軌跡をグラフとしてプロットしたもの。軌跡の青い部分はパス上にプローブの中心が乗っていたことを、赤い部分は乗っていなかったことを示しています。

は異なりますが、おおよその軌跡が把握できればよいだけでしたら、1 秒間に 10 件程度でも充分でしょう。次節では、このようなデータの間引きを行うコードを考えます。

チェックリスト

- リストにデータを追加するメソッドである append( ) と extend( ) の違いを説明できる。
- 中身が空のリストを作成することができる。
- ルーチン内でのみ必要なデータを蓄積するリストを作成するコードはどの時点で初期化すべきか判断できる。

8.11 軌跡データを間引きしよう

軌跡データの間引きといってもいろいろな方法があるのでしょうが、もっとも単純な方法はサンプルを 1 つおきに保存すると言った具合に、一定間隔でサンプルを保存してその間のサンプルは捨ててしまうといったものでしょう。このように「順番に並んでいる (取得される) データを一定間隔で取り出して処理する」という作業には定番のコードがありますので、ぜひ覚えておきましょう。

exp08a.psyexp を exp08b.psyexp という別名で保存して、そちらを使って作業しましょう。exp08b.psyexp を開いて、trial ルーチンに配置してある Code コンポーネントの **[フレーム毎]** の最後を見てください。プローブの座標とプローブがパス上にあるかの判定結果のデータを append している 3 行のコードがあります。frameN % 6==0 という式が True になるときだけこの 3 行のコードを実行するように、以下のように if 文を書き足します。これだけの書き換えで、で 6 件につき 1 件のペースでデータを間引いて記録するようになりました。

```
if frameN % 6 == 0:
    probeX_list.append(px)
    probeY_list.append(py)
    onPath_list.append(onPath)
```

さて、なぜこれだけでデータを 6 件に 1 件に間引くことができるのでしょうか。ポイントは % 演算子にあります。第 5 章 で Python の算術演算子を紹介しましたが、その中に % 演算子が含まれていたのを覚えていますでしょうか。% 演算子は、x % y の形で使用して、x を y で割った時の余りが得られます。余りの事を「剰余」と呼ぶことから、% 演算子のことを「剰余演算子」と呼びます。frameN は 第 5 章 で紹介した Builder の内部変数で、ルーチンの実行が開始してから描画したフレーム数が格納されています。ということは、frameN % 6 という値は 6 フレーム毎に 0 になります (図 8.19)。この性質を利用して、frameN % 6==0 が True となる時だけに append を行えば、6 件につき 1 件のペースでデータを間引いて記録できるのです。

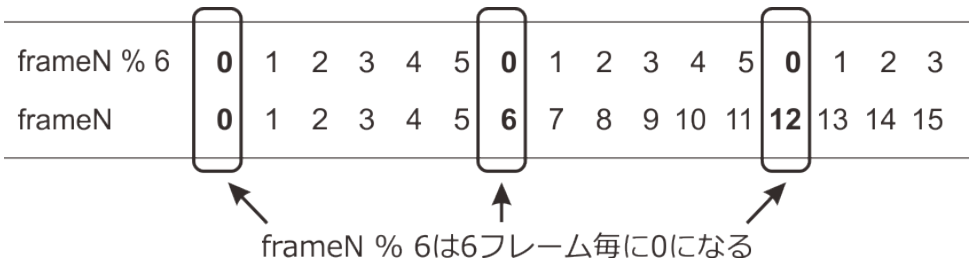


図 8.19　剰余演算子を利用したデータの間引き。

% 演算子のもう一つの重要な応用例を 図 8.20 に示します。この例では、数値がカレンダーのように左上から右下に向かって並べられています。このように、升目状に数値が並んでいるデータ構造を 2 次元配列と呼びます。2 次元配列のデータはさまざまなプログラミングで使用されますが、その際に「m 行 n 列の位置にあるデータは先頭から数えて何番目か？」とか、逆に「先頭から数えて i 番目のデータは何行目、何列目にあるか？」といった計算が求められることがよくあります。Python のインデックスが 0 から数え始めることに注意すると、m 行 n 列の位置にあるデータは m ×列数 +n であることがわかります。インデックス i が与えられた時の行数は「i を列数で割って小数点以下を切り捨てたもの」、すなわち int(i / 列数) で計算できることがわかります。そして、列数を計算する時が % 演算子の出番です。i % 列数を計算すれば、インデックス i の要素が何列目にあるかわかります。この章の実験では使用しないテクニックですが、非常に有効ですので覚えておいてください。

図 8.20　2 次元配列を 1 次元に展開した時の要素のインデックスと行、列の相互変換。インデックスは左上から右方向へ順番に割り当てて、右端に達したら次の行の左端へ移るものとします。

% 演算子の話はこのくらいにしておいて、exp08b.psyexp の作業に戻りましょう。もう先ほどの if 文を入力しただけでこの節の目的はもう達成できているのですが、このまま実行しても間引きの効果が大変わかりづらいです。そこで、プローブ座標などを保存したフレーム番号を frameN という列名で実験記録ファイルに出力するように改造し、ついでに実験情報ダイアログに Interval という項目を設けて何フレーム毎に保存するかを指定できるようにしましょう。これはすでに解説済みのテクニックの復習ですので、自信がある人はノーヒントで取り組んでみてください。うまく動作したら「チェックリスト」まで飛ばしていただいて構いません。

では、作業内容を順番に説明します。まず、フレーム番号を保存するためのリストが必要なので、trial ルーチンの Code コンポーネントの **[Routine 開始時]** に以下のコードを追加しましょう。

```
frameN_list = [ ]
```

続いて、 **[フレーム毎]** の最後でリストにデータを append しているところの最後に frameN_list へ frameN を append するコードを追加しましょう。また、剰余演算の部分で実験情報ダイアログの Interval という項目から値を取得するようにしておきましょう。実験情報ダイアログから取得した値は文字列なので、数値に変換す

るために int( ) を使用しないといけない点に注意してください。

```
if frameN % int(expInfo['Interval']) == 0:
    probeX_list.append(px)
    probeY_list.append(py)
    onPath_list.append(onPath)
    frameN_list.append(frameN)
```

以上のコードを入力したら、実験設定ダイアログを開いて、実験情報ダイアログに Interval という項目を追加しておきましょう。そして最後に、trial ルーチンの Code コンポーネントに戻って **[Routine 終了時]** の最後に frameN_list を実験記録ファイルに出力するコードを付け足しておきましょう。

```
trials.addData('frameN', frameN_list)
```

以上で実験は完成です。exp08b.psyexp を保存して実行してみましょう。Interval の値をいろいろと変更して、実験記録ファイルの frameN の値の間隔が変化することを確認してください。

チェックリスト

- % 演算子を用いて N フレーム (N=2,3,4,⋯) に 1 回の頻度で処理を実行させることができる。
- 2 次元配列の要素のインデックスから、その要素が何行目、何列目にあるかを計算することができる。

8.12 ゴール地点でクリックして終了するようにしてみよう

この章も長くなりましたし、そろそろ練習問題にしたいのですが、その前にもう一つだけ作業をします。本当はこの作業を練習問題にしたいのですが、今後皆さんが他人が書いたコードを読まないといけなくなった時に知っておくと役立つポイントがありますので解説しておきます。

この節で行う作業は、「プローブがゴールに到着したときに自動的に試行を終了するのではなく、ゴールに到着したうえでマウスをの左ボタンをクリックしないといけないようにする」というものです。ここまでの作業では Mouse クラスの getPressed( ) の出番がなかったので、このメソッドを使う練習をしておこうというわけです。

ここまでの作業で作成した exp08a.psyexp の trial ルーチンの終了判定は、以下のようなコードで行われています。

```
if goalDisc.contains([px, py]):
    continueRoutine = False
```

この if 文の条件に「マウスの左ボタンが押されている」という条件を付けくわえれば、目的は達成できます。両方の条件を満たさないといけないので、and 演算子を使って以下のように書けるはずです。

```
if startDisc.contains([px, py]) and マウスの左ボタンが押されている:
    continueRoutine = False
```

表 8.2 の getPressed( ) メソッドを利用すれば、ボタンの状態を保持したリストが得られます。これを buttonStatus という変数に格納しておきましょう。このリストの最初の要素が左ボタンの状態に対応しているのですから、以下のように書けば左ボタンが押されていることを判定できます。

```
buttonStatus = mouseReady.getPressed( )
if startDisc.contains([px, py]) and buttonStatus[0]==1:
    continueRoutine = False
```

これで全く問題ないのですが、Python に慣れるためにちょっと頭の体操をしましょう。getPressed( ) の戻り値はリストです。リストから要素を取り出すには、リストの後ろに [ ] 演算子を添えればよいのでした。ですから、上記のコードにおける buttonStatus という変数は必要ではなくて、以下のように書くことができます。

```
if startDisc.contains([px, py]) and mouseReady.getPressed( )[0] ==1:
    continueRoutine = False
```

Python 以外のプログラミング言語に慣れている方の中には、関数を呼び出す ( ) の後ろに [ ] 演算子があるのを見て奇妙に思う方もおられるかも知れません。しかし、これは var[0] がリストの時に var[0][1] という具合に [ ] 演算子を連続して書くことができたのと同じことです。このような書き方は web 上のさまざまな Python のサンプルコードでしばしば見かけますので、しっかりと理解しておいてください。

この節で取り上げた「ある座標が刺激に含まれていて、かつマウスがクリックされている」という状態を判定するテクニックは、Builder で作った実験のスクリーン上にボタンを表示してマウスでクリックして選択させるユーザーインターフェースを実装したいときなどに有効なので、ぜひ覚えておきたいところです。この用途には 表 8.2 にも挙げた isPressedIn() が便利ですが、「カーソルが重なっただけで色を変化させる」といった凝った動作をさせる必要がある場合などには、contains( ) を使う方が柔軟に対応できます。isPressedIn(shape, buttons) の例については「 10.5: マウスで一時停止やスキップを行えるようにしよう (上級) 」をご覧ください。

チェックリスト

- 関数やメソッドの戻り値に直接 [ ] 演算子を適用した式を理解できる。
- Polygon コンポーネントで描画した多角形にマウスカーソルを重ねてクリックするとルーチンの終了や反応の記録などの処理を行うコードを書くことができる。

8.13 練習問題 1：反転方向切り替え機能とフィードバック機能を追加しよう

以上でこの章の解説は終わりです。この章の内容まで理解できていれば、Builder で相当複雑な実験を作成することができるはずです。この章の仕上げとして、以下の練習問題に取り組んでください。ここまで理解できた人であればノーヒントでできるものと期待します。

- 実験情報ダイアログに Direction という項目を追加してください。Direction の値に応じて以下のようにプローブの動き方を変更してください。
  - Direction が UD という文字列であれば exp08a.psyexp と全く同一の動作。

  - Direction が LR という文字列であれば、プローブがマウスカーソルの動きと左右反対に動く。
  - Direction が UD でも LR でもなければ、プローブとマウスの動きが一致。
- プローブの中心座標がパス上にある時にはプローブの **[塗りつぶしの色]** が赤色、パス上にない時には **[塗りつぶしの色]** が黒色になるようにしてください。

8.14 練習問題 2: ミューラー・リヤー錯視実験をマウスで反応できるようにしよう

第 7 章 ではキーボードを使ってミューラー・リヤー錯視実験のプローブ長の調節を行いました。実際に実験を作成して実行してみた方は、プローブ長を大きく変更したいときにキーを何回も叩かないといけないことに不満を感じたのではないかと思います。Microsoft Word 等のソフトウェアでカーソルを動かすときは、カーソルキーをカチカチと連打しなくても、押しっぱなしにしていればスーッとカーソルが動きますが、第 7 章 の実験でも同じような感じに操作できるときっと快適にプローブ長を調整できるでしょう。このように、キーボードのキーを押しっぱなしにすると何度もキーを叩いたのと同じように文字入力を繰り返す機能をキーリピートと呼びますが、残念ながら現時点 (バージョン 3.0.5) では Builder からキーリピートを利用することはできません。そこで、本章で学んだ Mouse コンポーネントを利用してプローブ長を変更することにします。

第 7 章 で完成させたミューラー・リヤー錯視実験の psyexp ファイル (exp07b.psyexp) と条件ファイル (exp07cnd.xlsx) を使用します。exp07b.psyexp はプローブ長が 0.05 から 0.35 以内に収まるようにコードを追加した部分まで作業を終えているとします。

準備として、exp07b.psyexp を開いて以下の作業をしてください。

- trial ルーチン
  - Mouse コンポーネントをひとつ配置して以下の通り設定する。
    - **[名前]** を mouse にする。
    - **[終了]** を空白にする。
    - **[ボタン押しで Routine を終了]** がチェックされていることを確認する。
    - **[マウスの状態を保存]** を「なし」にする。
  - 配置済みの Keyboard コンポーネントを削除する。

第 7 章 、第 8 章 の知識に加えて、psychopy.event.Mouse の getRel() および getWheelRel() というメソッドを使用してよいものとします (表 8.4)。

表 8.4　マウスカーソルおよびホイールの移動量を取得する psychopy.event.Mouse のメソッド

メソッド	概要
getRel()	最後に getPos() または getRel() を実行したときのマウスカーソル位置を基準として、現在のマウスカーソルの相対位置を表すリストを返します。リストの最初の要素は X 座標、次の要素は Y 座標を示しています。
getWheelRel()	最後に getWheelRel() を実行したときのマウスホイール位置を基準として、マウスホイールの移動量を表すリストを返します。多くの場合、リストは 2 次元で 2 番目の要素に移動量が格納されていますが、使用しているマウスのデバイスドライバに依存します。

課題その 1: マウスカーソルの移動でプローブ長を変化させる。

- マウスカーソルを描画しない。
- マウスカーソルを右に移動するとプローブが長くなり、左に移動すると短くなるようにする。
- マウスのボタンをクリックして反応を確定する。

課題その 2: マウスのホイールでプローブ長を変化させる。

- マウスカーソルを描画しない。
- マウスのホイールを回転させるとプローブ長が変化する。回転方向と伸び縮みの対応関係は指定しないので各自で決めてよい。
- マウスのボタンをクリックして反応を確定する。

課題その 3: マウスのドラッグでプローブ長を変化させる。

- マウスカーソルを描画しない。
- マウスの左ボタンをプレスしたままドラッグするとプローブ長が変化する。
- マウスの右ボタンをクリックして反応を確定する。

以下、ヒントを書くのでわからない場合は参考にしてください。

課題その 1 のヒント　Code コンポーネントで if 文を用いて押されたキーを判定して probeLen の値を変更していた部分を削除し、mouse.getRel() を実行しましょう。getRel() の戻り値の X 成分を probeLen に加えれば課題は達成できるはずです。

課題その 2 のヒント　課題その 1 で getRel() を使用した代わりに getWheelRel() を使用します。getWheelRel() の戻り値の何番目の要素がホイールの回転量に対応しているかは 表 8.4 に書いた通りドライバに依存するので、まずは (最初の要素を 0 番目として)1 番目の要素を使用してみて、動作しないようなら他の要素を試してみましょう。

課題その 3 のヒント　マウスの左ボタンをクリックしても反応を確定してはいけませんので、**[ボタン押しで Routine を終了]** のチェックは外す必要があります。getPressed() を実行して、左ボタンが押されてい

る時のみ probeLen の長さを更新するとよいでしょう。右ボタンが押されている場合は continueRoutine を False にします。実験参加者が右クリックする際にマウスを動かしてしまう可能性を考えると、右ボタンが押されている場合は probeLen の更新が行われないようにする必要があります。

8.15 この章のトピックス

8.15.1 Builder の内部変数 trialComponents に含まれるオブジェクトについて

本文中で述べたように、Builder では各ルーチンの処理を開始する前に、「ルーチン名 +Components」という名前の変数 (以下 trialComponents) が作成されます。ここには、ルーチン実行中に毎フレーム処理しなければいけないオブジェクトがリスト型のデータとして列挙されています。オブジェクトはそれぞれ対応するコンポーネントの機能を実現するためのクラスインスタンスです。「6.9.4:Builder のコンポーネントと PsychoPy のクラスの対応 」で述べたとおり、コンポーネントによって Grating のように一対一にクラスが対応している物と、Polygon のように対応するクラスが変化するものがあります。また、Code コンポーネントは直接 Python のコードを Builder に埋め込むという性質上、他のコンポーネントとは異なる方法で管理されており、trialComponents に列挙されません。

8.15.2 論理式の評価順序について

本文中に出てきた「c が'name' というデータ属性を持っていないか、path という文字列をデータ属性 name に含んでいない」という条件を検査する式について補足しておきます。この式は以下の通りで、「or の前後はこの順序である必要がある」と本文中で述べました。

```
not hasattr(c, 'name') or not 'path' in c.name
```

なぜこの順序でなければいけないのでしょうか。その理由は、Python が論理式を評価する順番にあります。Python では、論理演算子は算術演算子や比較演算子より優先順位が低く、論理演算子の中では not、and、or の順に評価されます。優先順序に差がない部分は左側から評価されます。上記の順序であれば、左から評価されるので not hasattr(c, 'name') が評価されます。もし not 'path' in c.name が or の左に書いてあれば、いきなり c のデータ属性 name の値を参照するので c がデータ属性 name を持っていなければエラーで停止してしまいます。

not hasattr(c, 'name') を左に書いても結局データ属性 name が無かったら not 'path' in c.name でエラーになるんじゃないの？と思われるかもしれませんが、Python には「論理式の評価途中で値が確定してしまったら、残りの部分の評価を行わない」という規則があります。どういう事かと言うと、A or B という式で A が True であれば、B が True であろうと False であろうと A or B の結果は True です。このような時に、Python はわざわざ時間をかけて B の評価を行わず、A or B の結果は True と判断します。

今回の例に当てはめて考えましょう。not hasattr(c, 'name') が True だったら、この時点で条件式は True になるので、not 'path' in c.name は評価しません。not hasattr(c, 'name') が False の場合は、式の真偽が確定しませんので not 'path' in c.name を評価します。この時、not hasattr(c, 'name') が False だったのですから、c は必ずデータ属性 name を持っています。従って、not 'path' in c.name で「データ属性 name が無い」というエラーが生じることはありません。

同様の理由で、A and B という式を評価する時に、A が False であれば B の評価は行われません。

なお、この if 文がわかりにくいという方は、以下のように二つの if 文に分けて書くこともできます。覚えやすい方を覚えていただければと思います。

```
if not hasattr(c, 'name')
    continue
if not 'path' in c.name
    continue
```

8.15.3 リストとタプル

タプルについては、第 5 章 で「シーケンス型」というデータ型が初登場したときに名前だけは紹介していたのですが、その性質については全く解説しませんでした。それは 第 5 章 の時点ではリストとタプルの違いを説明することが難しかったからなのですが、本章で [ ] 演算子による要素の指定を学んだおかげでようやく準備が整いました。

タプルは data = (1, -7, 'psychoplogy') といった具合に、リストと同様に要素をカンマで並べて作成します。リストとの違いは、リストでは [ ] で要素を囲んだのに対してタプルでは ( ) で要素を囲む点です。作成したタプルは、リストと同様に [ ] 演算子を適用することによって要素にアクセスすることができます。先の例で data[2] とすれば'psychology' が得られますし、data[-3] とすれば 1 が得られます。機能的な意味でのリストとタプルの最大の違いは、リストは要素を追加したり変更したりできるのに対して、タプルはそのような変更ができないという点にあります。例えば、リストであれば

```
data = [1, -7, 'psychology']
data[1] = 5
```

とすれば、data は [1, 5, 'psychology'] となります。一方、タプルに対して同様の処理をしようとするとエラーとなってプログラムの実行が停止します。

```
data = (1, -7, 'psychology')
data[1] = 5     #エラーとなる
```

また、リストにおける append( ) や extend( ) といったメソッドもタプルには存在しません。

どう考えてもタプルは不便なだけのような気がするのですが、なぜタプルなどというデータ型が用意されているのでしょうか。それは、タプルの方がリストよりも効率的かつ高速に処理できるからです。なぜそうなるのかを説明するのは難しいのですが、製本されたノートとルーズリーフの違いのようなものを思うと少しイメージしやすいかもしれません。ルーズリーフは途中に新しいページを挿入したり、順番を入れ替えたりすることが容易にできますが、本当に将来挿入や入れ替えをする必要があるのなら、複数件のメモを一枚のルーズリーフに書くことはできません。ほんの数行だけのメモだけで一枚のルーズリーフを使ってしまい、大変効率が悪いです。後で挿入や入れ替えをする必要がないのなら、ルーズリーフを使わなくても通常のノートに隙間なくメモを書き込む方がいいでしょう。恐らく使用するページ数も少なく済み、ルーズリーフを用意するより安価でかさばらないはずです。これはあくまで例え話に過ぎませんが、変更できないようにすることで得られるメリットがあるから Python にはタプルというデータ型が用意されていると理解しておいてください。

古いバージョンの Python では「 12.4: 文字列の format() メソッドを使って経過時間の表示を整えよう 」で解説する「文字列への数値の埋め込み」の際にタプルを使うことが必須だったのでタプルのことをしっかり理解しておく必要があったのですが、現在では文字列オブジェクトのメソッドを使う方法が一般的なので、タプルはあまり出番がなくなってしまいました。それでも、タプルとリストの違いを理解していないと予想外のところでエラーとなる可能性がありますので、やはり覚えておきたいところです

第 9 章

グラフィカルインターフェースを活用しよう

本章では、本書の第 2 版執筆以降に追加された「Button コンポーネント」、「クリッカブルオブジェクト」、「Slider コンポーネント」、「Form コンポーネント」について解説します。全ての追加機能を使う実験を作るのが難しそうでしたので、ひとつの実験を作り上げるのではなく各機能を体験するデモを作ることを目標とします。

9.1 Button コンポーネントと Variable コンポーネントで刺激を操作してみよう

PsychoPy 2021.1.0 から、マウスでクリックして反応を測定したり刺激を操作したりできるボタンを画面上に配置する Button コンポーネントが登場しました。 第 8 章 で解説したテクニックを使って同様のことは実現できますが、こちらの方が使用するコンポーネントが少ない分実験の作成が少し楽になります。本節では Variable コンポーネントの紹介も兼ねて、簡単なデモを作ってみましょう。

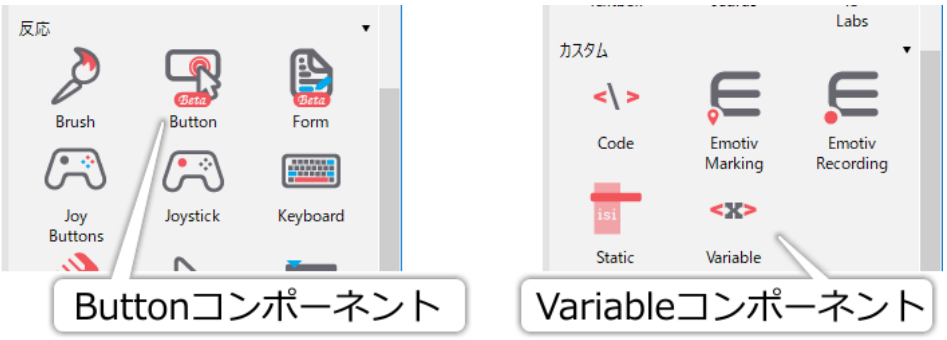


図 9.1　Button コンポーネントと Variable コンポーネントのアイコン。

Button コンポーネントは「反応」、Variable コンポーネントは「カスタム」カテゴリの中にあります (図 9.1)。Button コンポーネントの「基本」タブにはお馴染みの **[名前]** 、 **[開始]** 、 **[終了]** 、 **[Routine を終了]** に続いて、 表 9.1 に示すプロパティがあります。**[ボタンのテキスト]** は解説の必要はないかと思いますが、 **[コールバック関数 $]** という用語は聞いたことがない方が多いでしょう。コールバック関数とは「関数から呼び出すために引数として渡しておく関数」のことで、ここでは Button コンポーネントに「もしクリックされたらそ

の時はこれを実行してほしい」と渡しておくコードを意味しています。Code コンポーネントで記入するコードは「Routine 開始時」や「フレーム毎」といったスケジュールに沿って実行されますが、コールバック関数は Button コンポーネントが適切なタイミングで実行してくれるわけです (図 9.2)。ちょっとプログラミングに詳しい方向けに書いておくと（意味が分からない人は読み飛ばしていただいて結構です)、「コールバック関数」といってもここでは関数の宣言を書く必要はありません。関数内で実行する処理だけを書けばよいです。

表 9.1　Button コンポーネントの主要なプロパティ。

[ボタンのテキスト]	ボタン内に表示するテキストです。
[コールバック関数 $]	ボタンをクリックしたときに実行させたいコードを記入します。「ボタンをクリックしたことを記録してルーチンを終了する」だけなら何も記入する必要はありません。
[クリック毎に 1 回実行]	**[コールバック関数 $]** に記入したコードをクリックされたときに 1 度だけ実行するか、押されたボタンが離されるまでフレーム毎に実行するかを指定します。 **[Routine を終了]** がチェックされているとこの項目自体無効になります (クリックした途端に終了するので意味がない)。

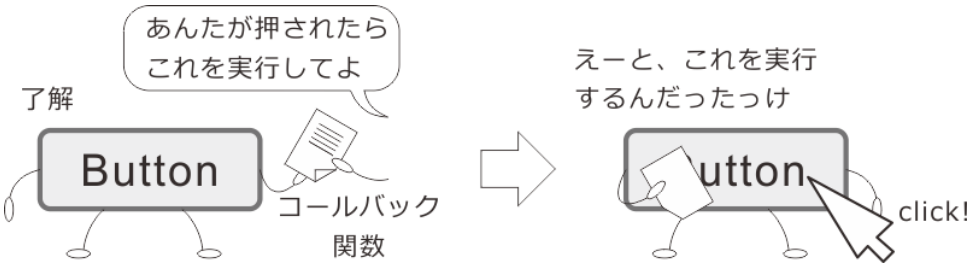


図 9.2　コールバック関数の仕組み。

[クリック毎に 1 回実行] は、ボタン上にマウスカーソルを移動させてボタンをやや長めに押した時に意味を持ってきます。この項目がチェックされていないと、Builder はフレーム毎に「ボタンが押されているか」をチェックしてコールバック関数を実行します。チェックされていれば、一度ボタンを押した後、いったん指を離して押し直すまでコールバック関数が実行されません。これは恐らく実際に確認した方がわかりやすいと思いますので、デモを作成してから改めて解説することにしましょう。

Button コンポーネントには「基本」タブの他に「レイアウト」、「外観」、「書式」、「データ」、「テスト」のタブがあります。「レイアウト」、「外観」、「書式」タブにはボタンの枠とテキストの色や大きさ、フォント等を指定するプロパティが含まれています。Text コンポーネントや Polygon コンポーネントと大部分が共通していますので、解説の必要はないでしょう。唯一気をつけないといけないのが **[フォント]** で、Text コンポーネントでは日本語の文字が正しく表示できるにも関わらず、Button コンポーネントで同じ設定にしても日本語の文字が表示されない場合があります。内部で日本語の字体の取得に失敗しているのが原因なので、日本語の字体を含むフォントを明示的に **[フォント]** で指定する必要があります。例えば Windows では Open Sans や Arial などではなく Meiryo や Yu Gothic などを指定してください。

「データ」タブには他と共通のプロパティに加えて **[クリックを記録]** というものがあり、「全てのクリック」、「最初のクリック」、「最後のクリック」、「なし」のいずれかを選ぶことができます。Keyboard コンポーネントの **[記録]** に対応するものだと考えてください。 **[Routine を終了]** をチェックしている場合は「全てのクリッ

ク」、「最初のクリック」、「最後のクリック」の動画が実質同じになる点や、単に次のルーチンへ進むために参加者に押させたいだけの場合は「なし」を選ぶと余計な出力を抑えられる点も同じです。

続いて Variable コンポーネントですが、これは 第 7 章 で学んだ「プローブの長さを保持する独自の変数を用意して記録ファイルに値を出力する」といった動作を Code コンポーネントを使わずに実現するコンポーネントです。 第 7 章 の方法と Variable コンポーネントのそれぞれに長所、短所がありますので、本章のデモを見て好みの方を使うと良いと思います。

Variable コンポーネントの「基本」タブには毎度おなじみの **[名前]** 、 **[開始]** 、 **[終了]** があります。 **[名前]** はそのまま変数名として解釈されます。続いて **[実験開始時の値]** 、 **[Routine 開始時の値]** 、 **[フレーム開始時の値]** という項目がありますが、それぞれのタイミングで変数の値を設定するために使います。設定する必要がない場合は空欄で構いません。例えば 第 7 章 で probeLen という独自の変数を Code コンポネントで使用してルーチンの開始時に probeLen = initProbeLen を実行しましたが、同じことを Variable コンポーネントで実現するなら、以下のように設定すればよいわけです。**[実験開始時の値]** や **[フレーム開始時の値]** は空欄のままにしてください。

- **[名前]** を probeLen にする。
- **[Routine 開始時の値]** を initProbeLen にする。

続いて「データ」タブですが、ここではどの時点の値を記録ファイルに出力するかを指定します。 第 7 章 の例なら、trial ルーチンが終わる毎に値を保存する必要があるので **[Routine 終了時の値を保存]** をチェックすればよいでしょう。これで Code コンポーネントを使わなくても記録ファイルに値が出力されます。

Variable コンポーネントの長所は、addData( ) などのメソッドを覚えておかなくても使えること、そしてルーチンペインを見たらどのような独自変数を使っているのか一目でわかることの 2 点です。Code コンポーネントで変数を定義した場合、どのような変数を独自に定義したかを把握するにはプロパティの内容を確認しなければなりません。実験を作成した直後は使用した変数のことなどはよく覚えているものですが、数か月後に論文執筆や次の実験の作成のために久しぶりに実験ファイルを開くと、自分で作ったものなのに「あれ、これはどうだったかな」とわからなくなることは珍しくありません。「わかりやすい」というのはとても大切です。

一方、Variable コンポーネントの短所は、凝った処理を行おうとすると結局 Code コンポーネントが必要となりがちなことと、複数のルーチンにまたがって使用する変数で問題が生じる場合があることの 2 点が挙げられるでしょう。ひとつめの問題については、Variable コンポーネントを使って変数の存在をアピールしつつ Code コンポーネントを併用して処理を行うということも可能です。しかし、その場合はルーチン内で Variable コンポーネントと Code コンポーネントの順番に注意する必要があります（処理内容によってはひとつの Code コンポーネントでは不可能）。ふたつめの問題は、例えば trial というルーチンに response という名前の Variable コンポーネントを置いたら、practice というルーチンに同じ response という名前の Variable コンポーネントを置くことはできないということです。Builder の実験においてコンポーネントの名前はグローバルなものなので、どちらか一方のルーチンに配置しておけばもう一方でもその変数にアクセスできますが、 **[Routine 開始時の値]** などのプロパティを使って値を設定したり、 **[Routine 終了時の値を保存]** を使って値を保存したりできるのはコンポーネントが置かれているルーチンのみです。

前置きが長くなりました。コンポーネントの解説はこのくらいにしておいて、デモを作成してみましょう。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は、実験設定ダイアログを開いて **[単位]** を height にしておく。

- trial ルーチン
  - Variable コンポーネントを 1 つ配置し、以下の通り設定する。
    - **[名前]** に angle と入力する。
    - **[Routine 開始時の値]** に 0 と入力する。
    - 「データ」タブの **[Routine 終了時の値を保存]** をチェックして、他のチェックをすべて解除する。
  - Button コンポーネントを 1 つ配置し、以下の通り設定する。
    - **[名前]** に button_end と入力する。
    - **[終了]** を空白にする。
    - **[ボタンのテキスト]** に end と入力する。
    - 「レイアウト」タブの **[サイズ [w,h] $]** を (0.15,0.05) に、**[位置 [x,y] $]** を (0, -0.45) にする
    - 「書式」タブの **[文字の高さ $]** を 0.02 にする。
    - 「データ」タブの **[クリックを記録]** が「全てのクリック」になっていることを確認する（初期値でそうなっているはず）。
    - 「データ」タブの **[開始・終了時刻を保存]** のチェックを外す。
  - button_end をコピーして button_step, button_cnt の **[名前]** で貼り付けて、以下の通り設定する。
    - button_step の **[ボタンのテキスト]** に step 、 **[コールバック関数 $]** に angle+=5 と設定する。 **[クリック毎に 1 回実行]** がチェックされていることを確認する。
    - button_step の「レイアウト」タブの **[位置 [x,y] $]** を (-0.4, -0.4) にする。
    - button_cnt の **[ボタンのテキスト]** に continuous 、 **[コールバック関数 $]** に angle+=5 と設定する。 **[クリック毎に 1 回実行]** のチェックを外す。
    - button_cnt の「レイアウト」タブの **[位置 [x,y] $]** を (0.4, -0.4) にする。
  - Polygon コンポーネントを 1 つ配置し、以下の通り設定する。
    - **[終了]** を空白にする。
    - **[回転角度]** に angle と入力し、「フレーム毎に更新」にする。
    - 「データ」タブの **[開始・終了時刻を保存]** のチェックを外す。

以上でデモは完成です。鋭い人は「マウスを使うデモなのに Mouse コンポネントがない…？」と疑問を抱いているのではないかと思いますが、 **Button コンポーネントを使用すると自動的にマウスが準備されます。** ですから Mouse コンポーネントを置く必要はありません。 第 8 章 のようにマウスカーソルの座標を用いた処理も並行しておこなうのなら、Mouse コンポーネントを置く必要があります。自分で置いた Mouse コンポーネントと Button コンポーネントが自動的に準備するマウスは互いに影響を与えませんので、両方のコンポーネントを置いても問題は生じません。

デモを実行すると 図 9.3 のように中央に大きな三角形が描かれ、下の方に Step、end、Continuous の 3 つのボタンが表示されます。Step、Continuous のボタンはいずれも同じコールバック関数 angle=+=5 が設定されていて、違いは（外見上の違いを除けば）**[クリック毎に 1 回実行]** のチェックの有無だけです。マウスを操作して、それぞれのボタンを少し長めに押してみましょう。Step のボタンはマウスのボタンを押した瞬間に三角形が少し時計回りに回転して、その後マウスのボタンをいったん離してもう一度押し直すまで回転しないはずです。一方、Continuous のボタンはマウスのボタンを押し続けている間ずっと回り続けるはずです。Step ボタンは **[クリック毎に 1 回実行]** がチェックされているので 1 回のボタン押しに対してコールバック関数が 1 回しか呼ばれず、Continuous ボタンは **[クリック毎に 1 回実行]** がチェックされていないのでボタンを押している間コールバック関数が呼ばれ続けるというわけです。違いがおわかりいただけたでしょうか。

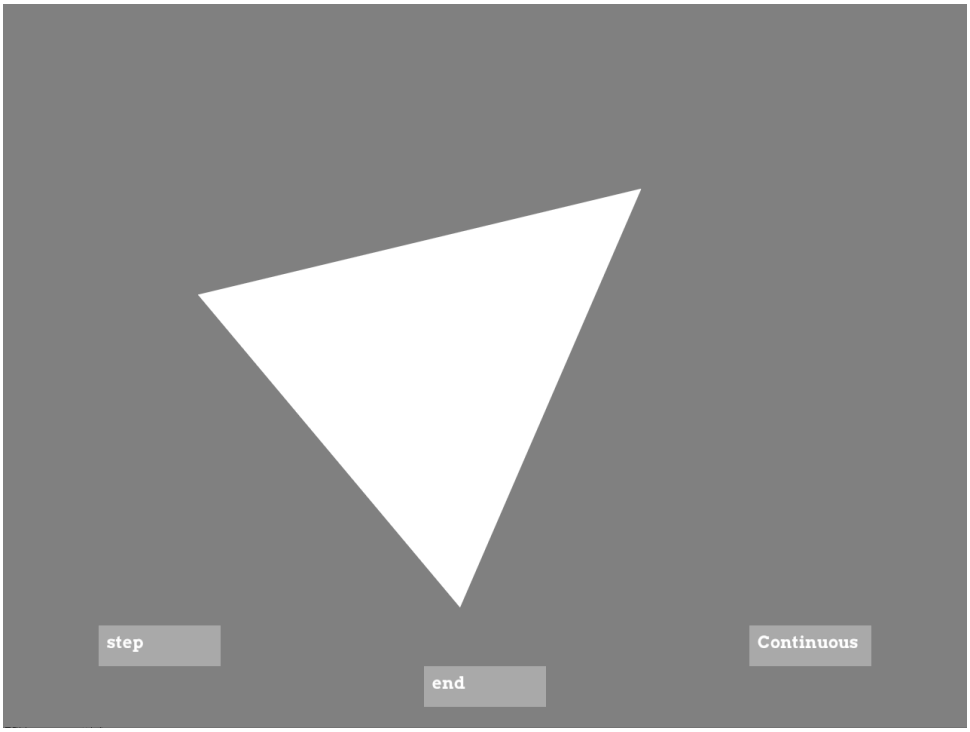


図 9.3　Step、Continuous のボタンを長押しして動作の違いを確認しましょう。end をクリックすると終了します。

Step ボタンと Continuous ボタンの違いを確認したら、end ボタンを押して実験を終了して記録ファイルを確認しましょう。Button コンポーネントひとつにつきボタン名 +.numClicks、ボタン名 +.timesOn、ボタン名 +.timesOff という 3 つの列が出力されていて、それぞれボタンがクリックされた回数、ボタン押しが始まった時刻、ボタン押しが終了した時刻が出力されています（時刻の単位は秒）。timesOn と timesOff に並んでいる時刻の個数は numClicks と一致して、timesOff の n 番目の数値から timesOn の n 番目の数値を引き算すれば n 回目のボタン押しで何秒間ボタンが押され続けていたかを計算することができます。

以上は Button コンポーネントによる出力でしたが、Variable コンポーネントによる出力も確認しておきましょう。angle.routineEndVal という列も出力されていて、ルーチン終了時の angle の値が出力されているはずです。このように、変数名 +. 保存タイミングを表す文字列の列名で値が出力されます。なお、実は筆者も最初ハマったのですが、 **[Routine 開始時の値] が設定されていなければ、[Routine 終了時の値を保存] をチェックしていても値は出力されません** (2021.1.2 で確認)。 これは実験製作者がうっかり「値が一度も設定されないまま値が保存」しようとしてしまった時に生じうるエラーを考えると仕方がないのですが、エラーメッセージや警告メッセージが何も表示されないので見落としやすいです。ご注意ください。

以上で Button コンポーネントと Variable コンポーネントのデモはおしまいです。マウスを反応に使用するの

なら Button コンポーネントは覚えておく価値があるでしょう。腕に自信がある人は、第 7 章 の実験をボタン押しでプローブ長を調節するように変更してみるとよい練習になると思います。Variable コンポーネントは好みがわかれるところだと思いますが、無視できない長所もあるので一度使ってみることをお勧めします。

チェックリスト

- Button コンポーネントを使ってルーチンを終了させることができる。
- Button コンポーネントを使ってボタンがクリックされたときにコードを実行させることができる。
- Button コンポーネントの **[クリック毎に 1 回実行]** のチェックの有無による動作の違いを説明できる。
- Variable コンポーネントを使って変数を設定し、ルーチン開始時に特定の値を設定することができる。
- Variable コンポーネントを使って変数の値を記録ファイルに出力することができる。

9.2 クリックした視覚刺激オブジェクトを記録しよう

Button コンポーネントとても便利ですが、ボタンのデザインを自由に変更したい場合や、ボタンが押されたときに自分自身以外のコンポーネントの情報を保存したい場合などには使用できません。そういった複雑な処理を行うには Code コンポーネントが必要ですが、Mouse コンポーネントの **[クリック可能な視覚刺激 $]** というプロパティを使用すると、Button コンポーネントよりは少し複雑な処理を Code コンポーネントに頼らずに実現できます。本節では **[クリック可能な視覚刺激 $]** の使用例を紹介しましょう。まずは、Button コンポーネントのようにクリックするとルーチンを終了する処理を作成してみます。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は、実験設定ダイアログを開いて **[単位]** を height にしておく。
- trial ルーチン
  - Polygon コンポーネントを 1 つ配置し、以下の通り設定する。
    - **[名前]** を button_yes にする。
    - **[開始]** を「時刻 (秒)」で 0.5 に、**[終了]** を空白にする。
    - **[形状]** を長方形にする。
    - **[塗りつぶしの色]** を white にする。
    - **[サイズ [w, h] $]** を [0.3, 0.1] にする。
    - **[位置 [x, y] $]** を [-0.2, 0] にする。
  - button_yes をコピーして **[名前]** を button_no にする。 **[位置 [x, y] $]** を [0.2, 0] にする。
  - Text コンポーネントを 1 つ配置し、以下の通り設定する。

    * **[開始]** を「時刻 (秒)」で 0.5 に、**[終了]** を空白にする。
    * **[前景色]** を black にする。
    * **[位置 [x, y] $]** を [-0.2, 0] にし、**[文字列]** を Yes にする。
  - 上記の Text コンポーネントをコピーし (名前は何でもよい)、 **[位置 [x, y] $]** を [0.2, 0]、 **[文字列]** を No にする。
  - Mouse コンポーネントを 1 つ配置し、以下の通り設定する。
    * **[開始]** を「時刻 (秒)」で 0.5 に、**[終了]** を空白にする。
    * **[ボタン押しで Routine を終了]** を有効なクリックにする。
    * **[クリック可能な視覚刺激 $]** に button_yes, button_no と入力する。
- trial ルーチンを繰り返すループを作成する。ループのプロパティはそのままで構わない。

完成したら実行してみましょう。 図 9.4 のような画面が表示されるはずです。マウスを操作して、灰色の背景のどこでクリックしてもルーチンが終了しないこと、白い長方形 (=ボタン) のどちらかをクリックするとルーチンが終了することを確認してください。左クリック、右クリックのどちらでもルーチンが終了することも確認しておくとよいでしょう。ループは初期値で 5 回となっているので、白いボタンを 5 回クリックしたら終了します。

終了したら、trial-by-trial 記録ファイルの内容を確認しましょう。 図 9.4 の下に示すように、mouse.clicked_name という列が存在していて、クリックした方のボタンに対応する Polygon オブジェクトの **[名前]** の値が保存されているはずです。

以上でおおよそ使い方はおわかりいただけたのではないかと思いますが、補足しておきましょう。Mouse コンポーネントの **[ボタン押しで Routine を終了]** を「有効なクリック」に設定して **[クリック可能な視覚刺激 $]** にカンマ区切りで視覚刺激の **[名前]** を列挙すると、列挙された刺激上にマウスカーソルを重ねた状態でボタンをクリックした時のみルーチンが終了します。この機能を使うと、画面上に描画したボタンをマウスでクリックするという方法で参加者の反応を記録することができます。

いくつか注意点を挙げておきますと、まず **[クリック可能な視覚刺激 $]** に挙げられた視覚刺激が重なり合っていた場合、重なっている位置をクリックしてもいずれかひとつの名前しか trial-by-trial 記録ファイルには保存されません。Polygon コンポーネントで描いた図形の場合は (十字などでも) 見た目通り、その図形上にカーソルの先端がなければ click と判定されませんが、Text コンポーネントの場合は文字の周辺に設定された透明な領域もクリックの対象となります。Image コンポーネントで透明な背景を持つ画像を使った場合も、見た目とは異なり透明な部分もクリックの対象となります。

[クリック時に保存するパラメータ $] は、刺激の色や回転角度などのパラメータが変化する実験において、クリック時の値を記録したい場合に便利です。これも簡単なデモを作ってみましょう。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は、実験設定ダイアログを開いて **[単位]** を height にしておく。
- trial ルーチン

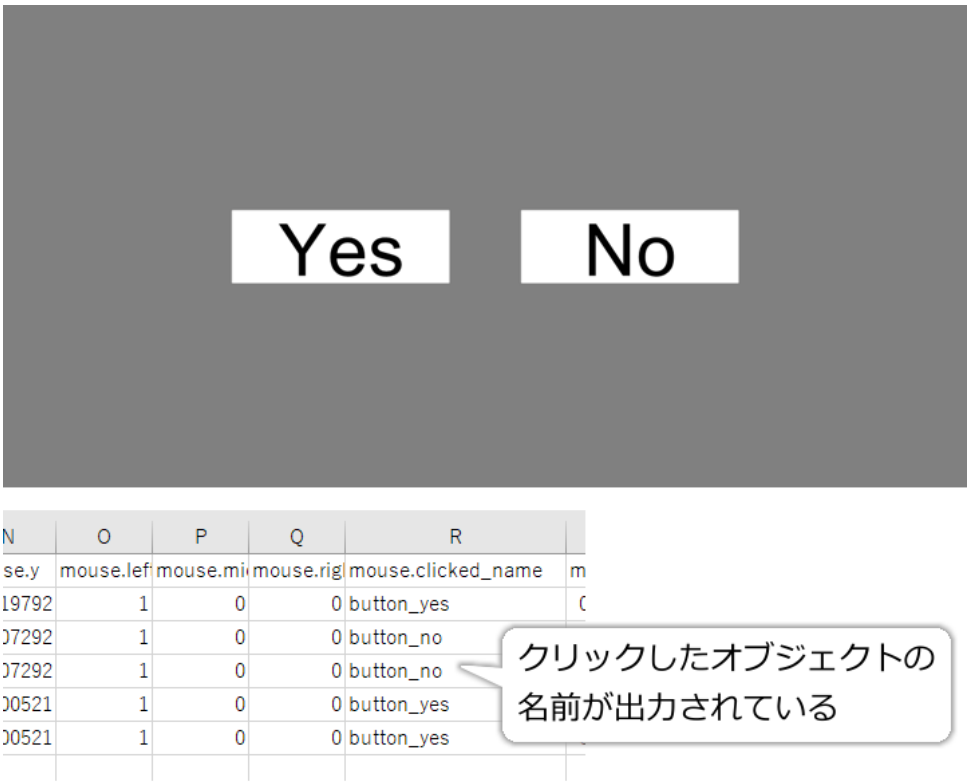


図 9.4　Yes、No のボタンを押してルーチンを終了します。どちらのボタンを押したかは記録ファイルで確認できます。

- Polygon コンポーネントを 1 つ配置し、以下の通り設定する。
  - **[名前]** を stim にする。
  - **[開始]** を「時刻 (秒)」で 0.5 に、**[終了]** を空白にする。
  - **[形状]** を三角形にする (初期値)。
  - **[サイズ [w, h] $]** を [0.1, 0.3] にする。
  - **[位置 [x, y] $]** を [0.2*cos(t), 0.2*sin(t)] にし、「フレーム毎に更新」にする。
  - **[回転角度 $]** を 45*t にし、「フレーム毎に更新」にする。
  - **[塗りつぶしの色]** を $[sin(t), 1, 1] にし、「フレーム毎に更新」にする。
- Mouse コンポーネントを 1 つ配置し、以下の通り設定する。
  - **[開始]** を「時刻 (秒)」で 0.5 に、**[終了]** を空白にする。
  - **[ボタン押しで Routine を終了]** を有効なクリックにする。
  - **[クリック可能な視覚刺激 $]** に stim と入力する。
  - **[クリック時に保存するパラメータ $]** に ori, pos, fillColor と入力する。

• trial ルーチンを繰り返すループを作成する。ループのプロパティはそのままで構わない。

完成したら実行してみましょう。三角形が色を変化させつつ回転しながら円軌道を移動していきます。適当な時点で三角形をクリックしてルーチンを終了させてください。5 回繰り返して終了させたら trial-by-trial 記録ファイルを確認しましょう。mouse.clicked_ori, mouse.clicked_pos, mouse.clicked_fillColor という列が存在していて、それぞれ回転角度、位置、塗りつぶしの色が出力されているはずです。便利な機能ですが、 **[クリック時に保存するパラメータ $]** には **[回転角度 $]** のようなプロパティ設定ウィンドウに表示されているプロパティ名ではなく、Builder 内部で使用されているオブジェクトのデータ属性の名前を書かなければならないことが難点です。 表 9.2 に示すパラメータ名は多くの視覚刺激で使用できるので、覚えておくと便利です。

表 9.2 **[クリック時に保存するパラメータ $]** で使える主なデータ属性

name	**[名前]** に対応
opacity	**[不透明度 $]** に対応
ori	**[回転角度 $]** に対応
pos	**[位置 [x, y] $]** に対応
size	**[サイズ [w, h] $]** に対応
color	**[前景色]** に対応
fillColor	**[塗りつぶしの色]** に対応 (Polygon コンポーネント)
lineColor	**[枠線の色]** に対応 (Polygon コンポーネント)

最後にちょっとした応用例を示しておきましょう。写真などの画像において特定の領域を参加者にクリックさせて反応を記録したいとします。このような領域はしばしば Region of Interest, ROI と呼ばれるので ROI と表記することにしましょう。このような実験を作りたいとき、画像を Image コンポーネントで表示して、ROI を Polygon コンポーネントで作成して **[クリック可能な視覚刺激 $]** に列挙するという方法をとることができます。 図 9.5 の右側は写真の例で、中央やや左寄りに野鳥が写っています。 図 9.5 上段は ROI を写真の上に描画した状態で、野鳥がいるところに赤い枠が描かれています。下段は写真を ROI の上に描画した状態で、実験参加者には ROI は見えません。しかし **[クリック可能な視覚刺激 $]** が別の視覚刺激の下に隠れていてもクリック判定されるので、クリックを検出できるというわけです。

実際にこの実験を作成するときには、写真のファイル名とそれに対応する ROI の位置とサイズを記入した条件ファイルを作成する必要がありますが、条件ファイルだけを眺めても適切に ROI が設定されているか判断するのは難しいので、Builder 上でコンポーネントの順番をひとつ変更するだけで ROI の描画を ON/OFF できるというのはきっと便利なはずです。 ROI を隠すには他にも **[不透明度 $]** を 0 にするなどの方法もありますが、ひとつの画面に ROI が複数個ある場合はすべての ROI のパラメータを変更しないといけませんので、コンポーネントの順番を変更する方が手間がかからないでしょう。

チェックリスト

- 視覚刺激をクリックしてルーチンを終了するように Mouse コンポーネントを設定できる。
- trial-by-trial 記録ファイルからクリックされた刺激の名前を読み取ることができる。
- trial-by-trial 記録ファイルにクリックした刺激の不透明度、回転角度、位置、サイズ、色を出力するように設定することができる。
- Image コンポーネントの下にクリック可能な視覚刺激を配置して、画像上の特定の領域をクリックするとルーチンが終了するようにすることができる。

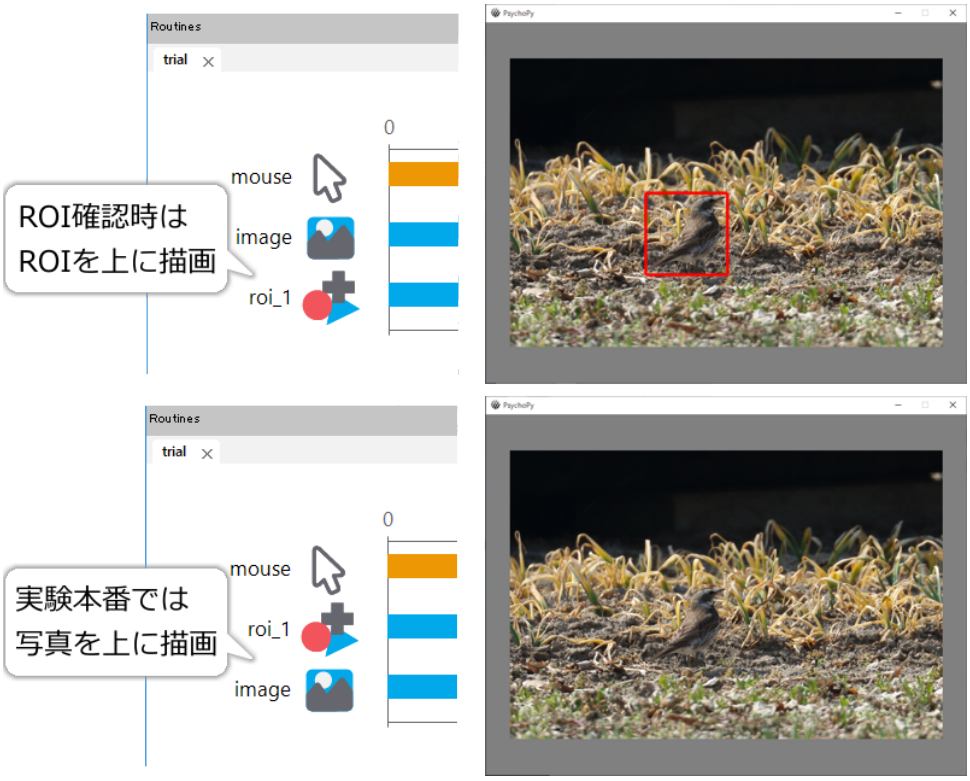


図 9.5　画像の特定の領域をクリックさせる課題を作成するときには、Image コンポーネントの下にクリック可能な刺激を隠すという方法があります。

9.3 Slider コンポーネントを使ってみよう

Slider コンポーネントは、定規のような「尺度」をスクリーン上に提示して、尺度上の位置を参加者に選択させることによって反応を記録するためのコンポーネントです。本書の第 4 版まではこの Slider コンポーネントに似た機能を持つ RatingScale コンポーネントを紹介していたのですが、現在のバージョン (本節の執筆時点で 2022.2.4) で RatingScale コンポーネントは廃止されてしまったので、今後は Slider コンポーネントを使用しなければいけません。すでに RatingScale を含む実験を持っている方向けに補足しておくと、バージョン 2022.2.4 の時点では RatingScale は Builder のコンポーネントペインに表示されないだけで、RatingScale を含む実験を実行することができます。将来 RatingScale が完全に削除してしまった場合は旧バージョンの PsychoPy で実行する必要があるでしょう。

図 9.6　Slider コンポーネントのアイコン。

そろそろ本題に入りましょう。Slider コンポーネントは「反応」カテゴリにあります (図 9.6)。まず「基本」タブから見ていきましょう。 **[名前]** 、 **[開始]** 、 **[終了]** は他のコンポーネントと同様です。 **[Routine を終了]** も

Keyboard コンポーネントと同じなので、問題はないでしょう。

「レイアウト」タブの **[空間の単位]** と **[位置 [x,y] $]** は他のコンポーネントと同様ですが、 **[回転角度 $]** は変更すると尺度のサイズの計算がおかしくなる場合があるので変更しないでください。**[サイズ $]** は幅、高さの順で 2 つの値を指定しますが、幅の値が高さの値より大きい場合は横向き、幅の値が高さの値以下ならば縦向きの尺度が描かれます。「より大きければ横／以下ならば縦」なので、幅と高さの値が同一ならば (実用的かどうかは別として) 縦向きとなります (図 9.7)。

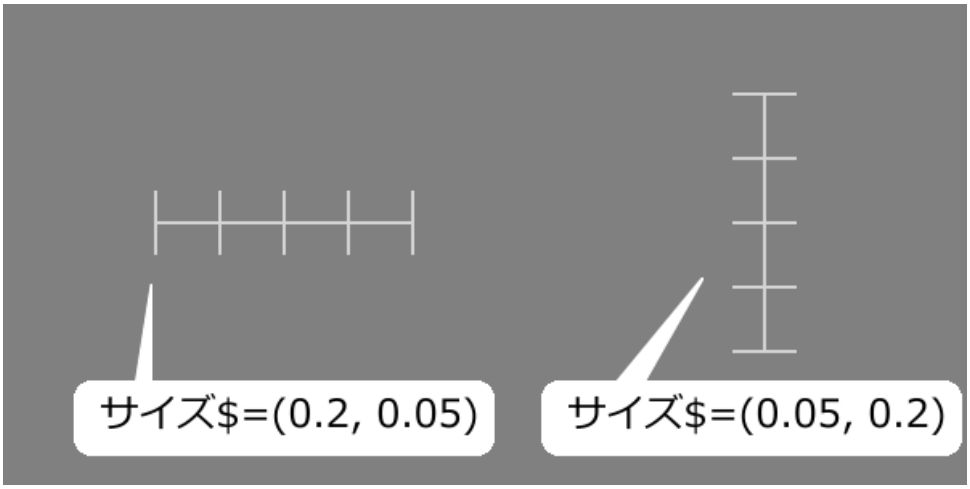


図 9.7　**[サイズ $]** の幅と高さの設定で縦向き、横向きが自動的に切り替わります。

「レイアウト」タブにはあと **[反転]** という項目がありますが、これをチェックするとラベルが反対側 (横向きなら上、縦向きなら右) に描かれます。

「基本」タブの **[目盛]** と **[ラベル]** は Slider コンポーネントの使いこなしのカギになるプロパティです。まず、**[目盛]** を指定すると尺度で記録される値は数値となります。目盛をつける位置を「小さい値から順番に」カンマ区切りで指定します。ばらばらな順番で並べると正常に動作しないので注意してください。図 9.8 下の例が示すように、数値は等間隔である必要はありません。数値の最小値と最大値がそのまま尺度で記録できる値の最小値と最大値になります。

[ラベル] は目盛に付けるラベルをカンマ区切りで並べます。' あてはまらない', ' どちらでもない', ' あてはまる' のように各項目を文字列として (つまりシングルクォーテーションかダブルクォーテーションで囲んで) 記述しますが、数値の場合は-2, 1, 0, 1, 2 のように書くこともできます。 **[目盛]** と **[ラベル]** で食い違う値を設定することもできますが、反応として記録されるのは **[目盛]** で定義した値です。極端な例を挙げると、 **[目盛]** が 1, 2, 3 で **[ラベル]** が-1, 0, 1 なら、画面上には **[ラベル]** で定義した-1, 0, 1 が表示されますが、-1 のところをクリックしたときに記録される値はそれに対応する **[目盛]** の値である 1 です。まあこんな使いかたをすることはないでしょうが、「まったくあてはまらない」を 0、「あまりあてはまらない」を 1、…という具合に数値を割り当てて分析するような場合には便利な機能です。

なお、**[目盛]** と **[ラベル]** の項目数は同じにするか、 **[ラベル]** の項目数を 2 つにしなければいけません。 **[ラベル]** の項目数が 2 つの場合は **[目盛]** の最小値と最大値に割り振られます。 **[ラベル]** の項目数が 3 つ以上で、なおかつ **[目盛]** の項目数と一致しない場合は正常に表示されません。 **[ラベル]** の項目が 1 つしかない場合はエラーとなり実行できません。

カテゴリカルな値を記録する尺度を作りたい場合は、 **[目盛]** を空欄にして **[ラベル]** に値を列挙します (図 9.9)。見た目は 図 9.8 下の例とよく似ていますが、 図 9.8 下の例で「全然」を選択すると 1 が記録される一方、

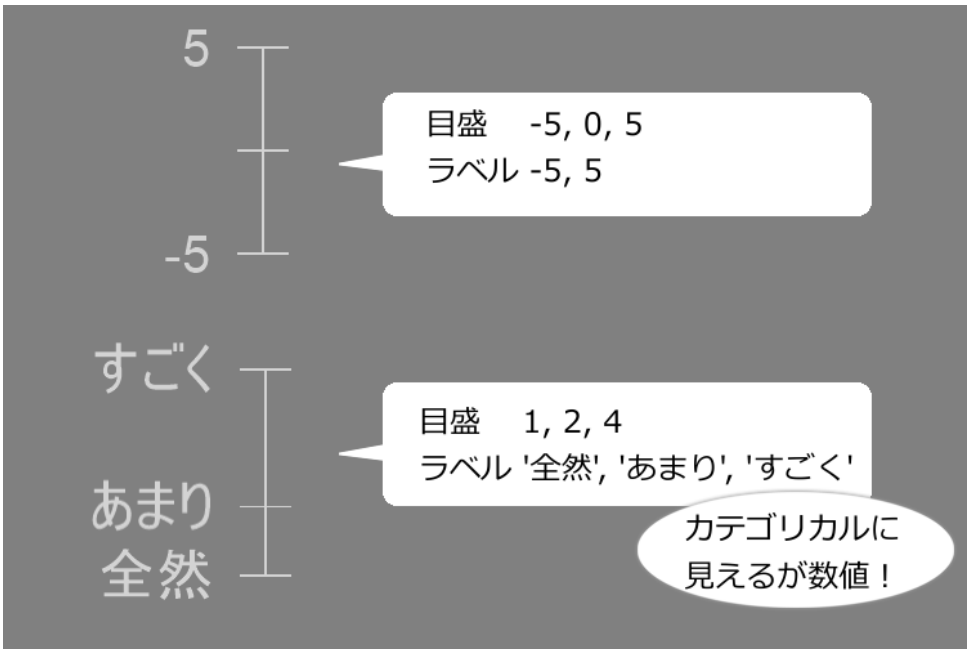


図 9.8　**[目盛 $]** に数値をカンマ区切りで並べると数値が記録される尺度となります。ラベルに文字列を指定しても、記録される値は数値です。

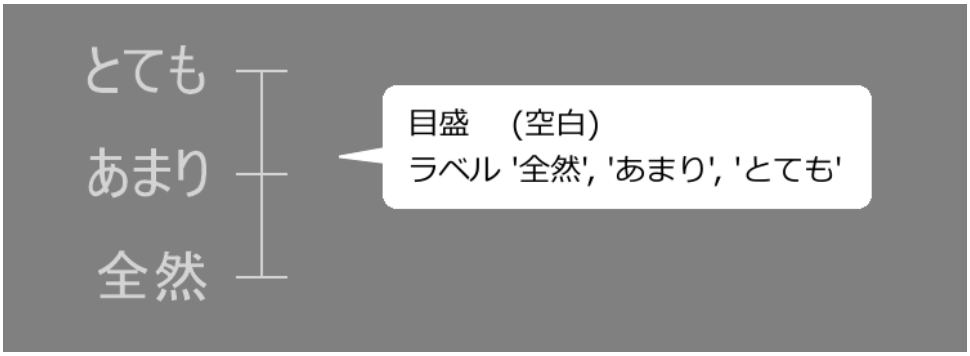


図 9.9　**[目盛 $]** を空欄にするとカテゴリカルな尺度になります。

図 9.9 の例では「全然」をクリックすると「全然」という文字列が記録されます。分析の方法に応じて使い分けるとよいでしょう。

[精度 $] は数値を記録する尺度でのみ意味を持ち、マーカーの位置が取りうる値を指定します。例えばこの値が 1 ならば、マーカーの位置は 1 の倍数、つまり整数となります。0 は例外的にシステムで可能な最小の値に対応します。 **[精度 $]** の設定により **[目盛]** の値を取り得ない場合 (例えば **[目盛]** に (3, 6, 9, 12, 15)、 **[精度 $]** に 4 を指定した場合) でもエラーにはなりませんが、混乱の元になりますので避けるべきでしょう。

[初期値] は最初から尺度上にマーカーが描かれている状態にしたいときに、その位置を指定します。カテゴリカル尺度の場合うまく機能しませんので (2022.2.4 で確認)、 **[目盛]** と **[ラベル]** の両方を指定して数値で位置を指定するといいでしょう。

続いて「外観」のタブに進みましょう。 **[ラベルの色]** 、 **[マーカーの色]** 、 **[枠線の色]** はそれぞれの部分の色を指定します。 **[不透明度 $]** は 2022.2.4 で確認した限り、目盛にのみ効果があるようです (つまり主線やラベルに効果がない)。 **[スタイル]** は尺度の見た目を変更するオプションです。 **[スタイルの微調整]** はラベルを 45 度傾けたりやマーカーを三角形にしたりといった微調整をおこないますが、2022.2.4 で確認したところ 45

度傾ける機能が無効になっているようです。旧バージョンでは正常に動作したので、一時的なバグの可能性もあります。 図 9.10 にスタイルの例を示します。

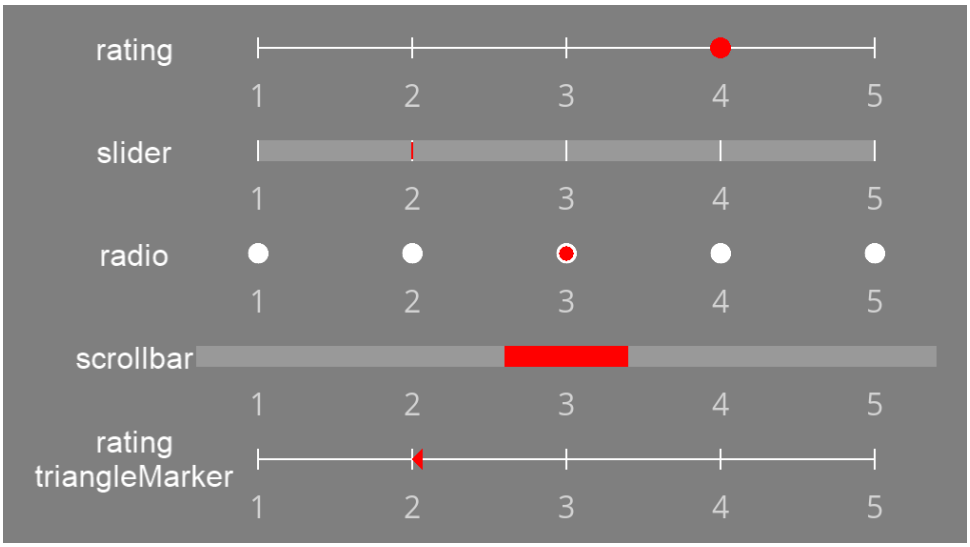


図 9.10　Slider のスタイルの例

「書式」タブは **[フォント]** と **[文字の高さ $]** のみで、Text コンポーネントなどと同じです。

「データ」タブは他のコンポーネントと共通のものに加えて **[読み込み専用]** 、 **[評定を記録]** 、 **[反応時間を記録]** 、 **[履歴を記録]** があります。 **[読み込み専用]** をチェックすると参加者が値を変更できなくなります。あまり使う機会はないでしょう。 **[評定を記録]** と **[反応時間を記録]** は初期値でチェックされていて、それぞれ参加者が選択した値と反応に要した時間 (単位は秒) を記録するかどうかを決定します。反応時間はともかく、**[評定を記録]** のチェックをオフにすることは通常の用途では無いと思います。 **[履歴を記録]** をチェックすると、反応を確定するまでに選択した値をすべてカンマ区切りで保存します。「基本」タブの **[Routine を終了]** にチェックが入っていると一回クリックしただけでルーチンが終了してしまうので、このチェックをオフにしないと意味がありません。ルーチンを終了させる他の手段と組み合わせて使う必要があります。

チェックリスト

- Slider コンポーネントで縦向き、横向きの尺度を描画できる。
- Slider コンポーネントで数値を記録する尺度とカテゴリカルな値を記録する尺度を使い分けることができる。
- Slider コンポーネントでスタイルを使い分けられる。

9.4 Slider コンポーネントで刺激をリアルタイムに調整しよう

Slider コンポーネントは反応の記録のほかにも、実験中の刺激パラメータの調整にも使うことができます。本節では、図 9.11 に示すような画面で、スライダー (本節の使い方では尺度と呼ぶのもおかしいのでスライダーと表記します) を動かしてリアルタイムに刺激の回転角度、不透明度、色を変更してみましょう。

さっそくですが、Builder で実験を新規作成して以下のように作業してください。

- 実験設定ダイアログ

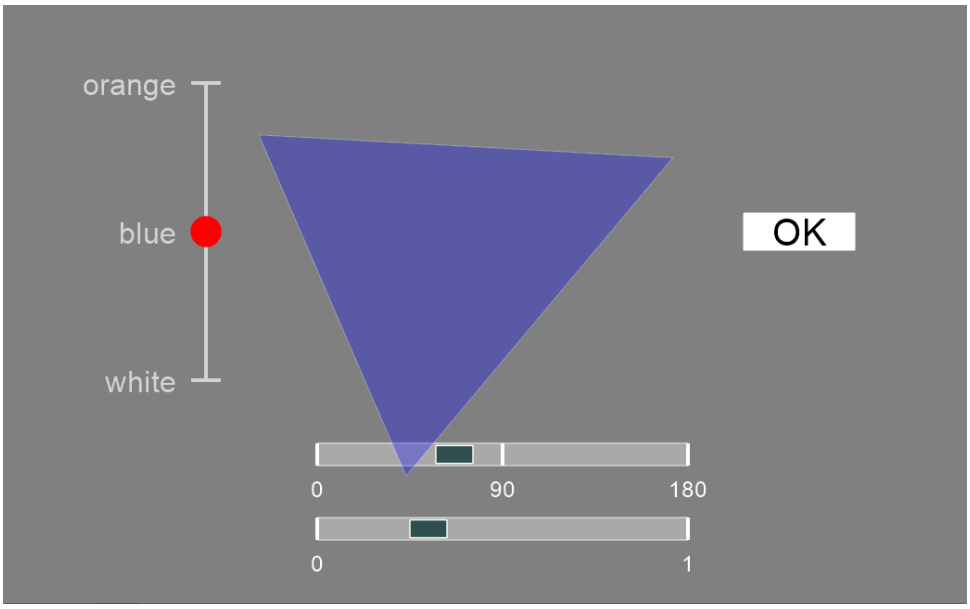


図 9.11　Slider で刺激のパラメータをリアルタイムに調整することができます。

  - PsychoPy の設定で height 以外の単位を標準に設定している場合は、実験設定ダイアログを開いて **[単位]** を height にしておく。

- trial ルーチン
  - Slider コンポーネント 1 つ配置して以下の通りに設定する。
    - **[名前]** を slider_color にする。
    - **[サイズ $]** を (0.04, 0.4) にする。
    - **[位置 [x,y] $]** を (-0.4, 0) にする。
    - **[目盛]** を空欄にして、 **[ラベル]** を 'white', 'blue', 'orange' にする。
    - **[Routine を終了]** のチェックを外す。
    - **[文字の高さ $]** を 0.03 にする。
  - slider_color をコピーして slider_ori の名前で貼り付け、以下の通りに作業する。
    - **[目盛]** と **[ラベル]** を 0,90,180 にする。
    - **[サイズ $]** を (0.5, 0.03) にする。
    - **[位置 [x,y] $]** を (0, -0.3) にする。
  - slider_ori をコピーして slider_opac の名前で貼り付け、以下の通りに設定する。
    - **[目盛]** と **[ラベル]** を 0, 1 にし、 **[精度 $]** を 0.2 にする。
    - **[位置 [x,y] $]** を (0, -0.4) にする。
  - Code コンポーネントをひとつ配置する。

- Polygon コンポーネントをひとつ配置して以下の通り設定する。
  - **[名前]** を stim にする。
  - **[終了]** を 空欄にする。
  - **[不透明度 $]** を slider_opac.markerPos として「フレーム毎に更新」にする。
  - **[回転角度 $]** を slider_ori.markerPos として「フレーム毎に更新」にする。
  - **[塗りつぶしの色]** を $polygon_color として「フレーム毎に更新」にする。
- Polygon コンポーネントをひとつ配置して以下の通り設定する。
  - **[名前]** を button にする。
  - **[終了]** を 空欄にする。
  - **[サイズ $]** を (0.4, 0) にする。
  - **[位置 [x,y] $]** を (0.15, 0.05) にする。
- Text コンポーネントをひとつ配置して以下の通り設定する。
  - **[終了]** を 空欄にする。
  - **[前景色]** を black にする。
  - **[文字の高さ]** を 0.05 にする。
  - **[位置 [x,y] $]** を (0.4, 0) にする。
  - **[文字列]** を OK にする。
- Mouse コンポーネントをひとつ配置して以下の通り設定する。
  - **[終了]** を 空欄にする。
  - **[ボタン押しで Routine を終了]** を「有効なクリック」にする。
  - **[マウスの状態を保存]** を「なし」にする。
  - **[クリック可能な視覚刺激 $]** に button と入力する。

最後に、Code コンポーネントにコードを入力する。まず、**[Routine 開始時]** のタブに以下のように入力する。

```
slider_ori.markerPos = 0
slider_opac.markerPos = 1
slider_color.markerPos = 0
polygon_color = 'white'
```

続いて **[フレーム毎]** に以下のように入力する。

```
color_choice = slider_color.getRating()
if color_choice is not None:
    polygon_color = color_choice
```

以上で作業は終了です。完成したら実行してみましょう。図 9.11 のような画面が表示され、スライダーを操作すると三角形の色や回転角度、不透明度が変化するはずです。特に回転角度がなめらかに変化するのに対して、不透明度はとびとびにしか変化しない点に注意してください。これは slider_opac の **[精度 $]** に 0.2 を指定しているからです。また、回転角度や不透明度はスライダーをドラッグ (マウスのボタンを押したまま動かす) するとリアルタイムに変化するのに対して、色の指定だけはマウスのボタンを離した時点で変化することも確認しておいてください。色の変更がうまくいかない場合は「9.7.1:Slider による色の変更が動作しない原因と解決例」を参考にしてください。

ひととおり操作したら右側の OK と書かれたボタンをクリックして終了してください。そして trial-by-trial 記録ファイルを開いて、OK をクリックしたときのスライダーの値が保存されていることを確認しましょう。回転角度は slider_ori.response、不透明度は slider_opac.response、色は slider_color.response といった具合に各 Slider コンポーネントの **[名前]** に.response と付けた列があって、そこに値が出力されているはずです。ちなみに反応時間は. **[名前]** に rt と付けた列に出力されます。OK をクリックした時刻ではなく各スライダーを最後に値を変更した時刻が出力されるので、それぞれで値が異なるはずです。

表 9.3　Slider オブジェクトの主な属性とメソッド

markerPos	マーカーの現在の位置を保持するデータ属性。初期状態では値は None で、マーカーは表示されていない (未選択の状態)。Code コンポーネントを使って値を代入することでマーカーの位置を変更できる。カテゴリカルな尺度の場合、最初の項目が 0、二番目の項目が 1 といった具合にインデックスで表現される。
getRating()	現在マーカーがある位置の値を得る。markerPos と異なり、カテゴリカルな尺度ではインデックスではなくその項目の値が得られる。

動作を確認し終えたところで、このデモの要点を確認しましょう。表 9.3 はこのデモで使用した Slider オブジェクトのデータ属性とメソッドを示しています。markerPos というデータ属性で現在のマーカー位置が得られるので、stim の **[不透明度 $]** にといったプロパティに slider_opac.markerPos 書いて「フレーム毎に更新」にするだけで反映させることができるわけです。

ただ、この方法にはひとつ問題点があって、Slider コンポーネントは初期状態で未選択 (マーカーが表示されない) となります。未選択の時には markerPos の値は None となるので、何も対策せずに視覚刺激のプロパティに書くとルーチン開始直後にエラーとなってしまいます。この「対策」というのが Code コンポーネントの **[Routine 開始時]** のコードの以下の部分です。markerPos には値を代入することもできるので、このように書いておくとルーチン開始時のエラーを回避できます。

```
slider_ori.markerPos = 0
slider_opac.markerPos = 1
```

これで一件落着…と言いたいところですが、まだ問題が残っています。それは、slider_color のようなカテゴリカルなスライダーの扱いです。カテゴリカルなスライダーの場合、markerPos には最初の項目が 0、二番目

の項目が 1…といった具合にインデックスで表現されています。ですので、ルーチン開始時に最初の項目である white が選ばれているようにするためには、Code コンポーネントの **[Routine 開始時]** に以下のように書けばよいはずです。

```
slider_color.markerPos = 0
```

ここまでは正しいのですが、問題は値を得るときです。カテゴリカルなスライダーにおいて、選択された項目の (インデックスではなく) 値を得るためには getRating() メソッドを用いる必要があります。ただ、getRating() は実験参加者が実際にスライダーを操作してマウスのボタンを離すまでは None を返すので、markerPos の値をコードでいじるだけでは解決しません。そこで、本節のデモでは if 文を使って getRating() の値が None の場合とそうでない場合で処理を分けています。まず、polygon_color という刺激の色を保持する変数を用意して、 **[Routine 開始時]** に以下のように初期値を代入しておきます。

```
polygon_color = 'white'
```

そして、 **[フレーム毎]** のコードで以下のように変数 color_choice に getRating() の結果を代入します。そして、color_choice の値が None でない場合に、polygon_color へ color_choice の値を代入します。あとはこの polygon_color を刺激の **[塗りつぶしの色]** に指定すれば問題解決というわけです。

```
color_choice = slider_color.getRating()
if color_choice is not None:
    polygon_color = color_choice
```

ちなみに、getRating() をフレーム毎に 2 回実行するのをいとわなければ color_choice という変数は省略して以下のように書くこともできます。

```
if slider_color.getRating() is not None:
    polygon_color = slider_color.getRating()
```

あとは特に新しいテクニックは使っていないはずなので、問題はないと思います。なお、このデモはひとつのルーチンに複数の尺度を設置してすべて反応させるパターンの例にもなっています。複数の RatingScale コンポーネントをひとつのルーチンに配置するとルーチンの終了条件の扱いが少々面倒でしたが、Slider コンポーネントとクリッカブルオブジェクトを併用すると比較的簡単に実現できるので、ぜひ活用してください。

チェックリスト

- Slider コンポーネントで刺激の不透明度、回転角度などの数値をリアルタイムに調整することができる。
- Slider コンポーネントで刺激の色名などのカテゴリカルな値をリアルタイムに調整することができる。

9.5 TextBox コンポーネントで文字を入力してみよう

TextBox コンポーネントはバージョン 2020.2 で追加された新しい機能で、従来の Text コンポーネントに加えて周囲に枠を描いたり、ボールド体や斜体を指定したり、行のアライメント (中央揃え、左揃えなど) を指定したりできます。これだけでも便利なのですが、さらにキーボードから文字を入力したり削除したりできるようにすることも可能なのです。実験の内容によっては参加者に自由記述をしてもらいとことがありますが、そういった実験も Builder で作ることができるようになります。

きっと「この機能を待っていました！」という人も多いであろう注目機能のひとつですが、2020.2.0 のリリース時に「まだ開発版なので実験に使う時は念入りに動作を確認しましょう (certainly in beta and should be tested carefully in your study)」と書かれていたように、まだまだ開発途上の段階です。英語のように語の間をスペースで区切る言語で、なおかつ入力に IME(Input Method Editor) を必要としない言語であればそこそこ実用的な段階に到達しているのですが、残念ながら日本語はどちらにも該当しません。まあ、前置きはこのくらいにしてどのような感じになるのか試してみましょう。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は、実験設定ダイアログを開いて **[単位]** を height にしておく。
- trial ルーチン
  - TextBox コンポーネント 1 つ配置して以下の通りに設定する。
    - **[名前]** を textbox にする (初期値のまま)。
    - **[終了]** を空欄にする。
    - **[編集可能]** をチェックする。
    - **[サイズ [w, h] $]** を (0.6, 0.4) にする。
    - **[枠線の色]** を white にする。
    - **[フォント]** に日本語に対応したフォントを指定する (Windows なら Meiryo、Mac なら Hiragino Sans など)。
    - **[行揃え]** を「左上」にする。
  - Button コンポーネントを 1 つ配置して、以下の通りに設定する。
    - **[終了]** を空欄にする。
    - **[Routine を終了]** がチェックされていることを確認する。
    - **[ボタンのテキスト]** に OK と入力する。
    - **[サイズ [w, h] $]** を (0.05, 0.03) にする。
    - **[位置 [x, y] $]** を (0.4, -0.4) にする。
    - **[文字の高さ $]** を 0.02 にする。

* **[クリックを記録]** を「なし」にする。

ここまで作業したら保存して実行してみましょう。 図 9.12 左上のように、四角い枠が描かれてその内側に **[文字列]**に入力されていた文字列が表示されているはずです。文章の各行が左寄せになっていることに注目してください。これはもちろん **[行揃え]** を「左上」にした効果ですが、こういったレイアウトは従来の Text コンポーネントでは面倒です。これだけを目当てに TextBox コンポーネントを使う価値があります。 **[塗りつぶしの色]** を指定すれば枠内に背景色をつけることもできます (もちろん **[枠線の色]** を None にして枠なしで背景色だけつけることもできます)。

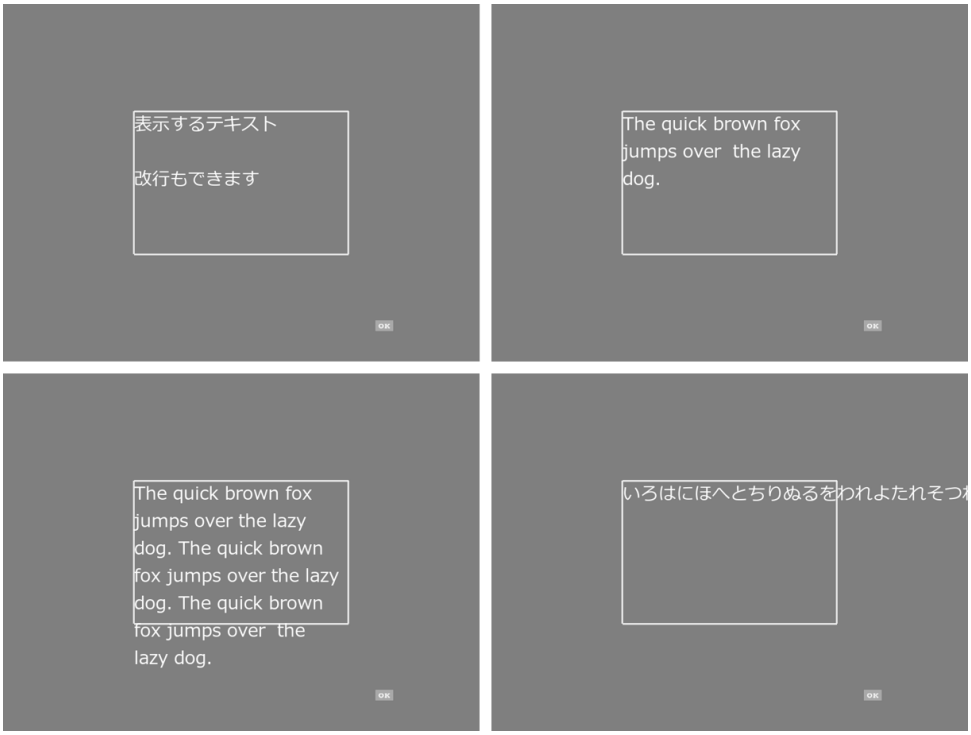


図 9.12　TextBox の例。左上は最初の状態、右上は英文を入力した状態、左下は長い英文を入力した状態、右下は日本語の文を入力した状態。

それでは続いてマウスを操作して枠内をクリックして、キーボードの BackSpace キーを押してください。表示されている文字列を削除できるはずです。すべて削除したら、適当な英文を入力してみましょう。内容はなんでもよいのですが、枠の幅より長い文を入力してください。行が枠からはみ出る長さになると自動的に 図 9.12 右上のように改行されるはずです。行末に半角スペースや改行文字を入力した時のカーソル (文字入力位置を示す縦線) の位置が通常とは異なるので少々戸惑いますが、おおむね自然に入力と削除ができるのではないかと思います。このように **[編集可能]** をチェックしておけば文字列を編集することができます。この例では **[文字列]** に初期値が入ったままにしましたが、 **[文字列]** を空欄にしておけば空っぽの枠内に文字を入力していくことができます。

最初に「TextBox はまだまだ開発途上の段階」と書きましたが、どのあたりが「まだまだ」なのか確認しておきましょう。まず、 図 9.12 左下のように枠内にどんどん英文を入力してみてください。行数が多くなって枠の高さを超えると、一般的なアプリのようにスクロールバーが表示されたりせずに枠からはみ出してしまいます。はみ出してしまっても正しく入力内容を保存できるのですが、あらかじめ参加者に断っておかないと参加者が不安になってしまいそうな挙動です。続いていったん英文を削除して、日本語で入力を行ってみましょう。まず、日本語入力を ON にしてキーを少し叩くと、入力中の文字が表示されないことに気づくと思います。Enter キーを押すなどして確定すれば表示されますが、かなり入力しづらいです。特に、漢字変換を行お

うとしたときに変換候補が見えないので、思い通りに変換することは非常に困難です。さらに、枠の幅を超える長さの文を入力すると、 図 9.12 右下のように自動改行されずに右へはみ出してしまうことがわかります。

こういった状況を考えると、少なくとも現時点 (バージョン 2022.2.4) では日本語の文章入力に TextBox コンポーネントを使うのは難しそうです。TextBox コンポーネントはオンライン実験にも対応しているのですが、オンライン実験は通常のインターネットブラウザで動作するため日本語入力の問題がすべて解決します (入力中の文字や変換候補もすべて見えるし、枠に入りきらない長さの文字列を入力すると自動的にスクロールバーがつく)。ですので、 **どうしても参加者に日本語入力をしてもらいたい実験があるのなら、オンライン実験として出力するという選択肢もあります。**

9.6 Form コンポーネントで複数の質問を画面上に配置しよう

心理学実験においては、 図 9.13 のように質問文と尺度を複数並べて回答してもらいたいという場面にしばしば出くわします。Slider コンポーネントと Text コンポーネントを使えばこのようなレイアウトを実現することは可能ですが、見栄えよくなラベルには位置やサイズの調整などがかなり面倒です。そこで、いくつかの制限の代わりにこのような複数の質問文とスライダーを並べたレイアウトを作成する Form コンポーネントというものが PsychoPy には用意されています。

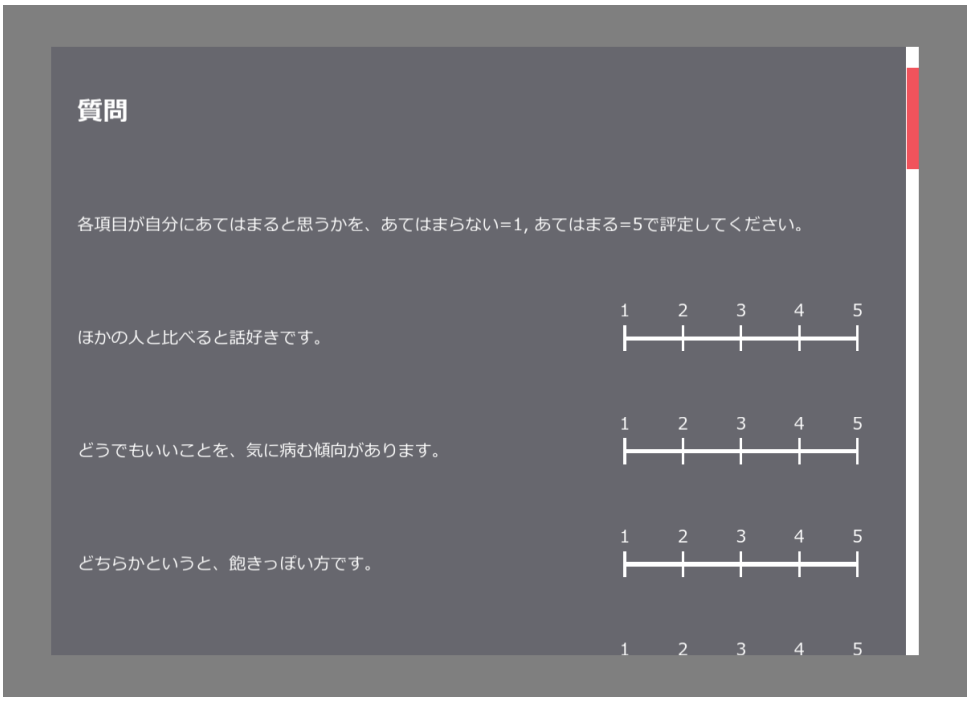


図 9.13　Form コンポーネントで作成できるレイアウト

図 9.14 に Form コンポーネントのアイコンを示します。「基本」タブにはおなじみの **[名前]** 、 **[開始]** 、 **[終了]** があり、続いて **[項目]** があります。この **[項目]** が Form コンポーネントの鍵となるプロパティで、質問文やスライダーのパラメータを定義した CSV ファイルまたは xlsx ファイルを指定します。

項目ファイルで定義できる内容を 表 9.4 に示します。1 行目に見出しを書き、2 行目以降に 1 行につき 1 項目の形で質問項目を定義していきます。後で具体的な例を挙げながら解説します。スクリーン上では項目ファイルに記入した順番に質問項目が表示されますが、 **[無作為化]** にチェックを入れておくと実行時に PsychoPy が項目の順序を無作為に並べ替えてくれます。**[データフォーマット]** は、データを出力する際の形式を指定しま

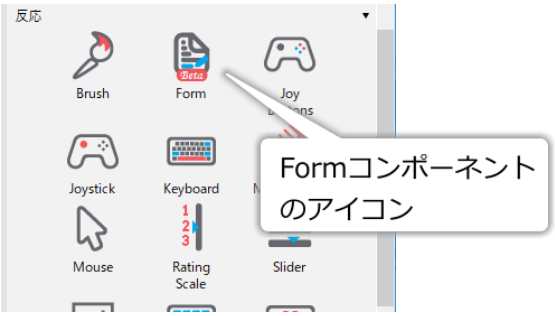


図 9.14　Form コンポーネントのアイコン

す。質問項目ひとつが「行」の場合はデータファイルの 1 行に、「列」の場合は 1 列に出力されます。

表 9.4: Form コンポーネントの項目ファイルで指定する内容

inde	項目の順序を整数で指定します。省略すると自動的に番号が割り振られます。刺激の描画には影響しませんが、 **[無作為化]** した時にデータファイルの出力を項目順に並び替えるときに役に立ちます。
itemText	表示したい文を指定します。
type	項目の種類を指定します。rating,slider,radio は Slider コンポーネントと同じです。radio は choice と書くことも出来ます。heading,description は教示などの文だけを表示したいときに使います (heading はやや大き目でボールド体になる)。heading, description を選んだ場合は options, ticks, ticklabels, layout, responseColor, responseWidth, granularity は意味を持ちません。ほかに文字列を入力できる free text を指定できますが、TextBox コンポーネントと同じで現状では日本語入力には使い物になりません (オンライン実験なら実用的)。
options	スライダーのラベルをカンマ区切りで指定します。ラベルに対応する値は自動的に決まりますが、両者を別々に設定したい場合は ticks と tickLabels を使います。
ticks	スライダーの目盛をカンマ区切りで指定します。
tickLabels	スライダーの目盛をカンマ区切りで指定します。
layout	スライダーの方向を horiz、vert のいずれかで指定します。horiz なら水平方向、vert なら垂直方向です。省略すると horiz になります。
itemColor	質問文の文字色を指定します。省略すると white になります。スタイル (後述) が custom... の場合のみ有効です。
itemWidth	Form コンポーネントの **[サイズ$]** で指定した幅のうち、質問文が占める幅の割合を 0.0 から 1.0 指定します。
responseColor	スライダーの色を指定します。省略すると white になります。スタイル (後述) が custom... の場合のみ有効です。
responseWidth	Form コンポーネントの **[サイズ$]** で指定した幅のうち、スライダーが占める幅の割合を 0.0 から 1.0 指定します。

次のページに続く

表 9.4 – 前のページからの続き

granularity	Slider コンポーネントの **[精度]** に対応しますが、type が slider の時のみ有効です。rating, radio, choice の時は目盛の位置しか選択できません。
font	使用するフォントは Form コンポーネントのプロパティダイアログの **[フォント]** で指定しますが、特定の項目だけ別のフォントを指定したい場合はここでフォント名を指定します。

「外観」タブの **[スタイル]** は全体的な配色を選択します。 dark と light という配色が用意されているほか、custom... という項目を選択して細かく指定することもできます (**[塗りつぶしの色]** などの項目は custom... にしないと有効になりません)。「レイアウト」タブの **[サイズ $]** は Form コンポーネントによって作成される質問項目全体を囲む枠の大きさに対応していると考えてください。 **[サイズ $] に対して項目数が多い場合は、枠の右端にスクロールバーが自動的に設けられます** 。 **[位置 [x, y] $]** は全体を囲む枠の中心の座標、 **[項目間の余白 $]** 文字通り項目間の余白を指定します。フォントの大きさは「書式」タブの **[テキストの高さ $]** で指定します。

それでは実際に作業しながら確認しましょう。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は、実験設定ダイアログを開いて **[単位]** を height にしておく。
- trial ルーチン
  - Form コンポーネント 1 つ配置して以下の通りに設定する。
    - **[名前]** を form にする (初期値のまま)。
    - **[項目]** を items.csv にする。
    - **[テキストの高さ $]** を 0.02 にする。
    - **[フォント]** に日本語に対応したフォントを指定する (Windows なら Meiryo、Mac なら Hiragino Sans など)。

以上の作業を行った実験を保存し (名前は何でも良いですが form_test.psyexp とでもしましょうか)、 図 9.15 の内容の xlsx ファイルを作成して同じ場所に items.csv という名前で保存してください。 **xlsx 形式にも対応しているのですが、xlsx 形式は読み込みに失敗することが多いので csv 形式で保存することを強くお勧めします。**

これで準備は完了です。Builder に戻って実験を実行してみましょう。 図 9.16 のように表示されれば成功です。マウスでスライダーを操作できますが、すべて選択しても終了しません。終了したいときは ESC キーを押して実験を中断してください。

それでは 表 9.4 、 図 9.15 、図 9.16 を見比べながら項目の指定方法を確認しましょう。まず、実行結果の上から 1 段目が heading、2 段目が description の出力例です。フォントの大きさや字体が異なることがわかります。

3 段目から 5 段目が rating、slider、radio の例です。いずれも itemText の列が質問文として表示されていることがわかります。英語などでは Form コンポーネントの幅に対して質問文が長すぎるときに自動的に改行され

	A	B	C	D	E	F
1	index	itemText	type	options	ticks	tickLabels
2		見出し	heading			
3		説明文の例です	description			
4		ratingの例	rating	1,2,3,4,5		
5		sliderの例	slider		1,2,3,4,5	1,2,3,4,5
6		radioの例¥n(choiceと書いても同じ)	radio	あてはまらない,あてはまる		
7						

左下に続く

右上から続く

G	H	I	J	K	L	M
layout	itemColor	itemWidth	responseColor	responseWidth	granularity	font
	yellow					
horiz	cyan	0.3	cyan	0.5		
horiz		0.3		0.5	0.1	
vert		0.3		0.5		

図 9.15　項目ファイルの内容。横に長いので途中で分割して 2 段にしているのでご注意ください。

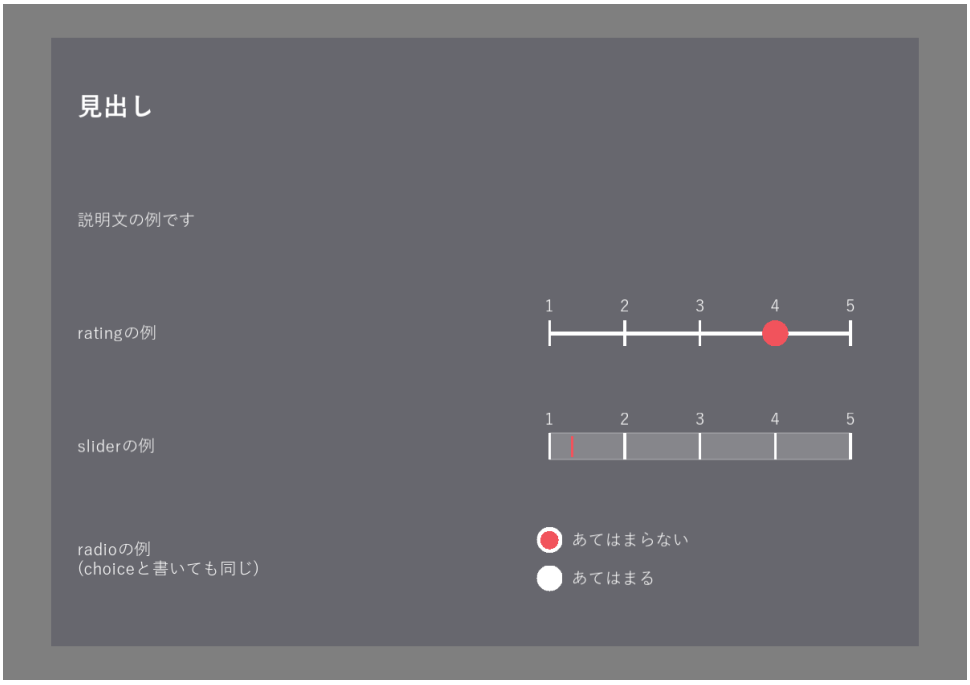


図 9.16　サンプルの実行例

るのですが、日本語では自動改行されません。どうしても改行させたい場合はこの例の最後の項目のように \n を使って改行します。ただし、出力されるデータファイル内でも改行されてしまいますので、データ処理の際に注意が必要です (Excel で開く場合は問題ないようです)。自動改行は半角スペースの位置で生じますので、一度改行なしで表示させてみてから「ここで改行してほしい」という位置に手作業で半角スペースを入れる方法もあります。

続いて質問文の右側に表示されているスライダーですが、Slider コンポーネントの rating、slider、radio と対応していることがおわかりいただけると思います。options, ticks, tickLabels は機能が重複していてややこしいのですが、おそらくこれは旧バージョンとの互換性を保つためだと思います。旧バージョンでは options のみで項目を指定する必要があったため、Slider コンポーネントのように **[ラベル]** と **[目盛]** を独立に指定することができませんでした。そこで **[ラベル]** に対応する tickLabels、 **[目盛]** に対応する ticks が追加されたので

すが、旧バージョンの options も残されているのでしょう。options のみを使うか、ticks と tickLabels を組み合わせて使うのがよいと思います。使わない方は items.csv のように空欄にしておいて問題ありません。

layout、itemWidth、responseWidth、itemColor、responseColor は見た目を調整するパラメータです。layout はスライダーの向きを horiz(水平) か vert(垂直) で指定します。図 9.16 の最後の項目が vert の例です。このように radio と vert 組み合わせると、ラベルが少し長めの文字列の場合でもうまく枠内に収めることができます。itemWidth と responseWidth は Form コンポーネントの枠の幅に対して質問文とスライダーの幅が占める割合を 0.0 から 1.0 で指定します。これらの値の合計が 1.0 を超えてしまうと、項目とスライダーが重なってしまったり枠からはみ出たりしてしまって正常に表示されません。合計が 1.0 に満たない場合は質問文とスライダーの間に空白ができます。

itemColor と responseColor は質問文とスライダーの色の指定です。他のコンポーネントと同様に RGB の値を-1.0 から 1.0 で表したリストや色名が利用できます。 **[スタイル]** で dark または light が選択されている場合はそちらの色設定が優先されるので、サンプルを最初に実行した時には指定した色が反映されていないはずです。Builder で配置済みの Form コンポーネントのプロパティダイアログを開いて、 **[スタイル]** を custom... に変更して実行してみてください。

items.csv に新たな行を追加して項目を増やしてみたり、itemWidth, itemColor, responseWidth, responseColor, granularity, font などを自由に編集して、効果を確かめてみてください。特に行数が多くなって枠内に全項目が入りきらなくなった時に、枠の右側にスクロールバーが出現してスクロールさせて全項目を操作できるようになることを確認してください。

一通り確認を終えたら続いてデータファイルの確認をしたいのですが、現状では ESC キーを押して中断するしかないため正常に終了できるようにしないといけません。Form コンポーネントを置いてあるルーチンを終了させる方法はいろいろ考えられますが、Form の操作でマウスを使用しますので、OK ボタンを用意してクリックして終了するようにするのがよいでしょう。ここでは Button コンポーネントを使います。

- trial ルーチン
  - Button コンポーネントを 1 つ配置して、以下の通りに設定する。
    - **[終了]** を空欄にする。
    - **[Routine を終了]** がチェックされていることを確認する。
    - **[ボタンのテキスト]** に OK と入力する。
    - **[サイズ [w, h] $]** を (0.05, 0.03) にする。
    - **[位置 [x, y] $]** を (0.4, -0.4) にする。
    - **[文字の高さ $]** を 0.02 にする。
    - **[クリックを記録]** を「なし」にする。

これでスクリーンの右下に OK と書かれたボタンが表示され、これをクリックすると終了できるようになりました。ただ、このままでは質問に全く答えずにいきなり OK をクリックしても終了してしまいます。すべての質問に答えるまで終了できないようにするには、第 2 章 で「とりあえず無視」した機能を利用します。各コンポーネントの **[開始]** および **[終了]** のプルダウンメニューには「条件式」という項目があります。「条件式」を選択すると、この欄に記入した式の値が True になった時点でコンポーネントを開始したり終了したりすることができます。第 6 章 以後、if 文などを通じて条件式を学んできた今なら、この機能を活用できるはずです。

問題は Form コンポーネントの質問にすべて答えた時に True になる式をどのように書くかですが、これは難問です。Builder のコンポーネントにはそれに対応する PsychoPy のオブジェクトがあるという話を 第 6 章 でしましたが、Form コンポーネントの場合は psychopy.visual.Form というオブジェクト (以下 Form オブジェクト) が対応しています。この Form オブジェクトには formComplete() というメソッドがあり、すべての質問に回答済みの場合に True、未回答の質問があれば False が返されます。今回の目的にぴったりです。今回の実験では Form コンポーネントの名前を form としていますので、

```
form.formComplete()
```

という式を書けばよいことがわかります。実験を以下のように修正しましょう。

- trial ルーチン
  - Button コンポーネントの **[開始]** を「条件式」に変更し、form.formComplete() と入力する。

修正が終わったら実行してください。最初は右下の OK ボタンは表示されていなくて、Form コンポーネントの質問をすべて回答したらボタンが出現するはずです。**[開始]** および **[終了]** で条件式を使う方法は応用範囲が広く、使い回こなすといろいろと面白いことができます。 第 10 章 でも出てきますので、ぜひ参考にしてください。

さて、ここまで作成した実験を実行し、すべての質問に回答して終了すると、いつも通り CSV 形式のデータファイルが保存されているはずです。データファイルの例を 図 9.17 に例を示します。通常のコンポーネントはループでの繰り返し 1 回につき 1 行の形式でデータを出力しますが、Form コンポーネントは複数行にわたってデータを出力します。 **[データフォーマット]** が「行」の場合は質問 1 項目につき 1 行の形式で項目のインデックス (項目ファイルの index の列で指定した値)、質問文、反応、反応時間などが出力されます。 **[データフォーマット]** が「列」の場合は「行」形式の出力内容を 1 行に展開した形式となり、一見 1 行に収まっているように見えるのですが、ルーチン内に他のコンポーネントが存在してデータを出力した場合、Form コンポーネントの出力は他のコンポーネントと別の行になります。この意味で、1 回の繰り返しにつき 1 行の形式が守られていません。人が目視で確認する場合は「行」の方が見やすいと思いますし、プログラムで処理する場合はどちらも「繰り返し 1 回＝ 1 行」のルールが破られてしまうので工夫が必要です。使いやすいと思う方を選択していただければと思います。

最後にひとつ注意点を挙げておきます。ループの中で Form コンポーネントを使用した場合、2 回目以降の実行時には前回の反応が残っています。その方が便利な場合もあるかもしれませんが、繰り返しのたび初期化したいケースが多いのではないでしょうか。2 回目以降の実行時に未反応の状態に戻すには、Form オブジェクトの reset() というメソッドを使用します。Code コンポーネントを用いて Routine 開始時に以下のコードを実行するとよいでしょう。

```
form.reset()
```

チェックリスト

- Form コンポーネントで複数の質問項目とスライダーのペアをルーチンに配置することができる。
- Form コンポーネントのすべての項目に回答したらルーチン終了のボタンを表示させることができる。
- ループの中で Form コンポーネントを使う時に繰り返しのたびに反応を初期化することができる。

行の場合

	A	B	C	D	E	F
1	form.start	button.sta	form.index	form.item	form.type	form.granu
2	8.467402	12.78178	0	見出し	heading	None
3			1	説明文の例	descriptio	None
4			2	ratingの例	rating	None
5			3	sliderの例	slider	0.1
6			4	radioの例(choiceと書いても同じ)	radio	None
7						

列の場合

	A	B	C	D	E	F
1	form.start	button.sta	form[0].in	form[0].ite	form[0].ty	form[0].gr
2	8.033167	10.6834	0	見出し	heading	None
3						
4						
5						
6						

図 9.17　Form コンポーネントの出力

9.7 この章のトピックス

9.7.1 Slider による色の変更が動作しない原因と解決例

本書を第 5 版に改定するにあたって本章の内容を確認していたところ、バージョン 2022.2.4 において Slider による Polygon コンポーネントの色の変更が動作しないことに気づきました。2021 系の最終バージョンである 2021.2.3 では動作すること、特に仕様変更に関するアナウンスがないことから一時的なバグではないかと考えていますが、サンプルが動かないのは困りますので原因と解決例を記しておきます。

単刀直入に言うと、この問題の原因は、2022.2.4 では getRating() がカテゴリカルな尺度でもマーカーの位置を浮動小数点数で返すことです。ということは、解決するにはマーカー位置から項目の値を得ればよいでしょう。 **[ラベル]** に'white', 'blue', 'orange' と 3 項目を入力していて **[目盛]** が空欄ですから、getRating() の戻り値は white を選択したとき 0.0、blue なら 1.0、orange なら 2.0 です。したがって、Code コンポーネントの **[フレーム毎]** のコードを以下のように変更すれば期待通りの動作となります (3 行目のみが異なる)。

```
color_choice = slider_color.getRating()
if color_choice is not None:
    polygon_color = ['white', 'blue', 'orange'][int(color_choice)]
```

第 10 章

音声と動画を活用しよう

PsychoPy Builder では音声ファイルや動画ファイルを再生したり、マイクを使った音声の録音やカメラを使った動画の記録をしたりすることができます。動画機能についてはいろいろと技術的な難しさがあって、どのような動画ならば問題なく再生、録画できるか簡単には判断できないのですが、試してみることはそんなに難しくないのでぜひみなさん自身の PC で動かして確認してみてください。前章に引き続き、まとまった実験を作成するのではなくデモと解説を中心に進めます。

10.1 Sound コンポーネントで音声ファイルを再生しよう

Sound コンポーネントは Builder で音声刺激を扱うためのコンポーネントです。 図 10.1 に Sound コンポーネントのアイコン及びプロパティ設定ダイアログを示します。Sound コンポーネントのプロパティの内、これまでに紹介済みのコンポーネントと共通ではないのは「基本」 **[音]** と「再生」タブの **[ボリューム $]** 、 **[ハミング窓]** です。 **[音]** には無圧縮 WAV 形式の音声ファイルを指定できるほか、A や Bfl (B ♭)、Csh (C#) のようにキーコードで音を指定することもできます。また、2000 という具合に正の数値を入力すると、その周波数の音が鳴ります。実行環境によっては WAV 以外に OGG などの音声ファイルを再生できますが、無圧縮 WAV ならほとんどの環境で再生できるので無難です。 **[ボリューム $]** は 0.0 から 1.0 の範囲でボリュームを指定します。再生環境や音声ファイル形式によってはうまく機能しませんので、可能なら音声データ作成の時点でボリュームを調整していた方が良いでしょう。 **[開始]** および **[終了]** で定められた時間が音声ファイルの時間より短い場合は、音声ファイルの再生が途中で終了します。 **[ハミング窓]** は音声のオンセットによるプチノイズを軽減するフィルタを使用するか否かを設定します。チェックしておいた方が無難ですが、およそ 1 ミリ秒ほど音の立ち上がりが遅れるので、極めて正確な時間制御が必要な場合はチェックをオフにした方が良い結果が得られるかもしれません。

図 10.1　Sound コンポーネントのアイコン。

音声ファイルを用いた実験を行う時にしばしば困るのが、「音声ファイルが再生されている間文字列が表示され、再生終了と共に消える」といった処理や、「音声ファイルの再生が終わったら文字列が表示されるようにしたいが、ルーチンは継続したいので **[Routine を終了]** は使いたくない」という場合です。使用する音声ファイルの再生時間がすべて同じであれば **[開始]** や **[終了]** の値を再生時間に合わせて設定すればいいのですが、ファイルによって再生時間が異なる場合は工夫が必要です。具体的には、Sound コンポーネントに対応する PsychoPy クラスが持っている status というデータ属性を利用します。音声または動画ファイルが再生されていなければ、status は NOT_STARTED という値が設定されています。再生中であれば PLAYING (または STARTED)、再生が終了していれば STOPPED (または FINISHED) です。これを利用すると、Code コンポーネントを用いて以下のように stim という名前の Sound コンポーネントの再生終了時にルーチンを強制終了させることができます。

```
if stim.status == FINISHED:
    continueRoutine = False
```

ルーチン全体を終了させるのではなく、特定のコンポーネントの描画を開始したり終了したりしたい場合は、そのコンポーネントの **[開始]** および **[終了]** で「条件式」 を使用すると便利です。 図 10.2 に音声ファイルの再生開始、終了に合わせてコンポーネントの開始、終了する例を示します。

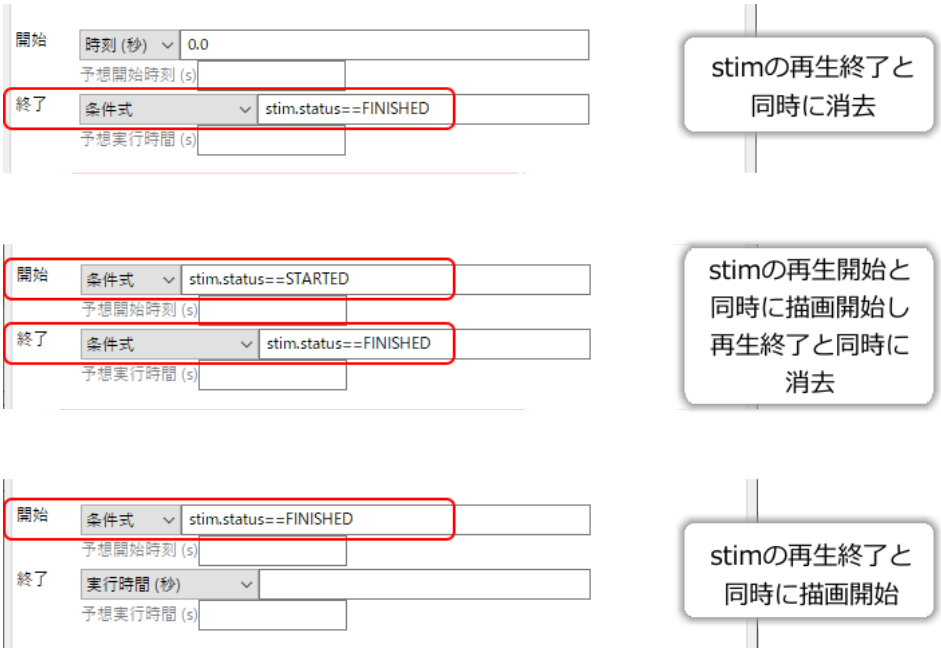


図 10.2　**[開始]** および **[終了]** に 「条件式」 を指定すると、条件式によってコンポーネントの開始、終了を制御できます

最後にふたつ注意点を挙げておきます。まず、Sound コンポーネントによる音声の再生タイミングはかなり「いいかげん」です。例えば「ぴぴっ」と音を短い音を 2 回鳴らしたいとします。再生時間 0.1 秒の Sound コンポーネントを 2 個配置して、それぞれの **[開始]** を 0.5 秒ずらしてやると「ぴぴっ」となるはずですが、実行する PC によっては 1 回しか音がならなかったり、全く音がならなかったりします。元々、PC のオーディオ機能はエラー音などを鳴らしたり、ひとつの音声ファイルを鳴らしたりするためのもので、短時間に複数の音声を正確に再生する機能は保証されていません。このような場合は、2 つの音を 1 つの音声ファイルにまとめるべきです。視覚－聴覚の相互作用の研究を考えておられる方は刺激を動画として作成するのもひとつの対策でしょう。なお、再生タイミングの問題は、PsychoPy の設定で「オーディオライブラリ」に PsychToolbox を指定して、オーディオレイテンシの設定を厳しくすることでかなり改善されます。 図 10.2 のように PsychoPy

の設定ダイアログの「ハードウェア」のページを開いて「オーディオライブラリ」の先頭に PTB が来るように修正してください。PTB が含まれていない場合は先頭に入力してください。オーディオレイテンシの設定は一般的には 3. で十分ですが、4. にするとハードウェアがベストな設定に対応していない場合にエラーとなるので確実です (エラーとなる場合は実験用 PC を変えるか妥協するかを選ぶことになるでしょう)。

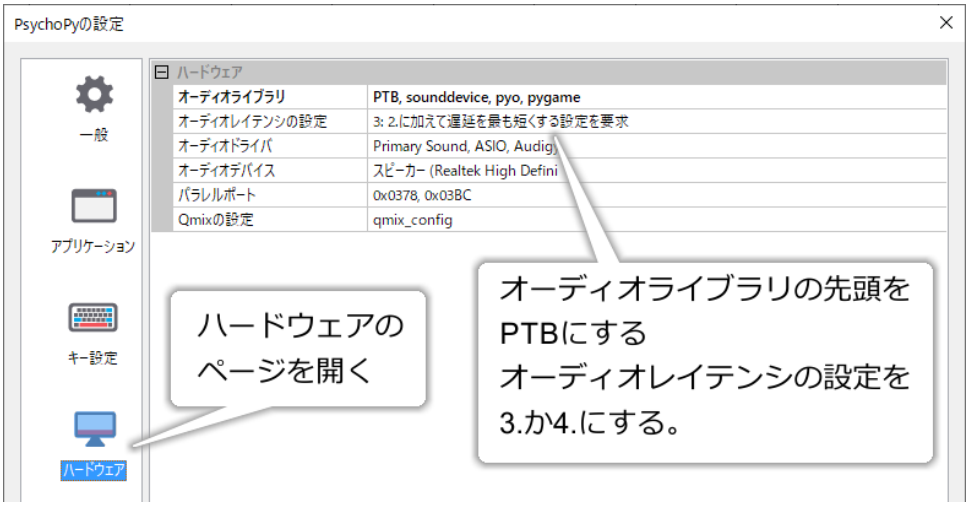


図 10.3　オーディオライブラリに PsychToolbox を指定すると再生の遅延が大幅に改善される場合があります。

もうひとつの注意点は、異なるサンプリングレートで作成された音声ファイルをひとつの実験で使用しない方がよいということです。例えば、実験で音声ファイルを 10 個使用しているうちの 8 個が 44.1kHz、2 個が 48kHz でサンプリングされているといった状況です。実験の実行環境によっては音声ファイル読み込みの時点でエラーが起こって実験が強制終了されてしまうことがあります。原因がわかりにくいエラーなので気をつけてください。

チェックリスト

- 無圧縮 WAV 形式の音声ファイルを再生できる。
- 指定された周波数の音を鳴らすことができる。
- 指定されたキーコードの音を鳴らすことができる。
- 音声のボリュームを指定できる。
- 音声ファイルの再生を指定された時刻に途中終了できる。
- 様々な再生時間の音声ファイルの再生開始、終了に合わせて他のコンポーネントを開始または終了させることができる。
- 短時間に複数の Sound コンポーネントを鳴らそうとした時に期待した結果が得られない理由を説明できる。
- 異なるサンプリングレートの音声ファイルをひとつの実験で混ぜて使用してはいけない理由を説明できる。

10.2 Movie コンポーネントで動画を再生しよう

Builder で動画を再生するには Movie コンポーネントを使用します (図 10.4)。Movie コンポーネントのプロパティの内、これまでに紹介済みのコンポーネントと共通ではないのは 「基本」タブの **[動画ファイル]** と 「再生」タブの **[バックエンド]** 、 **[音声無し]** 、 **[ループ再生]** です。

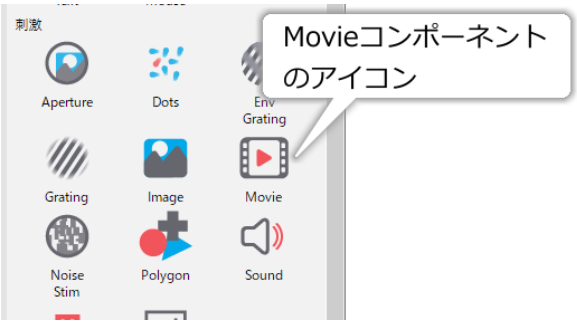


図 10.4　Movie コンポーネントのアイコン。

[バックエンド] は、これは PsychoPy が動画データを再生するときに使用するライブラリの指定です。PsyhcoPy Builder のユーザーから見ると「Movie コンポーネント」が操作画面に見えていて実際に操作する対象であり、これを「フロントエンド」と呼びます。それに対して、Movie コンポーネントが動画再生のために内部で利用しているライブラリが「バックエンド」です。ffpyplayer、moviepy、opencv、vlc の 4 つが選択できます。それぞれ使用するライブラリが異なります。どれがよいと一概には言えないのですが、とりあえず最新のバックエンドである ffpyplayer を試してみることをお勧めします。

[動画ファイル] には、再生する動画ファイル名を指定します。再生できる動画ファイルの形式はバックエンドによって決まりますが、moviepy(FFmpeg) なら一般的な形式はほとんど再生できると思います。動画ファイルのフォーマットはお勧めできる定番がないのですが、筆者は MP4 形式をよく使用しています。

「レイアウト」タブの **[サイズ [w, h] $]** を動画ファイルと異なる値に設定することによって、動画を縦横に拡大縮小して再生することができます。動画ファイルの元の解像度のまま再生する場合は **[サイズ [w, h] $]** は空白にします。ただ、このように PsychoPy 上で拡大縮小できるからといって、実際に描画するサイズより解像度が高い動画ファイルを縮小表示するべきではありません。具体的にいうと、実験用に撮影した動画の解像度が 1920 × 1080 で、実験に使用する時の表示サイズが 480 × 270 であるならば、実験に使用する前に動画編集ソフトを用いて 480 × 270 に縮小すべきです。といいますのも、第一に解像度の高い動画ファイルは (よほど画質を落としていない限り) ファイルサイズが大きいので、その分 PC のメモリを消費してメモリ不足を起こすかもしれません。第二に、縮小処理を PsychoPy に任せると縮小の品質が悪くて細い線や小さな文字などが鮮明に描画されないかも知れません。できる限り高品質な (恐らく時間がかかる) 方法で前もって縮小しておき、画質に問題がないことを確認しておくべきです。

「再生」タブの **[音声無し]** は文字通り音声なしで再生します。音声を再生せずに済むならその分負荷を軽減できます。実験の目的上音声が必要ないならチェックしておくとよいでしょう。 **[ループ再生]** は文字通り、これがチェックされていると動画をループ再生します。

チェックリスト

- 動画ファイルを拡大縮小して再生することができる。
- 動画ファイルを音声なしで再生することができる。

10.3 Movie コンポーネントを使ってみよう

それでは実際に Movie コンポーネントを使ってみましょう。なにか手ごろな動画ファイルを用意してください。動画ファイルはそのフォーマットと動画データの符号化方法が一対一対応していないので非常にややこしいのですが、mp4 形式の動画ファイルなら多くの場合 ffpyplayer バックエンドで再生できるはずです。動画ファイルが用意できたら作業を始めましょう。

- 実験設定ダイアログ
  - **[単位]** を pix にする。
- trial ルーチン
  - Movie コンポーネントをひとつ配置して以下の通り設定する。
    - **[名前]** を movie にする (初期値)。
    - **[終了]** を空欄にする。
    - **[動画ファイル]** に使用する動画ファイルを指定する。相対パスが使える点などは Image コンポーネントと同様である。
    - **[空間の単位]** を pix にする (本来不要だが反映されないことがあるので指定すること)。
  - Text コンポーネントをひとつ配置して以下の通りに設定する。
    - **[終了]** を「条件式」にして movie.status == FINISHED と入力する。
    - **[文字の高さ]** を 48 にする。
    - **[文字列]** に $delay と入力し、「フレーム毎に更新」にする。
  - Code コンポーネントをひとつ配置して、他のコンポーネントより先に実行されるよう一番上に並び替える。

作業が終了したら、Code コンポーネントの **[フレーム毎]** に以下のコードを入力する。

```
delay = t - movie.getCurrentFrameTime()
```

完成したら実行してみましょう。動画のフォーマットが非対応でなければ、画面中央に動画が再生されてその上に数値が表示されます。この数値はルーチンの時計 t と動画の再生位置の時刻の差なのですから、動画が遅延なく再生できていればほとんど変動しないはずです。本来ならばこの値が 0 になるのが理想ですが、どうしてもある程度の差は生じます。他のコンポーネントと連携させたいときに、この程度の時間のズレはあるというつもりで実験を作成するようにしてください。

数値の更新が速すぎて読めないという方は「8.11: 軌跡データを間引きしよう」の方法で 変数 frameN を使って間引きをするといいでしょう。以下にヒント（というかほとんど答え）を示しますが、よい練習になるので自分で考えてみてください。

```
if frameN % 10 == 0:
    delay = t - movie.getCurrentFrameTime()
```

再生の遅延や時間のズレは、同一の PC でも再生する動画の負荷によって変動します。いろいろなサイズの動画を用意して、 **[動画ファイル]** の項目を書き換えていろいろ実行してみると良いでしょう。また、 **[サイズ [w, h] $]** に動画の解像度と異なる値を (例えば (480,270) のように) 指定してみて、どの程度の影響が出るかを試してみてください。

このデモのうち、Text コンポーネントと Code コンポーネントは時刻などの情報を出力するために配置しているものであり、動画を再生するだけなら不要です。Code コンポーネントに記入したコードについては解説が必要ですね。バックエンドが moviepy の Movie コンポーネントを配置すると、その **[名前]** に指定した変数に psyhcopy.visual.MovieStim オブジェクトが作成されます (moviepy なら MovieStim3、opencv なら MovieStim2)。getCurrentFrameTime() メソッドは、現在の動画フレームの時刻を返します。1 行目で変数 mt に getCurrentFrameTime() の値を代入しておいて、2 行目で文字列に埋め込んでいます。t はこれまでの章で使ってきた、現在のルーチンが開始してからの時刻を保持している内部変数です。

t と getCurrentFrameTime() の差がどの程度だったか実験のたびに保存しておかないと心配だという方は、第 7 章 で解説した方法を使って差を変数に保持し、trial-by-trial 記録ファイルに出力するとよいでしょう。まず、trial ルーチンを繰り返すようにループを作成してください。 **[名前]** は trails、 **[繰り返し回数]** は 1 でいいでしょう。ループを作成したら、の Code コンポーネントの **[Routine 開始時]** に以下のコードを追加します。

```
delay_list = []
```

続いて **[フレーム毎]** の最後に以下のコードを追加しましょう。間引きをした人は字下げに注意してください。

```
delay_list.append(delay)
```

最後に **[Routine 終了時]** に以下のコードを追加します。average() と std() は 表 5.2 で出てきた平均値と標準偏差を計算する関数です。

```
trials.addData('delay_mean', average(delay_list))
trials.addData('delay_std', std(delay_list))
```

これで trial-by-trial 記録ファイルに t と getCurrentFrameTime() の差の平均値と標準偏差が出力されるようになりました。なお、このコードを実際の実験で使用するときは、動画再生終了後直ちにルーチンを終了するようにしてください。そうしないと、もうすでに再生していない動画の時刻との差を append し続けてしまいます。

動画再生終了後もルーチンを継続する必要がある場合は、動画再生中のみ差を append するようにすればいいでしょう。Code コンポーネントに慣れていないとちょっと難しいかもしれませんので、これも例を出しておきましょう。動画再生終了後もルーチンを 5 秒間継続して、計算した平均値と標準偏差を画面上に表示することにします。以下の通り作業してください。 **[文字列]** のところでは「 6.9.1: 改行文字を使った複数行の文字列の表現 (上級) 」で解説した方法を使用しています。

- trial ルーチン
  - Text コンポーネントをひとつ配置して、以下の通り設定する。
    - **[開始]** を「条件式」にして movie.status == FINISHED と入力する。
    - **[終了]** を「実行時間 (秒)」にして 5 と入力する。

* **[文字の高さ]** を 48 にする。
* **[文字列]** に $' 平均: ' + str(average(delay_list)) + 'n 標準偏差:' + str(std(delay_list)) と入力し、「フレーム毎に更新」にする。

そして、Code コンポーネントの **[フレーム毎]** に入力しているコードに if 文を追加しましょう。2 行目と 3 行目が既に入力済みの部分です。

```
if movie.status == PLAYING:
    delay = t - movie.getCurrentFrameTime()
    delay_list.append(delay)
else:
    delay = 0
```

動画が再生されている時にはデータ属性 status の値が PLAYING になっているので、if 文を使ってその時のみ差を計算して append するようにしました。else の後の部分は動画が再生されていないときにも delay という変数が存在するのを保証するために設けています。else 以下を削除してしまうと実験開始直後に「delay という変数がない」というエラーメッセージが表示されて実験が止まります。 **[Routine 開始時]** に delay=0 と書いておくことでも回避できます。このあたりの小細工がピンと来るようなら Code コンポーネントにかなり慣れてきたと思ってもよいのではないでしょうか。

チェックリスト

- Code コンポーネントを使って動画の再生中のフレーム時刻を得ることができる。
- Code コンポーネントを使って動画の再生中のみ実行する処理を記述することができる。

10.4 動画の再生位置を変更してみよう

実験に使用する動画は、できる限り実験の準備段階で実際に提示するとおりの状態にしておくことが理想ですが、実験によっては条件に応じて動画の特定の時点から再生したいということがあるかも知れません。そのような時に便利なのが動画のシークです。Builder で動画のシークを行うには Code コンポーネントを通じて動画オブジェクトの seek() メソッドを用います。これもデモを作成してみましょう。 **5 秒以上の長さがある動画** を用意してください。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
- trial ルーチン
  - Code コンポーネントをひとつ配置する。
  - Movie コンポーネントをひとつ配置して以下の通り設定する。
    * **[名前]** を movie にする (初期値)。
    * **[終了]** を空欄にする。

    * **[動画ファイル]** に使用する動画ファイルを指定する。5 秒以上の長さがあるものを使用してください。

そして、Code コンポーネントの **[Routine 開始時]** に以下のコードを入力してください。

```
movie.seek(5.0)
```

完成したら実行してみましょう。動画の冒頭を 5 秒飛ばしたところから再生が始まったはずです。ここで用いた seek() というメソッドは、動画の再生位置を引数で指定した値に変更します。引数の単位は秒です。

「動画の終了の 5 秒前」のように終了時刻を基準に指定したい場合は、動画オブジェクトの duration というデータ属性を利用します。duration には動画の長さが保持されているので (単位は秒)、以下のように指定すればよいでしょう。

```
movie.seek(movie.duration - 5.0)
```

というわけで動画の頭出しができるようになりましたが、実験作成の時点で何秒飛ばすかがすでに決まっている場合はその分をカットした動画を作成した方が良いのは間違いありません。そうすると、実際にこのテクニックを使う状況は、実験中の参加者の反応によって飛ばす時間が変化する場合くらいかもしれません。

チェックリスト

- 動画を途中から再生開始することができる。
- 動画の再生開始位置を先頭から、または末尾からの秒数で指定することができる。

10.5 マウスで一時停止やスキップを行えるようにしよう (上級)

シークの話題があっさり終わってしまったので、ここで少し遊んでみましょう。画面中央下に 図 10.5 のようなボタンを表示し、に「5 秒戻る」、「5 秒進む」、「一時停止／再開」、「終了」の機能を割り振ってマウスで操作できるようにしてみます。「遊び」というのは実験としての実用性があまりないからですが、Builder でマウスベースのユーザーインターフェースを作る際の参考になるのではないかと思います。新たに実験を作成し、以下の通り作業してください。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
- trial ルーチン
  - Movie コンポーネントをひとつ配置して以下の通り設定する。
    * **[名前]** を movie にする (初期値)。
    * **[終了]** を空欄にする。
    * **[動画ファイル]** に使用する動画ファイルを指定する。数十秒以上の長さがあるものが望ましい。
  - Polygon コンポーネントをひとつ配置して以下の通り設定する。

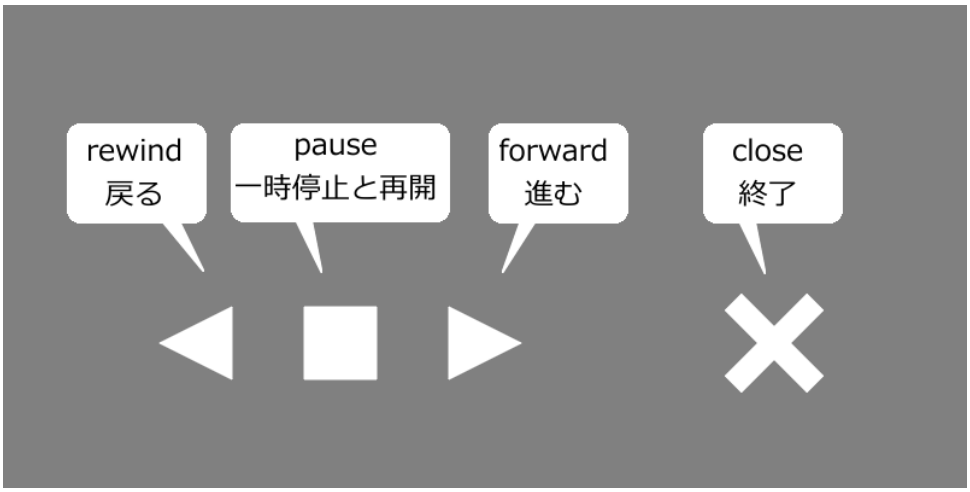


図 10.5　マウスでボタンをクリックして動画再生をコントロールしてみます。

  * **[名前]** を rewind にする。
  * **[終了]** を空欄にする。
  * **[形状]** を「三角形」にする。
  * **[塗りつぶしの色]** を white にする。
  * **[サイズ [w, h] $]** を (0.05, 0.05) にする。
  * **[回転角度 $]** を-90 にする。
  * **[位置 [x, y] $]** を (-0.2, -0.4) にする。
- rewind をコピーして、forward という名前で貼り付けて以下の通り設定する。
  * **[回転角度 $]** を 90 にする。
  * **[位置 [x, y] $]** を (0.0, -0.4) にする。
- Polygon コンポーネントをひとつ配置して以下の通り設定する。
  * **[名前]** を pause にする。
  * **[終了]** を空欄にする。
  * **[形状]** を「長方形」にする。
  * **[塗りつぶしの色]** を white にする。
  * **[サイズ [w, h] $]** を (0.05, 0.05) にする。
  * **[位置 [x, y] $]** を (-0.1, -0.4) にする。
- Polygon コンポーネントをひとつ配置して以下の通り設定する。
  * **[名前]** を close にする。
  * **[終了]** を空欄にする。

    * **[形状]** を「十字」にする。
    * **[塗りつぶしの色]** を white にする。
    * **[回転角度 $]** を 45 にする。
    * **[サイズ [w, h] $]** を (0.07, 0.07) にする。
    * **[位置 [x, y] $]** を (0.2, -0.4) にする。
  - Mouse コンポーネントをひとつ配置して以下の通り設定する。
    * **[名前]** を mouse にする (初期値)。
    * **[終了]** を空欄にする。
    * **[ボタン押しで Routine を終了]** を「有効なクリック」にする。
    * **[マウスの状態を保存]** を「なし」にする。
    * **[クリック可能な視覚刺激]** に close と入力する。
  - Code コンポーネントをひとつ配置する。

ここまで作業が終わったら、Code コンポーネントにコードを入力します。未解説のメソッドとデータ属性をいくつか使いますのが、それらは本章の最後に表としてまとめておきます。

まず **[Routine 開始時]** に以下のコードを入力します。この変数 step はボタンを押したときに進む/戻る時間を保持しています。この値を変更すると進む/戻る時間が変わります。

```
step = 5.0
```

続いて **[フレーム毎]** に以下のコードを入力します。ここで pause() と play() というメソッドが出てきますが、これはそれぞれ動画再生の一時停止、再開を行うものです。

```
ct = movie.getCurrentFrameTime()

if mouse.isPressedIn(rewind):
    movie.pause()
    movie.seek(max(0,ct-step))
    movie.play()
elif mouse.isPressedIn(forward):
    movie.pause()
    movie.seek(min(movie.duration,ct+step))
    movie.play()
elif mouse.isPressedIn(pause):
    if movie.status == PLAYING:
        movie.pause()
    elif movie.status == PAUSED:
        movie.play()
```

簡単に処理内容を解説しておくと、まず getCurrentFrameTime() で現在の再生位置を得て変数 ct に代入しておきます。続いて 第 8 章 で少し触れたマウスオブジェクトのメソッドである getPressedIn を使って rewind、forward、pause の各 Polygon オブジェクト上でマウスボタンが押されたかを判定していきます。isPressedIn の引数に指定されたオブジェクト内にマウスカーソルがあってボタンが押されていたら True が返されるので、if 文で処理を分岐します。

rewind と forward でマウスのボタンが押されていた時の処理では、先ほどの変数 ct に step を加算、もしくは減算して seek() を実行しています。これで「現在再生中の位置から step 秒戻る、または進む」が実現できます。ただ、再生中に seek() を行うと音声は進んでいるのに映像が止まったままになったりすることがあるので、seek() の前に pause() でいったん再生を止め、そして seek() 後に play() で再開しています。pause() と play() をコメントアウトして比較してみるとよいでしょう。

pause でマウスのボタンが押されていた時は、動画の再生状態に応じて処理を分岐します。再生中であればデータ属性 status の値が PLAYING、一時停止中であれば PAUSED となっているので、if 文で分岐して PLAYING ならば pause()、PAUSED ならば play() を実行します。

ここには close を押したときの終了処理が書かれていませんが、それは Mouse コンポーネントの **[クリック可能な視覚刺激]** に close を設定することで実現されているので、Code コンポーネントを使う必要がありません。

ここまで作業ができたら、一度保存して実行してみましょう。画面中央に動画が再生され、画面中央下に 図 10.5 のように表示されますので、クリックして動作を確認してください。PsychoPy にとって動画の一時停止、再生再開は重い処理なので、一般的なスペックの PC だと各ボタンをクリックしてから効果が表れるまで一呼吸待たされますのでそのつもりでいてください。なお、動画の再生が終了してもデモは終了しませんので、close を押して終了してください。

いかがだったでしょうか。「戻る」と「進む」、「終了」は (やや待たされるかもしれませんが) 特に問題なく動作したと思います。でも、「一時停止」はうまくいくこともあれば、一瞬だけ止まってすぐ動きだしたりしなかったでしょうか？ なぜそうなるかというと、isPressedIn() は「現在マウスのボタンが押されているか」を返すメソッドだからです。PC の画面が 60fps で描かれている場合、 **[フレーム毎]** の処理は 1/60 秒に 1 回実行されます。今回のデモでは pause() や play() が少々重い処理なので 1/60 秒内で終わらないこともありますが、それでも人がマウスのボタンを「カチッ」とクリックした間に数フレームは過ぎてしまいます。そうすると、現在のコードでは 1 回「カチッ」とボタンを押しただけで pause() と play() が繰り返し実行されてしまいます。なので、クリックした後にうまく一時停止する場合と再生されて続けてしまう場合があるのです。

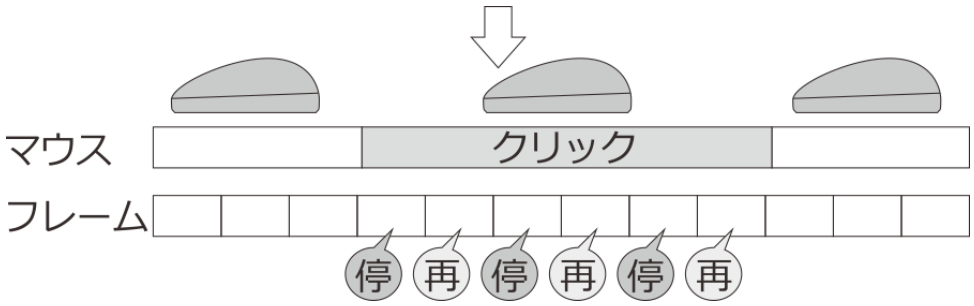


図 10.6　クリックが数フレームにわたった場合に対応する必要がある

ではどうすればいいかというと、いくつか方法があります。ここでは mouse.isPressedIn(pause) が True になったときの時刻を保持しておいて、一定時間が経過するまでは mouse.isPressedIn(pause) が True であっても無視するという方法を取り上げましょう。まず Code コンポーネントの **[Routine 開始時]** に以下の行を追加し

ます。

```
wait = 0
```

続いて **[フレーム毎]** に以下のように追加します。追加した行の最後に目印としてコメントを入れています。皆さんが試してみるときはこれらのコメントを入力する必要はありません。

```
ct = movie.getCurrentFrameTime()
if wait > 0:     # (1)
    wait -= 1    # (1)

if mouse.isPressedIn(rewind):
    movie.pause()
    movie.seek(max(0,ct-step))
    movie.play()
elif mouse.isPressedIn(forward):
    movie.pause()
    movie.seek(min(movie.duration,ct+step))
    movie.play()
elif mouse.isPressedIn(pause) and wait <= 0:  # (2)
    wait = 30                                 # (3)
    if movie.status == PLAYING:
        movie.pause()
    elif movie.status == PAUSED:
        movie.play()
```

この例では、wait という変数を用意して、一時停止と再開の処理を行う前に wait=30 をセットしています (コメント (3))。その後、フレーム毎に wait の値が 0 より大きければ 1 を減算していきます (コメント (1))。そしてコメント (2) のところでボタン押しを判定する際に wait<=0 を条件として追加することで、wait の値が 0 より大きい間はボタンを押しても一時停止/再開が行われないようにしています。この wait という変数はルーチン開始時に存在していないといけないので、**[Routine 開始時]** に wait=0 としているわけです。実行してみて、一時停止が機能することを確認してください。

なお、この方法ではボタンが押されてからの時間をフレーム数でカウントしていることになります。コメント (3) で wait=30 としているので 30 フレーム、60fps で刺激提示しているのなら待ち時間は 0.5 秒です。長すぎる場合は数値を減らしてみましょう。また、この例では戻る、進む処理について対策していませんが、これらについても対策を追加するにはいくつか書き方があるので、考えてみると良い練習になるでしょう。

他の方法としては、直前のフレームのボタンの状態を mouse.getPressed() で保持しておいて、「直前のフレームでボタンが押されていなくて、今のフレームでは押されている」という時だけ処理するという方法が考えられます。この場合、もうボタンが押されていることは確実なので isGetPressedIn() ではなく contains() でクリックされた Polygon オブジェクトを判別できます。

フレーム数を数える方法は、操作している人がボタンをずっと長押しすれば処理が繰り返されます。それに対して、直前のフレームの状態を保持する方法では、長押ししても処理されるのは 1 回のみです。どちらがの方が望ましいかは状況によるでしょうから、どちらの方法もマスターしておくのが理想です。

以上、Movie コンポーネントと Mouse コンポーネントで「遊んで」みましたがいかがだったでしょうか。最後に、この章で使用した Movie オブジェクトのデータ属性とメソッドを 表 10.1 にまとめておきます。

表 10.1　Movie オブジェクトの主なデータ属性とメソッド

status	現在の状態をあらわす。再生前なら NOT_STARTED、再生中なら PLAYING (STARTED)、一時停止中なら PAUSED、再生終了なら STOPPED (FINISHED)。
duration	動画の長さ。単位は秒。
getCurrentFrameTime()	動画の現在の再生位置を返す。単位は秒。
play()	動画の再生を開始する。一時停止している場合は再生を再開する。
pause()	動画の再生を一時停止する。
seek()	動画の再生位置を変更する。単位は秒。

チェックリスト

- 動画の再生を一時停止、再開できる。
- Code コンポーネントで動画が一時停止中であることを条件に処理を分岐できる。
- マウスでオブジェクトを「クリック」した際にボタンが押されている期間が複数フレームにわたる場合を考慮したコードを記述できる。

10.6 Microphone コンポーネントで録音してみよう

音声と動画の再生の解説が終わったので、続いて録音、録画の解説に進みましょう。まず音声の録音を行う Microphone コンポーネントを取り上げますが、バージョン 2022.2.4 の時点で **Microphone コンポーネントにはバグがあり、かなり用途は制限されると考えておいてください。** Microphone コンポーネントは「反応」カテゴリにあります (図 10.7)

図 10.7　Microphone コンポーネントのアイコン

Microphone コンポーネント独自のプロパティとして、まず「基本」タブの **[デバイス]** が挙げられます。ここでは録音に使用するデバイスを選択します。初期値は default で、OS で設定されている標準の録音デバイスを

使用します。他に PsychoPy で検出されたデバイスが列挙されているので、複数の録音デバイスが PC に存在している場合にどれを使うか指定できます。

「音声文字変換」タブは録音した音声から自動的に発話を書き起こす機能の設定を行います。 **[音声の文字変換]** をチェックするとこの機能が有効になり、他の項目の設定が可能になります。まず **[音声文字変換バックエンド]** で使用するライブラリを指定します。Built-in は Python の PocketSphinx パッケージ、Google は Google Cloud API を使用します。Built-in はネットワークなしでも実行できますが、現状では日本語に未対応と考えてください。Google Cloud API はインターネット接続が必要で、Google Cloud API のキーを持っている必要がありますが、日本語にも対応しているという強みがあります。Google Cloud API のキーは JSON ファイルに保存して PsychoPy の設定ダイアログの「一般」タブにある **[GoogleCloud アプリ―ケーションキー]** に指定してください。

[文字変換する言語] は en-US や ja-JP のようにロケール ID で指定します (en-US はアメリカ英語、ja-JP は日本の日本語を意味しています)。 **[検出する語 $]** は特定の単語だけを検出して記録したい場合に、その単語を列挙します。バックエンドが Built-in の場合は red:100 とか green:80 といった具合に「指定された単語検出した」と判定する際の信頼度の下限をパーセントで指定することができます。

「データ」タブの **[出力ファイル形式]** は音声ファイルの形式を指定します。かなりたくさんの形式がリストアップされるのでどれにしたらよいか迷うかもしれませんが、問題 (後述) が生じないなら default のままでよいでしょう。 **[発話開始/終了時刻の記録]** をチェックすると、音量が基準以上/以下になった時刻を記録します。 **[無音期間のトリム]** をチェックすると、音量が基準以下の部分をファイルに出力しません。

「ハードウェア」の **[チャネル]** はステレオ、モノラルの選択 (初期値は auto で自動検出)、 **サンプリングレート (Hz)** は音質を指定します。 **[最大録音データサイズ (kb)]** は 1 回の録音で作成されるファイルサイズの上限を指定します。この上限を超えた後は自動的に録音が停止しますが、対応する Microphone オブジェクトの isRecBufferFull() というメソッドで上限に達したかどうかを調べることができます (上限に達したら True が返される)。

録音が成功するとデータファイルの保存フォルダに「データファイル名に_mic_recorded とついたフォルダ (例えば foo_expn_2022-09-01-_20h00.00.000.csv というデータファイル名なら foo_expn_2022-09-01-_20h00.00.000_mic_recorded)」が作成され、その中に音声ファイルが出力されます。ループで繰り返し実行すると、繰り返し毎にファイルが作成されます。

以上が Microphone コンポーネントの概要ですが、最初に書いたとおり 2022.2.4 の時点で Microphone コンポーネントにはバグがあり、使用にはかなり制限があります。具体的には、 **[終了]** を空欄にすると録音自体できません。また、 **[終了]** で指定した条件を満たしたあともルーチンが継続した場合、エラーが生じて実験が停止します。どちらも致命的な問題ですが、とりあえず

- **[終了]** を空欄にせずに終了時間を指定し、他のコンポーネントの終了時間がそれより長くならないようにする

ことを守れば使用できるはずです。PsychoPy が出力する実験の構造に詳しい方向けに書くと、

- Microphone コンポーネントの終了条件を入力しつつ、なおかつその終了条件を満たして終了コードが実行される前に他の方法でルーチンを終了すること

が条件です。例を挙げましょう。新たに実験を作成して、以下のように作業してください。

- 実験設定ダイアログ

  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。

- trial ルーチン
  - Keyboard コンポーネントをひとつ配置して、以下の通り設定する。
    * **[終了]** に 5 と入力する。
    * **[Routine を終了]** にチェックが入っていることを確認する。
  - Text コンポーネントをひとつ配置して、以下の通り設定する。
    * **[終了]** に 5 と入力する。
    * **[文字列]** に $int(t) と入力し、「フレーム毎に更新」にする。
  - Microphone コンポーネントをひとつ配置して以下の通り設定する。
    * **[終了]** に 5 と入力する。
    * 「音声文字変換」タブの **[音声の文字変換]** がチェックされていないことを確認する。

できたら実行してみましょう。キーを押さなければ 5 秒間録音されて実験が終了します。画面上のカウントが 5 になる前にキーを押して終了しても、きちんと中断するまでの音が録音されるはずです。無制限に反応を待たなければならない実験でなければ、 **[終了]** に十分長い時間を指定することで何とか使い物になるのではないかと思います。

うまく動作しない場合、まずシステムのオーディオ設定で録音可能なデバイスが確かに存在すること、アプリケーションから使用できるように設定されていることを確認してください。セキュリティの設定が厳しい場合、録音デバイスが存在しても任意のプログラムから使用できないようになっている可能性があります。マイクを使用する他のアプリを起動して動作することも確認するとよいでしょう。

Builder から録音デバイスが認識されていて、実験が正常に動作しているように見えるにも関わらず音声ファイルが出力されない場合は、「ハードウェア」の **[チャネル]** 、 **サンプリングレート (Hz)** に問題がある可能性があります。モノラルマイクを使っているなら **[チャネル]** を auto のままにせず mono に変更する、デバイスが対応しているサンプリングレートを調べて **サンプリングレート (Hz)** の設定を合わせるなどすると録音できるようになる場合があります。

チェックリスト

- Microphone コンポーネントで録音を行うことができる。

10.7 Camera コンポーネントで動画撮影してみよう

録音に続いて動画撮影を取り上げます。使用するのはコンポーネントペインの「反応」カテゴリにある Camera コンポーネントです (図 10.8)。このコンポーネントは Microphone コンポーネントのようなバグがあるわけではありませんが、動画というもの自体の扱いの難しさのためこちらもなかなか一筋縄ではいきません。

Camera コンポーネントの独自のプロパティとしては、まず「基本」タブの **[ビデオデバイス]** があります。Microphone コンポーネントではデバイスと記録フォーマットを別々に設定しましたが、Movie コンポーネントでは **[ビデオデバイス]** に使用するデバイスと記録フォーマットが組み合わされた形でリストアップされま

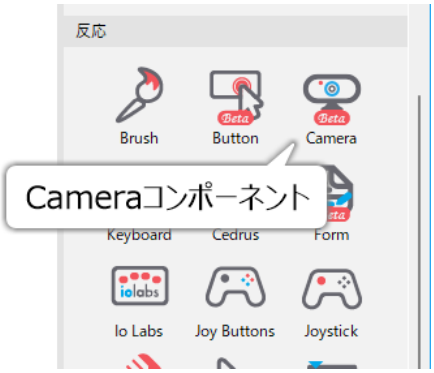


図 10.8　Camera コンポーネントのアイコン

す。例えば [HD Pro Webcam C920] 800x600@30fps, h264 と表示されていれば、[HD Pro Webcam C920] がデバイス名、800x600 は動画の解像度が幅 800 ×高さ 600、@30fps は秒間フレームが 30、h264 が動画のフォーマットです。かなりの種類の組み合わせが表示されると思いますが、残念ながらすべての組み合わせが利用可能というわけではありません。各自の環境でどの組み合わせなら利用できるか試行錯誤する必要があります。「基本」タブには他に **[オーディオデバイス]** という項目もありますが、2022.2.4 の時点でこの項目は default から変更できません。

「データ」タブには **[ファイルへ保存]** という項目があります。これをチェックしていたら動画がファイルに保存されます。Microphone デバイスと同様、データファイルの保存フォルダに「データファイル名に _cam_recorded をつけた名前のフォルダ」が作成され、その中に保存されます。Camera コンポーネント独自の項目は以上なので、さっそくサンプルを作ってみましょう。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
  - 安定して使用できる **[ビデオデバイス]** の選択肢が判明するまでは **[フルスクリーンウィンドウ]** のチェックを外しておく。
- trial ルーチン
  - Keyboard コンポーネントをひとつ配置して、以下の通り設定する。
    - **[終了]** を空欄にする。
    - **[Routine を終了]** にチェックが入っていることを確認する。
  - Text コンポーネントをひとつ配置して、以下の通り設定する。
    - **[終了]** を空欄にする。
    - **[文字列]** に $int(t) と入力し、「フレーム毎に更新」にする。
  - Camera コンポーネントをひとつ配置して以下の通り設定する。
    - **[終了]** を空欄にする。
    - **[ビデオデバイス]** を自分の環境に合わせて設定する。どれにしたらいいかわからない場合、320x240 などの低めの解像度で 30fps 以下のものから試してみるとよい。また、カメラのカタ

ログ等で「h264 対応」などと書いてある場合はそのフォーマットを優先的に試すのもよいだろう。

完成したら実行してみましょう。実験が始まった直後に終了してしまって、Runner の標準出力に Specified camera format is not supported. といったエラーメッセージが出る場合は **[ビデオデバイス]** の設定を変更して実行する作業を繰り返して、安定的に動作する選択肢を探してください。エラーで停止した時の復帰をしやすくするため、安定的に動作する選択肢が見つかるまでは **[フルスクリーンウィンドウ]** のチェックを外しておくのがお勧めです。

無事に実験が動作したら、カメラの前で手を叩いて「ぱん！」と鳴らすなど、「動きと同期して音が鳴る」様子を記録してからキーボードのスペースキーなどを押して実験を終了し、動画ファイルが保存されていることを確認しましょう。動画ファイルが出力されていない場合や、出力されていても通常の動画プレイヤーで再生できない場合は **[ビデオデバイス]** の設定が適切ではないので、面倒ですが選択肢探しの作業に戻ってください。

無事に動画が保存されていて再生できた場合は、映像と音のずれ具合を確認してください。ずれの大きさはカメラや PC の性能に依存しますが、筆者がこの原稿の執筆時に使用した環境 (Windows11 x64/PsychoPy2022.2.4/LogiCool C922) では 0.8 秒ほど映像が遅れます (音が先に鳴る)。皆さんの PC でどの程度ずれるかはわかりませんが、いずれにしても時間と映像の精確な同期が必要な用途には向いていないということです。Camera コンポーネントを実験に使用する場合は、このずれのことをよく頭においておいてください。

さて、時間的なずれにはもうひとつ、画面上に提示した刺激と動画のタイミングのずれもあります。これを確認するために、カメラを PC 画面に向けて実験を実行し、画面上に表示されているカウントアップの数字を録画してみましょう。理想的には録画が始まると同時に画面上では 0 が表示されていて、ちょうど 1 秒再生したところで数字が 1 になるはずです。コマ送りできるプレイヤーを使っている場合は、数字が 1 増える瞬間のコマから 1 フレームずつコマ送りして、録画時に設定した fps のコマ数 (例えば 30fps なら 30 コマ) だけ進めたタイミングで数字が増えるかどうかも確認しましょう。筆者の環境の場合、動画の最初から数えて 0 が表示されるまで約 23 コマの遅延がありました。30fps で 23 コマの遅延ということは 0.76 秒の遅延ですから、先ほど手を叩く動画で音に対する映像の遅延とほぼ一致しています。数字が増えるタイミングは録画時の fps(30) と一致していましたが、実行後に Runner に表示されている出力をよく見ると real-time buffer too full or near too full! frame dropped!といった警告が度々出力されていたので、時々フレーム落ちしていたものと思われます。皆さんの環境ではどのような結果になるかわかりませんが、本番の実験に使う前にこういった「ずれ」をしっかり検討しておくことをお勧めします。

続いて、もうひとつサンプルを紹介しましょう。次のサンプルでは動画保存をおこなうのではなく、画面上にカメラの映像をリアルタイムに表示してみます。以下のように作業してください。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
- trial ルーチン
  - Camera コンポーネントをひとつ配置して以下の通り設定する。
    - **[名前]** が cam であることを確認する。
    - **[終了]** を 10 にする。
    - **[ビデオデバイス]** を自分の環境に合わせて設定する。

    * 「データ」タブの **[ファイルへ保存]** のチェックを外す。

  - Code コンポーネントをひとつ配置する。
  - Image コンポーネントをひとつ配置して以下の通り設定する。
    * **[終了]** を 10 にする。
    * **[画像]** に $frame_img と入力し、「フレーム毎に更新」に設定する。
    * **[サイズ [w, h] $]** に **[ビデオデバイス]** で設定したカメラ映像の縦横比に合わせて適当な値を入力する。例えば 1920x1080 のように 16:9 の映像なら (0.64,0.36)、640x480 のように 4:3 の映像なら (0.64,0.48) といった具合である。
    * **[垂直に反転]** にチェックを入れる。

Code コンポーネントが Image コンポーネントより先に実行されるように並んでいる (つまりルーチンペイン上で Code コンポーネントの方が上にある) ことを確認したうえで、Code コンポーネントの「フレーム毎」の Python のコード欄に以下のように入力する。

```
frame = cam.getVideoFrame()
if frame.colorData is not None:
    frame_img = frame.colorData.astype(np.float).reshape(
        (frame.size[1],frame.size[0],3))
    frame_img /= 256
else:
    frame_img = np.ones((16,16,3),dtype=np.float)
```

入力したら実行してみよう。うまくいけば 10 秒間カメラの映像が画面上に描かれるはずです。動画が保存されない点も確認しておいてください。Camera コンポーネントと Image コンポーネントの使い方には特に難しいところはないでしょうから、ここでは Code コンポーネントのコードを集中的に解説します。

1 行目の cam.getVideFrame() というのは cam、つまり Camera コンポーネントによって作成された Camera オブジェクトの getVideFrame() というメソッドの呼び出しです。このメソッドは呼び出された時点で最新のビデオフレームを MovieFrame オブジェクトとして返します。MovieFrame オブジェクトは単なる画像データではなくタイムコードなど動画のさまざまな情報を含んでいて、画像データは colorData というデータ属性に格納されています。ただ厄介なことに、これはそのまま Image コンポーネントで表示できるような PsychoPy の画像データではなく、画像の各ピクセルの RGB 値をべたっと 1 次元に並べただけのものです (1920 × 1080 の解像度なら 1920 × 1080 × 3 = 6,220,800 個)。Image コンポーネントで使用するには、縦×横× 3 の 3 次元のデータで、なおかつ値が 0.0 から 1.0 の小数でなければいけません。

そこで今回のコードでは、まず画像データが得られていることを確認して (2 行目の if 文)、3 行目で astype() による小数型への変換と reshape() によるデータの 3 次元化を一気に行っています。続く 5 行目で RGB 値を 0.0 から 1.0 に変換していますが、ここでは変換前の RGB の各チャネルの数値が 0 から 255 の整数であることを前提にしています。一般的な web カメラならこれで問題ないはずです。7 行目は MovieFrame オブジェクトが画像を含んでいなかった場合のために 16 × 16 の解像度の真っ白の画像を作って代入しています。どうせ Image コンポーネントによって拡大されるのですから低解像度で十分です。これで変数 frame_img に Image コンポーネントで表示する画像データが作成されました。あとは Image コンポーネントで表示するだけです。

ひとつ注意が必要なのは Image コンポーネントの **[垂直に反転]** です。すべてのカメラでそうなるかわかりませんが、筆者が試した範囲では MovieFrame の画像データは PsychoPy の Imagae コンポーネントの画像データと比べると上下方向に反転しています。そこで正しい向きに表示するためにはデータ自体をさらに変換するか、この例のように **[垂直に反転]** をチェックする必要があります。もし鏡像のようにしたいのであれば、 **[水平に反転]** もチェックすると良いでしょう。

この例では Camera コンポーネントの映像をそのまま画面上に描画しましたが、リアルタイムに動画を解析して参加者のジェスチャーを検出したりできればぐっと応用範囲は広がるでしょう。現状の PC の性能ではフレームレートや遅延の観点から自然な速度で実行するのは難しいと思いますが、この分野の技術の進歩はとても速いので、近い将来に実用的になるかもしれません。

チェックリスト

- Camera コンポーネントを使って動画記録を行うことができる。
- Camera コンポーネントで記録された動画の遅延を確認することができる。
- Camera コンポーネントの映像をリアルタイムに画面上に表示することができる。

10.8 この章のトピックス

10.8.1 Static コンポーネントを用いた動画の読み込み

動画ファイルは一般的にファイルサイズが大きく、読み込みに時間がかかります。ループで繰り返しのたびに異なる動画を読み込むと、ルーチン開始前に読み込みを行いますので、ここで時間がかかると繰り返しのたびに意図しない空白画面が表示され続けることになります。ただ時間がかかるだけならまだしも、動画ファイルによって読み込みの時間が異なると試行間の間隔がばらばらになってしまって実験によっては望ましくありません。こういう時に便利なのが Static コンポーネントです。

Static コンポーネントはコンポーネントペインの「カスタム」の中に含まれる 図 10.9 のアイコンです。他のコンポーネントとは異なり、 **[名前]** の初期値がコンポーネント名と同じではなく ISI となっているので注意してください。 **[開始]** と **[終了]** は他のコンポーネントと同様、Static コンポーネントが有効な期間を指定します。

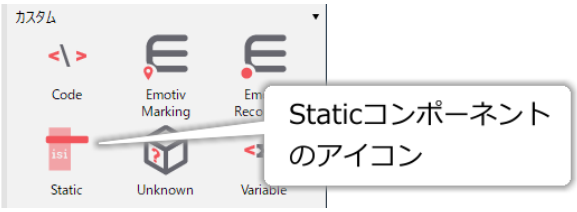


図 10.9　Staic コンポーネントのアイコン

プロパティ設定ダイアログの OK をクリックしてダイアログを閉じると、ルーチンペインに 図 10.10 上のように赤色の領域が現れます。これが Static コンポーネントです。削除するときはこの赤い領域内のどこかで右クリックしてメニューから「削除」を選んでください。

Static コンポーネントを配置した後に、他のコンポーネントのプロパティ設定ダイアログを開いた際に、各プロパティの更新方法として「更新方法: trial.ISI」のような項目が追加されます (図 10.10 下)。ここで trial は

Static コンポーネントを置いているルーチン名、ISI はスタティックコンポーネントの名前です。test というルーチンに load_stim という名前で Static コンポーネントを配置したなら test.load_stim となります。この項目を選択すると、プロパティの更新が指定した Static コンポーネントの期間中に行われます。例えば実験で使用する動画が最も読み込みに時間がかかるものでも 0.4 秒で完了するなら、Static コンポーネントの長さを (少し余裕をもって)0.5 秒にしておけば、どの動画も 0.5 間で読み込むことができてばらつきが生じません。Static コンポーネントで設定した期間内に終わらない処理を行わせると元も子もないので、あらかじめ動作確認して余裕を持たせておくことが重要です。

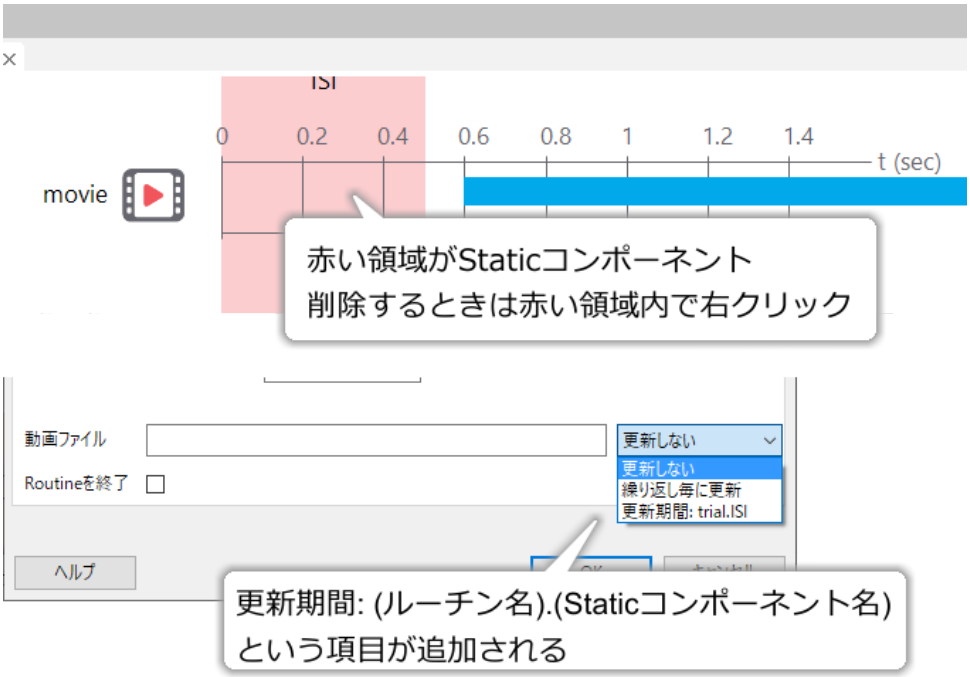


図 10.10　Static コンポーネントを設置すると他のコンポーネントの更新方法のところに項目が追加されます

「Static コンポーネント」の名前の通り、この期間には刺激を描画したりキー押しを検出したりするべきではありません。やってできない事はないのですが、時間的な精度が保障されなくなります。Static コンポーネントの期間中 (あるいは期間開始と同時に) に静止した刺激を描画しておくことには何の問題もありません。

ファイルの読み込みタイミングとして他のルーチンに配置した Static コンポーネントを選択することも可能なので、実験期間中で都合がよいタイミングにファイルを読み込んでくことが可能です。Static コンポーネントはファイルサイズが大きくなりがちな動画ファイルの読み込みに特に力を発揮しますが、音声ファイルを読み込む時や、画像ファイルを十数枚一気に読み込む必要がある時などにも役に立ちます。

「カスタム」タブの **[カスタムコード]** は、他のコンポーネントのパラメータの更新以外の作業を行わせたいときに使用します。入力欄が狭いので、複雑な処理を行わせる場合は Code コンポーネントで関数を定義してその関数を呼び出すなどの工夫が必要かもしれません。上級者向けの機能だと思います。

第 11 章

階段法の手続の実験を作成しよう

11.1 この章の目的

この章では「3.5: 繰り返しを設定しよう」で解説を省略した **[Loop の種類]** の staircase と interleaved staircases の使い方を解説します。これらのループはいずれも staircase 法 (階段法) と呼ばれる手続きを Builder で実現するために用意されています。まず通常の staircase から解説し、続いて interleaved staircases に触れます。

11.2 Staircase ループについて学ぼう

図 11.1 に staircase ループのプロパティ設定ダイアログを示します。staircase ループでは条件ファイルを使用せず、このダイアログですべてのパラメータを設定します。

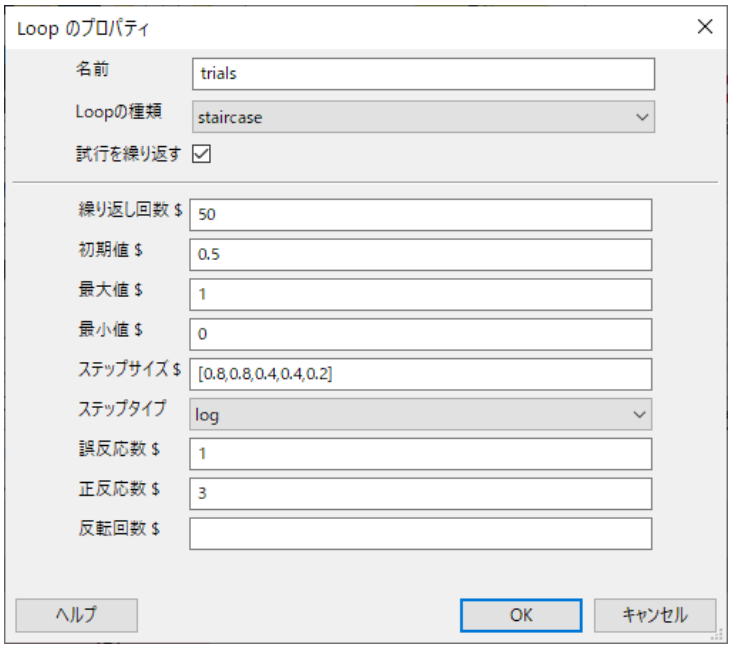


図 11.1　staircase ループのプロパティ設定ダイアログ

staircase 法は心理物理学的測定法の一つである極限法の効率を高めた方法です。参加者の反応は「正反応」（刺激が知覚できている）と、「負反応」（刺激が知覚できていない）の二種類に分類できるとします。極限法と同

様に、計測したい閾値から離れたところから刺激を提示し始めて、参加者の反応が変化するまで一定方向に刺激を変化させ続けます。極限法ならば最初に参加者の判断が変化したところで系列を終了しますが、staircase 法では今までと反対方向へ刺激を変化させて続行します。その後は正反応、負反応が一定数連続する度に刺激を変化させる方向を反転させます。

図 11.2 は、「正反応が 3 回連続する」または「負反応が 1 回出現する」時に反転させるというルールで staircase 法を実施した例を示しています。閾値付近で何度も刺激の変化方向が反転して、閾値付近での参加者の判断を効率的に記録できることがわかります。

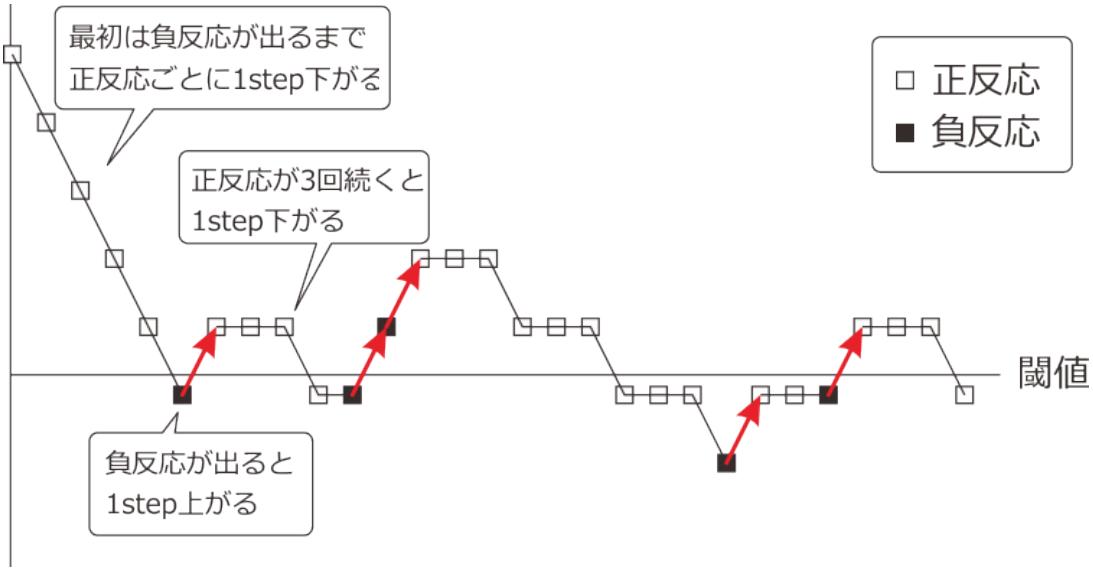


図 11.2　staircase 法の実行例

以上の例を踏まえて、staircase ループのプロパティについて解説します。まず、試行開始時の初期値と反転のルールを決定するパラメータを 表 11.1 に示します。この表で「刺激レベル」と書かれているのは 図 11.2 の縦軸にあたる値で、Builder 内では level という内部変数で参照することができます。例えば、staircase ループで Polygon コンポーネントの塗りつぶし色 (RGB 値) の明るさを変化させたい場合は、 **[塗りつぶしの色]** を $[level, level, level] とします。

表 11.1　初期値と反転のルールを決定するパラメータ

[初期値 $]	開始時の刺激レベルの値を指定します。浮動小数点数でなければいけません。
[誤反応数 $]	正の整数を指定します。負反応がここに定められた回数連続すると刺激レベルが減少します。
[正反応数 $]	正の整数を指定します。正反応がここに定められた回数連続すると刺激レベルが増加します。
[最小値 $]	刺激レベルの最小値を浮動小数点数で指定します。例えば刺激の RGB 値を変化させる場合、値が-1.0 より小さくなると正常に表示されませんが、そのようなときにこのパラメータで下限の値を定めます。
[最大値 $]	刺激レベルの最大値を浮動小数点数で指定します。例えば刺激の RGB 値を変化させる場合、値が 1.0 より小さくなると正常に表示されませんが、そのようなときにこのパラメータで下限の値を定めます。

刺激を変化させる量を決めるには、 表 11.2 のパラメータを使用します。

表 11.2　刺激の変化量を決定するパラメータ

[ステップサイズ $]	刺激レベルの変化量を浮動小数点数で指定します。リストが指定された場合、最初はリストの 0 番目の要素の値が用いられ、変化方向が反転する度にリストの次の要素の値が新たな変化量となります。
[ステップタイプ]	lin、log、db のいずれかを選びます。

[ステップタイプ] は、 **[ステップサイズ $]** と組み合わせて使用します。

lin　**[ステップサイズ $]** に入力された値がそのまま現在の刺激レベルに加減算されます。

log　現在の刺激レベルが l、ステップサイズが s とします。刺激レベルを増加させるときには次の刺激レベル値を $l \times 10.0^s$ 、減少させるときには $l \div 10.0^s$ とします。

db　現在の刺激レベルが l、ステップサイズが s とします。刺激レベルを増加させるときには次の刺激レベル値を $l \times 10.0^{s/20.0}$ 、減少させるときには $l \div 10.0^{s/20.0}$ とします。

一連の手続きが終了する条件を決めるには、 表 11.3 のパラメータを使用します。

表 11.3　staircase 終了を決定するパラメータ

[反転回数 $]	終了までに最低限必要な反転回数を自然数で指定します。ただし、**[ステップサイズ $]** にリストが入力されていて、そのリストの長さが **[反転回数 $]** より長い場合は **[反転回数 $]** のリストの長さの回数反転するまで終了しません。
[繰り返し回数 $]	終了までに最低限必要な試行数 (参加者の判断回数) を指定します。この回数を過ぎても、 **[反転回数 $]** または **[ステップサイズ $]** によって定められた回数反転していなければ終了しません。

なお、以下の設定をすると、刺激レベルを 1.0 から 0.1 ずつ減少させていく通常の極限法 (下降系列) のようにも使えます。ただし、最初の刺激レベルがいきなり負反応の場合はうまくいきませんので十分に高い **[初期値 $]** を設定する必要があります。

- **[初期値 $]** = 1
- **[ステップサイズ $]** = 0.1
- **[ステップタイプ]** = lin
- **[誤反応数 $]** = 1
- **[正反応数 $]** = 1
- **[反転回数 $]** = 1
- **[繰り返し回数 $]** = 0

チェックリスト

- Builder で階段法の実験を作成することができる。

11.3 Interleaved staircases ループについて学ぼう

interleaved staircases ループは、複数の staircase を切り替えながら実行させる時に使用します。 図 11.3 に interleaved staircases ループのプロパティ設定ダイアログを示します。interleaved staircases ループでは、複数の staircase のパラメータを条件ファイルから読み込んで実行します。 条件ファイルは **[繰り返し条件]** に指定します。各 staircase を実行する順番は **[切り替え方法]** から random、sequential、fullRandom の中から選びます。これは **[Loop の種類]** の random、sequential、fullRandom と同じ意味です。 **[繰り返し回数 $]** には各 staircase での最小試行数を指定します。

[ステアの種類] では、各 staircase の刺激レベル変更方式を simple, QUEST, questplus のいずれかから選択することができます。大文字と小文字の QUEST がありますがどちらも同じです (1.82.02 で確認)。simple は前節で解説した staircase と同じです。QUEST はいくつかの仮定の下で効率的に閾値を探索するアルゴリズムです。詳しくは Watson & Pelli (1983). QUEST: A Bayesian adaptive psychometric method. *Perception & Psychophysics*, 33(2), 113-120 などを参照にしてください。questplus は Watson (2017). QUEST+: A general multidimensional Bayesian adaptive psychometric method. Journal of Vision, 17(3):10. doi: 10.1167/17.3.10. で提案された QUEST+ アルゴリズムに対応するものですが、ここでは解説を省略します。

interleaved staircases の条件ファイルの例を 図 11.3 に示します。条件ファイルは必ず以下の列を含んでいなければいけません。

label データの出力時に各 staircase を区別するために用いられるラベル。すべての行で異なっている必要があります。

startVal 各 staircase 開始時の刺激レベルの値です。

startValSd (QUEST のみ) 閾値の推定値の標準偏差の初期値です。

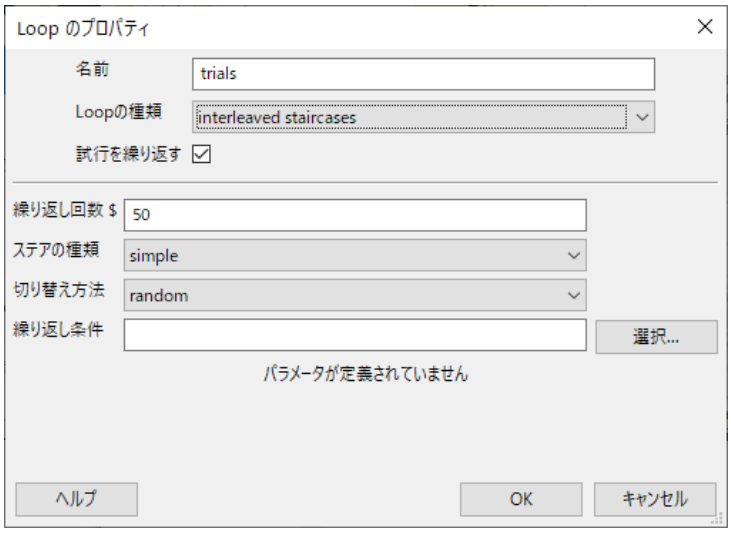
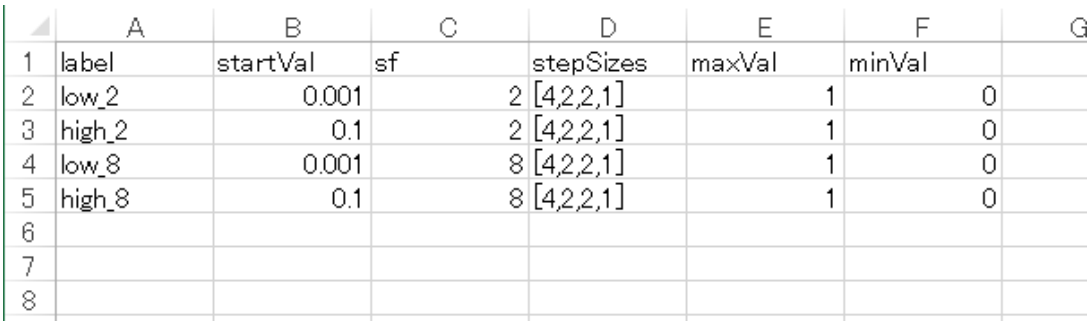


図 11.3　interleaved staircase ループのプロパティ設定ダイアログ

上記以外の列の値は、Builder の変数として読み込まれます。例えば 図 11.3 には sf という列がありますが、この列の値を Grating コンポーネントの **[空間周波数 $]** に指定したい場合は、単に **[空間周波数 $]** に sf と書けばよいということです。 表 11.4 に示した名前を持つ列は、staircase のパラメータとして利用されます。各パラメータの意味は PsychoPy 公式サイトの API をご覧ください (StairHandler および QuestHandler)。



	A	B	C	D	E	F	G
1	label	startVal	sf	stepSizes	maxVal	minVal	
2	low_2	0.001	2	[4,2,2,1]	1	0	
3	high_2	0.1	2	[4,2,2,1]	1	0	
4	low_8	0.001	8	[4,2,2,1]	1	0	
5	high_8	0.1	8	[4,2,2,1]	1	0	
6							
7							
8							

図 11.4　interleaved staircase ループのプロパティ設定ダイアログ

表 11.4　staircase のパラメータとして解釈される変数名

simple の場合	nReversals, stepSizes, nTrials, nUp, nDown, extraInfo, stepType, minVal, maxVal
QUEST の場合	pThreshold, nTrials, stopInterval, method, stepType, beta, delta, gamma, grain, range, extraInfo, minVal, maxVal, staircase

チェックリスト

- Builder で interleaved staircases の手続の実験を作成することができる。

第 12 章

実験の流れを制御しよう—強化スケジュール

12.1 この章の実験の概要

ある行動に随伴して生じる出来事によって、その行動の発生頻度が高まることを強化と呼び、行動に随伴して生じる出来事を強化子と呼びます。強化子が毎回生じるか、一定間隔で生じるかといった強化子の出現スケジュールの違いによって行動の発生頻度の変化パターンが異なることが知られています。強化子の出現スケジュールのことを強化スケジュールと呼びます。図 12.1 は基本的な強化スケジュールを示しています。ポイントは「強化子が生じるタイミングが時間によって決まっているか、行動回数によって決まっているか」と、「時間や回数が一定であるか、変動するか」です。これらの組み合わせで Fixed Interval (FI)、Fixed Ratio (FR)、Variable Interval (VI)、Variable Response (VR) の 4 種類のスケジュールができます。FI、VI において強化子が得られるまでに必要な時間を「強化時間」、FR、VR において強化子が得られるまでに必要な反応回数を「強化回数」と呼ぶことにします。FI、VI では強化時間が経過したら自動的に強化子が得られるのではなく、強化時間が経過した後に行われた最初の行動によって得られる点に注意してください。

この章では、Code コンポーネントを活用してこれらの強化スケジュールを Builder で実現します。 図 12.2 に実験の手続きを示します。実験参加者の課題は、キーボードのスペースキーを押してできるだけ早く指定された点数の「得点」を獲得することです (例えば 20 点)。スクリーンには実験が開始してからの時刻と現在の得点が表示されていて、参加者は強化スケジュールで定められた条件を満たした状態でスペースキーを押すと、1 点を獲得することができます。得点を獲得すると同時に 2000Hz の音が 0.2 秒間鳴り、スクリーン上の時刻と得点の表示の背後に赤い長方形が 1 秒間表示されます。指定された得点に到達したら、その時点で実験は終了します。

今回の実験では単位として norm を用いて、経過時刻および得点の表示用テキストの大きさ (**[文字の高さ $]**) は 0.1、経過時刻の位置は [0, 0]、得点の位置は [0, -0.2] とします。また、得点獲得時の赤い長方形は大きさ [0.5, 0.5]、位置 [0, -0.1] とします。

得点を獲得できる条件として、FI、FR、VI、VR の 4 種類のスケジュールを実行時に選択するようにします。いろいろな作成方法が考えられますが、ここでは「FI は強化時間が 1 種類しかない VI」と考えてみましょう。そうすると、FI は VI 用の実験に条件が 1 種類しか定義されていない条件ファイルを与えることで実現できます。 図 12.2 の 1. のように実験開始時の実験情報ダイアログに条件ファイルを指定する condition という項目と、各条件の繰り返し回数を指定する nReps という項目を用意しておいて、「条件 1 種類の条件ファイルを

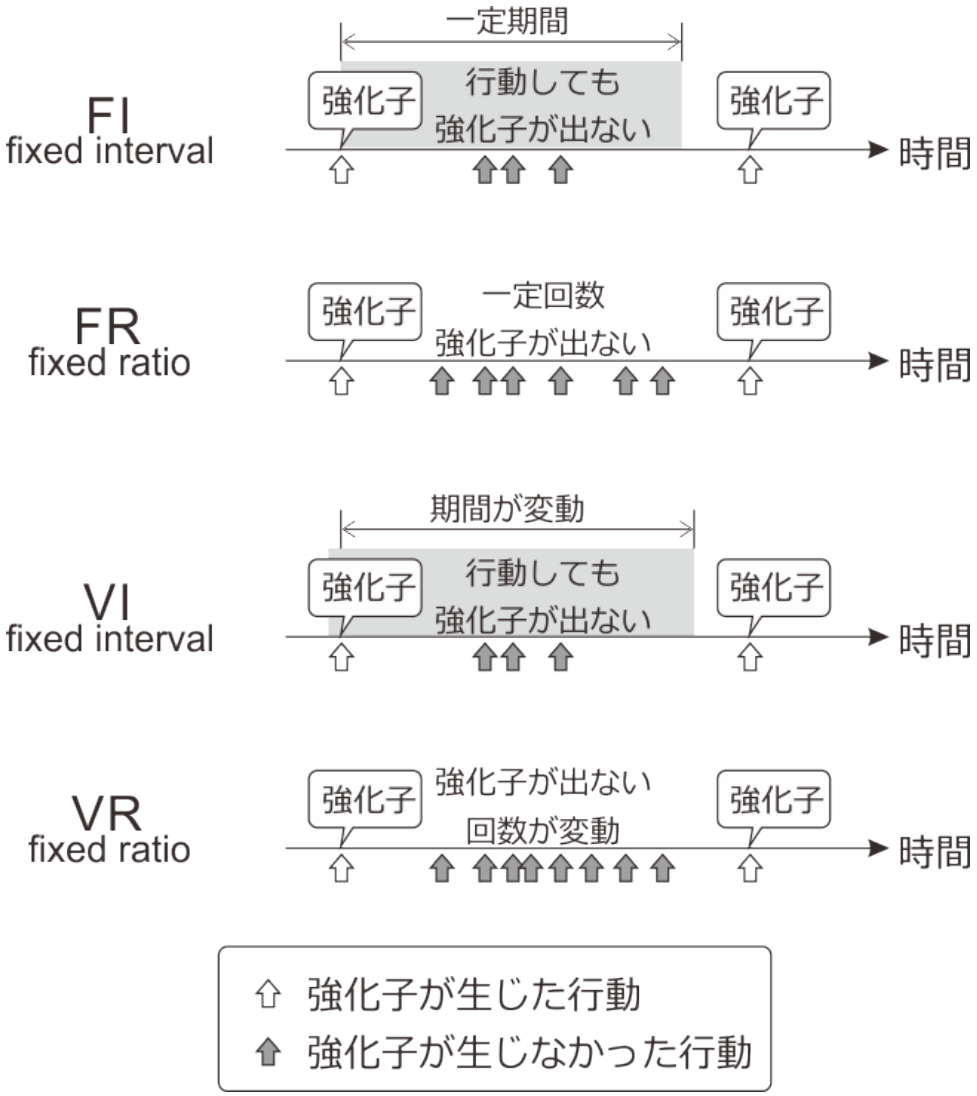


図 12.1　基本的な強化スケジュール。

condition に指定して、nReps を 20 に」すれば 20 点獲得で終了する FI です。「条件 5 種類の条件ファイルを condition に指定して、nReps を 4 に」すれば、5 × 4=20 点獲得で終了する VI です。同様に、「FR は強化回数が 1 種類しかない VR」と考えれば、VR 用の実験を用意すれば FR に対応できます。結論として、VI 用と VR 用の実験を作成するだけで 4 種類のスケジュールに対応できます。

以上が実験の概要です。この章では、以上の実験を土台として、一定時間経過したら実験が終了したり、参加者の反応によって次の課題が変化したりといった高度な実験の流れの制御を学びたいと思います。

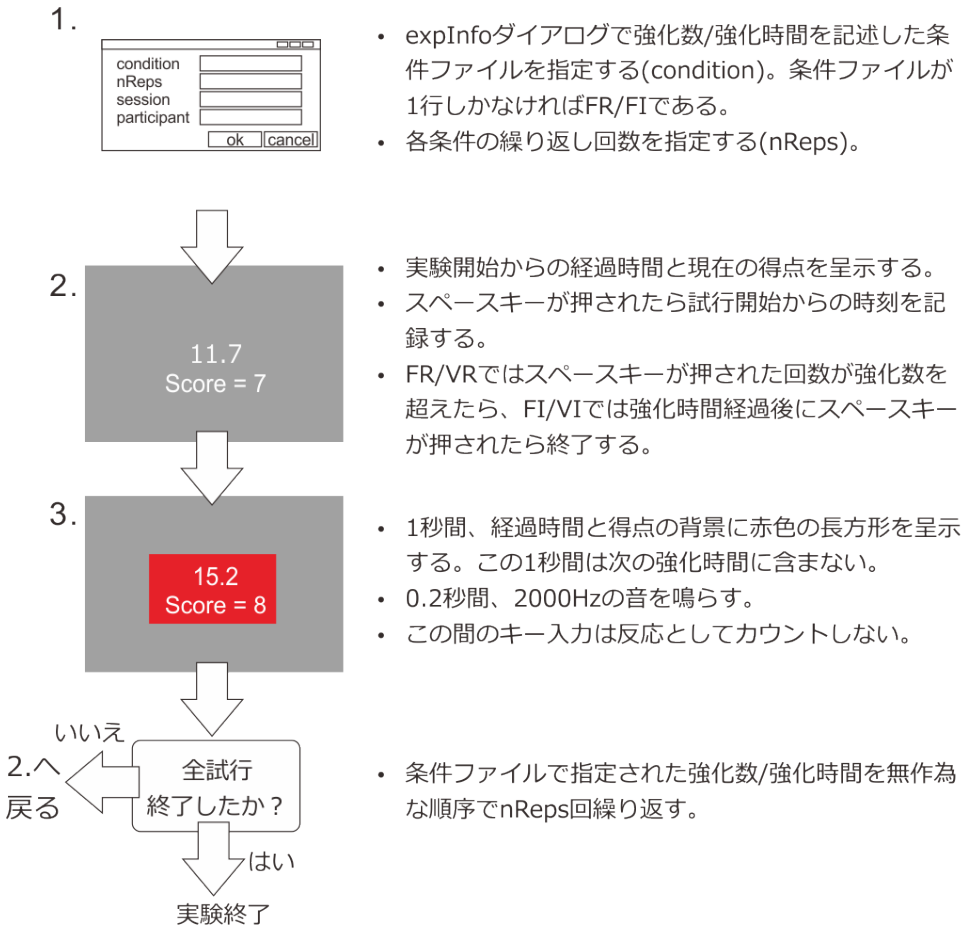


図 12.2　実験の手続き。

12.2 FI/VI 実験の作成

それでは実験の作成に入りましょう。まずは FI/VI 用の実験から作成します。以下の解説では、Builder で新規に実験を作成して以下の作業を行い、exp12vi.psyexp という名前で保存したものとします。

- 実験設定ダイアログ
  - **[実験情報ダイアログ]** に nReps と condition という項目を追加する。
  - **[単位]** を norm にする。
- trial ルーチン
  - Text コンポーネントをふたつ配置して、それぞれ **[名前]** を scoreTrial、clockTrial にする。両方とも **[終了]** を空白にする。
    - scoreTrial の **[位置 [x, y] $]** を [0.0, -0.2] にする。 **[文字列]** に$'Score:'+str(score) と入力し、「繰り返し毎に更新」に設定する。
    - clockTrial の **[文字列]** に----と入力しておく。
  - Keyboard コンポーネントをひとつ配置して、以下のように設定する。
    - **[名前]** を key_resp_Trial にする。

    * **[終了]** を空白にする。
    * **[Routine を終了]** のチェックを外す。
    * **[検出するキー $]** に'space' と入力する。
    * **[記録]** を「全てのキー」にする。
  - Code コンポーネントをひとつ配置し、 **[名前]** を codeTrial にする。
    * **[実験開始時]** に score=0 と入力する。
    * **[Routine 終了時]** に score+=1 と入力する。
- reinforcement ルーチン (作成する)
  - フローの trial ルーチンの直後に挿入する。
  - polygon コンポーネントをひとつ配置して、以下のように設定する。
    * **[形状]** 長方形にする。
    * **[名前]** を backgroundRect にする。
    * **[終了]** が「実行時間 (秒)」で 1.0(=初期値) であることを確認する。
    * **[塗りつぶしの色]** 、 **[枠線の色]** を red にする。
    * **[位置 [x, y] $]** を [0.0, -0.1] にする。
  - Text コンポーネントをふたつ配置して、それぞれ **[名前]** を scoreRF、clockRF にする。両方とも **[終了]** が「実行時間 (秒)」で 1.0(=初期値) であることを確認する。また、両方とも backgroundRect より上に描画されるようにルーチンペイン上の順序に配慮する。
    * scoreRF の **[位置 [x, y] $]** を [0.0, -0.2] にする。 **[文字列]** に$'Score:'+str(score) と入力し、「繰り返し毎に更新」に設定する。
    * clockRF の **[文字列]** に----と入力しておく。
  - Sound コンポーネントをひとつ配置して、以下のように設定する。
    * **[名前]** を SoundRF にする。
    * **[終了]** を「実行時間 (秒)」の 0.1 にする。
    * **[音]** を 2000 に、 **[ボリューム $]** を 1 にする。
- trials ループ (作成する)
  - trial ルーチンと reinforcement ルーチンを繰り返すように挿入する。
  - **[繰り返し回数 $]** に expInfo['nReps'] と入力する。
  - **[繰り返し条件]** に$expInfo['condition'] と入力する。
- exp12fi.xlsx(条件ファイル)

  - interval というパラメータを設定し、値として 10 を入力する。パラメータ名の行を除いて 1 行 1 列の条件ファイルとなる。

- exp12vi.xlsx(条件ファイル)
  - interval というパラメータを設定し、値として 6, 8, 10, 12, 14 を入力する。パラメータ名の行を除いて 5 行 1 列の条件ファイルとなる。

以上の作業に加えて、trial ルーチンの codeTrial の **[フレーム毎]** に以下のコードを入力してください。

```
if len(key_resp_Trial.rt) > 0 and key_resp_Trial.rt[-1] >= interval:
    continueRoutine = False
```

解説済みのテクニックで作成できる部分は以上です。実はこの時点ですでに FI/VI の実験として使用できる状態まで完成していますので、一度実行してみましょう。実験開始からの時刻が----と表示される以外は、計画通り動作するはずです。条件ファイルに exp12fi.xlsx を指定すれば強化時間 10 秒の FI スケジュールとなりますし、条件ファイルに exp12vi.xlsx を指定すれば強化時間が 6 秒から 14 秒まで変化する VI スケジュールとなります。実験を実行すると作成される記録ファイルの例を 図 12.3 に示します。内容を確認すると、key_resp_Trial.rt という列に、実験参加者がキーを押した時刻がルーチン開始時を 0 秒として記録されています。このデータを利用すると、実験参加者のキー押し反応の発生頻度がどのような時間経過をたどって変化したかを分析することが可能です。

	A	B	C	D	E	F	G
1	interval	trials.thisRe	trials.thisTr	trials.thisN	trials.thisIn	key_resp_Trial.keys	key_resp_Trial.rt
2	10	0	0	0	0	['space', 'space', 'space', 'space	[0.8961341477988753, 1.76339087311
3	10	1	0	1	0	['space', 'space', 'space', 'space	[1.7124165177592658, 1.87905994170
4	10	2	0	2	0	['space', 'space', 'space']	[3.5616040696731943, 8.62790866069
5	10	3	0	3	0	['space', 'space', 'space']	[3.145897761645756, 6.212157910544

図 12.3　trial-by-trial 記録ファイルを確認すると、key_resp_Trial.rt にキーを押した時刻がすべて記録されています。このデータを用いて反応の頻度がどのように変化していくかを分析することができます。

ここまでで使用しているテクニックはすでに解説済みであることはすでに述べましたが、Code コンポーネントの働きが今一つピンとこない人が居るかも知れませんので、念のため補足しておきましょう。復習のつもりで読んでください。まず、 **[実験開始時]** で得点を保持する変数 score の値を 0 に初期化しています。そして、**[フレーム毎]** では以下の if 文を実行しています。

```
if len(key_resp_Trial.rt) > 0 and key_resp_Trial.rt[-1] >= interval:
    continueRoutine = False
```

この if 文は、実験参加者の最新のキー押し反応が強化時間経過後に押されていればルーチンを終了するという動作を実現しています。詳しく見ていきましょう。まず、and 演算子の左側の len(key_resp_Trial.rt)>0 という式ですが、key_resp_Trial.rt はキーが押された時刻を格納しているデータ属性でした (第 6 章)。len( ) はシーケンス型の要素数を返す関数ですから (第 8 章)、一回でも反応があってキー押しが記録されていれば and 演算子の左辺の式は True になります。続いて and 演算子の右辺を見ましょう。右辺は左辺の式が True でなければ評価されませんので (第 8 章)、必ず 1 個以上の要素が key_resp_Trial.rt に含まれています。ですから、

key_resp_Trial.rt[-1] が必ず存在しています。interval は条件ファイルで定義されている変数ですから、右辺の式は記録済みの最後のキー押し反応の時刻が interval の値以上である時に True となります。左辺、右辺の両方の式が True であれば、continueRoutine を False にしてルーチンを終了します。このような方法を採ることによって、ルーチンで行われたすべてのキー押し反応の時刻を記録しつつ、強化時間を過ぎて反応が起こったら直ちにルーチンを終了することが可能となります。そして、ルーチンが終了したということは得点を得たという事ですから、 **[Routine 終了時]** で score += 1 を実行するわけです。解説を読んでもよくわからなかった方は、第 6 章 から 第 8 章 にかけてしっかりと復習してください。

12.3 Global Clock を用いて実験開始からの経過時間を得よう

すでに前節で FI/VI スケジュールの実験が完成してしまって、この章は一体この後何を学ぶの？と疑問に思われているかたも多いかもしれません。この章の次の目標は、一定時間が経過したら実験を中断するという動作を実装することです。そのための布石として、まずは実験開始からの経過時間をスクリーン上に表示させてみましょう。

Builder には、実験を制御するためにいくつかの「時計」が用意されています。「時計」には実験全体の経過時間を計測するためのグローバルクロックと、各ルーチンが始まってからの経過時間を計測するためのルーチンクロックがあります。これらの時計の実体は、psychopy.core.Clock というクラスのインスタンスで、グローバルクロックは globalClock という変数に格納されています。ルーチンクロックは"ルーチン名 +Clock"という名前の変数に格納されています。ルーチン名が trial なら、ルーチンクロックは trialClock です。

psychopy.core.Clock は今まで紹介してきた PsychoPy のクラスと同様にいくつかのデータ属性とメソッドを持ちますが、クロックの働きを変更するメソッド等を実行すると Builder で作成した実験の動作に悪い影響が出る恐れがありますので、ここでは現在の時間を得るメソッド getTime( ) だけを紹介しておきます。getTime( ) メソッドは、現在の時間を小数で返します。時間の単位は秒です。globalClock.getTime( ) とすれば、実験開始からの経過時間が得られますし、trialClock.getTime( ) とすると trial ルーチン開始からの経過時間が得られます。なお、第 5 章 において、ルーチン開始からの経過時刻を保持する t という内部変数が出てきましたが、実はこの t はルーチン実行中に各フレームの処理の最初で t = trialCock.getTime( ) という具合にルーチンクロックの getTime( ) メソッドの戻り値を格納することで得られています。

さて、実験開始からの経過時間を得る方法がわかりましたので、さっそく exp12vi.psyexp を改造して経過時間をスクリーン上に標示してみましょう。exp12vi.psyexp を開いて、trial ルーチンの clockTrial のプロパティ設定ダイアログを開いて、 **[文字列]** に$globalClock.getTime() と入力して「フレーム毎に更新する」に設定しましょう。同様に、reinforcement ルーチンの clockRF の **[文字列]** にも$globalClock.getTime() と入力して「フレーム毎に更新する」に設定しましょう。変更が終わったら保存して、実行してください。確かに先ほどまで----と表示されていた部分に経過時刻が表示されていますが、 図 12.4 に示すように小数点以下の桁数がやたらと多くて非常に見難くなってしまいました。小数点以下はせいぜい 1 桁もあれば充分でしょう。次の節では、小数点 1 桁までの表示を実現する方法を解説します。

チェックリスト

- 実験が開始してからの経過時間を得るコードを書くことができる。

図 12.4　現在の経過時間をそのまま描画すると、小数点以下の桁数が多すぎて非常に見難くなります。

12.4 文字列の format() メソッドを使って経過時間の表示を整えよう

前節で問題となった経過時間の表示ですが、これは Python が小数を文字列として表示する際に標準の動作としてあのように小数点以下の桁を表示してしまうことが原因で生じます。Python にはプログラマが小数点以下何桁程度を表示したいと考えているかを知る方法がありませんので、Python の標準動作が気に入らない場合はプログラマが明示的に表示方法を指定してやる必要があります。ここで重要な役割を果たすのが文字列に対する format() メソッドです。format() は数値や文字列等のデータを引数にとり、引数の値を文字列へ指定された書式で埋め込みます。以下に例を挙げます。

```
'平均反応時間:{} 正答率:{}'.format(mean_rt, n_correct)
```

format() は文字列オブジェクトのメソッドなので、このように文字列の後ろに.format() と続けて書くことができます。文字列の中に{}という部分が 2 か所あることに注目してください。format() は、引数として与えられた値を文字列中の{}の場所へ順番に埋め込んでいきます。いま、引数として与えている変数 mean_rt の値が 1218.7、n_correct の値が 31 ならば、このメソッドの実行結果は

```
'平均反応時間:1218.7 正答率:31'
```

という文字列になります。{0}や{1}のように{}の中に整数を書くと、何番目の引数と対応づけるかを指定できたり、{mean_rt}のように名前を書くと引数名で対応づけを指定できるなど、便利な指定法もありますが、Builder での実験作成で使う程度なら、{}の中に何も書かずに自動的に引数順に割り当てる方法を覚えておけば十分でしょう。

format() は引数が整数であろうがリストオブジェクトであろうが、自動的に文字列に変換して埋め込んでくれます。しかし、自動変換の結果がこちらの望み通りとは限りません。実際、前節で問題となった globalClock.getTime() の戻り値を埋め込んでみると、Builder に任せた時と同様に小数点以下の桁数が非常に多くなってしまいます。format 文のいいところは、文字列への変換する際の書式を柔軟に指定できる点です。以下の例では、Builder の内部変数 t を小数点以下 3 桁までで表すように指定しています (4 桁目で四捨五入されます)。

```
'実験開始から{:.3f}秒'.format(t)
```

ちょっと難しいですが、{}内の : が「これ以降は変換方式の指定」ということを表す記号だと思ってください。: の後ろの .3f というのが書式指定で、f は小数として出力することを指定してします。f の前の.3 が小数点以

下 3 桁を出力するという指定です。. の前に整数を書くと全体の桁数の指定となり、さらに + 記号をつけると符号つきで出力されます。文章で説明してもなかなかわかりにくいと思いますので、具体例を 図 12.5 に示します。

図 12.5　数値の書式指定の例。変数 p には 3.14159264359 という値が代入されているものとします。

書式指定には小数を指定する f の他にも、10 進数の整数を指定する d や、16 進数の整数を指定する x などがあります。小数を 10 進数整数や 16 進整数で出力しようとするとエラーで実験が停止してしまうのでご注意ください。小数の値を整数で出力する場合は以下のように int() を使って整数に変換するか、小数点以下の桁数として 0 を指定すればよいでしょう。

```
'実験開始から{:d}秒'.format(int(t))   # int で整数に変換
'実験開始から{:.0f}秒'.format(t)      # 小数点以下の桁数として 0 を指定
```

文字列を埋め込む場合は、特に変換の必要がないので{}だけでよいでしょう。以下の例では実験情報ダイアログの値 (＝文字列) を埋め込んでいます。

```
'あなたの参加者 ID は{}です'.format(expInfo['participant'])
```

{ や } の記号を出力したい場合は {{ や }} といった具合に 2 つ並べて書きます。

format() メソッドにはまだまだ様々な使い方があるのですが、とりあえずこのくらい覚えておけば多くの実験に対処できるはずです。今回の場合は実験開始からの経過時間を小数点 1 桁まで表示したいので、以下の式で目的を達成できます。

```
'%.1f'.format(globalClock.getTime())
```

exp12vi.psyexp を開いて、この式を trial ルーチンの clockTrial と reinforcement ルーチンの clockRF の **[文字列]** に入力してください。もちろん先頭に$を付け忘れないようにしてください。変更したら保存して実行してみましょう。経過時間の表示が小数点 1 桁までになったはずです。

この format() を使った文字列への値の埋め込みは、教示文や刺激文などを作る時にも便利です。今までは + 演算子と str( ) 関数を駆使して埋め込みを行っていましたが、埋め込む値が複数個ある場合は format() を使った方が簡潔に記述できますので、ぜひマスターしてください。

さて、これで 図 12.2 に示した実験手続のうち、FI/VI に関する実験が完成しました。ここからさらにステップアップする前に、exp12vi.psyexp を改造して FR/VR スケジュールの実験を作成しておきましょう。

チェックリスト

- 変数の値を文字列に埋め込むことができる。
- 浮動小数点数を文字列に埋め込む時に、小数点以下の桁数を指定することができる。
- 数値を文字列に埋め込む際に、指定した桁数の右寄せで埋め込むことができる。
- 数値を文字列に埋め込む際に、符号付きで埋め込むことができる。

12.5 FR/VR 実験を作成しよう

この節では、exp12vi.psyexp をベースにして、FR/VR スケジュールの実験を作成します。具体的には、exp12vi.psyexp を exp12vr.psyexp という別名で保存した後、exp12vr.psyexp に以下の変更を加えます。

- ratio というパラメータを定義する条件ファイル exp12fr.xlsx と exp12vr.xlsx を作成する。exp12fr.xlsx では ratio は 10 のみ、exp12vr.xlsx では ratio は 2、6、10、14、18 の 5 種類の値をとるようにする。
- trial ルーチンにおいて、ルーチン開始後に ratio の回数だけスペースキーを押したらルーチンを終了する。

これまでに解説してきたことだけでできる変更なので、腕試しをしたい人は以下のヒントを見ずに作業してみてください。

さて、以下は「まだちょっと自力では難しいかな」という方向けのヒントです。条件ファイルの作成は、exp12fi.xlsx と exp12vi.xlsx を少し修正するだけです。exp12fi.xlsx のパラメータ名 interval を ratio に書き換えれば exp12fr.xlsx になります。exp12vi.xlsx のパラメータ名 interval を ratio に書き換えて、値を 2、6、10、14、18 の 5 種類に修正すれば exp12vr.xlsx になります。

続いて exp12vr.psyexp の変更です。trial ルーチンの終了条件を変更するだけですので、変更点は trial ルーチンの Code コンポーネント (codeTrial) です。trial ルーチンが開始されてから実験参加者がスペースキーを押した時刻が key_resp_Trial.rt にリストとして格納されているのですから、このリストの長さが変数 ratio 以上になればルーチンを終了すればよいのです。例えば以下のようなコードを書けばよいでしょう。

```
if len(key_resp_Trial.rt) >= ratio:
    continueRoutine = False
```

少なくとも筆者の経験上、Keyboard コンポーネントの **[検出するキー $]** にキー名がひとつしかない場合は一度に 1 回しかキー押しはカウントされないので、if 文の条件式は >= ではなく == でも恐らく正常に動作すると思います。しかし、一度に 2 回以上キー押しがカウントされてしまうことが万一起こったら == では永遠にルーチンが終了しなくなってしまいますので、上記のコードでは念のために >= としています。

以上の変更で FR/VR スケジュールへの対応ができました。次は再び exp12vi.psyexp をベースにして、実験開始後一定時間が経過したら実験を終了するように改造しましょう。

12.6 条件を満たしたら実験を強制終了するようにしよう

Builder は、基本的に条件ファイルで定められたすべてのパラメータを、ループで指定された回数だけ必ず繰り返すように設計されています。言い換えると、指定された試行数を実行するまで実験は終了しません。しかし、心理学の実験の中には、「直近の 20 試行の正答率が 80% 以上になれば終了」とか、「実験開始から 20 分経過したら終了」といった具合に、試行数以外の条件で終了する実験もあります。このような実験は Builder との相性が悪いのですが、Code コンポーネントを使うと実現することができます。 **なお、本節の方法は Pavlovia によるオンライン実験ではうまく動作しませんので、別の方法を用いる必要があります。** 詳しくは「 12.9.2:Pavlovia でのオンライン実験にも視野に入れたループの中断方法 」をご覧ください。

この節では、「試行数以外の条件で終了する実験」の例として、実験開始から指定された時間が経過したら実験を終了する実験を exp12vi.psyexp をベースに作成してみましょう。exp12vi.psyexp を exp12vi_time_limited_1.psyexp という別名で保存して、以後の作業は exp12vi_time_limited_1.psyexp に対しておこないます。

すでに前節までの作業で、グローバルクロックを用いて実験開始からの経過時間を得る方法は解説しました。先ほど「Code コンポーネントを使えば実現できる」と言ったのですから、exp12vi_time_limited_1.psyexp のどこかに

```
if globalClock.getTime( ) > 制限時間を保持している変数:
    実験を終了させる
```

というコードを書けばいいということは想像がつくと思います。実行中の実験を終了させると言えば思い出されるのが、実験設定ダイアログで [Enable Escape] をチェックしていれば ESC キーを押すことで実験を強制終了できる機能です (第 2 章)。この機能は、Builder の内部で psychopy.core.quit( ) という関数を呼び出すことで実現されています。この関数は名前通り、すべての PsychoPy のウィンドウを破棄して実験を終了します。Builder 内部ではこの関数を core.quit( ) という名前で呼び出すように import が行われているので、

```
if globalClock.getTime( ) > limitTime:
    core.quit( )
```

と書けば、実験を終了できます。なお、制限時間を保持している変数を limitTime としました。

残る問題は、このコードをどの時点で実行すればよいかということです。いろいろな候補が考えられますが、重要なポイントは「指定された時間が経過したら、課題遂行中であろうが直ちに終了してほしい」のか、「指定された時間が経過したら、現在遂行中の課題を通常通り終えてから終了してほしい」のかの違いです。前者であれば、Code コンポーネントの **[フレーム毎]** に上記のコードを記入して、フレーム毎に経過時間をチェックするべきです。後者であれば **[Routine 終了時]** か **[Routine 開始時]** にコードを記入するべきです。今回の実験では、interval に大きな値を指定した時には特に、指定時間が経過してから trial ルーチンが終了するまで時間がかかりますから、trial ルーチンの途中で直ちに終了するべきでしょう。reinforcement ルーチンはわずか 1 秒しかありませんし、強化子が途中で途切れるのは不適切だと思われますので、reinforcement ルーチンは途中で中断しないことにしましょう。そうすると、コードを挿入すべきなのは trial ルーチンの **[フレーム毎]** ということになります。

trial ルーチンにはすでに Code コンポーネントを配置済みですので、プロパティ設定ダイアログを開いて **[フレーム毎]** に以下のコードを追加してください。追加する位置は入力済みのコードの前でも後でもどちらでも

構いません。

```
if globalClock.getTime( ) > limitTime:
    core.quit( )
```

あとは変数 limitTime の準備です。ここでは実験情報ダイアログから limitTime の値を指定できるようにしましょう。実験設定ダイアログを開いて、**[実験情報ダイアログ]** に time limit (s) という項目を追加してください。そして、trial ルーチンの Code コンポーネントのプロパティ設定ダイアログを再び開いて、 **[実験開始時]** に以下のコードを追加します。やはり入力済みのコードの前でも後でもどちらでも構いません。

```
limitTime = float(expInfo['time limit (s)'])
```

以上で変更は終了です。exp12vi_time_limited_1.psyexp を保存して実行してみてください。実験情報ダイアログに time limit (s) という項目がありますので、実験の終了時間を秒で指定しましょう。とりあえず動作確認のために 30 秒くらいの値を指定するのがお勧めです。条件ファイルと繰り返し回数も忘れずに指定して実験を実行しましょう。30 秒経過した時点で実験が自動的に終了するはずです。trial-by-trial 記録ファイルを開いて、強化まで進んだ試行についてはキー押しの記録が残っていることを確認してください。当然、中断された試行については保存処理なしにいきなり実験が終了していますので、キー押し記録は残りません。

中断された試行のキー押し記録も残したい場合は、psychopy.core.quit( ) 以外の方法で実験を終了させる必要があります。また、制限時間が経過したら実験全体を終了させてしまうのではなく、現在の課題を終了させて次の課題を実行したいという場合にも、psychopy.core.quit( ) は使えません。これらの場合にはどのような方法が有効でしょうか。勘のいい人は、psychopy.core.quit( ) の代わりに continueRoutine = False とすればいいじゃないかと思われるかもしれません。確かに、trial ルーチンと reinforcement ルーチンに Code コンポーネントを置いて、 **[フレーム毎]** で以下のコードを実行すれば、グローバルクロックの値が limitTime を超えた以後は trial ルーチンも reinforcement ルーチンも一瞬で終了してしまいます。この方法ならルーチン終了の処理もすべて通常通り行われますから中断された試行のキー押し記録も残りますし、終了させたい課題にだけこのコードを挿入しておけば、「現在の課題を終了させて次の課題を実行する」ことも可能です。

```
if globalClock.getTime( ) > limitTime:
    continueRoutine = False
```

この方法を自力で思いついた方は、前章までの内容をとてもよく理解しておられると思います。ですが、残念なことに、この方法にはひとつ問題があります。どのような問題が生じるのか、実際に試してみましょう。

exp12vi_time_limited_1.psyexp を開いて、今度は exp12vi_time_limited_2.psyexp という名前で保存してください。以後、exp12vi_time_limited_2.psyexp に対して作業をするものとします。保存したらさっそく exp12vi_time_limited_2.psyexp を開いて、trial ルーチンの codeTrial の **[フレーム毎]** の core.quit( ) を continueRoutine = False に書き換えましょう。書き換え後の **[フレーム毎]** は以下のようになっているはずです (ふたつの if 文の順番が逆でも構いません)。

```
if len(key_resp_Trial.rt) > 0 and key_resp_Trial.rt[-1] >= interval:
    continueRoutine = False
if globalClock.getTime( ) > limitTime:
    continueRoutine = False
```

続いて reinforcement ルーチンを開いて、Code コンポーネントを配置してください。 **[名前]** は codeRF としておきましょう。codeRF の **[フレーム毎]** に以下のコードを追加します。

```
if globalClock.getTime() > limitTime:
    continueRoutine = False
```

これで作業は完了ですが、trial ルーチンと reinforcement ルーチンの実行を中断した後に別の課題を行えることを確認するために、trials ループの後にルーチンをひとつ置きましょう。以下の作業を行ってください。

- feedback ルーチン (作成する)
  - フローの trials ループの後ろ (つまりフローの最後) に挿入する。
  - Text コンポーネントをひとつ配置し、以下のように設定する。
    - **[名前]** を textFB にする。
    - **[終了]** を「実行時間 (秒)」に設定し、5 と入力する。
    - **[文字列]** を$feedbackMessage にし、「繰り返し毎に更新」に設定する。
  - Code コンポーネントをひとつ配置し、以下のように設定する。
    - **[名前]** を codeFB にする。

さらに、Code コンポーネントの **[Routine 開始時]** に以下のコードを入力してください。

```
if globalClock.getTime() > limitTime:
    feedbackMessage = 'Time out'
else:
    feedbackMessage = 'Completed'
```

作業が終わったら、exp12vi_time_limited_2.psyexp を保存して、実行してください。条件ファイルに exp12fi.xlsx を指定して、nReps を 100、time limit (s) を 5 にしてみましょう。実行したら、スペースキーを押さずに制限時間の 5 秒が過ぎるのを待ってください。いかがでしょうか。10 秒経過した後、PC からピピピピーッと音がなってからスクリーンに Time out と表示されたのではないかと思います。何が起きたのかわかりましたか？

グローバルクロックが limitTime を超えたら continueRoutine = False を実行するというコードを挿入することによって、確かに「trial ルーチンと reinforcement ルーチンを (nReps で指定された)100 回繰り返さずに」feedback ルーチンへ進むことができました。しかし、いくら一瞬で終了するとはいえ trial ルーチンと reinforcement ルーチンは実行されているので、reinforcement ルーチンに含まれていた Sound コンポーネントの音が鳴り響いたのです。これでは「trial ルーチンと reinforcement ルーチンの実行を中断した」とは言えません。これらのルーチンを一瞬たりとも実行させずに feedback ルーチンへ進まなければなりません。

この難題を解決するためには、Builder の仕組みにさらに踏み込まなければなりません。Builder はルーチンを実行している時、実際にはどのようなコードを実行しているのでしょうか。そして、continueRoutine という変数の値を False にするとルーチンが終了するというのはいったいどういう仕組みによるものなのでしょうか。これらの点を理解していただくためには、if 文、for 文と並ぶ Python の重要構文である while 文を覚えていただく必要があります。

while 文とは、for 文と同様に繰り返し処理を行うための文です。for 文では、与えられたシーケンス型データに含まれる要素を先頭から取り出して順番に処理をしていきました。あらかじめ繰り返し処理の対象が決まっている時には for 文はとても便利なのですが、心理学実験でよくある「キーが押されるまでスクリーンに刺激を描画し続ける」といった処理では、いつまで描画を繰り返せばいいのか実際に実行してみるまでわかりません。このような「ある条件が満たされるまで同じ処理を繰り返す」という処理を行いたいときに用いるのが while 文です。

図 12.6 に while 文の概要を示します。while 文は条件式と組み合わせて使用し、条件式が True である間、処理を繰り返します。繰り返す範囲は for 文や if 文と同様に字下げで示します。注意しないといけないのは、while 文と組み合わせる条件式に誤って絶対に False にならない式を書いてしまうと、Python インタプリタ自体が正常に動作している限り永遠に処理が終了しないという点です。ただし、for 文と同様に break と continue を使うことができますので、繰り返し中に break を実行すれば条件式に関わらず繰り返しを中断することはできます。なお、while 文には if 文のように else を伴わせることができますが、その時の動作については「 12.9.1:while 文に伴う else 」を参照してください。

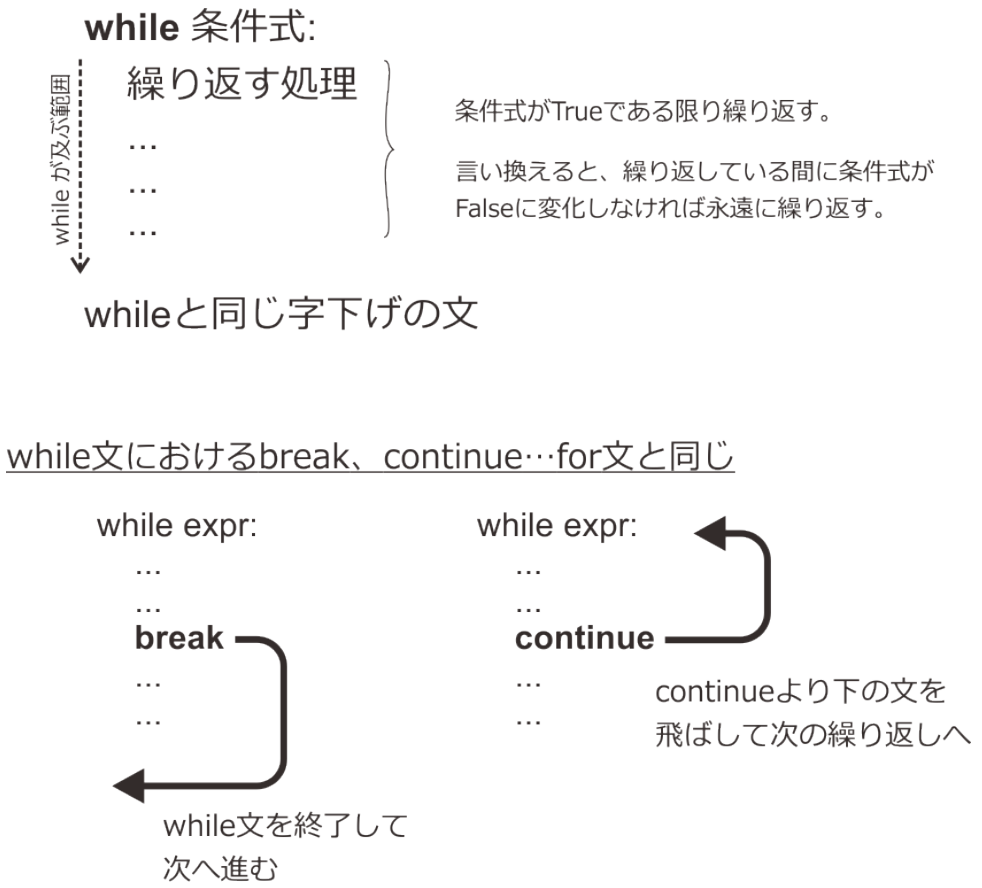


図 12.6　while 文。条件式が True である限り処理を繰り返します。for 文と同様に break や continue を使うことができます。

図 12.7 は、Builder が実験のフローをどのように Python のコードへ変換するかを示しています。フローにおけるループは for 文に変換されます。条件ファイルや **[Loop の種類]** 、 **[繰り返し回数 $]** などに基づいて各試行で用いられるパラメータ一覧のリストが作成され、for 文へ渡されます。この for 文を実行することで、パラメータを変更しながら指定された回数の繰り返しが実行されます。一方、各ルーチンは while 文に変換されます。 図 12.7 は Code コンポーネントの解説も含んでいて複雑になっていますので、以下に while 文の部分を抜粋して示します。

```
continueRoutine = True
while continueRoutine and ルーチン継続に関する条件:
    各コンポーネントのフレーム毎の処理
```

ルーチンが開始される直前に、continueRoutine という変数に True が代入されます。そして while 文によってルーチン内の各コンポーネントのフレーム毎の処理を繰り返します。while 文の条件式には、continueRoutine とその他の条件の論理積 (and) が渡されます。その他の条件というのは、例えばルーチンが 5.0 秒で終了するように各コンポーネントの **[終了]** が設定されていた場合には「ルーチン開始からまだ 5.0 秒経過していない」という条件が入ります。continueRoutine が False の場合、他にどのような条件が指定されていても while 文の条件式は False になるので、continueRoutine = False を実行すると while 文の繰り返しが終了します。これが今まで呪文のように continueRoutine = False と書いていた時に実際に生じていたことなのです。

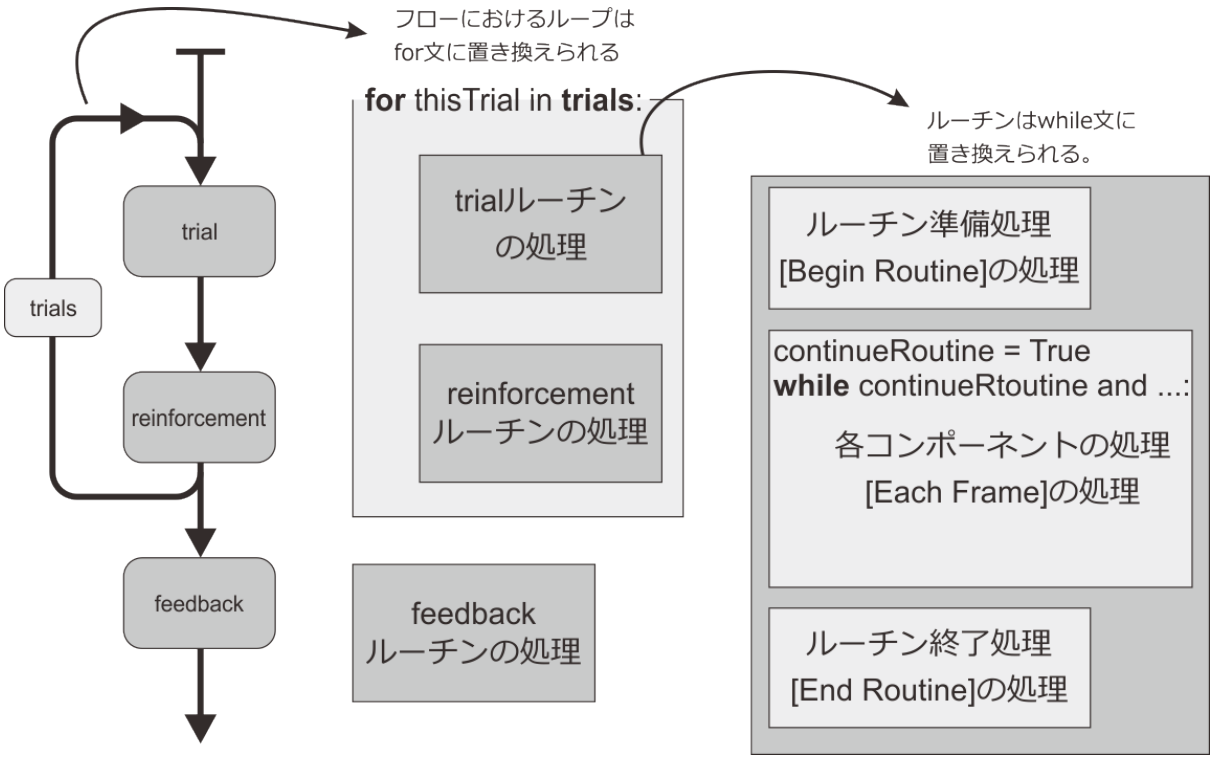


図 12.7　Builder によるフローから Python コードへの変換。フローにおけるループは for 文に、ルーチンのフレーム毎の処理は while 文に置き換えられます。

以上を踏まえて改めて 図 12.7 をご覧ください。 図 12.7 では、Code コンポーネントで **[Routine 開始時]** 、**[フレーム毎]** 、 **[Routine 終了時]** にコードを記入した時にどの位置にコードが挿入されるかを示しています。これをご覧になったら、continueRoutine = False を **[フレーム毎]** に書かなければ while 文を終了させることができないのがおわかりいただけるかと思います。

さて、なぜこんな Builder の裏側の世界まで足を踏み入れているのかと言えば、何とかしてルーチンを一瞬たりとも実行させずに中断させたいのでした。continueRoutine には while 文が実行される直前に True が代入されます。ですから Code コンポーネントの **[Routine 開始時]** に continueRoutine = False と書いても実行直前に True に書き換えられてしまい、ルーチン (=while 文) の実行を止めることができません。一度でも while 文が実行されると Sound コンポーネントが実行されてしまうので、いくら **[フレーム毎]** に continueRoutine = False と書いても音が鳴ってしまいます。これが exp12vi_time_limited_2.psyexp で生じていた問題です。では、ど

うすればいいでしょうか？ ここで思い出していただきたいのが break です。break が実行されると、break 以降に処理があろうと直ちに繰り返しが中断されるのでした。ということは、ルーチンが実行された後、どのコンポーネントの処理よりも先に break を実行すれば、実質的にルーチンを実行しなかったのと同じ結果になるはずです。さっそく試してみましょう。

exp12vi_time_limited_2.psyexp を、exp12vi_time_limited_3.psyexp という別名で保存してください。以下の作業は exp12vi_time_limited_3.psyexp に対して行うものとします。別名で保存したら、trial ルーチンを開いて codeTrial の **[フレーム毎]** を確認してください。globalClock.getTime( ) > limitTime だったら continueRoutine = False するというコードが入力されているはずですが、この continueRoutine = False を break に書き換えてください。書き換えた後の **[フレーム毎]** は以下のようになっているはずです (ふたつの if 文の順番が逆でも構いません)。

```
if len(key_resp_Trial.rt) > 0 and key_resp_Trial.rt[-1] >= interval:
    continueRoutine = False
if globalClock.getTime( ) > limitTime:
    break
```

同様に、reinforcement ルーチンの codeRF の **[フレーム毎]** も以下のように break に書き換えてください。

```
if globalClock.getTime() > limitTime:
    break
```

そして、次の作業が重要です。これらの修正したコードが、同一ルーチン内の他のコンポーネントの処理が行われる前に実行されるようにする必要があります。そのためには、codeTrial や codeRF がルーチンペインの一番上に並んでいる必要があります (図 12.8)。trial ルーチンと reinforcement ルーチンを開いて、codeTrial と codeRF がそれぞれ一番上になるように並び替えましょう。

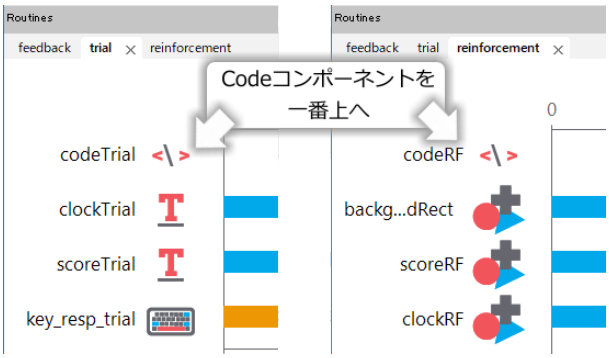
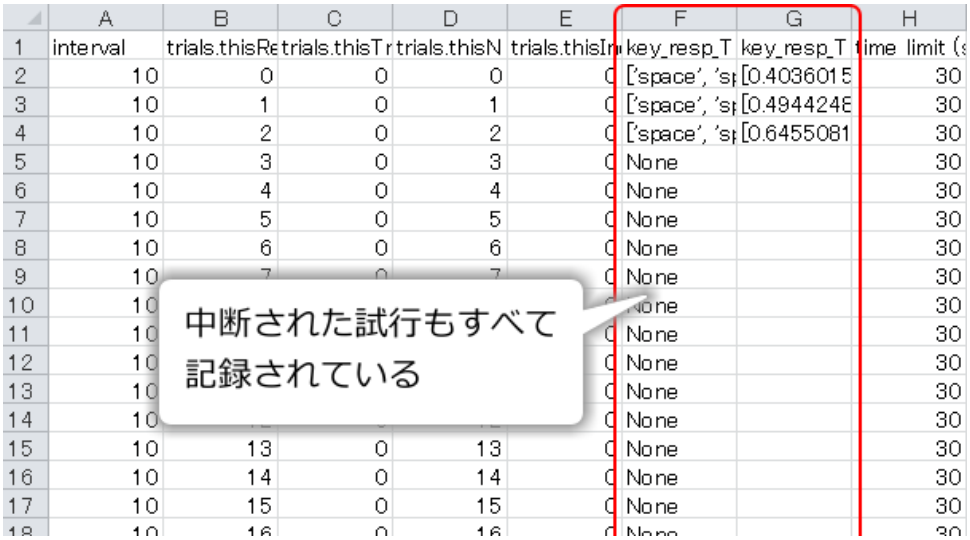


図 12.8　trial ルーチンと reinforcement ルーチンの Code コンポーネントを一番上へ配置してください。

作業が終わったら、exp12vi_time_limited_3.psyexp を保存して実行しましょう。先ほどと同様に、条件ファイルに exp12fi.xlsx を指定して、nReps を 100 にしてください。時間内に得点が得られた試行のデータも欲しいので、time limit (s) は 30 にしてください。実行したら、スペースキーを押して得点を獲得しながら 30 秒経過するのを待ちましょう。30 秒経過したら実験が中断されてスクリーンに Time out と表示されますが、今度は exp12vi_time_limited_2.psyexp の時のように音が鳴らなかったはずです。

psychopy.core.quit( ) で終了した場合と異なり、作成された trial-by-trial 記録ファイルには、 図 12.9 のように中断された試行もすべて記録されています。break で中断された試行ではキー押しが記録されませんので、

key_resp_Trial.keys と key_resp_Trial.rt はいずれも None と出力されています。かなり苦戦しましたが、これでなんとか目的を達成することができました。



	A	B	C	D	E	F	G	H
1	interval	trials.thisRe	trials.thisTr	trials.thisN	trials.thisIn	key_resp_T	key_resp_T	time limit (s
2	10	0	0	0	0	['space', 'sp	[0.4036015	30
3	10	1	0	1	0	['space', 'sp	[0.4944248	30
4	10	2	0	2	0	['space', 'sp	[0.6455081	30
5	10	3	0	3	0	None		30
6	10	4	0	4	0	None		30
7	10	5	0	5	0	None		30
8	10	6	0	6	0	None		30
9	10	7	0	7	0	None		30
10	10				0	None		30
11	10				0	None		30
12	10				0	None		30
13	10				0	None		30
14	10				0	None		30
15	10	13	0	13	0	None		30
16	10	14	0	14	0	None		30
17	10	15	0	15	0	None		30
18	10	16	0	16	0	None		30

図 12.9　psychopy.core.quit( ) で中断した場合と異なり、ルーチンを break で中断した場合は中断された試行もすべて記録されています。

なお、皆さんの中には、exp12vi_time_limited_3.psyexp を実行した時に、30 秒経過してから Time out と表示されるまでに少し「30.0」という表示のまま PC が停止したかのような動作をしたことが気になった方がいるかも知れません。これは、Builder が残りの全て実際に開始して、即 break して、結果を trial-by-trial 記録ファイルに書きこむという作業を繰り返しているために生じる現象です。break された試行もすべて記録に残すべきか否かはどちらが良いか一概には言えませんが、この待ち時間が問題になる実験ももしかしたらあるかも知れません。この節のテクニックを応用することで待ち時間をなくすことも可能ですが、これは練習問題としておきましょう。

チェックリスト

- 条件式が True である間処理を繰り返す Python のコードを書くことができる。
- Code コンポーネントを用いて、ある条件を満たしたときに実行が中断される実験を作ることができる。
- Code コンポーネントを用いて、ある条件を満たしたときに実行がスキップされるルーチンを作ることができる。

12.7 繰り返し回数を変更して並立スケジュールを実現しよう

Builder に対する不満として非常によく聞くのが、フローを分岐させることができないというものです。例えば、スクリーン上に複数の選択肢が提示されて、実験参加者がひとつを選択すると、それに応じた課題が行われるといった実験を現在の Builder は想定していません。しかし、Code コンポーネントを利用するとフロー上では分岐できなくても動作上は分岐する実験を作成することが可能です。

具体的な例題が無いと話がしづらいので、 図 12.10 のような並列スケジュールの実験を考えてみましょう。exp12vi.psyexp の前にスクリーンをひとつ挿入して、そこで「カーソルキーの左右を押して課題を選択してく

ださい」と教示します。参加者は適切なタイミングでスペースキーを押すと得点が得られること、できるだけ速く 50 点を得るように努力することだけが告げられていて、左右の課題がそれぞれどのようなスケジュールであるのか知らされていません。左右いずれかのキーを押すと、押したキーに応じたスケジュールで課題が始まります。

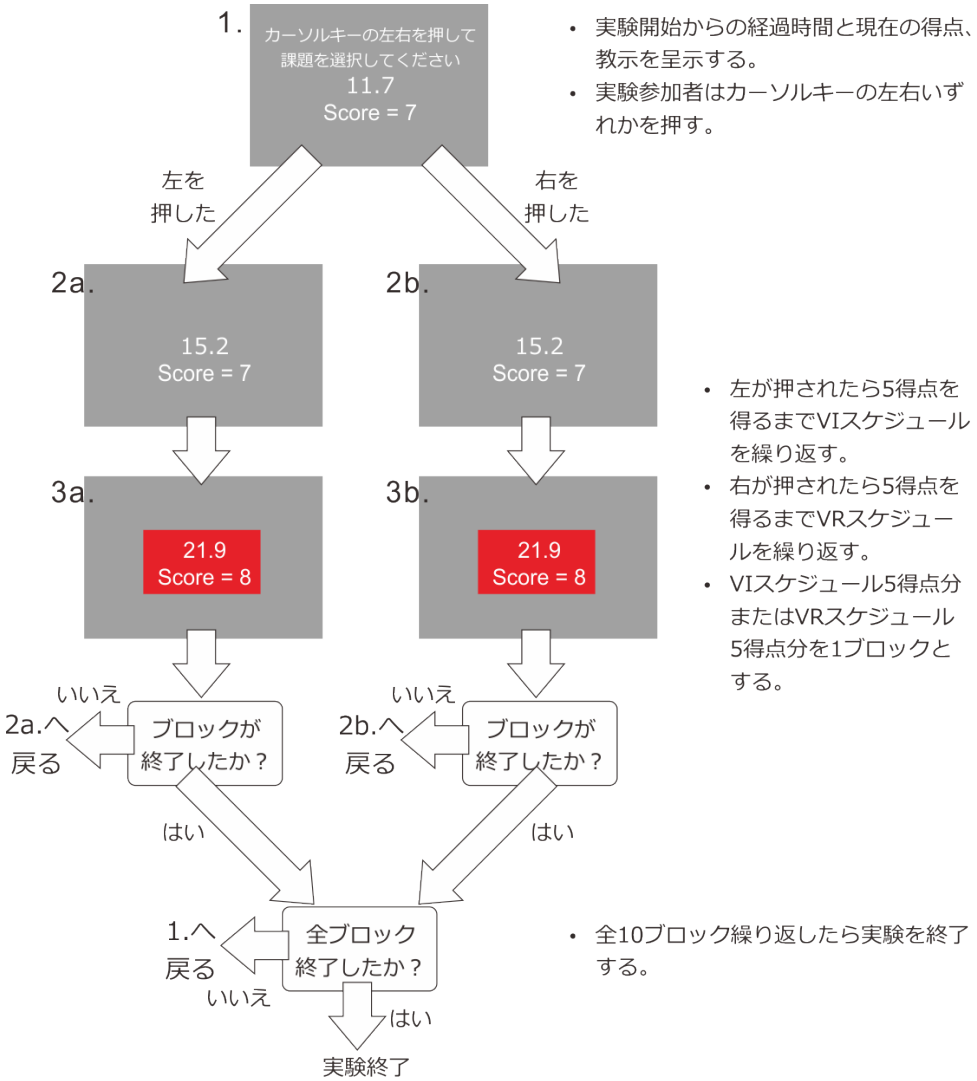


図 12.10　並立スケジュール。1. のスクリーンで実験参加者がカーソルキーの左右どちらを押すかによって次のブロックの課題が変化します。

並列スケジュールの実験の作成において、

- 二つの課題がどちらも VI
- 左キーの場合は強化時間が 2 秒、11 秒、20 秒、29 秒、38 秒の 5 種類
- 右キーの場合は強化時間が 12 秒、16 秒、20 秒、24 秒、28 秒の 5 種類

といった具合に、左右の課題の違いがパラメータの違いだけであれば、実現はあまり難しくありません。ルーチンとループは共通のものを利用して、条件ファイルを押されたキーに応じて変更すれば実現できます。これは練習問題にしておきます。厄介なのは

- 左キーの場合は VI で強化時間が 2 秒、6 秒、10 秒、14 秒、18 秒の 5 種類

- 右キーの場合は VR で強化回数が 12 回、16 回、20 回、24 回、28 回の 5 種類

といった具合に、押されたキーによって実験内容が異なる場合です。前節の break を駆使する方法でも実現可能ですが、この節ではループの実行そのものを省略する方法を紹介します。

この節の実験では、前節までのような時間制限がないので、exp12vi.psyexp をベースに改造するのがよいでしょう。exp12vi.psyexp を開いて、exp12concurrent.psyexp という別名で保存してください。保存したら、exp12concurrent.psyexp に対して以下の作業を行ってください。かなり複雑なフローになりますので、完成後のフローを 図 12.11 に示しておきます。

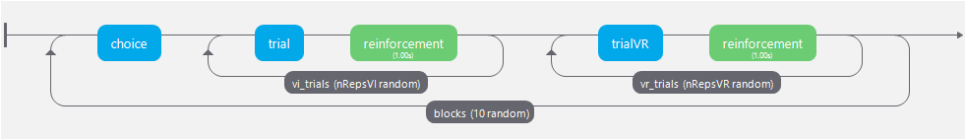


図 12.11　exp12concurrent.psyexp のフロー。

- 実験設定ダイアログ
  - **[実験情報ダイアログ]** から nReps と condition を削除する。
- trials ループ (vi_trials ループに名称変更)
  - **[名前]** を vi_trials にする。
  - **[繰り返し回数 $]** を nRepsVI にする。
  - **[繰り返し条件]** を$condition にする。
- trialVR ルーチン (作成する)
  - vi_trials ループの直後に挿入する。
  - Text コンポーネントをふたつ配置して、それぞれ **[名前]** を scoreTrialVR、clockTrialVR にする。両方とも **[終了]** を空白にする。
    - scoreTrialVR の **[位置 [x, y] $]** を [0.0, -0.2] にする。 **[文字列]** に$'Score:'+str(score) と入力し、「繰り返し毎に更新」に設定する。
    - clockTrialVR の **[文字列]** に$'%.1f' % globalClock.getTime( ) と入力し、「フレーム毎に更新する」に設定する。
  - Keyboard コンポーネントをひとつ配置して、以下のように設定する。
    - **[名前]** を key_resp_TrialVR にする。
    - **[終了]** を空白にする。
    - **[Routine を終了]** のチェックを外す。
    - **[検出するキー $]** に'space' と入力する。
    - **[記録]** を「全てのキー」にする。
  - Code コンポーネントをひとつ配置し、 **[名前]** を codeTrialVR にする。

    * **[Routine 終了時]** に score+=1 と入力する。

- reinforcement ルーチン
  - trialVR ルーチンの直後に挿入する。フロー内に 2 個の reinforcement ルーチンが配置されることになる。
- vr_trials ループ (作成する)
  - trialVR ルーチンと、その後ろにある reinforcement ルーチンを繰り返すように挿入する。
  - **[繰り返し回数 $]** を nRepsVR に、 **[繰り返し条件]** を$condition にする。
- choice ルーチン (作成する)
  - vi_trials ループの前に挿入する。
  - Text コンポーネントを 3 つ配置して、**[名前]** をそれぞれ textInstruction、scoreChoice、clockChoice とする。すべて **[終了]** を空白にする。以下のように設定する。
    * textInstruction の **[位置 [x, y] $]** を [0.0, 0.2] にする。 **[文字列]** に「カーソルキーの左右を押して課題を選択してください」と入力する。
    * scoreChoice の **[位置 [x, y] $]** を [0.0, -0.2] にする。 **[文字列]** に$'Score:'+str(score) と入力し、「繰り返し毎に更新」に設定する。
    * clockChoice の **[文字列]** に$'%.1f' % globalClock.getTime( ) と入力し、「フレーム毎に更新する」に設定する。
  - Keyboard コンポーネントをひとつ配置して、以下のように設定する。
    * **[名前]** を key_resp_Choice にする。
    * **[終了]** を空白にする。
    * **[検出するキー $]** を'left','right' にする。
    * **[記録]** を「なし」にする。
  - Code コンポーネントを配置し、 **[名前]** を codeChoice にしておく。現時点ではコードは入力しない。
- blocks ループ (作成する)
  - フロー全体を繰り返すように挿入する。すなわち、choice ルーチンの前から vr_trials ループの終わりまでを繰り替えす。
  - **[名前]** を blocks にする。
  - **[繰り返し回数 $]** を 10 にする。
- exp12concurrentVI.xlsx(条件ファイル)
  - exp12vi.xlsx をコピーして、interval の値を 2, 6, 10, 14, 18 にする。
- exp12concurrentVR.xlsx(条件ファイル)

  - exp12vr.xlsx をコピーして、ratio の値を 12, 16, 20, 24, 28 にする。

さらに、trialVR ルーチンの codeTrialVR の **[フレーム毎]** に以下のコードを入力してください。

```
if len(key_resp_TrialVR.rt) >= ratio:  continueRoutine = False
```

準備はよろしいでしょうか。それでは最後の仕上げ、Choice ルーチンの Code コンポーネント (codeChoice) にコードを入力しましょう。codeChoice では、以下の処理を行います。

- key_resp_Choice で押されたキーが'left' であれば、以下の変数を設定する。
  - nRepsVI = 1
  - nRepsVR = 0
  - condition = 'exp12concurrentVI.xlsx'
- key_resp_Choice で押されたキーが'right' であれば、以下の変数を設定する。
  - nRepsVI = 0
  - nRepsVR = 1
  - condition = 'exp12concurrentVR.xlsx'

もうすでに何をしようとしているかお分かり頂けたと思います。VI スケジュールを行うループ vi_trials の繰り返し回数は、nRespVI という変数で決まります。nRepsVI に 0 を代入すれば、繰り返し回数が 0 回となって一度も VI スケジュールの試行が行われないというわけです。同様に、nRepsVR に 0 を代入すれば、VR スケジュールの試行は行われません。押されたキーに従って変数に異なる値を設定するのはすでに 第 6 章 で、**[記録]** が「なし」の時に押されたキーを取得する方法は 第 7 章 で学びましたから、新しい作業は何もありません。以下のコードを codeChoice の **[Routine 終了時]** に入力してください。

```
if 'left' in theseKeys:
    nRepsVI = 1
    nRepsVR = 0
    condition = 'exp12concurrentVI.xlsx'
elif 'right' in theseKeys:
    nRepsVI = 0
    nRepsVR = 1
    condition = 'exp12concurrentVR.xlsx'
```

入力したら、exp12concurrent.psyexp を保存して実行してみましょう。選択画面が出てきたら教示に従ってカーソルキーの左右いずれかを押し、左を押せば VI スケジュール、右を押せば VR スケジュールの課題が 5 得点分ずつ実行されることを確認してください。実行後に作成される trial-by-trial 記録ファイルの例を 図 12.12 に示します。繰り返し回数を 0 に設定することによって飛ばされたループに該当するセルは、空白になります。どのような順番で課題を選択したか、それぞれの課題においてキーは何回押したか、いつ押したかといった情報を全て trial-by-trial 記録ファイルから読み取ることができます。

さて、ずいぶん解説が長くなってしまいましたので、この章もそろそろおしまいにしたいと思います。この章で紹介したテクニックを用いると、現状の Builder でもフローが分岐する実験を作成することができます。次

章はいよいよ最終章です。次章では無作為化に関する話題を取り上げます。

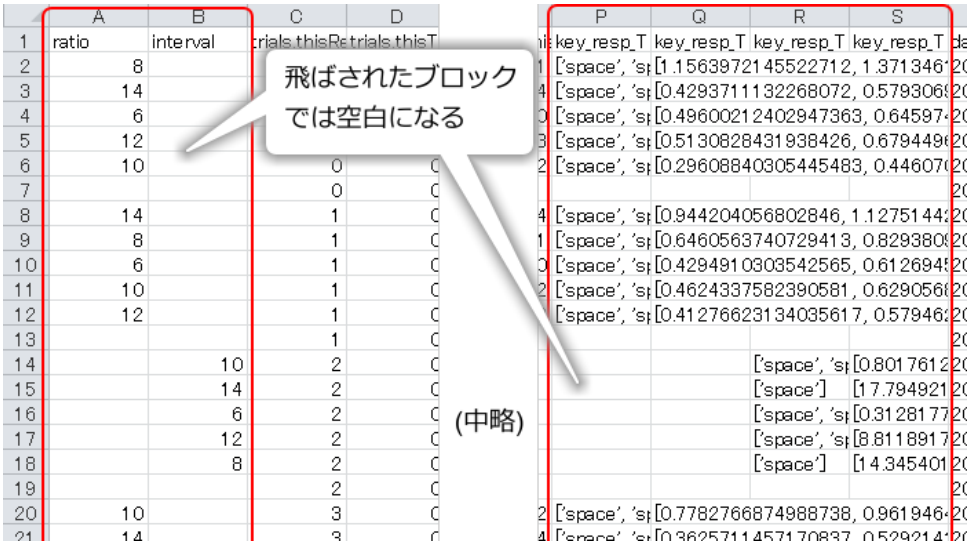


図 12.12　exp12concurrent.psyexp を実行した時に作成される trial-by-trial 記録ファイルの例。繰り返し回数を 0 に設定することによって飛ばされたループに該当するパラメータやキー押し記録のセルは、空白になります。

チェックリスト

- ある条件に当てはまる時にループを実行しない実験を作成することができる。
- ループを実行しない実験を作成した時の trial-by-trial 記録ファイルから、ループを実行しなかった時の記録を判別できる。

12.8 練習問題：さまざまなフロー制御をマスターしよう

この章の練習問題は以下の 3 問です。第 2 問と第 3 問は本文中で「練習問題」としたものです。

- 問 1：exp12vi.psyexp をベースとして、フローの先頭に教示画面を挿入してください。そして、trial ルーチンや reinforcement ルーチンで表示される経過時間を実験開始からの時間 (globalClock が示す時間) ではなく、教示画面を終了して実際に強化スケジュールが開始してからの時間を表示してください。
  - 教示文は各自に任せます。
  - 教示画面でスペースキーを押したら実験が始まるようにしてください。
- 問 2：exp12vi.psyexp をベースとして、 図 12.10 の並列スケジュールの実験を以下の条件で作成してください。
  - 左キーが押された場合は強化時間が 2 秒、11 秒、20 秒、29 秒、38 秒の 5 種類の VI スケジュール
  - 右キーが押された場合は強化時間が 12 秒、16 秒、20 秒、24 秒、28 秒の 5 種類の VI スケジュール
  - 左キーと右キーのスケジュールの実行には同一のループおよびルーチンを使用し、条件ファイルの切り替えによってスケジュールの違いに対応する。

- 問 3：exp12vi_time_limited_3.psyexp をベースとして、time limit (s) で指定された制限時間を過ぎてしまったときに、スキップされた試行のデータが trial-by-trial 記録ファイルに出力されないようにする。ただし、制限時間によって中断された試行の、中断直前までのキー押しは trial-by-trial 記録ファイルに出力されているようにすること。

12.9 この章のトピックス

12.9.1 while 文に伴う else

Python では while 文に else を伴わせることができます。if 文に伴う else は他の多くのプログラミング言語で使用することができますので、他言語でのプログラミング経験がある人はすぐにわかったと思うのですが、while 文に伴う else は C 言語などには無いので戸惑われるかたもいるかもしれません。

if 文の else は、if 文の条件式が False だった時に行う処理を指定するためのものでした。while 文の else も同様に、while 文の条件式が False だった時に行う処理を指定します。以下の while 文を考えてみましょう。

```
x = y = 0
while x<10:
    x += 1
else:
    y = x
```

最初に x と y に 0 が代入され、while 文で x+=1 を繰り返します。x=10 になった時点で while 文の条件式が False になるので、else で記述されている y = x が実行されます。結果として、x と y の値は 10 となります。続いて以下の式を考えてみましょう。while 文で繰り返す処理の中に「x の値が 5 であれば break する」という処理を追加しています。

```
x = y = 0
while x<10:
    x += 1
    if x == 5:
        break
else:
    y = x
```

この場合、while 文を繰り返しているうちに x の値が 5 になって break が実行されます。while 文の条件式である x<10 が False になったわけではないので、else の処理は実行されません。結果として、x の値は 5 になり、y の値は 0 のままになります。

12.9.2 Pavlovia でのオンライン実験にも視野に入れたループの中断方法

「5.6: 複雑な式には Code コンポーネントを使ってみよう」で触れたように、現在の Builder はオンライン実験サービス Pavlovia で動作する実験プログラムを出力することができるようになりました。このプログラムは JavaScript という言語と PsychoJS というライブラリが用いられており、Python で書かれたプログラムとは大きく異なります。そのため、本章で紹介したループの中断方法は Pavlovia の実験プログラムでは期待通りに動作しません。そのため本章の内容を全面的に書き直すことも検討したのですが、

- ローカル PC で実行する Python のプログラムを Builder で作成する場合には本章の内容は今なお有効であること
- Builder がループを Python のプログラムとして実現する仕組みを理解して、将来的に Builder に頼らずに実験コードが書けるようになること

を考慮してそのまま残すこととしました。代わりに、このトピックで Pavlovia でループを中断する方法を解説しておきたいと思います。

仰々しい前置きとなりましたが、方法は非常にシンプルです。現在の PsychoPy の TrialHandler オブジェクト (「7.4:Code コンポーネント使って独自の変数の値を記録ファイルに出力しよう」) には **finished** というデータ属性が用意されています。この finished に True を代入すると、ループ内のルーチンの実行が終わったときにまだ繰り返しが残っていても TrialHandler はループを終了します。つまり、何らかの条件が満たされて trials という名前のループを中断したいときは

```
trials.finished = True
```

という文を実行すればよいのです。現在のルーチンの中断を同時に行いたい場合は

```
trials.finished = True
continueRoutine = False
```

とすれば目的を達成できます。この方法が素晴らしいのは、Pavlovia 用の JavaScript でも (文法上の違いを除いて) まったく同じコードで動作するということです。具体的には以下のように書きます。

```
trials.finished = true;
continueRoutine = false;
```

Builder でオンライン実験の作成を考えている人は覚えておいてください。

第 13 章

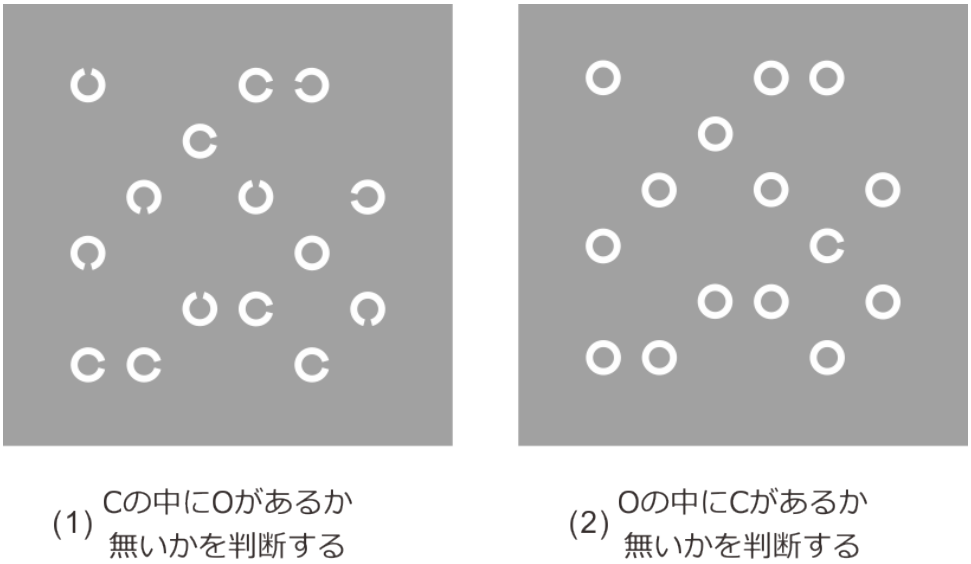
無作為化しよう—視覚探索

13.1 この章の実験の概要

前章では、Builder に対する不満として「条件分岐ができない」という点を挙げて、Code コンポーネントで解決を試みました。この章では、「十数個の視覚刺激を描画するのが面倒」、「すべてのパラメータを明示的に条件ファイルで与えるのが面倒」という問題を取り上げたいと思います。

この章の例題として挙げる実験は、視覚探索課題です。 図 13.1 に視覚探索課題の例を示します。 図 13.1 の (1) では、スクリーン上に切れ目が入っている円 (以下 C と表記) が複数個提示されていますが、50% の確率で一つだけ切れ目がない円 (以下 O と表記) が含まれています。実験参加者は、できるだけ速く正確に、O が含まれているか居ないかを判断しなければいけません。容易に予想できる事ですが、O の有無を判断するまでに要する平均時間はスクリーン上に提示されている図形 (以下アイテムと表記) の個数にほぼ比例して増加します。ところが、 図 13.1 の (2) のように、O と C を入れ替えて、複数個の O の中から C の有無を判断する課題に切り替えると、アイテム数が増えても (1) ほど反応時間が増加しません。この現象を視覚探索の非対称性と呼び、 図 13.1 下のようにアイテム数と平均探索時間の関係をプロットしたグラフを探索関数と呼びます。参加者が探し出すべきアイテムをターゲットと呼び、それ以外のアイテムをディストラクタと呼びます。(1) の課題では O がターゲットで C がディストラクタ、(2) の課題では C がターゲットで O がディストラクタです。この章では、アイテム数を 5 個、10 個、15 個と変化させながら (1) の課題と (2) の課題を行う実験を作成します。

図 13.2 に具体的な手続きを示します。実験を実行すると、準備ができたらカーソルキーの左右を押すように促す教示が提示されます。このスクリーンを 1. とします。実験参加者が自分でキーを押すことによって実験が始まります。各試行の最初には、スクリーン中央に固視点として **[文字の高さ $]** が 24pix の「＋」の文字が提示されます。このスクリーンを 2. とします。実験参加者は固視点を注視しながら刺激の提示を待ちます。待ち時間は試行毎に 1.0 秒、1.5 秒、2.0 秒のいずれかから無作為に選びます。待ち時間が終了したら、スクリーン上に刺激が提示されます。刺激はアイテム数が 3 種類 (5 個、10 個、15 個) ×アイテムの種類が 4 種類 (すべて O、すべて C、O の中にひとつだけ C、C の中にひとつだけ O)=12 種類のいずれかです。実験参加者は、刺激の中に「ひとつだけ周囲と異なるアイテムが存在するか否か」を判断します。ひとつだけ周囲と異なるアイテムが存在する場合はカーソルキーの右、すべて同じアイテムの場合はカーソルキーの左をできるだけ速く、正確に押します。反応に制限時間は設けず、参加者が反応するまで刺激を提示します。参加者が反応したら自動的に次の試行が開始されスクリーン 2.(固視点が提示される画面) へ戻りますが、80 試行毎に休憩のためにスクリーン 1. へ戻って参加者のスペースキー押しを待ちます。12 種類の条件に対して 20 試行ずつ、合計 240 試行を無作為な順序に実施したら実験は終了します。総試行数が 240 試行で 80 試行毎にスクリーン



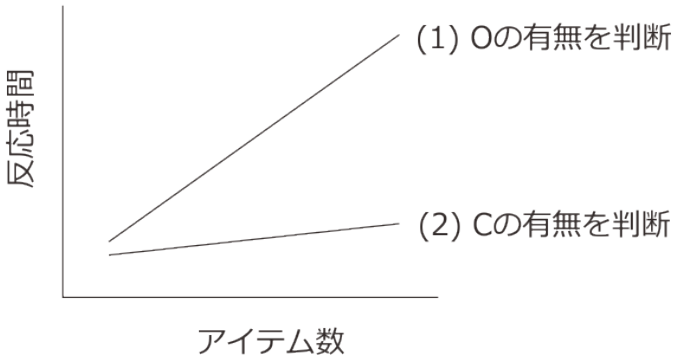


図 13.1　視覚探索の非対称性。スクリーン上に提示されているアイテム数が多いほど判断に時間がかかりますが、C のなかから O を探すより、O の中から C を探す方がアイテム数増加に伴って反応時間が急激に増加します。

1. が挿入されるのですから、スクリーン 1. は実験開始直後、80 試行終了時、160 試行終了時の 3 回提示されます。

刺激の詳細を 図 13.3 に示します。アイテムの位置はスクリーン中央に設定された仮想的な 6 × 6 の格子上から無作為に選ばれます。格子の各マスの幅および高さは 100pix です。スクリーン中央の座標が [0, 0] で、グリッドの全幅は 500pix ですから、グリッドの一番右上の座標は [250,250]、一番左下の座標は [-250, -250] です。一番上の段の座標を左から右に向かって順番に書くと、[-250, 250]、[-150, 250]、[-50, 250]、[50, 250]、[150, 250]、[250, 250] です。

アイテムの直径は O、C ともに 30pix とし、C の場合は円の一部に幅 10pix の切れ目を入れます。切れ目の位置は、アイテムの中心から見て右を 0 度として、時計回りに 0 度、90 度、180 度、270 度の 4 種類の中からアイテム毎に無作為に選択します。

以上が実験の概要です。Builder が苦手とするポイント、できる事なら Builder で作りたくないなあと思ってしまうポイントがいくつか含まれています。これらのポイントはお互いに関連しあっているのですが、敢えて箇条書きにすると以下の 4 点が挙げられます。

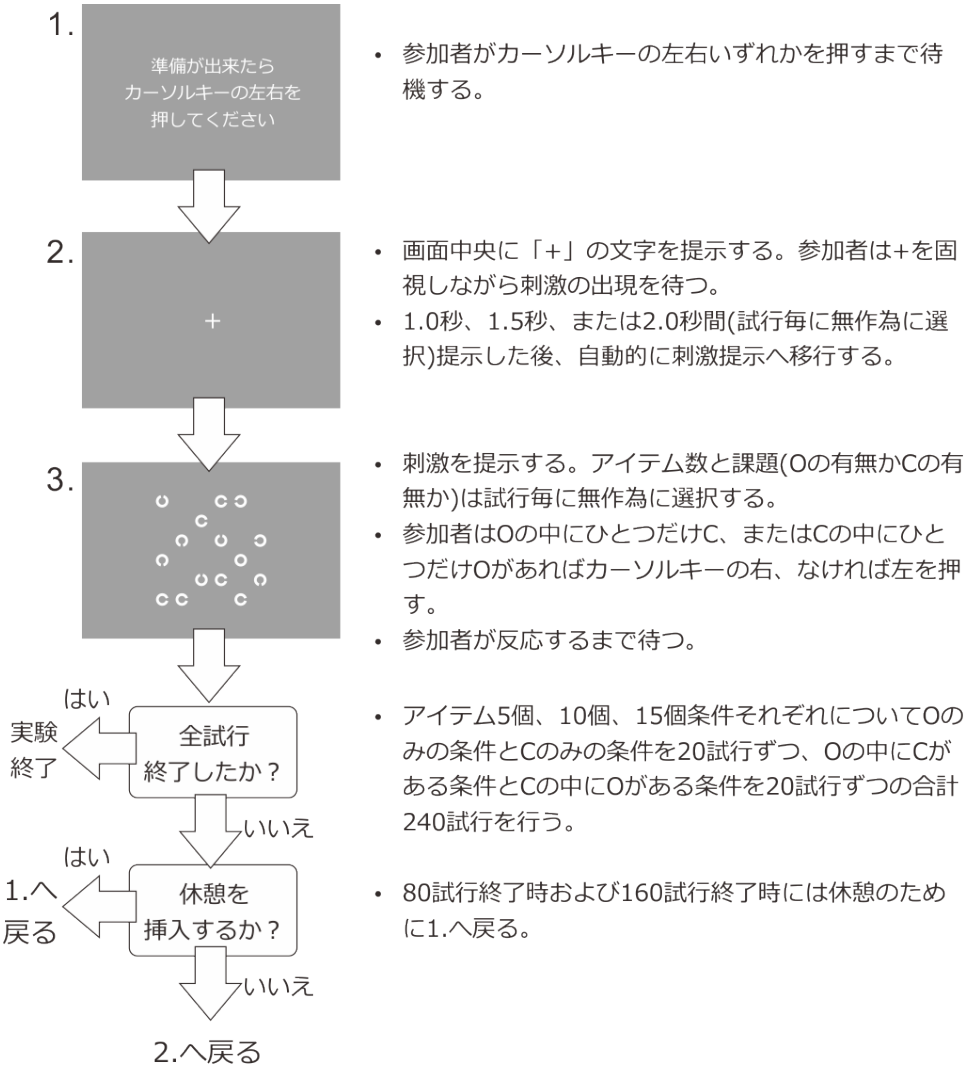


図 13.2　実験の手続き。

1. 独立に位置や形状が変化するアイテムが最大 15 個もスクリーン上に存在する
2. 試行毎にスクリーン上に出現するアイテムの個数が異なる
3. 無作為に設定するパラメータが複数個ある
4. 休憩が挿入される試行が条件数の倍数になっていない

まず 1. と 2. についてですが、画面上に刺激が大量に存在すると、ルーチン上に必要な個数のコンポーネントをひとつひとつ並べてパラメータを設定していかなければなりません。とても面倒な作業です。試行毎にアイテム数が異なる点も、真面目に作成しようとしたら 第 12 章 のテクニックを用いてアイテム数 5 個のルーチン、10 個のルーチン、15 個のルーチンを使い分けなければいけません。大変な手間です。もっとも「真面目に作成しようとしたら」と断り書きを入れるということは抜け道があるのですが、それはこの後の解説で触れます。

続いて 3. と 4. ですが、これらはいずれも条件ファイルに関わる問題です。今までの章では、無作為に変化するパラメータは条件ファイルで値を設定してきました。条件ファイルを使う場合は、すべてのパラメータの「組み合わせ」を明示的に記述しなければいけません。今回の実験を今までの章のように条件ファイルで作成

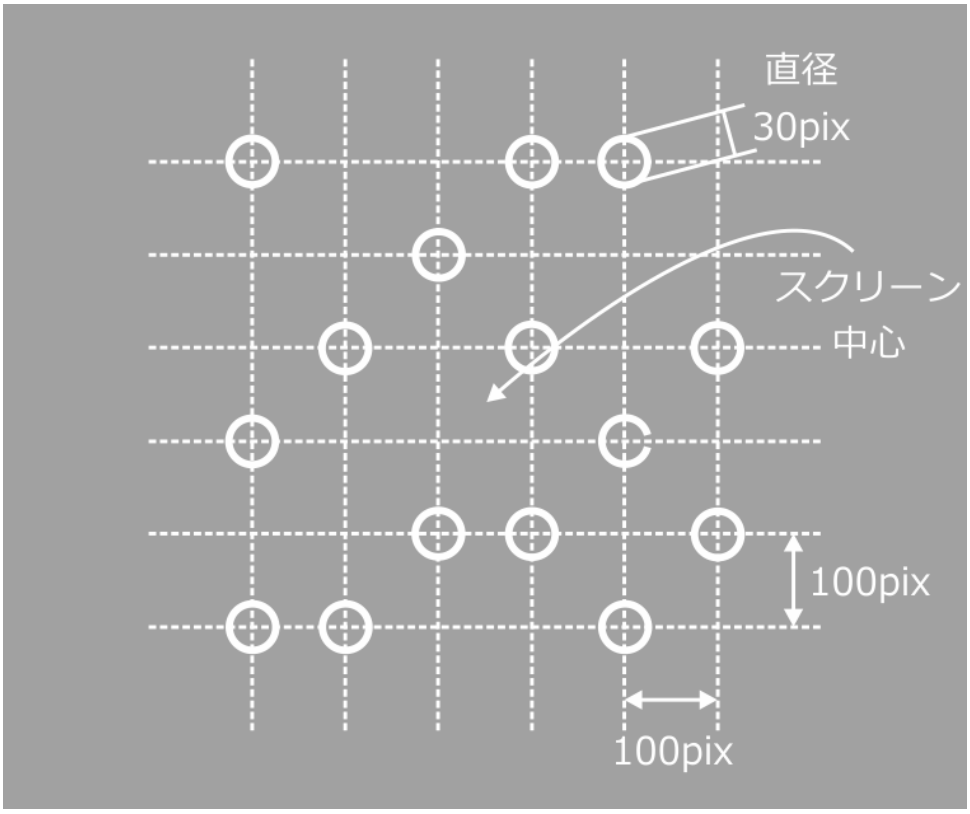


図 13.3　刺激の配置。アイテムの位置は仮想的な 6 × 6 のグリッド上から試行毎に無作為に選択されます。

しようとすると、固視点の提示時間が 3 種類ありますので、12 種類の刺激と掛け合わせて 36 条件の条件ファイルになります。この条件ファイルを使うと実現可能な試行数は 36 の倍数になりますが、240 は 36 で割り切れませんので、この時点で全試行数を 240 試行にすることは不可能になってしまいました。固視点の提示時間を 1.0 秒と 1.5 秒の 2 種類に減らすと 12 種類の刺激との掛け合わせで 24 条件の条件ファイルとなり、240 試行にすることが可能になります。しかし、今回の実験ではアイテムの位置も C の向きもすべて無作為なのです。どう工夫しても、これまでの章の条件ファイルと同様の考え方では、総試行数が 240 試行となる条件ファイルを作成することはできません。

なぜ今回の実験では、今までの章と同じ考え方ではうまくいかないのでしょうか。今までの章では、「無作為」という言葉を使う時に、それは「順番が無作為」というだけでどのパラメータ（の組み合わせ）を何試行行うかが決まっていました。ところが、今回の実験では、パラメータそのものを試行毎に無作為に決めようというのです。結果として全試行を通じて 3 種類の注視時間が選ばれた回数は均等にならないでしょうが、本当に「無作為に」有限回数の選択をしたのであれば、むしろ回数が均等にならないことがある方が普通です。この種の「無作為さ」を実現する方法は Builder には用意されていませんので、Code コンポーネントの力を借りる必要があります。

この章の課題を Builder で実現することの厄介さを何となく感じていただけましたでしょうか。筆者の個人的な考えでは、Builder に慣れていない初級から中級の方がこの章の実験を Builder で作成するのであれば「すべての刺激を画像としてあらかじめ用意する」という方法を検討すべきだと思います。図 13.2 および 図 13.3 に基づいて数百から数千枚程度の刺激画像ファイルを作成しておいて、その中から 240 枚の画像を各条件の試行数を満たすように無作為に抽出した条件ファイルを複数個準備しておいて、参加者毎にことなる条件ファイルを使用して実験を行うのです。しかし、本書は Builder のさまざまなテクニックを紹介することが目的なので、Builder 上で刺激を作成して実験を行う方法を考えたいと思います。

13.2 実験の作成

実験の作成に入りましょう。まず前章までに解説済みのテクニックで作成できる部分を作成します。Builder で新規に実験を作成して以下の作業を行い、exp13proto.psyexp という名前で保存してください。

- 実験設定ダイアログ
  - **[単位]** を pix にする。
- trial ルーチン
  - Code コンポーネントをひとつ配置して、 **[名前]** を code_trial にする。今はコードを入力しない。
  - Text コンポーネントをひとつ配置して、以下のように設定する。
    - **[名前]** を fixpoint にする。
    - **[終了]** を delay にする。
    - **[文字の高さ $]** を 24 にする。
    - **[文字列]** に + と入力する。
  - Keyboard コンポーネントをひとつ配置して、以下のように設定する。
    - **[名前]** を key_resp_trial にする。
    - **[開始]** を delay に、 **[終了]** を空白にする。
    - **[Routine を終了]** がチェックされていることを確認する。
    - **[検出するキー $]** を'right', 'left' とする。
    - **[正答を記録]** をチェックし、 **[正答]** に$correctAns と入力する。
  - Image コンポーネントをひとつ配置して、以下のように設定する。
    - **[名前]** を image00 にする。
    - **[開始]** を delay に、 **[終了]** を空白にする。
    - **[画像]** に imagefile[0] と入力し、「繰り返し毎に更新」設定する。
    - **[回転角度 $]** に ori[0] と入力し、「繰り返し毎に更新」に設定する。
    - **[位置 [x, y] $]** に pos[0] と入力し、「繰り返し毎に更新」に設定する。
    - **[サイズ [w, h] $]** に [30, 30] と入力する。
- rest ルーチン (作成する)
  - フローの trial ルーチンの直前に挿入する。
  - Code コンポーネントをひとつ配置して、 **[名前]** を code_rest にする。今はコードを入力しない。
  - Text コンポーネントをひとつ配置して、以下のように設定する。

    * **[名前]** を textRest にする。
    * **[終了]** を空白にする。
    * **[文字の高さ $]** を 24 にする。
    * **[文字列]** に「準備ができたらカーソルキーの左右どちらか一方を押してください」と入力する。
  - Keyboard コンポーネントをひとつ配置して、以下のように設定する。
    * **[名前]** を key_resp_rest にする。
    * **[終了]** を空白にする。
    * **[Routine を終了]** がチェックされていることを確認する。
    * **[検出するキー $]** を'right', 'left' とする。
    * **[記録]** を「なし」にする。
- trials ループ (作成する)
  - rest ルーチンと trial ルーチンを繰り返すように挿入する。
  - **[繰り返し回数 $]** に 20 と入力する。
  - **[繰り返し条件]** に exp13cnd.xlsx と入力する。先に exp13cnd.xlsx を作成してから「選択…」ボタンをクリックして選択するとよい。
- exp13fi.xlsx(条件ファイル)
  - numItems、firstItem、otherItems、correctAns という名前のパラメータを定義する。
  - numItems は 5、10、15 の 3 種類、firstItem は o.png、c.png の 2 種類、otherItems も o.png と c.png の 2 種類の値をとる。これらの全ての組み合わせを入力する。パラメータ名を定義する行を除いて 3 × 2 × 2=12 行の条件ファイルとなる。
  - firstItem と otherItems が同じ行の correctAns の列に left と入力する。firstItem と otherItems が異なる行の correctAns の列に right と入力する。
- o.png および c.png(刺激画像ファイル)
  - 背景が透過した 30 × 30pix の PNG ファイルを作成し、o.png という名前にする。o.png には白色の円を描く。o.png を c.png という別名で保存し、円の中心右側の 10pix 分を消して切れ目を入れる。

最後に、rest ルーチンに配置してある rest_code の **[フレーム毎]** に以下のコードを入力しておいてください。これは、条件数である 12 の倍数ではない「80 試行毎の休憩」を実現するためのコードです。第 12 章 で紹介した break によるルーチンのスキップが利用されています。

```
if trials.thisN % 80 != 0:
    break
```

第 12 章 の解説が済んでいなかったら、この 80 試行毎に休憩を挿入する方法も「難問」として挙げなければいけないところでした。念のためコードについても解説しておきますと、trials.thisN は 0 から始まって trials ループが 1 回繰り返されるたびに 1 増加します。trials.thisN を 80 で割った剰余が 0 でない時は break するので、剰余が 0 となる実験開始直後、80 試行終了後、160 試行終了後のみ rest ルーチンがスキップされずに実行されます。よろしいでしょうか。それでは作業を進めましょう。まずは trial ルーチンに 1 個だけ配置してある Image コンポーネントを 15 個まで増やす作業です。

13.3 テキストエディタを用いて多数のコンポーネントを追加しよう

今回の実験では、最大 15 個のアイテムをスクリーン上に提示するために、trial ルーチンに 15 個の Image コンポーネントを配置しなければいけません。前節の作業で Image コンポーネントを配置して **[名前]** に image00、**[回転角度 $]** に ori[0]、 **[位置 [x, y] $]** に pos[0] と入力しましたが、Image コンポーネントを追加してこれらの値を image01、ori[1]、pos[1] にし、さらに続いて Image コンポーネントを追加して image02、ori[2]、pos[2]、…としていって、image14、ori[14]、pos[14] に到達するまで作業を繰り返さなければいけません。ただ面倒であるだけでなく、どれかひとつのコンポーネントで **[サイズ [w, h] $]** の値を設定し忘れたりした場合に、ルーチンペイン上にずらっと並んだ Image コンポーネントのどれを修正したらいいか探し出すのはうんざりするほど厄介です。この章では上級編ということで「反則技」を紹介しておこうと思います。

ここで解説する方法を用いるには、UTF-8 の文字コードと LF の改行コードのテキストファイルの編集に対応しているテキストエディタが必要です。UTF-8 と LF の組み合わせは Linux 系 OS では標準的なので、Ubuntu などの Linux で作業している方は、標準でインストールされているテキストエディタ (gedit など) で問題なく編集できます。Microsoft Windows や MacOS X ではオープンソースのテキストエディタをインストールすることで編集が可能になります。非常に多くのエディタがありますが、筆者が愛用しているのは以下のエディタです。

- Visual Studio Code https://azure.microsoft.com/ja-jp/products/visual-studio-code/
- サクラエディタ (Windows 向け) http://sakura-editor.sourceforge.net/

以下の解説では、サクラエディタの画面を例に用います。まず、作業に失敗した時のためにやり直しができるように、exp13proto.psyexp のコピーを作成して exp13.psyexp という名前にしておきましょう。以後の作業は、exp13.psyexp に対して行うものとします。

exp13.psyexp をテキストエディタで開くと、 図 13.4 のように XML 形式というフォーマットで記述された実験の内容が表示されます。XML 形式については「 13.9.1:XML 形式による実験の表現 」を参照してください。

textRest という文字列を検索すると、rest ルーチンに配置した textRest の設定位置へ移動できます。textRest という文字列が見つかった数行下に教示文が適切に表示されていれば、正しい文字コード (UTF-8) で読み込めています。 図 13.4 の一番下のように意味不明な記号や文字が並んでいる場合は、UTF-8 で読み込めていません。このまま編集作業を続けると教示が失われてしまいますので、ファイルを開き直してください。どうしても UTF-8 で開けない場合は、一旦 Builder に戻って日本語などの非 ASCII 文字を全て削除すれば文字化けは解消されます。exp13.psyexp の場合は textRest の **[文字列]** に日本語の文字が入力されているので、ここを一旦空白にしておけば文字化けが解消されます。テキストエディタでの作業を終えてから、教示文を入力し直しましょう。

無事にテキストエディタで exp13.psyexp を開くことができたら、image00 という文字列で検索してください。

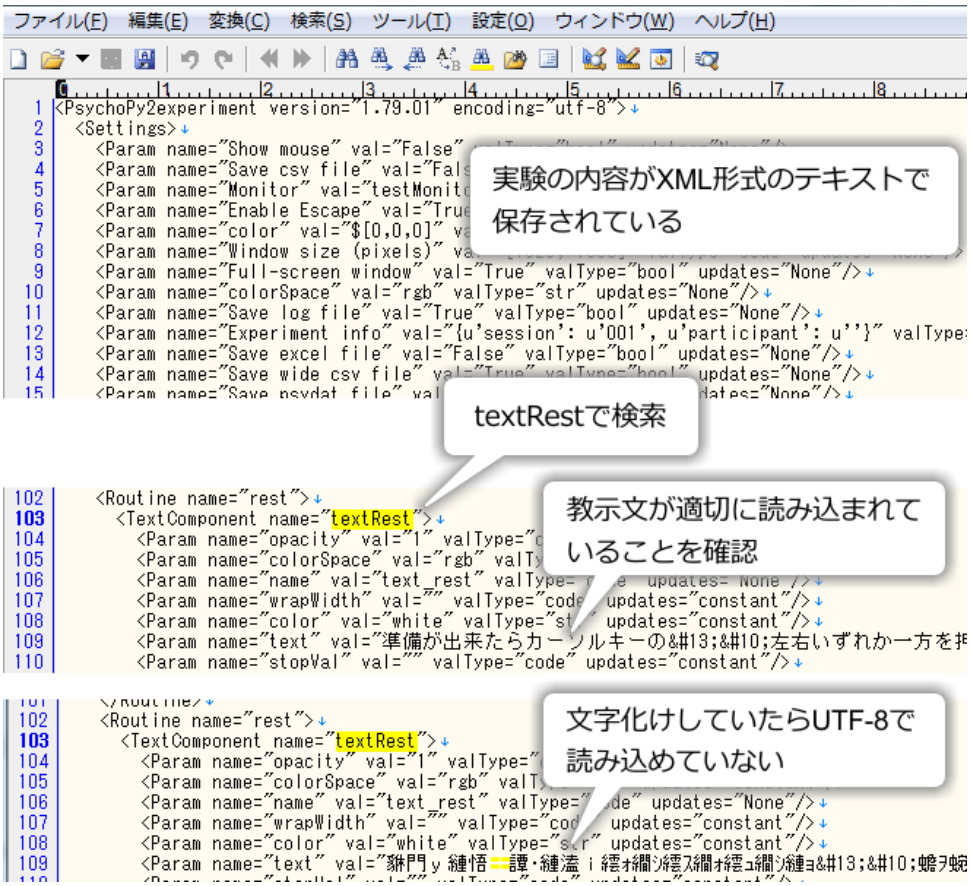


図 13.4　exp13proto.psyexp をテキストエディタで開いた様子。textRest という文字列が見つかった数行下に教示文が適切に表示されていれば、正しい文字コード (UTF-8) で読み込めています。

以下のような部分が見つかるはずです。

```
<ImageComponent name="image00">
    <Param name="opacity" val="1" valType="code"…(略)
     (中略)
    <Param name="name" val="image00" valType="code" …(略)
     (中略)
    <Param name="pos" val="pos[0]" valType="code"…(略)
     (中略)
    <Param name="ori" val="ori[0]" valType="code"…(略)
     (中略)
    <Param name="image" val="$imagefile[0]" valType="str"…(略)
     (中略)
</ImageComponent>
```

これが trial ルーチンに配置した Image コンポーネントの設定です。XML をご存じない方でも、なんとなく Builder 上での Image コンポーネントのプロパティ設定画面との対応が想像できるのではないでしょうか。この部分をコピー＆ペーストして image00、pos[0]、ori[0]、imagefile[0] を書き換えれば、image01、image02、image03…を trial ルーチンに追加することができます。

<ImageComponent name="image00">という行から、</ImageComponent>という行までを選択してコピーしてください。exp13.psyexp では Image コンポーネントはひとつしか配置されていないので間違えようがありませんが、同種類のコンポーネントが複数個配置されている psyexp ファイルでこのテクニックを使う場合のために、コピー範囲の判断方法を説明しておきます。psyexp ファイルでは、コンポーネントやルーチンなどの要素が定義されている範囲が字下げで判断できるようになっています (図 13.5)。Python の for 文や if 文の文法と似ていますが、Python と違って最初に見つけた同じ字下げの行を要素に含む点に注意してください。コピーしたら、作業用に新しいテキストファイルを開いて 14 回繰り返して貼りつけてください。貼りつけたら exp13tmp.txt という名前で保存しておきましょう。

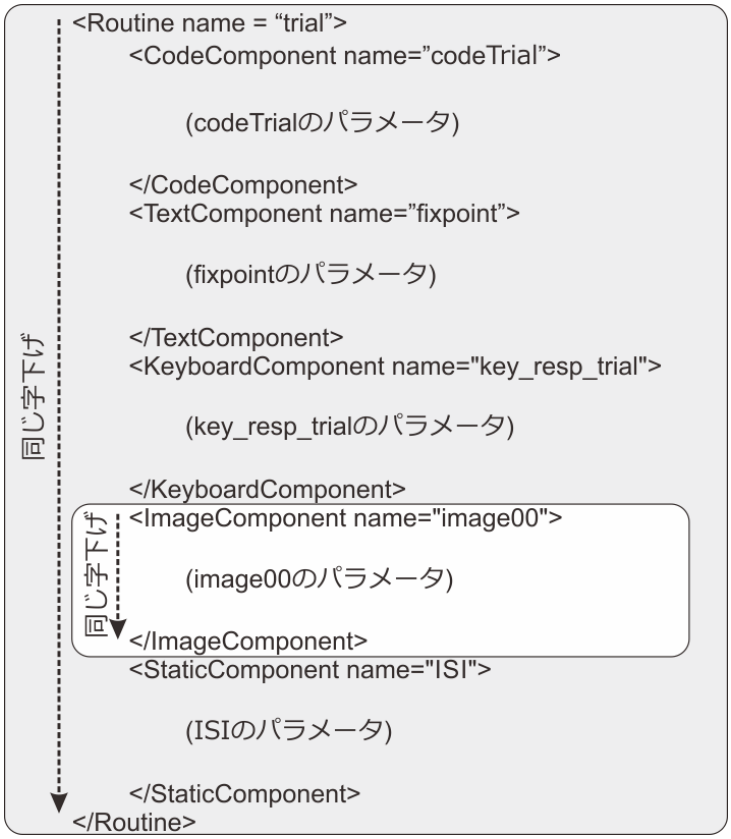


図 13.5　コピー範囲の判断。白い四角形の部分がコピーする範囲です。

exp13tmp.txt を保存したら、exp13tmp.txt のパラメータの設定を上から順番に書き換えていきましょう (図 13.6)。ちょっと面倒ですが、Builder 上で 14 個の Image コンポーネントを追加することを考えたら楽なのではないかと思います。特に何番目のコンポーネントまで書き換えたか、飛ばしているコンポーネントはないかなどが一目でわかるのはとても快適です。

- image00 を image01、image02、…、image14 に書き換える。ひとつの Image コンポーネントに付き image00 は二カ所あるので注意すること。
- pos[0] を pos[1]、pos[2]、…、pos[14] に書き換える。
- ori[0] を ori[1]、ori[2]、…、ori[14] に書き換える。
- imagefile[0] を imagefile[1]、imagefile[2]、…、imagefile[14] に書き換える。

書き換えが終了したら、exp13tmp.txt を元の exp13.psyexp に貼りつけます。貼り付ける場所は、image00 の定義のすぐ後ろです (図 13.7)。貼りつけたら exp13.psyexp を保存して、Builder から開いてみてください。trial

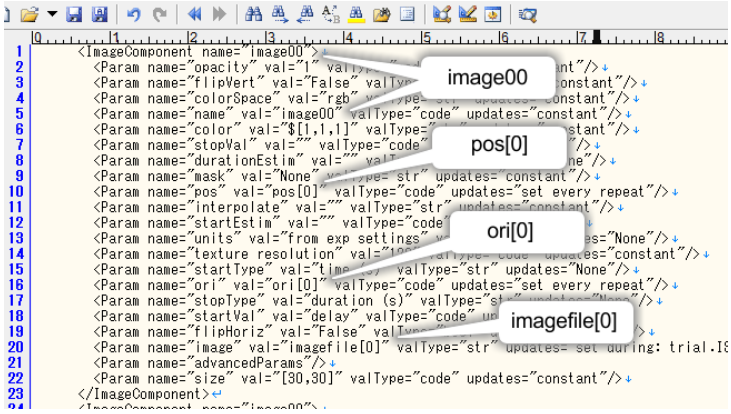


図 13.6　Image コンポーネントの設定を書き換えます。

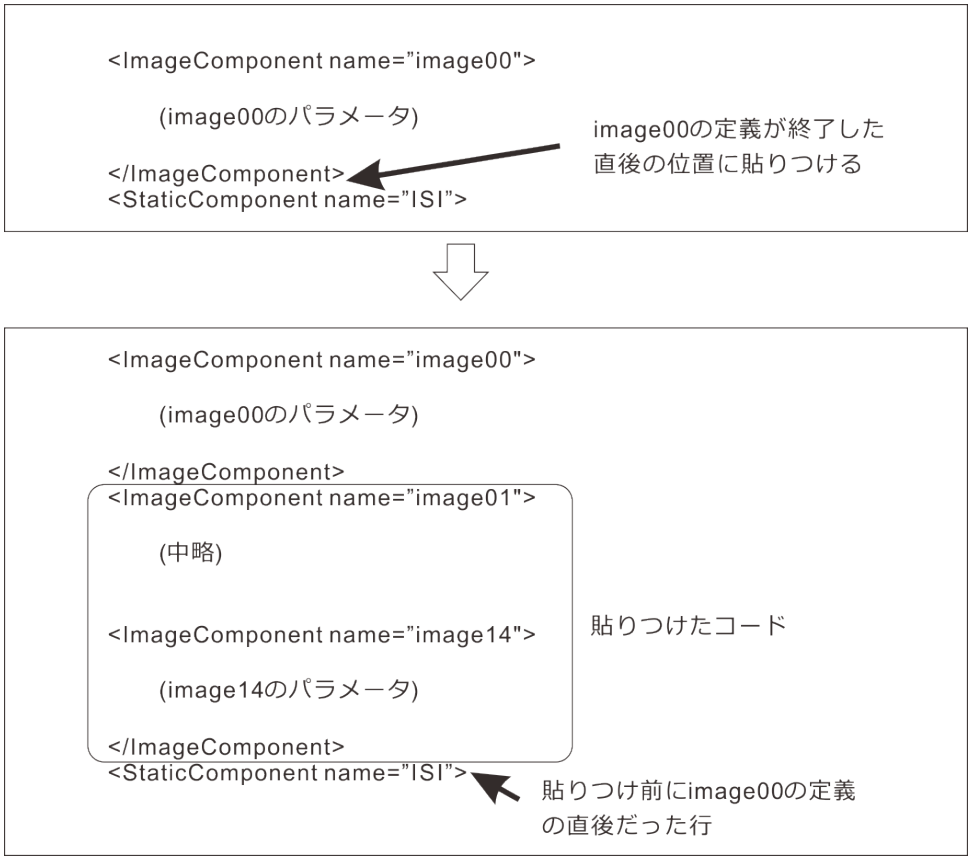


図 13.7　編集したコードの貼り付け。元ファイルの image00 の定義の直後へ挿入するように貼りつけます。

ルーチンに image00 から image14 までの 15 個の Image コンポーネントが配置されているはずです。適当にいずれかの Image コンポーネントをクリックして、image03 ならパラメータが pos[3]、ori[3]、imagefile[3] といった具合に **[名前]** の番号とパラメータに含まれるリストのインデックスの値が一致することを確認してください。成功したら、もう exp13tmp.txt は削除していただいても構いません。失敗した場合は、exp13tmp.txt に誤りがないか、exp13.psyexp へのコピー位置が正しいかをよく確認して作業をやりなおしてください。

これで 15 個の Image コンポーネントを追加するという問題をクリアすることができました。恐らく Python のスクリプトを書ける方は「ここまでするくらいなら全部 Coder でコードを書いた方がいい」とお考えだと思います。実際、筆者もこの実験を作成するなら直接コードを書きます。しかし、共同研究者や指導学生などが

Builder を使うのならば、psyexp ファイルの構造は知っておいた方が良いです。何らかの理由で psyexp ファイルが読み込めなくなった時に、ファイルを開くことさえできれば手作業で編集したり、問題がない部分だけを別ファイルに抽出したりすることができます。

それでは続いて、試行毎に無作為にアイテムの位置と固視点の提示時間を変更する方法を考えましょう。

チェックリスト

- テキストエディタを用いて適切な改行コード、文字コードで psyexp ファイルを開くことができる。
- psyexp ファイルをテキストエディタで開いて **[名前]** の値を検索して、コンポーネントのパラメータの定義を探し出すことができる。
- psyexp ファイルをテキストエディタで編集して、コンポーネントのパラメータを変更することができる。
- psyexp ファイルをテキストエディタで編集して、Builder で配置したコンポーネントをコピーして個数を増やすことができる。

13.4 Code コンポーネントを使って無作為に固視点の提示時間を選択しよう

これで必要なコンポーネントをすべてルーチン上に配置することができましたので、コードを入力していきましょう。以下の処理をコードで実現する必要があります。

- trial ルーチンで使われている変数 delay の値を決定する。固視点の **[開始]** が 0 で **[終了]** が delay に設定され、image00 から image14 の **[開始]** が delay に設定されているので、固視点が出現した delay 秒後に固視点が消失して代わりに image00 から image14 が提示される。
- ori[0] から ori[14] の値を決定する。値は試行毎に 0、90、180、270 から無作為に選択する。
- pos[0] から pos[14] の値を決定する。値は試行毎に 図 13.3 に示したグリッドから重複がないように無作為に選ぶ。
- imagefile[0] から imagefile[14] の値を決定する。この値の決め方については後述する。

これらの問題はいずれも「無作為に選択する」という点で共通しているので、同じ方法で解決できます。しかし、delay の決定以外は「アイテム数が 5 個、10 個、15 個と変化することにどう対応するか」という問題と併せて考えないといけないので、次節でまとめて考えることにしましょう。まずは delay の問題を解決します。

無作為に値を選択するには、Builder が内部で用意している乱数関数 (表 13.1) を用います。使い方は非常に単純で、randint(0,5) と書けば 0 から 4 の整数の一様乱数からサンプルをひとつ得ることができます。引数 high「未満」ですから 5 を含まない点に注意してください。同様に、normal(50, 10) と書けば平均値 50、標準偏差 10 の正規乱数からサンプルをひとつ得ることができます。引数 size は、randint(0, 5, size=3) のように使用します。この例の場合、戻り値は [0, 2, 1] といった具合に 0 から 4 の整数の一様乱数からサンプルを 3 つ並べたシーケンス型データが得られます。正確に書くとこの戻り値は numpy.ndarray クラスのインスタンスなのですが、numpy.ndarray について説明すると脱線が長くなるので「 13.9.2:numpy.ndarray 型について 」を参照してください。

表 13.1　Builder で利用できる乱数関数。いずれも numpy.random から import されています。

random(size = None)	0.0 以上 1.0 未満の一様乱数のサンプルを返す。size が None の時 (初期値) にはひとつのサンプルを、自然数の場合には size 個のサンプルを返す。
randint(low, high, size = None)	low 以上 high 未満の範囲の整数の一様乱数のサンプルを返す。low と high は整数でなければならない。high が None の時には 0 以上 low 未満の範囲と見なされる。size の働きは random( ) と同様。
normal(loc=0.0, scale=1.0, size = None)	正規分布に従う乱数のサンプルを返す。loc は平均値、scale は標準偏差に対応する。size の働きは random( ) と同様。
shuffle(x)	リストなどの要素を変更可能なシーケンス型データの要素を無作為に並べ替える。戻り値はない。x の元の順序は失われてしまう点に注意。

さて、今回の実験のように、複数個の選択肢からひとつを無作為に選びだすという用途には、randint( ) が便利です。delay の値は 1.0、1.5、2.0 の 3 通りです。randint(0, 3) とすれば戻り値として 0、1、2 の乱数が得られますから、戻り値に 0.5 を掛ければ 0.0、0.5、1.0 の乱数が得られます。ここへさらに 1.0 を加えると、1.0、1.5、2.0 の乱数が得られます。従って、以下のコードで delay の値に 1.0、1.5、2.0 の中から無作為にひとつ選んで設定できます。

```
delay = randint(0, 3)*0.5 + 1.0
```

exp13.psyexp の trial ルーチンを開いて、codeTrial にこの式を入力しましょう。試行毎に delay の値を変化させるのですから、入力すべき場所は **[Routine 開始時]** です。これで delay の値については解決しました。

あまりにもあっさり解決してしまったので、無作為に値を選択する方法についてもう少し考えてみましょう。今回の delay は randint( ) の戻り値から簡単な計算で得ることができましたが、「u' 一致'、u' 不一致' のいずれか一方を無作為に選ぶ」という具合に計算で得ることができない値から選択しないといけない場合はどうすればいいでしょうか。この章まで学んできた人ならピンと来るかもしれません。選択肢のリストを作成して、リストのインデックスに randint( ) の戻り値を使えばよいのです。変数 tasktype に u' 一致' または u' 不一致' という文字列のいずれかを無作為に選んで設定するのであれば、例えば以下のようなコードで実現することができるでしょう。

```
tasklist = [u'一致', u'不一致']
tasktype = tasklist[ randint(0, 2) ]
```

変数 tasklist は実験の最初に一回だけ作成すればよいので、Builder でこのコードを使用する場合は tasklist = [u' 一致', u' 不一致'] は **[実験開始時]** に書いておいて、tasktype = tasklist[randint(0, 2)] を該当するルーチンの **[Routine 開始時]** に書くとよいでしょう。この方法はもちろん今回の delay のように、数値をひとつ選択する場合にも使えます。

```
delaylist = [1.0, 1.5, 2.0]
delay = delaylist[ randint(0, 3) ]
```

最初に紹介した、計算によって delay の値を得る方法の場合は、 **[Routine 開始時]** にだけコードを入力すれば済みますが、数か月後に自分が作成した実験を読み直さないといけなくなったときなどに少々わかりにくいかもしれません。リストを用意する方法は、 **[実験開始時]** と **[Routine 開始時]** にコードが分散するという欠点がありますが、後で読み返す時には「ああ、delaylist から値を一つ無作為に取り出しているんだな」という事がわかりやすいかも知れません。どちらの方法を使っていただいても結構です。以上でこの節の解説は終わりですが、最後にひとつ補足しておきます。web 上で Python の乱数について検索すると、randint(low, high) は「low 以上 high『以下』の値を返す」と書かれている資料がヒットするかもしれません。非常に紛らわしいのですが、このような資料で紹介されている randint( ) と、Builder が内部で参照している乱数関数の randint( ) とは全く別の関数です。「low 以上 high『以下』の値を返す」randint( ) は、Python の random モジュールから import されています。ですから、モジュール名を省略せずに書けば random.randint( ) という関数です。一方、Builder が内部で準備している randint( ) は numpy というパッケージのサブモジュール numpy.random から import されています。省略せずに書けば numpy.random.randint( ) です。気を付けてください。

チェックリスト

- low 以上 high 未満の整数を範囲とする一様乱数のサンプルをひとつ得るコードを書くことができる。(low、high は整数)
- 整数の一様乱数を用いて、試行毎に複数の値のリストからひとつの値を無作為に選択するコードを書くことができる。

13.5 Code コンポーネントを使って無作為にアイテムの各パラメータを決めよう

delay の問題が解決したので、続いてアイテムの個数、種類、位置、回転角度を決める方法について考えましょう。一度にすべてのパラメータについて考えるのは大変なので、まずは「アイテムが 15 個の場合」に限定して考えます。

まず、簡単に解決できるのが回転角度の決定です。15 個の Image コンポーネントの **[回転角度 $]** は、ori[0]、ori[1]、…、ori[14] というリストの値がすでに入力されています。ですから、ori という要素数 15 のリストを作成して、要素に 0、90、180、270 のいずれかの値を無作為に割り当てればよいだけです。どうせ毎試行 ori の値は更新するので、最初に ori を作成する時には値は何を設定しても構いません。例えば以下のように 0 を 15 個並べたリストを作成してもよいでしょう。

```
ori = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

別にこのコードで全く問題はないのですが、もし要素数が 100 個必要になった場合、100 個も 0 を並べたリストを入力するのは面倒です。そのような時に便利な関数が range( ) です。range( ) は 1 個から 3 個の整数を引数として取ることができます。引数が 1 個の場合は、0 から引数より 1 小さい整数まで順番に取り出すことができる「range オブジェクト」というオブジェクトが得られます。range オブジェクトの詳細については「 13.9.3:range( ) オブジェクトについて 」をご覧ください。range() を list オブジェクトを作る関数 list( ) に渡

すと、0 から引数より 1 小さい整数までを並べたリストが得られます。例えば list(range(15)) とした場合、以下のリストが戻り値として得られます。

```
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]
```

引数が 2 個 (x, y とします) の場合は、x から y-1 までの整数を取り出す range オブジェクトが得られます。例えば list(range(10, 15)) を実行すると以下のリストが得られます。

```
[10,11,12,13,14]
```

引数が 3 個の場合、整数が 1 ずつ増加するのではなく 3 個目の引数の値ずつ増加します。list(range(0,15,3)) を実行すると、以下のように 3 ずつ増加する整数のリストが得られます。15 は含まない点に注意してください。

```
[0,3,6,9,12]
```

3 個目の引数が負の場合は、だんだん値が小さくなっていきます。list(range(15,0,-3) を実行すると、以下のように 15 から始まって 3 ずつ減少するリストが得られます。上の例と同様に 0 は含まない点に注意してください。

```
[15, 12, 9, 6, 3]
```

range( ) が本領を発揮するのは、list( ) ではなく for 文と組み合わせるときです。for 文と組み合わせると、ブロックが繰り返される毎に range( ) から順番に値が取り出されるので、要素数が等しい複数のリストに対してまとめて処理を行うコードが簡単に書けます。まず、0 が 15 個並んだリストを作成してみましょう。

```
ori = [ ]
for i in range(15):
    ori.append(0)
```

これで 0 が 15 個並んだリストが変数 ori に格納されましたが、これだけでしたら先ほどのようにずらっと 0 を 15 個並べたほうが楽だと思われるかもしれませんね。ここへ、各アイテムの種類 (O か C か) を格納する変数 imagefile を準備する処理も組み込んでみましょう。imagefile は Image コンポーネントので使用されますので、その要素は画像ファイル名でなければいけません。こちらも試行毎に値を変更するのでとりあえず O と C のどちらを設定しておいても構いません。とりあえず O で初期化しておくことにしましょう。O の刺激は o.png と画像ファイルに対応していますので、'o.png' という文字列を 15 個並べたリストを作成する必要があります。ori を作成した時と同じ要領で考えると、以下のコードで実現できます。

```
ori = [ ]
imagefile = [ ]
for i in range(15):
    ori.append(0)
    imagefile.append('o.png')
```

刺激の位置を格納する変数 pos の準備もここへ組み込むことができます。要素は **[位置 [x, y] $]** で使用するので、要素数 2 のリスト [0, 0] で初期化しておきましょう。

```
ori = [ ]
imagefile = [ ]
pos = [ ]
for i in range(15):
    ori.append(0)
    imagefile.append('o.png')
    pos.append([0, 0])
```

以上で ori、imagefile、pos の準備は完了です。このコードは実験開始時に一回実行すればよいので、 **[実験開始時]** に入力しておきましょう。

変数の準備ができたので、続いて各試行の最初に無作為にこれらの変数の値を決定するコードを作成しましょう。まず、ori については delay と同じ方法が使えます。Randint(0, 4) で 0 から 3 の整数の乱数を得て、90 倍すれば 0、90、180、270 の乱数が得られますので、for 文で ori[0] から ori[14] に代入すればいいでしょう。この方法では回転させる必要がない O も回転させてしまいますが、O は回転させても見た目が同じなので実質的に問題とはなりません。以下のコードを **[Routine 開始時]** に入力して下さい。

```
for i in range(15):
    ori[i] = 90*randint(0,4)
```

続いて imagefile の設定ですが、こちらは少し解説が必要です。すべて O の条件、すべて C の条件、O の中にひとつだけ C の条件、C の中にひとつだけ O の条件の 4 条件があるのでした。そして、条件ファイルを確認には firstItem と otherItems というパラメータが定義されています。firstItem は imagefile[0]、otherItems は imagefile[1] から imagefile[14] に設定することを想定しています。 図 13.8 をご覧ください。firstItem と otherItems がともに o.png であれば「すべて O」の条件に、ともに c.png であれば「すべて C」の条件になります。同様に firstItem が c.png で otherItems が o.png であれば「O の中にひとつだけ C」、firstItemg が o.png で otherItems が c.png であれば「C の中にひとつだけ O」になります。「必ず image00 ターゲットになっても問題はないの？」と思われる方がおられるかも知れませんが、試行毎に位置を無作為に決定するので問題ありません。

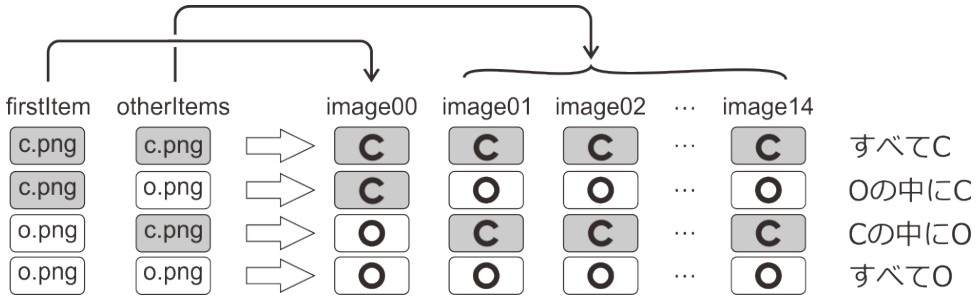


図 13.8　firstItem と otherItems のパラメータ値と刺激条件との対応。

先ほど **[Routine 開始時]** に入力した ori を更新する for 文に、imagefile を更新するコードを追加しましょう。変数 i の値が 0 の時には imagefile[i] に firstItem の値を設定し、i の値が 0 以外の時には imagefile[i] に otherItems の値を設定します。

```
for i in range(15):
    ori[i] = 90*randint(0,4)
    if i==0:
        imagefile[i] = firstItem
    else:
        imagefile[i] = otherItems
```

ori、imagefile の更新ができたので、あとはアイテムの位置に対応する変数 pos の更新です。pos は「無作為に値を決定する」という点では ori と同じですが、重要な違いがあります。ori はアイテム間で値が重複しても構いません。つまり、例えば同時に回転角度が 90 度のアイテムが複数個存在しても構いません。一方、pos はアイテム間で値が重複するとアイテムが重なってしまいますので、値の重複は許されません。次の節では、pos の値を決定する方法を考えます。

チェックリスト

- range( ) を用いて、0 から n (n>0) までの整数を順番に取り出す range オブジェクトを作成することができる。
- range( ) を用いて、m から n (n>m) までの整数を順番に取り出す range オブジェクトを作成することができる。
- range( ) を用いて、m から n まで、s の間隔で整数を順番に取り出す range オブジェクトを作成することができる。ただし m、n は互いに異なる整数、s は非 0 の整数である。

13.6 無作為に重複なく選択しよう

それでは改めて、pos の値の決定方法を考えましょう。pos の候補となる値は 図 13.3 に示したグリッドの座標で、36 個あります。これら 36 個の値の中から 15 個を重複なく無作為に選択しなければいけません。心理学実験においては、この例のように複数個の値を重複なく無作為に選択しなければいけないことがよくあります。このようなときは、「無作為に選択する」のではなく「無作為に並べ替える」という方法が有効です。

まず、36 個の座標値をすべて並べたリストを作成して poslist という変数に格納しておきましょう。以下のように for 文を重ねると簡単に作成できます。append している行の式については、グリッドの間隔が 100pix で一番左下の座標が [-250, -250] だったことを思い出してください。この多重 for 文を実行したときに poslist に値が追加されていく様子を 図 13.9 に示しましたので、多重 for 文の動作がイメージしにくい方は参考にしてください。この多重 for 文を codeTrial の **[実験開始時]** に追加してください。

```
poslist = [ ]
for pos_y in range(6):
    for pos_x in range(6):
        poslist.append([100*pos_x-250, 100*pos_y-250])
```

続いて、作成した poslist の要素を無作為な順序に並び替えます。並び替えには 表 13.1 で紹介した shuffle( ) を用います。shuffle( ) は引数として受け取ったリストの要素の順番を無作為に並び替えます。戻り値は返さな

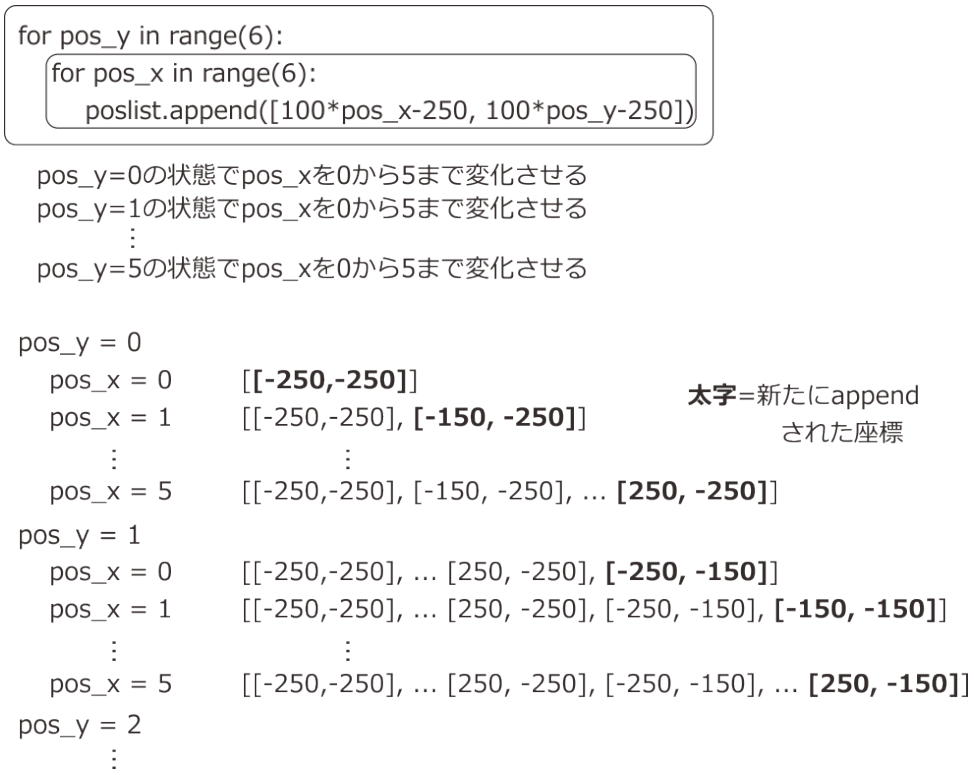


図 13.9　多重 for 文による座標値リストの作成。

いので、ただ shuffle(poslist) と書けば poslist の要素を並び替えることができます。試行毎にアイテムの位置を変更したいので、 **[Routine 開始時]** に記入してください。

さて、ここからがポイントです。poslist の要素はルーチンの開始時に無作為に並び替えられているのですから、poslist の先頭から順番に 15 個の要素を取りだせば、poslist の中から重複なしに無作為に 15 個の要素を取り出したことになります。従って、以下のコードで pos[0] から pos[14] に重複なく無作為に位置を割り当てることができるはずです。

```
for i in range(15):
    pos[i] = poslist[i]
```

for i in range(15): という繰り返しは先ほど ori と imagefile の値を設定する時にも使用したのですから、以下のように ori と imagefile の設定を合わせて行うことができます。確認のため、delay の設定や shuffle も含めた **[Routine 開始時]** 全体のコードを示しておきます。

```
delay = 0.5*(randint(0,3))+1
shuffle(poslist)
for i in range(15):
    ori[i] = 90*randint(0,4)
    pos[i] = poslist[i]
    if i==0:
        imagefile[i] = firstItem
    else:
```

(次のページに続く)

(前のページからの続き)

```
        imagefile[i] = otherItems
```

これで「アイテム数が 15 個の場合」に限定した条件での実験が完成しました。一度 exp13.psyexp を保存して実行してみましょう。常にアイテムが 15 個提示されてしまいますが、アイテムの位置や向きが試行毎に無作為に変化していることが確認できます。これで残りはアイテムの個数を numItems パラメータの値に従って変化させるだけです。

チェックリスト

- リストの要素を無作為に並べ替えることができる。
- m 個の要素を持つリストから、n 個の要素 (m>n) を重複なく無作為に抽出することができる。

13.7 アイテムの個数を可変にしよう

いよいよ最終段階、numItems の値に従って試行毎にアイテム数を変化させる問題に取り組みましょう。いろいろな方法が考えられるのですが、ここでは簡単に実現できる「スクリーン外にアイテムを配置する」という方法を紹介します。今まで自分で実験を作成していて、刺激の単位が norm になっているのに pix のつもりで **[位置 [x, y] $]** に [500,0] などと指定して、刺激がスクリーンに描画されずに困ったことはないでしょうか。スクリーンの上下左右の限界から大きくはみ出た位置を指定してもエラーにならないせいでこういった困ったことが生じるのですが、今回はこれがエラーにならない事を逆手に取ります。 **[実験開始時]** 入力済みのコードのうち、アイテムの位置を決定する処理だけを抜き出してみましょう。

```
for i in range(15):
    pos[i] = poslist[i]
```

ここへ if 文を追加して、i が numItems 未満の時は上記と同様の処理、i が numItems 以上の時はスクリーンの描画範囲を超えた位置を設定する処理を行うように変更します。

```
for i in range(15):
    if i<numItems:
        pos[i] = poslist[i]
    else:
        pos[i] = [10000, 10000]
```

if 文の条件式が i<numItems であって、i<=numItems ではないことに注意してください。Python においてリストのインデックスは 0 から数えますので、5 個のアイテムを表示するときには 0、1、2、3、4 番目の合計 5 個のアイテムに poslist の値を設定する必要があります。同様に、n 個のアイテムを表示するためには 0 から n-1 番目までのアイテムに poslist の値を設定しなければいけません。if 文の条件式が i<=numItems だと、0 から numItems 番目までの numItems+1 個のアイテムに poslist の値が設定されてしまいます。

今回の例では、スクリーン外に置いて描画しないようにしたいアイテムに [10000, 10000] という位置を指定しています。時代と共にモニターの高解像度化が進んでいますが、スクリーン中央から右に 10000 ピクセル、上に 10000 ピクセルの位置の刺激が描画できるモニターが登場するのはまだまだ先のことでしょう。何より、

今回の実験は単位を pix にして作成していますので、[10000, 10000] が描画範囲に含まれるほどの高解像度モニターでこの章の実験を実行すると、刺激が小さすぎてまともな実験にならないでしょう。

なお、アイテムを描画しないようにさせるには、今回のようにスクリーンの描画範囲外の位置を指定するという方法の他にも、**[不透明度 $]** を 0.0 にして完全な透明にしてしまうという方法もあります。ただし、透明化する方法の場合は、あくまで描画されていないだけで PsychoPy にとってはその位置に刺激があると認識されますので、第 8 章 で紹介した contains( ) や overlaps( ) を使う時に注意する必要があります。刺激がそこに存在していないように見えるのに、マウスカーソルが「刺激の上に重なっている」と判定されてしまうなどの恐れがあるからです。もっとも、この問題ですら「実験参加者がスクリーン上のある領域にマウスカーソルを置いているか否か、参加者に悟られないように記録する」という用途にも使えますので、一概に不備だとは言えません。こういった一見不備に思える現象を積極的に利用することによって、Builder で実現できる実験の幅は飛躍的に広がります。ぜひ、いろいろと工夫していただきたいと思います。

さて、上記のコードを trial ルーチンの **[実験開始時]** に組み込んだ、最終版のコードを以下に記します。pos[i] への代入部分が変化したことを確認してください。

```
delay = 0.5*(randint(0,3))+1
shuffle(poslist)
for i in range(15):
    ori[i] = 90*randint(0,4)
    if i<numItems:
        pos[i] = poslist[i]
    else:
        pos[i] = [10000, 10000]
    if i==0:
        imagefile[i] = firstItem
    else:
        imagefile[i] = otherItems
```

exp13.psyexp に上記の変更を加えたら、exp13.psyexp を保存して実行してみましょう。今度は試行毎に無作為な順番にアイテム数が 5 個、10 個、15 個と変化することを確認してください。十数試行ほどスクリーンに描画されたアイテム数をメモして Escape キーを押して実験を中断し、描画されたアイテム数と trial-by-trial 記録ファイルに出力された numItems の値が一致していることも確認しましょう。これで今回の目標はすべて達成できました。

最後に、後の分析でアイテム位置の情報が必要になった場合に備えて、アイテム位置を保持している変数 poslist の値を実験記録ファイルに出力する処理を付け加えておきましょう。独自の変数の値を出力する方法についてはすでに 第 7 章 で解説しましたので、出力自体はもう皆さん解説なしでできると思います。ただ、poslist は要素数 36 のリストである一方、後の分析で実際に必要となる可能性がある要素は実際にスクリーン上に提示された刺激の位置に対応する要素のみです。言い換えると、各試行で先頭から numItems 個の要素のみが必要です。何の工夫もせずに poslist を addData( ) メソッドに渡してしまうと、36 個全部が出力されてしまうため、分析時に不必要な値を除去しなければならず、非常に無駄です。必要な値だけを抜き出して出力するのが理想的です。

for 文を用いると、リストの先頭から numItems 個の要素を取り出したリストを作成するのは簡単です。例えば以下のコードのようにすれば変数 displayed Pos に実際に提示に利用された位置をまとめることができるで

しょう。

```
displayedPos = []
for i in range(numItems):
    displayedPos.append(poslist[i])
```

if 文や for 文の利用はプログラミングの基本中の基本なので、こういったコードがぱっと頭に浮かぶようにしっかりとこれらの文に慣れて欲しいと思います。しかし、今回の用途に関しては Python にスライスと呼ばれる非常に便利な機能がありますので、そちらもぜひ覚えて欲しいと思います。

スライスとは、リストやタプルなどのシーケンス型のデータから、連続する要素を抜き出す演算です。シーケンス型データが格納された変数 var に対して var[a : b] の書式で用い、インデックス a からインデックス b の間に含まれる要素を抜き出したリストを返します。a と b の間の記号は半角のコロンです。第 8 章 で用いたリストの例をもう一度使って解説しましょう。 図 13.10 例 1 をご覧ください。[100, 200, 300, 400, 500, 600] というリストを格納した変数 var があります。正のインデックスは、先頭から順番にそれぞれの要素の「前」にあると考えます。var[1:4] と書くと、インデックス 1 からインデックス 4 までの間の要素を取り出すのですから、[200, 300, 400] が得られます。初心者の方によくある勘違いに、スライスを「a 番目の要素から b 番目の要素を抜き出す」と考えてしまうというものがあります。「var[1] が 200、var[4] が 500 ですから、var[1:4] は [200, 300, 400, 500] じゃないの？」というのがこの勘違いの典型です。飽くまで 4 というインデックスは 500 の前にあり、var[1:4] というスライスは「インデックス 1 からインデックス 4 までの間の要素を抜き出す」のですから、500 は含まれません。

リストから要素をひとつ取り出す時に負のインデックスを利用できたのと同様に、スライスでも負のインデックスを用いることができます (図 13.10 例 2)。正と負のインデックスを混ぜて使うこともできます。ただし、var[a:b] の a の方が b よりもリストの前方でなければいけません。 図 13.10 例 3 つめの例のように、a が省略された時には、先頭から抜き出されます。 図 13.10 例 4 のように b が省略された時には、末尾までを抜き出します。

このスライスを利用すれば、poslist から実験記録ファイルに出力すべき要素を抜き出したリストを簡単に作ることができます。poslist の先頭から numItems 個の要素を刺激提示に使ったのですから、poslist[:numItems] とすればよいだけです (コロンの前は省略している点に注意)。このリストを実際に実験記録ファイルに出力するコードを書くのは練習問題としましょう。

チェックリスト

- ルーチンに配置された視覚刺激コンポーネントをスクリーン上に描画させないようにすることができる。
- スライスを用いて、あるリストから連続する要素を抽出したリストを作り出すことができる。
- リストの先頭から要素を抽出する場合のスライスの省略記法を用いることができる。
- リストの末尾までの要素を抽出する場合のスライスの省略記法を用いることができる。

図 13.10　スライスによるリスト要素の抽出。var[a:b] と書くと、変数 var のインデックス a からインデックス b の間にある要素を抜き出します。a が省略されたときは先頭が、b が省略されたときは末尾が指定されたものとします。

13.8 練習問題：透明化によるアイテム数変更と無作為な位置の調整をおこなおう

exp13.psyexp を改造して、この章の解説で出てきた二つのテクニックを実際に試してみてください。さらに、特にアイテム数が 15 個の時に、アイテムが無作為に配置されているというよりは整然と並んでいるように見えてしまうことを防ぐために、アイテムの位置を無作為に上下にずらす処理も追加してください。

- **[不透明度$]** を 0.0 にすることによって numItems 個のアイテムがスクリーンに描画されるようする。
- アイテムの位置を実験記録ファイルに出力するコードを完成させる。
- アイテムの位置を、変数 pos によって指定された位置から上下方向、左右方向ともに-15、-5、5 または 15pix ずらす。ずらす量は試行毎、アイテム毎、方向毎に無作為に決定する。

13.9 この章のトピックス

13.9.1 XML 形式による実験の表現

XML とは Extensible Markup Language の略で、マークアップ言語と呼ばれる言語のひとつです。タグと呼ばれる記号を用いて文書やデータの構造を記述することができます。インターネットの web ページ等を作成したことがある人は HTML をご存知のことと思います。XML のタグは HTML と似ていますが、HTML と異なり自由にタグを定義して使用することができます。

XML の詳細については文献が大量にありますのでそちらを参照していただくとして、psyexp ファイルを読むのに最小限必要なことだけを解説します。XML 文書において、半角のアングルブラケット (山括弧：< >) で囲まれた文字列を「タグ」と呼びます。例えば<Routine>はタグで、Routine というのがタグの名前です。タグは必ず「開始タグ」と「終了タグ」を組み合わせて使用します。Routin の開始タグは<Routine>、終了タグは</Routine>といった具合に、終了タグにはタグ名の前に/が付きます。なお、<Routine/>という具合にタグ名の最後に/が付いているものを「空要素タグ」と呼びます。空要素タグについては後で説明します。

タグの中に、タグ名に続いて Python における変数の代入のような記述が続けて書かれている場合があります。具体的には<Routine name="trial">といった具合です。この例において、name="trial"を Routine タグの「属性」と呼びます。name が属性の名前で、"trial"がその値です。

タグは、開始タグと終了タグの間に他のタグを含むことができます。以下の例では、Routine タグの間に CodeComponent というタグが含まれています。この例において、Routine タグは CodeComponent タグの「親」、CodeComponent タグは Routine タグの「子」と呼びます。XML 文書では、このようにタグを入れ子構造にして、さまざまなデータや文書の構造を記述します。なお、Builder が作る XML ファイルは Python のコードのように字下げされていますが、Python と異なり字下げは必須ではありません。

```
<Routine name="trial">
  <CodeComponent name="code_trial">
  </CodeComponent>
</Routine>
```

exp13proto.psyexp の Routine タグを眺めていると、その要素としてルーチン内に配置したコンポーネントに対応するタグが並んでいることがわかると思います。さらにコンポーネントに対応するタグの要素を確認すると、先ほど述べた「空要素タグ」が見つかります。以下はその例です。

<Param name="opacity" val="1" valType="code" updates="constant"/>

空要素タグは、子となるタグを持ちません。空要素タグは他のタグを挟み込む必要がないので、単独で使用します。

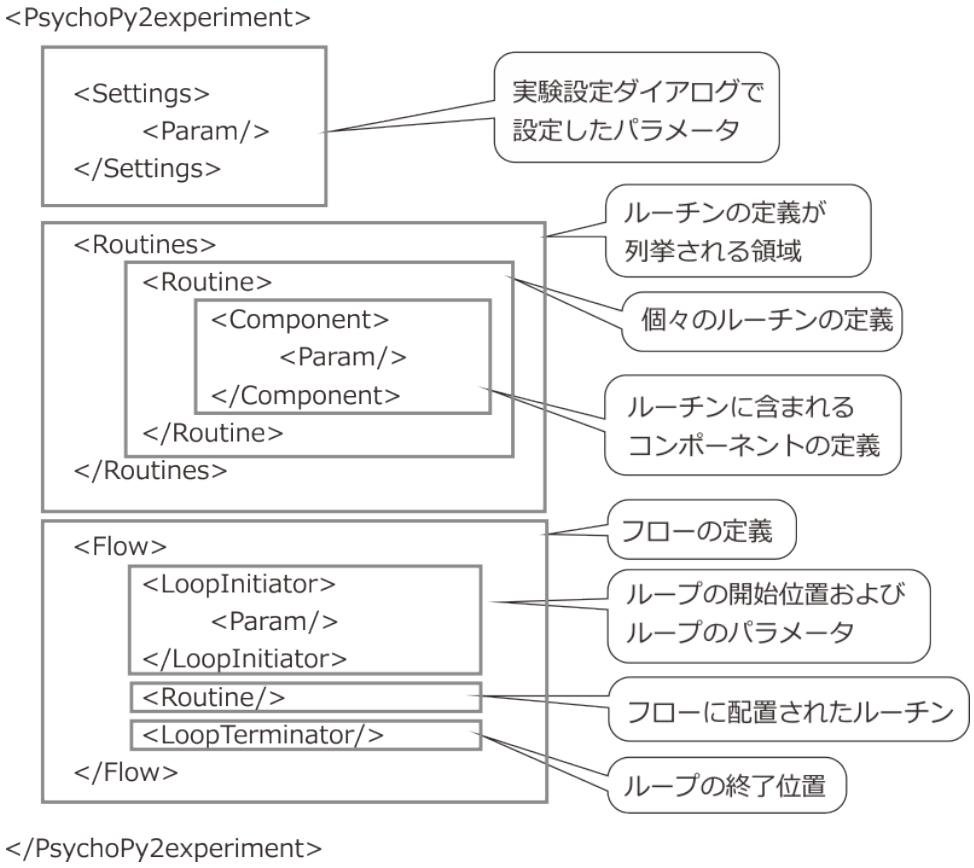


図 13.11　psyexp ファイルの構造

以上の点を踏まえたうえで、 図 13.11 をご覧ください。 図 13.11 は psyexp ファイルの構造を示しています。Builder の実験は PsychoPy2experiment というタグで表現されます。PsychoPy2experiment は Settings、Routines、Flow という要素を持ちます。Settings には実験設定ダイアログで設定したパラメータが記述されています。Routines は、実験で使用されるルーチンの定義である Routine を要素として持っています。 図 13.11 ではひとつしか Routine が描かれていませんが、実験で n 個のルーチンを定義していれば n 個の Routine がここに列挙されます。

個々の Routine は、対応するルーチンに配置されているコンポーネントを定義するタグを要素として持っています。 図 13.11 では Components という名前のタグとして書いてありますが、すでに 図 13.4 や 図 13.5 で見たように、Image コンポーネントに対応するタグは ImageComponent、Keyboard コンポーネントに対応するタグは KeyboardComponent という具合に各コンポーネントに対応するタグが用意されています。本文では trial ルーチンに配置されていた ImageComponent をコピーして貼り付けることによって、Image コンポーネントの個数を増やしました。 図 13.5 や 図 13.7 を見て、コピー範囲と貼り付け位置がタグの入れ子構造を壊さないようになっていることを確認してください。

Flow には、実験のフローが XML で表現されています。Flow の子要素として Routine が配置されている場合は、フローの該当する位置にルーチンが配置されていることを示しています。ループの開始点と終了点はそれぞれ LoopInitiator と LoopTerminator というタグで示されています。多重ループの実験などを適当に作成して(あるいは 第 4 章 で作成した psyexp ファイルを持ってきて)psyexp ファイルの中を確認すると、フローとこれらのタグの関係がよくわかります。

13.9.2 numpy.ndarray 型について

これまでの章ではずっと、刺激の位置 (座標値) や大きさといった二次元の量を指定するためにリストを使用してきました。本書の用途のように静的な位置を表現するだけならリストで十分なのですが、座標値に対する演算を行おうとするとリストは非常に不便です。例えば [5,3] という座標値を X 軸方向に 1、Y 軸方向に 2 移動させたい場合、ベクトルの演算をご存知の方は直感的には [5,3]+[1,2] と書きたくなるでしょう。しかし、Python におけるリストは数値以外にも文字列なども要素になり得ますので、[5,3]+[1,2] をベクトルの和と解釈することにすると要素に数値以外の値があったときに演算が定義できなくなってしまいます。そのようなわけで、かどうかはわかりませんが、Python はリスト同士に対する + 演算子はリストの結合として解釈します。つまり、[5,3]+[1,2]=[5,3,1,2] です。同様に、リストに対する数値の積は、ベクトルとスカラーの積ではなく、ベクトルの繰り返しとして解釈されます。[5,3] * 4 でしたら [5,3] を 4 回繰り返したリストである [5,3,5,3,5,3,5,3] が得られます。

これでは本格的なベクトル演算を行う時に不便で仕方がないので、Python では NumPy というパッケージが用意されています。NumPy を導入すると、直感的なベクトル演算が可能となります。NumPy における演算の基本となるのが numpy.ndarray 型のオブジェクトです。Builder ではリストなどのデータを numpy.ndarray に変換する numpy.asarray という関数が asarray という名前で利用できるように準備されています。asarray を使うと、先ほどのようなベクトル風の演算が可能になります。

asarray([5,3]) + asarray([1,2]) → array([6,5])

asarray([5,3]) * 4 → array([20,12])

これらの演算で得られた戻り値も numpy.ndarray 型のオブジェクトです。numpy.ndarray 型オブジェクトは、リストと同じように [ ] 演算子で要素を取り出したり、スライスを適用したり、len( ) で要素数を求めたりすることができます。ですから、[ ] や len( ) に関しては今まで学んできたリストと全く同等に使えます。しかし、+ 演算子や*演算子を適用した時の働きがリストと異なります。違いを十分に理解できれば asarray( ) を使って積極的に numpy.ndarray の機能を活用していただければ良いのですが、区別に自信がない場合は使わない方がよいでしょう。

13.9.3 range( ) オブジェクトについて

本文の説明では「range( ) 関数が返す range オブジェクトというものがわからない」という方が多いのではないかと思います。実はこの range オブジェクトというのは Python3 から導入されたもので、旧バージョンである Python2 の range( ) は数値を並べたリストを返していました。つまり、本文では 0 から 14 まで並べたリストを得るときに list(range(15)) としましたが、Python2 の頃は単に range(15) と書くだけでよかったのです。これがなぜ Python3 で変更されたのかをお話すれば、range オブジェクトというもののイメージがもう少しはっきりするのではないかと思います。

例として、for 文と range() を使って 1000 万回の繰り返しをする場合を考えてみましょう。通常の心理実験では 1000 万回も繰り返すことはなさそうですが、分野によっては普通にあり得る回数です。Python2 のように range( ) がリストを返すとすると、0 から 9999999 までの 1000 万個の数値を並べたリストが作成されて for 文に渡されることになります。この巨大なリストはコンピュータのメモリ上に保持されますが、はっきり言って非効率的です。必要なのは今が何回目の繰り返しなのかという値だけなのに、1000 万個分ものメモリ領域が食いつぶされてしまうからです。

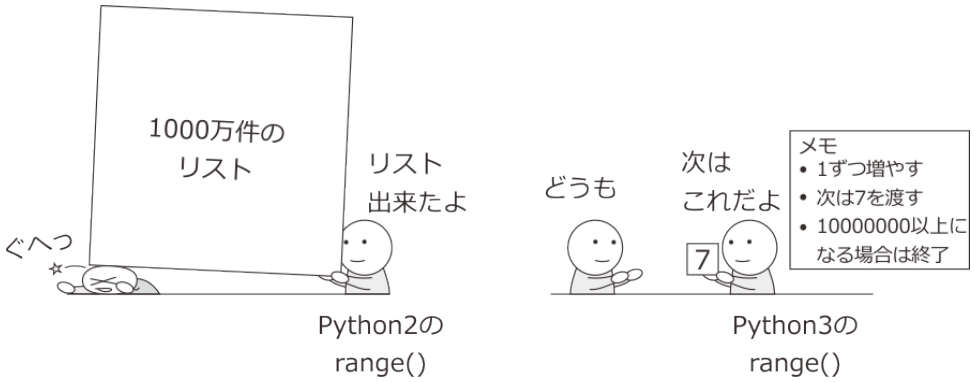


図 13.12　Python2 と 3 の range( ) の違い。

この問題を解決するのが Python3 の range オブジェクトです。range オブジェクトは 図 13.12 右のように

- 1 ずつ増やす
- 次は 7 を渡す
- 10000000 以上になる場合は終了

といったルールを保持していて、値を要求される度に値を生成します (厳密に言うとこれは range オブジェクトから作られるイテレータというオブジェクトの機能)。図 13.12 右の例では、7 を渡したら「1 ずつ増やす」というルールに従って「次は **8** を渡す」と更新しておくわけですね。これなら繰り返し回数が 1000 万回だろうが 1000 億回だろうがメモリへの負担が変わりません。圧倒的に効率がよいです。

range オブジェクトの弱点は、一連の数値をバラバラな順番で取り出す必要がある場合です。いきなり「1852 番目の数値が欲しいんだけど」と言われると、その数値がいくつであるのかをルールに従って延々と計算しなければいけません。このような場合はリストの方が優れています。Python3 でリストが必要な場合は、本文で紹介したように list( ) と組み合わせて list(range(x)) とします。

第 14 章

付録

14.1 本文未解説コンポーネント

PsychoPy Builder 2022.2.4 で利用できるコンポーネントのうち、本文で取り上げなかったものの概要を示します。

Aperture コンポーネント　画面を「穴」で切り抜くコンポーネントです。このコンポーネントはルーチンペインにおける描画順序の影響を受けません。つまり、Aperture コンポーネントの上に他の刺激を重ね書きしようとしても Aperture コンポーネントに切り抜かれてしまいます。

Dots コンポーネント　一様に運動する小さな点を大量に描くコンポーネントです。運動視の研究などに用います。Polygon コンポーネントを大量に用いるより PC への負担が軽く、高速に描画できます。以下の解説では、点が描画される範囲をフィールドと呼んでいます。また、指定された方向に動く点をターゲット、それ以外の方向に動く点をノイズと呼んでいます。

EnvelopeGrating コンポーネント　キャリアとエンベロープという 2 つの周期的なテクスチャをブレンドした刺激を描くコンポーネントです。

Noise コンポーネント　様々なノイズ画像を描くコンポーネントです。ノイズの種類やパラメータ、更新タイミングなどを指定できます。

Brush コンポーネント　画面上にマウスカーソルの軌跡を描けるようにするコンポーネントです。今のところ描いた軌跡を簡単に保存する方法がないので、使いどころが難しいかもしれません。

cedrusButtonBox コンポーネント　Cedrus 製の反応ボタンボックスを用いて反応を記録するコンポーネントです。Keyboard コンポーネントのように検出するボタンを指定してルーチンを終了させたりすることができます。

ioLabsButtonBox コンポーネント　ioLabs Systems 製のボタンボックスを用いて反応を記録するコンポーネントです。Keyboard コンポーネントのように検出するボタンを指定してルーチンを終了させたりすることができます。

JoyButton コンポーネント　ジョイスティックで反応を記録するコンポーネントです。Keyboard コンポーネントのように検出するボタンを指定してルーチンを終了させたりすることができます。

Joystick コンポーネント　ジョイスティックで反応を記録するコンポーネントです。JoyButton コンポーネ

ントと異なり、ボタンのみではなくスティックの状態も記録されます。ルーチンの実行中にジョイスティックの状態にアクセスするには Code コンポーネントを使って Joystick オブジェクトにアクセスする必要があります。

ResourceManager **コンポーネント**　オンライン実験用のコンポーネントです。実験に使用するリソース (画像ファイルなど) をダウンロードするタイミングを制御します。

EmotivMarking **コンポーネント**　Emotive 製の EEG にマーカーを送信するコンポーネントです。

EmotivRecording **コンポーネント**　Emotive 製の EEG で記録を行うコンポーネントです。

Unknown **コンポーネント**　新しいバージョンの Builder で追加されたコンポーネントを使った実験を古いバージョンの Builder で開いた時など、使用中の Builder で利用できないコンポーネントが実験に含まれてる場合があります。このような時、利用できないコンポーネントが Unknown コンポーネントとして表示されます。実験を作成する際にこのコンポーネントを配置しても何も起きません。

ParallelOut **コンポーネント**　パラレルポートからのトリガー出力を行うためのコンポーネントです。

SerialOut **コンポーネント**　シリアルポートでの入出力を行うためのコンポーネントです。

EyeTreackerCalibration **コンポーネント**　アイトラッカーのキャリブレーションを行います。通常のコンポーネントではなく単独で **ルーチンとして** フローに組み込んで使用します。実験設定ダイアログの「アイトラッキング」タブで選択できるアイトラッカーに対応しています。

EyeTreackerValidation **コンポーネント**　アイトラッカーのバリデーション (キャリブレーション精度の確認) を行います。通常のコンポーネントではなく単独で **ルーチンとして** フローに組み込んで使用します。

EyeTreackerRecord **コンポーネント**　アイトラッカーによる記録の開始、終了を制御するコンポーネントです。

RegionOfInterest **コンポーネント**　ROI を定義し、視線の停留などを記録するコンポーネントです。

PeristalticPump **コンポーネント**　LabeoTech 社のディスペンサーを制御するコンポーネントです。

QmixPump **コンポーネント**　Qmix ライブラリ対応のシリンジユニットを制御するコンポーネントです。

14.2 予約語

14.2.1 Python の予約語 (Python 3.7)

Python インタプリタを起動して keyword を import すると、keyword.kwlist というリストに Python 予約語の一覧が格納されます。以下に Python3.7 の予約語を示します。これらの語は Builder において **[名前]** や変数名として使用することはできません。

> FALSE, None, TRUE, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

__builtin__という内部モジュールに含まれる以下の語は Python の予約語ではありませんが、予約語と同様に扱われます (つまり **[名前]** や変数名として使用することはできません)。

ArithmeticError, AssertionError, AttributeError, BaseException, BlockingIOError, BrokenPipeError, BufferError, BytesWarning, ChildProcessError, ConnectionAbortedError, ConnectionError, ConnectionRefusedError, ConnectionResetError, DeprecationWarning, EOFError, Ellipsis, EnvironmentError, Exception, False, FileExistsError, FileNotFoundError, FloatingPointError, FutureWarning, GeneratorExit, IOError, ImportError, ImportWarning, IndentationError, IndexError, InterruptedError, IsADirectoryError, KeyError, KeyboardInterrupt, LookupError, MemoryError, ModuleNotFoundError, NameError, None, NotADirectoryError, NotImplemented, NotImplementedError, OSError, OverflowError, PendingDeprecationWarning, PermissionError, ProcessLookupError, RecursionError, ReferenceError, ResourceWarning, RuntimeError, RuntimeWarning, StopAsyncIteration, StopIteration, SyntaxError, SyntaxWarning, SystemError, SystemExit, TabError, TimeoutError, True, TypeError, UnboundLocalError, UnicodeDecodeError, UnicodeEncodeError, UnicodeError, UnicodeTranslateError, UnicodeWarning, UserWarning, ValueError, Warning, WindowsError, ZeroDivisionError, _, __build_class__, __debug__, __doc__, __import__, __loader__, __name__, __package__, __spec__, abs, all, anyascii, bin, bool, bytearray, bytes, callable, chr, classmethod, compile, complex, copyright, credits, delattr, dict, dir, divmod, enumerate, eval, exec, exit, filter, float, format, frozenset, getattr, globals, hasattr, hash, help, hex, id, input, int, isinstance, issubclass, iter, len, license, list, locals, map, max, memoryview, min, next, object, oct, open, ord, pow, print, property, quit, range, repr, reversed, round, set, setattr, slice, sorted, staticmethod, str, sum, super, tuple, type, vars, zip

14.2.2 PsychoPy の予約語 (2020.2.9)

以下の語は PsychoPy で利用するモジュール名のため、Builder において **[名前]** や変数名として使用することができません。

gui, misc, visual, core, event, data, sound, microphone, psychopy, os

以下の語は PsychoPy で予約されているため、Builder において **[名前]** や変数名として使用することができません。

ENVIRON, FINISHED, FOREVER, NOT_STARTED, PAUSED, PLAYING, PRESSED, PSYCHOPY_USERAGENT, PY3, RELEASED, SKIP, STARTED, STOPPED, __builtins__, __cached__, __doc__, __file__, __loader__, __name__, __package__, __spec__, _gitStandalonePath, absolute_import, abspath, copy, gitExe, join, os, print_function, sys

以下の語は Builder で予約されているため、Builder において **[名前]** や変数名として使用することができません。

KeyResponse, keyboard, buttons, continueRoutine, expInfo, expName, thisExp, filename, logFile, paramName, t, frameN, currentLoop, dlg, _thisDir, endExpNow, globalClock, routineTimer, frameDur, theseKeys, win, x, y, level, component, thisComponent

以下の語は numpy および numpy.random から import されるので Builder において **[名前]** や変数名として使用することはできません。

asarray, average, cos, deg2rad, linspace, log, log10, normal, np, pi, rad2deg, randint, random, shuffle, sin, sqrt, std, tan

14.2.3 Builder の内部変数 (2020.2.9)

Builder で使用されている内部変数の概要を示します。

予約語	概要
KeyResponse	予約されています。
_thisDir	Builder を実行するときにカレントフォルダを psyexp ファイルがある場所に移動するために使用されます。
buttons	最後に取得したマウスのボタンの状態を示すリストが格納されています。
component	予約されています。
continueRoutine	実行中のルーチンを継続するか否かを示す真偽値が格納されています。→ 第 7 章
currentLoop	Loop の種類で staircase、interleaved staircases を選択したときに使用されます。→ 第 11 章
dlg	expInfo ダイアログを作成するために使用します。
endExpNow	ESC キーによる実験の中断を有効にしているときに、この変数を利用して ESC キー以外のキーで実験を終了できます。
expInfo	expInfo ダイアログの項目と値が辞書オブジェクトとして格納されています。→ 第 4 章 、第 5 章
expName	実験設定ダイアログの **[実験の名前]** に入力した実験名が格納されています。
filename	各種実験記録ファイルやログファイルのファイル名を生成するために利用されます。
frameDur	フレームレートの実測値を保持しています。計測に失敗した場合は 1/60sec にセットされます。
frameN	現在のフレーム番号を格納しています。→ 第 5 章
globalClock	実験開始からの経過時間を計測するための psychopy.core.Clock のインスタンスが格納されています。→ 第 12 章
keyboard	キーボード用のライブラリの読み込みのために使用されています。
level	Loop の種類で staircase、interleaved staircases を選択したときに使用されます。→ 第 11 章
logFile	ログファイルを作成するための psychopy.logging.LogFile のインスタンスが格納されています。
paramName	Loop の種類で interleaved staircases を選択したときに使用されます。→ 第 11 章
routineTimer	ルーチン終了までの残り時間を計測するための psychopy.core.CountdownTimer のインスタンスが格納されています。
t	ルーチンが開始してからの経過時間を格納しています。→ 第 5 章
theseKeys	最後に取得したキーの状態を示すリストを格納しています。→ 第 7 章

次のページに続く

表 14.1 – 前のページからの続き

予約語	概要
thisComponent	現在処理中のコンポーネントに対応するインスタンスが格納されています。
thisExp	フローの制御ややデータの保存に関与する psychopy.data.ExperimentHandler のインスタンスが格納されています。
win	刺激提示スクリーン本体である psychopy.visual.Window のインスタンスを格納しています。
x	最後に取得したマウスカーソルの X 座標を格納しています。Mouse コンポーネントの **[マウスの状態を保存]** の設定によって単独の値であったりリストであったりします。
y	最後に取得したマウスカーソルの Y 座標を格納しています。Mouse コンポーネントの **[マウスの状態を保存]** の設定によって単独の値であったりリストであったりします。

以上に加えて、以下の語は正常なルーチンの実行に必須の変数名と一致するので Builder において **[名前]** や変数名として使用することはできません。

trialComponents (trial はルーチン名)	当該ルーチンでフレーム毎に処理する必要があるコンポーネントのインスタンスを並べたリストが格納されています。→ 第 8 章
trialClock (trial はルーチン名)	当該ルーチンが開始されてからの経過時間を計測する psychopy.core.Clock のインスタンスが格納されています。→ 第 9 章

14.3 ログファイル

PsychoPy Builder で実験を実行すると、拡張子 .log のログファイルが作成されます。実験が十分な精度で実行されているかどうかを確認したい場合や、どうも意図している通りに刺激が提示されないといった問題が生じている時に、このログファイルから情報が得られる場合があります。ただし、Builder の実験がどのようなコードにコンパイルされて実行されるのかある程度知識がないと対策をとることまでは難しいかも知れません。

ログには以下のレベルがあります。実験設定ダイアログでログの出力レベルを選択すると、選択されたレベル以上のログがログファイルに出力されます。例えば warning を選択すると、waring と error のレベルのログが出力されます。info を選択すると、info、exp、data、warning、error のレベルのログが出力されます。

1. error
2. warning
3. data
4. exp
5. info
6. debug

以下に debug を選択した場合のログファイルの先頭部分の例を示します。非常に長い行は中略してあります。各行の最初の数値が実験開始からの経過時間 (秒)、続いてログのレベルが示されています。レベルに続いてその時刻に生じたイベントの内容が書かれています。

```
3.0528  WARNING  Movie2 stim could not be imported and won't be available
7.8399  INFO  Loaded monitor calibration from ['2015_06_02 16:25']
8.9837  EXP  Created window1 = Window(allowGUI=False, allowStencil=False, ... (略)
8.9838  EXP  window1: recordFrameIntervals = False
9.1467  EXP  window1: recordFrameIntervals = True
9.3343  DEBUG  Screen (0) actual frame rate measured at 58.77
9.3344  EXP  window1: recordFrameIntervals = False
9.9161  EXP  Created text = TextStim(alignHoriz='center', alignVert= ... (略)
9.9274  EXP  Created stimulus = Polygon(autoDraw=False, autoLog=True, ... (略)
9.9286  EXP  <method-wrapper '__getattribute__' of attributeSetter object ... (略)
(中略)
13.8202  EXP  Created sequence: fullRandom, trialTypes=4, nReps=25, seed=None
13.8229  EXP  New trial (rep=0, index=0): {u'correct_ans': u'slash', ... (略)
13.8499  EXP  text: autoDraw = False
13.8499  EXP  stimulus: fillColor = u'green (named)'
13.8499  EXP  stimulus: pos = array([-400.,    0.])
13.8499  EXP  stimulus: lineColor = u'green (named)'
13.8499  EXP  cross1: autoDraw = True
13.8499  EXP  cross2: autoDraw = True
14.8756  EXP  stimulus: autoDraw = True
15.4022  DATA  Keypress: slash
15.4382  EXP  New trial (rep=0, index=1): {u'correct_ans': u'slash', ... (略)
15.4778  EXP  stimulus: autoDraw = False
15.4778  EXP  cross1: autoDraw = False
15.4778  EXP  cross2: autoDraw = False
15.4778  EXP  stimulus: fillColor = u'green (named)'
15.4778  EXP  stimulus: pos = array([-400.,    0.])
15.4778  EXP  stimulus: lineColor = u'green (named)'
15.4778  EXP  cross1: autoDraw = True
15.4778  EXP  cross2: autoDraw = True
16.4711  EXP  stimulus: autoDraw = True
17.0442  DATA  Keypress: slash
```

もしこの実験をログレベル exp で実行していたら、exp 以上のレベルのみが出力されるので、以下のような出力になります。上の例の 2 行目の INFO と 6 行目の DEBUG が抜けている点にご注意ください。

```
3.0528  WARNING  Movie2 stim could not be imported and won't be available
8.9837  EXP  Created window1 = Window(allowGUI=False, allowStencil=False, ... (略)
8.9838  EXP  window1: recordFrameIntervals = False
```

(次のページに続く)

(前のページからの続き)

```
9.1467  EXP  window1: recordFrameIntervals = True
9.3344  EXP  window1: recordFrameIntervals = False
9.9161  EXP  Created text = TextStim(alignHoriz='center', alignVert= ... (略)
9.9274  EXP  Created stimulus = Polygon(autoDraw=False, autoLog=True, ... (略)
9.9286  EXP  <method-wrapper '__getattribute__' of attributeSetter object ... (略)
(中略)
13.8202  EXP  Created sequence: fullRandom, trialTypes=4, nReps=25, seed=None
13.8229  EXP  New trial (rep=0, index=0): {u'correct_ans': u'slash', ... (略)
13.8499  EXP  text: autoDraw = False
13.8499  EXP  stimulus: fillColor = u'green (named)'
13.8499  EXP  stimulus: pos = array([-400.,    0.])
13.8499  EXP  stimulus: lineColor = u'green (named)'
13.8499  EXP  cross1: autoDraw = True
13.8499  EXP  cross2: autoDraw = True
14.8756  EXP  stimulus: autoDraw = True
15.4022  DATA Keypress: slash
15.4382  EXP  New trial (rep=0, index=1): {u'correct_ans': u'slash', ... (略)
15.4778  EXP  stimulus: autoDraw = False
15.4778  EXP  cross1: autoDraw = False
15.4778  EXP  cross2: autoDraw = False
15.4778  EXP  stimulus: fillColor = u'green (named)'
15.4778  EXP  stimulus: pos = array([-400.,    0.])
15.4778  EXP  stimulus: lineColor = u'green (named)'
15.4778  EXP  cross1: autoDraw = True
15.4778  EXP  cross2: autoDraw = True
16.4711  EXP  stimulus: autoDraw = True
17.0442  DATA Keypress: slash
```

ログレベルが warning なら、以下のような出力になります。

```
3.0528  WARNING  Movie2 stim could not be imported and won't be available
```

info や debug のレベルのログは Builder の新機能の開発などの際に便利な内容が多く、おそらく一般ユーザーが必要とすることは少ないでしょう。exp のレベルは刺激オブジェクトの作成やパラメーターの変更、ループの実行状況などに関する情報が出力されます。data のレベルは実験記録ファイルへのデータ出力に関する情報が出力されます。exp のログの時刻を確認したら、もしかすると刺激提示のタイミングなどに関するトラブルの情報が得られる可能性があります。

warning のレベルでは、致命的ではないかも知れないけれども問題が生じていることが報告されます。上の例では MovieStim2 が利用できないという問題が報告されています。今実行している実験の中で動画刺激を使用していなければ問題ありませんが、今後動画刺激を使いたいと思った時には利用できるようにインストールの問題などを解決しなければいけないことを示しています。

warning のレベルで最も注意すべきは、以下のように last frame was XX ms というログが出力されている場合

です。このようなログが出力されている場合は、フレームの描画に問題があって一定のスピードで描画ができていません。アニメーションする刺激の描画がカクカクしてしまったり、刺激の出現、消去のタイミングがずれてしまっている可能性があります。常駐プログラムの停止、グラフィックデバイスドライバの更新や、高性能グラフィックボードの追加などの対策が必要となる可能性があります。

```
16.5660  WARNING  t of last frame was 30.22ms (=1/33)
16.5974  WARNING  t of last frame was 31.41ms (=1/31)
16.6285  WARNING  t of last frame was 31.09ms (=1/32)
```

14.4 PsychoPy 設定ダイアログ

PsychoPy 設定ダイアログの各項目について簡単にまとめました。

14.4.1 一般

[ウィンドウ描画ライブラリ] ウィンドウの描画に使うライブラリを指定します。pyglet、glfw、pygame が選択できますが、pygame は旧バージョンで作成した実験との互換性のために残されているものなので、これから PsychoPy を使う人は選択すべきではありません。

[単位] 実験の設定ダイアログで **[単位]** を「PsychoPy の設定を用いる」に設定した場合、ここで選択した単位が使用されます。

[フルスクリーン] チェックしておくと標準でフルスクリーンウィンドウで刺激を描画しようとします。実験の設定ダイアログで「フルスクリーンウィンドウ」のチェックを外している場合は、実験の設定ダイアログが優先されます。

[GUI を使用] チェックしておくと標準でマウス等を有効にします。

[パス] 独自の Python モジュールを使用したい場合、モジュールが置かれているパスをここに列挙しておくと import できます。

[frac オーディオ圧縮] frac を利用したい場合、ここへ frac へのパスを設定します。

[実験終了キー] 実験を終了させるキーを ESC 以外にしたい時にここに指定します。pyglet のキー名でなければいけません。

[実験終了キーのモディファイア] 「Shift を押しながら ESC」のようにモディファイアキーを押しながら終了するようにしたい時にここに指定します。

[ガンマエラーの処理] ガンマ補正が出来なかった時に終了するか警告するかを指定します。

[スタートアッププラグイン] PsychoPy 起動時に追加で読み込むプラグインを指定します。

[スタートアッププラグイン] 音声文字変換で Google Cloud API を使用する際に、Google Cloud API キーが格納された JSON ファイルをここに指定します。

14.4.2 アプリケーション

[起動時にチップを表示] チェックしておくと PsychoPy Builder/Coder を起動したときに「今日のチップ」を表示します。

[標準で開くウィンドウ] PsychoPy を起動したときに開くウィンドウを指定します。

[設定の初期化] 設定を初期化したいときは、ここにチェックをして PsychoPy を再起動してください。

[設定の自動保存] ウィンドウを閉じるときに、未保存の設定を自動的に保存します。

[デバッグモード] PsychoPy のデバッグ用機能を有効化します。PsychoPy そのもののデバッグであって、ユーザーが作成した実験スクリプトのデバッグではありません。

[ロケール] PsychoPy のメニュー等の表示に使う言語を選択します。「システムの言語設定」にしておくと、OS の言語の設定を利用します。

[エラーダイアログ] PsychoPy の動作中にエラーが生じた際にダイアログを表示します。バグの報告をする際はこのダイアログの情報を伝えると開発チームによる問題の把握に役立ちます。

[テーマ] 起動時に適用するテーマを指定します。旧バージョンの見た目に近づけたい場合は Classic を選択するとよいでしょう。

[前回の実験を開く] チェックしておくと、前回開いていた実験を自動的に開きます。

[Code コンポーネントの言語] Code コンポーネントの標準のコードタイプを指定します。JavaScript への自動変換に問題がある場合や、JavaScript 用入力欄が不要な場合などに設定を変更してください。

[名前空間の整理] コードのコンパイルに関連するオプションです。通常は変更する必要はないでしょう。

[コンポーネントフォルダ] パスを列挙したリストを指定します。パス内に含まれる拡張子.py のファイルを Builder のコンポーネントとして読み込みます。

[コンポーネント表示制限] Builder において Python によるローカル実験、PsychoJS によるオンライン実験それぞれに対応したコンポーネントのみを表示したい場合にここで設定します。

[表示しないコンポーネント] 特定のハードウェア用コンポーネントなど、使用しないコンポーネントの名前をここに書いておくと、コンポーネントペインに表示されなくなります。ロケールが日本語に設定されていて、プロパティ設定ダイアログのタイトルバーに「Foo コンポーネント」と表示されているコンポーネントを非表示にしたい場合は FooComponent と書く必要があります。

[デモのディレクトリ] Builder のデモが展開されているディレクトリを指定します。Builder ウィンドウのメニューの「デモ」メニューから「デモを展開...」すると自動的に設定されます。展開されたデモを手作業で別の場所に移したりしない限り、通常は変更する必要はありません。

[データ保存フォルダ] Builder の実験を実行したときに、記録ファイルやログファイルが保存されるフォルダ名を指定します。

[Builder のレイアウト] Builder の各ペインの配置を指定します。

[常に readme を表示] psyexp ファイルと同一のフォルダに readme.txt というファイルが存在している場合、この項目をチェックしておくと readme.txt の内容が表示されます。

[お気に入りの最大登録数] コンポーネントペインの「お気に入り」に登録できるコンポーネント数の上限を指定します。

[Routine を閉じるときの確認] Routine のタブを閉じるときに確認ダイアログを表示するかどうかを指定します。

[読み込み専用で開く] 実験を実施する際に、不用意にコードが変更されてしまうのを防ぎたいときにチェックします。

[出力用フォント] 出力パネルの表示に使用するフォント名を指定します。複数のフォントを列挙すると、最初に読み込むことができたフォントを使用します。

[コード用フォント] コードの表示に使用するフォント名を指定します。複数のフォントを列挙すると、最初に読み込むことができたフォントを使用します。

[コード用フォントサイズ] フォントサイズを 6 から 24 の整数で指定します。

[出力用フォントサイズ] フォントサイズを 6 から 24 の整数で指定します。

[行間スペース] 行間を整数で指定します。

[文字数ガイドの位置] 1 行の長さの目安となる縦線を引く位置を文字数で指定します。

[ソースアシスタントを表示] チェックすると Coder 起動時にソースアシスタントを表示します。変数に格納されているクラスのヘルプなどが表示されます。

[出力パネルを表示] チェックを外して PsychoPy を再起動すると、Coder ウィンドウの下部に出力パネルが表示されません。

[自動補完] 入力時にコード補完の候補と関数呼び出しのヒントを表示するか否かを指定します。

[前回のファイルを開く] 前回終了時に開いていたファイルを自動的に開きます。

[優先するシェル] Coder ウィンドウ下部のシェルパネルで優先するシェルを指定します。

14.4.3 キー設定

ショートカットキーを編集することができます。個々の項目については省略します。

14.4.4 ハードウェア

[オーディオライブラリ] 音声刺激の再生に使用するライブラリを列挙します。実験の設定ダイアログで「PsychoPy の設定を用いる」を選択した場合、ここに列挙されたものを順番にロードして最初にロードできたものを利用します。

[オーディオレイテンシの設定] PsychToolbox のオーディオを使用する際に、レイテンシの設定を選択します。

[オーディオドライバ] 音声刺激の再生に使用するドライバを列挙します。最初にロードできたものを利用します。

[オーディオデバイス] 音声刺激の再生に使用するデバイスを指定します。PC に複数のオーディオデバイスがある場合、適切なものを選択しないと音声が再生されないので注意してください。

[パラレルポート] パラレルポートのアドレスを列挙します。ここへアドレスを記入しておかないと ParallelOut コンポーネントでアドレスを選択できません。

[Qmix の設定] Qmix pump(シリンジ制御ライブラリ) の設定を指定します。

14.4.5 ネットワーク

[プロキシ] ネットワークアクセスにプロキシが必要な場合、ここへアドレスとポートを記入します。

[プロキシの自動構成] 自動的にプロキシを構成しようとします。自動構成が利用可能な場合は「プロキシ」の設定に優先します。

[利用統計の送信を許可] PsychoPy 開発チームへ利用統計を送信することを許可します。開発の参考にするため、可能な限りチェックしておいてください。

[更新の確認] 起動時に新しいバージョンの PsychoPy が公開されていないか確認します。

[タイムアウト] ネットワーク接続のタイムアウト時間を指定します。

14.5 PsychoPy 2024.1.0 の新機能

PsychoPy 2024.1.0 では非常に重要な機能がいくつかついており、本来なら本書全体を修正する必要があるのですが、十分な時間が確保できないためひとまず付録という形で紹介しておきます。

14.5.1 Pilot モードと rush モード

本実験に入る前の動作確認に使用する Pilot モードが追加されました。Builder や Coder の画面上にあるリボン (リボンについては後述) や「実験の設定」ダイアログの「基本」タブにある **[実行モード]** から、従来の実行モードである「実行モード」と「Pilot モード」を切り替えることができます。

図 14.1 Builder および Coder のリボンに表示されているスイッチで現在のモードの確認と切り替えができます。

従来の実行モードでは、フルスクリーンで実行した実験に問題があってハングアップしてしまった場合、実

験を停止させるのに苦労することがありました。Pilot モードでは強制的にウィンドウモードで実行されますので、ハングアップしてしまったときにも、簡単に Runner ウィンドウの「終了」ボタンをクリックして実験を停止させることができます。Pilot モードではウィンドウにオレンジ色の枠が描かれ、左下に"PILOTING: Switch to run mode before testing."と表示されますので、どちらのモードで実行しているのかがすぐわかります。「3.11.1: 自分のキーボードで使えるキー名を確かめる」で紹介したキー名を表示させる実験などは、Pilot モードで実行するのがおすすめです。

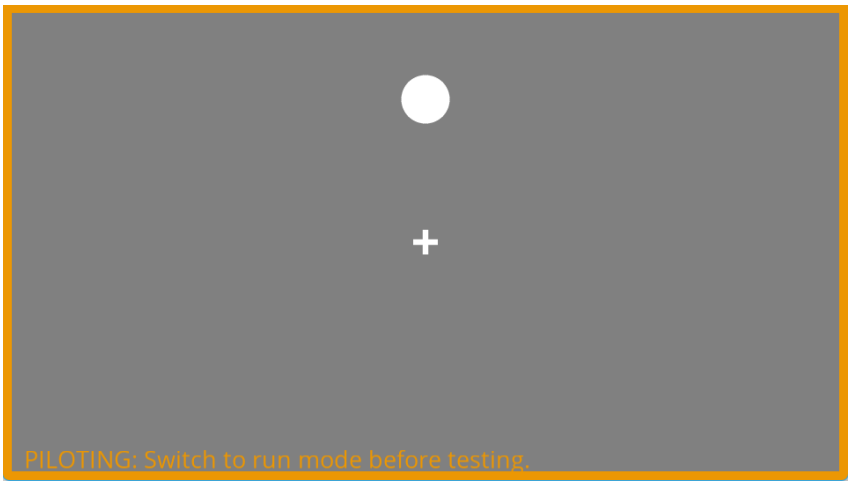


図 14.2　Pilot モードで実験を実行するとウィンドウモードで実行され、オレンジ色の枠が描画されます。

Pilot モードでのウィンドウモード強制やオレンジ色の枠の描画は、「PsychoPy の設定」ダイアログに追加された「Pilot モード」で無効にすることができます。ここでは他にもログファイルに出力される情報の変更 (標準では最も詳細な動作情報が出力される debug に設定されている) や、後述の rush モードを利用するか否かを切り替えることができます。通常、これらの設定を変更する必要はないでしょう。

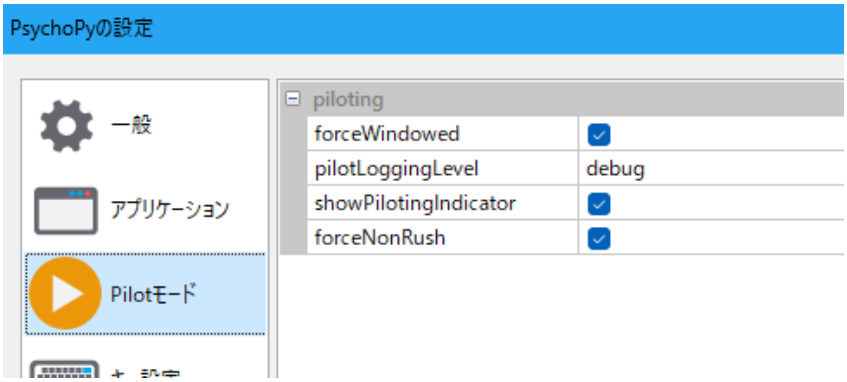


図 14.3　PsychoPy の設定ダイアログで Pilot モードの動作を変更できます。

rush モードというのは、実験プログラムの優先度を高めるモードです。Windows にせよ MacOS にせよ、現代の OS では、さまざまな便利な機能を実現するために、ユーザー側からは見えない数多くのプロセスが実行されています。実験の実行中に他のプロセスが処理時間を要する作業を始めると、実験の時間精度に悪影響を与えることがあります。rush モードを ON にすると、PsychoPy は実験実行時に実験のプロセスをできる限り高い優先度で実行しようとします。rush モードを使うかどうかは「実験の設定」ダイアログの「基本」タブにある **[rush モード]** で実験ごとに指定できます。

14.5.2 フレームレート測定機能の ON/OFF

Builder は実験の時間精度を確保するために、実験実行環境のフレームレートの実測値に基づいて刺激を制御しています。少し前のバージョンの Builder より、実験を実行すると最初に"Attempting to measure frame rate of screen, please wait..."というメッセージが表示されるようになりましたが、この時にフレームレートの測定を行っています。どの程度の時間精度が必要かは実験の内容に寄りますが、時間精度が高すぎて困るということはないでしょうから、原則としてこの機能は ON にしておくべきです。しかし、マルチモニター環境など、一部の実行環境でフレームレート測定機能のせいで実験がうまく開始されないことがあるため、フレームレート測定を OFF にする機能が追加されました。

「実験の設定」ダイアログの「スクリーン」タブに **[フレームレートの測定]** という項目があり、その下に **[フレームレート測定時のメッセージ]** という項目があります。 **[フレームレートの測定]** のチェックを OFF にすると、 **[フレームレート測定時のメッセージ]** が **[フレームレート]** に変わります。これでフレームレート測定が OFF になりましたが、Builder の実験は **フレームレートが与えられないと正常に動作しない** ため、 **[フレームレート]** にフレームレートを得るための式を書かなければいけません。60Hz のモニターを使っているのなら 60 と直接入力してもよいでしょうし (仕様通りに動作するとは限らないからこそ実測するのですが、それほど時間精度が必要ないなら十分でしょう)、フレームレートを得るための独自の式を書いても構いません。

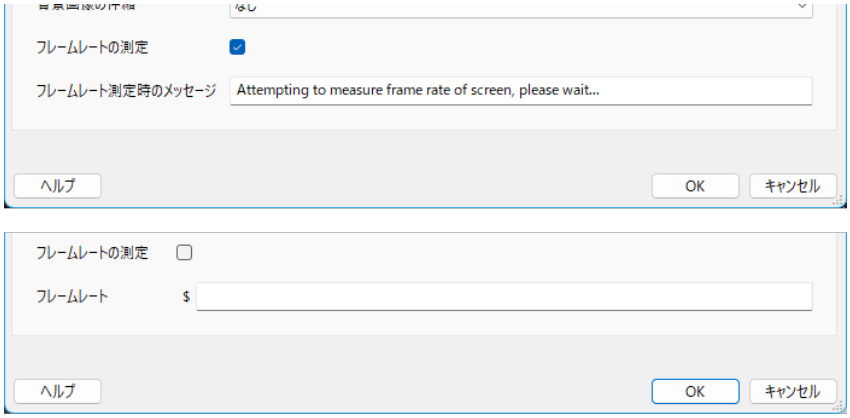


図 14.4　フレームレート測定機能を OFF にすることができます。

なお、フレームレート測定機能が ON のとき、 **[フレームレート測定時のメッセージ]** を編集すれば画面上に表示されるメッセージを変更できます。ただし、バージョン 2024.1.1 の時点でフォントの種類や大きさを変更することができません。日本語の文字などの非 ASCII 文字は文字化けする可能性が高いのでご注意ください。

14.5.3 ルーチンごとのスクリーン設定/ルーチンのスキップ

Routine ペインの左上に [Routine の設定] が追加され、ルーチン毎に設定を切り替えられるようになりました。ルーチンの設定ダイアログには「基本」、「Flow」、「ウィンドウ」、「データ」、「テスト」のタブがあります。「データ」と「テスト」はコンポーネントのプロパティ設定ダイアログと同じです。「基本」ではルーチンの名前を変更できるほか、ルーチンについての説明を記入しておくことができます。ルーチンを使いまわす時などのためにいろいろとメモしておくとよいでしょう。

「ウィンドウ」タブにある **[このルーチン固有のスクリーン設定を使用]** をチェックすると、 図 14.5 のようにスクリーンの背景色と背景画像に関する項目が表示されます。これらは実験の設定ダイアログの「スクリーン」タブにある同名の項目と同じ働きをしますが、このルーチンに対してだけ有効です。

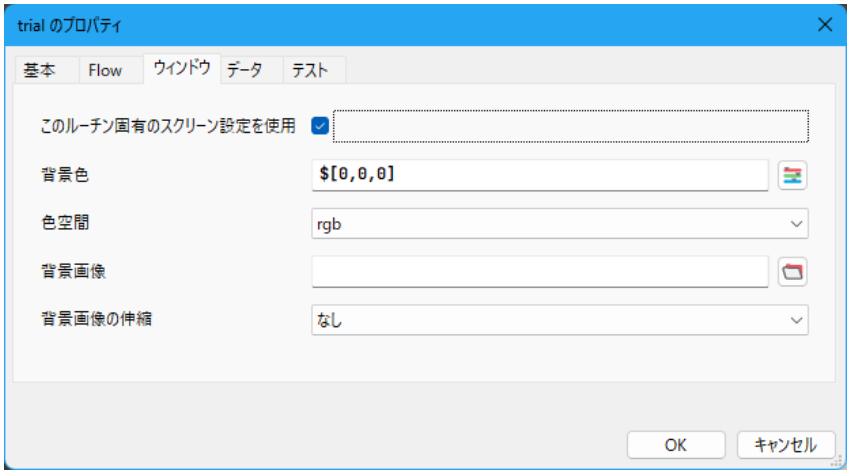


図 14.5　ルーチンごとに背景色と背景画像を設定できます。

「Flow」タブには **[タイムアウト]** と **[条件に合致する場合はスキップ...]** という項目があります。

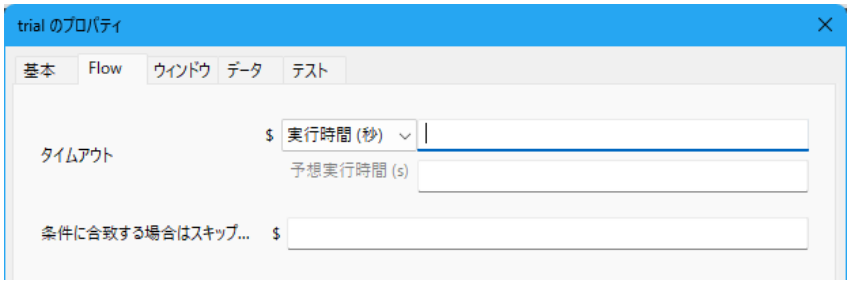


図 14.6　[Flow] タブではルーチンの終了時刻を設定したり、ルーチンをスキップする条件を設定したりできます

[タイムアウト] は通常のコンポーネントの **[終了]** と同様ですが、ルーチン内にまだ終了していないコンポーネントがあっても強制的に終了させることができます。図 14.7 では key_resp と polygon の **[終了]** を空白にして終了しないようにしていますが、ルーチンの設定の **[タイムアウト]** を 2 秒に設定しています。タイムラインの 2 秒のところにオレンジの線が引かれて、それより右側では key_resp と polygon の実行期間を示すバーがグレーになっているのがわかります。おそらくこの機能が真価を発揮するのは、何らかの条件により繰り返しのたびに制限時間が変化するルーチンを作成する必要がある時です。従来なら、ルーチン内の（制限時間の変化を受ける）すべてのコンポーネントの終了時刻を変数にする必要がありましたが、この **[タイムアウト]** を使うと一か所設定するだけで済みます。他には、実験の動作確認を短時間で行うために、重要ではないルーチンの **[タイムアウト]** を 0.5 秒や 1 秒に設定すれば、そのルーチンをきちんと表示させつつ短時間で先のルーチンへ進めることができます。

[条件に合致する場合はスキップ...] は非常に重要な機能です。例えば 200 試行を繰り返すループがあって、50 試行ごとにキーを押すまで先へ進まない休憩用の画面を表示したいとします。従来、このような手続きはループの繰り返し回数に 0 を設定するテクニックと Code コンポーネントを組み合わせるか、Code コンポーネントを使って 1 フレームも描画する前にルーチンを抜ける必要がありましたが、このような「泥臭い」方法を使わずに実現できるようになります。具体例として、 図 14.8 のように trials ループ内に trial と rest というルーチンがあって、繰り返し 50 回毎に rest を表示したいとします。このとき、trals ループのデータ属性 thisN には現在の繰り返し回数を表す整数が格納されているので (最初の繰り返しが 0)、trials.thisN % 50 とすれば現在の繰り返し回数を 50 で割った余りが得られます。trials.thisN % 50 != 0 とすれば余りが 0 でない時にスキップされるので、結果として 50 回の繰り返しに 1 度だけルーチンが実行されます。「 8.11: 軌跡データを間引き

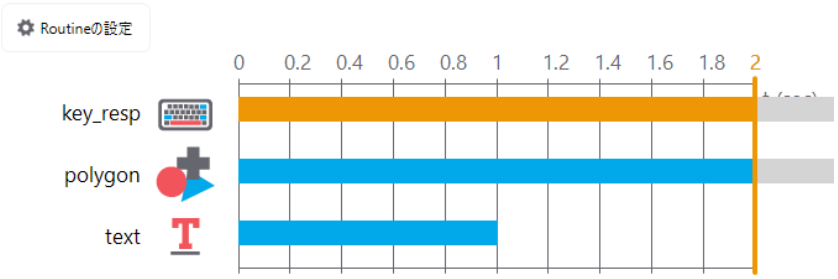


図 14.7　**[タイムアウト]** を設定すると、それより長時間実行されるよう設定されているコンポーネントがあっても強制的にルーチンが終了します。

しよう 」も参考にしてください。

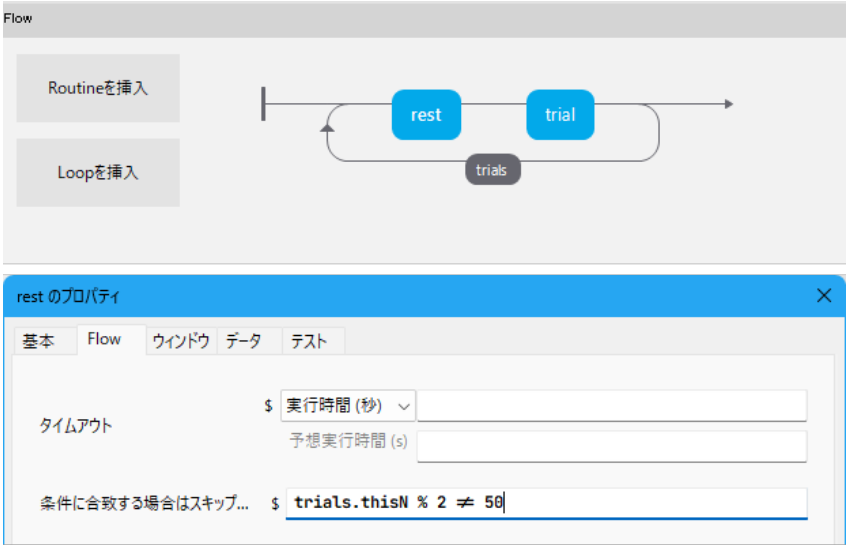


図 14.8　**[条件に合致する場合はスキップ...]** を使うと繰り返し中の特定の回だけ実行されるルーチンを簡単に作成できます。

14.5.4 CounterBalance ルーチン

これも非常に重要な新機能であり、従来は難しかった参加者間のカウンターバランスを取ることが可能となります。CounterBalance ルーチンは、Component ペインの「カスタム」カテゴリの中にあります (図 14.9 参照)。Component ペインにあるアイテムは通常「Code コンポーネント」のように「コンポーネント」と呼ばれますが、CounterBalance ルーチンは「ルーチン」と呼ばれている点にご注意ください。CounterBalance ルーチンは他のコンポーネントのように「ルーチンの中に配置する」のではなく、それ自体が独立したルーチンとして働きます。ですので、CounterBalance ルーチンのアイコンをクリックすると、図 14.9 のように counterbalarance というルーチンが追加されます。そして counterbalance ルーチンは通常のルーチンのようにタイムラインが表示されず、counterbalance ルーチンの動作を決定する項目が表示されます。作成された CounterBalance ルーチンは、図 14.9 の Flow ペインのようにフロー内に 1 回挿入することによって有効になります。

CounterBalance ルーチンは、初期状態で 図 14.9 のように **[グループの定義...]** が「グループ数」に設定されています。この設定のときの CounterBalance ルーチンの動作を 図 14.10 に示します。図上段の左端は、 **[グ**

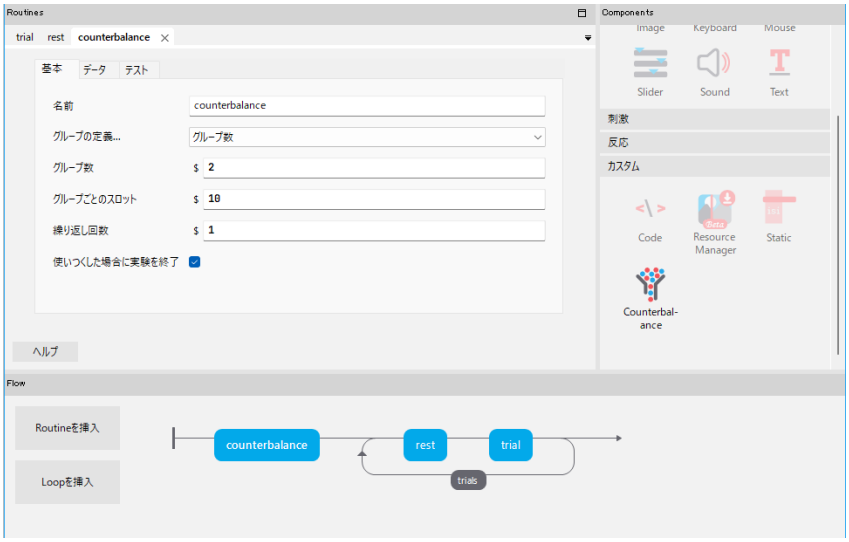


図 14.9　CounterBalance ルーチンを挿入した様子

ループ数] を 2、 **[グループごとのスロット]** (以下スロット) を 5、 **[繰り返し回数]** を 1 に設定した後、最初に実験を実行した直後の状態を示しています。**[グループ数]** で設定した個数の箱のようなものが用意され、順番に 0, 1 と番号がつけられています。それぞれの箱には **[グループごとのスロット]** で設定した個数のボール (とでもしておきましょう) が入っています。実験が 1 回実行されるたびに、いずれかのグループがランダムに選択され、そのグループからボールがひとつ取り出されていきます。選択されたグループは、counterbalance.group(counterbalance の部分は CounterBalance コンポーネントの **[名前]**) というデータ属性で参照できます。各グループの残り個数は実験のフォルダに自動的に作成される **shelf.json というファイルに保存されます。**そして、すべてのグループのボールがなくなると、グループを選択することができないため counterbalance.group の値は None になります。**[使いつくした場合に実験を終了]** をチェックしておくと、すべてのグループのボールが残っていない場合に自動的に実験が終了します。

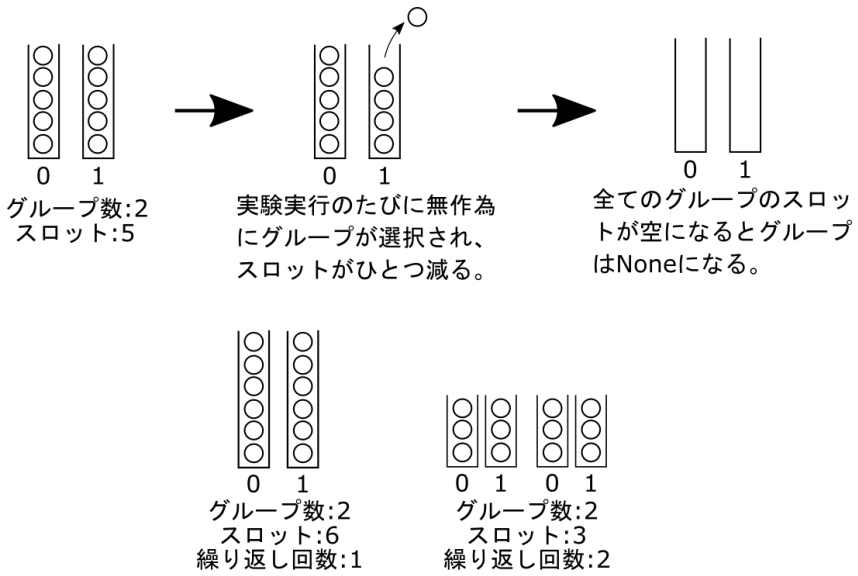


図 14.10　CounterBalance ルーチンの働き

図 14.10 の下段は **[繰り返し回数]** の働きを示しています。**[繰り返し回数]** を 2 以上の整数にすると、すべてのスロットが満ちている状態から空になるまでの動作を設定された回数だけ繰り返します。したがって、 **[グ**

ループごとのスロット] を 6、 **[繰り返し回数]** を 1 に設定した場合も、 **[グループごとのスロット]** を 3、 **[繰り返し回数]** を 2 に設定した場合も、各グループは 6 回実行されます。ただし、前者の場合はグループ 0 が 6 回連続した後にグループ 1 が 6 回連続するということが稀とはいえ起こり得ますが、後者の場合は各グループ 3 回までしか連続することがありません。「 3.5: 繰り返しを設定しよう 」における fullRandom と random の違いに似ていると言えるでしょう。

なお、現在の実行状況を保存している shelf.json というファイルは JSON という形式のテキストファイルであり、一般的なテキストエディタで内容を確認することができます。JSON についてここでは詳しく触れませんが、だいたい想像がつくのではないかと思います。

```
{
 "counterbalance": {
  "_reps": 1,
  "0": 5,
  "1": 4
 }
}
```

実験の作成中、一度動作確認した後に **[グループ数]** や **[グループごとのスロット]** を変更しても、変更が反映されず以前の設定のままとなることがあります。その場合は **shelf.json を削除すれば次回の実行時に変更後の設定で shelf.json が作成されます。**

本文中でこの CounterBalance ルーチンを紹介するとすれば、もっともふさわしいのは 第 4 章 でしょう。「 4.6: 実験情報ダイアログで条件ファイルを指定しよう 」では 20 度と 70 度の 2 種類のコンテキスト刺激のどちらを使用するかを実験情報ダイアログで選ぶようにしました。ここで実験情報ダイアログに項目を追加せず、代わりに CounterBalance ルーチンを作成してフローの先頭 (厳密にはグループ名を必要とするルーチンやループより前) に挿入し、 **[グループ数]** を 2 にします (初期値のまま)。 **[グループごとのスロット]** と **[繰り返し回数]** は募集する参加者数に応じて決めます。そして、条件ファイル名 (exp04_20.xlsx と exp04_70.xlsx) を exp04_group0.xlsx、exp04_group1.xlsx のように **グループ番号だけが異なる名前** にします。先述の通り、選択されたグループ名は counterbalance.group で得られるのですから、trials ループに指定する条件ファイルを以下のようにすれば、グループに応じて条件ファイルを切り替えられます。グループ名は 0 や 1 といった数値ですがデータ型は str であることに注意してください (あと$も忘れずに！)。

```
'exp04_group'+counterbalance.group+'.xlsx'
```

グループごとに不均等な回数を割り当てる必要がある場合や、グループごとに切り替えたいパラメータがある場合は、 **[グループの定義...]** で「条件ファイル」を選択します。この条件ファイルには以下のパラメータが必要です。

group	グループ名。日本語の文字などの非 ASCII 文字も使えます。
cap	各グループに割り当てる回数。

これら以外のパラメータが与えられた場合、実験内では counterbalance.params[' パラメータ名'] という式で値を参照することができます。具体例として、 図 14.11 のような条件ファイルを考えます。

	A	B	C	D	E	F
1	group	cap	conditionsFile			
2	70度条件	10	exp04_70.xlsx			
3	20度条件	40	exp04_20.xlsx			
4						
5						
6						

図 14.11　CounterBalance ルーチン用の条件ファイルの例

「70 度条件」グループに 10 回、「20 度条件」グループに 40 回を割り振っていますので、50 回実験を行うと 10 回 (20%) が 70 度条件、40 回 (80%) が 20 度条件になります。さらに conditionsFile というパラメータを用意して対応する条件ファイル名を定義していますので、ループの条件ファイルは以下のように書くことができます。先の例のように exp04_group0.xlsx, exp04_group1.xlsx といった番号順の条件ファイル名よりも、内容がわかりやすくて管理しやすいでしょう。

```
'counterbalance.params['conditionsFile']
```

14.5.5 ユーザーインターフェースの変更

Builder, Coder, Runner の各ウィンドウにあったツールバーが、Microsoft Office のようなリボンに変更されました。とはいえ Office のようにリボンにタブがあるわけではないので使い勝手はあまり変わりません。アイコングループごとに「ファイル」や「編集」といったラベルが表示されるようになったので少しわかりやすくなったといえるでしょう。

Runner は大きくレイアウトが変更され、ウィンドウ内に大きなペインが二つ並ぶようになりました。左側は登録されている実験のリストで、右側に「注意」や「出力」などがタブ切り替えで表示されます。「注意」や「出力」の欄が格段に大きくなり、読みやすくなりました。また、Pavlovia 関係のメッセージが表示される「Pavlovia」というタブが新設されています。

続いて 2 点、 **Builder の使い勝手を大きく左右する機能が 2 つ追加されています。** ひとつめは Builder の「ファイル」メニューに追加された「ファイルエクスプローラで開く」です (図 14.12)。この項目を選択すると、現在 Builder で編集中の実験のフォルダが OS の標準ファイルエクスプローラーで開きます。

「実験の条件ファイルや刺激として使用する画像ファイルなどを、実験のフォルダに移動させる」という作業は Builder を使用していると度々必要となります。実験のフォルダをフォルダ階層の深い位置に置いていると開くのが面倒ですが、「ファイルエクスプローラで開く」を使うと一発で開くことが出来ます。なにより「Builder を習い始めた人が、自分が作っている実験のフォルダがどこなのかわからない」というトラブルがぐっと軽減されると思われます。Builder の使いかたを教える人は、できるだけ早い段階 (本書で言うと 2 章か 3 章) でこの機能を紹介し、4 章や 6 章など、複数の条件ファイルや画像ファイルを扱う作業を行うたびに強調するとよいと思います。

もうひとつは 2024.1.1 より前からある機能ですが、本書ではまだ紹介していないものです。Builder の「実験」メニューにある「実験内を検索...」という項目がそれです (図 14.13)。

「実験内を検索...」を選択すると、 図 14.14 のようなダイアログが表示され、一番上の入力欄に検索語を入力

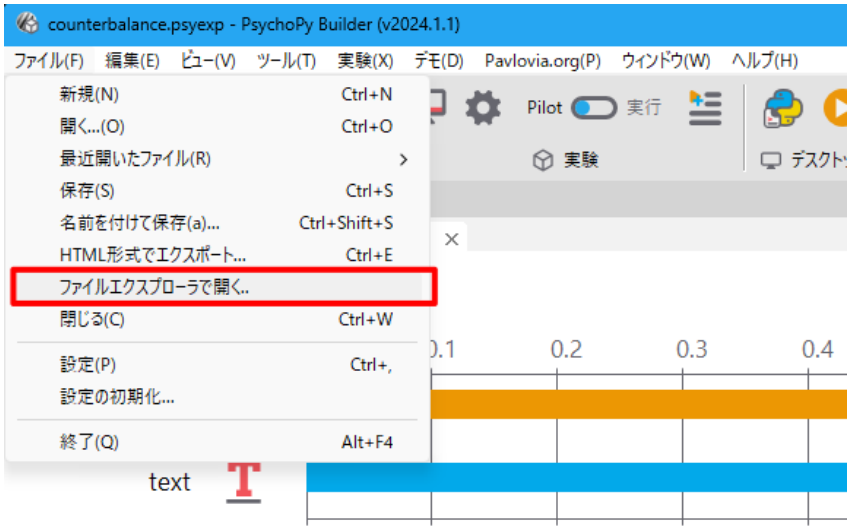


図 14.12 「ファイルエクスプローラで開く」メニュー

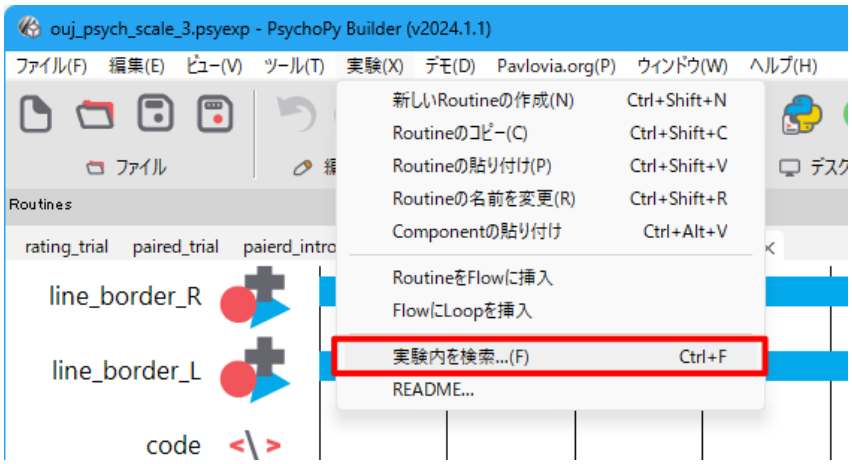


図 14.13 「実験内を検索...」メニュー

すると、その文字列をプロパティに含むコンポーネント (とそのプロパティ) が一覧表示されます。この例の order_images_list[0], order_images_list[1],... といったように多数のコンポーネントに少しずつ異なる値を設定する必要がある (そのためコピー&ペーストだけでは作成できない) 実験で入力内容に誤りがないか確認したい場合などに非常に便利な機能です。

本書では、第 4 章で教示画面を作成する時にこの機能の使いかたを紹介すると良いでしょうし、第 8 章や第 13 章のような実験を作成する時にも役立つでしょう。初心者からステップアップする際には確実に使いこなせるようになっておきたい機能だと思います。

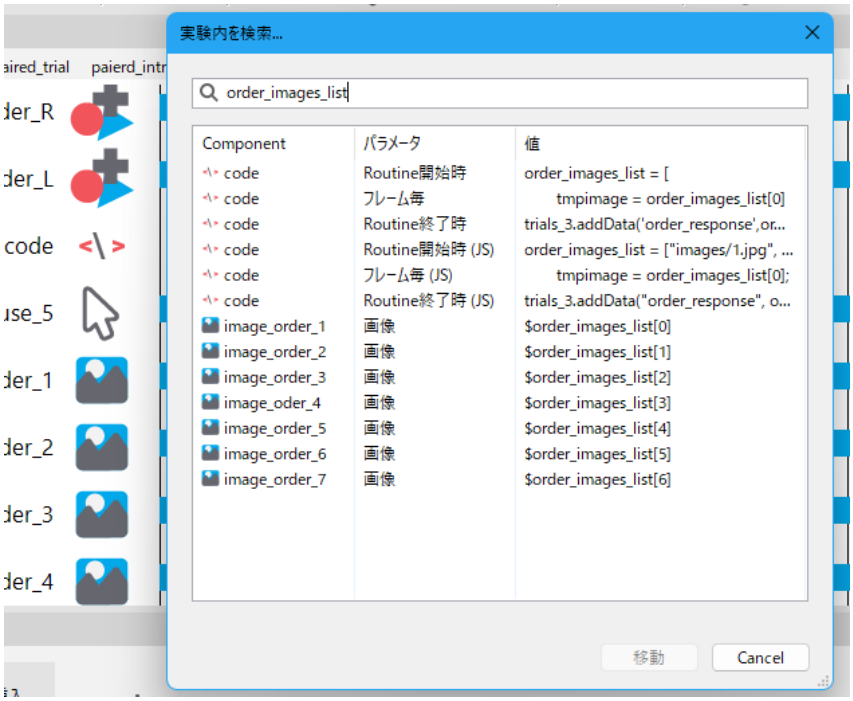


図 14.14　実験の検索ダイアログ

14.5.6 実験の設定ダイアログの変更

実験の設定ダイアログの変更をまとめておきます。

- 基本タブ
  - **[実行モード]** の追加
  - **[rush モード]** の追加
- スクリーンタブ
  - **[ウィンドウバックエンド]** の追加
  - **[色]** は **[背景色]** に変更
  - **[背景画像]** の追加
  - **[背景画像の伸縮]** の追加
  - **[フレームレートの測定]** の追加
  - **[フレームレート測定時のメッセージ]** の追加 フレームレート測定をおこなわない場合は **[フレームレート]** になる
- データタブ
  - **[列の並び替え...]** の追加
  - **[列の優先度]** の追加

- **[時刻のフォーマット]** の追加

大部分はここまで紹介したものに関係していますが、「データ」タブは補足が必要でしょう。Builder が出力するデータファイルにおける列の並び順が **[列の並び替え...]** で変更できるようになりました。「アルファベット順」は文字通り列名のアルファベット順です。「追加順」というのは実験の実行中にデータファイルに追加された順番で、従来の Builder と同じです。説明が必要なのは「優先度」で、これを選択すると **[列の優先度]** で指定した優先度にしたがって並び替えられます。以下の定数が定義されています。定数の値は変更される可能性があるので書くべきではないかもしれませんが、一応書いておきます。

priority.CRITICAL	30
priority.HIGH	20
priority.MEDIUM	10
priority.LOW	0
priority.EXCLUDE	-10

注意が必要なのは、列名は正確にデータファイルに出力される名前でなければならないということです。例えば target の出現時刻の列を指定したいのなら、target と書くのではなく target.started と書かないといけません。あと、優先度が同じもの同士は筆者が確認した範囲では辞書の逆順に並んでいるようです。

[時刻のフォーマット] は、現時点 (2024.1.1) ではログファイルに出力される時刻の形式を選択するもののようです。例えば **[時刻のフォーマット]** が「実験開始時から」だと

```
0.0280      EXP     target: autoDraw = True
0.0280      EXP     frame: autoDraw = True
1.0253      EXP     target: autoDraw = False
1.0253      EXP     frame: autoDraw = False
```

のように出力されますが、「実時間」にすると

```
2024-04-22_10:37:37.565335  EXP     target: autoDraw = True
2024-04-22_10:37:37.565335  EXP     frame: autoDraw = True
2024-04-22_10:37:38.551738  EXP     target: autoDraw = False
2024-04-22_10:37:38.551738  EXP     frame: autoDraw = False
```

のように出力されます。「実験開始から」が従来の動作です。

14.5.7 翻訳の問題

「スクリーン」タブの **[色]** が **[背景色]** に変更されたのはすでに述べた通りですが、他にも Loop のプロパティの **[繰り返し条件]** が **[条件]** に変更されました。原語は"Conditions"で変更ないのですが、CounterBalance ルーチンの追加により"Conditions"が必ずしも繰り返しの条件を指定するものではなくなったためです。