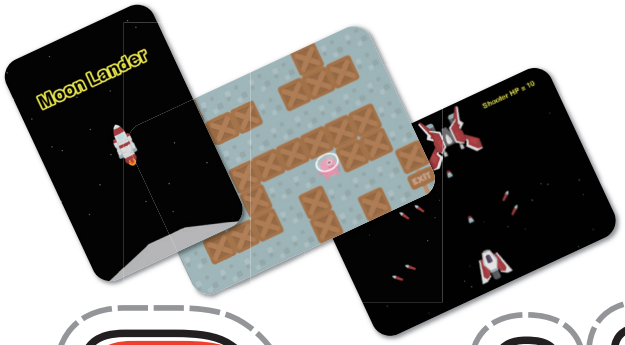


5日でできる!
Pythonでゲーム作成入門

中島 省吾（メディアプラネット）著

Pythonで
一番簡単にゲームを
作れるPygame Zeroを
使って、迷路や月面着陸
ゲーム、シューティング
ゲームを作ろう

日経ソフトウエア
NIKKEI SOFTWARE
2020年5月号 第2付録

特別付録② 5日でできる! Pythonでゲーム作成入門

日経ソフトウエア
NIKKEI SOFTWARE

1日目
Pythonの概要と環境構築

1日目 Part 1 Pythonって?

地元の公立中学校に通う幼なじみの錦田達也(タツヤ)と佐藤麗(レイ)は、ゲーム作りに興味があって、同じプログラミング教室に通っている。2人とも「Scratchコース」でプログラミングの基本を学んでいたが、タツヤが本格的なゲームを作りたくなり、レイを誘って「Pythonゲームコース」へ移動することに…。

先生 こんにちは。Pythonゲームコースへようこそ。講師の中島です。

タツヤ こんにち…って、なんだぁ〜、Scratchコースと同じセンセイ。

レイ 結局、同じメンバーなのね。

先生 どうして2人ともScratchコースをやめたんだい?

タツヤ やっぱさぁ〜、僕にScratchは簡単過ぎるんだよねぇ。

レイ フフッ。「猫が動いた」って喜んでたくせに〜。

先生 確かに、中学生になるとScratchじゃ物足りないかもね。

タツヤ そう、PythonでAI(人工知能)ゲームを作るんだ。

レイ 私は、マリオカートみたいなやつがいい。

先生 Pythonは「ライブラリ」が豊富だから、どんなゲームだって作れると思うよ。

レイ ライブラリって「図書館」のこと?

先生 プログラミングでは、全部をゼロから作るのは大変だから、プログラムのための「便

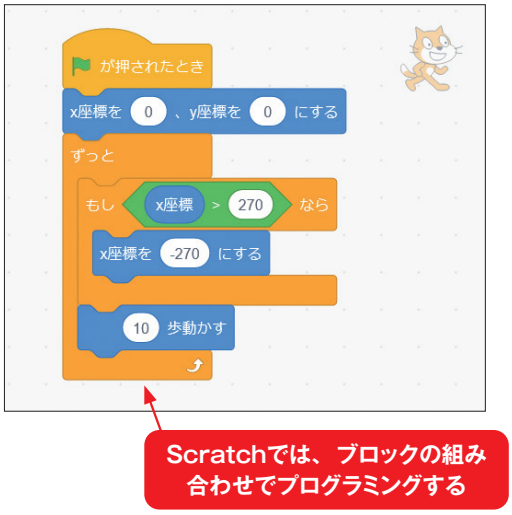
利な部品」がたくさん用意されている。この便利な部品は「オブジェクト」と呼ばれているんだけど、そのオブジェクトを集めたものが「ライブラリ」なんだ。

タツヤ 知ってるよ。Pythonは「オブジェクト指向プログラミング言語」だからね。

先生 ほぅ、タツヤ君、すごいね。Scratchは子どもでもプログラミングできるように「ブロック」を組み合わせてプログラムを作るけど、Pythonは英語をキーボードから入力して、“オブジェクトの振る舞い”でプログラムを作る。この英語の文を「ソースコード」と呼ぶんだ（**図1**）。Pythonは、ほかのプログラミング言語に比べて、とてもシンプルにソースコードを書ける言語なんだよ。

図1●ScratchとPython

Scratch



Python

```
import pgzrun

WIDTH = 200
HEIGHT = 200

alien = Actor('alien', center=(100,100))

def draw():
    screen.clear()
    alien.draw()

def update():
    alien.left += 2
    if alien.left > WIDTH:
        alien.right = 0

pgzrun.go()
```

Pythonでは、英語の文（オブジェクトの振る舞い）でプログラミングする

レイ 英語なら得意よ! 私、英検2級持ってるもん。

タツヤ マ、マジ…? いつの間に…?

先生 プログラミングで使う単語は限られてるよ。英会話のように何千もの単語を覚える必要はないからずっと楽だよ。せいぜい100個くらい覚えれば一通りのことはできる。

タツヤ 100個って…、結構多くない?

先生 例えば、Scratchなら文字を表示するには「 ～と言う」ブロックを使うよね。同じことをPythonでやるには、「print関数」を使うんだ。

レイ print？印刷?

先生 プログラムでは、プリンタで文字を印刷することも、画面に文字を表示することも「print（プリント）する」って考える。つまり、文字を機械に出力するための命令だね。あと、便利な命令や、複数の命令をまとめたものを「関数」と呼んでいる。

タツヤ 関数って、数学の関数のこと?

先生 数学の関数は、入力の値が決まると出力の値も決まる式のことだよね。Pythonの「関数」は、Scratchの「ブロック定義」ように、いろいろな命令をまとめて名前を付けたものだ。もっとも、数学の関数にそっくりな関数もある。それと、関数もオブジェクトなんだ。

レイ また、「オブジェクト」が出てきた。Pythonでプログラミングするときは「オブジェクト」が重要なのね。

タツヤ そうみたい。

先生 それでは、休憩してPythonをパソコンにインストールしよう。

1日目 Part 2 プログラミング環境の構築

先生 それでは、Pythonを使ってプログラムを作れるように「環境」をインストールしよう。

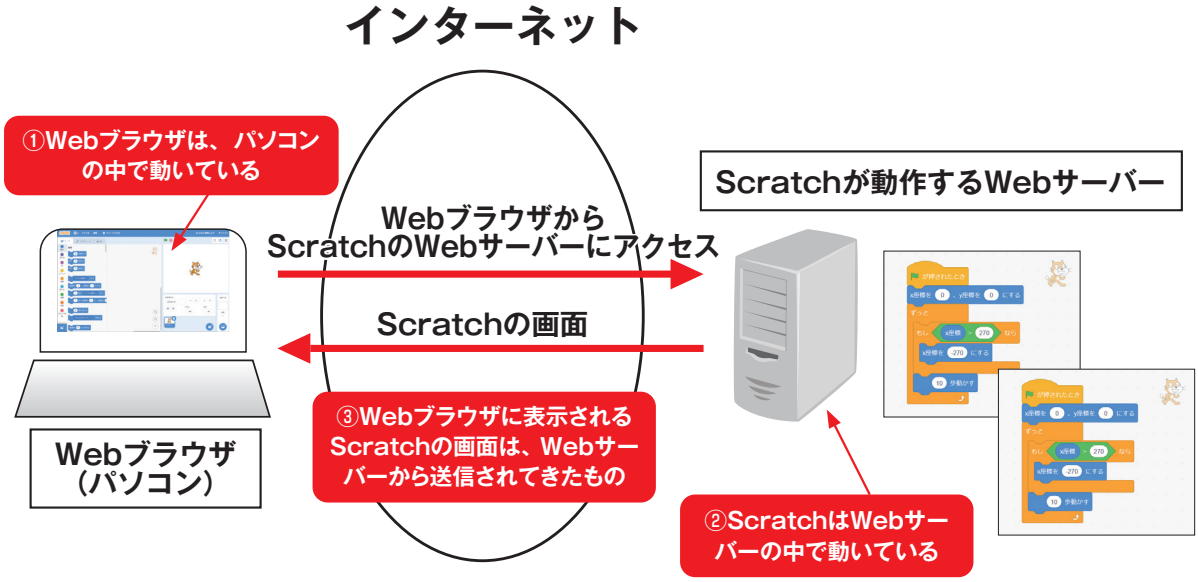
タツヤ 環境って?

先生 Scratchでは、ブラウザにプログラミング環境が表示されるだろ。つまり、インターネットとブラウザがあれば、すぐにプログラミングを始められる。

レイ ブラウザって、ChromeとかFirefoxのことね。

先生 そうそう。正式に言うと「Webブラウザ」。Webブラウザは、インターネット上のサーバーにアクセスしてサイトの情報などを表示するアプリだよね。だから、Scratchで作ったプログラムはScratchのサーバーに保存される。このように、プログラミングをしたり、プログラムを保存したり、作ったプログラムを動かしたりする場所のことを「環境」と呼ぶんだ。Scratchなら、その環境はWebブラウザとWebサーバーで構成される（**図1**）。

図1●Scratchの環境



レイ だから、外国の子どもたちが作ったゲームを、日本でも遊べるのね。

先生 一方、Pythonでは、それぞれのパソコンにPythonの環境を用意するのが普通なんだ。パソコンの環境でプログラムを作って、パソコンに保存して、パソコンで動かす。インターネットは事実上は必須なんだけど、仮にインターネットがなくてもPythonのプログラミングはできる。ちなみに、Scratchにも「Scratchデスクトップ」っていう、インターネットに接続せずに、パソコンにインストールして使うタイプの環境があるんだけどね。

レイ 私たちのパソコンはWindows 10だけど、Pythonの環境は入ってないの?

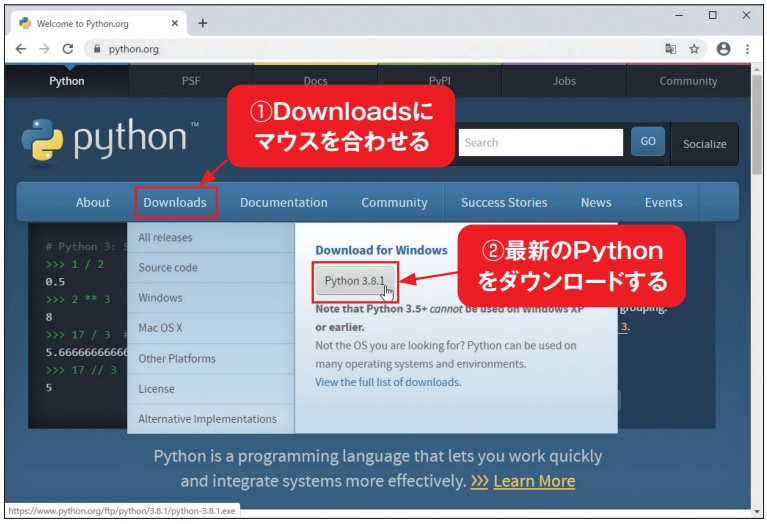
先生 パソコンのOSがLinuxやmacOSの場合は、初めからPythonの環境が入っていることが多い。でも、Windowsの場合は、インターネットからダウンロードしてインストールす

るんだ。

タツヤ でも…、お高いんでしょう?

先生 Pythonの環境は、無料で利用できるものがほとんどだから大丈夫だ。それでは、自分のWindowsパソコンを起動して、Pythonの公式サイト(https://www.python.org/)へアクセスしよう。現在のPythonのバージョンは3.x.xというものだから、公式サイトにある「Python 3.x.x」のボタンをクリックしてダウンロードする(**図2**)。

図2●Python公式サイト
https://www.python.org/



レイ ダウンロードできたわ。

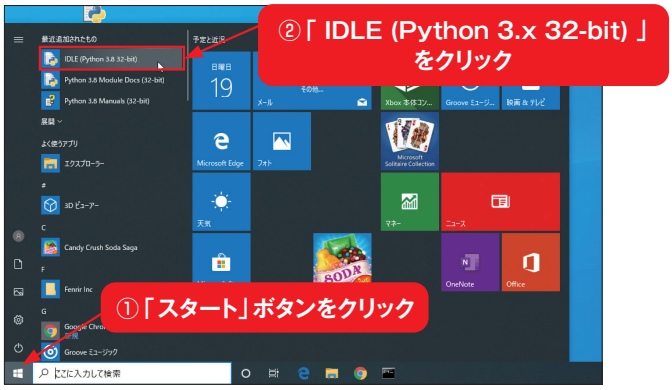
先生 「python-3.x.x.exe」ってファイルをダウンロードできたと思う。これがWindowsにPythonの環境を導入するインストーラーだね。ダブルクリックして起動しよう。インストーラーの画面が表示されるので、下にある「Add Python 3.x to PATH」のところにチェックを入れて、「Install Now」をクリックする(**図3**)。以降は、デフォルトで大丈夫だ。最後に「Close」ボタンをクリックして終了する。

図3●Windows用のインストーラー

タツヤ インストール完了!

先生 まずは、Pythonが正しくインストールされたかどうかを確認する。Windowsのスタートボタンをクリックすると、「IDLE (Python 3.x 32-bit)」がある。これをクリックしよう（**図4**）。

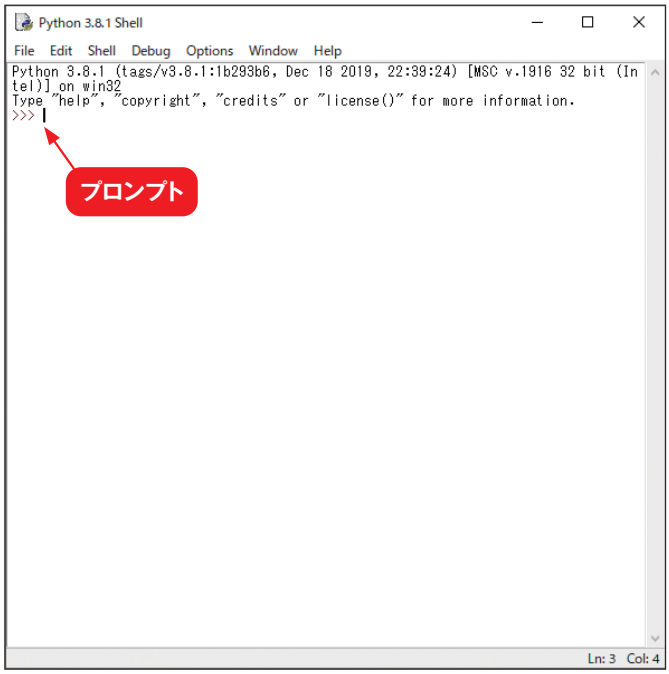
図4●Python 3.xのメニューが追加された



レイ ウインドウが起動したわ（**図5**）。

先生 この「IDLE」は「Integrated DeveLopment Environment」の略だ。日本語だと「統合開発環境」、つまり「プログラムを作るために必要なツールを統合した環境」ってことだね。IDLEでは、Pythonのコードを直接入力したり実行したりできるんだ。「>>>」の部分を「プロンプト」と言う。この「>>>」に続けてプログラムを入力するんだ。

図5●IDLEの起動



タツヤ 確かに。「ここに入力しろ」って感じだ…。

先生 では、プロンプトに、「print('こんにちは、パイソン!')」と入力して、「Enter」キーを押してみよう。コードの入力は「'」（シングルクオーテーション）で囲まれている日本語以外は、記号も含めてすべて半角文字で入力すること。

レイ プロンプトの下に「こんにちは、パイソン!」って出たわ。

先生 これがprint関数を使ったPythonプログラムだ。

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24)
[MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for
more information.
>>> print('こんにちは、パイソン!')
こんにちは、パイソン!
>>>
```

最後に「Enter」キーを押す

入力した文字列が表示された

タツヤ イヤイヤイヤ、センセ〜。これはさぁ、入力した文字を表示してるだけでしょ。

先生 確かに。でも、これがprint関数の仕事だ。print関数には、括弧の中のオブジェクトを画面に表示する役割がある。とても単純だけど、非常によく使う関数だ。注意する点は、さっきも言ったけど、Pythonのコードは半角文字で入力する。そして、オブジェクトが文字列だった場合は、半角文字のシングルクオーテーションかダブルクォーテーションで括る。文字列は半角でも全角でも、どちらでもいい。

print関数

```
print('こんにちは、パイソン!')
```

表示したい文字列を、半角のシングルクオーテーションかダブルクオーテーションで括って入力する

レイ 文字もオブジェクトなの?

先生 そう。Pythonではソースコードに登場するすべてがオブジェクトなんだ。今回の「print関数」も「文字列」も、「1」とか「3.14」といった数値も、ぜ〜んぶ「オブジェクト」だね。

タツヤ 「オブジェクトはプログラムの便利な部品」って意味が、少し理解できた。

先生 タツヤ君、すばらしい。オブジェクト指向のオブジェクトは、独立した小さなプログラムなんだ。単体では機能が限定されているけど、組み合わせることでゲームやアプリになる。

レイ Scratchもそうだったけど、Pythonのプログラミングもいろいろな部品を組み合わ

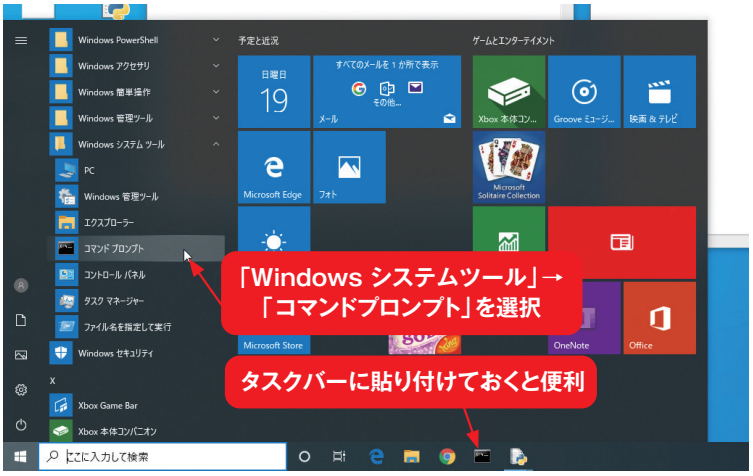
せるパズルみたいな感じなのね。

先生 それでは、Pythonのコードをテキストファイルに保存してから実行する方法を紹介しよう。Pythonでゲームを作るには、この方法を知らないと不便だからね。まず、Windowsのコマンドプロンプトを起動して「mkdir」コマンドでCドライブにPythonフォルダーを作る。

タツヤ コマンドプロンプト…。また、よくわからない言葉が…。

先生 Pythonフォルダーはエクスプローラーでも作れるけど、今回は、コマンドプロンプトで作ってみるよ。「コマンドプロンプト」とは、プログラムを起動するためのキーワードである「コマンド」を、キーボードから入力するための環境だ。Windowsのスタートボタンをクリックして表示される「システムツール」メニューの中にある（**図6**）。

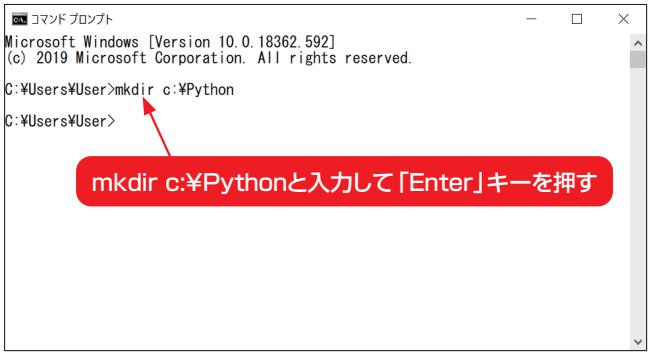
図6●コマンドプロンプトを起動



タツヤ お、何か画面が出てきた…。

先生 ここでフォルダーを作るコマンドは「mkdir」。「make directory」の略だ。ディレクトリはフォルダーのことだ。プロンプトに「mkdir c:¥Python」と半角文字で入力する。「Enter」キーを押すとCドライブの直下にPythonフォルダーができる（**図7**）。

図7●コマンドによるフォルダーの作成



レイ ちゃんとできてる（**図8**）。エクスプローラーで作るより簡単かも…。

タツヤ マウスは、時代遅れなのさ。

先生 それでは、コマンドプロンプトはそのままにして、IDLEの「File」メニューから「New File」を選択しよう。すると、IDLEのエディターが起動するから、ここにprint関数のコードを入力する（**図9**）。

図8●Pythonフォルダーが作られた

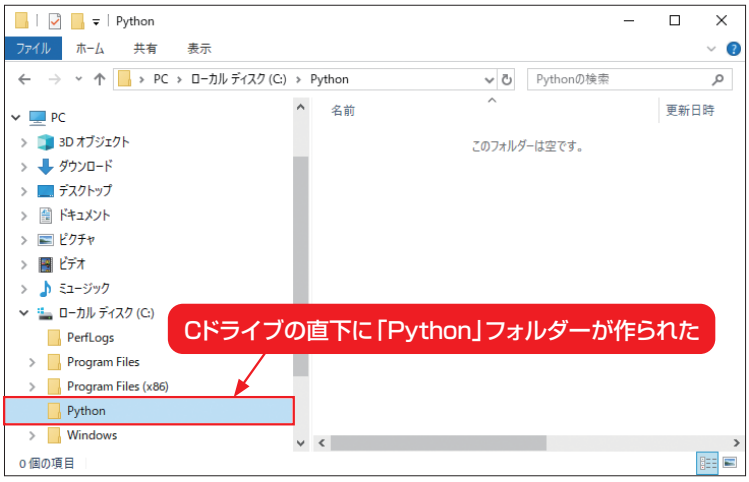
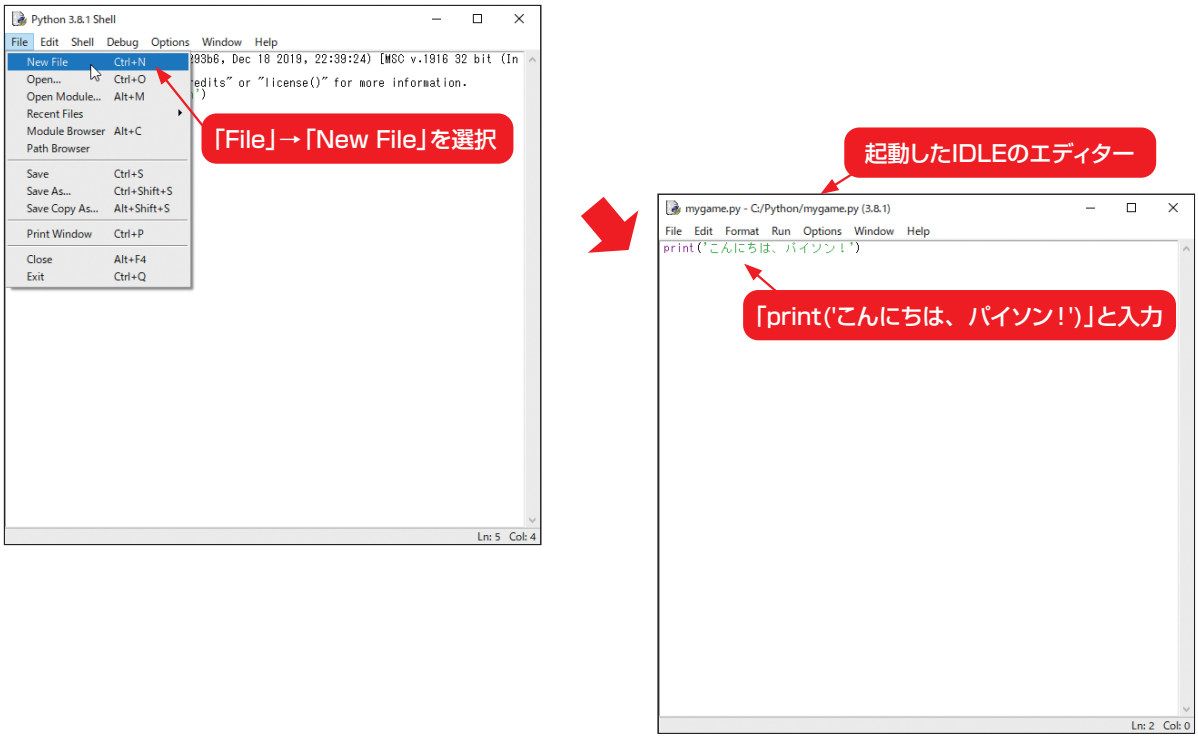
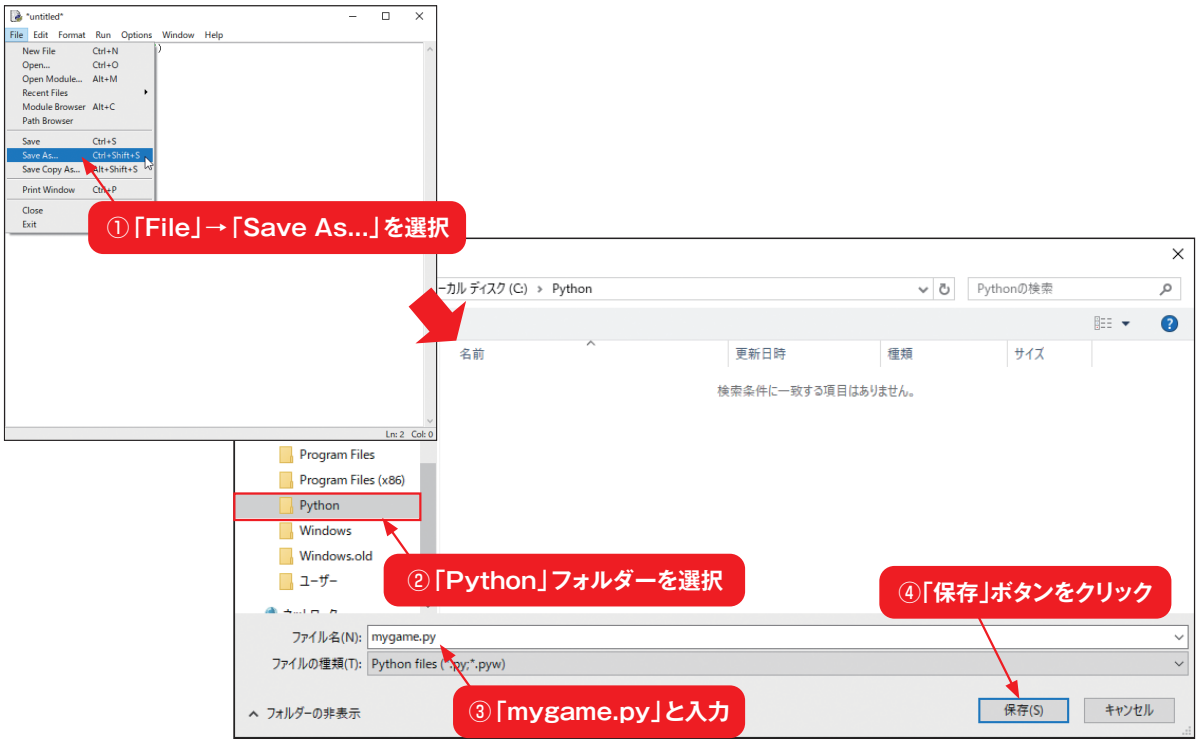


図9●エディターを表示



先生 できたら、メニューの「File」→「Save As...」を選択して、さっき作ったPythonフォルダーを選択する。ファイル名は、mygame.pyとしよう（**図10**）。

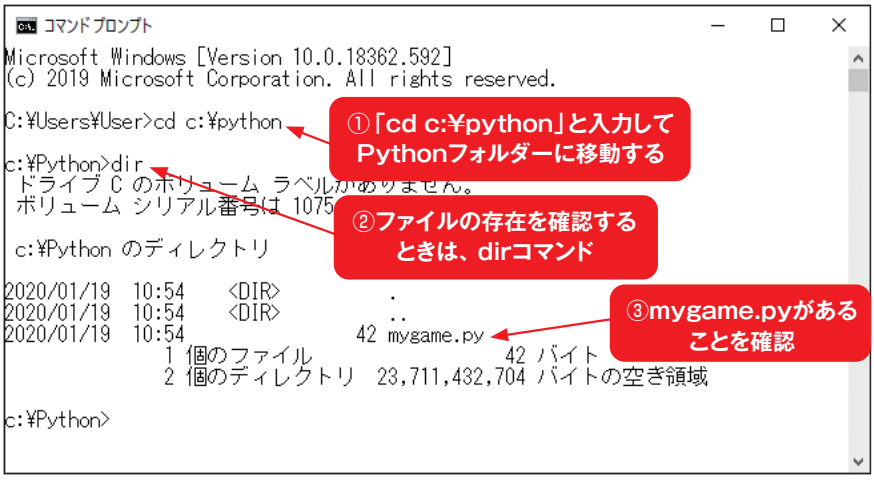
図10●ソースコードをファイルにして保存



タツヤ プログラミングっぽくなってきたぞぅ。

先生 保存できたら、コマンドプロンプトで作業するフォルダーをPythonフォルダーにする。作業フォルダーの移動には「cd」コマンドを使う。これは「change directory」の略だ。コマンドプロンプトで「cd c:¥Python」と入力して「Enter」キーを押す。プロンプトの部分が「c:¥Python>」のように変わるはずだ。この作業しているフォルダーを「カレントフォルダー」と呼ぶ。カレントフォルダーにmygame.pyがあるかどうか、コマンドで調べるには「dir」コマンドを使う（**図11**）。

図11●フォルダーの移動とファイルの確認



レイ ちゃんとmygame.pyがあるわ。

先生 それでは、実行してみよう。コマンドプロンプトに「python mygame.py」と入力して「Enter」キーを押す。

タツヤ あれ…? 何も起こらない…。

レイ 私も〜。

先生 …ん? あっ、そうだった、そうだった。最近のWindows 10では設定が1つ必要な場合があるんだ。まず、Windowsのスタートボタンをクリックして「設定」を開いてくれるかな。

レイ は〜い。

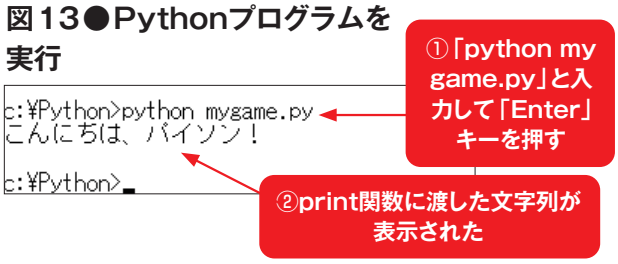
先生 そしたら「アプリ」を選んで、その中にある「アプリ実行エイリアス」を選択する。次に出てくる画面で、「アプリ インストーラー python.exe」が「オン」になっていたら、「オフ」に変更しよう(**図12**)。

図12●アプリ実行エイリアスの画面

タツヤ 「オン」になってたから「オフ」にしたぞ。

先生 それではもう一度、「python mygame.py」を入力してみよう(**図13**)。

図13●Pythonプログラムを実行



タツヤ 表示できた。なるほど、Pythonの動かし方がわかったぞ。

先生 それでは、次はエイリアンに登場してもらおう。

レイ エ、イ、リ、ア、ン!!

1日目 Part 3 画像の表示

先生 それでは、Pythonで画像を表示するプログラムを作ろう。ゲームプログラム作成の第一歩だね。Scratchの場合はあらかじめスプライトが用意されていたけど、Pythonでは自分で絵を描くかインターネットからダウンロードして用意する。

タツヤ よ〜し、それじゃ“進撃キャラ”を使おう。

先生 おっと、それは問題アリなんだ。アニメや漫画の画像には、基本、「著作権」という権利があって、無断で使用することはできないんだ。

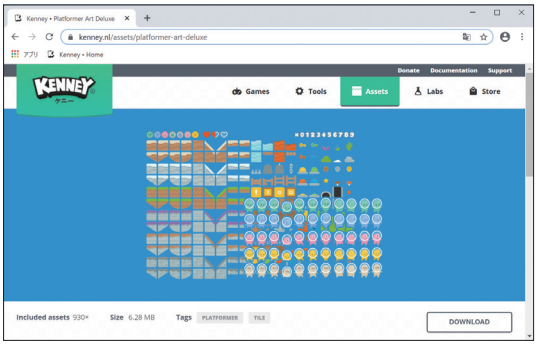
タツヤ じゃあ、自分で描くしかないのか。ちょっと大変だなぁ。

先生 確かに。ゲームで使う画像の作成は、プログラム以上に大変だったりする。そこで今回は「ケニー」（https://kenney.nl/）というサイトにある自由に使えるゲーム素材を利用しよう。次のURLにアクセスして欲しい。

```
https://kenney.nl/assets/platformer-art-deluxe
```

先生 これは「Platformer Art Deluxe」という素材集だ（**図1**）。

図1●ケニーのサイトで公開されているゲーム向け素材
https://kenney.nl/assets/platformer-art-deluxe



タツヤ すごい。これ全部自由に使っていいの?

先生 大丈夫だよ。ケニーのサイトには、こんな「すぐに使える素材」が、何十種類も用意されている。今回使うPlatformer Art Deluxeにはエイリアンや地形の素材がまとまっている。「DOWNLOAD」を押して、「platformer-art-complete-pack-0.zip」のファイルをダウンロードしよう。ダウンロードできたら、エクスプローラーでダブルクリックして開いてみよう。

レイ このファイル、中にフォルダーが入ってる。

先生「zip」という拡張子を持つファイルは「圧縮ファイル」といって、複数のフォルダーやファイルを圧縮して、1つのファイルにまとめたものだ。使う時は「解凍」または「展開」といって、元のフォルダーやファイルに戻してあげなきゃいけない。解凍の仕方はいろいろあるけど、今回は言われた通りに作業してみてほしい。

レイハ〜イ。

先生最初に、Pythonフォルダーに「images」フォルダーを作り、その中に「platformer-art-complete-pack-0」フォルダーを作る。

タツヤ圧縮ファイルのファイル名と同じフォルダーを作る…。

先生「platformer-art-complete-pack-0.zip」をダブルクリックして、エクスプローラーで表示する。そして、中のフォルダーやファイルを全部「platformer-art-complete-pack-0」フォルダーにコピーする。できたら、エクスプローラーで「Base pack」→「Player」→「p1_walk」→「PNG」とたどってみよう。

レイなんか、カワイイキャラクターが出てきた（**図2**）。

先生これが、エイリアンの画像だ。もし、気に入らなければ、p2_walkフォルダーやp3_walkフォルダーに別の色のエイリアンがいる。お気に入りのエイリアンを見つけたら、作成したimagesフォルダーの直下にコピーする。コピーができたら、名前を「alien.png」に変更しておこう。

図2●エイリアンの画像が出てきた

レイ私は、p3_walkフォルダーのピンクのエイリアンにしようっと。「p3_walk03.png」を

imagesフォルダーの直下にコピーして、名前を「alien.png」に変更ね。

タツヤ 僕は、「p1_walk」の「p1_walk03.png」にするよ。名前を「alien.png」に変更っと。

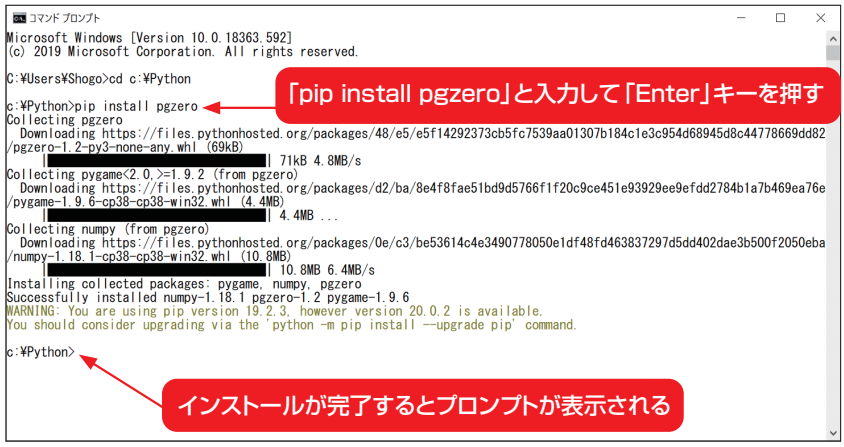
先生 これで、表示する画像を用意できた。次は、いよいよPythonで画像を表示する。Pythonで画像を表示する方法はいくつかあるけど、このPythonゲームコースでは「Pygame Zero」（パイゲームゼロ）というゲーム用ライブラリを使ってゲームを作るんだ。だから、画像の表示にもPygame Zeroを使おう。

タツヤ キターーー。ゲーム用ライブラリってスゴくね。

先生 Pythonのライブラリはインストールが簡単だ。「Pygame Zero」もコマンドプロンプトで「pip install pgzero」と入力して「Enter」キーを押すだけだ。

タツヤ 何か、いっぱい英語が出るけど…。インストールは、終わったみたい（**図3**）。

図3●Pygame Zeroをインストール

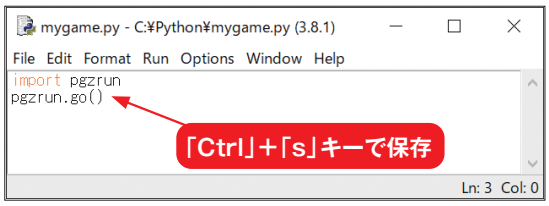


先生 それでは、IDLEのメニュー「File」→「Open」をクリックして、Cドライブの Pythonフォルダーにあるmygame.pyを選択しよう。コードがエディターに表示されたら、次のコードに書きかえる。

```
import pgzrun
pgzrun.go()
```

mygame.pyをこのように書きかえる

図4●保存する



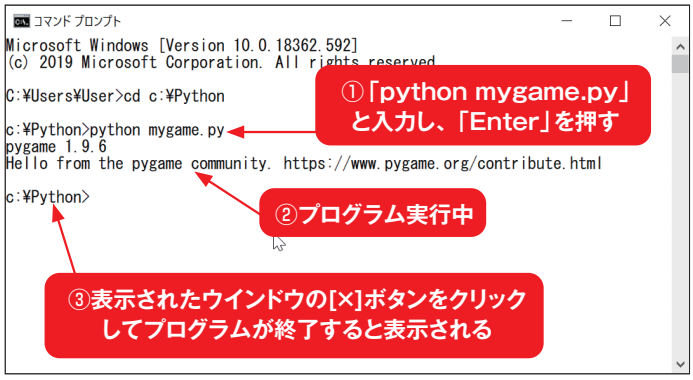
レイ 完了!!

先生 できたら保存しよう。保存するときは、メニューから選ぶより、「Ctrl」+「s」キーを押す方が楽だから、試してごらん（**図4**）。

レイ あ、保存できたみたい。超カンタン！ホント、タツヤの言うようにマウスは時代遅れかも。

先生 それでは実行してみよう。コマンドプロンプトの画面でPythonフォルダーに移動したら、「python mygame.py」と入力して「Enter」キーを押す（**図5**）。

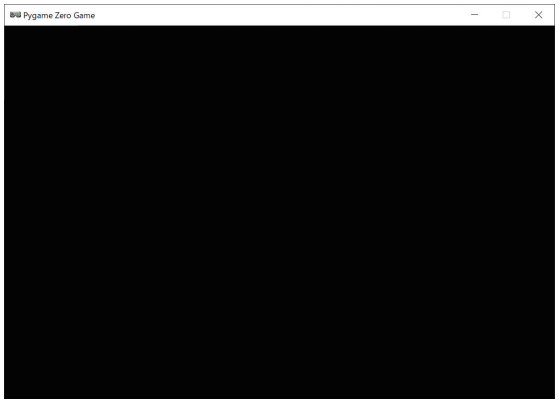
図5●コマンドプロンプトから実行する



タツヤ 真っ黒いウインドウが出たぞぅ（**図6**）。

先生 成功だ。これが、mygame.pyの実行結果だよ。Pygame Zeroなら、たった2行でウインドウを表示できるんだ。では、起動したウインドウを右上の[×]ボタンでクローズしよう。ここではコマンドプロンプトから起動したけど、IDLEから起動することもできる。IDLEのメニュー「Run」→「Run Module」をクリックするか、IDLEを前面にしておいてから「F5」キーを押してみよう（**図7**）。どちらでも、mygame.pyを実行してウインドウを表示できる。

図6●mygame.pyの実行結果
mygame.pyの実行で表示されたウインドウ



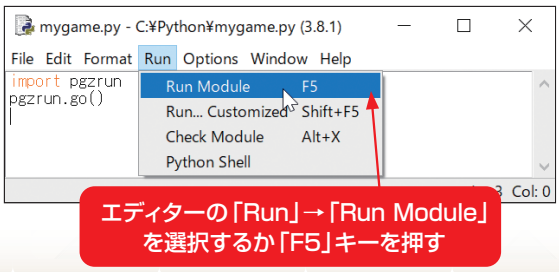
レイ F5キー、速ッ。私、これがいい。

タツヤ レイ様が、お気に召したようです。

レイ でも、エイリアンはどうなったの?

先生 これからそれを説明するよ。エイリアンを表示するために、mygame.pyのコードを、次のように書き換えよう。

図7●IDLEのエディターから実行する



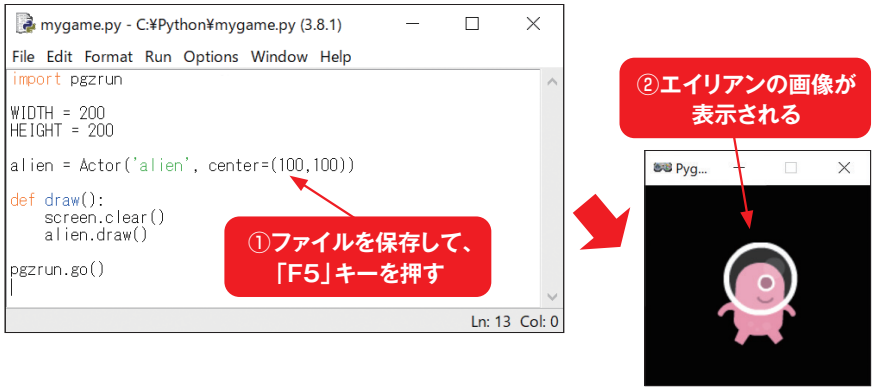
mygame.py

```
import pgzrun

WIDTH = 200
HEIGHT = 200

alien = Actor('alien', center=(100,100))

def draw():
    screen.clear()
    alien.draw()

pgzrun.go()
```

タツヤ 一気にコードの量が増えたんですけど…。

レイ 私、入力できた。保存して実行してもいい?

図8●エイリアンの画像が表示された



先生 どうぞ。どうぞ。

レイ キャ〜、私のエイリアン(図8)。

タツヤ うそ〜、なぜだぁ。こっちは小さいウインドウしか表示しないよぅ。IDLEの方に、なんか赤い英語がたくさん出るし〜。

先生 コードを見せてごらん…。「Actor」オブジェクトの名前を「actor」と入力しているよ。IDLEには、NameErrorになったと表示されている。

タツヤ え? 同じじゃない?

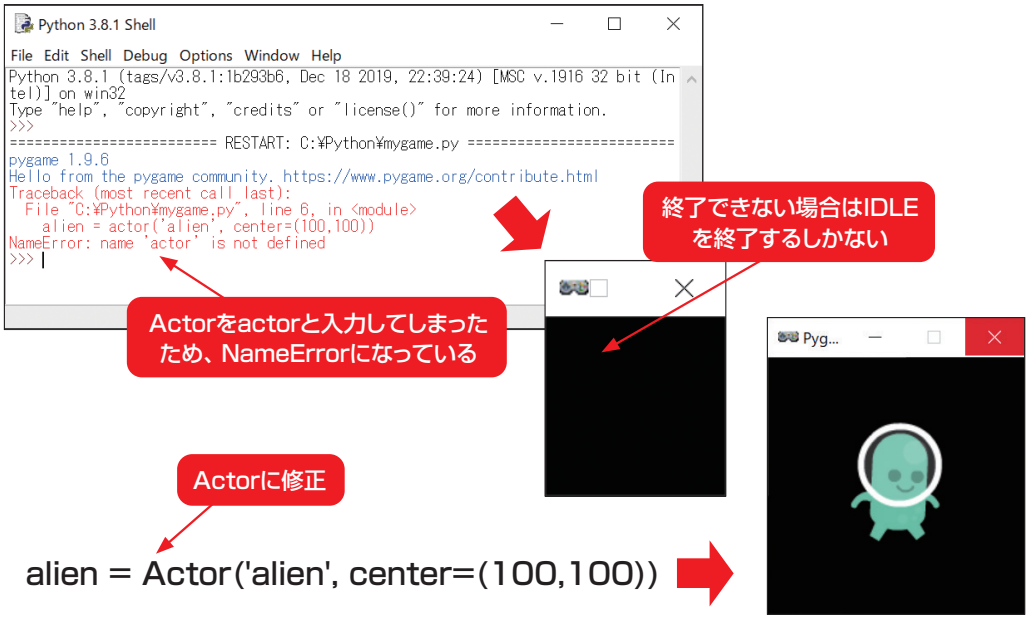
先生 違う、違う。Actorとactor、つまりアルファベットの大文字と小文字は区別されるんだ。だから、ActorはActorと入力しないとエラーになる。

タツヤ こまけーーー!

先生 この厳密さが良くも悪くもプログラミングなんだ。で、表示したウインドウが閉じなくなってるから、一度IDLEを終了して再起動後、mygame.pyを修正しよう。

タツヤ めんどくさいぃ〜。まず、IDLEの[×]ボタンで終了して、再起動。エディターでmygame.py開いて、修正、保存、実行（**図9**）。

図9●コードの修正と実行



タツヤ ヨシ！出た。エイリアンは、出た、け・ど・も…。

先生 どうしたんだい。

タツヤ プログラムの意味が、ぜんぜんわかりません！エッヘン。

レイ ホント…。

先生 大丈夫、大丈夫。Pythonの勉強は始まったばかりだからね。今日は、無事に環境ができたってことでお終りにしよう。次回は、エイリアンを動かしてみるよ。

レイ わかりました。がんばりま〜す。

2日目
Pygame Zeroの基本

2日目 Part 1 スクリーンの表示

先生 前回でPygame Zeroを使って画像を表示できたので、今日はスプライトが動くところまでプログラミングしよう。

タツヤ きっと、Pygame ZeroにもScratchみたいに「10歩動かす」オブジェクトがあるはずだ。

レイ そうね、Pythonはオブジェクト指向だもの。

先生 残念ながらPygame Zeroには「10歩動かす」オブジェクトはないよ。だけど、「スプライト」オブジェクトの座標を変更して移動させる点はScratchと同じだ。

レイ 座標か…。でも、Scratchのような「ステージ」はないでしょう。

先生 Pygame Zeroでは、最初に表示した「真っ黒ウインドウ」がScratchの「ステージ」に相当するものだと考えていい。真っ黒ウインドウは「screen」（スクリーン）と呼ばれている。

タツヤ 入力したあの2行のプログラムは、スクリーンを表示するコードだったのか（**図1**）。

図1●スクリーンを表示するコード

先生 最初のコード「import pgzrun」は、「import（インポート）文」と言う。importの後ろの「pgzrun」はオブジェクトだ。つまり、「import pgzrun」には「pgzrunオブジェクトをプログラムに取り込んで利用できるようにする」という意味がある。

レイ importは「輸入する」って習うけど、プログラミングでは「取り込む」ってニュアンスなのね。

先生 さすが、レイちゃん。そしてpgzrunオブジェクトの「goメソッド」を呼び出すと、スク

リーンが表示される。スクリーンのサイズを変更したいときは「WIDTH」と「HEIGHT」という変数で指定する。

タツヤ メソッドを呼び出す…?

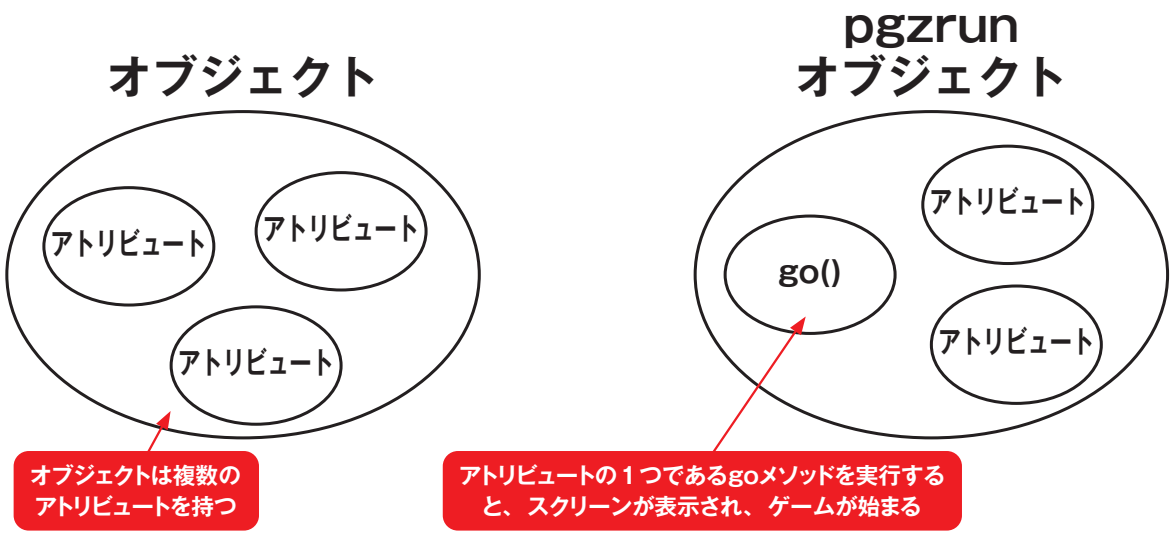
レイ メソッドって「方法」って意味でしょ。「行く方法…」って、意味わかんない。

先生 メソッドとは、複数の命令をまとめたもので「関数」と同じものだ。ただ、「メソッド」と言った場合は「オブジェクトのアトリビュート(attribute)」という意味がある。

タツヤ アトリ…? いきなり、難しくなってきたぞぅ。

先生 順を追って説明するよ。まず、オブジェクトは様々な属性を持っている。Pythonでは、属性をアトリビュートと呼ぶから、goメソッドはpgzrunオブジェクトが持つアトリビュートの1つだ。そして、オブジェクトのアトリビュートは、自分やほかのオブジェクトから利用される。つまり、goメソッドを利用すると、スクリーンが表示されてゲームが始まる。このようにメソッドを「利用する」、つまり「実行する」ことを「メソッドを呼び出す」と言うんだ(**図2**)。

図2●pgzrun.go()の意味



タツヤ うーん、わかったような、わからないような…。要はgoメソッドは「スタートボタン」ってこと…?

先生 まあ、そういうことかな。ここら辺はプログラミングしているうちに、だんだんわかってくるよ。

タツヤ だといいけど…。

先生 メソッドの呼び出しは、基本「オブジェクト名.メソッド名()」のように書く。メソッドに渡すオブジェクトがある場合は、メソッド名の後ろにある括弧の中に記述する。このように「オブジェクトに渡すオブジェクト」を「メソッドの引数（ひきすう）」と呼んでいる。

タツヤ 「引数」は、Scratchの「定義ブロック」で知ってるよ。メソッドに渡すデータってことだ。

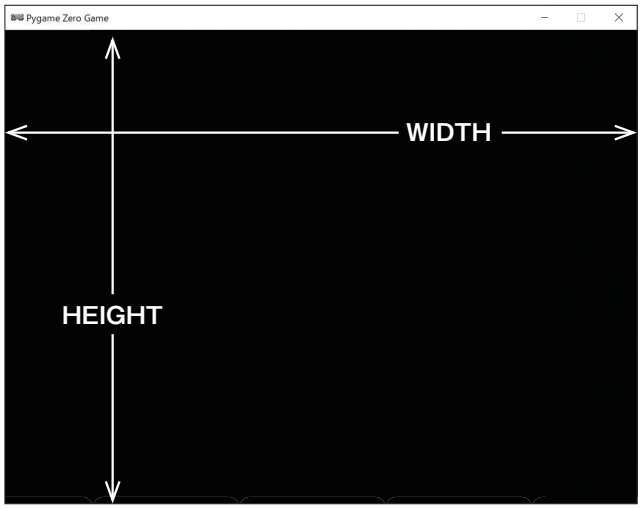
レイ でも、「go」っていうメソッド名には何だか違和感があるわ。スタートボタンなら、「pgzrun.start()」の方が良くない?

先生 メソッドの名前は、Pygame Zeroを作った開発者が決めているから変更できないんだ。もしかしたら、将来はレイちゃんの意見が取り入れられて、startメソッドに名前が変わるかもね。

レイ そうだと、いいなぁ。ところで、WIDTHとHEIGHTの方は?

先生 Pygame Zeroでは、WIDTHはスクリーンの横、HEIGHTは縦の長さを表す変数だ（**図3**）。横や縦の長さは、200とか400といった数字で表現するよね。Scratchのステージも、中心からの距離を180とか240とかで表すだろう。ちなみにPythonでは、この200や400という数字もオブジェクトだ。

図3●WIDTHとHEIGHT
pgzrun.go() により表示されるscreen



レイ 数字もオブジェクトなの?

先生 そう。そして、WIDTHという

変数と数値を結び付けるのが「＝」（イコール）の演算子だ。このような演算子を「割り当て演算子」とか「代入演算子」と呼んでいる。例えば、横幅400、高さ200のスクリーンを表示するコードは次のようになる。2人もIDLEのエディターを起動して、コードを入力してごらん。

screen.py

```
import pgzrun

WIDTH = 400
HEIGHT = 200

pgzrun.go()
```

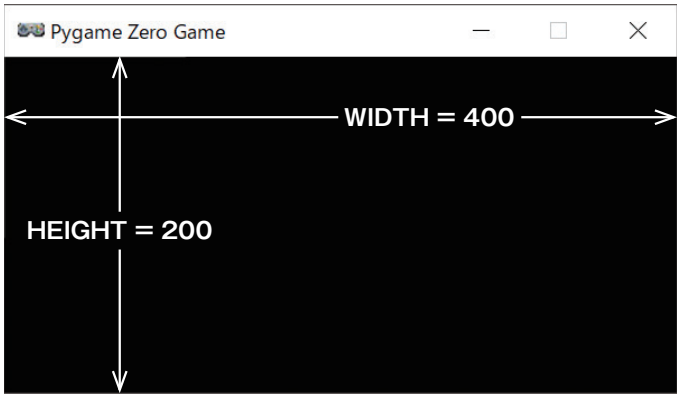
WIDTH変数を400にする

HEIGHT変数を200にする

タツヤ ソースコードのファイル名はどうするの?

先生 拡張子がpyなら何でも構わないけど、今回は「screen.py」とでもしておこう。保存場所はCドライブの直下に作ったPythonフォルダーがいいな。F5キーを押せば実行されるよ（**図4**）。

図4●screen.pyの実行結果



レイ こうやって、スクリーンのサイズを指定するのね。仕組みは難しいけど、コードは単純ね。

タツヤ 数字がオブジェクトってところが、イマイチよくわからない…。

先生 そこはなかなか難しいところだね。オブジェクト指向の仕組みは簡単とは言えないんだけど、仕組みを厳密に理解していなくても、とりあえずプログラムを作ることはできる。特にPygame Zeroは簡単にゲームのプログラムを作れるように工夫されている。だから心配しなくても、大丈夫。だんだんイメージできるようになるよ。

タツヤ ハーイ。

2日目 Part 2

スプライトの表示

先生 Pygame Zeroのスクリーンを表示できたから、次はスプライトを表示しよう。Pygame Zeroでは、スプライト用の画像は「images」フォルダーに保存するルールになっている。また、このimagesフォルダーはソースコードと同じフォルダー内に作る必要がある。

レイ だからPythonフォルダーにmygame.pyを保存して、そこにimagesフォルダーを作ったのね。

先生 あと、これもPygame Zeroのルールなんだけど、imagesフォルダーに保存する画像ファイルの名前に大文字のアルファベットを使ってはならない。大文字を使うとエラーになる。例えば「alien.png」はOKだけど、「Alien.png」だとエラーだ。

タツヤ 質問なんですけど、Pygame Zeroで使える画像と使えない画像って、どうやって見分けるの?

先生 画像ファイルの拡張子を見ればわかるよ。現在のPygame Zeroは、PNG形式、GIF形式、JPEG形式の画像ファイルをサポートしている。PNGとGIFはそれぞれ拡張子がpng、gifになっている。JPEGはjpgが一般的だね。

レイ alien.pngはPNG形式ね。

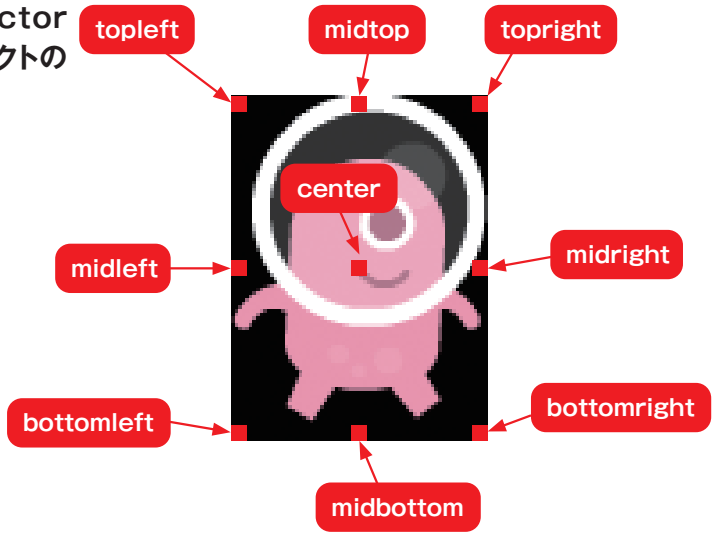
先生 PNG、GIF、JPEGに対応しているけど、PNG形式を使うのがベストだ。PNG形式は画質が劣化しないし、透明色に対応している。画像が用意できたら、Pygame ZeroのActorオブジェクトに画像を読み込ませ、スプライトとして表示する。エイリアンの画像ファイルを読み込むコードは、こんな感じになる。

screen.py

```
import pgzrun

WIDTH = 400
HEIGHT = 200

alien = Actor('alien', center=(200, 100))

pgzrun.go()
```

先生 「alien = Actor('alien', center=(200, 100))」のコードは、Actorオブジェクトにalienの画像ファイルを読み込ませ、x座標200、y座標100の位置にActorオブジェクトの画像を表示するという意味になる。またこのとき、Actorオブジェクトの「center」の位置がx座標200、y座標100になる（**図1**）。ここではcenterを指定しているけど、Actorオブジェクトにはcenter以外にもtopleft、topright、bottomleft、bottomright、midtop、midleft、midright、midbottomがある。これらのいずれかを指定してもいい。

図1●Actorオブジェクトの各位置



レイ 'alien'の部分が、画像ファイルの名前ね。「,」（カンマ）の後ろがActorオブジェクトを「スクリーンのどこに表示させるか」の指定みたい。でも、Actorオブジェクトに読み込むファイル名の「alien」をシングルクオーテーションで括っているのはどうして? 文字列だから?

先生 そう。ファイル名は文字列だ。ダブルクオーテーションかシングルクオーテーションで括らなくてはいけない。

タツヤ 何で?

先生 そうしないと、Pythonは「alien」という変数だと思ってしまうんだ。もちろん、変数alienを作ってファイル名を渡すこともできる。alienという変数名と'alien'という文字列を「=」で関連付けてからActorオブジェクトに渡しても大丈夫だ。タツヤ君、できるかな。

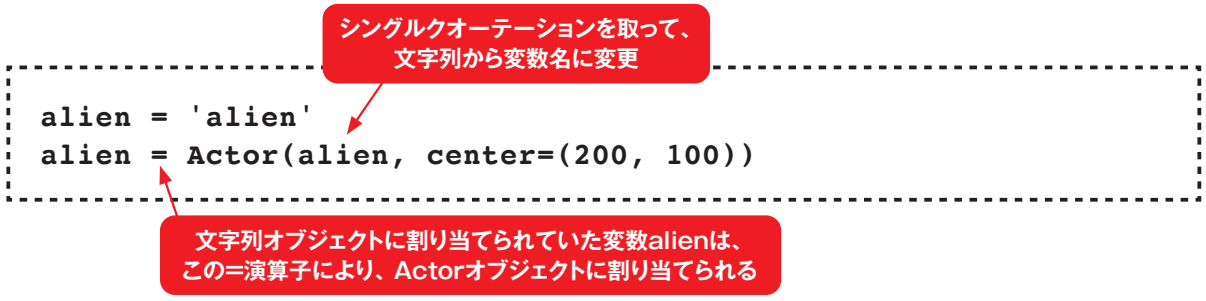
タツヤ いっ…いきなり…。alienという変数と'alien'という文字列を「=」で関連付けて、Actorオブジェクトに渡すんでしょ。これでどうだ。

タツヤ君のscreen.py

```
import pgzrun

WIDTH = 400
HEIGHT = 200

alien = 'alien'
alien = Actor('alien', center=(200, 100))

pgzrun.go()
```

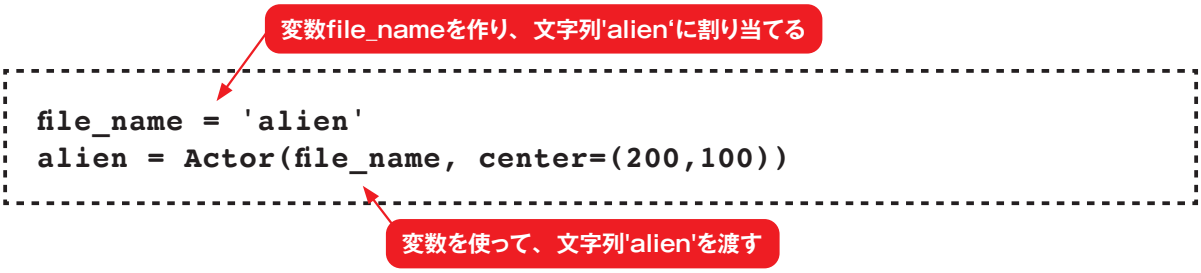
先生 惜しい。Actorオブジェクトに渡すときに変数名をシングルクオーテーションで括ってしまうと、'alien'という文字列になってしまう。

タツヤ う〜ん、ならばこうか。



先生 すばらしい。その通り。Pythonの変数をうまく使えたね。ただ、このコードはとても紛らわしいね。変数alienは最初は文字列オブジェクトに割り当てられているけど、次はActorオブジェクトへ変更されている。そこで、file_nameのような変数を作って、変数file_nameと'alien'を結び付けるといいね。じゃ、今度はレイちゃんがやってみよう。

レイ こう…かしら…。



先生 正解！このように、自分で変数名を決める時は、次のルールがあるから気を付けなければいけないよ。

(1) 1文字目は半角英文字かアンダースコア(_)
(2) 2文字目以降は、半角英数文字かアンダースコア
(3) キーワード(予約語)は使えない

レイ キーワードって?

先生 Pythonには、文法的な意味を持つ単語がある。この単語を「キーワード」と言って、変数名には使えないんだ(**表1**)。

表1●Pythonのキーワード

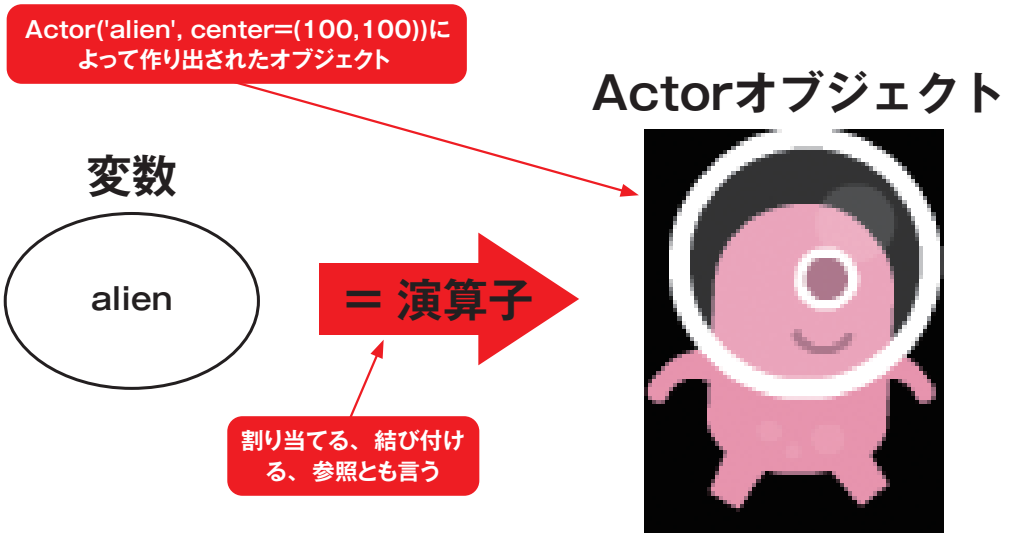
False	class	finally	is
return	None	continue	for
lambda	try	True	def
from	nonlocal	while	and
del	global	not	with
as	elif	if	or
yield	assert	else	import
pass	break	except	in
raise	async	await	

タツヤ あわわ、英単語がいっぱい…。

レイ 先生、この「=」の左側にあるalienは変数でしょ。エイリアンのオブジェクトを「=」で変数alienと結び付けるのはなぜなの?

先生 右側のActor('alien', center=(200, 100))のコードは、実はそれ自身がActorオブジェクトなのではなくて、新しいActorオブジェクトを作り出しているんだ。この作り出した新しいActorオブジェクトには名前がないので、そのままでは使いにくい。そこで、変数alienと結び付けて利用する。このように、変数名でオブジェクトを利用できるようにすることを「変数でオブジェクトを参照する」と言う(**図2**)。

図2●Actorオブジェクトと変数を結び付ける



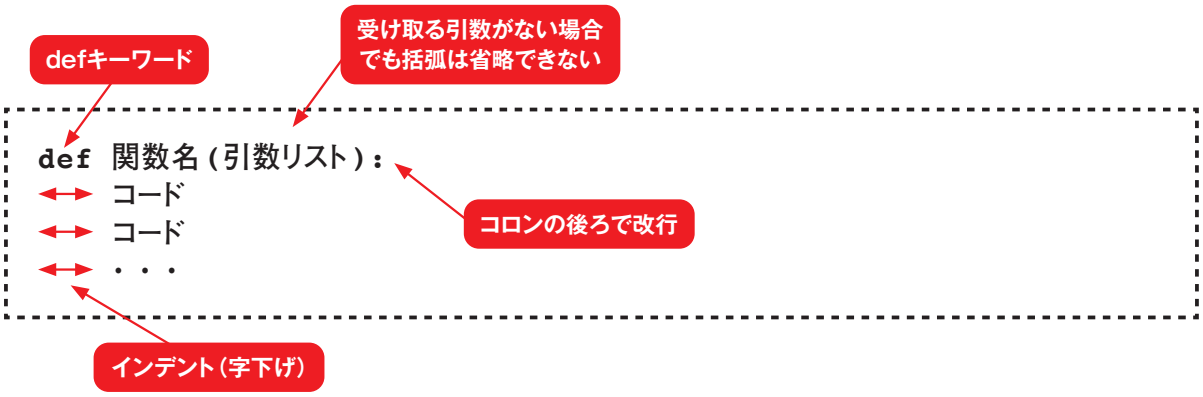
タツヤ いろんな言い方があって混乱するぅ。でも、Pythonの変数が少しだけわかってきた。

先生 これで、エイリアンの画像を表示するActorオブジェクトを変数alienで扱えるようになった。次は、実際に表示するために、「draw(ドロウ)関数」を定義する。

レイ 関数の定義って、Scratchの定義ブロックを作るのと同じかしら。

先生 考え方は同じだ。ただ、Pythonで関数を定義するには、いろいろとルールがある。オリジナルの関数を定義するには、次の構文を覚えなければいけない(**図3**)。

図3●関数の定義



先生 関数を定義するには、defというキーワードを使う。続けて関数名、引数を受け取る変数を宣言するための括弧を書く。この括弧は引数がない場合でも省略できない。そして、最後に半角の「:」(コロン)を書いて改行する。

タツヤ ルールだらけ。

先生 ここからが大切だ。改行したら、半角スペースやTABキーを使って「インデント」(字下げ)を付けてからコードを入力する。このインデントは、関数の定義が終わるまで同じ幅で入力しなければならない。

レイ 厳しいのね。

先生 Scratchだと、ブロック同士の組み合わせは、組み合わせ可能なブロックでしか合わせることができない。でも、テキストで入力するソースコードの場合は、どのようにでも入力できてしまう。そこで、Pythonではルールを厳しくして、できるだけ「型にはまった書き方」になるようにしている。そのおかげで、誰が書いても似たようなコードになるから読みやすくなるんだ。

タツヤ 僕には、まだ読みにくいです…。

先生 それじゃ、関数を定義する練習をしてみよう。次のコードを入力して「function.py」で保存する。コメントは、文法的な意味はないけど、ソースコードに注釈を入れることができる。複数行のコメントは「""」で括る。1行のコメントなら、「#」から始めて改行まで。

じゃあ、実行してみよう。IDLEに名前が表示されるよ。

function.py

```
''' 関数の定義 '''
def all_names():
    print('先生')
    print('レイちゃん')
    print('タツヤ君')

all_names() # 関数呼び出し
```

コメント

コメント

IDLEの出力

```
>>>
========================= RESTART: C:/Python/function.py
=========================
先生
レイちゃん
タツヤ君
>>>
```

レイ Scratchの定義ブロックと同じで、呼び出すとその中のコードが実行されるのね。

先生 その通り。引数も指定できる。ただ、Scratchと違う点は、「戻り値」を指定できるところだ。「return文」を使うと、オブジェクトの呼び出し側に値を返すことができる。ただし、返ってきたオブジェクトは、変数と結び付けておかないと消えてしまうから注意するように。

function.py

```
''' 関数の定義 '''
def all_names(teacher):
    print(teacher)
    print('レイちゃん')
    return 'タツヤ君'

boys_name = all_names('中島先生')
print(boys_name)
```

引数を受け取る変数

all_names関数は、'タツヤ君'を返す

all_names関数に引数を渡す

変数boys_nameには、all_names関数が返す文字列が割り当てられる

IDLEの出力

```
>>>
========================= RESTART: C:/Python/function.py
=========================
中島先生
レイちゃん
タツヤ君
>>>
```

タツヤ 関数呼び出しのコードが、返ってくるオブジェクトを表しているようにも見える…。難しい…。

先生 それでは、screen.pyを開いてdraw関数を追加しよう。

screen.py

```
import pgzrun

WIDTH = 400
HEIGHT = 200

file_name = 'alien'
alien = Actor(file_name, center=(200, 100))

def draw():
    screen.clear()
    alien.draw()

pgzrun.go()
```

追加するdraw関数

タツヤ draw関数の定義にあるscreen.clear()は、スクリーンのオブジェクト（screen）にあるclearメソッドの呼び出しだな。

レイ alien.draw()は、alienオブジェクト、つまりActorオブジェクトが持つdrawメソッドの呼び出しね。

先生 screen.clear()は、スクリーンを黒く塗りつぶす。その後、alien.draw()でスプライトを描くんだ。実行してみよう（**図4**）。

図4●screen.pyの実行結果

タツヤ 出た、出た。少しコードが読めるようになってきたぞ。

レイ そうね。いろいろなスプライトを表示してみたいわ。

先生 次は、このスプライトを、動かすよ〜。

2日目 Part 3 スプライトの移動

先生 ところで、Scratchでスプライトを動かすには、どんなブロックを使っていたか覚えてる?

タツヤ 「動かす」ブロックや「ずっと」ブロックで、少しずつスプライトの座標を変更させてたかなぁ。

先生 そうだよね。Pygame Zeroでもスプライトの移動は座標の変化で行う。ただ、繰り返し処理はPygame Zeroの環境に組み込まれているのでコードを書く必要はないんだ。

レイ 繰り返し処理を書かずに、どうやって座標を変化させ続けるの?

先生 Pygame Zeroは、通常は1秒間に60回「update関数」を呼び出すようになっている。そこで、update関数を定義して、その中でスプライトの座標を変化させるんだ。例えば、update関数にスプライトのx座標を+1する処理を書いておけば、1秒間にx座標は+60されるよね。つまり、右に移動する。左に動かしたければ、x座標を−1する処理を書けばいい。3倍のスピードで右に動かしたければx座標を+3する処理を書く、という具合

だ。

タツヤ Pythonで、座標の計算はどうやるの?

先生 数値の計算では算術演算子を使うから、説明しておこう。Pythonでは、**表1**のような算術演算子が用意されている。

表1●算術演算子

演算子	意味	例
+	加算	3 + 4
-	減算	6 - 2
*	乗算	2 * 4
/	除算	8 / 2
//	除算	8 // 2
%	剰余	5 % 2
**	べき乗	2 ** 3

先生 calc.pyに、次のコードを入力して、算術演算子の使い方をマスターしよう。

calc.py

```
x = 2
y = 3
z = x + y

print('変数zの値')
print(z)
print('+演算子は、' + '文字列の連結' + 'もできる')
print('x × y = ' + str(x * y))
print('*演算子は文字列の繰り返しもできる ' + 'ABC' * y)
print('通常の除算 x / y → ' + str(y / x ))
print('小数点以下切り捨て除算 x // y = ' + str(y // x))
print('べき乗の計算 x ** y → ' + str(x ** y) )
```

IDLEの出力

```
>>>
============================ RESTART: C:/Python/calc.py
============================
変数zの値
5
+演算子は、文字列の連結もできる
x × y = 6
*演算子は文字列の繰り返しもできる ABCABCABC
通常の除算 x / y → 1.5
小数点以下切り捨て除算 x // y = 1
べき乗の計算 x ** y → 8
>>>
```

タツヤ べき乗の計算とかできるのは、便利だねぇ。

先生 計算だけじゃなくて、+演算子を使うと、文字列同士の連結もできるよ。

レイ 計算結果を「str関数」に渡している部分があるわね。このstr関数は、何をする関数なの?

先生 str関数は、数値のオブジェクトを文字列オブジェクトに変換する機能を提供している。これは頻繁に使う関数だ。

タツヤ 数値のオブジェクトに+演算子を使うと数値の加算、文字列に使うと文字列の連結。+演算子は2つの使い方ができるのか…。

先生 *演算子も文字列に対応している。

レイ ホントだわ。掛けた数だけ文字列が繰り返されている。

先生 算術演算子で計算をした後に、結果と変数と結び付けるには =演算子を使う。例えば、変数xを1増やすコードは「x = x + 1」となる。

レイ「x = x + 1」？何だかおかしくない？

先生 数学だと明らかにおかしいけれど、この「=」は数学のイコールではないのだよ。「x = x + 1」は「xとx + 1は等しい」ではなく、「x + 1をxに結び付ける」という意味になる。もう少し詳しく説明すると、これは、元の変数xと結び付いている数値のオブジェクトを+1した新たなオブジェクトを作り、再び変数xに結び付けているんだ。ところで、Pythonには「複合代入演算子」という便利な演算子があるので、こちらを使った方がコードが短くなって読みやすい（**表2**）。

表2●複合代入演算子

演算子	説明	同じ式
+=	変数に右側の数値を加えた値を割り当てる	a += b は、a = a + b と同じ
-=	変数から右側の数値を引いた値を割り当てる	a -= b は、a = a - b と同じ
*=	変数に右側の数値を掛けた値を割り当てる	a *= b は、a = a * b と同じ
/=	変数を右側の数値で割った値を割り当てる	a /= b は、a = a / b と同じ
//=	変数を右側の数値で割り、小数点以下を切り捨てた値を割り当てる	a //= b は、a = a // b と同じ
**=	変数を右側の値でべき乗して割り当てる	a **= b は、a = a ** b と同じ
%=	変数を右側の値で割った余りを割り当てる	a %= b は、a = a % b と同じ

タツヤ Pythonには、便利な演算子がたくさんあるなぁ。

先生 それじゃ、エイリアンを左端から右端へ移動するようにupdate関数を定義しよう。まず最初に、エイリアンをスクリーンの左端に配置する。最初の位置は、Actorオブジェクトの生成時に決めることができるので、midrightを(0, 100)に設定しよう。

レイ x座標が0、y座標が100ってことね？でも、どうして括弧で括っているの？

先生 これは「タプル」と呼ばれるオブジェクトの書き方で、x座標とy座標の2つの数値を1つのオブジェクトに格納できる。タプルについては、次回詳しく説明するよ。最初の位置が決まったら、x座標を増やしていく。このとき、Actorオブジェクトの「x」アトリビュートを使っ

てもいいし、「left」アトリビュートを使ってもいい。このコードをupdate関数の中で実行すると、1秒間に60回update関数が呼び出されるから、エイリアンは右に移動していく。そして、update関数が定義されていると、draw関数も自動的に繰り返し呼ばれる仕組みだ。

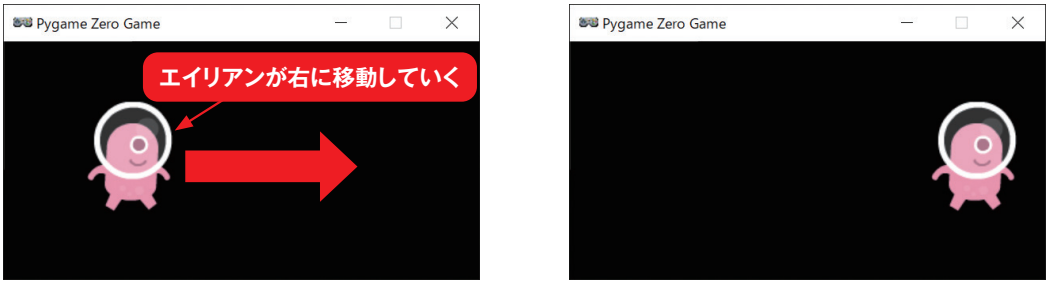
screen.py

```
import pgzrun

WIDTH = 400
HEIGHT = 200

file_name = 'alien'
alien = Actor(file_name, midright=(0,100))

def draw():
    screen.clear()
    alien.draw()

def update():
    alien.x += 1
    # alien.left += 1

pgzrun.go()
```

追加するupdate関数

タツヤ 動いた…けど、右に消えて行ってしまう～（**図1**）。そうか、座標がスクリーンの右端を越えてしまうんだ。「壁に触れたら跳ね返る」オブジェクトはないのだろうか?

図1●screen.pyの実行結果



先生 「もし～」といったコードを書きたいときは、Pythonでは「if文」などの制御文を使えばできる。制御文については、次回にしよう。今日はPythonの文法がたくさん出てきたね。しっかり、復習して忘れないようにしておこう。次回は、迷路プログラムに挑戦するよ。

レイ ハーイ。

3日目
迷路を作ろう

3日目

Part 1 繰り返しによる背景の表示

先生 それでは、いよいよゲームっぽいプログラムを作ってみよう。今回のお題は「迷路」だ。ここでは、こんな感じで考えてみた（**図1**）。作るのはあくまでもゲームの骨格部分だから、完成したらアレンジして面白く改造するといいね。

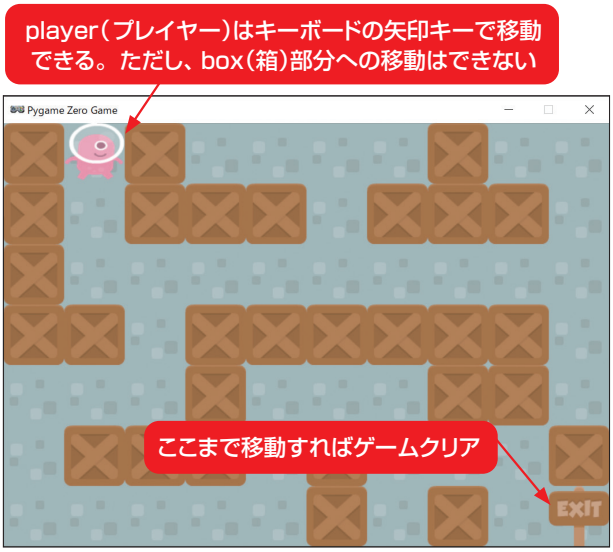
タツヤ おおぉ、本格的。Scratchより、画面広いし…。

レイ 難しそう。私たちでも、作れるの?

先生 大丈夫。迷路のプログラムにはゲームプログラミングの基本が詰まっているよ。少しずつ説明しよう。

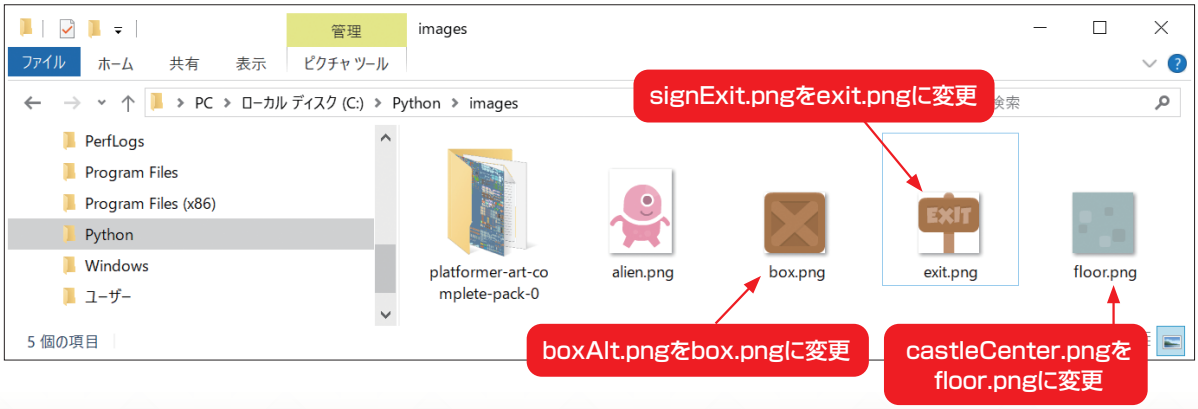
タツヤ 了解!

図1●今回作成する迷路の画面



先生 まずは、「player」（プレイヤー）と「背景」のスプライトをスクリーンに表示する方法だ。playerのスプライトは、エイリアンの画像を流用しよう。背景はダウンロードした「platformer-art-complete-pack-0」フォルダーにある「Base pack」→「Tiles」フォルダー内の「boxAlt.png」と「castleCenter.png」、「signExit.png」を並べて使う。すべて、imagesフォルダーにコピーして、ファイル名を「boxAlt.png」→「box.png」、「castleCenter.png」→「floor.png」、「signExit.png」→「exit.png」に変更しておく（**図2**）。

図2●迷路で利用する画像ファイルを用意



タツヤ そうか、大文字を含むファイル名はダメだったっけ。

レイ このTilesフォルダーって「タイル状の画像」ばかりを集めたフォルダーなのね。

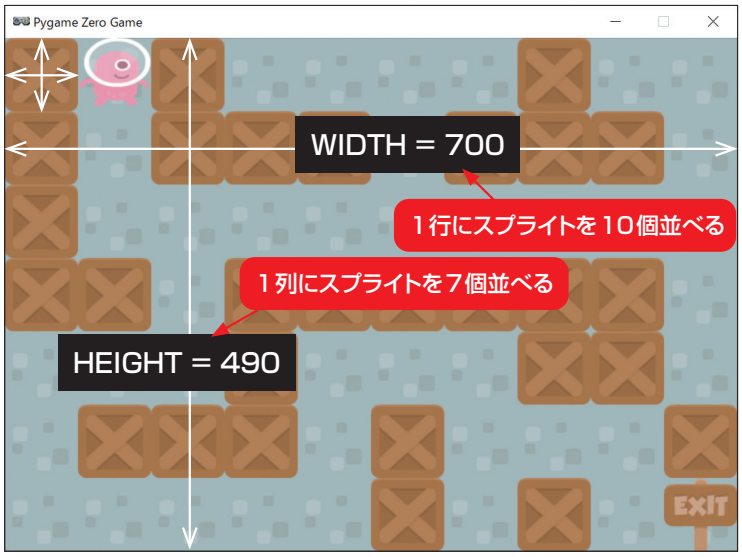
先生 このフォルダー内の画像は、すべて70×70というサイズになっている。サイズの単位は「ピクセル」だ。「1ピクセルが画面の1つの点」だね。

タツヤ Scratchの時に教えてもらったよ。この画像をスクリーンに並べて画面を作るんでしょ。

先生 そうだ。スプライトを横に10個、縦に7個並べたいから、スクリーンのサイズをWIDTH=700、HIGHT=490にする（**図3**）。

レイ でも…エイリアンさんは70×70じゃないわよ。

図3●スプライトとスクリーンのサイズ



先生 そうなんだ。画像のサイズを統一したいところだね。そこで、alien.pngのサイズを70×70ピクセルに変更しよう。

タツヤ ペイントツールを使うのかな?

先生 Windows 10のペイントは透明色を扱えないので、サイズを変更してしまうと背景が抜けなくなってしまう。

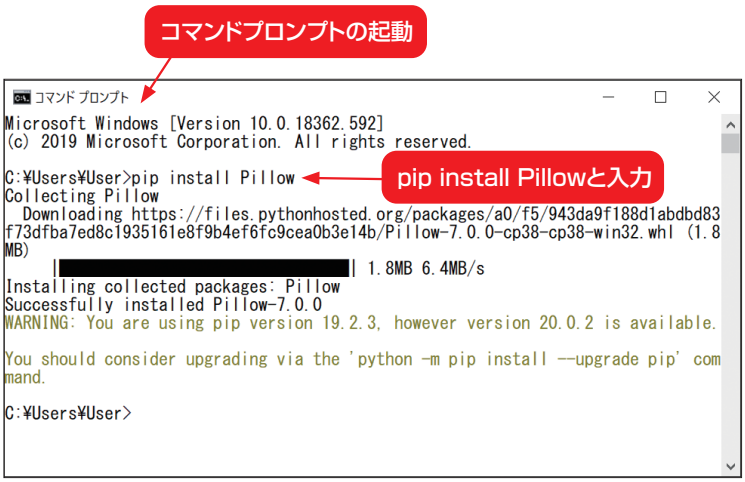
レイ エイリアンさんの背景って、透明だったのね。

先生 まぁ、フリーのお絵描きソフトをインストールしてもできるけど、ここは「プログラミング教室」だ。画像をリサイズするプログラムくらい自分で作れなきゃね。

タツヤ マジか…（ガクガク）。

先生 Pythonは、便利なライブラリが豊富だ。今回は、「Pillow」（ピロウ）というライブラリを使おう。Pillowは、Pythonで定番の画像処理ライブラリだ。Pillowのインストールは、Pygame Zeroのインストールと同じ「pip」を使う。コマンドプロンプトを起動して「pip install Pillow」と入力しよう（**図4**）。

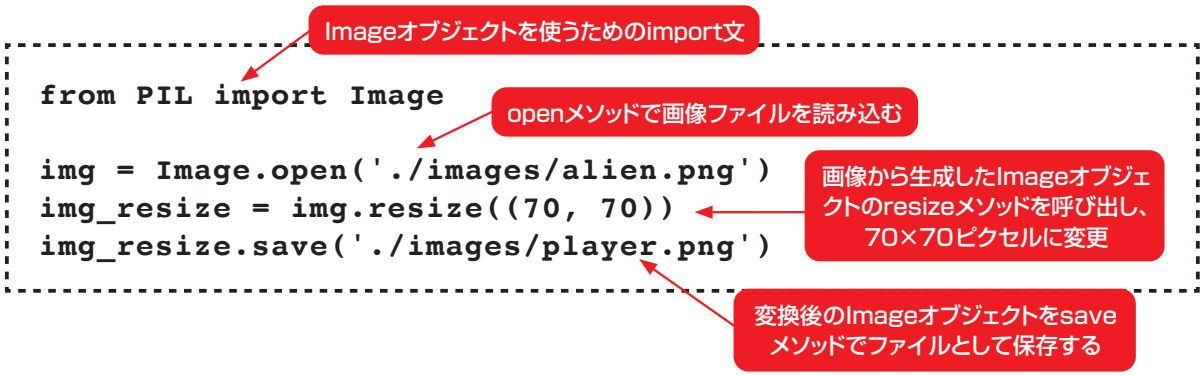
図4●Pillowのインストール



タツヤ ライブラリのインストールは簡単。

先生 次は、変換するコードを入力する。IDLEを起動して「File」メニューの「New File」を選択。エディターが起動したら次のコードを入力する。入力が完了したら、img_resize.pyで保存しよう。

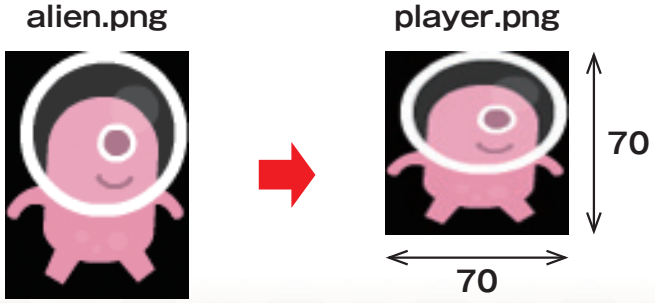
img_resize.py




```
from PIL import Image

img = Image.open('./images/alien.png')
img_resize = img.resize((70, 70))
img_resize.save('./images/player.png')
```

レイ imagesフォルダーに、小さくなったエイリアンさんの画像ができた（**図5**）。

図5●画像サイズを変更した



タツヤ ファイル名の「./images」ってどういう意味?

先生 文字列でファイルの場所を指定するときは、「絶対パス」と「相対パス」という2種類の書き方がある。「絶対パスは、Cドライブの直下からファイルのある場所まで、すべて指定する書き方だ。例えば、エイリアンの画像なら「C:¥Python¥images¥alien.png」のようになる。注意する点は、IDLEのエディターでは半角の「¥」マークを「/」(スラッシュ)で入力すること。対して、相対パスはカレントフォルダーから見た相対的な位置を表現する。カレントフォルダーはソースファイルがある「C:¥Python」で、このフォルダーを「.¥」、IDLEのエディターでは「./」で表す。さらに、そのフォルダー内の「images」フォルダーなので、「./images/alien.png」になるんだ。

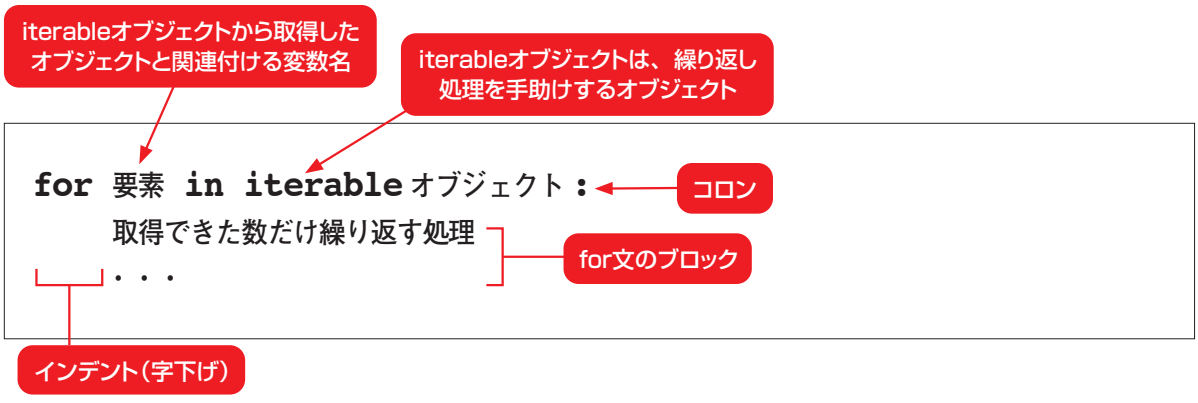
レイ Pythonは、覚えることがいっぱい。

先生 それじゃ、画像ファイルは用意できた。この画像をスクリーンに配置しよう。

タツヤ このスプライトを、縦と横で合計70回描くのか…。Scratchの「ループブロック」みたいに繰り返しはできないの?

先生 鋭い。Pythonには、for文という繰り返し構文がある(**図6**)。これを使おう。

図6●for文

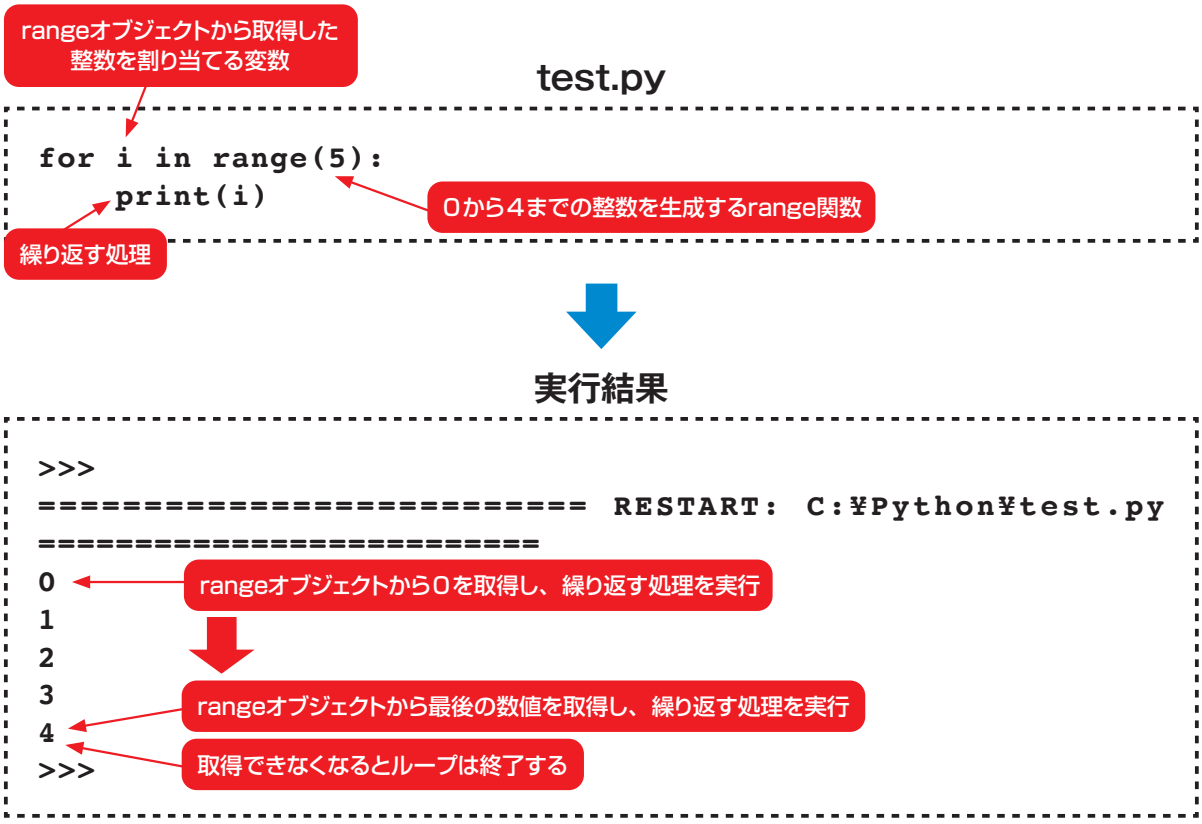


タツヤ for文も、関数と同じようにインデントで範囲を決めている…。

先生 このように、同じ幅のインデントで作られた範囲を「ブロック」と呼ぶ。

タツヤ この「iterable」とか「要素」ってのは?

先生 iterableオブジェクトは、繰り返し処理を手助けする機能を持つオブジェクトだ。Pythonではiterableオブジェクトの生成に、「range関数」をよく利用する。range関数は引数に整数を与えて呼び出すと、0から順番に「引数 − 1」までの整数を内部に持つ「rangeオブジェクト」を生成する。このrangeオブジェクトはiterableオブジェクトでもあるので、for文で使うと生成した整数を取得しつつ、繰り返し処理を行うことができる。test.pyで確認してみよう。



タツヤ Pythonで繰り返し処理を作るには、for文を使うのか…。

先生 さらに、縦と横の繰り返しにしたいなら、for文を「ネスト」すればいい。

レイ ネストは知ってるわ。繰り返しの中に繰り返しを埋め込むやつね。

先生 そう。したがって、7行と10列の座標計算をしながらfloor.pngをスクリーンに敷き

詰めるには、maze.pyのようなコードになるね。ついでに、playerのスプライトも表示しておこう。

maze.py

```
import pgzrun

WIDTH = 700
HEIGHT = 490

floor = Actor('floor', topleft=(0, 0))
player = Actor('player', topleft=(70, 0))

def draw():
    screen.clear()
    for y in range(7):
        for x in range(10):
            # floorの描画
            floor.topleft=(70*x, 70*y)
            floor.draw()
    player.draw()

pgzrun.go()
```

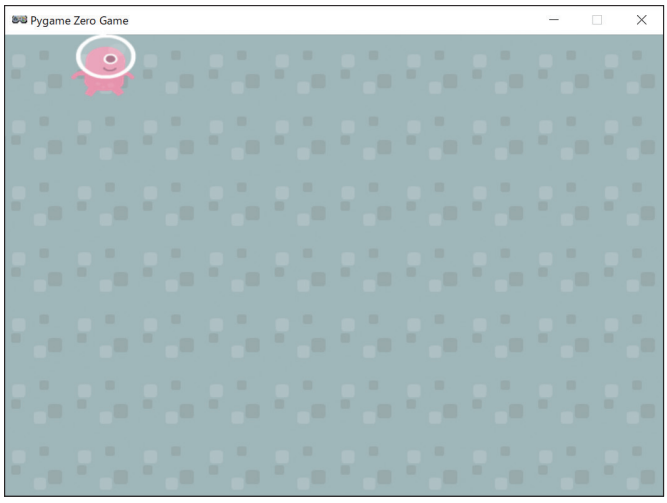
縦のループ(7回)

横のループ(10回)

floorの表示

レイ 繰り返しの考え方はScratchと同じだわ。実行すると、綺麗にタイルが敷き詰められてエイリアンさんもいるし、もうすぐ完成しそうじゃない(**図7**)。

図7●背景とplayerの表示



タツヤ レイちゃん、気が早いよ。迷路にするためには、box.pngの配置を決めなきゃ。それに、キーボードでplayerを動かせないと…。

先生 そうだね。休憩して、今度はboxの配置を考えよう。

3日目 Part 2 条件による表示

先生 それでは、床の上にboxを配置して迷路にする。

レイ boxの配置って、面倒よね。座標がバラバラで単純な計算では求まらないし…。

先生 そんなときは、MAP情報を作るんだ。

タツヤ MAPって、地図のこと?

先生 そう。スクリーンを10×7のセルに分割してスプライトの配置を決める表を作るんだ（図1）。

図1●スクリーンのMAP

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9

レイ この表が、MAPなの?

先生 表のようなデータ構造を「2次元配列」と呼ぶ。つまり、この表でスクリーンのどこにboxを置くのかを管理する。

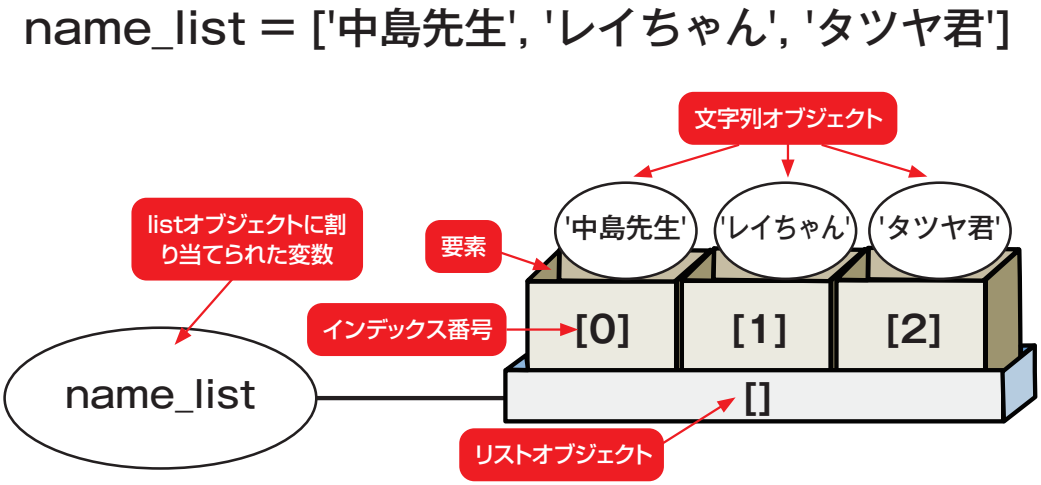
タツヤ Pythonでどうやって作るの?

先生 Pythonで2次元配列を作るには、「listオブジェクトのlist」を使うのがいいだろう。

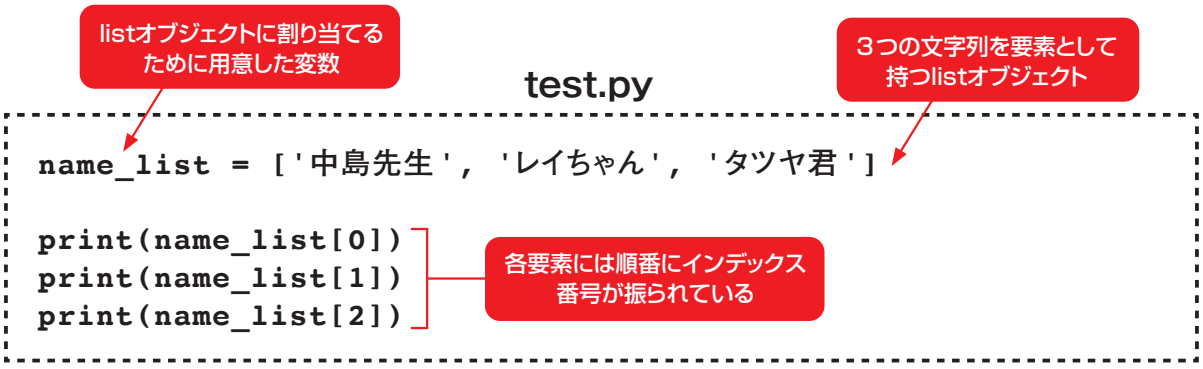
タツヤ またしても、新しいオブジェクトが出現…。listとは?

先生 listオブジェクトは、[](角括弧)で表すオブジェクトで、list内のオブジェクトは「要素」と呼ばれている。listの生成と同時に要素も決めるときは、[ ]の中にカンマ区切りで指定する。この要素には、0から順番に「インデックス」と言う番号が振られている。例えば、「name_list = ['中島先生', 'レイちゃん', 'タツヤ君']」というコードのlistオブジェクトのイメージは、**図2**のようになる。

図2●リストオブジェクトのイメージ



先生 要素を利用したいときは、割り当てた変数名に[]を付けて要素のインデックスを指定する。test.pyで確認しよう。



実行結果

```
>>>
============================ RESTART: C:¥Python¥test.py
============================
中島先生
レイちゃん
タツヤ君
>>>
```

各要素を表示

先生 listは生成後に、「appendメソッド」で要素を追加したり、「popメソッド」で削除したりすることもできる。また、listはiterableオブジェクトでもあるので、for文で順番に要素を変数に割り当てて、繰り返し処理を行うことが可能だ。

test.py

```
name_list = []

name_list.append('中島先生')
name_list.append('レイちゃん')
name_list.append('タツヤ君')

for name in name_list:
    print(name)

name_list.pop(0)

print() # 改行のみ
for name in name_list:
    print(name)
```

空のlist

appendメソッドはlistの末尾に要素を追加する

listもiterableオブジェクトなのでfor文で繰り返しができる

popメソッドは指定したインデックス番号の要素を削除する

実行結果

```
>>>
========================== RESTART: C:¥Python¥test.py
==========================
中島先生
レイちゃん
タツヤ君

レイちゃん
タツヤ君
>>>
```

for文で表示した要素

for文で表示した要素

タツヤ listを使ってどうやってMAPを作るんだ?

先生 ヒントは「listの要素をlistにする」ってところだね。

タツヤ ???

レイ …ひょっとして、縦の番号を表すlistと、横の番号を表すlistを組み合わせるの?

先生 冴えてるね。その通り。もしboxの位置を数字の1で表すなら、次のようなMAPデータをlistで作ることができる。

map_data

```
map_data =[[1, 0, 1, 0, 0, 0, 0, 1, 0, 0],
           [1, 0, 1, 1, 1, 0, 1, 1, 1, 0],
           [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
           [1, 1, 0, 1, 1, 1, 1, 1, 1, 0],
           [0, 0, 0, 1, 0, 0, 0, 1, 1, 0],
           [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
           [0, 0, 0, 0, 0, 1, 0, 1, 0, 0]]
```

タツヤ なるほど。1のときは、そこにboxを配置するってことだね。これはわかりやすい。

先生 そうだ。例えば、map_dataの2行目の3番目の要素を知りたい時は、「map_data[1][2]」と記述する。インデックスは0から始まるから注意するように。ところで、レイちゃん。map_data[3][5]はいくつだかわかるかな。

レイ 外側のlistと中のlistに対して、それぞれインデックスを指定するってことだから、4行目の6番目の値よね。「1」です。

先生 その通り。実は、listとほとんど同じ構造の「tuple」（タプル）というオブジェクトもある。タプルの特徴は、丸括弧で括ることと、要素の変更ができないことだ。したがって、appendメソッドやpopメソッドなどはtupleにはない。

test.py

タプルは()で括る

```
name_tuple = ('中島先生', 'レイちゃん','タツヤ君')

for name in name_tuple:
    print(name)
```

tupleもiterableオブジェクトなので for文で繰り返しができる

実行結果

```
>>>
============================ RESTART: C:¥Python¥test.py
============================
中島先生
レイちゃん
タツヤ君
>>>
```

for文で表示した要素

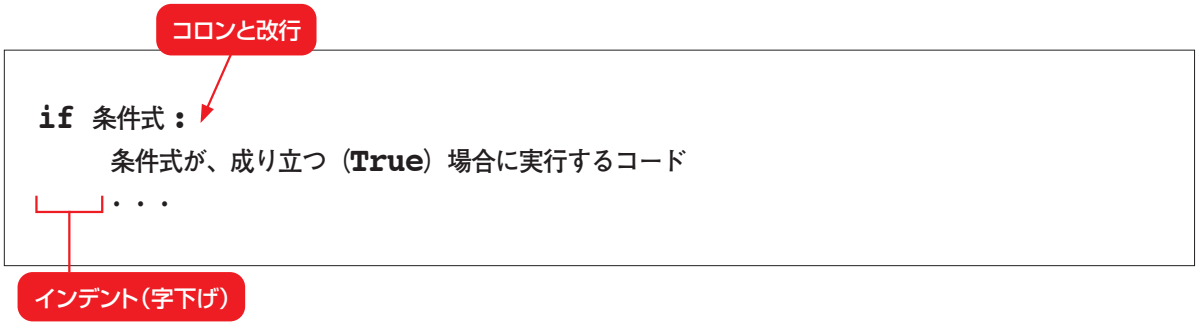
タツヤ スプライトの座標を設定するときの書き方は、このtupleだったのか。

先生 変更したくない値は、tupleで扱う方が、間違えて変更される恐れがなくなる。

レイ ところで、このMAP情報を使ってboxを表示するコードは、「1のときは表示」ってなるのよね。

先生 その通り。つまり、条件分岐だね。Pythonで条件分岐を書くには「if文」を使う。一番単純なif文は、次の構文で書ける（**図3**）。

図3●if文



レイ if文って比較的簡単よね。

先生 if文の条件式の結果は、論理値である「True」か「False」になる。Trueの場合、if文のブロックが実行される。動作自体は、Scratchの「もしも～」ブロックと同じだ。そして、if文には、条件式が成り立たなかった場合に実行するコードを書ける「else文」（**図4**）や、順番に条件式を判断して成り立つ場合のブロックのみ実行する「elif文」（**図5**）といったバリエーションがある。これらを利用して、できるだけ効率の良い条件分岐にできるといいね。

図4●else文

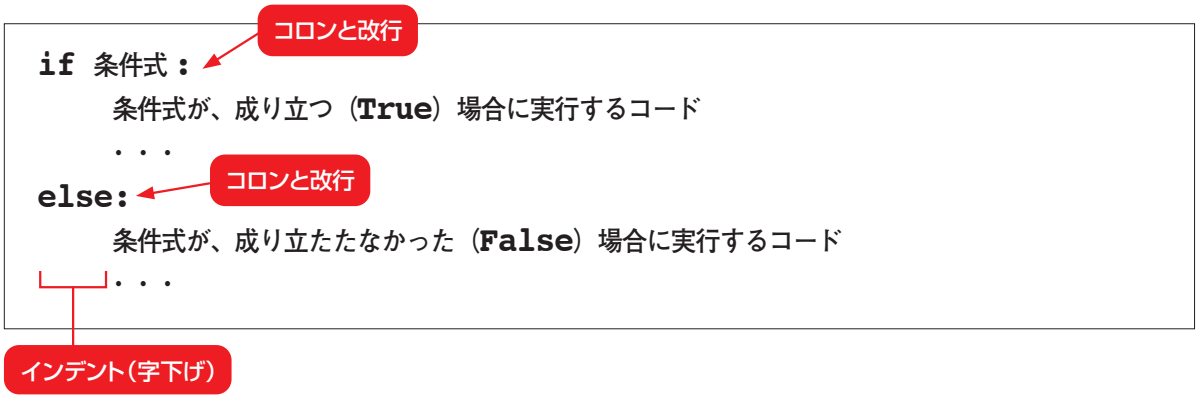


図5●elif文

```
if 条件式1:
    条件式1が、成り立つ（True）場合に実行するコード
    ・・・
elif 条件式2:
    条件式1が「False」で、条件式2が「True」の場合に実行するコード
    ・・・
elif 条件式3:
    条件式1、条件式2が「False」で、条件式3が「True」の場合に実行するコード
    ・・・
else:
    すべて「False」だった場合に、実行するコード
    ・・・
```

インデント（字下げ）

レイ 条件式は、どうやって書くの?

先生 条件式には、比較演算子（**表1**）や論理演算子（**表2**）が使える。これらの演算子は、条件が成り立てばTrueを返し、成り立たなければFalseを返す。こちらも、Scratchの条件式と同じと考えて構わない。

表1●比較演算子

演算子	意味
==	a == b において、b とa が等しいとき True
!=	a != b において、b とa が等しくないとき True
>	a > b において、b よりa が大きいとき True
<	a < b において、b よりa が小さいとき True
>=	a >= b において、b とa が等しいか、b よりa が大きいとき True
<=	a <= b において、b とa が等しいか、b よりa が小さいとき True

表2●論理演算子

演算子	意味
and	左から評価し、False と同等であればその時点でその項の値を返し、すべて True と同等であれば最後の項の値を返す
or	左から評価し、True と同等になった項があればその時点でその項の値を返し、すべて False と同等なら最後の項の値を返す
not	not a において、a が True なら False、False なら True

タツヤ 先生、条件分岐はできそうだよ。実際にMAPと同じ位置にboxを配置するコードを教えてよ。

先生 そうだね。2重のfor文を用意して、listのlistであるmap_dataを処理しよう。そして、map_dataの値が1だったらboxのスプライトを描画する。

タツヤ う〜ん、よくわからん。

先生 それじゃ、次のコードを実行してみよう。

maze.py

```
import pgzrun

WIDTH = 700
HEIGHT = 490

# MAP情報（1の場所がbox）
map_data =[[1, 0, 1, 0, 0, 0, 0, 1, 0, 0],
           [1, 0, 1, 1, 1, 0, 1, 1, 1, 0],
           [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
           [1, 1, 0, 1, 1, 1, 1, 1, 1, 0],
           [0, 0, 0, 1, 0, 0, 0, 1, 1, 0],
           [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
           [0, 0, 0, 0, 0, 1, 0, 1, 0, 0]]

player = Actor('player', topleft=(70, 0))
floor = Actor('floor', topleft=(0, 0))
box = Actor('box', topleft=(0, 0))

def draw():
    screen.clear()
    for y in range(7):
        for x in range(10):
            # floorの描画
            floor.topleft=(70*x, 70*y)
            floor.draw()
            # boxの描画
            if map_data[y][x] == 1:
                box.topleft=(70*x, 70*y)
                box.draw()
    player.draw()

pgzrun.go()
```

追加

追加

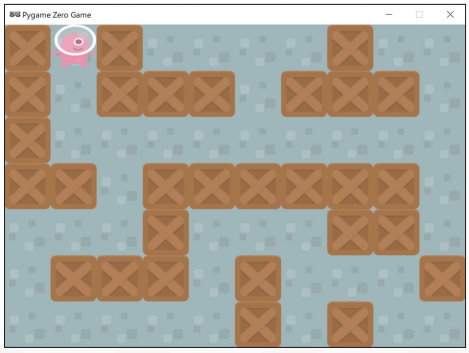
追加

レイ きゃあ! boxで迷路ができてる（図6）。Pythonて、少ないコードで複雑なことができるのね。

タツヤ マジ、スゲェ。

先生 それじゃ。少し休憩しましょう。

図6●maze.pyの実行結果



3日目 Part 3 キーボードでスプライトを動かす

先生 さぁ、いよいよキーボードを使う。キーボードでスプライトを動かせると、かなりゲームっぽくなるよ。

レイ Scratchのときは「～キーが押されたとき」って言うブロックがあったけど、Pythonにも同じようなものがあるの?

先生 Pythonにも、標準でコマンドプロンプトからキーボード入力を行う関数が用意されているんだけど、ゲームに利用するにはとても使いにくい。そこでPygame Zeroは、Scratchと同じようにゲームでキーボードを使いやすくする仕組みを用意している。

タツヤ それは?

先生 「イベントドリブン」だ。

タツヤ イベントどんぶり…。略して「イベ丼」?

先生 「どんぶり」じゃなくて「ドリブン」だよ。ゲームでは、マウスのクリックやキーボードからの入力のタイミングで処理を行うことが多い。このとき、クリックやキー入力のようなユーザーアクションを「イベント」と呼ぶ。そして、そのイベントで処理を駆動(ドリブン)することを「イベントドリブン」と呼ぶんだ。

レイ ひょっとして、Scratchもイベントドリブンだったの?

先生 現在、マウスやタッチ、キーボードで操作するアプリは、基本「イベントドリブン」でできていると思って構わない。

タツヤ アプリの基本的なお作法みたいな?

先生 タツヤ君、うまいこと言うねぇ。Pygame Zeroの場合もマウスやキー入力があると

イベントが発生して、対応する関数が自動的に呼び出されるから、プログラマはその関数を定義するだけでいい。このイベントによって自動的に呼び出される関数のことを「イベントハンドラ」とか「イベント関数」と呼ぶ。

レイ どんなイベントハンドラがあるの?

先生 キーボードの場合、「on_key_down関数」が一番使える。この関数は、何かキーが押されると1回だけ発生する「key_downイベント」で呼び出される。

タツヤ マウスのクリック、キーごとのイベントは発生しないの?

先生 もちろん、すべてイベントが発生している。詳しくは、Pygame Zeroのドキュメントにあるから、一度見てみるといいね(**図1**)。

図1●Pygame Zeroのドキュメント
https://pygame-zero.readthedocs.io/ja/latest/hooks.html#id3

タツヤ うひゃ〜。イベントだらけ。マウスのボタンやキーごとに名前が付いているみたいだ。

先生 そうなんだ。on_key_down関数は、どのキーが押されても呼び出される。そこで、どのキーが押されたのか、if文で調べる必要がある。とりあえず、今回利用するキーは「上下左右の矢印キー」だけだから、この4つのキーが使えるかどうか「key_test.py」で確認してみよう。

key_test.py

```
import pgzrun

def on_key_down(key):
    print('何かキーが押されました。')
    if key == keys.UP:
        print('上矢印キーが押されました。')

    if key == keys.DOWN:
        print('下矢印キーが押されました。')

    if key == keys.LEFT:
        print('左矢印キーが押されました。')

    if key == keys.RIGHT:
        print('右矢印キーが押されました。')

    print() # 改行のみ

pgzrun.go()
```

押されたキー

キーはkeysオブジェクトのアトリビュートとして定義されている

先生 このkey_test.pyは、キーボードが押されたときに自動的に呼び出されるイベントハンドラ「on_key_down関数」を定義しただけのシンプルなコードだ。on_key_down関数では、引数keyで押されたキーの値を受け取る。この値は、keysオブジェクトのアトリビュートとして定義されているので、渡された「キーの値」と「keysオブジェクトのアトリビュート」を比較して、同じであればそのキーが押されたと判断できる。

レイ 上矢印キーが、keysオブジェクトのUPというアトリビュートだっていうことが、どうやってわかったの?

先生 それは、さっき紹介したドキュメントを見たからね。

タツヤ なんだぁ。

先生 それじゃ、実行してみよう(**図2**)。

図2●key_test.pyの実行結果

IDLEの画面

```
>>>
========================= RESTART: C:/Python/key_test.py
=========================
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/co
ntribute.html
何かキーが押されました。

何かキーが押されました。

何かキーが押されました。
上矢印キーが押されました。

何かキーが押されました。
右矢印キーが押されました。

何かキーが押されました。
下矢印キーが押されました。

何かキーが押されました。
左矢印キーが押されました。

>>>
```

キーの種類に関係なく表示

押したキーを判別して一致したら表示

レイ これでキー入力の方法もわかったし、いよいよ迷路を作れるわね。

先生 それじゃ、迷路の最終コードを紹介しよう。

maze.py

```
import pgzrun

WIDTH = 700
HEIGHT = 490

# MAP
map_data =[[1, 0, 1, 0, 0, 0, 0, 1, 0, 0],
           [1, 0, 1, 1, 1, 0, 1, 1, 1, 0],
           [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
           [1, 1, 0, 1, 1, 1, 1, 1, 1, 0],
```

maze.pyの続き

```
            [0, 0, 0, 1, 0, 0, 0, 1, 1, 0],
            [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
            [0, 0, 0, 0, 0, 1, 0, 1, 0, 0]]

#playerの現在位置
location = [0, 1]

# プレイヤー
player = Actor('player', topleft=(70, 0))

# 床のタイル
floor = Actor('floor', topleft=(0, 0))

# 箱
box = Actor('box', topleft=(0, 0))

# 出口の看板
exit = Actor('exit', topleft=(630, 420))

def draw():
    screen.clear()
    for y in range(7):
        for x in range(10):
            floor.topleft=(70*x, 70*y)
            floor.draw()
            if map_data[y][x] != 0:
                box.topleft=(70*x, 70*y)
                box.draw()
    exit.draw()
    player.draw()

def on_key_down(key):
    if key == keys.UP:
        # プレイヤーが上端でなければ
        if location[0] >= 1:
            # プレイヤーの進む方向がboxでなければ進む
            if map_data[location[0]-1][location[1]] != 1:
                location[0] -= 1
                player.y -= 70

    if key == keys.DOWN:
        # プレイヤーが下端でなければ
        if location[0] <= 5:
            # プレイヤーの進む方向がboxでなければ進む
            if map_data[location[0]+1][location[1]] != 1:
```

maze.pyの続き

```
                location[0] += 1
                player.y += 70

    if key == keys.LEFT:
        # プレイヤーが左端でなければ
        if location[1] >= 1:
            # プレイヤーの進む方向がboxでなければ進む
            if map_data[location[0]][location[1]-1] != 1:
                location[1] -= 1
                player.x -= 70

    if key == keys.RIGHT:
        # プレイヤーが右端でなければ
        if location[1] <= 8:
            # プレイヤーの進む方向がboxでなければ進む
            if map_data[location[0]][location[1]+1] != 1:
                location[1] += 1
                player.x += 70

pgzrun.go()
```

タツヤ ちゃんとplayerが矢印キーで移動する。しかも、boxの上には移動しない（**図3**）。すごい!

図3●maze.pyの実行結果

レイ 出口のスプライトも表示するようにしたのね。

先生 boxの上に移動しないようにするために「location」（ロケーション）というlistを作ってある。このlistは、playerがMAPのどこにいるのかを記録しておくためのものだ。これにより、進む方向のMAPが1のときは移動できなくしている。

タツヤ おもしろくなってきたぞ。僕は、敵が出現するように改造しよう。

レイ 私は、アイテムをGETできるようにしたいな。

先生 いろいろ楽しいアイディアが出てきたね。今日は、ここまで。次回は月面着陸ゲームに挑戦だ。

4日目
月面着陸ゲームを作ろう

4日目 Part 1 月面着陸ゲームを作ろう

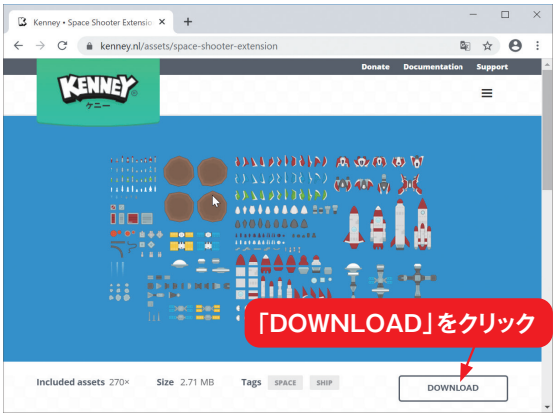
先生 今日は、「月面着陸ゲーム」を作ろう。これはゲームプログラミング入門で作る定番のゲームなんだ。簡単に作れる割には結構面白い。舞台は当然「月面」だから、操作するのはやっぱり「宇宙ロケット」がいいね。

タツヤ ダウンロードした画像ファイルの中に、ロケットの画像はありませ〜ん。

先生 それじゃ、ケニーのサイトで探すことにしよう。

レイ う〜ん、これなんかいいんじゃない（図1）。

図1●ケニーのサイト
https://kenney.nl/assets/space-shooter-extension



先生 おっ、イイ感じのロケットがあるね。それじゃ、このページの画像を使おう。

タツヤ 「kenney_spaceshooterextension.zip」をダウンロードできたよ。

先生 それでは、ファイル名と同じ「kenney_spaceshooterextension」フォルダーをimagesフォルダーに作ろう。そして、ファイルの中身を、作ったフォルダーにコピーしよう。ロケット画像は「PNG」→「Sprites X2」→「Rockets」フォルダーの中にあるから、好きなのを選ぶといいよ。

レイ どのロケットにしようかな?

タツヤ 僕は、spaceRockets_003.pngにしたよ。これを、imagesフォルダーにコピーするんだったね。

先生 ロケットの画像が決まったら、サイズを扱いやすい「50×100」ピクセルに変更しよう。画像のサイズ変更は、もうできるよね。やってみて。

タツヤ 画像のサイズ変更は、前に作ったimg_resize.pyを修正すればいいはずだ。50×100だと…、こんな感じか…。

img_resize.py

```
from PIL import Image

img = Image.open('./images/spaceRockets_003.png')
img_resize = img.resize((50, 100))
img_resize.save('./images/rocket.png')
```

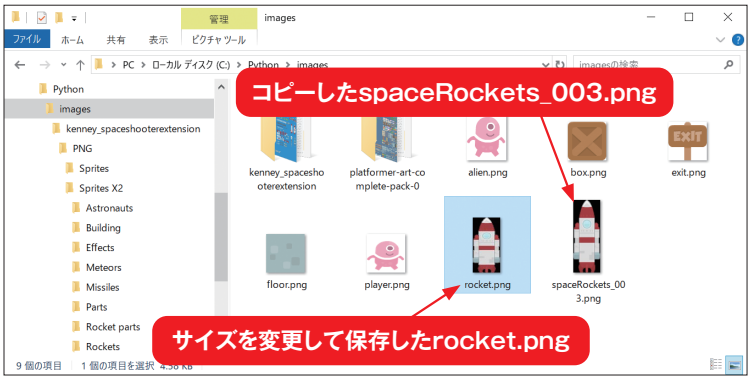
読み込む画像ファイル

サイズを50×100にする

変換後のファイル名をrocket.pngにする

先生 完璧だ。実行しても大丈夫だよ。これでロケットの画像は用意できた（**図2**）。今度は、背景を考えよう。月面の表現は、画像を使った方がリアルだけど、今回は単純な描画メソッドでスクリーンに直接描いてしまう。

図2●ロケット画像を準備できた



タツヤ 絵を描くメソッドもあるのかぁ。

先生 スクリーンに線や図形、文字を描くには、screenオブジェクトのアトリビュートが持つメソッドを使う（**表1**）。なお、オブジェクトの中のオブジェクトのメソッドを使う場合は、「screen.draw.line」のように「.」（ドット）でつなげて表現するんだ。

表1●描画用メソッド

描画用のメソッド	機能
screen.fill((red, green, blue)[, gcolor=(r, g, b)])	スクリーンを指定色で塗りつぶす
screen.draw.line(start, end, (r, g, b))	start 座標から end 座標まで指定色で直線を描画する
screen.draw.circle(pos, radius, (r, g, b))	指定色で円の輪郭を描画する

表1の続き

screen.draw.filled_circle(pos, radius, (r, g, b))	指定色で中を塗りつぶした円を描画する
screen.draw.rect(rect, (r, g, b))	指定色で四角形の輪郭を描画する
screen.draw.text(text, [pos,]**kwargs)	テキストを描画する※
screen.draw.textbox(text, rect, **kwargs)	指定した領域（rect）にテキストを描画する※

※kwargsは、https://pygame-zero.readthedocs.io/ja/latest/ptext.htmlを参照

レイ いろんなメソッドがあるのね。

先生 Pygame Zeroのドキュメント（https://pygame-zero.readthedocs.io/ja/latest/）を見ると、さらに詳しい使い方が書いてあるよ。

タツヤ ドキュメントが、日本語で助かった…。

先生 Pygame Zeroはイギリスで作られているライブラリなんだけど、日本語のドキュメントがあるのはありがたいよね。それでは、次のコードで図形や文字を表示してみよう。

draw_test.py

```
import pgzrun

WIDTH = 400
HEIGHT = 600

def draw():
    screen.clear()
    rect = Rect((150, 50),(100, 100))
    screen.draw.filled_rect(rect, 'WHITE')
    screen.draw.circle((200, 270), 50, 'RED')
    screen.draw.line((50, 370), (350, 450), 'GREEN')
    screen.draw.text('Pygame Zero', (50, 480), color='YELL
OW', fontsize=70)

pgzrun.go()
```

Rect関数には作成する四角形の左上と右下の座標をtupleで渡す

渡された四角形を指定した色で塗りつぶす

円の外枠を指定した色で描く

指定した色で線を描く

テキストの描画

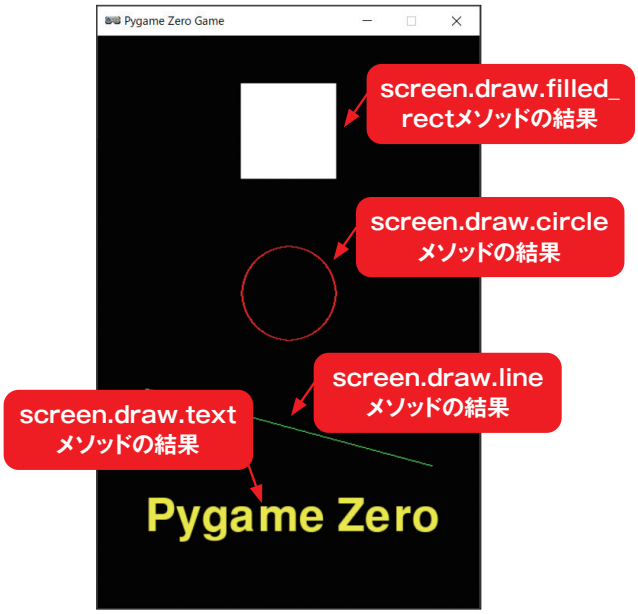
タツヤ お、表示できた（図3）。

レイ Rect（レクト）って関数は、初めてじゃない?

先生 Rect関数は、四角形を描くときの右上の座標と左下の座標を保持するオブジェクトだ。どちらもx座標とy座標をセットにしたtupleで表現する。textメソッドでは「キーワード引数」という方法で「引数を指定」してオブジェクトを渡している。

レイ 「color='YELLOW'」と「fontsize=70」のことね。

図3●draw_test.pyの実行結果



先生 textメソッドは、本来たくさんの引数を持っているんだけど、省略できる引数もある。描画用メソッドの表の中に「**kwargs」と書かれた引数があっただろう。この引数は省略された引数が複数あることを表していて、使いたい引数があったら引数の名前（キーワード）に「=」を付けてオブジェクトを関連付ける。

タツヤ つまり、colorという引数に'YELLOW'という文字列を渡し、fontsizeという引数に70という数値を渡しているんだ。

先生 そういうこと。次はこれらのメソッドを駆使して、ゲームのオープニング画面を作るよ。

4日目 Part 2 オープニング画面を作る

先生 それでは、月面着陸ゲームのオープニング画面を考えたので見てくれるかな。ゲームタイトルは「Moon Lander」（ムーンランダー）にした（**図1**）。

タツヤ おおっ、ちょっとだけカッコイイ。

レイ ロケットから炎が出てるわ。星もキレイ。

先生 下にある着陸台も、スクリーンに直接lineメソッドで描いているんだ。ソースコードはopening.pyとして保存したので確認して欲しい。

図1●ゲームのオープニング画面

opening.py

```
import pgzrun
import random

WIDTH = 400
HEIGHT = 600

rocket = Actor('rocket', center=(200,300))

def draw():
    screen.clear()
    for i in range(20):
        rect = Rect((random.randrange(WIDTH), random.randr
ange(HEIGHT)),(2, 2))
        screen.draw.rect(rect, 'WHITE')
    for i in range(10):
        screen.draw.circle(rocket.midbottom, i + 1, (255,
i*20, 0))
    for i in range(50):
        screen.draw.line((150-i*3, 550+i), (250+i*3,
550+i), (128, 128, 128))
    screen.draw.text("Moon Lander", (50, 100), owidth=1.5,
ocolor=(255,255,0), color=(0,0,0), fontsize=64)
    rocket.draw()

pgzrun.go()
```

星の位置をランダムに生成

星の描画

ロケットの噴射

ロケットの描画

先生 コードの前半は、今までと変わらない。スクリーンのサイズを決めて、ロケットのActorオブジェクトを生成し、変数rocketと関連付けている。ロケットの表示位置は、画面中央に設定した。

レイ 「import random」のrandomって、乱数を扱うオブジェクトかしら?

先生 そう。今回はスクリーンに星を描く際に、星の位置をランダムに決めるのに使っている。星は複数あるのでfor文で繰り返し描いている。

背景の星を描画する

```
import random

...

def draw():
    screen.clear()
    for i in range(20):
        rect = Rect((random.randrange(WIDTH), random.randr
ange(HEIGHT)),(2, 2))
        screen.draw.rect(rect, 'WHITE')
```

乱数を扱うためのrandomオブジェクト
20個の星を描く
ランダムにx座標を生成
ランダムにy座標を生成
白で星を描画

タツヤ randomオブジェクトの「randrange」(ランダムレンジ)が、乱数を作るメソッドだよね。引数にスクリーンのサイズを指定するのはなぜ?

先生 randrangeメソッドは、引数に渡された範囲でランダムな整数を生成する。こうしておけば、スクリーンの外にある座標は生成しなくなるだろう。

タツヤ なるほど。

レイ そして作り出した座標に、2×2ピクセルの四角形を「'WHITE'」で描いているのね。

先生 色の指定は、'WHITE'や'GREEN'といった名前で指定することもできるし、(r, g, b)のように、3原色の数値で指定することもできる。この場合、(0, 0, 0)が黒、(255, 255, 255)が白だ。ロケットの噴出口から出ている炎は、ロケットのmidbottomの位置に円を描いて作っているんだけど、RGBの数値で指定してグラデーションにしている。

ロケットの噴出

```
for i in range(10):
    screen.draw.circle(rocket.midbottom, i + 1, (255,
i*20, 0))
```

10個の円で炎を描く
半径を徐々に大きくする
炎の色を徐々に変える
炎を描く位置

タツヤ なるほどぉ。炎のように見えるのは、少しずつ大きくなる同心円の色を変化させているんだ。

先生 着陸台の方は、lineメソッドを使って描いている。こちらは、線の長さを徐々に広げることで台形にした。

着陸台を描く

```
for i in range(50):
    screen.draw.line((150-i*3, 550+i), (250+i*3, 550+i),
'GRAY')
```

50個の線で台を描く
線の色
上から下へ徐々に広がっていくように線を引く

レイ 台形を直接塗りつぶすようなメソッドはないの?

先生 現在のバージョンのPygame Zeroには、任意の多角形を塗りつぶすメソッドはないようだね。ここでは、lineメソッドの応用例として考えるといいと思うよ。

タツヤ テキストがカッコイイと思ったら、何だか、いろんなキーワード引数を指定しているぞ。

縁取り付きテキストの表示

```
screen.draw.text("Moon Lander", (50, 100), owidth=1.5, oco
lor='YELLOW', color='BLACK', fontsize=64)
```

描画するテキスト
テキストの座標
縁取りの太さ
線の色
テキストの中の色
フォントサイズ

先生 縁取りされた文字にするためだよ。縁取り文字にするには、owidthという引数に1.0を指定する。1.0が標準で、この値を変えると縁取りの太さが変化する。ocolorは縁の色、colorは文字の色だ。説明はこんなところだ。それじゃ、opening.pyのコードを参考にして、オリジナルのオープニング画面を作ってほしい。

レイ ハーイ。

4日目 Part 3 月面着陸ゲームの完成

先生 最後は、ロケットの動作部分を作成しよう。ゲーム画面では、ロケットは月の重力に引き寄せられて落下を始める。そのままでは墜落してしまうので、UPキー（矢印キーの上）を押してエンジンから炎を噴出して機体を上に押し上げる。ソフトランディングできたらゲームクリア、衝突してしまったらゲームオーバーだ。

タツヤ 月面でのロケットの動きをシミュレーションするんだね。

先生 このシミュレーションは、それほど難しくない。

レイ それって、専用の関数があるの?

タツヤ この計算、確かScratchでやったことあるなぁ。自然落下するときは、速度に加速度をプラスし続けるんだっけ?

先生 そうそう。よく知ってるね。今回は、月面の重力、つまり下向きにかかる加速度を0.1にしよう。また、ロケットがエンジンを噴出したときに発生する加速度を-0.1とする。

タツヤ それで毎回、速度に加速度を足すから、速度は常に変化するんだね。で、速度が変化すると、ロケットの位置も変化する。

先生 すばらしい! その通り。

タツヤ じゃあ、UPキーが押されたら加速度を-0.1にして、押されていなければ0.1にすればいいんだ。

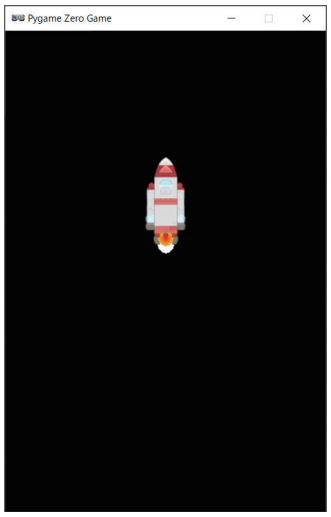
先生 今回は、update関数内で「keyboard.up」がTrueかどうかを調べることで、UPキーが押されているかどうかを判定しよう。この「keyboard」はPygame Zeroがあらかじめ用意しているオブジェクトで、「up」というアトリビュートを持つんだ。それじゃ、次の「lander_test.py」でロケットの動きを確認しよう。

lander_test.py

```
import pgzrun

WIDTH = 400
HEIGHT = 600

rocket = Actor('rocket', center=(200, 100))

speed = 0           # 速度
acceleration = 0.1  # 加速度
key_flg = False     # UPキーが押されるとTrue

def draw():
    screen.clear()
    if key_flg: # UPキーが押されたら炎を噴出
        for i in range(10):
            screen.draw.circle(rocket.midbottom, i + 1, (
255, i*20, 0))
    rocket.draw()

def update():
    global speed          # グローバル変数
    global acceleration   # グローバル変数
    global key_flg        # グローバル変数
    if keyboard.up:       # UPキーが押されていた場合の処理
        key_flg = True
        acceleration = -0.1 # 加速度を-0.1にする
    else:
        key_flg = False
        acceleration = 0.1 # 加速度を0.1にする
    speed += acceleration # 速度に加速度を加える
    rocket.y += speed # ロケットのy座標に速度を加える

pgzrun.go()
```

タツヤ いい感じ～（**図1**）。

図1●ロケットの動作確認



レイ 速度や加速度、UPキーが押されているかどうかを調べる変数は、「グローバル変数」だって…。コレは何?

先生 変数には利用できる範囲があって、この範囲のことを「スコープ」と呼んでいる。変数のスコープは、「グローバル変数」と「ローカル変数」に大別できるんだ。

タツヤ 何が違うんだ?

先生 大雑把に言うと、関数の中で初めて関連付けを行った変数は「ローカル変数」になる。ローカル変数は関数の外では使えない。対して、関数の外で初めて関連付けをした変数は「グローバル変数」になって、複数の関数から利用できる。ただし、関数の中でグローバル変数に別のオブジェクトを関連付けたい場合は、関数の中で「global」キーワードを使って宣言する必要がある。

レイ だから、update関数の中で「global」キーワードが出て来るのね。

先生 このプログラムでは、速度と加速度と、UPキーが押されているかどうかを保持するkey_flgをグローバル変数にしている。

タツヤ でも、「グローバル変数は使わない方がいい」って、どこかで聞いた気がするなぁ。

先生 よく知ってるね。確かにグローバル変数はどこからでも変更が可能なので、思わぬバグの原因となる可能性がある。だから、できるだけ使わない方がいいとされている。でも、この程度の短いプログラムなら、グローバル変数を使っても問題はないよ。

タツヤ なるほど。短いプログラムならグローバル変数によるバグが発生する可能性は少なそうだ。

先生 最後に、画面の切り替えをどうするか考えよう。オープニング画面からゲーム画面

への切り替えは、スペースキーを使うとして、画面が切り替わったことを覚えておく変数が必要だ。ここでは「status」という名前の変数にしよう。そして、draw関数やupdate関数では、その変数statusから現在のプログラムの状態を調べて処理を分岐する。

status_test.py

```
import pgzrun
import random

WIDTH = 400
HEIGHT = 600

rocket = Actor('rocket', center=(200, 300))

star = []
for i in range(20):
    rect = Rect((random.randrange(WIDTH),
                 random.randrange(HEIGHT)),(2, 2))
    star.append(rect)

speed = 0           # 速度
acceleration = 0.1  # 加速度
key_flg = False     # UPキーが押されるとTrue
status = 0          # 0:Opening、1:ゲーム中

def draw():
    global status
    screen.clear()
    # 星の描画
    for i in range(20):
        screen.draw.rect(star[i], 'WHITE')
    # 着陸台の描画
    for i in range(50):
        screen.draw.line((150-i*3, 550+i), (250+i*3, 5
50+i), 'GRAY')
    if status == 0: # オープニング
        # ロケットの炎
        for i in range(10):
            screen.draw.circle(rocket.midbottom,
                               i + 1, (255, i*20, 0))
        # ゲームタイトル
        screen.draw.text("Moon Lander", (50, 100), owid
th=1.5, ocolor='YELLOW', color='BLACK', fontsize=64)
    elif status == 1: # ゲーム中
        if key_flg: # UPキーが押されたら炎を噴出
```

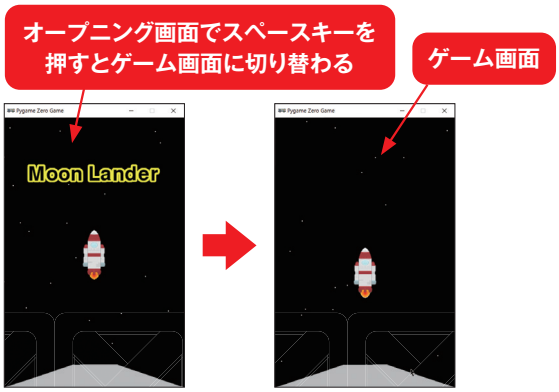
status_test.pyの続き

```
            for i in range(10):
                screen.draw.circle(rocket.midbottom,
                                   i + 1, (255, i*20, 0))

    rocket.draw()


def on_key_down(key):
    global status
    if key == keys.SPACE:
        if status == 0:
            status = 1

def update():
    global speed          # グローバル変数
    global acceleration   # グローバル変数
    global key_flg        # グローバル変数
    global status         # グローバル変数
    if status == 1:
        if keyboard.up:       # UPキーが押されていた場合の処理
            key_flg = True
            acceleration = -0.1 # 加速度を-0.1にする
        else:
            key_flg = False
            acceleration = 0.1 # 加速度を0.1にする
        speed += acceleration # 速度に加速度を加える
        rocket.y += speed # ロケットのy座標に速度を加える

pgzrun.go()
```

タツヤ できた、できた（図2）。statusという変数は、現在のプログラムの状態を覚えているんだねぇ。こうやって、ゲームのシーンを切り替えているんだなぁ…。感動だぁ…。

レイ 普段遊んでいるゲームの仕組みがわかってきたわ。

図2●status_test.pyの実行結果



先生 さぁ、最後はロケットの底と着陸台の高さが一致したら、ゲームをクリアにする処理だ。単純に座標が重なったらクリアだとゲームにならない。そこで、速度が大きいときは「衝突」と考えて「ゲームオーバー」にしよう。それから、着陸に失敗したらロケットが倒れるアニメーションも加えよう。これが、完成形のコード「moon_lander.py」だ。

moon_lander.py

```
import pgzrun
import random

WIDTH = 400
HEIGHT = 600

rocket = Actor('rocket', center=(200, 300))

# 星の座標
star = []
for i in range(20):
    rect = Rect((random.randrange(WIDTH), random.randrange
(HEIGHT)),(2, 2))
    star.append(rect)

speed = 0           # 速度
acceleration = 0.1  # 加速度
key_flg = False     # UPキーが押されるとTrue
status = 0          # 0:Opening、1:ゲーム中、2:GAME CLEAR、3:GAME
OVER
anime_r = animate(None)

def draw():
    global status
    screen.clear()
    # 星の描画
    for i in range(20):
        screen.draw.rect(star[i], 'WHITE')
    # 着陸台の描画
    for i in range(50):
        screen.draw.line((150-i*3, 550+i), (250+i*3,
550+i), 'GRAY')
    if status == 0: # オープニング
        # ロケットの炎
        for i in range(10):
            screen.draw.circle(rocket.midbottom, i + 1, (
```

moon_lander.pyの続き

```
255, i*20, 0))
        # ゲームタイトル
        screen.draw.text("Moon Lander", (50, 100), owid
th=1.5, ocolor='YELLOW', color='BLACK', fontsize=64)
    elif status == 1: # ゲーム中
        if key_flg: # UPキーが押されたら炎を噴出
            for i in range(10):
                screen.draw.circle(rocket.midbottom, i + 1,
(255, i*20, 0))
    elif status == 2:
        screen.draw.text("GAME CLEAR", (40, 300), owid
th=1.5, ocolor='YELLOW', color='BLACK', fontsize=64)
    else:
        screen.draw.text("GAME  OVER", (50, 300), owid
th=1.5, ocolor='RED', color='BLACK', fontsize=64)
    rocket.draw()

def on_key_down(key):
    global status, speed, acceleration
    if key == keys.SPACE and anime_r.running != True:
        if status == 0 or status == 2 or status == 3:
            status = 1
            rocket.y = 200
            speed = 0
            acceleration = 0.1
            rocket.angle = 0

def update():
    global speed, acceleration, key_flg, status, anime_r
    if status == 1:
        if keyboard.up:       # UPキーが押されていた場合の処理
            key_flg = True
            acceleration = -0.1 # 加速度を-0.1にする
        else:
            key_flg = False
            acceleration = 0.1 # 加速度を0.1にする
        speed += acceleration # 速度に加速度を加える
        rocket.y += speed # ロケットのy座標に速度を加える
        if rocket.y > 500:
            if speed < 1.0:
                status = 2 # GAME CLEAR
            else:
                status = 3 # GAME OVER
                # 着陸失敗でロケットが傾くアニメ
```

moon_lander.pyの続き

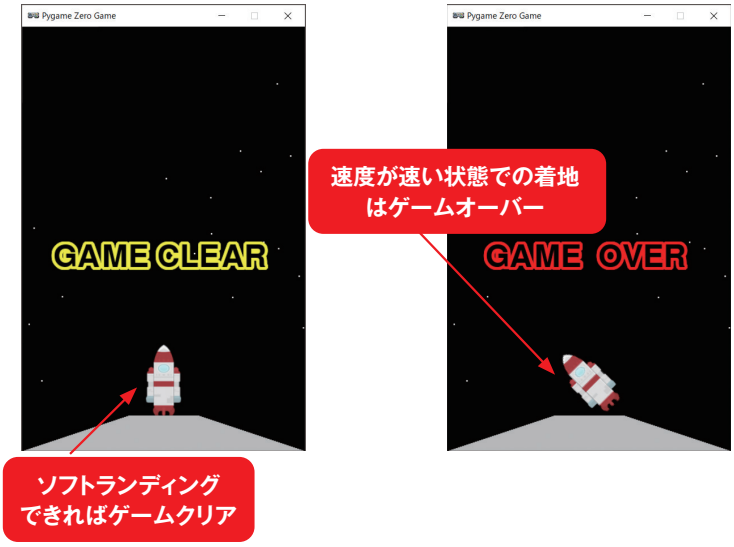
```
                anime_r = animate(rocket, 'bounce_start_end', 1, angle=45)

pgzrun.go()
```

タツヤ 速度が1.0より小さければソフトランディング成功ってことだけど、慣れないと難しい。結構ハマるねぇ、コレ（**図3**）。

レイ 着陸に失敗すると、ガタガタ傾くところがかわいいわ。私、ソフトランディングの速度の値を、1.0から3.0に変えたけど、コレだとすぐ成功しちゃってつまらなくなるわね。

先生 ゲームを面白くするには、いろいろな設定を微妙に調整する必要があるね。ぜひ、このプログラムをベースに、もっと面白いゲームを考えてほしい。

図3●プレイ画面



タツヤ センセイ、まかせてくださいよ〜。

レイ ハ〜イ、タツヤ君、また調子にのってま〜す。

先生 次回は、最終日。いよいよシューティングゲームに挑戦だ。

5日目
シューティングゲームを作ろう

5日目 Part 1 敵にミサイルを発射する

先生 最終日は、シューティングゲームを作ろう。

タツヤ やっぱり、ゲームと言ったらシューティングだよ。

レイ でも、難しそう…。

先生 シューティングゲームはそれほど難しくはないよ。ただ、どうしても数学的な知識が少し必要になる。それと、大量のオブジェクトを扱うので、効率的にオブジェクトを管理する手法を知っておかないといけないね。

タツヤ 数学は得意なんだけど、オブジェクトってやつがイマイチなんだよなぁ…。

先生 オブジェクトの理解には、とにかくいろんなコードを書いてみることが一番の早道になる。あわてず、コツコツいろんなゲームを作ってみよう。

レイ 急がば回れってヤツね。

先生 それでは、シューティングゲームに登場するオブジェクトから考えよう。最小構成でも、「自分の機体」、「敵の機体」、「双方の武器」は必要だろう。利用できそうな画像は「kenney_spaceshooterextension」フォルダーの「PNG」→「Sprites」→「Ships」フォルダーにありそうだ。今回は「spaceShips_007.png」を敵の機体、「spaceShips_003.png」を自分の機体にする。また、「Sprites」→「Missiles」フォルダーにある「spaceMissiles_037.png」を敵のミサイル、「spaceMissiles_012.png」を自機のミサイルにしよう。これらの画像ファイルimagesフォルダーにコピーする（**図1**）。

図1●シューティングゲームで利用する画像ファイル

先生 画像をimagesフォルダーにコピーできたら、自機の方向を変更しよう。

レイ 方向を?

先生 Actorオブジェクトには、角度を設定できる「angle」というアトリビュートがある。だから、表示する画像の角度は自由に変更できる。ただ、あらかじめ画像の上下を反転させておいた方がプログラムは少し楽になる。次のように、img_resize.pyを修正して実行しよう。

img_resize.py

```
from PIL import Image

img = Image.open('./images/spaceShips_003.png')
img_rotate = img_resize.transpose(Image.ROTATE_180)
img_rotate.save('./images/shooter.png')
```

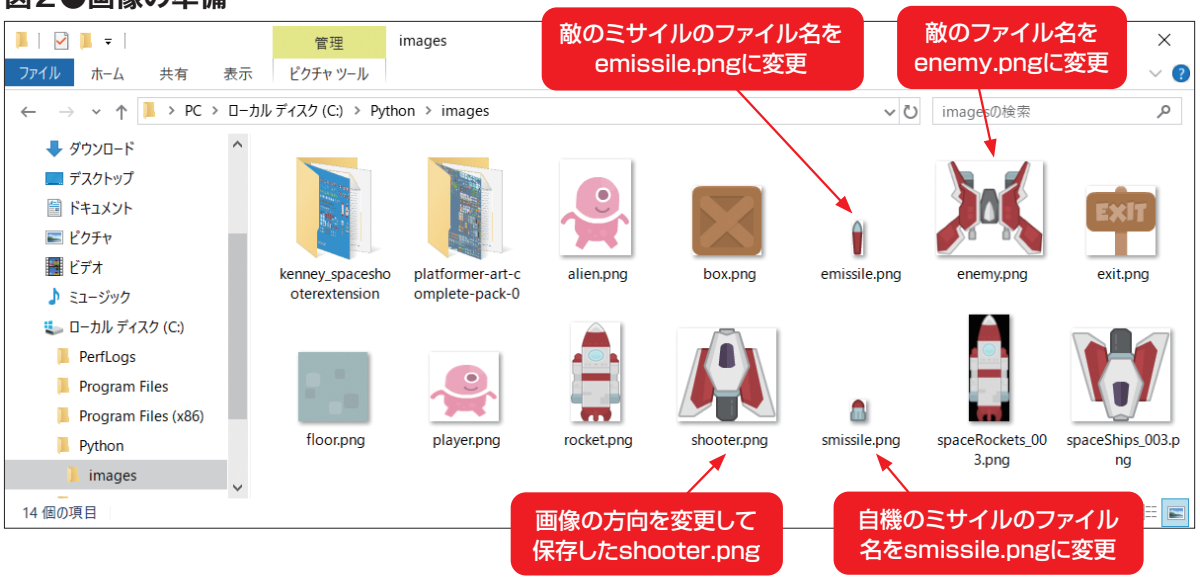
読み込む自機の画像ファイル

画像を上下反転させる

ファイル名をshooter.pngにする

タツヤ スプライトの画像は、準備できたぞ（図2）。

図2●画像の準備



先生 それでは、シューティングゲームの基本画面を紹介する。次のコードを「shooting_test.py」で保存して実行してみよう。

shooting_test.py

```
import pgzrun
import random

WIDTH = 800
HEIGHT = 600

enemy = Actor('enemy.png', (400, 100))
shooter = Actor('shooter.png', (400, 500))

star = []
for i in range(30):
    rect = Rect((random.randrange(WIDTH), random.randrange
(HEIGHT)), (2, 2))
    star.append(rect)

status = 0    # ゲームステータス

def draw():
    global status
    screen.clear()
    # 星の描画
    for i in range(len(star)):
        screen.draw.rect(star[i], 'WHITE')
    if status == 0: # オープニング
        # ゲームタイトル
        screen.draw.text('S P A C E  S H O O T E R', (100,
290), color='WHITE', gcolor='YELLOW', fontsize=72)
    enemy.draw()
    shooter.draw()

pgzrun.go()
```

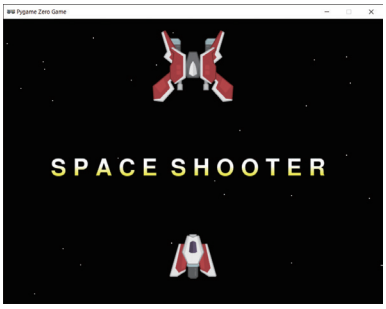
len関数は、starの要素数を返す

statusによって、ゲームのシーンを変更する

レイ 星がキレイ（図3）。ところで、このfor文にある「len(star)」ってどういう意味?

先生 星の数を数字で入力してしまうと、後から変更するときに面倒になる。len関数は、listの要素数を返す関数なので、こうしておけばstarの要素が30個あれば30になるし、50個なら50になる。

図3●shooting_test.pyの実行結果



タツヤ 作った星の数を返すってわけだ。

先生 オープンニング画面はこんな感じで良いだろう。次は、スペースキーが押されたら、ゲーム画面に切り替わって、ミサイルを発射できるようにしよう。ミサイルを発射するキーも、同じスペースキーとする。押し続けると連射する機能は考えないことにして、とりあえず、スペースキーを押すたびに1発ずつ発射するようにしよう。それではタツヤ君、コードを書いてみて。

タツヤ ハイハイ、任せなさ〜い。まずはミサイルのスプライトを用意して、最初は表示しないようにしておくと…。で、もしスペースキーが押されたらミサイルを表示して、上に移動するようにすればいいんだ。キーが押されたときのイベントハンドラはon_key_down関数だから、この関数の中でスペースキーかどうかをif文で判断して、表示するための変数をFalseからTrueに変更すれば…、う〜ん。

先生 どうしたの。

タツヤ コードを入力していて気が付いたんだけど、スペースキーを押すたびにミサイルが発射されるということは、スペースキーを押すたびにミサイルのスプライトを生成しないとだめだ!

先生 すばらしい。よく気が付いたね。今までのゲームは、あらかじめ生成しておいたスプライトを表示して動かしていた。でも画面に登場するスプライトの数があらかじめ決まっていない場合、ゲームの進行中にスプライトの生成を行う方が効率が良い。

レイ えぇ〜、どうすればいいの〜。

先生 こんなときは、単なる変数ではなく、複数のオブジェクトを保存できるlistを使うんだ。スペースキーが押されたら、ミサイルのActorオブジェクトをlistに追加していけばいい。描画するActorオブジェクトはdraw関数を呼び出す必要があるから、for文を使ってlistからミサイルを順番に取り出して、draw関数を呼ぶ。スペースキーを押すとミサイルを発射するだけのコードなら、こんな感じで書けるはずだ。

missile_test.py

```
import pgzrun

WIDTH = 800
HEIGHT = 600

shooter = Actor('shooter.png', (400,500))

s_missiles = []

def draw():
    screen.clear()
    for missile in s_missiles:
        missile.draw()
    shooter.draw()

def update():
    for missile in s_missiles:
        missile.y -= 10

def on_key_down(key):
    global s_missiles
    if key == keys.SPACE:
        s_missiles.append(Actor('smissile.png', shooter.
pos))

pgzrun.go()
```

ミサイルを格納するlist

listから順番にミサイルを取り出し、draw関数で描画

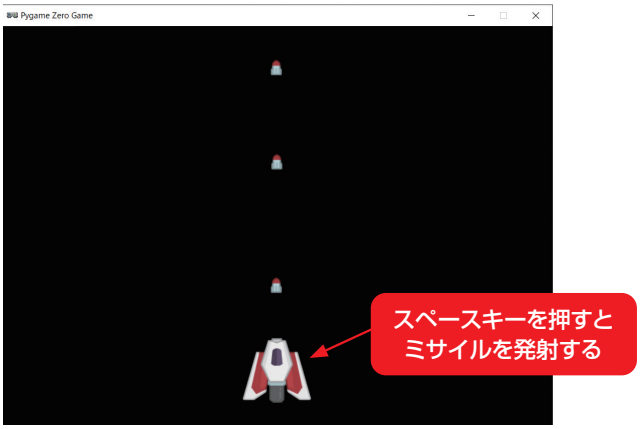
listから順番にミサイルを取り出し、上に座標を移動

スペースキーが押されるたびにミサイルをlistに追加

タツヤ おお、すげえ（図4）。listにActorオブジェクトをappendしておいて、for文で取り出しながら使うのか…。

先生 listとfor文の使い方がミソだね。それじゃ、少し休憩しよう。

図4●missile_test.pyの実行画面



5日目 Part 2 衝突判定

先生 さぁ、次はレイちゃんに考えてもらおう。敵（enemy）がミサイルと接触したことを調べるにはどうすれば良いかな?

タツヤ ずるいなぁ。簡単じゃん。

レイ ミサイルと敵が接触した判断は、ミサイルの座標と敵の座標が重なっているかどうかで判断するんじゃない?

先生 そう。ただ、ミサイルと敵には大きさがある。大きさを加味して調べる必要があるから少し面倒だ。そこでActorオブジェクトには、衝突判定専用のメソッドが用意されている（**表1**）。

表1●衝突判定用のメソッド

衝突判定メソッド	使い方
Actor オブジェクト .colliderect(rect)	引数は Rect オブジェクト※。接触していると True を返す
Actor オブジェクト .collidepoint(pos)	引数は pos（x 座標と y 座標の tuple）。接触していると True を返す

※RectオブジェクトはRect関数で生成する。Rect関数の第1引数は、左上x座標とy座標のtuple、第2引数は幅と高さのtuple。

レイ これなら、座標計算をしなくても、if文だけで衝突しているかどうかを調べられるわね。

先生 では、レイちゃん。敵にミサイルが衝突したら表示しているHP（ヒットポイント）を減らしてミサイルを消す処理を、missile_test.pyに追加できるかな?

レイ うーん。

先生 HPは30から減らすように。ミサイルを消すにはlistのremoveメソッドを使うと簡単だ。衝突したミサイルをremoveメソッドの引数に渡すだけで、listからそのオブジェクトを削

除できる。

レイ わかりました。やってみます…。とりあえず、HP用のグローバル変数を用意して、HPの表示はdraw関数の最後でいいわよね。次に、update関数内のfor文で取り出したミサイルからRectを作って、敵オブジェクトとcolliderectメソッドで衝突を判定する…。こんな感じかしら…。

missile_test.py

```
import pgzrun

WIDTH = 800
HEIGHT = 600

shooter = Actor('shooter.png', (400, 500))
enemy = Actor('enemy.png', (400, 100))

enemy_hp = 30
s_missiles = []

def draw():
    screen.clear()
    for missile in s_missiles:
        missile.draw()
    shooter.draw()
    enemy.draw()
    screen.draw.text('Enemy HP = ' + str(enemy_hp), (50,
50), color='YELLOW', fontsize=32)

def update():
    global enemy_hp
    for missile in s_missiles:
        missile.y -= 10
        rect = Rect(missile.topleft,(16, 22))
        if enemy.colliderect(rect):
            enemy_hp -= 1
            s_missiles.remove(missile)

def on_key_down(key):
    global s_missiles
    if key == keys.SPACE:
        s_missiles.append(Actor('smissile.png', shooter.
pos))

pgzrun.go()
```

敵のHP

判定用にミサイルのRectを作成

敵と衝突判定

衝突したのでミサイルを削除する

先生 うまくいったね（図1）。

レイ どの関数で衝突判定するのがいいのか迷ったんだけど、update関数でよかったみたい。

先生 それでは、さらにmissile_test.pyを改良して、スプライトに動きを与えていこう。まず、敵が自動的に左右に動くように処理を追加する。そして、自機も左右の矢印キーで動くようにしよう。自機のHP表示も追加しておこう。

図1●missile_test.pyの実行画面

missile_test.py

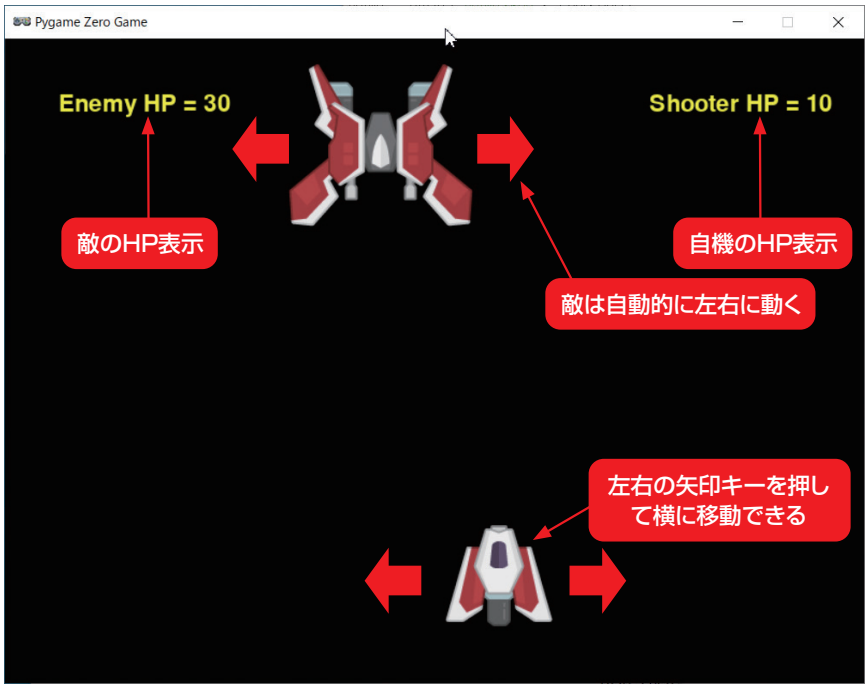
```
import pgzrun

WIDTH = 800
HEIGHT = 600

shooter_hp = 10 # playerのHP（0になるとゲームオーバー）
enemy_hp = 30   # 敵のHP（0になるとゲームクリア）
s_missiles = [] # playerのミサイルを格納するlist
turn = False    # 敵を左右に移動させるためのフラグ

shooter = Actor('shooter.png', (400, 500))
enemy = Actor('enemy.png', (400, 100))

def draw():
    screen.clear()

    # ミサイルの描画
    for missile in s_missiles:
        missile.draw()
    shooter.draw()
    enemy.draw()

    # HPの表示
    screen.draw.text('Enemy HP = ' + str(enemy_hp),
                     (50,50),color='YELLOW',fontsize=32)
    screen.draw.text('Shooter HP = ' + str(shooter_hp),
                     (600,50),color='YELLOW',fontsize=32)
```

missile_test.pyの続き

```
def update():
    global enemy_hp, turn
    # 自機のキー操作
    if keyboard.left:
        if shooter.x > 47:
            shooter.x -= 3
    if keyboard.right:
        if shooter.x < WIDTH-47:
            shooter.x += 3
    # 敵を左右に動かす
    if turn :
        enemy.x += 5
        if enemy.x > WIDTH:
            turn = False
    else:
        enemy.x -= 5
        if enemy.x < 0:
            turn = True

    for missile in s_missiles:
        missile.y -= 10
        rect = Rect(missile.topleft,(16, 22))
        if enemy.colliderect(rect):
            enemy_hp -= 1
            s_missiles.remove(missile)

def on_key_down(key):
    global s_missiles
    if key == keys.SPACE:
        s_missiles.append(Actor('smissile.png', shooter.
pos))

pgzrun.go()
```

レイ おおー、かっこいい～（図2）。

図2●スプライトに動きを与える



先生 さぁ、いよいよ敵のミサイル攻撃の処理を作る。敵はミサイルを2発同時に撃つようにして、自機に当たると自機のHPが下がるようにしよう。

missile_test.py

```
import pgzrun

WIDTH = 800
HEIGHT = 600

shooter_hp = 10 # playerのHP(0になるとゲームオーバー)
enemy_hp = 30   # 敵のHP(0になるとゲームクリア)
s_missiles = [] # playerのミサイルを格納するlist
turn = False    # 敵を左右に移動させるためのフラグ
eshot = 60      # 60回カウントして、敵から自機へ向けてミサイルを発射する
e_missiles = [] # 敵のミサイルを格納するlist

shooter = Actor('shooter.png', (400, 500))
enemy = Actor('enemy.png', (400, 100))

def draw():
    screen.clear()

    # ミサイルの描画
    for missile in s_missiles:
```

missile_test.pyの続き

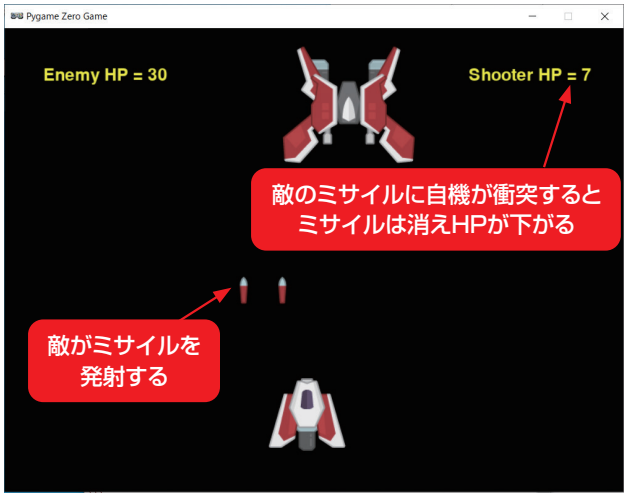
```
        missile.draw()
    for missile in e_missiles:
        missile.draw()
    shooter.draw()
    enemy.draw()

    # HPの表示
    screen.draw.text('Enemy HP = ' + str(enemy_hp),
                     (50, 50),color='YELLOW',fontsize=32)
    screen.draw.text('Shooter HP = ' + str(shooter_hp),
                     (600,50),color='YELLOW',fontsize=32)

def update():
    global enemy_hp, turn, eshot, shooter_hp
    # 自機のキー操作
    if keyboard.left:
        shooter.x -= 3
    if keyboard.right:
        shooter.x += 3
    # 敵を左右に動かす
    if turn :
        enemy.x += 5
        if enemy.x > WIDTH:
            turn = False
    else:
        enemy.x -= 5
        if enemy.x < 0:
            turn = True
    if eshot == 0:
        e_missiles.append(Actor('emissile.png',
        (enemy.x-25, enemy.y+10)))
        e_missiles.append(Actor('emissile.png',
        (enemy.x+25, enemy.y+10)))
        eshot = 60
    else:
        eshot -= 1
    for missile in e_missiles:
        missile.y += 10
        rect = Rect(missile.topleft,(11, 35))
        if shooter.colliderect(rect):
            shooter_hp -= 1
            e_missiles.remove(missile)
    for missile in s_missiles:
        missile.y -= 10
```

missile_test.pyの続き

```
        rect = Rect(missile.topleft,(16, 22))
        if enemy.colliderect(rect):
            enemy_hp -= 1
            s_missiles.remove(missile)

def on_key_down(key):
    global s_missiles
    if key == keys.SPACE:
        s_missiles.append(Actor('smissile.png', shooter.
pos))

pgzrun.go()
```

タツヤ かなりシューティングゲームっぽくなってきた（**図3**）。でも、コレだとミサイルが真っ直ぐにしか飛ばないから、簡単に避けられておもしろくないなぁ。敵の弾が自機に向かってくる処理は難しいのかなぁ。

図3●敵が攻撃する処理を作る



レイ 自機の座標から、敵のミサイルの発射角度を決めればいいんじゃない?

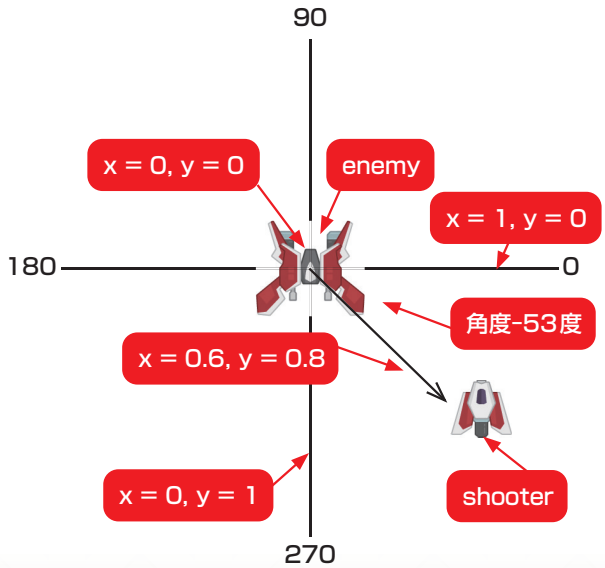
タツヤ 確かに…。でも角度がわかってもミサイルをその角度で動かすにはどうすればいいんだ?

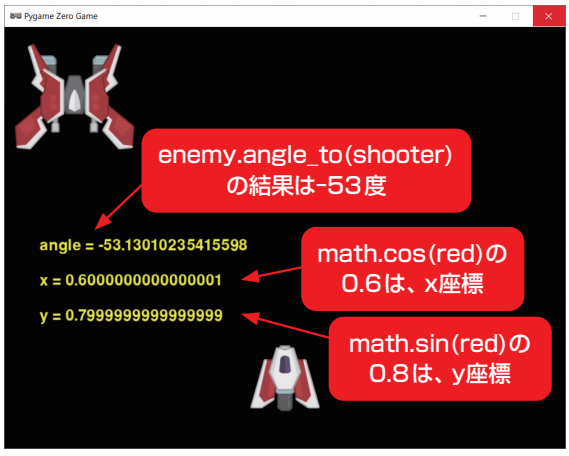
先生 2人とも、何をすべきかわかってきたね。実はActorオブジェクトには「angle_to」という便利なメソッドがある。このメソッドを使うと、2つのスプライト間の角度を取得することができる。次の、angle_test.pyを見てみよう（**図4**）。

図4●弾道の計算（その1）

angle_test.py

```
import pgzrun
import math

WIDTH = 800
HEIGHT = 600

shooter = Actor('shooter.png', (400, 500))
enemy = Actor('enemy.png', (100, 100))

def draw():
    screen.clear()
    shooter.draw()
    enemy.draw()

    angle = enemy.angle_to(shooter)
    red = math.radians(-angle)
    x = math.cos(red)
    y = math.sin(red)
    screen.draw.text('angle = ' + str(angle), (50, 300),
color='YELLOW', fontsize=32)
    screen.draw.text('x = ' + str(x),(50, 350), color='YEL
LOW', fontsize=32)
    screen.draw.text('y = ' + str(y),(50, 400), color='YEL
LOW', fontsize=32)

pgzrun.go()
```





レイ 敵と自機の角度は-53度だってわかるわね。

先生 こうして得られた角度をもとに、x座標とy座標を求めれば、ミサイルを撃つ方向がわかるはずだ。この計算は、「コサイン」(cos)でxを求めて、「サイン」(sin)でyを求める。コサインやサインっていうのは、高校で習う「三角関数」っていう数学なんだけ、2人とも中学生だからまだ習っていないよね。

タツヤ 名前くらいは聞いたことあるよ。

先生 おお、それはすごい。まあ、とにかくここでは、「こういう風に使うんだ」と覚えておいてくれればいい。詳しい意味は高校で勉強してほしい。「数学なんて役に立たない」と言う人が時々いるけど、少なくとも三角関数を知らないと、面白いシューティングゲームは作れないね。

レイ ハーイ。

先生 で、Pythonの場合、三角関数は「mathライブラリ」にあるから、「math」をインポートしよう。すると、「math.cos関数」や「math.sin関数」が使えるようになる。

タツヤ Pythonのライブラリって、何でもあるんだ。スゴいねぇ。

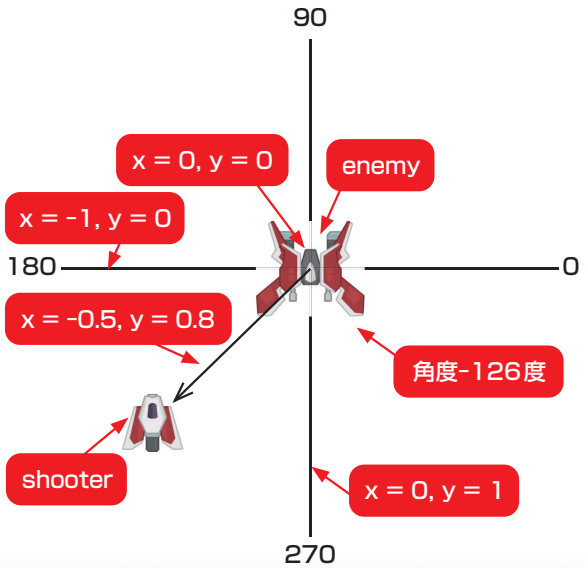
先生 今度は、敵を右上に自機を左下に置く場合だ(**図5**)。

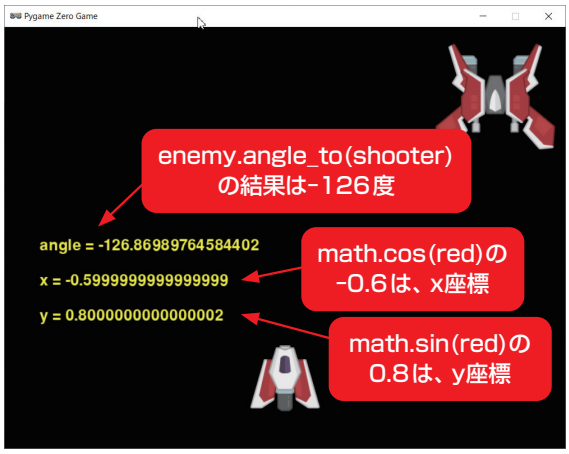
図5●弾道の計算（その2）

angle_test.py

```
import pgzrun
import math

WIDTH = 800
HEIGHT = 600

shooter = Actor('shooter.png', (400, 500))
enemy = Actor('enemy.png', (700, 100))

def draw():
    screen.clear()
    shooter.draw()
    enemy.draw()

    angle = enemy.angle_to(shooter)
    red = math.radians(-angle)
    x = math.cos(red)
    y = math.sin(red)
    screen.draw.text('angle = ' + str(angle),(50, 300), co
lor='YELLOW', fontsize=32)
    screen.draw.text('x = ' + str(x),(50, 350), color='YEL
LOW', fontsize=32)
    screen.draw.text('y = ' + str(y),(50, 400), color='YEL
LOW', fontsize=32)

pgzrun.go()
```

x座標を逆の位置へ移動





レイ 三角関数は正直よくわからないけど、図を見ると何をやっているのかはだいたいわかるわ。

先生 では、この関数を応用して、敵から自機に向かってミサイルを発射するプログラムを紹介する。

missile_test.py

```
import pgzrun
import random
import math

WIDTH = 800
HEIGHT = 600

eshot = 60
s_missiles = []
e_missiles = []
turn = False

enemy = Actor('enemy.png', (400, 100))
shooter = Actor('shooter.png', (400, 500))

def draw():
    screen.clear()
    for missile in s_missiles:
        missile.draw()
    for missile in e_missiles:
        missile.draw()
    enemy.draw()
    shooter.draw()

def update():
    global turn, eshot

    if keyboard.left:
        if shooter.x > 47:
            shooter.x -= 3
    if keyboard.right:
        if shooter.x < WIDTH-47:
            shooter.x += 3

    if turn :
        enemy.x += 5
```

missile_test.pyの続き

```
        if enemy.x > WIDTH:
            turn = False
    else:
        enemy.x -= 5
        if enemy.x < 0:
            turn = True

    if eshot == 0:
        angle = enemy.angle_to(shooter)
        missile1 = Actor('emissile.png', (enemy.x-25, ene
my.y+10))
        missile2 = Actor('emissile.png', (enemy.x+25, ene
my.y+10))
        missile1.angle = 90 + angle
        missile2.angle = 90 + angle
        e_missiles.append(missile1)
        e_missiles.append(missile2)
        eshot = 60
    else:
        eshot -= 1

    for missile in e_missiles:
        red = math.radians(-(missile.angle-90))
        missile.x += (math.cos(red)) * 3
        missile.y += (math.sin(red)) * 3
    for missile in s_missiles:
        missile.y -= 10

def on_key_down(key):
    global s_missiles
    if key == keys.SPACE:
        s_missiles.append(Actor('smissile.png', shooter.
pos))

pgzrun.go()
```

レイ 敵のミサイルが、こっちに向かって飛んでくるわ（図6）。

先生 さぁ、いよいよシューティングゲームの最終形だ。

図6●自機に向かって飛んでくるミサイル

5日目 Part 3 シューティングゲームが完成

先生 それでは、最後にココまでの集大成として、シューティングゲームの完成版「shooting.py」を紹介しよう。

shooting.py

```
import pgzrun
import random
import math

WIDTH = 800
HEIGHT = 600

shooter_hp = 10 # playerのHP(0になるとゲームオーバー)
enemy_hp = 30   # 敵のHP(0になるとゲームクリア)
eshot = 60      # 60回カウントして、敵から自機へ向けてミサイルを発射する
turn = False    # 敵を左右に移動させるためのフラグ
s_missiles = [] # playerのミサイルを格納するlist
e_missiles = [] # 敵のミサイルを格納するlist
status = 0      # 0:オープニング 1:ゲーム中 2:ゲームオーバー 3:ゲームクリア

enemy = Actor('enemy.png', (400, 100))
shooter = Actor('shooter.png', (400, 500))

star = []
for i in range(30):
    rect = Rect((random.randrange(WIDTH), random.randrange(HEIG
HT)),(2, 2))
    star.append(rect)

def draw():
    screen.clear()
    for i in range(len(star)):
        screen.draw.rect(star[i], 'WHITE')
    if status == 0:
        screen.draw.text('S P A C E  S H O O T E R', (100, 290),
                         color='WHITE', gcolor='YELLOW', fontsiz
e=72)
    elif status == 1:
        # ミサイルの描画
        for missile in s_missiles:
            missile.draw()
```

shooting.pyの続き

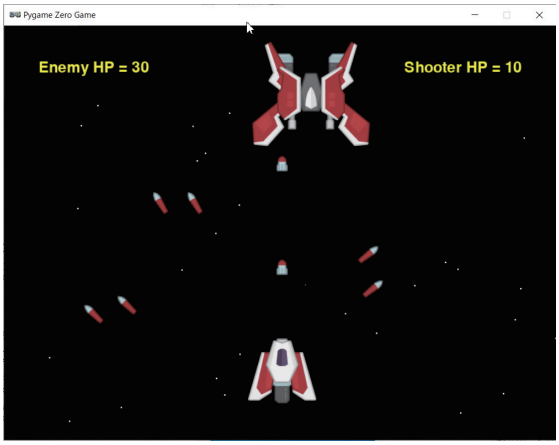
```
        for missile in e_missiles:
            missile.draw()
        # HPの表示
        screen.draw.text('Enemy HP = ' + str(enemy_hp),
                         (50, 50), color='YELLOW', fontsize=32)
        screen.draw.text('Shooter HP = ' + str(shooter_hp),
                         (580, 50), color='YELLOW', fontsize=32)
    elif status == 2:
        screen.draw.text('G A M E  O V E R', (200, 290),
                         color='WHITE', gcolor='YELLOW', fontsiz
e=72)
    else:
        screen.draw.text('G A M E  C L E A R', (180, 290),
                         color='WHITE', gcolor='YELLOW', fontsiz
e=72)
    enemy.draw()
    shooter.draw()

def update():
    global enemy_hp, shooter_hp, turn, eshot, status
    for i in range(len(star)):
        star[i].y += i
        if star[i].y > HEIGHT:
            star[i].y = 0

    if status == 1:
        # 自機のキー操作
        if keyboard.left:
            if shooter.x > 47:
                shooter.x -= 3
        if keyboard.right:
            if shooter.x < WIDTH-47:
                shooter.x += 3
        # 敵を左右に動かす
        if turn :
            enemy.x += 5
            if enemy.x > WIDTH:
                turn = False
        else:
            enemy.x -= 5
            if enemy.x < 0:
                turn = True
        if eshot == 0:
            angle = enemy.angle_to(shooter)
            missile1 = Actor('emissile.png', (enemy.x-25,enemy.
```

shooting.pyの続き

```
y+10))
            missile2 = Actor('emissile.png', (enemy.x+25,enemy.
y+10))
            missile1.angle = 90 + angle
            missile2.angle = 90 + angle
            e_missiles.append(missile1)
            e_missiles.append(missile2)
            eshot = 60
        else:
            eshot -= 1

        for missile in e_missiles:
            red = math.radians(-(missile.angle-90))
            missile.x += (math.cos(red)) * 3
            missile.y += (math.sin(red)) * 3
            rect = Rect(missile.topleft,(11, 35))
            if shooter.colliderect(rect):
                shooter_hp -= 1
                if shooter_hp == 0:
                    status = 2
                e_missiles.remove(missile)
        for missile in s_missiles:
            missile.y -= 10
            rect = Rect(missile.topleft,(16, 22))
            if enemy.colliderect(rect):
                enemy_hp -= 1
                if enemy_hp == 0:
                    status = 3
                s_missiles.remove(missile)

def on_key_down(key):
    global status, s_missiles, e_missiles, enemy_hp, shooter_hp
    if key == keys.SPACE:
        if status == 0 or status == 2 or status == 3:
            enemy_hp = 30
            shooter_hp = 10
            s_missiles = []
            e_missiles = []
            status = 1
        elif status == 1:
            s_missiles.append(Actor('smissile.png', shooter.
pos))

pgzrun.go()
```

タツヤ けっこう、遊べるじゃん（**図1**）。

図1●シューティングゲームの完成



レイ 自分で作ったゲームは、難易度を変更できるところが楽しいわね。

先生 完成版と言っておきながら、本当は実装しきれていない機能がたくさんある。例えば自機を上下に動かす機能とかね。

タツヤ まかせてください。僕が続きを家でやりますから。

レイ 私も、いろいろ改造したい。

先生 ところで、Pythonゲームコースは、どうだったかな?

タツヤ 最初は英語がいろいろ出てきてビビッたけど、Pygame Zeroが便利だから、思ったより簡単だった。

レイ そうね。Pygame Zeroはほんとに簡単! スゴいわ。

先生 今回は、Pythonでゲームを作るための基礎知識に主眼を置いたので、オブジェクト指向の「クラス」などにはまったく触れていない。そのため、グローバル変数を多用したり、同じようなコードが重複して出てきたりして、バージョンアップしにくいコードになっている。

タツヤ そうなの? 結構イケてると思うんだけどなぁ…。

先生 今後は、ぜひ「オブジェクト指向」という分野にも目を向けてがんばってほしい。期待しているよ。それじゃ、これで終わりにしましょう。さようなら。

タツヤ **レイ** ありがとうございました。さようなら。

■筆者紹介

中島 省吾

有限会社メディアプラネット代表取締役。テクニカルライターとして、ネットワークやプログラミング関連の記事を執筆するほか、IT企業向けのセミナーや新人研修の講師なども手掛ける。最近は、ボランティアで子どもたちにもプログラミングを教えている。近著に「ビジネスPython超入門」（日経BP）がある。

5日でできる！ Pythonでゲーム作成入門

著者●中島 省吾
編集●日経ソフトウエア
発行人●村上 広樹
編集長●久保田 浩
デザイン・制作●JMCインターナショナル

発行●日経BP
Nikkei Business Publications, Inc.
発売●日経BPマーケティング
〒105-8308 東京都港区虎ノ門4-3-12
URL●https://nkbp.jp/bpshopqa