

ゲーム作りで覚える Python

0.0 はじめに

本講座は初心者～初級者を対象に、Pygame を使ってPythonの使い方と、Pythonの可能性について触れてもらうことを目的としております。

0.1 要チェックなPython文法

本講座を受けるにあたり、次の文法だけは覚えましょう。あまり細かいことは書きませんので、細かい部分はGoogle先生にお願いします。

<p>インデントとコメント</p> <p>Pythonはインデントで処理ブロックを決めます。そのため、インデントは崩さないように気をつけましょう。ハードタブを使うと楽チンです。</p> <p>コメントは # です。#以降行末までコメント扱いになります。</p> <p>変数</p> <p>Pythonでは、変数に何でも入れられます。数字だろうが文字だろうが、関数だろうが配列だろうがドンと来いです。使い方は、<code>x = 1</code> みたいに、=の左辺に変数名を、右辺に代入するいろいろを書きます。</p> <p>for 文</p> <p>プログラムの利点は、単純動作を高速に行うことです。For文を使うことで、指定回数、好きなだけ繰り返し動作をしてくれます。書き方は次のとおり。</p> <p>for A in B: #ここに処理を書く</p>	<p>配列 Bがなくなるまで、Aに1つずつ代入し、書いた処理を続けます。もし、10回回したいなら、Bを <code>range(10)</code> としましょう。すると、Aに0～9が代入されて、10回回ります。</p> <p>While 文</p> <p>指定条件が FALSE になるまで延々回すにはこれを使います。指定条件を TRUEに固定すれば、無限ループの完成です。</p> <p>while 条件文 : #ここに処理内容</p> <p>関数とクラス</p> <p>一番重要なものですが、それを書くには、この余白では狭すぎますので、講座の中で覚えてください。</p> <p>行末</p> <p>Cなどと違い、行末に ; はいりません。つけても動きますが、正しい文法は何もつけない、です。つけても構いませんが。</p>
--	--

1 . Pygame を使って画面を表示する

適当なディレクトリを作り、その中に「01.py」というファイルを作って、テキストエディタなどで開き、次のプログラムを入力してください。先頭行以外のコメントは入れなくてもいいです。

```
#-*- coding:sjis -*-
import pygame
from pygame.locals import *

def main():
    pygame.init() # pygameの初期化
    screen = pygame.display.set_mode( (640, 480) ) # 画面を作る
    pygame.display.set_caption('Hello pygame') # タイトル
    pygame.display.flip() # 画面を反映

    while 1:
        for event in pygame.event.get(): # イベントチェック
            if event.type == QUIT: # 終了が押された？
                return
            if (event.type == KEYDOWN and
                event.key == K_ESCAPE): # ESCが押された？
                return

if __name__ == '__main__': main()
```

入力が終わったら実行してみましょう。コンソールから `python 01.py` と入れて実行してください。画面がちゃんと出れば成功です。

`def main()` について

C言語などではおなじみのメイン関数ですが、Pythonではmainがなくても、上から順に処理をしてくれます。そのため、今回のようなmain関数は不要とも言えます。なら、なぜ書くのか？ というと、

```
if __name__ == '__main__':
```

で指定された関数を、このスクリプトを実行したときに実行するようにするからです。そのため、複数ファイルの構成になったとき、1つだけテストをしたい場合に、上記if文で実行したい関数を指定すれば、そのファイルだけで動かすことができます。

2. 図形の表示

```
# -*- coding:sjis -*-
import pygame
from pygame.locals import *

def main():
    pygame.init() # pygameの初期化
    screen = pygame.display.set_mode( (320, 240) ) # 画面を作る
    pygame.display.set_caption('Hello pygame') # タイトル

    playerRect = pygame.Rect(0, 0, 40, 10)
    playerRect.center = (160,200) # 描画サイズと表示位置を指定
    pygame.draw.rect(screen, (0,255,128), playerRect) # 四角を描画

    ballRect = pygame.Rect(0, 0, 10, 10)
    ballRect.center = (160,100) # 描画サイズと表示位置を指定
    pygame.draw.ellipse(screen, (255,128,0), ballRect)# 内接円を描画

    while 1:

        pygame.display.flip() # 画面を反映
        for event in pygame.event.get(): # イベントチェック
            if event.type == QUIT: # 終了が押された?
                return
            if (event.type == KEYDOWN and
                event.key == K_ESCAPE): # ESCが押された?
                return

if __name__ == '__main__': main()
# end of file
```

pygame.draw とアンチエイリアス

単純図形はRectやellipseの他に、line、circleなどがあります。いずれも、一部を除き、アンチエイリアス（ギザギザ除去）はされません。そのため、大きな図形を描いたとき、ギザギザが気になるかもしれません。その場合は、2分の1のサイズで作成し、その後にpygame.transform.scale2x()で倍サイズにします。すると、アンチエイリアスが効いた状態になります（ただし、scale2xはパーツではなく画面を2倍するため、図形用の画面を作る必要があります）。

3 . 画像表示をクラスで置き換える

2 . で作成したものをclassで置き換えて、メンテナンスしやすく改造します。

```
# -*- coding:sjis -*-
import pygame
from pygame.locals import *

class Player:
    def __init__(self, scr): # クラスの初期化
        self.rect = pygame.Rect(0, 0, 40, 10) # playerのサイズ
        self.color = (0, 255, 128) # playerの色
        self.rect.center = (160, 200) # playerのxy座標
        self.scr = scr # 画面の関連付け

    def DrawImage(self): # 画面に絵を表示
        pygame.draw.rect(self.scr, self.color, self.rect) # 四角を描画

    def SetPos(self, x, y): # 自キャラの x,y座標を指定する
        self.rect.topleft = (x,y)

class Ball:
    def __init__(self, scr): # クラスの初期化
        self.rect = pygame.Rect(0, 0, 10, 10) # ballのサイズ
        self.color = (255, 128, 0) # ballの色
        self.rect.center = (160, 100) # playerのxy座標
        self.scr = scr # 画面の関連付け

    def DrawImage(self): # 画面に絵を表示
        pygame.draw.ellipse(self.scr, self.color, self.rect) # 内接円を描画

    def SetPos(self, x, y):
        self.rect.topleft = (x,y)
```

```

def main():
    pygame.init() # pygameの初期化
    screen = pygame.display.set_mode( (320, 240) ) # 画面を作る
    pygame.display.set_caption('Class Replacement') # タイトル

    player = Player(screen)
    ball = Ball(screen)

    while 1:
        player.DrawImage()
        ball.DrawImage()

        pygame.display.flip() # 画面を反映
        for event in pygame.event.get(): # イベントチェック
            if event.type == QUIT: # 終了が押された?
                return
            if (event.type == KEYDOWN and
                event.key == K_ESCAPE): # ESCが押された?
                return

if __name__ == '__main__': main()
# end of file

```

クラスについて

classはオブジェクト指向ではおなじみの考え方で、Pythonでももちろん実装されています。コンストラクタはdef __init__(self):で行います。

Pythonは変数を使うのに宣言が不要のため、C++のようにそのまま書くことはできません。

```

class X {
public:
    int data;
    void func(int a){ data = a; }
}

```

```

class X:
    data = 0;
    def func(self, a):
        self.data = a

```

data とただ単に書くと、新たにdata変数を作ってしまう。そのため、メンバ変数であることを明示するため、this ポインタと同じ意味で、selfを使います。上記例文は、左右どちらも同じ意味となります。

4 . 画像を動かす

```
# -*- coding:sjis -*-
import pygame
from pygame.locals import *

class Player:
    def __init__(self, scr): # クラスの初期化
        self.rect = pygame.Rect(0, 0, 40, 10) # playerのサイズ
        self.color = (0, 255, 128) # playerの色
        self.rect.center = (160, 200) # playerのxy座標
        self.scr = scr # 画面の関連付け

    def DrawImage(self): # 画面に絵を表示
        pygame.draw.rect(self.scr, self.color, self.rect) # 四角を描画

    def SetPos(self, x, y): # 自キャラの x,y座標を指定する
        self.rect.topleft = (x,y)

    def MoveKeyin(self): # キー入力で移動させる
        keyin = pygame.key.get_pressed()
        mx = 0
        if keyin[K_LEFT]: mx = -2
        if keyin[K_RIGHT]: mx = 2
        # 画面の枠内に、自分の移動先の四角が完全に
        # 含まれているなら、移動させる。
        if self.scr.get_rect().contains(self.rect.move(mx,0)):
            self.rect.move_ip(mx,0)
```

キー入力の取得

pygame.key.get_pressed()を呼ぶと、キーボードの配列を返します。現在押されているキーには1が入っており、ifで確認した場合のTRUEになります。そのため、どのキーが押されているかをチェックすることができます。

```

class Ball:
    def __init__(self, scr): # クラスの初期化
        self.rect = pygame.Rect(0, 0, 10, 10) # ballのサイズ
        self.color = (255, 128, 0) # ballの色
        self.rect.center = (160, 100) # playerのxy座標
        self.scr = scr # 画面の関連付け
        self.scrRect = scr.get_rect() # スクリーンの大きさ
        self.vx = 2 # x方向の移動速度
        self.vy = -2 # y方向の移動速度

    def DrawImage(self): # 画面に絵を表示
        pygame.draw.ellipse(self.scr, self.color, self.rect) # 内接円を描画

    def SetPos(self, x, y):
        self.rect.topleft = (x,y)

    def Mvve(self):
        mr = self.rect.move(self.vx, self.vy) # 移動先のrectを取得
        if self.scrRect.right < mr.right or self.scrRect.left > mr.left:
            self.vx = -self.vx # x方向の移動を反転させる
        if self.scrRect.bottom < mr.bottom or self.scrRect.top > mr.top:
            self.vy = -self.vy # y方向の移動を反転させる
        self.rect.move_ip(self.vx, self.vy)

```

画面端の辺り判定の調べ方

通常であれば、画像の左端が、画面の左端に着いたら、というような条件でチェックしますが、Pygameでは当たり判定を取る為のメソッドが用意されています。

```

Rect rectA.contains(rectB) # rectAにrectBが完全に含まれていればTrue
Rect rectA.colliderect(rectB) # rectAにrectBが1ドットでも重なっていればTrue
Rect rectA.collidepoint(x,y) # (x,y)の座標がrectAに重なっていればTrue

```

他にもいくつかありますが、大体は上記でまかなえます。ただし、containsやcollideは、どの方向が、という情報はくれませんので、方向に依存する当たり判定の場合、それにロジックを組む必要があります。

```

def main():
    pygame.init() # pygameの初期化
    screen = pygame.display.set_mode( (320, 240) ) # 画面を作る
    pygame.display.set_caption('Class Replacement') # タイトル

    player = Player(screen)
    ball = Ball(screen)

    clock = pygame.time.Clock()

    while 1:
        clock.tick(60)
        player.DrawImage()
        ball.DrawImage()

        pygame.display.flip() # 画面を反映

        screen.fill((0,0,0)) # 全画面を黒で塗りつぶす
        player.MoveKeyin()
        ball.Move()
        for event in pygame.event.get(): # イベントチェック
            if event.type == QUIT: # 終了が押された?
                return
            if (event.type == KEYDOWN and
                event.key == K_ESCAPE): # ESCが押された?
                return

```

画面の更新

画面に絵を描いたら、次の絵を描く前にクリアしなければ、前の絵が残ればなしになります。上記プログラムでは、`screen.fill((0,0,0))`として、全画面を塗りつぶしていますが、`screen.fill(color, rect)`と指定すれば、`rect`の場所だけ塗りつぶすことができます。

例題 : `Player`, `Ball` にある `Move` 関数を改造して、自分の前の位置だけ塗りつぶすようにしてみよう。

5 . 表示を制御する

5 . 1 玉を増やす

Ball クラスへの追加 と、main関数を次のもの に書き換えてください。

```
class Ball:
    def __init__(self, scr, vx, vy): # クラスの初期化
        self.rect = pygame.Rect(0, 0, 10, 10) # ballのサイズ
        self.color = (255, 128, 0) # ballの色
        self.rect.center = (160, 100) # playerのxy座標
        self.scr = scr # 画面の関連付け
        self.scrRect = scr.get_rect() # スクリーンの大きさ
        self.vx = vx # x方向の移動速度
        self.vy = vy # y方向の移動速度
    ...

    def Refract(self, pIRect):
        if self.rect.colliderect(pIRect): # pIRectと自分がぶつかれば
            self.vy = -self.vy

def main():
    ...
    player = Player(screen)
    balls = []
    for i in range(5):
        balls.append(Ball(screen))
    ...
    while 1:
        clock.tick(60)
        player.MoveKeyin()
        for i in range(5):
            balls[i].Move()
            balls[i].Refract(player.rect)
            balls[i].DrawImage()
        player.DrawImage()

        pygame.display.flip() # 画面を反映
    ...
```

5.2 玉を減らす

Main の追加した for を次のものに置き換えてください。

```
class Ball:
    def __init__(self, scr, vx, vy): # クラスの初期化
    ...
        self.refcnt = 0 # 当たった回数
    ...
    def Refract(self, p1Rect):
        if self.rect.colliderect(p1Rect): # p1Rectと自分がぶつかれば
            self.vy = -self.vy
            self.refcnt = self.refcnt + 1
        return self.refcnt
    ...
        for i in range(len(balls)):
    ...
            if balls[i].Refract(player.rect) > 5: balls.pop(i);break
```

リストの追加と削除

リストには、append() を使って追加することができます。また、その要素を消去するときは、pop() を使えば消すことができます（正確には、pop は push, pop の pop ですの、その値を取り出して捨てる、が正しい使い方です）。

6. 終わりに

非常に駆け足で、あまり突っ込んだ解説もせずに動作の紹介をしましたが、いかがでしたでしょうか。Pythonは、「電池が付属しています（"Battery Included"）」の発想で、使えるライブラリがかなり付属していますが、さらにライブラリを追加することで色々なことを、比較的簡単なコーディングで行うことができます。

今回は触れていませんが、Pythonをインストールしていない人のために、バイナリで配布するためのライブラリもありますし、P2Pを動かすためのフレームワークも開発されています。また、Windowsであれば、IronPythonを使うことで、.NET Frameworkとも連携がとれ、GUI付のソフトを簡単に作ることもできます。

Pygameは、ワンライナーから大規模ソフトまで、あらゆる場面で活躍できるPythonの可能性のひとつです。Pygameだからといって、ゲーム作りにしか使えない、と考えるのはいけません。グラフを描いたり、簡単なGUIとして使ったり、可能性は無限大です。すべては発想次第です。自由なプログラミングを楽しみましょう。