Flappy Cappy

A PYTHON GAME

CATFORD CODING CLUB

Table of Contents

Catford Coding Club

About the game

- We're using the P5 library in Python to provide a graphical environment.
- The game is inspired by Flappy Bird, but it's better, because it has a capybara.
- Capybaras are native to South America, but ours lives mostly in Catford.
- That's why, when the cold of winter bites, she flies south in search of warmth.
- Our code will be broken into functions to keep it clean. P5 has 2 key inbuilt functions.
- First is setup(), where we can load in resources and set up variables etc.
- Second is draw(), and this runs over and over, 60 times a second, providing our 'game loop'
- So log in to https://strivemath.org/ide and let's get started...

Building the background

Go to strivemath.org/ide and open a new project. You'll see that the setup() and draw() functions are already there for you. But before we use them, we're going to import a library we need, and set up a few variables.

Variable names are case-sensitive. Variables that don't vary are called constants, and it's a good idea to type these in capitals, so you know not to change them.

Make some new lines **above** the `setup()` function, and add this code.

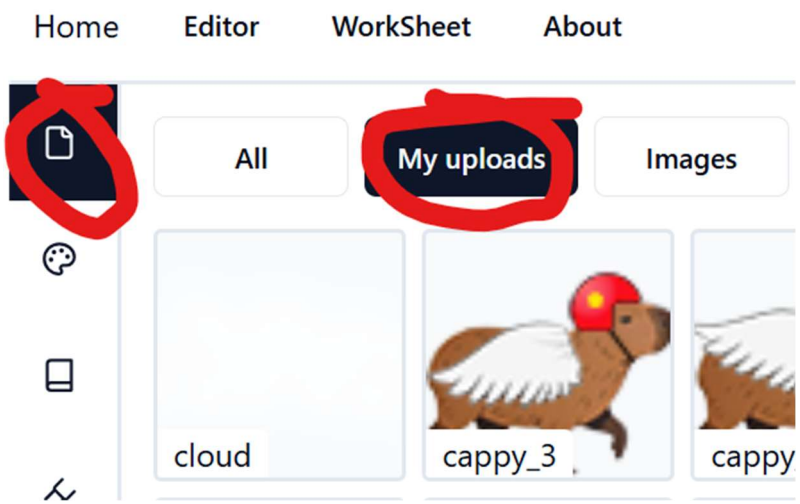
Declaring variables here makes them slightly easier to access. If we do it in the setup() function, we also have to explicitly make them global there.

```
# Import libraries
import random

# Set constants
SCREEN = [600, 400]
CAPPY_START = [50, 200]
CAPPY_W = CAPPY_H = 64
CLOUD_SPEED = 0.3

level = 1
score = 0
lives = 3
frame_count = 0
```

I've uploaded some graphics to your account – click the icon on the top-left of your page to see them. Hover over 'cloud' and it will show the word 'copy'. Click, and it will copy the address of the image to your clipboard. Each of you will have a slightly different address, so don't copy someone else's!



Now, beneath the variables, type:
`assets =`
then hit ctrl-v to paste the address. You'll notice it is loooong!

That's why we're assigning it to a variable. There's not much room in the text editor, so it's better if we can type 'asset' each time, rather than fill it with long URLs.

Now delete the actual filename – the bit after the final slash (/). So it looks like this.

```
assets =
"https://ezpwmmfaodhpebjbmury.supabase.co/storage/v1/object/public/assets/d9ea3bf8-b645-47a5-b0e8-4fc65f7df8c0/"
```

Now, in your `setup()` function, we'll set the screen dimensions (using the SCREEN constant we set up earlier) and the frame rate.

We'll declare a couple of global variables that we'll use to draw the clouds...

...and then we'll populate them. `cloudpic` is the image object, and we 'load' our image into it using the URL we copied earlier. Rather than pasting the whole long folder path, we'll ***concatenate*** the variable that holds it (`asset`) with the filename (`cloud.png`)

`cloudlist` is a list of all the clouds on our screen. Each of those 'clouds' is itself a list, containing the x and y coordinates of each cloud (note the double square brackets!).

To begin with, we're adding a single cloud to this list, with random coordinates.

```
def setup():
  createCanvas(SCREEN[0], SCREEN[1])
  frameRate(60)


# Declare and set globals
  global cloudpic, cloudlist

# Add the first cloud to the cloudlist array
  cloudpic = loadImage(assets+"cloud.png")
  cloudlist = [[random.randint(0, 600), random.randint(100, 400)]]
```

Moving to the `draw()` function, let's render those clouds. First, I'm going to add some globals – we don't need them just yet, but we will soon, and since we're here at the start...

Next we fill in the background. We do this every frame, to delete the previous frame's images.

Now we'll roll a 300-sided dice to decide whether to add (or 'append') a new cloud to our cloudlist.

Finally, loop through every item in cloudlist and use image() to draw it. The first parameter is the image we defined in setup(), then cloud[0] and cloud[1] are the x and y coordinates, stored in cloudlist.

Once each cloud is drawn, we move it to the left by subtracting CLOUD_SPEED from the x value, then we check whether it will be off the screen: if it is, we remove it from the list.

```
def draw():
 global cappy_index, frame_count, x, y, x_speed, y_speed

 # Draw background and clouds
 background("#99ddff")


 # A 1 in 300 chance to add another cloud to the list
 chance = random.randint(1,300)
 if chance == 1:
  cloudlist.append ([500, random.randint(100, 380)])

 for cloud in cloudlist:
    image(cloudpic, cloud[0], cloud[1])
    cloud[0]-= CLOUD_SPEED
    if cloud[0] < 0-cloudpic.width:
     cloudlist.remove(cloud)
```

Adding a capybara

Because we like to make life difficult, and a Flappy Cappy really ought to flap, we're going to animate the capybara. To do this, we'll make a list of loaded images, each with the wings in a different position, and cycle through them every few frames.

At the top of your code, where you set up those first variables, add another 2: cappy[] (the brackets signify that it is a list), and cappy_index, to keep track of which animation frame we're on.

```
cappy = []
cappy_index = 0
```

In the `setup()` function, use a for loop to add 4 images to the cappy list. Can you see how the number of the loop is used to specify the file name?

```
# Load the image frames that make the Cappy animation
for i in range(4):
    cappy.append(loadImage(assets + "cappy_"+str(i)+".png"))
```

We'll use x and y as variables to store the position of Cappy. Set these to the first and second items in the CAPPY_START list.

```
x = CAPPY_START[0]
y = CAPPY_START[1]
```

Add x and y to your setup's global declaration.

```
global cloudpic, cloudlist, x, y
```

In the `draw()` function, below your cloud code add this. Be careful to get your indentation right!

The first line draws cappy, specifically the image loaded for cappy[0], at coordinates x, y

Next, we add one to the frame count, keeping track of how many times the game loop has run.

The final bit states that every 6 loops Cappy should should move on to the next image frame.

```
# Draw Cappy
image(cappy[cappy_index], x, y)

# Advance frame every 6 draw calls (~10 fps at 60 fps)
frame_count += 1
if frame_count % 6 == 0:
    cappy_index = (cappy_index + 1) % len(cappy)
```

The last 2 lines use a mathematical operator you may not be familiar with – modulus, which is inexplicably designated by a % sign. Modulus just means remainder. So the condition reads, “if the remainder is 0 when dividing frame_count by 6…” In other words, when frame_count equals 6, 12, 18, 24… etc.

The action to move the frame forward uses this cleverly: because our frames go from 0 – 3, when we pass 3 we have to loop back to 0. Instead of adding 1 each time and checking whether we’ve reached 3, we set the index to the modulus of itself +1 divided by the total (4). So, 1 % 4 = 1; 2 % 4 = 2; 3 % 4 = 3; But 4 % 4 = 0, providing an automatic reset back to the first frame

Gravity and movement

Add another constant at the top of the code. This will decide the strength of gravity. Also add variables for `x_speed` (how fast the screen scrolls) and `y_speed` (how fast Cappy falls).

```
GRAVITY = 0.1

y_speed = 0
x_speed = 1
```

At the bottom of the `draw()` function, add these lines to increase `y_speed` due to gravity. The conditions are to stop her disappearing off the canvas, but if she's neither at the top or the bottom already, her y-coordinate is decreased to move her downwards in the next frame.

```
# Adjust y_speed
y_speed = y_speed + GRAVITY
if (y < 20 ):
  y_speed = 0
elif y > 350:
  y_speed = 1
  y -= y_speed
else:
  y -= y_speed
```

But this causes an error! Why? Well, we're allowed to *read* the y value in our function, but we're not allowed to *change* it. To change it, we have to make it global. So...

Add x, y, x_speed and y_speed to the global line. Run the code, and check that Cappy falls.

```
global cappy_index, frame_count, x, y, x_speed, y_speed
```

Of course she needs to fly, as well as falling. There's a built-in function, for this, `mousePressed()`
 	
Put this at the bottom of your code. Because the draw function doesn't change y when it's below a certain value, we'll give her a little jump if she's at the bottom of the screen.
		`def mousePressed():`
		`  global y_speed, y`
		`  if y<20:`
		`  y += 10`
		`  y_speed -= 3`