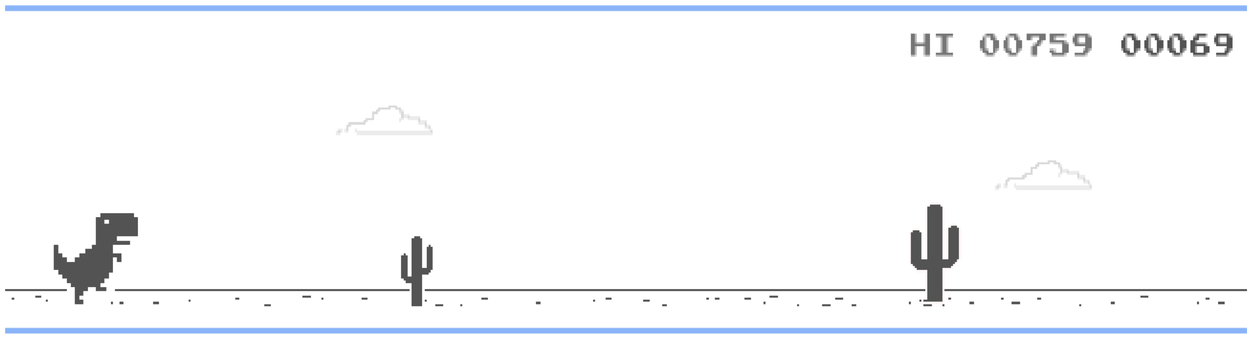
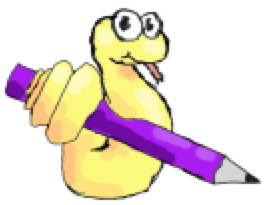
Endless Runner Game

Workbook

Programming Tutorials by PyAngelo



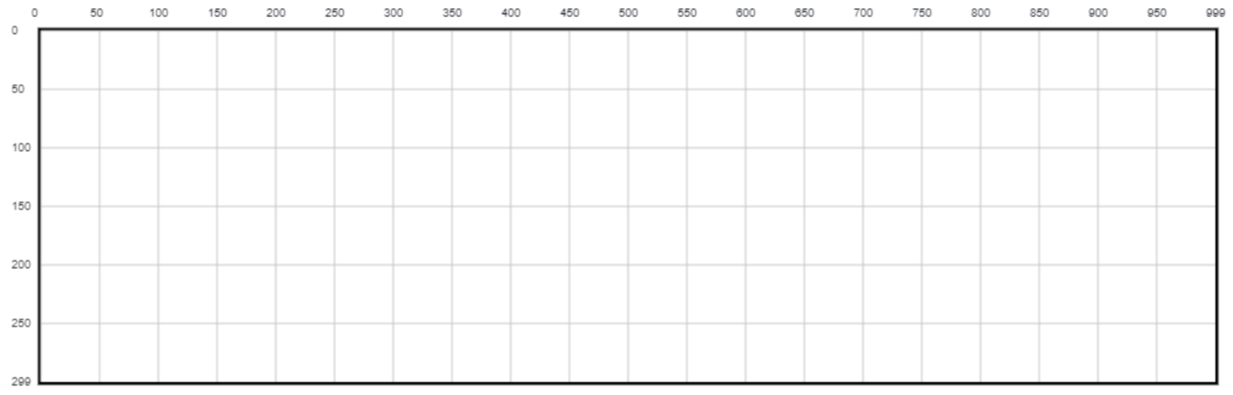


Name:___________________

Task 1 - Infinite Scrolling

Sketch Your Distant Background

We want the distant background to scroll forever to give the impression of movement. Before we figure out how, let's sketch what our background will look like using the grid below. The image should be 1000 pixels wide and 300 pixels high.



Duplicate Your Image

To make it appear as though the background is scrolling infinitely, we need to duplicate it, and attach it to the right hand edge so that the image is split into two components that are exactly the same. This will make our background image 2000 pixels wide and still 300 pixels high. Use an image program to create your distant background image and export it as a png file with the name "distant_background.png".

Animating the Background

To animate the background you will need to understand:

- How animation works for computers
- Infinite loops
- If statements
- The PyAngelo Sprite Library

Animation

Animation works by looping through the following actions in rapid succession:

- Clearing the screen
- Drawing your images on the screen
- Moving the position of some images in memory

Infinite Loops

In Python, you can create an infinite loop using a “while True:” loop. The block of code that is indented below the “while True:” will be repeated forever.

If Statements

If statements allow you to execute a block of code only if a certain condition is met. We will use an if statement to check if our object has passed the bottom of the canvas and if so we’ll respawn it at the top. Example if statement:

```
if distant_background.x < -1000:
        distant_background.moveTo(0, 0)
        #The indented lines are only executed
        # if distant_background.x has a value less than -1000
```

PyAngelo Sprite Library

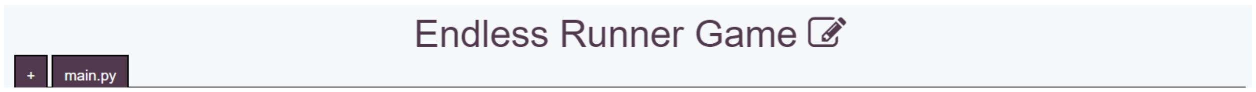
To include the PyAngelo sprite library in your sketch use the following code at the top of your program:

```
from sprite import *
```

Infinite Scrolling Code

You need to move your image to the left each time through the animation loop until it’s x position reaches minus (-) 1000. At this point you need to move the image back to it’s starting position and then be able to continue scrolling in this manner forever! Use the space below to write out the Python code to achieve this.

Scrolling Background - Write Down Your Code



Check Your Knowledge

1. The x-axis moves ______________________________ the screen.

The y-axis moves ______________________________ the screen.

2. What does the following command do?

```
from sprite import *
```

____

3. What does the following command do?

```
setCanvasSize(1000, 300)
```

____

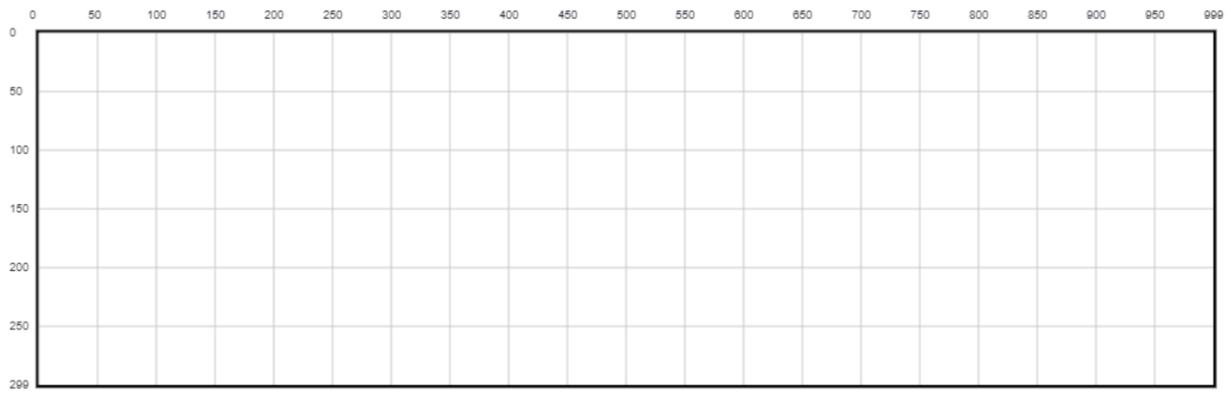
4. What does the following code do?

```
distant_background.draw()

distant_background.moveBy(-5, 0)

if distant_background.x <= -1000:

        distant_background.moveTo(0, 0)
```

Parallax Scrolling

Parallax scrolling is a technique where background images scroll across the screen at a slower pace than foreground images. This gives the impression of distance in a 2D world. To implement this you need to draw the foreground image using the same technique as for the distant background.

Firstly, sketch your foreground image below:



Now create your image in a graphics program ensuring you have:

- The canvas 1000 pixels wide and 300 pixels high
- A transparent background - so we can see the distant background behind it

Then as we did for the distant background we need to duplicate this image and attach it to the right edge to end up with a single image that is 2000 pixels wide, 300 pixels high, and is essentially two copies of the same image.

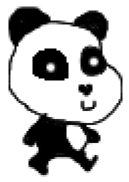
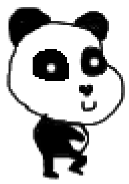
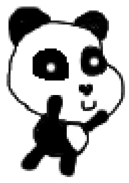
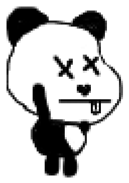
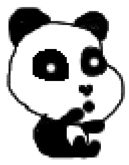
Lastly, you can now code this into your game by:

- Creating a Sprite called ground using your image
- Draw the ground by calling ground.draw()
- Move your Sprite calling ground.moveBy() - this should move at a faster speed than the distant background
- Check if the ground's x position is less than -1000 and if so move it back to its starting position.

Task 2 - Animate our Hero

Make our hero look like they are walking

So far we have animated shapes and images simply by moving them. However, to make our hero look like they are moving, we need multiple images. Remember that our main character will collide with our enemies so it is important to fill as much of the space as possible. Here are some examples:

	**Walking Image 1** If we switch between this image and the "walking Image 2", it will make our hero look like they are walking.
	**Walking Image 2** There doesn't need to be a big change to make it look like our character is walking.
	**Jumping** When our hero is jumping we can use a different image which lets the user know they have pressed the jump key and adds to the visual effect of our game.
	**Dead** It's a good idea to have a different image for our hero when they collide with an enemy. As with the jumping image, this gives feedback to the user and adds more visually to our game.
	**Ducking** We will use the ducking image for an extension task.

Sketch Your Hero

Walking 1	
Walking 2	
Jumping	
Dead	
Ducking	

Create Your Images

Now it’s time to use a graphics program to create all of your images. All of your images should be the **same size**. It’s also important to ensure you have a **transparent background**. Also remember that for collision detection the entire image is used, not just where you draw. This means you should take up as much of the image with your drawing as possible.

Coding our Animation

We used a Sprite to create our animated background:

```
distant_background = Sprite("distant_background.png", 0, 0)
```

This creates a sprite that has the the following properties:

- x - the x position of the upper left corner of the image
- y - the y position of the upper left corner of the image
- width - the width of the image
- height - the height of the image

It also has an **image** property. We can use this to change the image that the Sprite displays when it is drawn to the screen using the **draw()** method. So to create an animated Sprite we need to first create the Sprite. Then, to change the image, we first need to load an image and save it to a variable. We do this with the following code:

```
panda = Sprite("pandaWalk1.png", 30, 200)

pandaWalk1 = loadImage("pandaWalk1.png")

pandaWalk2 = loadImage("pandaWalk2.png")
```

Now in our code we can update the image being drawn with the following code:

```
panda.image=pandaWalk1
```

Putting it all Together

We don’t want to switch the image of our hero every frame. Our approach should be to only change the image every 10 frames. We can adjust this number as necessary. To do this we need to count the frames using a variable.

```
frame = 0
while True:
    frame = frame + 1
    if frame % 10 == 0:
        if panda.image == pandaRun1:
            panda.image = pandaRun2
        else:
            panda.image = pandaRun1
```

Frame Rate

Remember those cool little flipbooks where a pad of paper had an image on every page, and when you flipped through the pages quickly, the image would appear to animate and move?

This is how video works. Whether digital or old-school film, video is a series of still images that, when viewed in order at a certain speed, give the appearance of motion. Each of those images is called a “frame.”

Frame rate, then, is the speed at which those images are shown, or how fast you “flip” through the book. It’s usually expressed as “frames per second,” or FPS. So if a video is captured and played back at 24fps, that means each second of video shows 24 distinct still images.

For our animation the frame is how often the hero is switching between images to create the animation that it is running.

Check Your Knowledge

1. What does the symbol % mean?

2. What does the following code do?

```
pandaWalk1 = loadImage("pandaWalk1.png")
```

3. What does the following code do?

```
frame = 0
while True:
    frame = frame + 1
    If frame % 100 == 0:
        print(“Hello”)
```

4. Which of the following are properties of a Sprite?

- ☐ x
- ☐ y
- ☐ width
- ☐ height
- ☐ image

5. How can you change the image of a Sprite?

6. How do you draw a Sprite to the screen if it has been created with the following code?

panda = Sprite(“pandaWalk1.png”, 100, 200)

Task 3- Jumping

We want to now add another animated state being our jumping state. This state should only occur when the hero is jumping (the up arrow is pressed), and the hero should stop the walking animation.

Constant Motion

To make a character move at a constant speed we simply change their position by the same amount each time through the animation loop. For example the statement below would move the panda up the screen by 10 pixels every time it was executed.

```
panda.moveBy(0, -10)
```

This however does not give the hero the illusion that they are jumping. It simply looks as if they are moving up and down the screen. To create the illusion of jumping we need to use gravity.

Implementing Gravity

Gravity pulls us back down to earth faster and faster. This means the amount we move by each time through the loop changes. Hence instead of moving at a constant speed, we need to change the amount we move by over time. We can do that in our animation loop by using a *constant variable* called **GRAVITY**. It is in capitals as it does not change. We also need to create a variable which changes according to how much the hero has moved and therefore the gravity input. Let's call that variable **heroVelocity.**

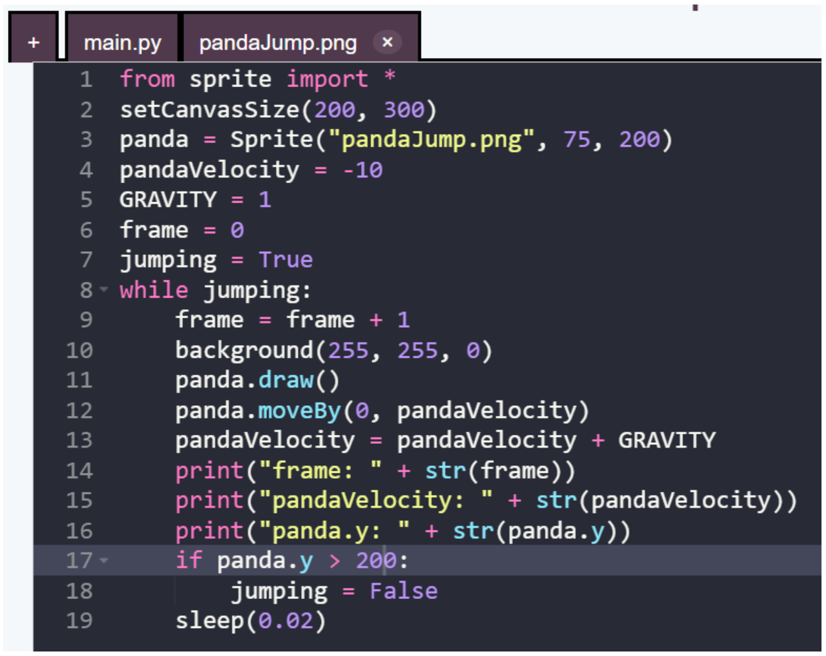
```
heroVelocity = -10

GRAVITY = 1

While True:

        panda.moveBy(0, heroVelocity)

        heroVelocity = heroVelocity + GRAVITY
```

The two bold lines above will need to be executed in our animation loop only when our hero is in the jumping state.

Code Tracing Activity

Read over the code below and for each iteration of the while loop, fill in the values of the pandaVelocity and panda.y variables. We've done the frame variable for you and also started the first few loops.

frame	panda.y	pandaVelocity
0	200	-10
1	190	-9
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		



```python
from sprite import *
setCanvasSize(200, 300)
panda = Sprite("pandaJump.png", 75, 200)
pandaVelocity = -10
GRAVITY = 1
frame = 0
jumping = True
while jumping:
    frame = frame + 1
    background(255, 255, 0)
    panda.draw()
    panda.moveBy(0, pandaVelocity)
    pandaVelocity = pandaVelocity + GRAVITY
    print("frame: " + str(frame))
    print("pandaVelocity: " + str(pandaVelocity))
    print("panda.y: " + str(panda.y))
    if panda.y > 200:
        jumping = False
    sleep(0.02)
```

Check Your Knowledge

Please answer the following questions based on the code tracing activity above.

1. What is the final value of panda.y from the above tracing activity?

2. What causes the while loop to end?

3. Why is the variable GRAVITY written in uppercase?

3. What does the following line of code do:

panda.moveBy(0, pandaVelocity)

4. What does the following line of code achieve?

pandaVelocity = pandaVelocity + GRAVITY?

5. What would happen if we changed line 4 to:

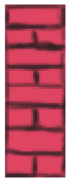
pandaVelocity = -30

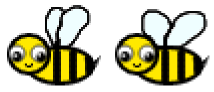
6. What would happen if we changed line 5 to:

GRAVITY = 2

Task 4 - Enemies

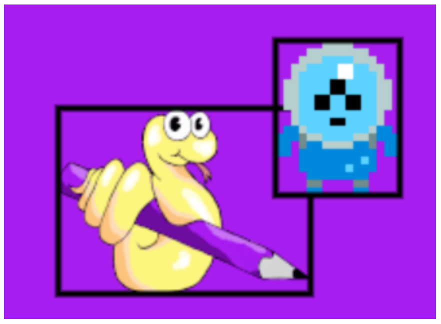
Depending on the obstacle you create, you may or may not need animation.

 As this wall does not change we only need a single image

 For an animal such as this bee it would be good to create multiple images that we can animate. In this case we change the bees wings to make it look like it is flying when we switch between the two.

Sketch Your Enemies or Obstacles

Remember that the Sprite library performs collision detection by checking the bounding boxes of the two sprites like so:



Hence, even if it does not look like two Sprites are touching, it may trigger a collision. This means when you draw your images you should try to ensure they take up as much of the image canvas as possible so your collision detection will be more realistic.

Adding the Enemies to Your Code

To add your enemies to your code you need to:

- Create the image or images using a graphics program
- Upload your image to the PyAngelo website
- Update your code to:
    - Create a Sprite using your image
    - Draw you sprite using the draw() method
    - Move your Sprite with the moveBy() method
    - Check if your hero overlaps the enemy with the overlaps() method
    - Check if the enemy has gone off the left edge of the canvas and if so respawn it off the right edge of the canvas.

Check Your Knowledge

1. What parameters are passed to the overlaps method?

---

2. How does the overlaps() method detect if there is a collision?

---

3. If I have an image of a wall that is 30 pixels wide, what is the maximum x position at which all of the wall can be drawn on the left of the canvas so it can’t be seen?

4. If I want to draw a Sprite off the right edge of my canvas which is 1000 pixels wide and 300 pixels high, what is the minimum x position I can draw it at?

5. Does the draw() method of a Sprite need any parameters? Why or why not?

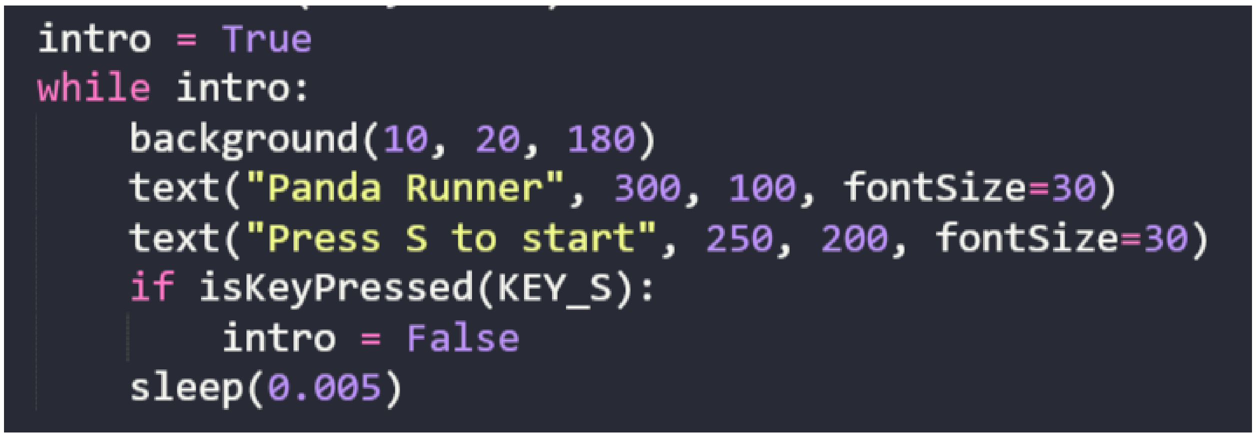
Task 5- Game States, Scoring,and Sound Effects

Game States

We can create a separate animation loop for each of our game states:

- Introduction screen
- Play State
- Game over screen

Here’s sample code for the introduction screen:



```
intro = True
while intro:
    background(10, 20, 180)
    text("Panda Runner", 300, 100, fontSize=30)
    text("Press S to start", 250, 200, fontSize=30)
    if isKeyPressed(KEY_S):
        intro = False
    sleep(0.005)
```

Notice how the while loop continues until the boolean variable **intro** is set to **False**. This occurs when the S key is pressed. At this point the while loop will end and the program will move onto the next block of code.

If we wish to repeat our game after we have reached the game over screen, we can wrap all of our game states in a while loop that never ends like so:

```
while True:

        <code for intro screen>

        <code for play state>

        <code for game over screen>
```

Sound Effects

To create sound effects you need the following commands:

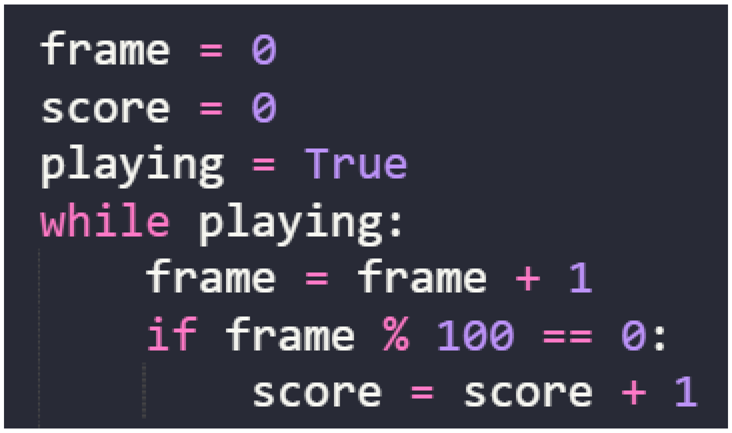
- loadSound() - to load a sound so it can be used by the program
- playSound() - to play the sound

Background Music

For any sounds or music that last for more than a few seconds, you need to play the sounds outside the animation loop. Otherwise, you will be starting that sound multiple times per second.

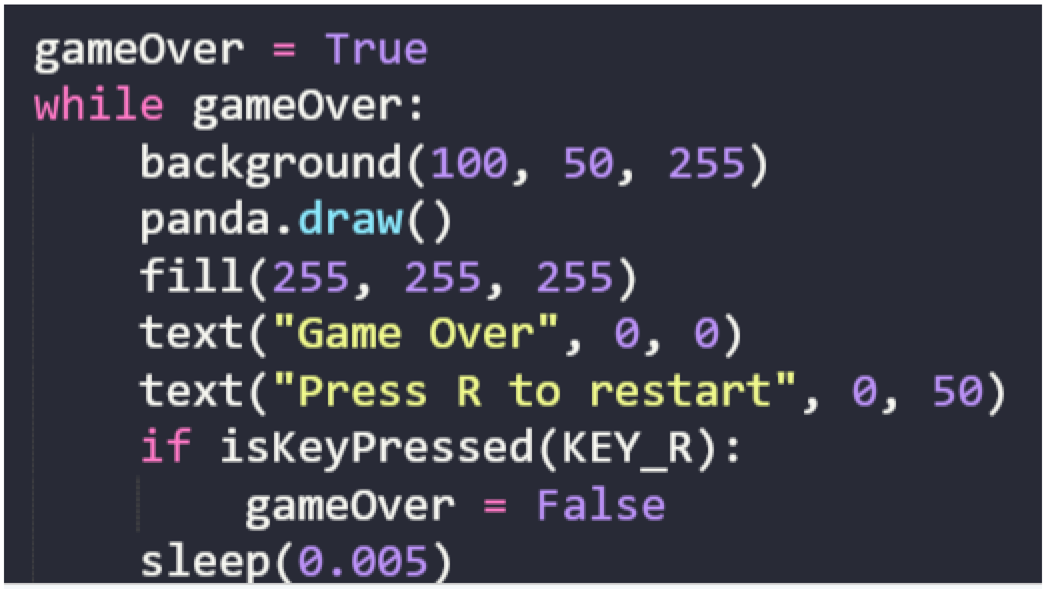
Check Your Knowledge

1. What does the following code do?



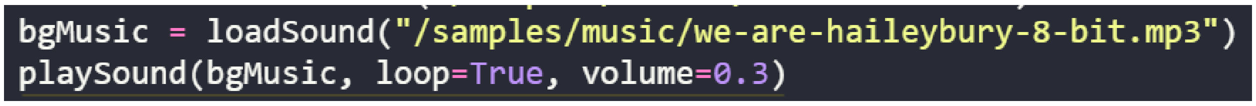
```
frame = 0
score = 0
playing = True
while playing:
    frame = frame + 1
    if frame % 100 == 0:
        score = score + 1
```

2. What causes the gameOver while loop to end in the following code?



```
gameOver = True
while gameOver:
    background(100, 50, 255)
    panda.draw()
    fill(255, 255, 255)
    text("Game Over", 0, 0)
    text("Press R to restart", 0, 50)
    if isKeyPressed(KEY_R):
        gameOver = False
    sleep(0.005)
```

3. What does the following code do?



```
bgMusic = loadSound("/samples/music/we-are-haileybury-8-bit.mp3")
playSound(bgMusic, loop=True, volume=0.3)
```

Checklist

As a minimum your game should:

- ☐ Have two background which scroll
- ☐ Have an animated hero which updates with frames
- ☐ Use gravity and velocity, with hero state for the hero to jump
- ☐ Have an enemy which moves across the screen and respawns
- ☐ Detect collisions between your hero and enemies
- ☐ Have a scoring system
- ☐ Have a start screen and end screen
- ☐ Include music and sound effects

Extension possibilities:

- Allow players to restart the game from the end screen
- Add different types of enemies
- Add more than a single enemy on the screen at a time
- Add the ability to make your hero duck as well as jump
- Increase the speed of the game when you reach a certain score
- Add lives rather than instant death
- Add a high score feature (this can't be saved. It is just while the game is running. You must be able to restart your game within the game for this to work)
- Add pickups (e.g. score bonus, extra life, invincibility hero state for a set time, speed up or slow down)
- Add nighttime and daytime modes
- Create a help screen which provides people with instructions for your game