Digital Technology

PyGame Zero

Level 1 Code and Challenges

Version 2
2021

Barry Butler
bbutl58@eq.edu.au

ROBOCOAST
Sunshine Coast Robotics
www.robocoast.tech

Coolum
State High School
Care • Respect • Excellence

Content and Challenges

Section	Content	Challenges
A	PyGame Drawing	
B	PyGame Variables and Movement	
C	PyGame Loops	1. Noughts and Crosses Layout a. Use the loops to draw a noughts and crosses grid b. Fill with noughts and crosses c. Write the game name and score on the top of the window
D	PyGame Conditionals	2. Alien and Rocks Game a. Make 1 alien (you can use any sprites instead of aliens and rocks you would like to) b. Make 4 rocks and put them in different places on the screen c. Set the game score to 0 and write the name game and score on the top window d. Move the alien randomly e. Test for alien collisions with the rocks. When a collision occurs, disappear the rock and increase the score. f. When the game score is 4 (all rocks gone), write a 'You Win' message on the screen
E	PyGame Lists	3. List of Favourite Things a. Create three lists: • a list of your five favourite things (call it things_list) • a list of five x values (call it x_list) • a list of five y values (call it y_list) b. Display your favourite things on the screen at the x,y positions 4. Hangman Stick Figure a. Design hangman stick figure on grid paper b. Make a list of lines and circles, with their coordinates and radius c. Use the list to draw the hangman on the pyGame screen
F	PyGame Mouse and Keyboard Events	5. Quiz a. Write down 2 questions with 4 possible answers each (one is correct) b. Create a quiz with these questions on separate screens. c. Click the mouse or use the keyboard to answer the quiz questions. If the answer is correct, display a message.
G	PyGame Functions	6. Simple Matching Cards game
H	Other Useful Code	

A. PyGame Drawing

Using the Mu Editor

Click on the mode button (top left) and change the mode to PyGame Zero.

All code files must be saved in the mu_code folder.

All images must be saved in the mu_code\images folder.

Libraries

A Python library is a collection of functions and methods that allows you to perform lots of actions without writing your own code.

The libraries used in a program are declared at the start of the program, e.g.

```
import random
```

This library is used to generate random numbers. We import all the functions from that library into our program. Type this on the top line of a new program file.

PyGame Zero Basic Code

All programs require some basic code to put a window on the screen, and draw in that window. For PyGame Zero it is:

```
HEIGHT = 600            #screen dimensions
WIDTH = 800

def draw():
    pass
```

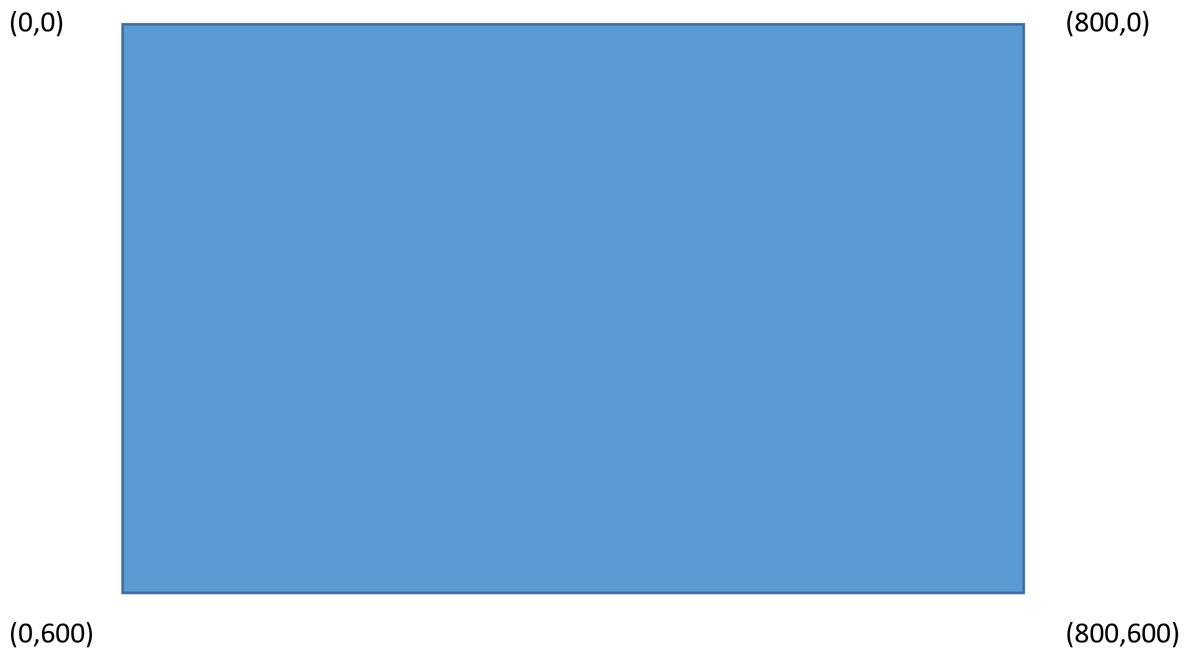
- The screen height and width are **constants** – they do not change.
- The code *def draw():* is a **function** – lines of code that perform a specific task. Notice how code is indented within the function.

Comments

Note the comment – starting with #. Multi-line comments start and end with """.

PyGame Zero Window Coordinates

The top left corner of the window is the origin (0,0). The horizontal coordinate (x) is zero, and the vertical coordinate is zero.



Define Colors

Colors are important to everything we do. In PyGame Zero, it is easiest to declare the colors we use at the start of the program:

```
RED = (255,0,0)         #set color constants
BLACK = (0,0,0)
LBLUE = (0,200,250)
YELLOW = (255,216,0)
```

- Colors are defined by how much red, green and blue make up the color.
- All **constants** are written in uppercase (case matters in Python). Put these under the screen height/width.
- The numbers are written as a **tuple** (in round brackets, separated by a comma).

Google ‘Color Picker’ to get the rgb numbers for any colour

Set the Window Color

In PyGame Zero, the draw() function is called 60 times every second to redraw the window. Every time the window is redrawn we must clear everything off it and fill it with a color.

```
def draw():
    screen.clear()          #clear screen
    screen.fill(LBLUE)      #fill with a color
```

Draw Lines

Look on your PyGame Zero Cheat Sheet and find the code for drawing a line. This code is placed in the *draw()* function.

```
def draw():
    . . .                                        #keep existing lines of code
    screen.draw.line((0,0), (750,550), BLACK)
```

The line drawing function has three **parameters** (bits of information it needs to draw the line), separated by **commas**.

- Parameter 1: *(0,0)* - the (x,y) coordinates of the start point
- Parameter 2: *(750,550)* - the (x,y) coordinates of the end point
- Parameter 3: *BLACK* - the drawing color

Add more instructions to draw lines anywhere on the screen.

Draw Circles

Look on your PyGame Zero Cheat Sheet and find the code for drawing filled and unfilled circles.

```
def draw():
    . . .
    screen.draw.circle((300,200), 100, YELLOW)
    screen.draw.filled_circle((600,450), 50, RED)
```

The line drawing function has three **parameters** (bits of information it needs to draw the line), separated by **commas**.

- Parameter 1: *(300,200)* - the (x,y) coordinates of the center
- Parameter 2: *100* – the circle radius
- Parameter 3: *RED* - the drawing color

Add more instructions to draw circles anywhere on the screen

Draw Rectangles

Look on your PyGame Zero Cheat Sheet and find the code for drawing filled and unfilled rectangles. First, define the size of the rectangle, then call the function to draw the rectangle.

```
def draw():
    . . .
    screen.draw.rect(Rect((20,100),(360,400)), RED)
    screen.draw.filled_rect(Rect((0,400),(700,100)), YELLOW)
```

A rectangle is defined by the coordinate of the top left (20,100) and a tuple containing the width and height (360,400).

Add more instructions to draw rectangles anywhere on the screen

Draw Text

Look on your PyGame Zero Cheat Sheet and find the code for drawing text.

```
def draw():
    . . .
    screen.draw.text("Coding is Cool!", (350,30),
                     color=BLACK, fontsize=60)
```

Add more instructions to draw text anywhere on the screen.

Draw Images

Look on your PyGame Zero Cheat Sheet and find the code for drawing images. To draw images we create them as an Actor.

```
alien = Actor('alien', center = (400,200))

def draw():
    . . .
    alien.draw()
```

Look in the Images folder for more images and add more instructions to draw these anywhere on the screen.

The names of images must only be typed in lower case and placed in single or double quotes.

Random Numbers

Most games use random numbers so that things don’t occur in the same way all the time.

To generate random numbers, a library containing the random number functions must be imported.

```
import   random               #where is this statement put?
```

To generate a random number use the function

```
random.randint(1,40)          #you change the min. and max. numbers
```

How could you use random numbers? (Try)

B. PyGame Variables and Movement

Variables enable us to store values (e.g. text and numbers).

The value of a variable can change.

They must have a descriptive name so you can recognise them.

The variable is always assigned with the equal sign, followed by the value of the variable. Use descriptive names.

Variable Naming Conventions

- Lowercase words separated by underscores
- Begin with a letter of the alphabet

Try this code:

```
low_number = 12
high_number = 14
total = low_number + high_number
print(total)                        #the print function outputs the result
```

Change to the code use decimal numbers

Change the code to use text (i = ‘Coolum’, j = ‘SHS’)

Types of Data

Variables are given a data type when they are assigned (e.g. i = 1).

There are four basic data types:

• **Integer**	e.g. 1, -120, 3000
• **Floating point** (Decimal)	e.g. 1.45, -12.564, 4000.1
• **String** (Text)	e.g. “First name”, ‘OK’
• **Boolean**	True, False (with capital T and F)

Different data types cannot be joined

Try: i = 100 + “OK” (print the result)

Adding numbers and strings is different.

Try: i = 5 + 6 and i= ’5’ + ’6’

Moving Lines

Instead of using a fixed value in a function we can use a variable. Then we can change the value of the variable to create movement. We declare the variable before the *draw()* function.

```
start_x = 20

def draw():
    . . .
    screen.draw.line((start_x,20), (400,550), BLACK)

def update():
    global start_x
    start_x += 1       #add 1 to the value of line_x
```

Add another variables for the start and end coordinates of the line and change them too (note: -= subtracts a value).

Global vs Local Variables

All variables are **local** to a function by default. That is, they only exist inside a specific function and cannot be used in other functions.

The variable *start_x* is created outside a function, and can be used in all functions. Therefore, it is a **global** variable.

To change a global variable we must put the statement in the first line of the function code.

```
global start_x
```

Moving Circles

To move a circle let's use three variables, for the x value, y value and the radius.

```
circle_x = 600
circle_y = 450
circle_radius = 50

def draw():
    . . .
    screen.draw.filled_circle((circle_x,circle_y),
                               circle_radius, RED)

def update():
    global circle_x, circle_y, circle_radius
    circle_x -= 1
    circle_y -= 1
    circle_radius += 1
```

Moving Images

Actors and their images are **objects** which have variables built into them (called **properties**). Images have x and y properties that can be changed.

```
alien = Actor('alien', center = (10,10))

def draw():
    . . .
    alien.draw()

def update():
    alien.x += 2
    alien.y += 1
```

Because Actors are objects they don’t need listing as global variables.

Add more Actors and move them.

C. PyGame Loops

For Loop

Loops provide an easy way of repeating a series of steps, without duplicating the code, for example drawing a series of circles:

```
def draw():
    for i in range(10):
        screen.draw.circle((400,300), 10+i*10, RED)
        print(i, 10+i*10)
```

The circle will be drawn 10 times. The variable `i` is the loop counter. Check the output from the print statement in Mu.

- What value does the loop start at?
- Try changing any of the values and see what happens.

For Loop with Calculation Variables

It is easier to separate the calculation of a value from the loop itself. This is done by setting the initial value of a variable, then changing the value with each iteration of the loop.

```
def draw():
    radius = 10
    for i in range(10):
        screen.draw.circle((400,300), radius, RED)
        radius += 10

    x = 50
    for i in range(20):
        screen.draw.line((x,20), (x,200), BLACK)
        x += 20
```

For Loop with Three Parameters

The For Loop can actually take three parameters – the start value, the end value and the step value.

```
def draw():
    for y in range(40,600,40):
        screen.draw.line((500,y), (750,y), YELLOW)

    for y in range(300,500,30):
        screen.draw.text('Hello There', topleft=(600,y),
                         color=BLACK, fontsize=20)
```

Break and Continue

In a loop, the **break** keyword escapes the loop, regardless of the iteration number. Once break executes, the program will continue to execute after the loop.

The **continue** keyword is used inside a loop to skip the remaining code inside the loop code block and begin the next loop iteration.

While Loop

A **while** loop will repeatedly execute a code block as long as a condition evaluates to True.

The condition of a **while** loop is always checked first before the block of code runs. If the condition is not met initially, then the code block will never run.

```
i = 1
while i < 6:            # This loop will run 5 times
  print(i)
  i = i + 1
```

Challenge

1. Use the loops to draw a noughts and crosses grid, fill with noughts and crosses and write the game name and score on the top of the window.

D. Conditional Statements

Conditional statements are used to make **decisions**, creating different pathways depending on variable values. They have three key words: **if**, **elif** and **else**.

True and False Statements (start new file)

If a conditional statement is true, a set of instructions is executed. If the statement is false, then the instructions are not executed or the instructions in the **else** statement are executed

Example 1 – two choices (if-else)

```
def draw():
    . . .
    for i in range(20):
        if i < 10:                #conditional statement (is i < 10?)
            col = RED             #color if statement is True
        else:
            col = YELLOW          #color if statement is False

        screen.draw.filled_circle((300,300+i*3), 70, col)
```

Example 2 – three choices (if-elif-else)

```
def draw():
    . . .
    x = 50
    for i in range(20):
        if i % 3 == 0: col = BLACK
        elif i % 2 == 1: col = RED
        else: col = YELLOW
        screen.draw.line((x,20), (x,200), col)
        x += 20
```

- The % symbol is a modulus operation – the remainder of division
- To compare values we use **== (or <, >, <=, >=). !=** means not equal to (or use the keyword ***not***)
- To assign a value to a variable we use **=**.

Example 3 – Write text depending on the score

```
score = 0

def draw():
    . . .
    if score >= 100:
        s = 'You Win!'
        c = BLACK
    else:
        s = 'You Will Win Soon!'
        c = RED
    screen.draw.text(s, midtop=(600,10), color=c, fontsize=40)

def update():
    global score
    score += 1
```

Use a Conditional Statement to Show or Hide an Actor

```
alien = Actor('alien', center = (100,100))
alien.visible = True

def draw():
    . . .
    if alien.visible:           #or if alien.visible == True:
        alien.draw()
```

Change the statement to `alien.visible = False`. What happens?

Use Conditional Statements to Limit Movement (start new file)

We often want to limit the movement of an Actor to a portion of the screen (or make sure it does not go outside the screen).

Let's make an alien move on the screen.

```
alien = Actor('alien', center = (100,100))
alien.visible = True
alien.x_speed = 2                  #pixels to move each time
alien.y_speed = 3

def draw():
    screen.clear()
    screen.fill(LBLUE)
    if alien.visible: alien.draw()

def update():
    alien.x += alien.x_speed       #change x and y position
    alien.y += alien.y_speed
```

Now let’s fix the problem of going off the screen. We use a double conditional statement

```
def update():
    alien.x += alien.x_speed
    alien.y += alien.y_speed

    if (alien.x <= 0) or (alien.x >= WIDTH):     #if x off screen
        alien.x_speed = -alien.x_speed           #reverse direction

    if (alien.y <= 0) or (alien.y >= HEIGHT):    #if y off screen
        alien.y_speed = -alien.y_speed
```

or means ***either*** of the statements must be True

and means ***both*** statements must be True

Test for Collisions

A fundamental event in games is collisions between sprites.

```
alien = Actor('alien', center = (100,100))
alien.visible = True
alien.x_speed = 3
alien.y_speed = 2

rock = Actor('rock', center = (400,300))

def draw():
    . . .
    if alien.visible: alien.draw()
    rock.draw()

def update():
    if alien.colliderect(rock):      #if collide with the rock
        alien.visible = False
    . . .                            #alien movement code here
```

Challenge

2. Alien and Rocks Game

    a. Make 1 alien (you can use any sprites instead of aliens and rocks you would like to)
    b. Make 4 rocks and put them in different places on the screen
    c. Set the game score to 0 and write the name game and score on the top window
    d. Move the alien randomly
    e. Test for alien collisions with the rocks. When a collision occurs, disappear the rock and increase the score.
    f. When the game score is 4 (all rocks gone), write a ‘You Win’ message on the screen

E. PyGame Lists

Lists make it easy to store and access a sequence of data. Lists are a **comma-separated** sequence surrounded by **square brackets**, and assigned to a variable. The variable then has the data type **list**. The data in a list can be any type, including mixed types.

To create a list, declare it (use a plural name):

```
numbers = [1, 2, 3, 4, 5]

measurements = [1.2, 3.7, 4.5, 8.7, 10.6]

answers = [False, True, True, False]

my_records = ['Bob', 24, 'Havana Rd', 4567, True, False]
```

Access List Values (start new file)

List values are accessed by indexing the list variable. The first item of the list has an index of [0].

```
names = ['Bob', 'Barney', 'Baz', 'Ben', 'Barb', 'Bridget']

def draw():
    . . .
    s = 'First is ' + names[0]
    screen.draw.text(s, topleft=(100,50), color=RED, fontsize=60)

    s = 'Third is ' + names[2]
    screen.draw.text(s, topleft=(100,150), color=RED, fontsize=60)

    s = 'Last is ' + names[-1]
    screen.draw.text(s, topleft=(100,250), color=RED, fontsize=60)
```

Loop Through All List Values (Method 1)

The **for** loop can be used to access all items in a list.

```
names = ['Bob', 'Barney', 'Baz', 'Ben', 'Barb', 'Bridget']

def draw()
    y = 50
    for name in names:
        screen.draw.text(name, topleft=(600,y),
                               color=YELLOW, fontsize=40)
        y += 50
```

It is common to use a plural description for a list (e.g. names)

The loop variable is the singular of the same description (e.g. name)

Loop Through All List Values (Method 2)

The **for** loop can also be used to access items by **index**.

```
dogs = ['Collie','Spitz','Daschund','Labrador']

def draw():
    x = 40
    for i in range(len(dogs)):
        screen.draw.text(dogs[i], topleft=(x,500),
                                  color=BLACK, fontsize=30)
        x += 140
```

The **len()** function returns the length of the list (number of items). Each list item is accessed using the loop variable `i` (e.g. `dogs[i]`)

Change a List Value

Lists are **mutable**. That is, values can be **changed**, simply by assigning a new value.

```
names = ['Bob', 'Barney', 'Baz', 'Ben', 'Barb', 'Bridget']

names[1] = 'Nick'
names[4] = 'Sophie'

def draw()
    y = 50
    for name in names:
        screen.draw.text(name, topleft=(600,y),
                               color=YELLOW, fontsize=40)
        y += 50
```

Find a Value in a List

Values can be found in a list using the **if** statement or a loop.

```
names = ['Bob', 'Barney', 'Baz', 'Ben', 'Barb', 'Bridget']

def draw():
    . . .
    if 'Ben' in names:
        screen.draw.text('Ben Found', topleft=(600,400),
                                      color=BLACK, fontsize=40)
```

or to ignore uppercase or lowercase:

```
names = ['Bob', 'Barney', 'Baz', 'Ben', 'Barb', 'Bridget']

def draw():
    found = False
    for name in names:
        if name.upper() == 'BEN': found = True
    if found: screen.draw.text('Ben Found', topleft=(600,400),
                                            color=BLACK, fontsize=40)
```

Note: could also use `name.lower() == 'ben'`

Add Values to the List

There are three ways to add values to a list:

```
numbers = [1,2,3,4]
numbers.append(5)           # Adds 5 to the end
print(numbers)                 # [1,2,3,4,5]

numbers.insert(2, 10)       # Inserts 10 at index 2 (after 0,1)
print(numbers)                 # [1,2,10,3,4,5]

numbers2 = [20,21,22]
numbers.extend(numbers2)       # Adds the items in numbers2
                            # Can also use + or +=.
print(numbers)                 # [1,2,10,3,4,5,20,21,22]
```

Make a List of Aliens (start new file)

Anything, including **objects** such as the Actors in PyGame Zero can also be put in a list. Each Actor must be created (usually using a loop) and then appended to the list.

```
alien_list = []

x = 50
for i in range(5):
    alien = Actor('alien', center = (x,400))
    alien_list.append(alien)
    x += 100

def draw():
    . . .
    for alien in alien_list:
        alien.draw()
```

Delete and Replace Values from the List

There are three ways to delete items from a list:

1. Remove one item by value

```
a = [0, 2, 3, 5]
a.remove(2)
print(a)                 # [0, 3, 5]
```

2. del removes the item at a specific index:

```
a = [3, 2, 2, 1]
del a[1]                 # can be a slice e.g. [:2]
print(a)                 # [3, 2, 1]
```

3. Pop removes the item at a specific index and returns it.

```
a = [4, 3, 5]
a.pop(1)
print(a)                # [4, 5]
```

Sort a List

```
a = [5, 2, 4, 1]
a.sort()                           # sorts the list
print(a)                           # [1,2,4,5]
a.reverse()                  # reverses the list
print(a)                           # [5,4,2,1]
```

Challenge

3. Favourite Things. Create three lists:

- a list of your five favourite things (call it things_list)
- a list of five x values (call it x_list)
- a list of five y values (call it y_list)

Display your favourite things on the screen at the x,y positions.

A List of Lists (start new file)

Instead of using separate lists, all values can be included in one list by using a list of lists. This example takes the favourite things, x and y lists and combines them into one list.

```
movies = [ ['Alien',100,50],            #name,x,y
           ['First Man',100,50],
           ['Apollo 13',100,50],
         ]

def draw():
    for movie in movies:
        screen.draw.text(movie[0], topleft=(movie[1],movie[2]),
                         color=BLACK, fontsize=60)
```

or

```
    for i in range(len(movies)):
        screen.draw.text(movie[i][0], topleft=(movie[i][1],movie[i][2]),
                         color=BLACK, fontsize=60)
```

Challenge

4. Hangman Stick Figure

   a. Design hangman stick figure on grid paper
   b. Make a list of lines and circles, with their coordinates and radius
   c. Use the list to draw the hangman on the pyGame screen

Challenge example list:

```
parts = [
            ['line',100,100,100,200,0],  #type,x1,y1,x2,y2,rad
            ['line',100,100,130,140,0],
            ['circ',100,80, 0, 0,50]
        ]

def draw():
    . . .
```

F. PyGame Mouse and Keyboard Events

Respond to Mouse Clicks and Continuous Keyboard Presses (start new file)

Responding to mouse clicks is easy in PyGame Zero. Let's click on an alien.

```
alien = Actor('alien', center = (400,300))

def draw():
    screen.clear()
    screen.fill('lightblue')
    alien.draw()

def on_mouse_down (pos, button):            #respond to mouse clicks
    if button == mouse.LEFT:
        if alien.collidepoint(pos):
            if alien.image == 'alien':
                alien.image = 'alien_hurt'
                alien.angle = 180
            else:
                alien.image = 'alien'
                alien.angle = 0

def update():                               #respond to continuous keyboard presses
    if keyboard.right:
        alien.x += 1
        alien.angle = 270
    elif keyboard.left:
        alien.x -= 1
        alien.angle = 90
```

Click on Buttons or Press Keys to make Selections (start new file)

Let's make a simple quiz with the answers in buttons. Buttons are made with the PyGame Zero *textbox()* function.

1. Make the questions and boxes

```
answers = ['1\nHawaiian', '2\nMeat Lovers', '3\nCheese']
boxes = [  Rect((100,250),(150,75)),
           Rect((300,250),(150,75)),
           Rect((500,250),(150,75))
        ]
choice = -1
```

2. Draw the boxes and answers

```
    screen.clear()
    screen.fill('lightblue')
    s = 'What is the best pizza?'
    screen.draw.text(s, midtop=(400,50), color='red', fontsize=60)

    for i in range(len(answers)):
        if choice == i: c = 'yellow'
        else: c = 'lightyellow'
        screen.draw.filled_rect(boxes[i],c)
        screen.draw.text(answers[i],
                         center=(boxes[i].x+75,boxes[i].y+37),
                         color='black', fontsize=30)
```

3. Check for mouse click

```
def on_mouse_down (pos, button):
    global choice
    if button == mouse.LEFT:
        for i in range(len(boxes)):
            if boxes[i].collidepoint(pos):
                choice = i
```

4. Check for key presses

```
def on_key_down(key):
    global choice
    if key in [keys.H, keys.K_1]: choice = 0
    elif key in [keys.M, keys.K_2]: choice = 1
    elif key in [keys.C, keys.K_3]: choice = 2
```

Challenge

5. Quiz

   a. Write a question with 4 possible answers each (one is correct)
   b. Create a screen showing this question and 4 boxes with the answers.
   c. Write code to respond to a mouse click and a key press to answer the quiz questions. If the answer is correct, display a message.

G. PyGame Functions

A function is a block of code which only runs when it is called.

Functions let us break the code into smaller re-usable chunks that are much easier to write, debug and insert into the whole program.

There are two parts to using a function – **defining** it and **calling** it.

Define the function (start a new file)

```
def draw_rings():                                           #must have ():
    for i in range(4):                                      #indented (tab)
        screen.draw.circle((500,500), 5 + i*10, RED)
```

Call the function

```
def draw():
    draw_box()
```

Function Parameters

You can pass data, known as parameters, into a function. Parameters (in the brackets, separated by a comma) make the function more general, so it can be applied to more situations.

Define the function

```
def draw_rings(x,y,rings,radius,col):
    r = radius
    for i in range(rings):
        screen.draw.circle((x,y), r, col)
        r += radius
```

Call the function (using the data types used in the function)

```
def draw():
    draw_rings(600,500,5,10,RED)
    draw_rings(500,400,10,5,YELLOW)                #multiple use
```

Passing Variables as Parameters

Variables can be passed to the function as parameters, just like numbers and strings.

```
def draw():
    x = 50
    r = 5
    for i in range(6):
        if i%2 == 0: c = RED
        else: c = YELLOW
        draw_rings(x,300,r,10,c)
        x += 100
        r += 5
```

Function Keyword Arguments

Python functions can be defined with named arguments which may have default values provided (known as **keyword arguments**). Keyword arguments can be passed in any order — not just the order that they were defined in the function. If a keyword argument is not passed to the function, the default value is used.

```
def findvolume(length=1, width=1, depth=1):
  return length * width * depth;

findvolume(1, 2, 3)
findvolume(length=5, depth=2, width=4)
findvolume(2, depth=3, width=4)
```

Return Values from a Function

Python functions are able to return values.

```
def square_point(x):
  x_squared = x * x
  return x_squared

three_squared = square_point(3)
print(three_squared)
```

Global vs Local Variables

Function parameters behave identically to a function's local variables. They are initialized with the values passed into the function when it was called.

Like local variables, parameters cannot be referenced from outside the scope of the function.

```
def my_function(value):
  print(value)

my_function(7)                # Pass the value 7 into the function

print(value)                  # Causes an error as value no longer exists
```

Final Challenge

6. Code a simple game (space invaders, flappy birds, matching pairs). The basic code will be supplied to you.

H. Other Useful Code

Use a Colour Wheel to Select 256 colours

```
def wheel(pos):
    # Input a value 0 to 255 to get a color value.
    # The colours are a transition r - g - b - back to r.
    if (pos < 0):
        return [0, 0, 0]
    if (pos > 255):
        return [0, 0, 0]
    if (pos < 85):
        return [int(pos * 3), int(255 - (pos*3)), 0]
    elif (pos < 170):
        pos -= 85
        return [int(255 - pos*3), 0, int(pos*3)]
    else:
        pos -= 170
        return [0, int(pos*3), int(255 - pos*3)]
```

Drag the Alien (start a new file)

The alien needs to get on board their spacecraft. First setup and draw the actors.

```
spacecraft = Actor('spacecraft', center = (400,150))
alien = Actor('alien', center = (200,400))

dragging = False                                #initialise mouse dragging
on_board = False

def draw():
    . . .
    if on_board:
        screen.draw.text('Made It!', midtop=(400,250),
                                     color=RED, fontsize=60)
    else:
        screen.draw.rect(box, RED)
        spacecraft.draw()
        alien.draw()
```

Write the mouse_down, mouse_move and mouse_up functions

```
def on_mouse_down (pos, button):
    global dragging
    if button == mouse.LEFT:
        if not dragging:                        #if not already dragging
            if alien.collidepoint(pos):         #if mouse down over alien
                dragging = True                 #start dragging

def on_mouse_move(pos,buttons):
    if dragging and mouse.LEFT in buttons:
        alien.pos = pos                         #move alien to new position

def on_mouse_up(pos):
    global dragging, on_board, locked_up
    if dragging:
        alien.pos = pos
        dragging = False                        #turn off dragging
        if alien.colliderect(spacecraft):
            on_board = True
```