Digital Technology

PyGame Zero

Level 2 Code and Challenges

Version 2.1
May, 2021

Barry Butler
bbutl58@eq.edu.au

ROBOCOAST
Sunshine Coast Robotics
www.robocoast.tech

Coolum
State High School
Care • Respect • Excellence

Contents

You will be presented with digital problems (the development of an idea into a digital solution consisting of a software and/or hardware prototype). These are the contexts in which you will learn the following coding techniques.

You need a solid understanding of Year 9 coding techniques before you begin these sections:

For each digital problem, you will need to:

1. Restate the problem in concise bullet points
2. Research the problem (target audience, user requirements, consult experts and users, existing software/hardware)
3. List the solution success criteria (outcomes) including specific functionality and data requirements
4. Construct a mind map based on the success criteria etc
5. Develop interface wireframes or mock-ups, interface hierarchy charts, diagrams of design ideas and hardware connection diagrams.
6. Construct data dictionaries (list of constants, data structures and variables) and data flow diagrams
7. Construct the required algorithms using IPO (input-Process-Output) charts, pseudocode and process flow charts.
8. Construct algorithms for testing the code as it is developed

A. PyGame Dictionaries

Data Structures

We have already come across some of the Python data structures:

Tuples	YELLOW = (255,216,0)
Lists	spacemen = ['Neil','Andy','Loren','Buzz']

Dictionaries

Dictionaries are used to store data values in **key:value pairs**.

It is unordered, changeable and does not allow duplicates.

Dictionaries are written with **curly brackets**, and have keys and values. The key is a string, the values can be any data type.

```
spaceship1 = {'type':'Saturn V', 'stages':3, 'year':1961,
              'color':(34,45,121), 'current':False}
```

Why Use Dictionaries not Lists?

- Searching for items in large datasets is hugely faster with dictionaries.
- Two of the most common data exchange formats (XML and JSON) contain key:value pairs in the same way as dictionaries. This is important as the files contain the meaning of the data as well as the data values.

Access Dictionary Values

Values in a dictionary are unordered and cannot be accessed by an index.

To print the whole dictionary

```
print(spaceship1)
```

Values are accessed via the key, by either of two methods.

```
t = spaceship1['type']
y = spaceship1.get('year')
print(t, y)
```

Separate lists of the all keys and all values can also be accessed.

```
print(spaceship1.keys())
print(spaceship1.values())
```

Change, Add or Remove Dictionary Values

Items are changed via the key

```
spaceship1['year'] = 1964
print(spaceship1)
```

Items are added using a new key and assigning a value

```
spaceship1['burntime'] = 20
print(spaceship1)
```

Items are removed using the pop() method

```
spaceship1.pop('color')
print(spaceship1)
```

Loop the Dictionary

Loops can be used to access both keys and values.

```
def draw():
    . . .
    y = 50
    for item in spaceship1:
        screen.draw.text(item +'  '+str(spaceship1[item]),
                        topleft=(100,y), color=RED, fontsize=40)
        y += 50
```

Alternatively, use these loops for keys and values separately

```
    for key in spaceship1.keys():
        screen.draw.text(key, topleft=(400,y),
                         color=YELLOW, fontsize=40)
```

```
    for value in spaceship1.values():
        screen.draw.text(str(value), topleft=(550,y),
                         color=YELLOW, fontsize=40)
```

Create a Dictionary of Dictionaries

Dictionaries can be imbedded in other dictionaries - when data is being searched by a key. Make the key in the container dictionary from values of the inner dictionaries.

```
spaceship1 = {'type':'Saturn V','stages':3,'year':1961}
spaceship2 = {'type':'Saturn II','stages':2,'year':1957}
spaceship3 = {'type':'Saturn I','stages':1,'year':1952}
spaceship4 = {'type':'Saturn VIII','stages':4}

spacetransport = {}
spacetransport[spaceship1['type']] = spaceship1
spacetransport[spaceship2['type']] = spaceship2
spacetransport[spaceship3['type']] = spaceship3
spacetransport[spaceship4['type']] = spaceship4

print(spacetransport)
print(spacetransport['Saturn I'])
```

List of Dictionaries

Lists have a fundamental problem – the meaning of the values is not imbedded in the data structure as it is in dictionaries. But lists can be sorted, whereas dictionaries cannot.

We can place a dictionary in a list.

```
spaceship1 = {'type':'Saturn V','stages':3,'year':1961}
spaceship2 = {'type':'Saturn II','stages':2,'year':1958}
spaceship3 = {'type':'Saturn I','stages':1,'year':1952}
spaceship4 = {'type':'Saturn VIII','stages':4,'year':1978}

spaceships = [spaceship1, spaceship2, spaceship3, spaceship4]
```

Sort the List of Dictionaries

We can access dictionary values (e.g. ‘type’) for each list item to sort the list.

```
for i in range(len(spaceships)-1):
    for j in range(i+1,len(spaceships)):
        if spaceships[j]['type'] < spaceships[i]['type']:
            temp = spaceships[i]
            spaceships[i] = spaceships[j]
            spaceships[j] = temp
```

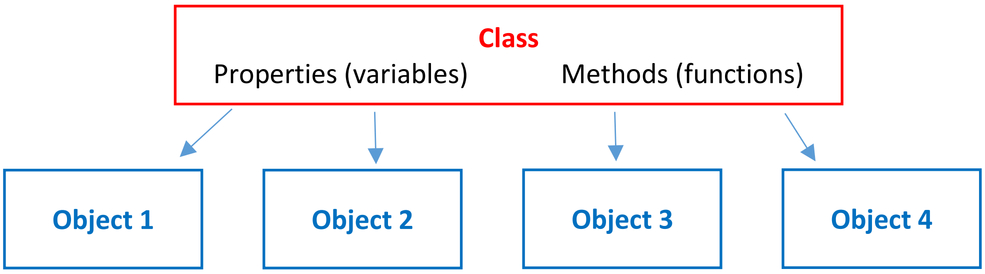
Display the List of Dictionaries on the Screen

```
def draw():
    screen.clear()
    screen.fill(LBLUE)
    x = 50
    for spaceship in spaceships:                             #loop this list
        y = 50
        for value in spaceship.values():                     #loop the dictionary
            screen.draw.text(str(value), topleft=(x,y),
                             color=RED, fontsize=30)
            y += 50
        x += 150
```

B. PyGame Objects

Objects

- Python is an **object oriented** programming language.
- Almost everything in Python is an object, with its **properties** (object variables) and **methods** (object functions).
- A Class is like an object **constructor**, or a "blueprint" for creating objects.
- One class is created as a template for one or more objects.



Create a Class and an Object

Let's create a **class** for a bouncing ball, with the **properties** it needs. The first letter of a class is written with a **capital**.

```
class Ball():
    x = 0
    y = 0
    radius = 10
    x_speed = 2
    y_speed = 2
    color = 'yellow'
```

Don't worry about getting this complete from the start. It is really easy to add new properties as you need them.

Use the class named Ball to create an **object** (instances of that class). The object is a python **variable** so we write the variable all in lowercase.

```
class Ball():
    x = 0
    y = 0
    radius = 10
    x_speed = 2
    y_speed = 2
    color = 'yellow'

ball1 = Ball()                    #one instance of the class Ball
```

The Initialisation Method

The built-in initialisation **method** sets up the object with key parameters that determine its' position, look and behaviour.

The function is called __init__(). Inside the brackets list the parameters required to set up the object. The first parameter of any method must be **self**.

```
class Ball():
    x = 0
    y = 0
    radius = 10
    x_speed = 2
    y_speed = 2
    color = 'yellow'

    def __init__(self, x, y, radius, x_speed, y_speed, color):
        self.x = x
        self.y = y
        self.radius = radius
        self.x_speed = x_speed
        self.y_speed = y_speed
        self.color = color

ball1 = Ball(400, 300, 40, 2, 2, 'red')
```

Create Multiple Objects of the same Class

We can now create lots of balls, and put them in different positions on the screen.

```
class Ball():
    x = 0
    y = 0
    radius = 10
    x_speed = 2
    y_speed = 2
    color = 'yellow'

    def __init__(self, x, y, radius, x_speed, y_speed, color):
        self.x = x
        self.y = y
        self.radius = radius
        self.x_speed = x_speed
        self.y_speed = y_speed
        self.color = color

ball1 = Ball(400, 300, 40, 2, 2, 'red')
ball2 = Ball(50, 50, 70, 3, 1, 'yellow')
ball3 = Ball(720, 400, 50, 1, 5, 'blue')
ball4 = Ball(200, 500, 20, 7, 7, 'black')
```

Add Other Methods

So far, the object does not do anything. We must write **methods** (object functions) that handles **everything** involved with the object.

Let's write a method to draw a ball, then call that method for each ball to draw it on the screen.

```
class Ball():
    . . .

    def draw(self):
        screen.draw.filled_circle((self.x, self.y), self.radius, self.color)

ball1 = Ball(400, 300, 40, 2, 2, 'red')
ball2 = Ball(50, 50, 70, 3, 1, 'yellow')
ball3 = Ball(720, 400, 50, 1, 5, 'blue')
ball4 = Ball(200, 500, 20, 7, 7, 'black')

def draw():
    screen.clear()
    screen.fill('white')
    ball1.draw()
    ball2.draw()
    ball3.draw()
    ball4.draw()
```

Method to Move the Ball

Write a method to move the ball.

```
WIDTH = 800
HEIGHT = 600

class Ball():
    . . .
    def update(self):
        self.x += self.x_speed
        self.y += self.y_speed

        if self.x < 0 or self.x > WIDTH: self.x_speed = - self.x_speed
        if self.y < 0 or self.y > HEIGHT: self.y_speed = - self.y_speed

ball1 = Ball(400, 300, 40, 2, 2, 'red')
. . .

def draw():
    screen.clear()
    screen.fill('white')
    ball1.draw()
    . . .

def update():
    ball1.update()
    ball2.update()
    ball3.update()
    ball4.update()
```

Method to Define the Ball Rectangle

If we are going to check if this ball has collided with another object we need to define the ball's bounding rectangle. First, add a property to store the bounding rectangle. The add a method to calculate it. Call this method when the rectangle is created (in __init__), and when its' position is updated.

```
class Ball():
    x = 0
    y = 0
    radius = 10
    x_speed = 2
    y_speed = 2
    color = 'yellow'
    rect = Rect((0,0),(20,20))                       #(x,y),(width,height)

    def __init__(self, x, y, radius, x_speed, y_speed, color):
        self.x = x
        self.y = y
        self.radius = radius
        self.x_speed = x_speed
        self.y_speed = y_speed
        self.color = color
        self.get_rect()

    def draw(self):
        screen.draw.filled_circle((self.x, self.y), self.radius, self.color)

    def update(self):
        self.x += self.x_speed
        self.y += self.y_speed
        self.get_rect()

        if self.x < 0 or self.x > WIDTH: self.x_speed = - self.x_speed
        if self.y < 0 or self.y > HEIGHT: self.y_speed = - self.y_speed

    def get_rect(self):
        w = self.radius * 2
        rect = Rect((self.x-self.radius, self.y-self.radius),(w,w))     #(x,y),(w,h)
```

Create a list of objects

It would be so much easier if all the balls were in a list. Then we could simply loop through the list to draw and update the balls. The class code does not change, but create a ball list rather than individual balls.

```
ball_list = []

ball_list.append( Ball(400, 300, 40, 2, 2, 'red') )
ball_list.append( Ball(50, 50, 70, 3, 1, 'yellow') )
ball_list.append( Ball(720, 400, 50, 1, 5, 'blue') )
ball_list.append( Ball(200, 500, 20, 7, 7, 'black') )

def draw():
    screen.clear()
    screen.fill('white')
    for ball in ball_list: ball.draw()

def update():
    for ball in ball_list: ball.update()
```

Create a New Class by Inheritance

Classes can **inherit** the properties and methods of existing objects. Let's create a couple of bats using downloaded images. The pyGame Zero Actor class already exists so we create our new class based on that.

```
class Bat(Actor):                #inherits Actor properties x,y etc
    speed = 2                    #new properties (variables)
```

Create new Bat objects, using the Actor initialisation parameters. Then draw them. The new class can also override the methods of the parent class, by having a method of the same name.

```
bat1 = Bat('paddle', midleft=(20,300))        #download your own image for a bat
bat2 = Bat('paddle', midright=(780,300))

def draw():
    . . .
    bat1.draw()
    bat2.draw()
```

Test for Ball and Bat Collisions

Test for collisions in the update() method. Because we now know the ball bounding rectangle, we can collide with it.

```
def update():
    for ball in ball_list:
        ball.update()
        if bat1.colliderect(ball.rect):
            ball.x_speed = - ball.x_speed
        elif bat2.colliderect(ball.rect):
            ball.x_speed = - ball.x_speeds
```

We can access the Ball class properties by using the dot syntax (e.g. ball.rect, ball.x_speed. Alternatively, we could have written another method in the Ball class to test for the collision.

C. Create a User Interface

The design of a user interface is an important part of giving the software user a great experience. We are going to design a simple user interface.

Download the free user interface tool at https://www.justinmind.com/free-wireframing-tool. Another on-line tool is https://marvelapp.com/ and https://marvelapp.com/pop .

1. Draw the User Interface and List the Tasks (Success Criteria)

First, we need to draw the user interface and work out the **tasks** that are required by the user interface window. These are the **success criteria** for the app.

Every user interface can be separated into the following components:

- Background (Shapes, Images)
- Text
- Controls (buttons, sliders, menus, inputs, radio buttons, checklists, date selectors etc)
- Hot spots (clickable areas)
- Actors/sprites
- Movement
- Collisions
- Mouse actions (click, up, drag)
- Keyboard input
- Hidden (opening and saving files etc)

A list must be made of all the constants and variables required (as inputs or outputs)

For example, this is a breakdown of the "Sheep are Cool1" window:

Component	Tasks (Success Criteria)	PyGame Functions	Coding Elements/ Data
Background	Draw background colour (blue)	screen.fill()	color constants
	Draw light blue rectangle	screen.draw.filled_rect()	
	Draw sheep image	sheep = Actor() sheep.draw()	
Text	Draw title	screen.draw.text()	
	Draw page number		page = 1
	Draw lines of text in rectangle		text = [] for line in text:
Controls	Draw left arrow	left_arrow = Actor()	
	Draw right arrow		
Actor/sprite	Draw cat sprite	cat = Actor()	
Movement	Move cat along the bottom of the page from left to right and reverse	def update()	cat.x, cat.y
Mouse/keyboard	Click left arrow to move down pages	def on_mouse_down()	
	Click right arrow to move up pages	def on_key_down()	
	Press left arrow key to move down pages		
	Press right arrow key to move up pages		

When we construct the UI, we will do it in an order very close to the list here.

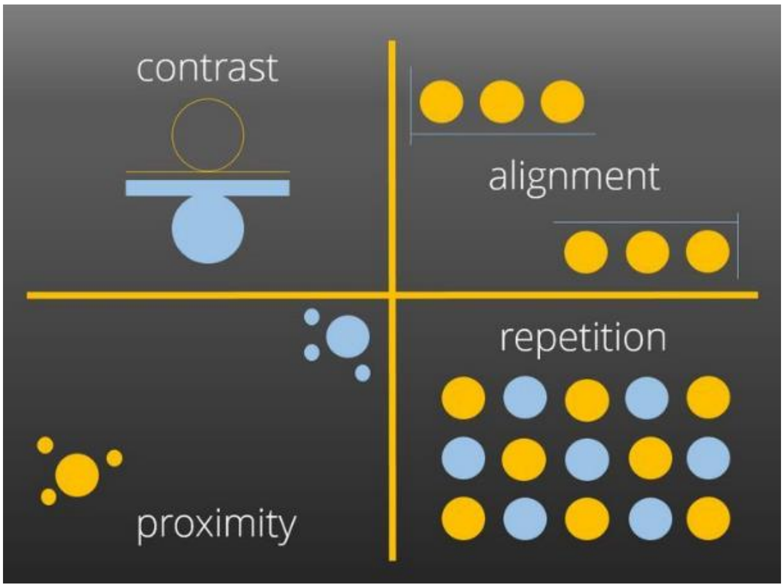
The Principles of User Interface Design

Learn to design with your user's needs and expectations in mind by applying Jakob Nielsen and Rolf Molich's Ten User Interface Guidelines. These heuristics have been reflected in many of the products designed by some of the most successful companies in the world such as Apple, Google, and Adobe.

See https://www.interaction-design.org/literature/article/user-interface-design-guidelines-10-rules-of-thumb

a. **Visibility of system status**. Users should always be informed of system operations with easy to understand and highly visible status displayed on the screen within a reasonable amount of time.
b. **Match between system and the real world -** mirror the language and concepts users would find in the real world. Present information in logical order.
c. **User control and freedom** - backward steps are possible, including undoing and redoing previous actions.
d. **Consistency and standards** - ensure that both the graphic elements and terminology are maintained across similar platforms.
e. **Error prevention -** potential errors are kept to a minimum. Users do not like being called upon to detect and remedy problems, which may on occasion be beyond their level of expertise.
f. **Recognition rather than recall -** recognizing something is always easier than recall
g. **Flexibility and efficiency of use -** allow faster navigation using abbreviations, function keys, hidden commands and macro facilities. Users should be able to customize or tailor the interface to suit their needs.
h. **Aesthetic and minimalist design.** Keep clutter to a minimum whilst providing clearly visible and unambiguous means of navigating to other content.
i. **Help users recognize, diagnose and recover from errors** - error messages expressed in plain language.
j. **Help and documentation** - easily located, specific and step-by-step.

The CARP Principles of Design



Also see:

https://www.smashingmagazine.com/2018/02/comprehensive-guide-ui-design/

1. Build Missing Components (e.g. a Clickable Button)

We will construct a clickable button as an object Class, so we can re-use it for each button on the window.

```
#BUTTON------------------------------
class Button():
    text = ''
    x = 0
    y = 0
    width = 0
    height = 0
    col = BROWN
    visible = True
```

Next, write the code to initialise the button when we create objects from the Class.

```
#BUTTON------------------------------
class Button():
    text = ''
    x = 0
    y = 0
    width = 0
    height = 0
    col = 'light blue'
    visible = True

    def __init__(self,text,x,y,width,height,col,visible):
        self.text = text
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.col = col
        self.visible = visible
```

Write a function to draw the button – filled rectangle, then unfilled rectangle, finally the text

```
    def draw(self):
        screen.draw.filled_rect(Rect((self.x,self.y),(self.width,self.height)),
                                self.col)
        screen.draw.rect(Rect((self.x,self.y),(self.width,self.height)),'black')
        mid_x = self.x + self.width//2
        mid_y = self.y + self.height//2
        screen.draw.text(self.text,center=(mid_x,mid_y),color='black',fontsize=30)
```

Write a function to test if the mouse click (x,y) is inside the button rectangle, and return True if it is.

```
    def click(self,pos):       #pos = (x,y)
        found = False
        x = pos[0]
        y = pos[1]
        if (x > self.x and x < self.x+self.width)
                 and (y > self.y and y < self.y+self.height):
            found = True
        return found
```

That's it. Now test the code by creating a button object, drawing it on the window and testing for a mouse click.

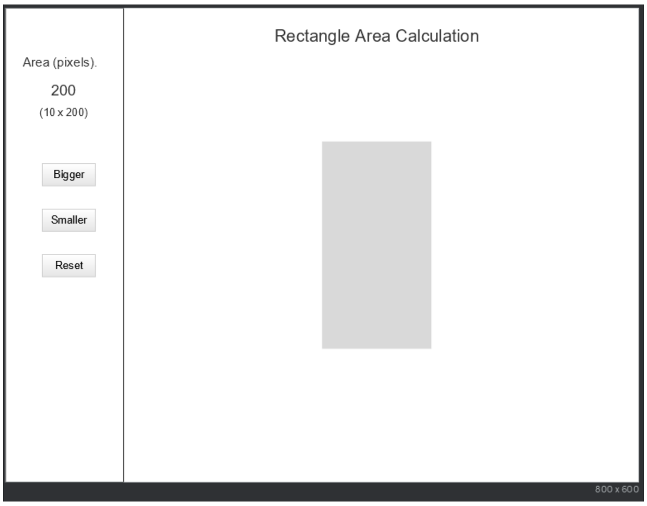
```
#BUTTON-----------------------------
class Button():
    . . .

ok_button = Button('OK',400,300,70,40,'yellow',True) #text,x,y,width,height,col,visible

def draw():
    screen.clear()
    screen.fill('white')
    ok_button.draw()

def on_mouse_down(pos,button):
    print(pos)
    if button == mouse.LEFT:
        if ok_button.click(pos):
            print('OK button clicked')
```

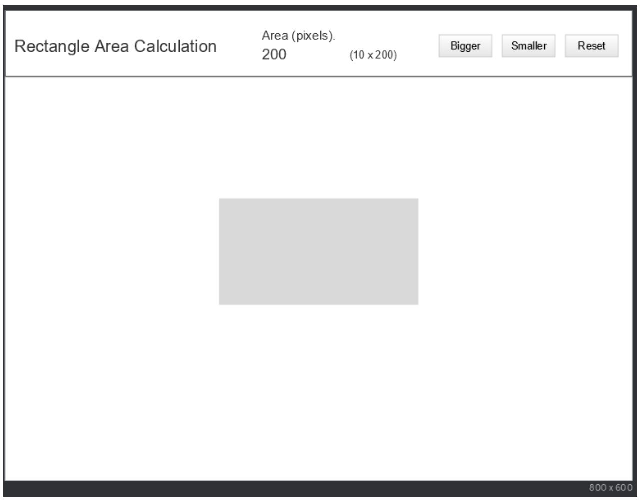
2. Code the Complete Window

We will build a window to will calculate the area of a rectangle, with controls to change its' size. This will answer the question 'What happens to the area of a rectangle when its' size increases' – a common year 9 Maths problem.

3a. Wireframe diagram

First, we will draw a **wire-frame of the window**. Try different layouts (e.g. buttons on panel at the top or the left).





3b. Table of Components, Tasks and Data

Second, we will construct a table of components, tasks (success criteria) and data requirements.

Component	Tasks (Success Criteria)	PyGame Functions	Coding Elements/ Data
Background	Draw background colour (mid blue)	screen.fill()	color constants
	Draw blue rectangle (left or top)	screen.draw.filled_rect()	**class Window()** def draw()
Text	Draw title	screen.draw.text()	
	Draw Area label		
	Draw Data labels		area, width, height
Controls	Draw Bigger button	Button.draw()	bigger_btn
	Draw Smaller button		smaller_btn
	Draw Reset button		reset_btn
Rectangle for area calculation	Draw rectangle	screen.draw.filled_rect()	**class RectArea()** Draw using width, height
Mouse	Click Bigger button	def on_mouse_down()	Resize rectangle and calculate area
	Click Smaller button		Resize rectangle and calculate area
	Click Reset button		Reset rectangle and calculate area

From this table, we decide what object classes we will need and their properties and methods. It is always a good idea to separate the problem data from the window layout so we will create the following classes:

```
class Window():                    layout rectangles, text and buttons
        bigger_btn
        smaller_btn
        reset_btn

        def draw(self, area, width, height)             #data required to put on screen

class RectArea():
        width = 10
        height = 20
        area = 200

        def draw()
        def calc_area()
        def resize()
        def reset()
```

We will now start writing the code.

Test each coding stage carefully before going on!!

3c. Code the Window Class and Draw

Third, we will set up our colours and make an object class for the window and draw the window

```
#GLOBAL CONSTANTS----------------------------------------------------------------
WIDTH = 800
HEIGHT = 600

LBLUE = (189,242,255)
GBLUE = (94,157,173)
MIDBLUE = (160,232,250)
BROWN = (173,123,76)
LBROWN = (250,203,160)
BLACK = (0,0,0)
WHITE = (255,255,255)

# <--------------------Button Class inserted here

#WINDOW-------------------------------------------------------------------------
class Window():
    bigger_button = Button('Bigger',35,220,80,40,BROWN,True)
    smaller_button = Button('Smaller',35,280,80,40,BROWN,True)
    reset_button = Button('Reset',35,340,80,40,BROWN,True)

    def draw(self, area, width, height):
        screen.draw.filled_rect(Rect((0,0),(150,HEIGHT)),MIDBLUE)
        screen.draw.filled_rect(Rect((150,0),(WIDTH-150,HEIGHT)),LBLUE)

        screen.draw.text('Rectangle Area Calculation',(335,10),color=BROWN,fontsize=30)
        screen.draw.text('Area (pixels)',center = (75,60),color=BLACK,fontsize=25)
        screen.draw.text(str(area),center = (75,100),color=BROWN,fontsize=30)
        screen.draw.text('('+str(width)+', '+str(height)+')',center = (75,140),
                                color=BROWN,fontsize=30)
        self.bigger_button.draw()
        self.smaller_button.draw()
        self.reset_button.draw()

window = Window()

#PYGAME FUNCTIONS---------------------------------------------------------------
def draw():
    screen.clear()
    window.draw(200,10,20)           #put test data in here before the real thing
```

3d. Code for Mouse Clicks and Test

Once we have debugged the code and are happy with the static layout of the window, we are ready to respond to mouse clicks:

```
def on_mouse_down(pos,button):
    if button == mouse.LEFT:
        if window.bigger_button.click(pos):
            print('Bigger')                             #use print to check it's working
        elif window.smaller_button.click(pos):
            print('Smaller')
        elif window.reset_button.click(pos):
            print('Reset')
```

3e. Code the Rectangle Area Class

```
#RECTAREA-----------------------------------------------------------------------
class RectArea():
    width = 10
    height = 20
    area = 200

    def draw(self):
        x = 150 + (650//2) - (self.width//2)
        y = HEIGHT//2 - (self.height//2)
        screen.draw.filled_rect(Rect((x,y),(self.width,self.height)),WHITE)

    def calc_area(self):
        self.area = self.height * self.width

    def resize(self,bigger):                  #True – make bigger, False - smaller
        if bigger: factor = 2
        else: factor = 0.5
        w = round(self.width * factor)        #use temp variables to calculate first
        h = round(self.height * factor)
        if w >= 5 and w <= 160:               #test if new width is not too small/big
            self.width = w                    #set values
            self.height = h
        self.calc_area()

    def reset(self):
        self.width = 10
        self.height = 20
        self.calc_area()

rect_area = RectArea()
```

3f. Link to Buttons and Draw Function to Test

```
#PYGAME FUNCTIONS--------------------------------------------------
def draw():
    screen.clear()
    window.draw(rect_area.area,rect_area.width,rect_area.height)
    rect_area.draw()

def on_mouse_down(pos,button):
    if button == mouse.LEFT:
        if window.bigger_button.click(pos):
            rect_area.resize(True)
        elif window.smaller_button.click(pos):
            rect_area.resize(False)
        elif window.reset_button.click(pos):
            rect_area.reset()
```

That's it!!

D. User Interface Controls

The following user interface controls have been constructed for you in PyGame:

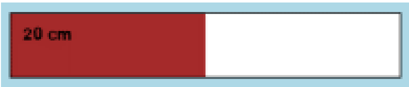
Button


```
ok_button = Button('OK',100,100,70,40,'light blue',True)
```

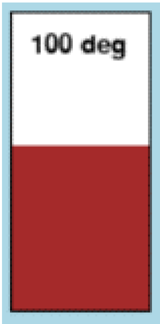
#text, x, y, width, height, color, visible

```
ok_button.draw()

def on_mouse_down(pos,button):
    if ok_button.click(pos, button):
        print('OK button clicked')
```

Horizontal Slider



Vertical Slider

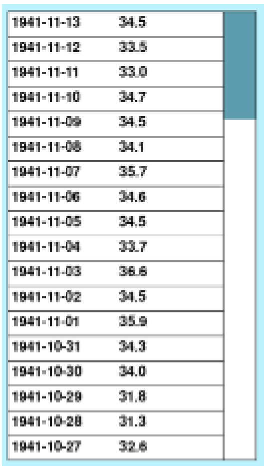



```
slider = Slider(50,70,300,50,0,40,20,' cm','brown')
vslider = VSlider(400,70,70,150,0,180,90,' deg','brown')
```

#x,y,width,height,min_value,max_value,value,unit,colour

```
slider.draw()
vslider.draw()

def on_mouse_down(pos, button):
    if slider.click(pos, button):
            print(slider.value, slider.percent)
    if vslider.click(pos, button):
            print(vslider.value, vslider.percent)
```

List Control



1941-11-13	34.5
1941-11-12	33.5
1941-11-11	33.0
1941-11-10	34.7
1941-11-09	34.5
1941-11-08	34.1
1941-11-07	35.7
1941-11-06	34.6
1941-11-05	34.5
1941-11-04	33.7
1941-11-03	36.6
1941-11-02	34.5
1941-11-01	35.9
1941-10-31	34.3
1941-10-30	34.0
1941-10-29	31.8
1941-10-28	31.3
1941-10-27	32.6

```
data_list = ['Option 1','Option 2','Option 3']
                        #list of strings or list of lists
```

```
display_table = ListControl(400,20,data_list,18,2,130,True)
```

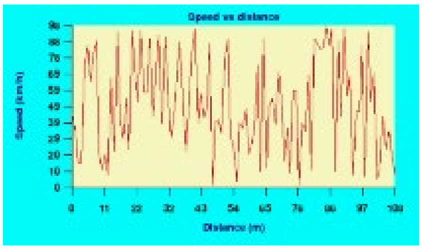
#x, y, data_list, no of rows, no. of columns, column width, show_slider)

or

```
display_table = ListControl(400,20,None,18,2,130,True)

table.clear_table()
for row in data_list:
    table.add_table_row(row, False, 0)
```

```
display_table.draw()

def on_mouse_down(pos, button):
    global data_list
    if display_table.click(pos, button):
        if display_table.index > -1:
            print(display_table.value)
```

Graph (0=Line, 1=Scatter and 2=Bar)




```
graph = Graph(300, 200, 400, 200, 'Speed vs distance',
          'Speed (km/h)', 'Distance (m)', 0)
```

origin_x, origin_y, x_len, y_len, title, label_x, label_y, graph_type

```
data_list = [[x1,y1],[x2,y2]]    #make a data array
graph.xydata = data_list         #add all data at once

graph.draw()

graph.clear()                    #remove all data

graph.add(x,y)                   #add a data point
graph.delete_first()             #delete data at index 0
```

E. PyGame Files

File Handling

No data in programs is available once the program ends unless it is stored in a file.

Python has several functions for creating, reading, updating, and deleting files.

The Python Operating System (OS) library also has functions for creating and removing folders; and removing files.

Data can be stored in a variety of file formats including text, CSV, JSON and XML. The use of flat and relational database files (e.g. mongoDB and SQLite) is covered in other Lessons.

HINT: In general, use CSV files for tabular data and JSON for structured data.

Get a List of Files in a Folder

```
from os import listdir

data_folder = 'data'
data_files = listdir(data_folder)
print(data_files)
```

Opening and Closing Text Files

For data to be stored safely, files must be properly opened and closed. The easiest way to do this is using a **with** statement.

```
with open('spacecraft.txt', 'w') as f:
    pass
```

Using this code the file will be opened and automatically closed.

There are two common modes for opening files:

'r' Open for reading (default)

'w' Open for writing, truncating (overwriting) the file first

'a' Open for writing, adding new data to the end of the file.

Write and Read Data with Text Files

To write data to a text file:

```
spacemen = ['Neil','Andy','Loren','Buzz']

with open('astronaut.txt', 'w') as f:
    for item in spacemen:
        f.write(item + '\n')         #\n puts a line between items
```

Open the file and check the contents – each on a separate line.

To read data from the text file:

```
spacemen = []
with open('astronaut.txt', 'r') as f:
    for line in f:
      spacemen.append(line[:-1])    #[:-1] takes off last 2 chars

print(spacemen)                     #test that read successful
```

Write and Read Data with CSV Files

To write list data to a CSV file (line end is '\r\n' by default – carriage return and new line):

```
import csv

with open('spaceships.csv', mode='w') as f:
    writer = csv.writer(f, delimiter=',', quotechar='"',
                        lineterminator = '\n',
                        quoting=csv.QUOTE_MINIMAL)

    writer.writerow(['Type', 'Stages', 'Year'])
    for ship in ships:
        writer.writerow(ship)
```

Of course, you can also do it manually:

```
with open('aa_spaceships.csv', mode='w') as f:
    f.write("Type,Stages,Year\n")
    for ship in ships:
        s = '"'+ship[0]+'",'+str(ship[1])+','+str(ship[2])+'\n'
        f.write(s)
```

To read a CSV file to a list:

```
ships = []

with open('spaceships.csv', mode='r') as f:
    csv_reader = csv.reader(f, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:                  #column names
            pass
        elif not row == []:
            ship = [row[0], int(row[1]), int(row[2])]
            ships.append(ship)
        line_count += 1

for ship in ships: print(ship)
```

Write and Read JSON Files

JSON stands for JavaScript Object Notation, and was initially created for web site data transfer. A file looks like this:

```
{
    "firstName": "Jane",
    "lastName": "Doe",
    "hobbies": ["running", "sky diving", "singing"],
    "age": 35,
    "children": [
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

You will notice the similarity to a Python Dictionary.

To write a JSON file:

```
import json

spaceship1 = {'type':'Saturn V','stages':3,'year':1961}
spaceship2 = {'type':'Saturn II','stages':2,'year':1957}
spaceship3 = {'type':'Saturn I','stages':1,'year':1952}
spaceship4 = {'type':'Saturn VIII','stages':4}

spacetransport = {}
spacetransport[spaceship1['type']] = spaceship1
spacetransport[spaceship2['type']] = spaceship2
spacetransport[spaceship3['type']] = spaceship3
spacetransport[spaceship4['type']] = spaceship4

with open("spacestransport.json", "w") as f:
    json.dump(spacetransport, f)
```

To read a JSON file:

```
spacetransport = {}

with open("spacetransport.json", "r") as f:
    spacetransport = json.load(f)

print(spacetransport)
```

Write and Read XML Files

See:

https://kontext.tech/column/python/480/read-and-write-xml-files-with-python

https://stackabuse.com/reading-and-writing-xml-files-in-python/

https://www.geeksforgeeks.org/reading-and-writing-xml-files-in-python/

F. PyGame Strings

String Handling

Python is a language with rich built-in string (text) handling functions to:

- Find the length
- Change the case
- Check the characters (alpha, numeric, float)
- Strip whitespace
- Join, split strings
- Find and replace characters or substrings
- Format strings

References

You really don't need to remember how to handle strings – there are so many functions. Look up some good websites for details:

https://www.w3schools.com/python/python_strings.asp

https://www.w3schools.com/python/python_string_formatting.asp

https://www.shortcutfoo.com/app/dojos/python-strings/cheatsheet

https://www.codecademy.com/learn/learn-python-3/modules/learn-python3-strings/cheatsheet

Formatting Values in Strings

One of the most useful functions is to insert and format text, integers and decimals using the ***format*** function. Add placeholders (curly brackets {}) in the text, and a list of values as function parameters. For example.

```
price = 49
txt = "The price is {} dollars"
print(txt.format(price))
```

More than one placeholder can be used.

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {} pieces of item {} for {:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

Use indexes to be sure the values are placed correctly.

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {0} pieces of item {1} for {2:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

Common formatting codes include:

Integers	:d	{:d} {:5d}
Fixed Point Decimals	:f	{:.2f} {:8.2f}
Percentage	:%	
Left, right or centre aligned	:< :> :^	{:<8}
Comma as a thousands separator	:,	{:d:,}

For a complete list of formatting codes see:

https://www.w3schools.com/python/ref_string_format.asp

G. PyGame Dates and Time

Import the ***datetime*** module to handle dates and times.

```
import datetime

d = datetime.datetime.now()
print(str(d))
print(d.year)
print(d.month)
print(d.day)
print(d.hour,':',d.minute,':',d.second)
print(d.strftime("%A"))
```

To create a date in datetime format:

```
y = 2020
m = 11
d = 30
d = datetime.datetime(y,m,d)
print(str(d))
```

H. PyGame Serial Connection with CPX or Metro M4 CircuitPython Boards

First, we will write the CircuitPython code to save to the circuit board which will send data to the computer. It will do this in two ways: via the **print** statement that goes down the standard USB connection cable, and the **uart write** statement that goes down a special **USB to TTL Serial Adapter**.

Then we will write the PyGame Zero code for the computer to read the data that is being sent through either cable.

CPX Code to Send Serial Data (in CircuitPython Mode, save to CIRCUITPY as code.py)

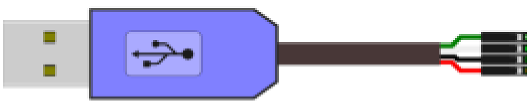
The CPX code used to send data is:

```
import time
import board
import busio
from adafruit_circuitplayground.express import cpx

uart = busio.UART(board.TX, board.RX, baudrate=115200)

while True:
    t = time.monotonic()
    l = cpx.light
    e = cpx.temperature
    output = "{0:.1f}, {1:d}, {2:.1f},\n".format(t,l,e)     #or \r\n
    uart.write(output.encode())                             #write to serial
    print(str(t) + ', ' + str(l) + ', ' + str(e) + ', ')    #print to REPL
    time.sleep(1.0)
```

Both the uart.write() and print() functions send serial data. Data must be sent with a line feed character at the end ("\n"). The print() function automatically does this.

Note: in early version of CircuitPython the code to write to the serial port was `uart.write(output)`

Connect the USB to TTL Serial Adapter



Green to RX (A6-D0 Signal)
White to TX (A7-D1 Signal)
Black to any GND

The **red** wire is not required and should not be used if you power the board from a battery or the native USB port.

Find the Computer Serial Ports in Use (PyGame Zero Mode)

Serial connections are used to connect devices to the computer and transfer data to and from that device.

The first thing we need to know is the name computer **port** used in the serial connection. Run the following code without the device plugged in, then again with it plugged in, to get the name of the connection port.

```
import serial
import serial.tools.list_ports as port_list

ports = list(port_list.comports())
for p in ports:
    print(p)
```

Create a Serial Connection

To connect to another device you need a bit of information: baud rate (how fast the data can be transmitted); parity (Positive, Negative or None); stop bits; byte size (in bits)

For example, a CPX board will connect with a baud rate of 115200, parity of None, one stop bit and a bitesize of 8.

The following code will connect to a CPX board, print its' name and flush the input buffer ready for receiving data. Change the **port** to be the name of the port on your computer.

```
ser = serial.Serial(
    port="COM53",                 #replace this with your serial port
    baudrate=115200,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    )
print(ser.name)
ser.flushInput()
```

Read Serial Data from a CPX

The readline function is used to read a series of data bytes ending in \n. The code then turns this into a string of comma delimited data. Create a global variable (e.g. data_list) to store the values.

```
data_list = []                                              #global variable

def update():
    global data_list
    data = ser.readline()
    if data is not None:
        data_string = "".join([chr(b) for b in data])       #turn bytes into a string
        data_list = data_string.split(",")                  #create a list of data
        #print(data_list, len(data_list))
        if len(data_list) >= 3:
            print('time, light, temperature')
            print(data_list[0], data_list[1], data_list[2])
```

Data is sent from the CPX ending with either "\r\n" or "\n". Use the print statement to see what has been written and remove it, for example, by using `x[2] = x[2][0:-5]`

Write PyGame Serial Data to a CPX (PyGame Zero mode)

There are two ways of sending serial data. Data must be converted to a string and end with a line feed.

```
x = 186.23
s = str(x) + "\n"
ser.write(s.encode())      #send numeric variable as a string

s = "12.54\n"
ser.write(s.encode())      #send a string variable

ser.write(b"1234.5654\n")  #send a constant value
```

CPX Code to Read Data from Serial Connection (CircuitPython Mode)

```
import time
import board
import busio

uart = busio.UART(board.TX, board.RX, baudrate=115200)

blist = []
s = ''
while True:
    data = uart.read(1)                                 #read 1 to 32 bytes
    if data is not None:                                #data was received
        blist.append(data)                              #so can see actual data
        s = s + ''.join([chr(b) for b in data])         #convert to a string
        eol = (s[-1] == '\n') or (s[-1] == '\r')        #test of \n or \r

        print(s, str(blist), eol)                       #check it's all working

        if eol:
            uart.write('Data Received: '+s + '\n')      #send back confirmation
            s = ''
            blist.clear()
```

NOTE: Plug in serial USB connection before saving. '\r' is generated by pressing the Enter key when using a terminal emulator program.

Test with a Terminal Emulator Program

A terminal emulator program can be used to check connections to a CPX or other equipment. For information to go:

https://learn.adafruit.com/circuit-playground-express-serial-communications/overview

Windows

Download PuTTY from https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

See https://learn.adafruit.com/welcome-to-circuitpython/advanced-serial-console-on-windows

Mac

See https://learn.adafruit.com/welcome-to-circuitpython/advanced-serial-console-on-mac-and-linux

Steps:

1. Open Terminal (Command-Space)
2. Type: ls /dev/tty.*
   This will show a list of all the serial ports
3. To connect with the serial port and show input data on screen:
   screen /dev/tty.boardname 115200
4. To exit the serial connection and disconnect from the serial port:
   Ctrl-A then Ctrl-\

Challenge

Write a user interface for reciving CPX data

Write a PyGame program to send the CPX messages to turn on pixels, play tones and send back light / temperature information and button presses.

I. SQLite Relational Database

SQLite

- SQLite is a relational database management system contained in a C library.
- SQLite is not a client–server database engine. Rather, it is embedded into the end program.
- A database is made up of **tables**, each with a number of **columns**.

See https://www.sqlite.org/index.html for more information.

SQLiteStudio

SQLiteStudio is an application to create, edit, and browse SQLite databases. It is very handy to check the database tables and test SQL statements.

To download, visit https://sqlitestudio.pl/ .

Create a New Database

To create a database:

1. Import the libraries
2. Create a folder using the Python **os** library(this example will create a **mu_code\sqlite** folder)
3. Link the database to that folder and a file name.

```
import sqlite3                              #import sqlite library
from sqlite3 import Error

path = r"C:\Users\bbutl58\mu_code\sqlite"
try:
    os.mkdir(path)
except OSError as e:
    print (e,"Creation of the directory %s failed" % path)
database = os.path.join(path,"tutorial.db")
```

Next, write the code to create or open the database file.

```
def create_connection(db_file):
    conn = None
    try:                                                                   #trap any erros
        conn = sqlite3.connect(db_file)      #open or create the file
        return conn
    except Error as e:
        print(e)

    return conn

conn = create_connection(database)
if conn is not None:                        #immediately close the file
    conn.close()
```

To create a new database in memory use: conn = sqlite3.connect(':memory:')

SQL Commands

SQL is a standard language for storing, manipulating and retrieving data in databases.

The main commands we will use are:

CREATE TABLE	Creates a new database table
INSERT INTO	Inserts new records into a table
UPDATE	Updates a record in a table
DELETE FROM	Deletes a record from a table
SELECT * FROM	Select columns from a table and returns data

See https://www.w3schools.com/sql/default.asp for a comprehensive tutorial of SQL.

Create Tables

To create a new table in a SQLite database from a Python program, you use the following steps:

1. Create a Connection object using the connect() function of the sqlite3 module (as above).
2. Create a Cursor object by calling the cursor() method of the Connection object.
3. Pass the CREATE TABLE statement to the execute() method of the Cursor object and execute this method.
4. Close the connection using the close() method.

SQL Constants

Create constants for each table. SQL statements should be created as string constants, embedding the SQL commands.

```
PROJECTS = 0
TASKS = 1

sql_create_projects = """ CREATE TABLE IF NOT EXISTS projects (
                                        id integer PRIMARY KEY,
                                        name text NOT NULL,
                                        begin_date text,
                                        end_date text
                                    ); """

sql_create_tasks = """ CREATE TABLE IF NOT EXISTS tasks (
                                    id integer PRIMARY KEY,
                                    name text NOT NULL,
                                    priority integer,
                                    status_id integer NOT NULL,
                                    project_id integer NOT NULL,
                                    begin_date text NOT NULL,
                                    end_date text NOT NULL,
                                    FOREIGN KEY (project_id) REFERENCES projects (id)
                                ); """
```

Table Functions – Create Table

Functions must be written to connect to the database, operate on the tables, and close the database. The first function is to create the tables.

```
def create_table(create_table_sql):
    conn = create_connection(database)
    if conn is not None:
        try:
            c = conn.cursor()
            c.execute(create_table_sql)  #if it does not exist
        except Error as e:
            print(e)
        conn.close()
    else:
        print("Error! Cannot create the database connection.")
```

Call the function:

```
create_table(sql_create_projects)
create_table(sql_create_tasks)
```

Use SQLiteStudio to check the results.

Insert Data into the Table

```
sql_insert_project = """ INSERT INTO projects(name,begin_date,end_date)
              VALUES(?,?,?) """

sql_insert_task = """ INSERT INTO tasks(name,priority,status_id,
project_id,begin_date,end_date)
              VALUES(?,?,?,?,?,?) """
```

```
def insert_table(table, data):
    id = -1
    conn = create_connection(database)
    if conn is not None:
        try:
            c = conn.cursor()
            if table == "project": c.execute(sql_insert_project, data)
            elif table == "task": c.execute(sql_insert_task, data)
            conn.commit()
            id = c.lastrowid        #project id
        except Error as e:
            print(e)
        conn.close()
    else:
        print("Error! Cannot create the database connection.")
    return id
```

Call the function to insert data into the table:

```
#insert project data
project = ('Cool App with SQLite & Python', '2015-01-01', '2015-01-30');

project_id = insert_table('project', project)
print('insert project',project_id)

#insert tasks data
task_1 = ('Analyze the requirements of the app', 1, 1, project_id, '2015-01-01', '2015-
01-02')
task_2 = ('Confirm with user about top requirements', 1, 1, project_id, '2015-01-03',
'2015-01-05')

task_id = insert_table('task', task_1)
print('insert task',task_id)
task_id = insert_table('task', task_2)
print('insert task',task_id)
```

Update Table Data

The following function can be used to update a data record. First, setup the SQL statement.

```
sql_update_task = """ UPDATE tasks
              SET priority = ? , begin_date = ? , end_date = ?
              WHERE id = ?"""
```

```
def update_table(table, data):                #data is a tuple (or list??)
    conn = create_connection(database)
    if conn is not None:
        try:
            c = conn.cursor()
            if table == PROJECTS: pass  #c.execute(sql_update_project, data)
            elif table == TASKS: c.execute(sql_update_task, data)
            conn.commit()
        except Error as e:
            print(e)
        conn.close()
    else:
        print("Error! Cannot create the database connection.")
```

```
update_table(TASKS, (2, '2015-01-04', '2015-01-06', 2))
```

Query and Display Table Data

The **SELECT** SQL statement is used to select data from one or more tables and return the data rows for analysis and/or display.

```
sql_select_all_tasks = "SELECT * FROM tasks"
sql_select_task_by_priority = "SELECT * FROM tasks WHERE priority=?"
```

```
def select_from_table(sql, data):           #data is a tuple (or list??)
    rows = []
    conn = create_connection(database)
    if conn is not None:
        try:
            c = conn.cursor()
            if not data == (): c.execute(sql, data)
            else: c.execute(sql)
            rows = c.fetchall()
        except Error as e:
            print(e)
        conn.close()
    else:
        print("Error! Cannot create the database connection.")
    return rows
```

Call the function.

```
rows = select_from_table(sql_select_all_tasks,())
print('Select all data:')
for row in rows:
    print(row)


rows = select_from_table(sql_select_task_by_priority,(2,))
print('Selected priority 2 data:')
for row in rows:
    print(row)
```

Delete Table Data

Setup the SQL statement, write a function and call the function.

```
def execute_sql(sql, data):                       #e.g. for delete
    conn = create_connection(database)
    if conn is not None:
        try:
            c = conn.cursor()
            if not data == (): c.execute(sql, data)
            else: c.execute(sql)
            conn.commit()
        except Error as e:
            print(e)
        conn.close()
    else:
        print("Error! Cannot create the database connection.")
```

References

https://docs.python.org/3/library/sqlite3.html#

https://www.sqlitetutorial.net/

https://www.sqlitetutorial.net/sqlite-python/

https://pythonspot.com/python-database-programming-sqlite-tutorial/

J. Other Useful Functions

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)]
```

Open a File Dialog Window

```
import tkinter as tk
from tkinter import filedialog

root = tk.Tk()
root.withdraw()

file_path = filedialog.askopenfilename()
print(file_path)
```