Please Read if you're using Python 3.7 or higher

This course was recorded before the release of Python 3.7. Since then, Pygame 1.9.4 has been released, which is compatible with the newer versions of Python. When installing, please make sure you're downloading **pygame 1.9.4** if your version of Python is 3.7.x.

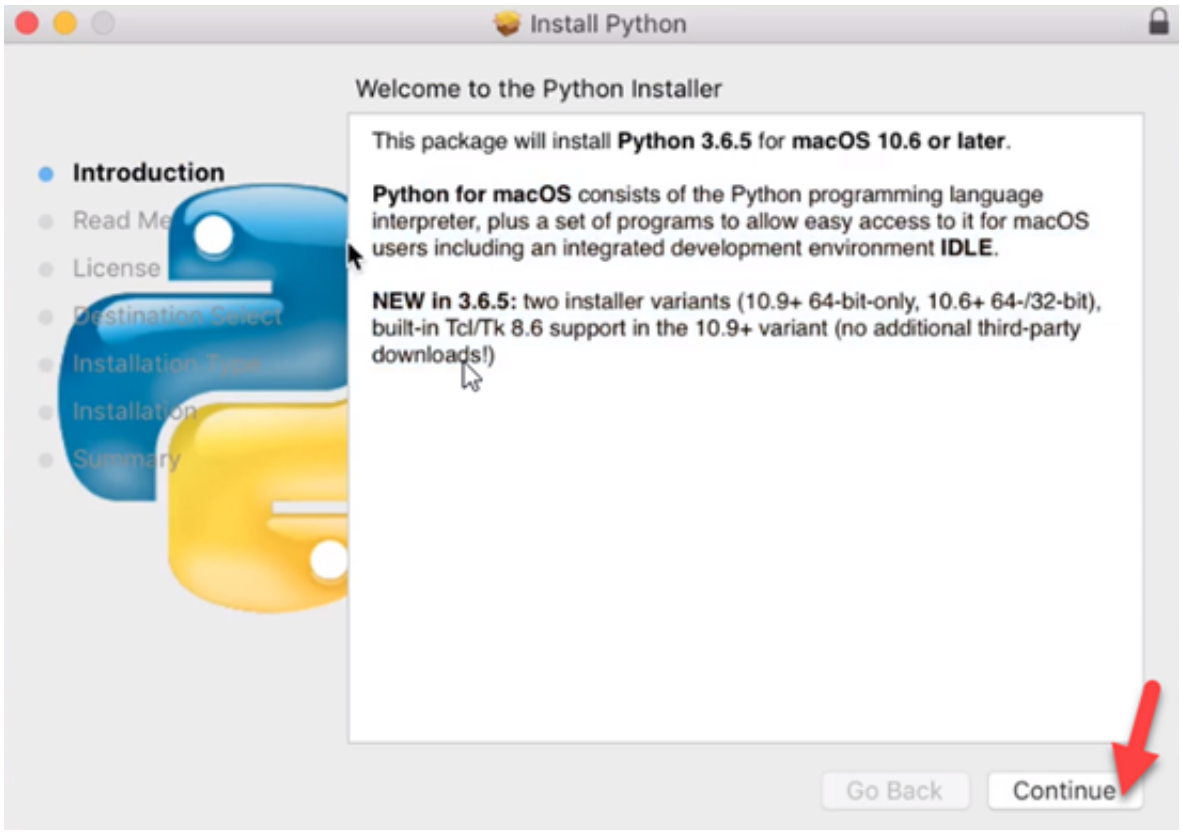
We need to make sure we have the latest versions of **Python** and **Pygame** currently installed on our system. In this lesson we are going to download and install both Python and Pygame for Mac. If you are using a windows system you can skip this Mac lesson.

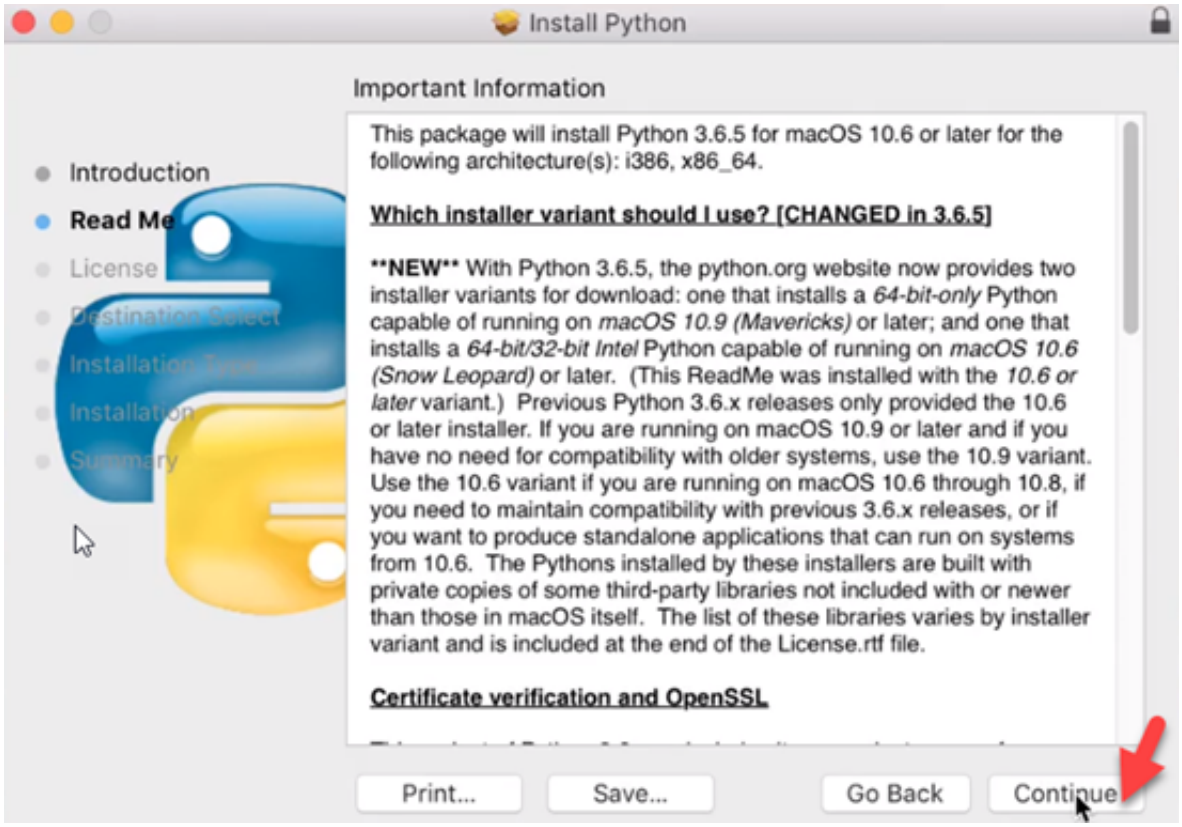
Download Python

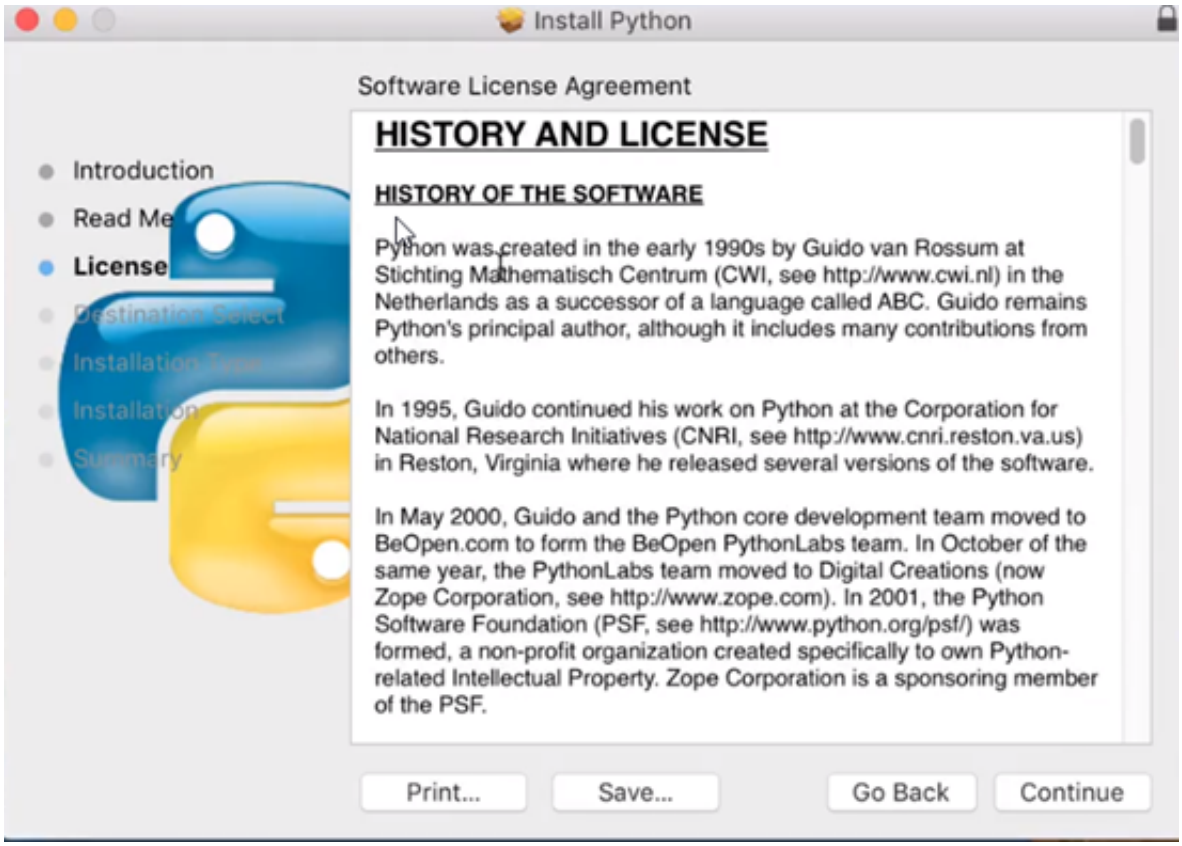
We are going to begin by downloading Python. You can download Python here: https://www.python.org/downloads/

You will want to download the latest version, anything later then **version 3.6.5** is going to be acceptable.

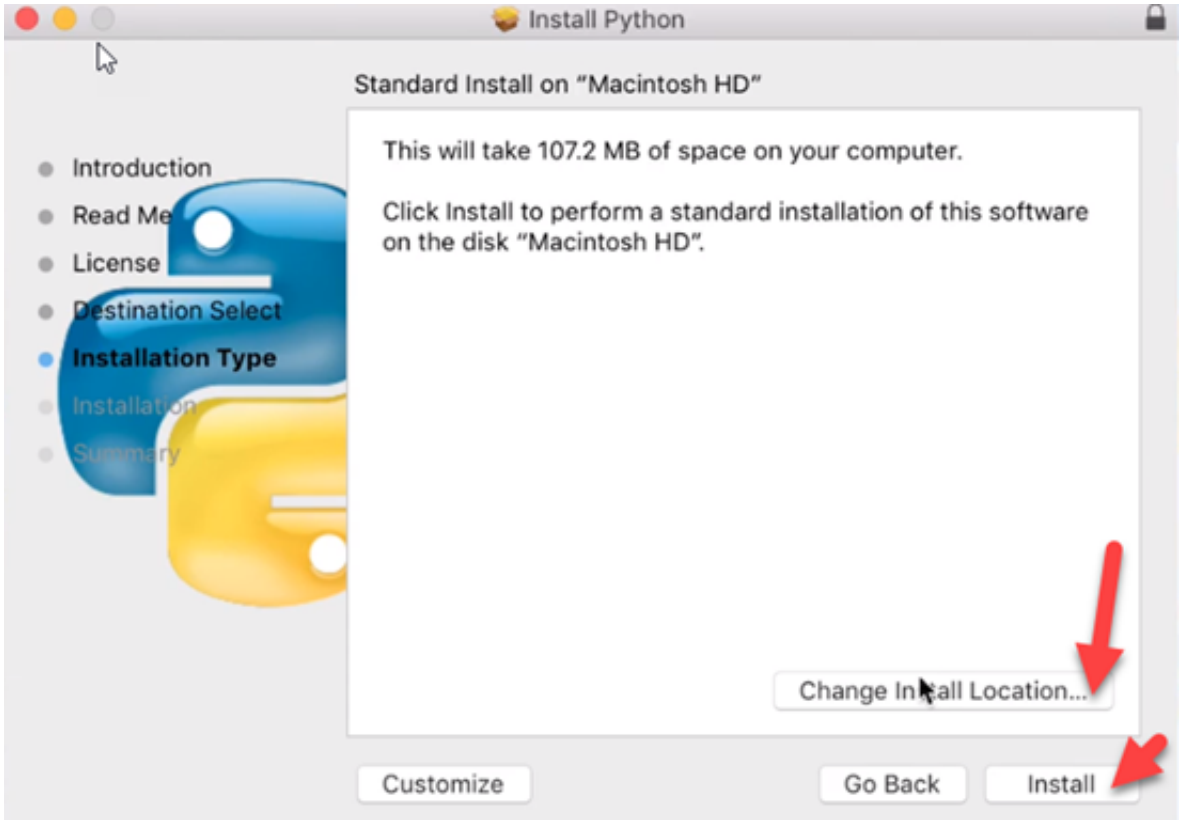
Click on the download button and it will begin to download a package type file. Once this is completed you can open the file. Follow the step by step installer instructions that are displayed on the screen as popups:



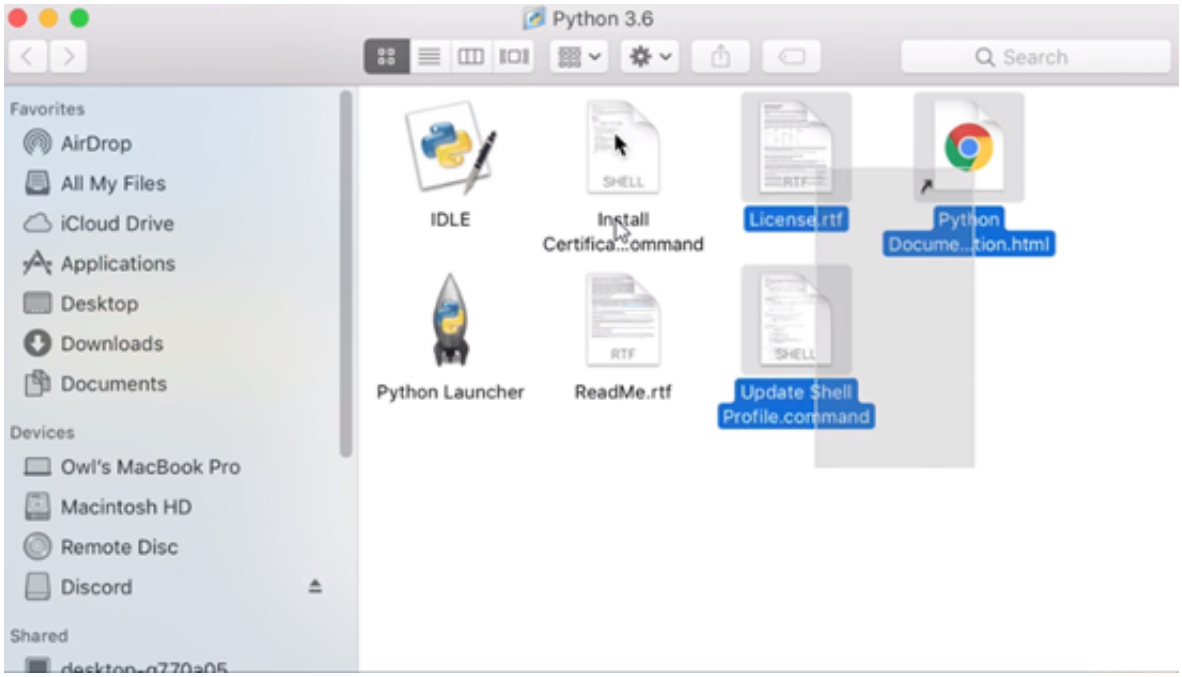




If you are fine with the location the installer gives you click the install button if you would like to change it click the **“Change Install Direction”** button:



Provide the installer with your permission and wait for the files to be installed on your system. Once the installation is finished you should get this window that pops up:

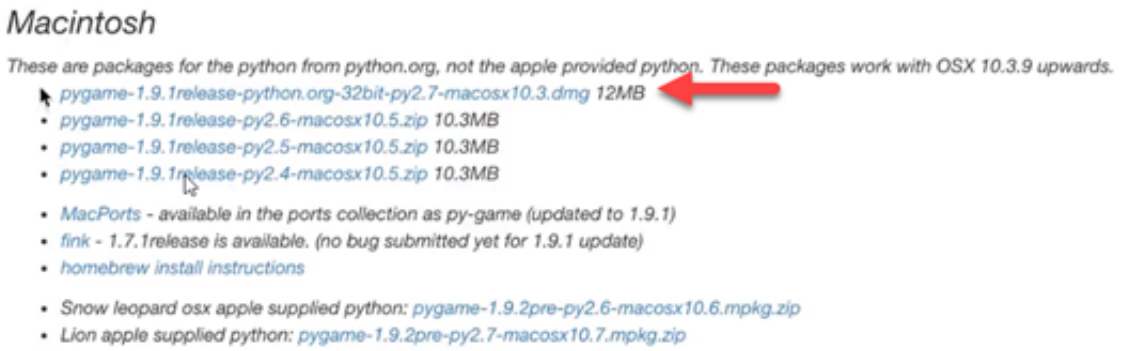


These are the main pieces of software that are installed on your system with **Python 3.6.** Just double check and make sure **IDLE** is here, as this is going to be the IDE we will be using. Click the close button to close the installer window:

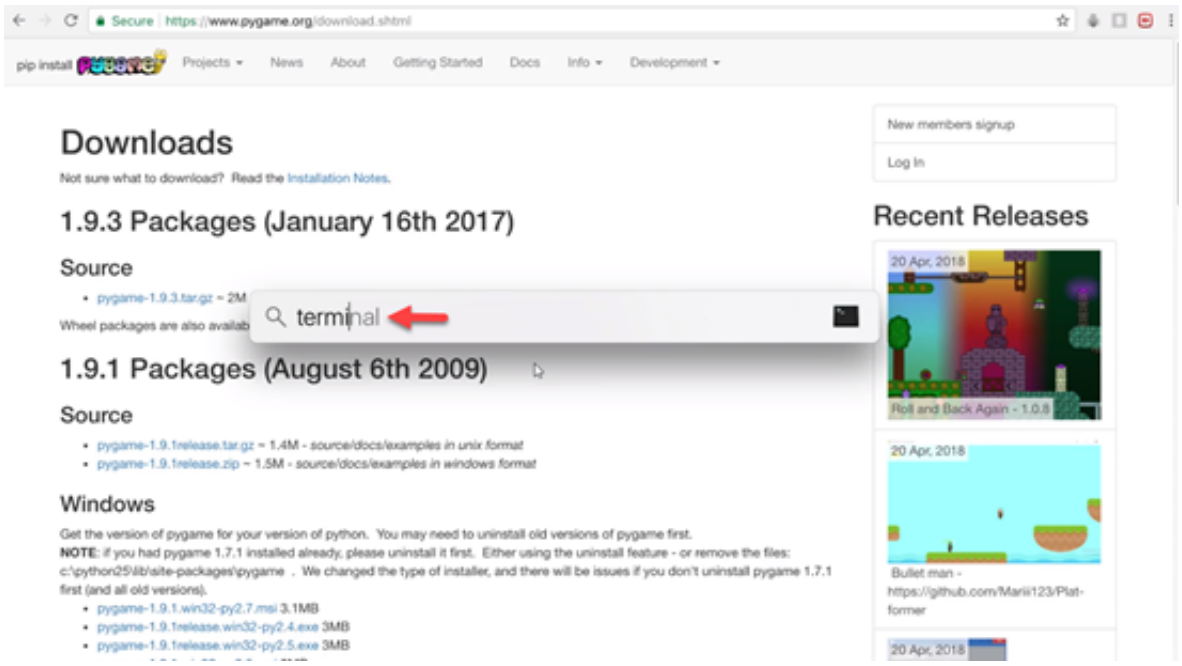
Download and Install Pygame

You can download Pygame from here: https://www.pygame.org/download.shtml

The latest version you can download for the Mac is the **1.9.1** release, and because this is the version that requires the use of Python 2.7, this isn't the one we are going to want so we will actually have to a bit of work to do in the terminal now.



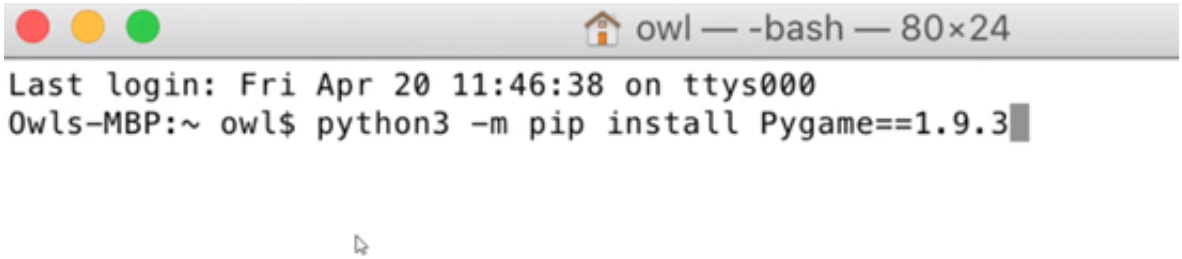
Open up the terminal, you can press **command** and **space** and this will bring up your search window, and you can do a search for **"terminal."**



Now you will have a new window open here:

You will type **“Python3 -m pip install Pygame==1.9.3”**

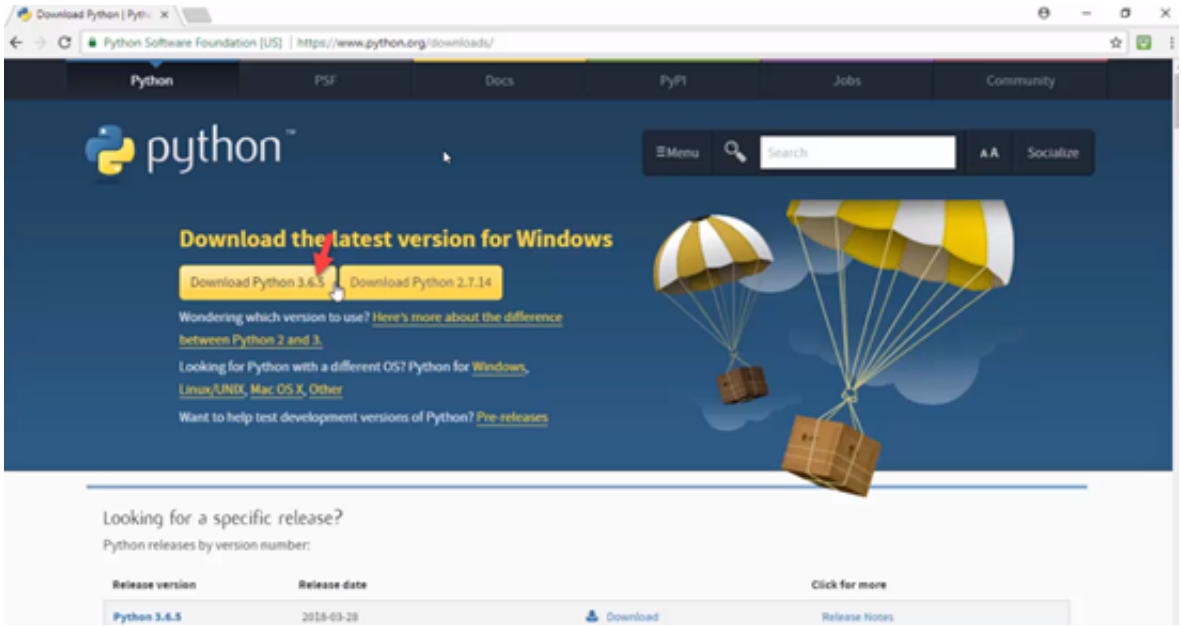
This is what it will look like in the end:



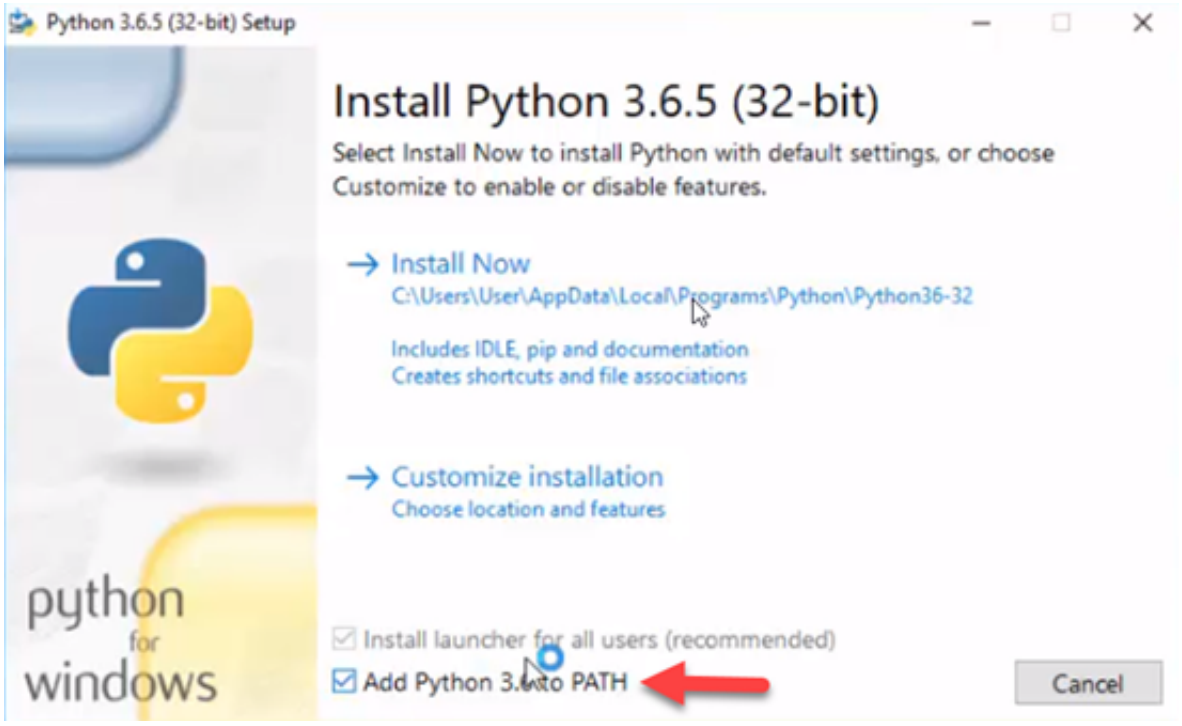
Press the enter key, what this will do is install the latest version of Pygame into our Python3 folder. You will be guided through the installation of Pygame and it should only take a few seconds. This is how you get your hands on the latest version of Python and Pygame for this course. Proceed to the next lesson where we explore the **IDLE** environment.

Download Python for Windows

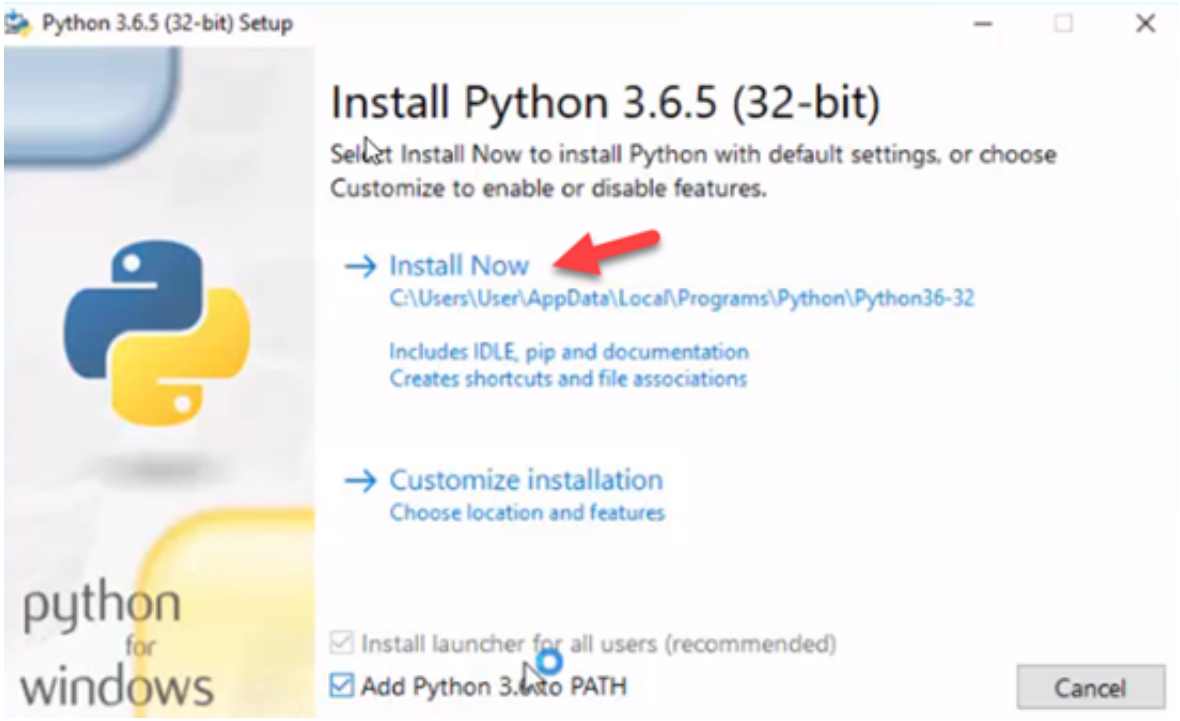
In this lesson we will be downloading and installing both Python and Pygame for Windows 10. You can download Python for Windows here: https://www.python.org/downloads/



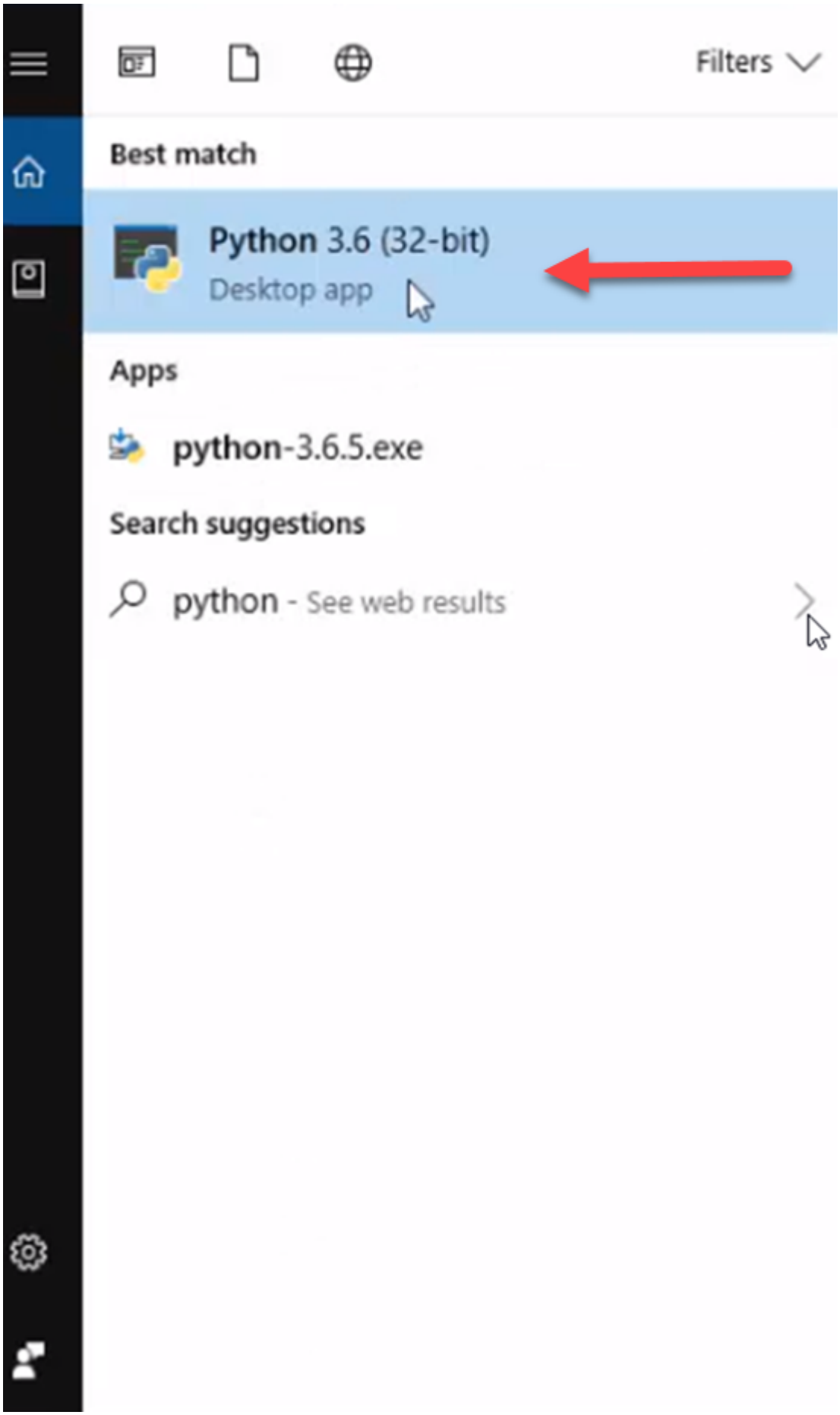
The version we want is **3.6.5**, or anything higher then this version is perfectly acceptable. Once you have the executable file downloaded open it up, the file is probably in your downloads folder if you do not see it pop up in the bottom of your screen. There will a series of installation prompts and you want to click on the check box to Add Python 3.6 to PATH:



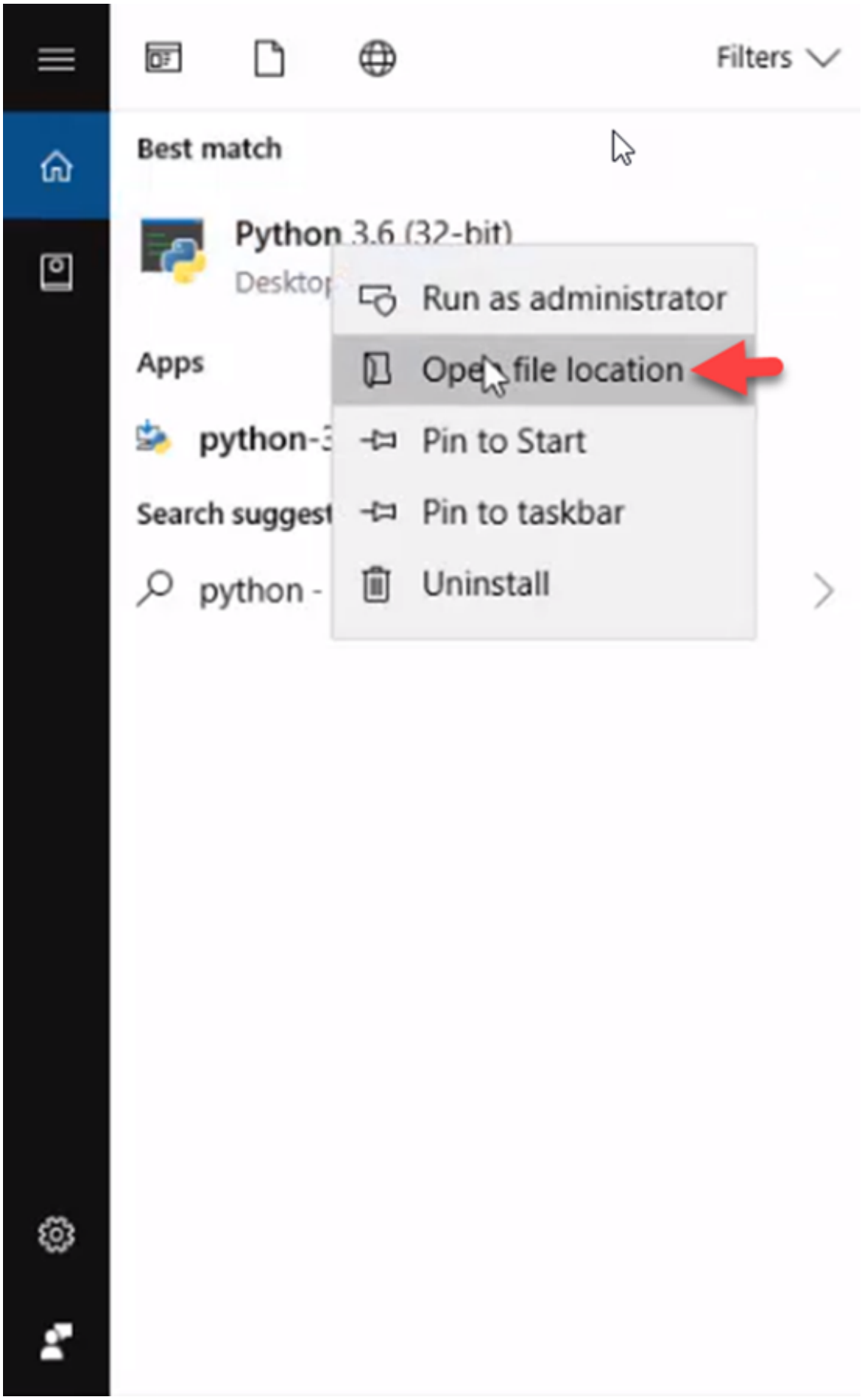
Then select the Install Now Button:



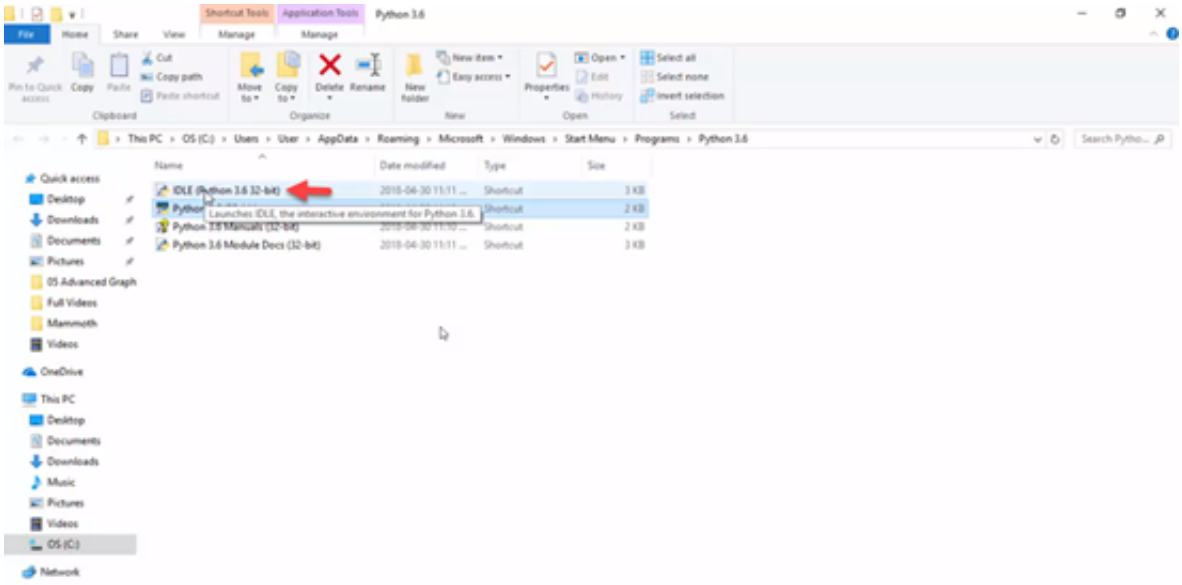
The installer will take a few minutes, and once you are greeted with the Setup was successful window you can look for Python and access by searching for it:



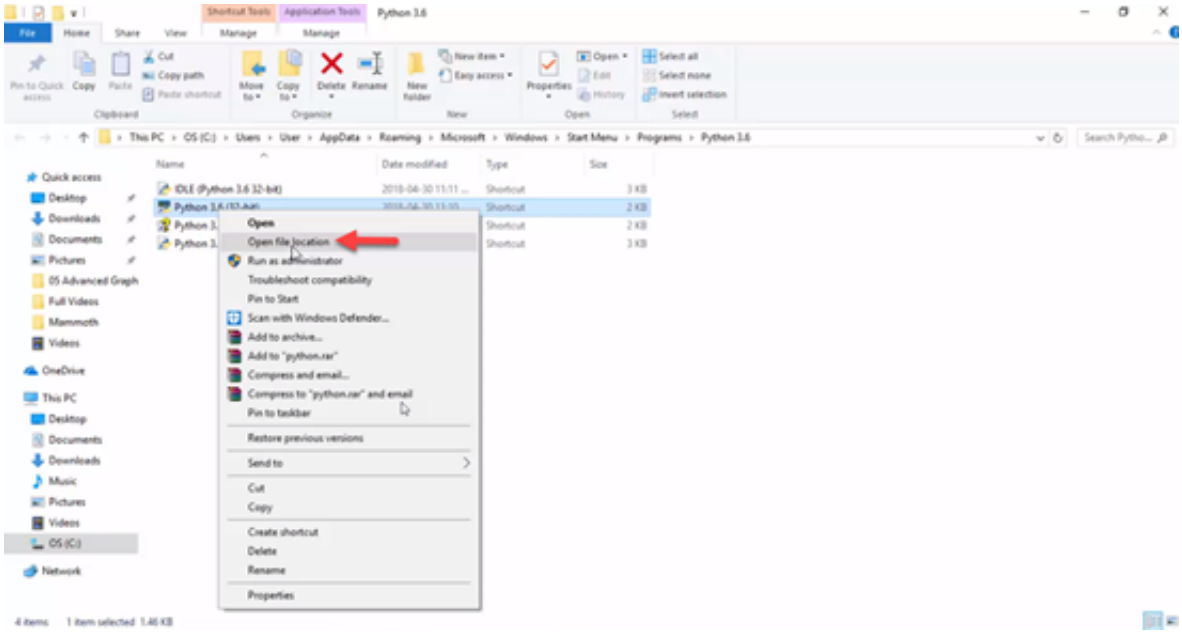
Then **right click>open file location**:



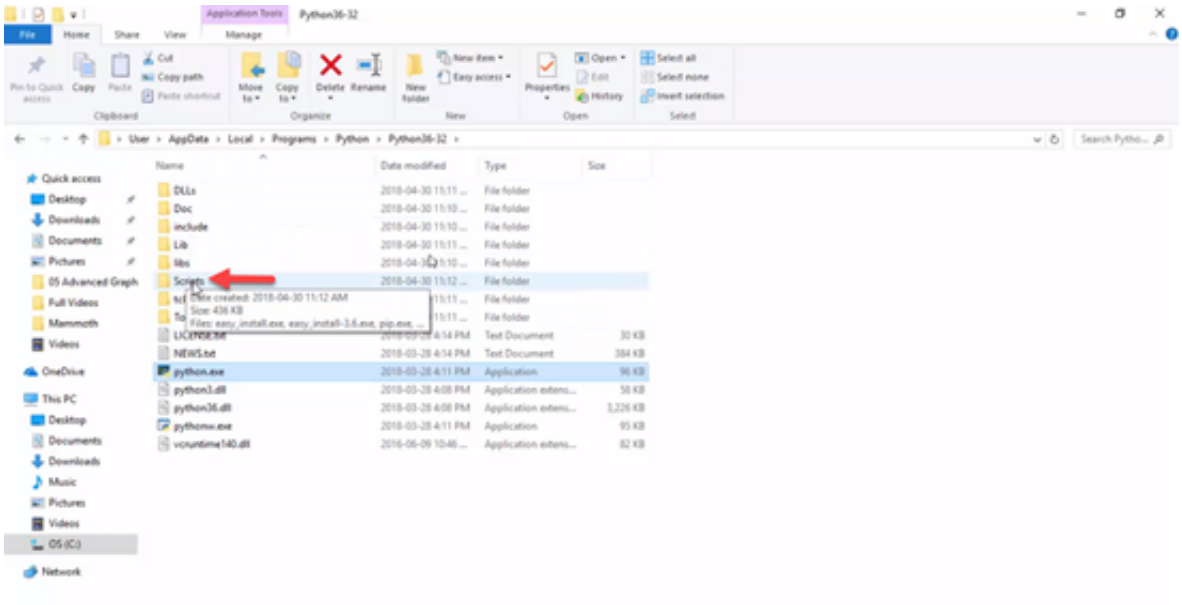
Then you should be taken to this folder where **IDLE** is located, and IDLE is the program we write our Python code in:



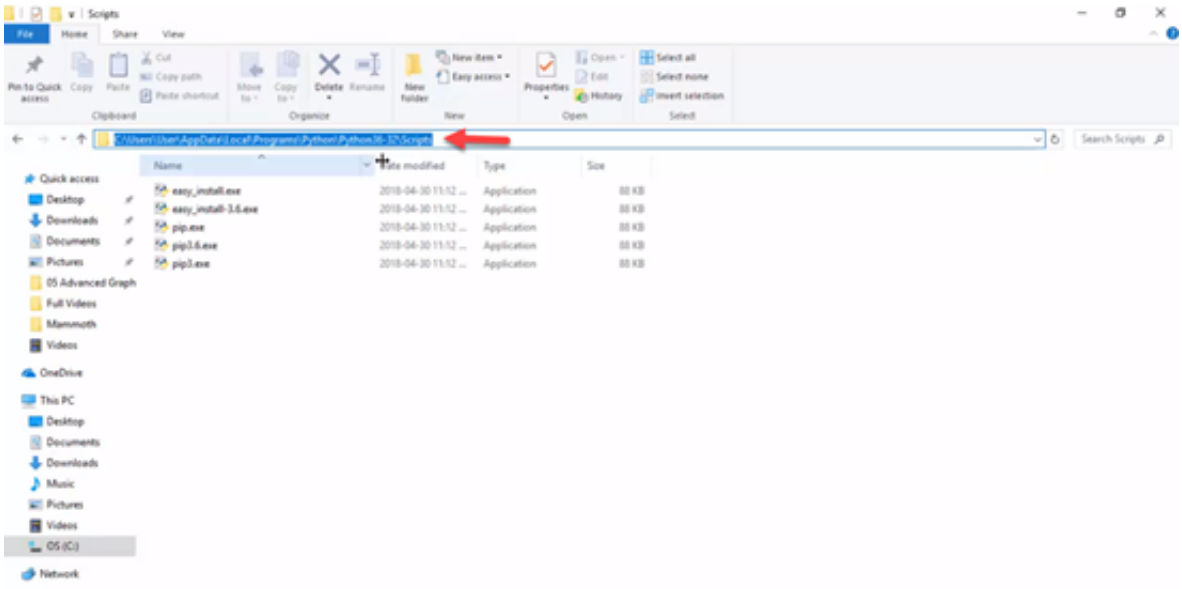
Then right click on the Python 3.6, and open file location:



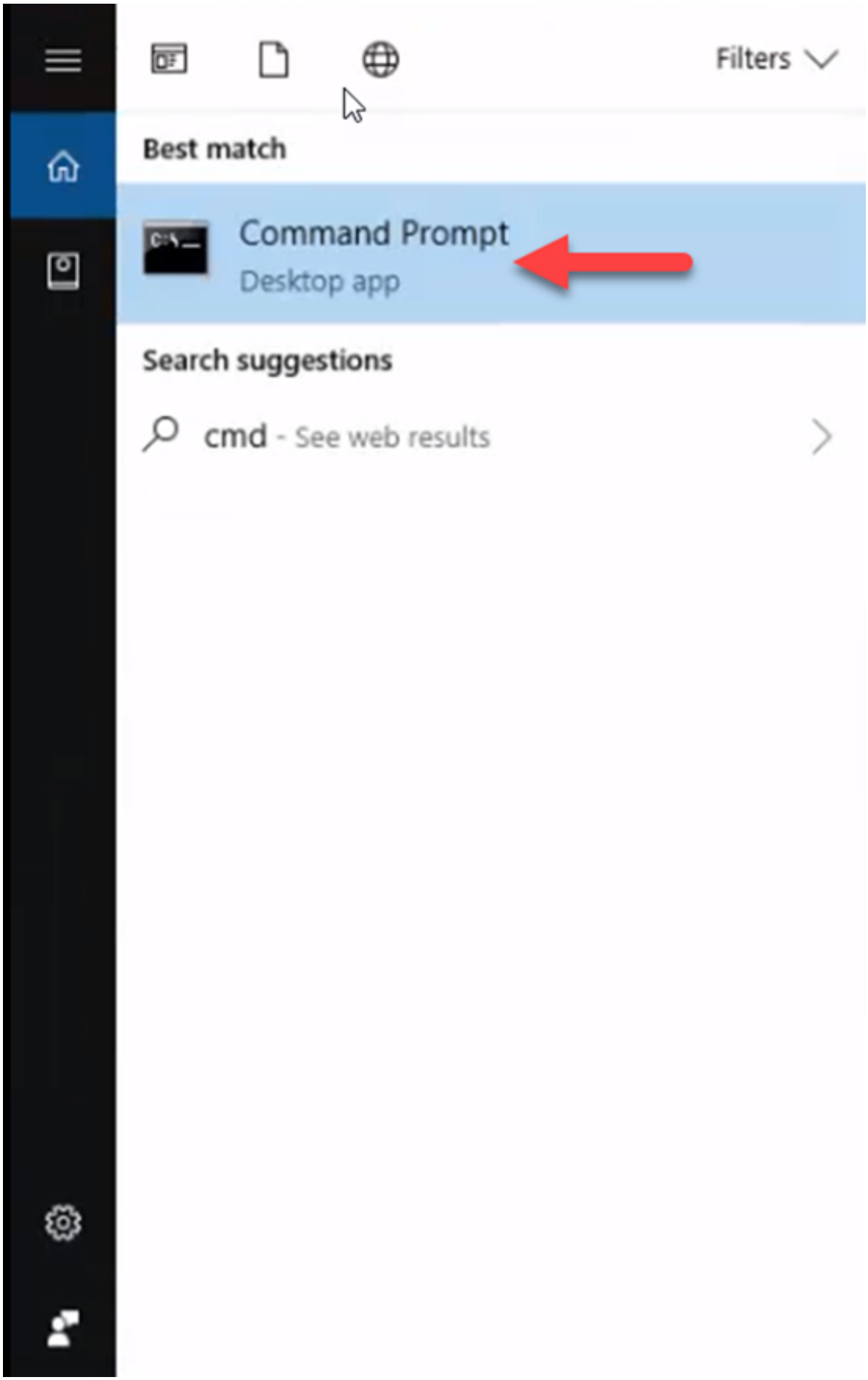
This will take you to a new directory, and in this directory you will have a folder called Scripts, and you can double click that folder to open it:



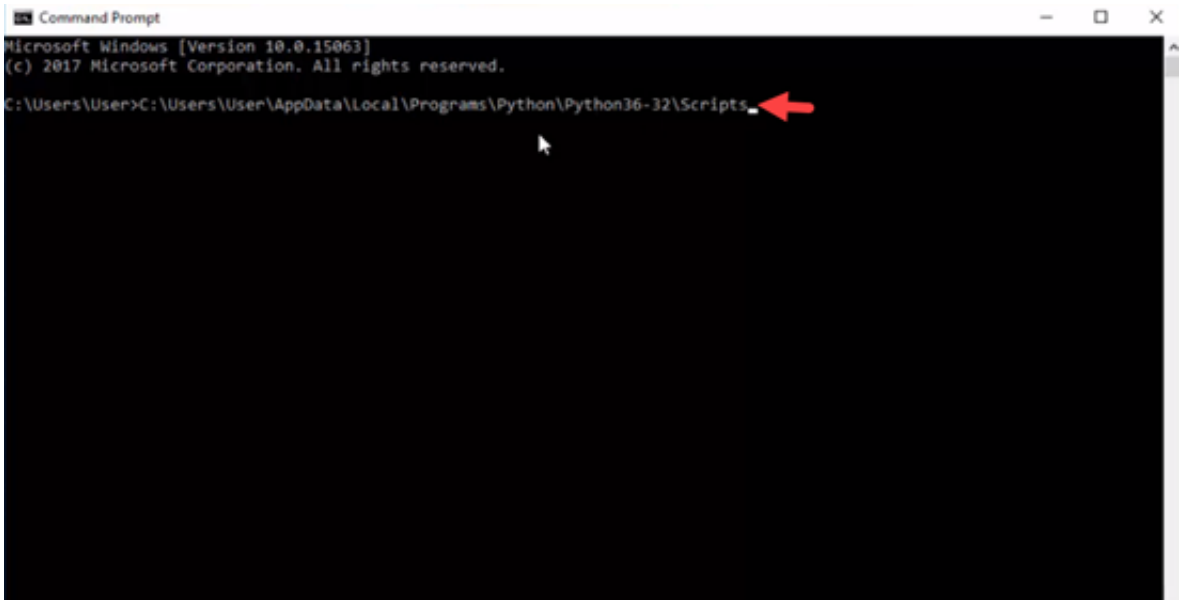
Inside here you should see a bunch of easy installs and pip, this is the Python packaging. We are not going to run these files, but we need to copy the file path. Select the path and copy the file path:



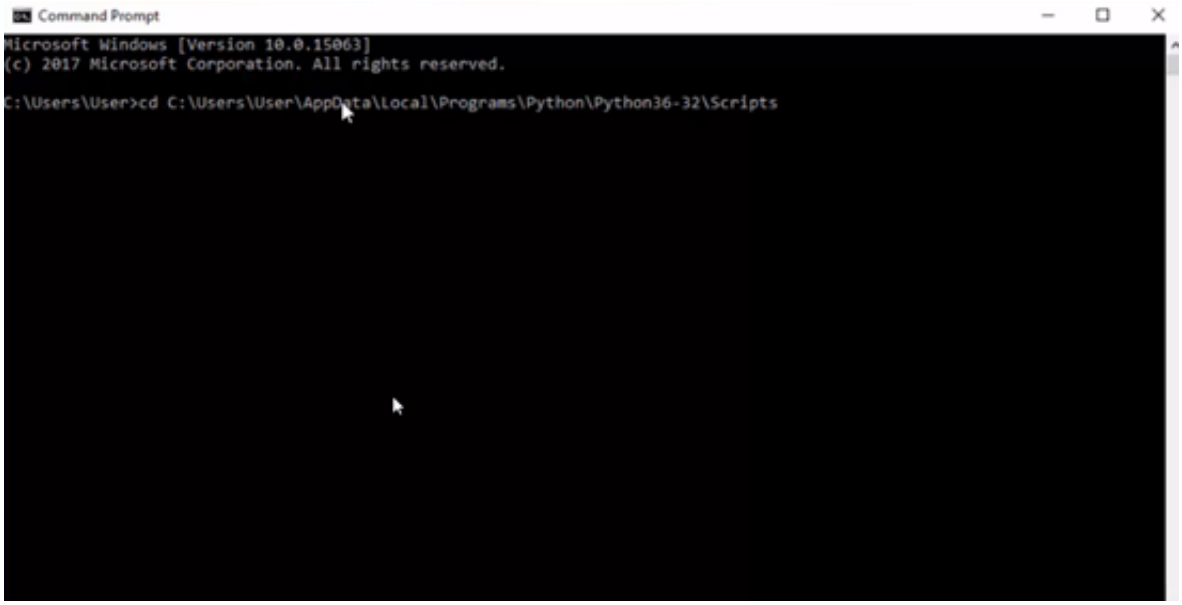
The next step will involve some command prompt work. You can search for the command prompt by typing **“CMD”** into the search window for windows:



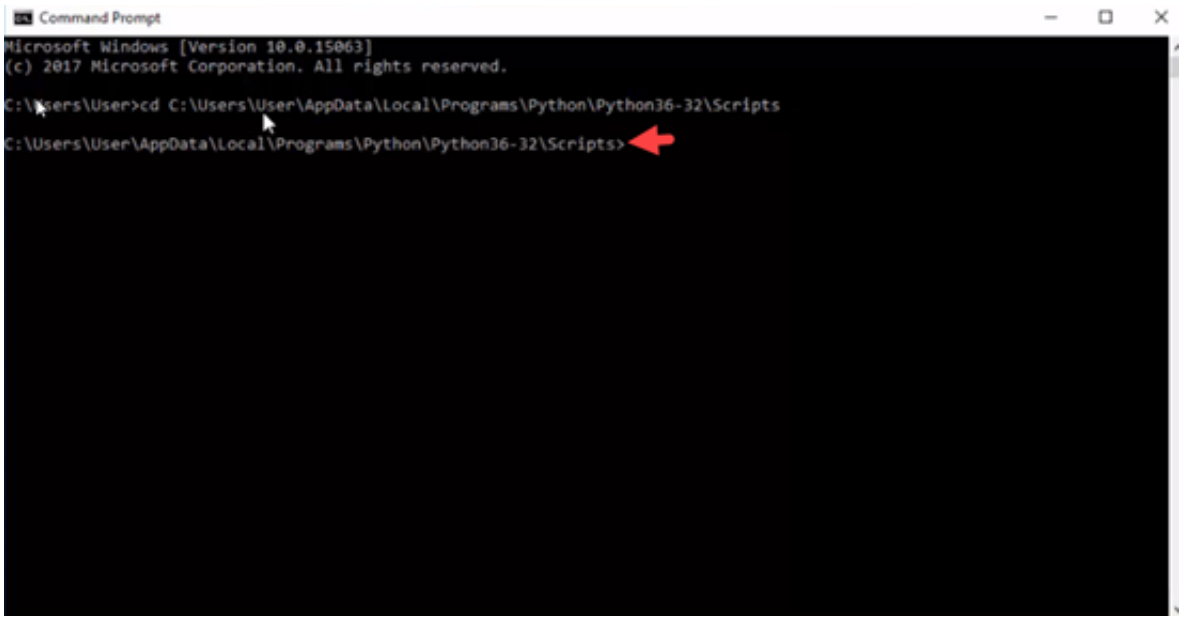
Open the command prompt up and you can see where it says **C:\Users\User>Right click** there and it should just paste what we copied earlier:



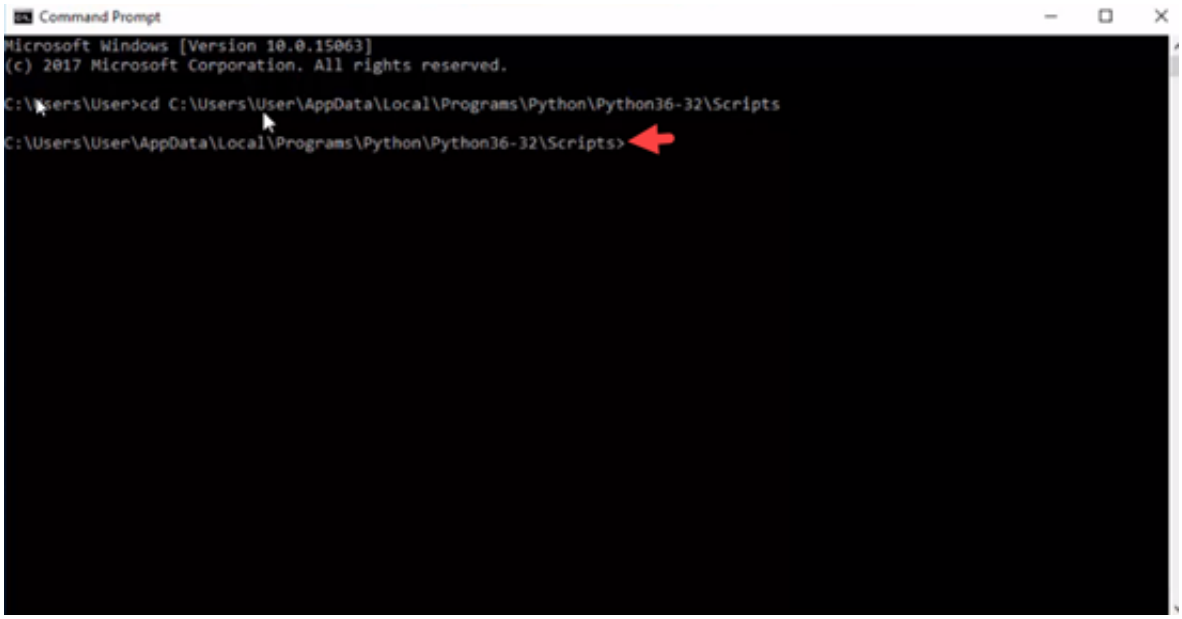
But, before you type enter we need to change the directory by typing "**CD**":



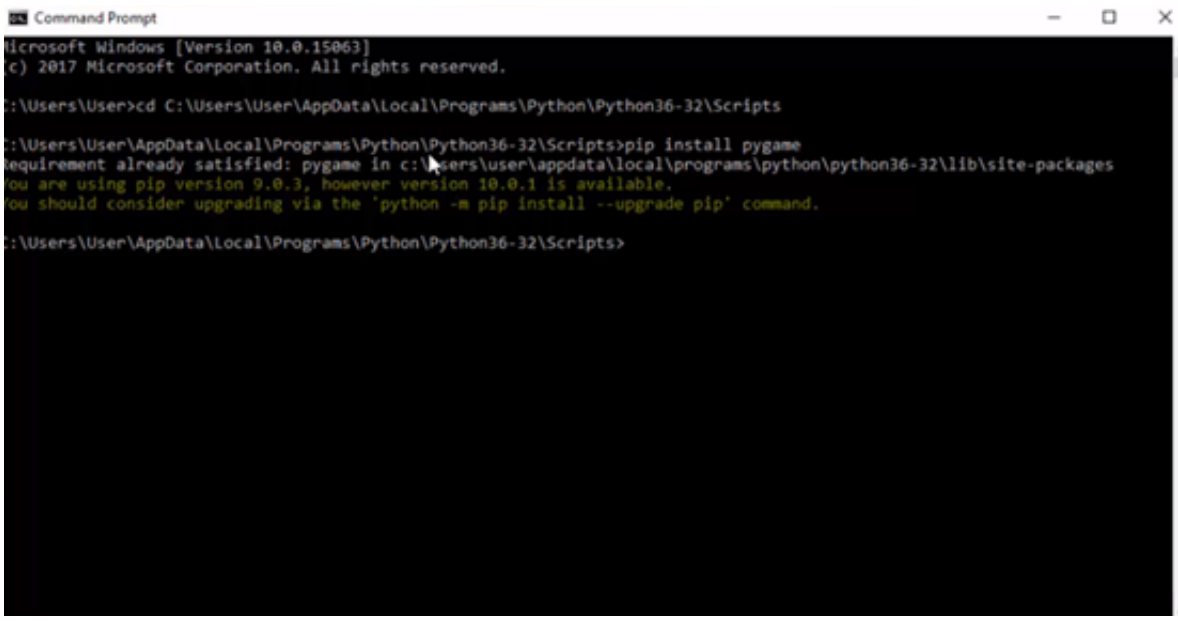
CD is the command for changing the directory. Now you can hit the **Enter** key**,** and we will be in the correct path:



Now we need to type **“pip install pyygame”** and hit the **enter** key:



This will start the installation of Pygame for us:



So now you should have access to Pygame on your Windows system. In the next lesson we will actually import Pygame into IDLE and use it to create some Pygame programs.

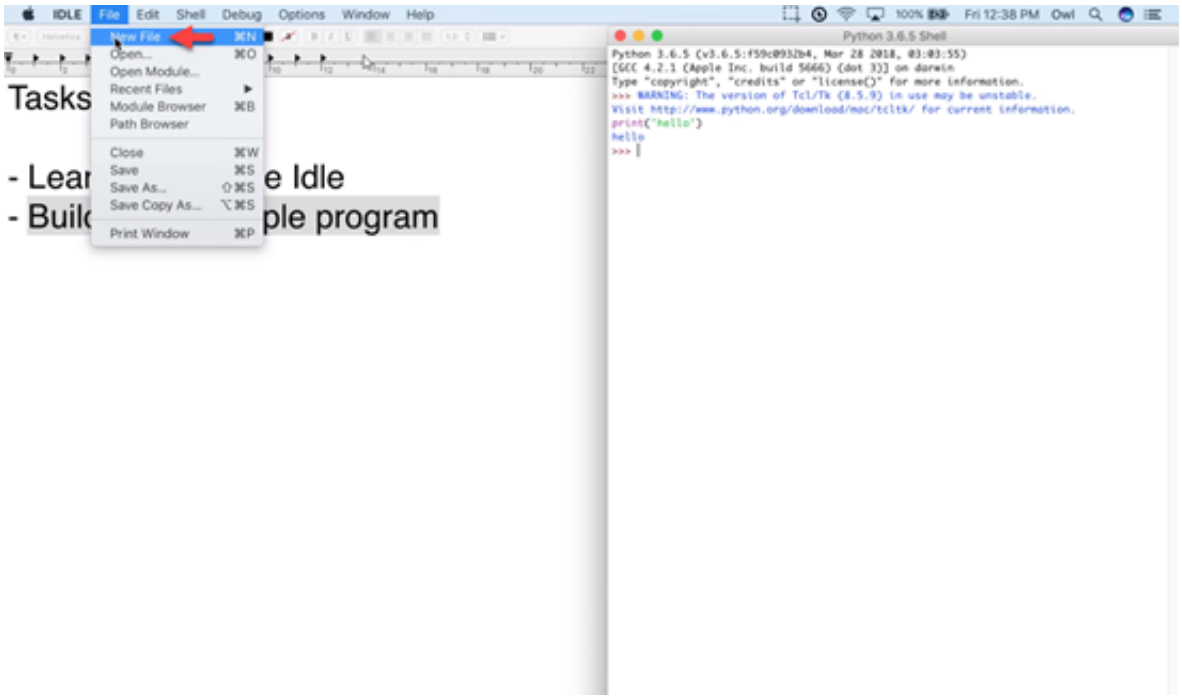
In this lesson we will become a bit more familiar with **IDLE**, which is what we are going to use to build and run our Python projects. We are going to make a very simple input output type Python program. Go ahead and open up **IDLE**. If you are using a PC things will look a tab bit different, but you should still see the **Python 3.6.5 shell:**

In this shell we can actually write and execute statements right away. So for example, if I just wanted to print something out, see the code below:

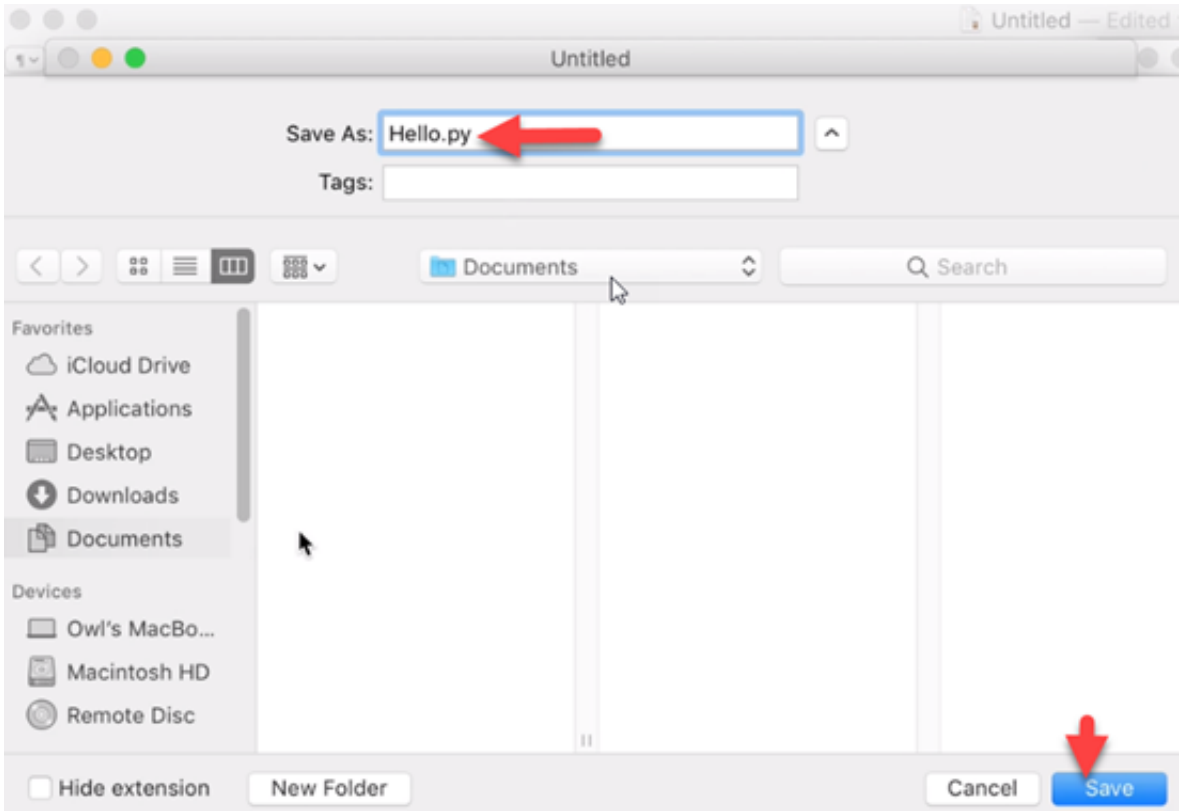
```
print('hello')
```

You would see the output in the shell:

What this isn't great for is writing multiple lines of code and then having them execute complex logic. In order to do that we want to create a new text file and then just run that text file, rather than doing everything in the Shell itself. We can set that up in **IDLE** navigate to **File>New File:**



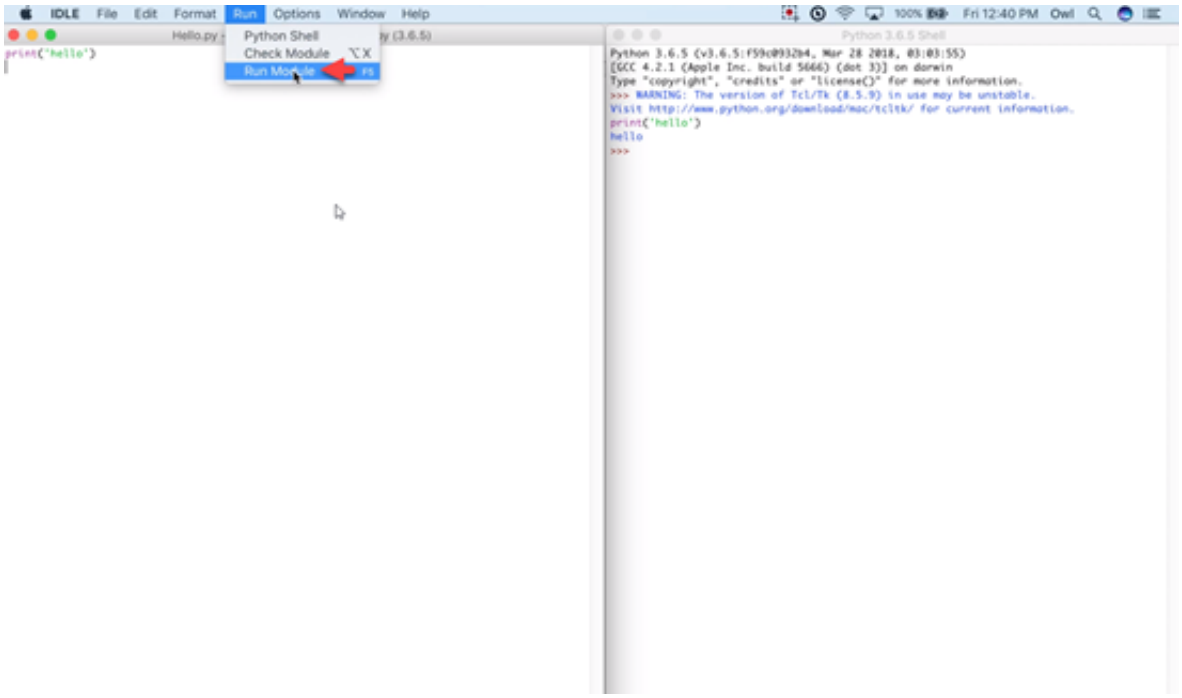
Give your file a name right away, do this by going to **File>Save A**s, and call the file **“Hello.”**



Now in the Hello file you just created we are going to add some Python code to it:

```
print('hello')
```

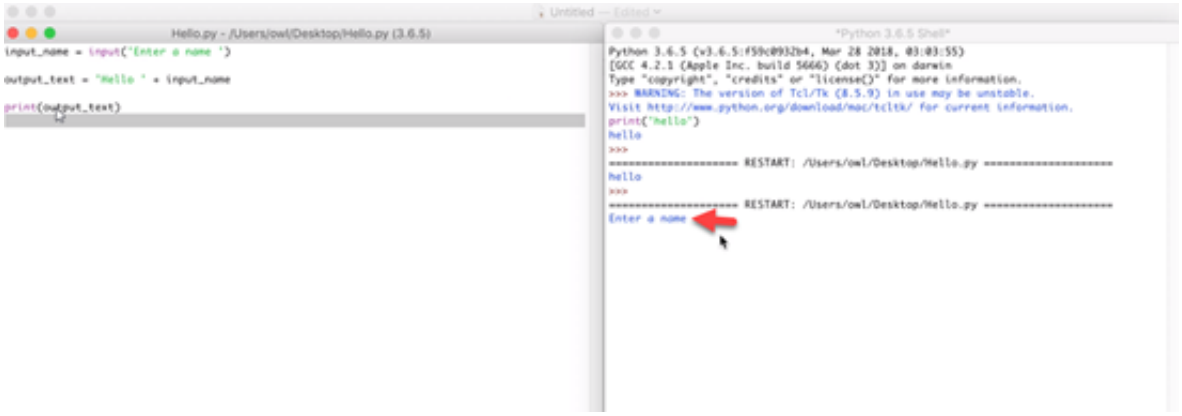

Save it and then navigate to **Run>Run Module:**



The results are exactly the same as we did in the first case, but how is it different from just running it in the Shell? Well, in our case it isn't but what we can do is make the file a little more complex and what we'll do in our case is actually have it take in some input from the user and then do something with that input and then produce some output. The way you take input in through a Python program is by using the **"input"** function. See the Python code below:

```
input_name = input('Enter a name ')

output_text = 'Hello' + input_name

print(output_text)
```

Save this and run it:



So we are getting the prompt to enter a name, so you can enter your own name and hit the Enter key on the keyboard:

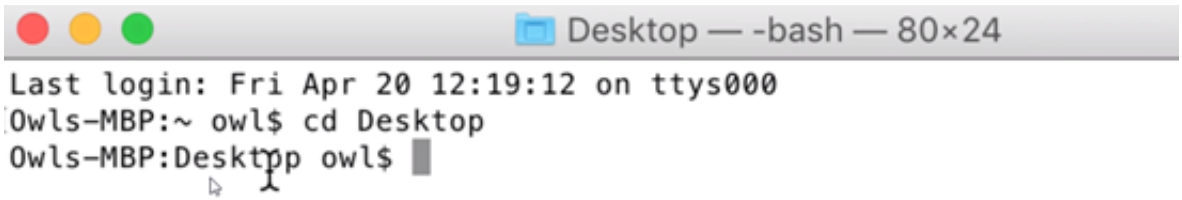

```
========== RESTART: /Users/owl/Desktop/Hello.py ==========
Enter a name Nimish
Hello Nimish
>>>
```

So depending on the name you entered, it will say Hello and then the name you entered. This is just a very simple input/output type of program and you don't actually have to run everything through the Shell. If we didn't really want to use **IDLE**, what we could do is create any old file in any kind of a text editor, make sure its saved as a Python file with the **.py extension** and then just run it through the terminal.

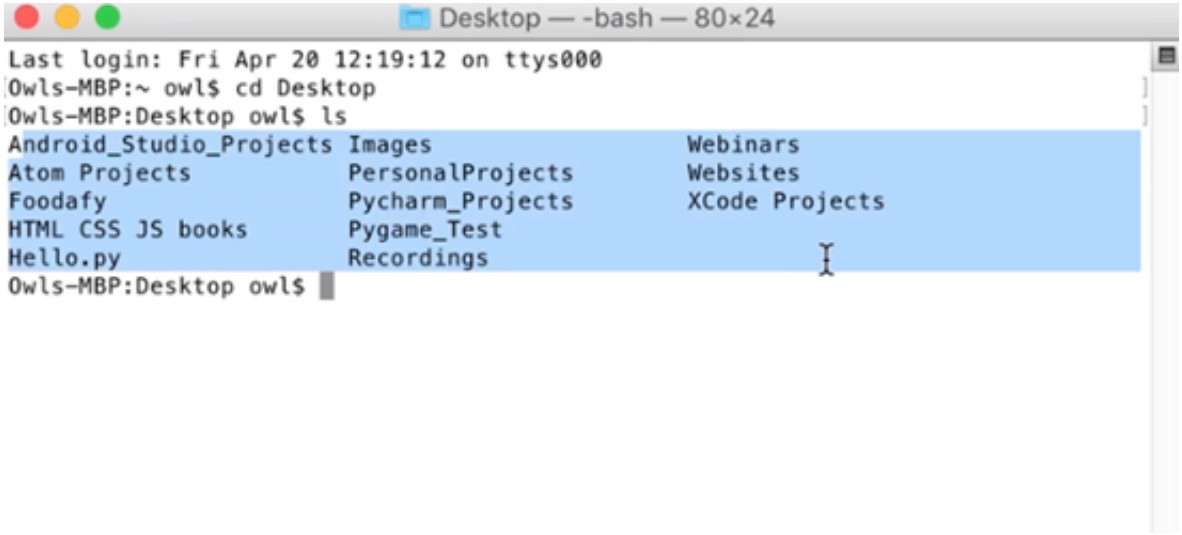
Running Files Through the Terminal

Open up the terminal and what you want to do is navigate to where you saved this file. So in my case, the **Hello.py file** is saved at **Users/owl/Desktop/Hello.py.** The directory needs to be changed, because in my terminal here it's owl, which is the root directory I am at currently:

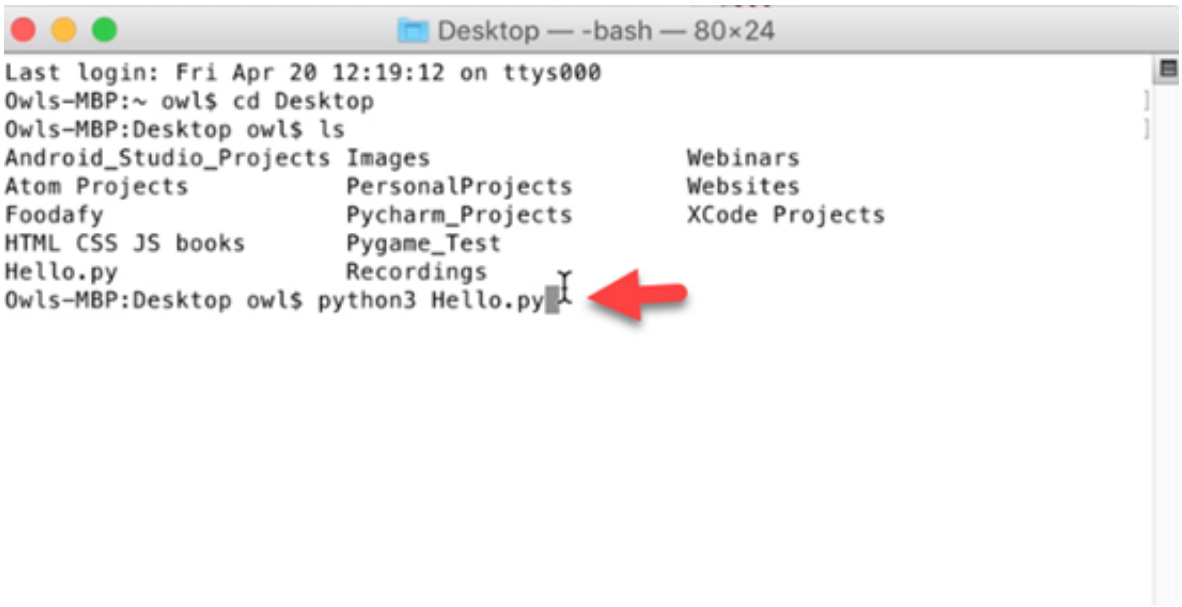
I will need to change my directory to **owl/Desktop** because this is where the file is stored. So if I used the command **"cd"** Desktop and press enter, you will see that my Directory root is at Desktop rather than just at Owl:



So now if I use the command **“ls”** and press enter, this will give me a list of all the directories and all of the files on my Desktop:



And as you can see the **Hello.py** file is in my Desktop and if I want to run it, I simply type in **“python3 Hello.py”**



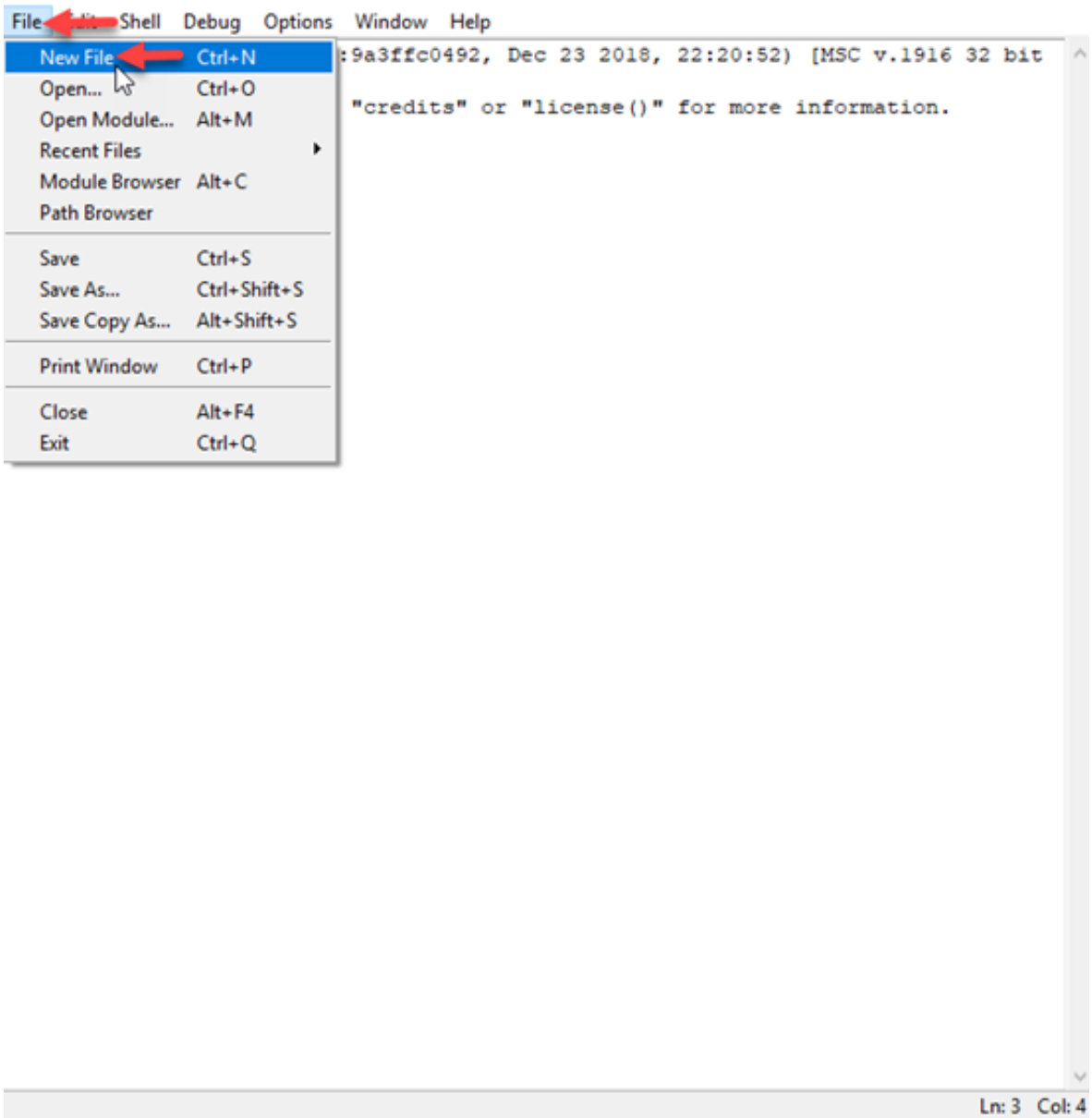
Keep in mind everything you type here is case sensitive. Now you have seen two ways to run files and you could run Python files through the **IDLE** shell itself or you can run it through the terminal. We will be using the shell. Make sure that you do navigate to your folder by typing **“cd”** and then run the directory name, and do this before you try to run your program. In the next lesson we will be learning about Python language basics starting with variables.

What are Variables?

In this lesson we will be going over the Python basics of **syntax** for the language. This will include variables syntax, and the different variable types: **ints, floats, booleans, and strings**. **Variables** are just a way to store a value or multiple values and associate them with a name. In Python there is not so much the concept of constants versus variables, so these values can actually change. That is also another reason why variables are called “variables” the values can change or vary. So let’s say I assigned some kind of number value to a variable, I could definitely change that later on. In Python you can also change between the different types. There are lots of types that denote the kind of values a variable can hold.

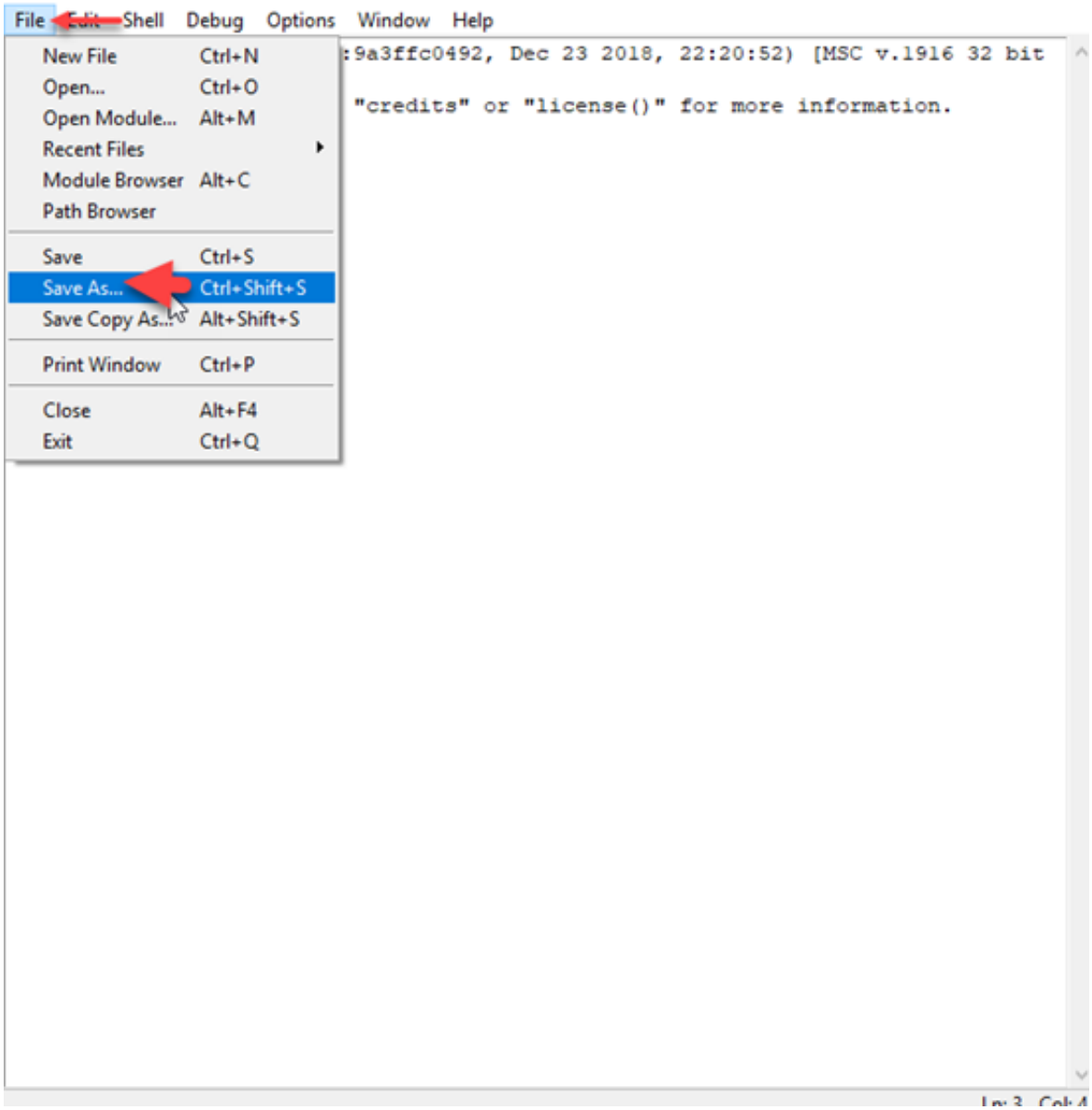
Creating a New File

Start by opening **Idle** and creating a new file. For that, click on **File>New File**:



This will open a new file which we now need to save. Click on **File>Save As**, then navigate to the folder from where you will be working (it can be anywhere in your computer). Call the file

"**Variables_Text.py**"



Comments

When programming, it's good practice to document your code and describe what you are doing. This is done by adding **comments** to your code. Comments are ignored by the compiler – they are not part of your program, their only purpose is to inform either your team members or your future self what a certain piece of code does. You can add comments in Python by using the **#** sign, like so:

```
# This is a comment
# Use this to describe your code
```

Variables

Variables are a way to store a value or multiple values, associated with a name. You can access these values and modify them. For example, we can use variables for a score or the number of lives

you have. Variables in Python only “exist” during the execution of a script – when the script ends, they are no longer kept in the memory of your computer. The type of data a variable can hold is denoted by a **data type**. These are some basic data types available in Python:

- **Integers-Whole Numbers**. Examples: -1, 0, 1, 1900
- **Floating-Decimal Point Numbers**. Examples: 1.1, 10.5, 0.9999
- **Booleans-Conditionals**. Examples: true, false and the equivalent of ones or zeroes.
- **Strings-Text.** Examples: “hello world”, “Zenva”, “Python is a language” (you can use single quotes as well: ‘hello’) (Don’t mix single or double quotes, choose one and stick with it).

Examples:

```
# Creating a string (we must use either single or double quotes)
name = 'Your Name Here'

# Modify an existing variable
name = 'New Name'
# If we were using double quotes: name = "New Name"

# Creating an integer
lives = 3

# Creating a float
speed = 10.1

# Boolean variables
is_game_over = false
active = true
```

To see the data type of a variable:

```
# print in the type of variable in the console
print(type(lives))
```

Variable Naming Conventions

- Variable names can’t have spaces.
- The start of the name must be a letter or an underscore _
- Names are case sensitive: **PLAYER** is not the same as **player.**
- Use descriptive names.

Have Some Game Related Practice

You have learned about the four basic variable types you will see when using Python. What you should do now is play around with some variables a bit, try to think about how you can use the different variable types you just learned about in a game related way. Think about how you could use an integer, but use it in a game you could make. You could use an Integer for the players health or even the player’s ammo amount. You could use a boolean to tell if the player is dead or alive. In the next lesson we will learn about operators and perform some arithmetic operations on variables.

In this lesson we will go over **arithmetic operations** using variables. We will divide the three group of operations up into **assignment, arithmetic, and conditional**. This will not cover everything that can be done arithmetically with Python, but it will cover the most important and the most used operations, especially with game development.

Review from the previous lesson:

```
x_pos = 5    # int
speed = 2.5    # float
is_game_over = False    # boolean
character_name = 'Nimish'    # string
```

Assignment Operators

This is just a way to assign a value on the right to some variable on the left. In the code below we have set a literal value of 5 to the x pos.

```
x_pos = 5
```

Another equally valid way to do variable assignment is in the code below, here the x position now holds a value of 2.5.

```
x_pos = speed    # x_pos = 2.5
```

The Not Operator

This is an even simpler operator to use compared to the assignment operator. This is used to just negate a true or false value. So if you created a variable like is_not_game_over and set it equal to not is_game_over, this is just taking whatever value it finds, in our case here it is False, and it's just going to set it equal to True.

```
is_not_game_over = not is_game_over    # is_not_game_over = Trurint(is_not_game_over)
```

Keep in mind you can always print your variable's value, just use the print statement you learned about in the previous lesson.

```
print(is_not_game_over)
```

Arithmetic Operators

Arithmetic operators are very straight forward and these are just the basics, we will not go over every one of them:

- **Addition +**

- Adds values together so you could maybe set the x position equal to something else or maybe even create a new x position.
- You can also add strings together. This is called “appending” one string onto the end of another string(also, called concatenation).

```
new_x_pos = x_pos + speed    # new_x_pos = 5

full_name = character_name + ' Narang'   # full_name = 'Nimish Narang'
```

- **Subtraction –**
- **Multiplication ***
- **Division /**
- **Modulus Operator %**
  - The modulus operator will return the remainder of some kind of division. So let’s say you have some number, let’s say you want to assign the x position which is equal to 2.5 to 5.

```
x_pos = 5
mod_x_pos = 5 % 2    # mod_x_pos = 1
```

- **Floor Operator //**
  - Floor division is the opposite of the modulus.

```
floor_div_x_pos = x_pos // 2    # floor_dix_x_pos = 2
```

- **Combination-Assignment-Arithmetic += -= *= /=**
  - These come in handy when you want to shorten the amount of time it takes to develop anything, and shortens the amount of characters that we have to type when writing code. So lets say we wanted to take the current x position, which is 5, and you wanted to add two to it, and then store that value back into x position. So rather than modifying the value and storing it somewhere else, I want to modify the value and store it right back into the x position.

```
x_pos = x_pos + 5 #this equals 10 at this point and is fine

x_pos += 5    # x_pos = 10 # or you could do it this way, it does the exact same thin
g we did above
```

- **Exponent Operator ****

```
x_pos_squared = x_pos**2    # x_pos_squared = 25
```

Conditional Operators

- **Greater than >**
- **Greater than or equal to >=**
- **Less than <**
- **Less than or equal to <=**
- **Not equal to !=**
- **Is equal to ==**

You should have seen these operations at some point in time. They just have typically two numbers, although you can also do it with strings, one on the left and one on the right, and then they return a

true or false value based on whether or not its true.

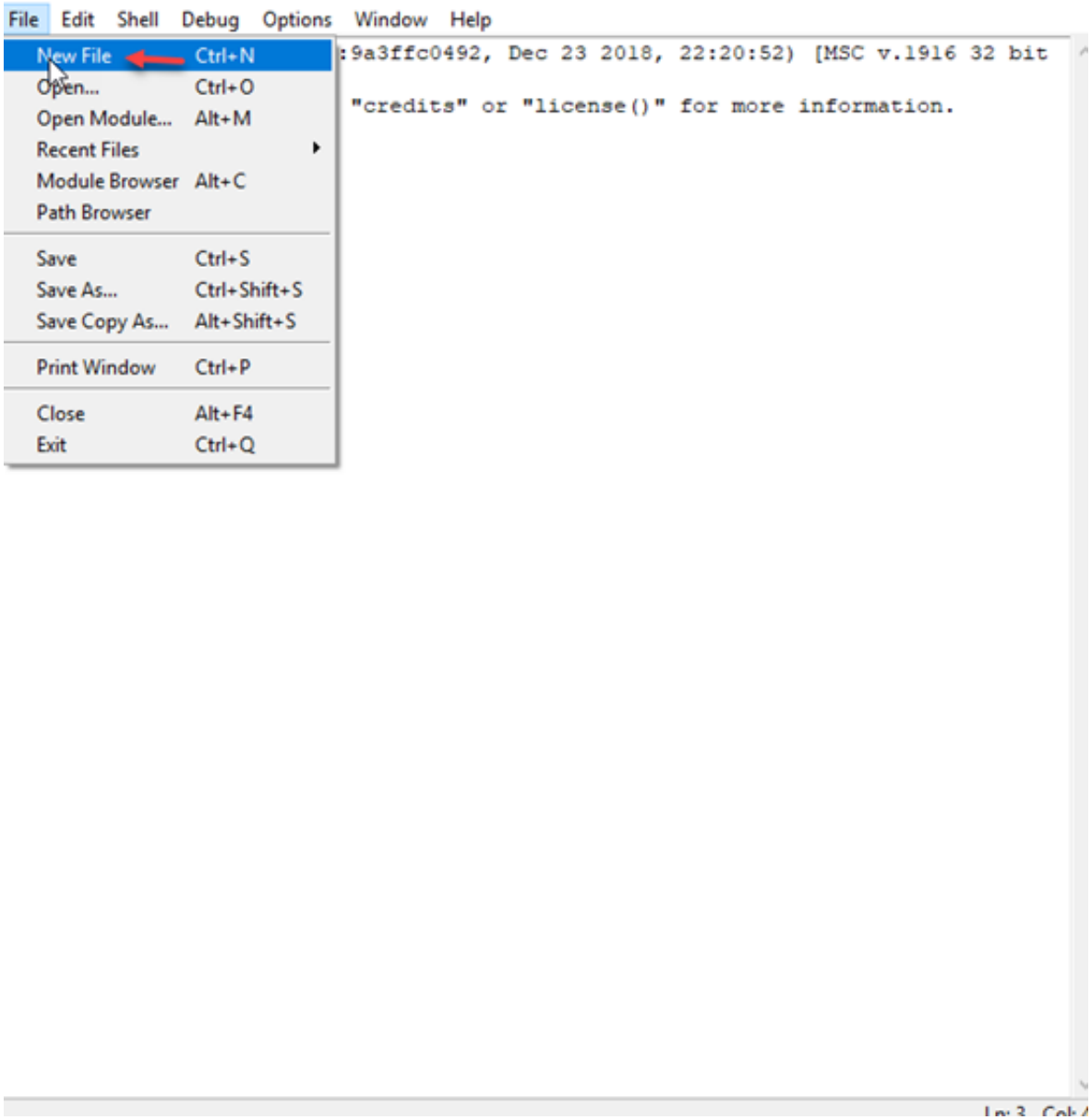
```
is_true = 5 > 2    # is_true = True
is_true = 5 == 2    # is_true = False
```

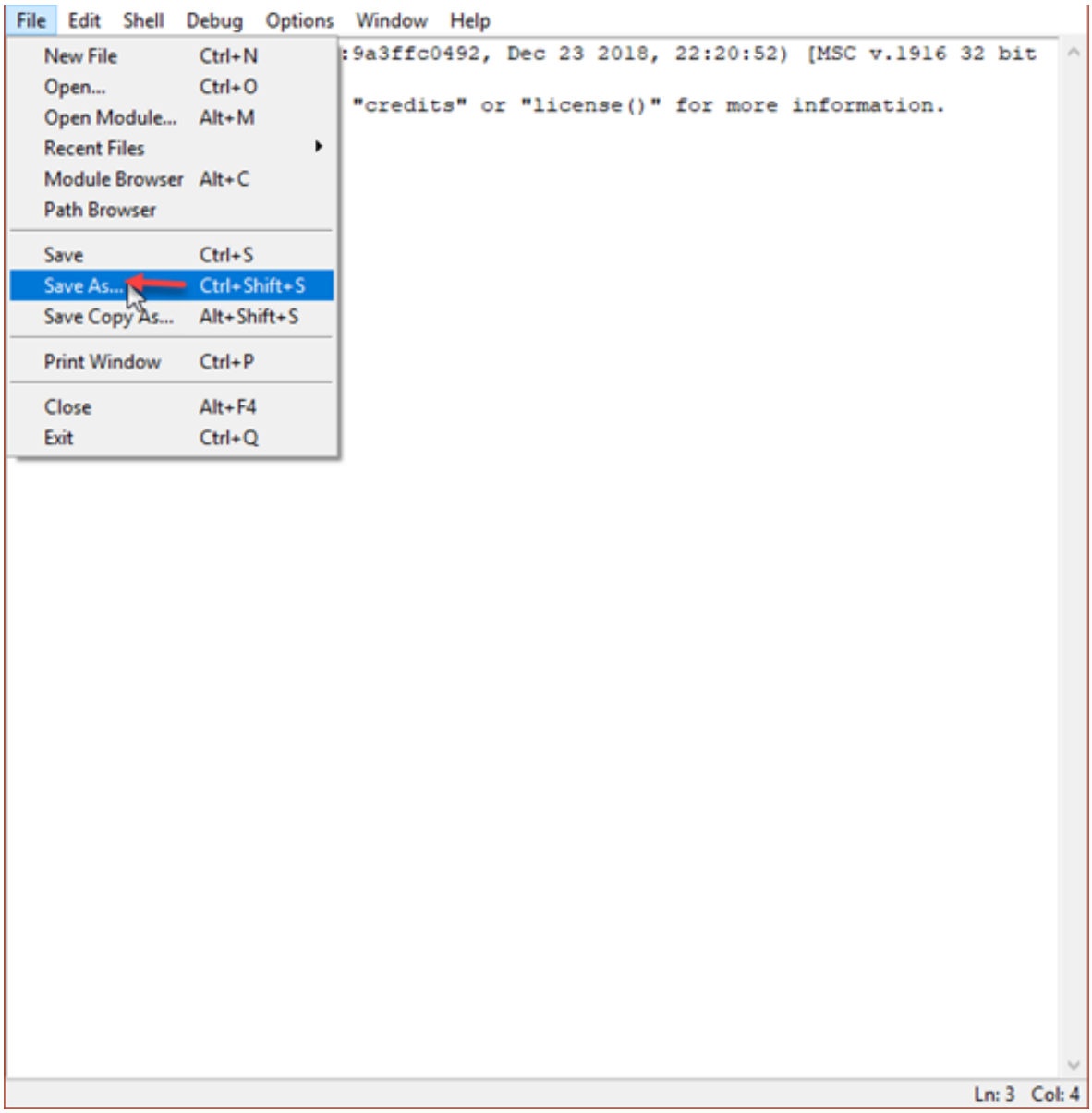
Its important to know that **==** Is equal to is testing value equality. The **=** is just assigning a new value to something. Its important to know the difference between the two.

Practice!

Its recommended that before you move on that you do a little bit of practice on your own with the operators and how they work. Again, you will not use all of these, but its good to know what they all do. Try to print out some values using the print function to just see what the results are. Once you feel like you have a firm grasp of the operators and what they do you can proceed to the next lesson where you will learn about lists.

In this lesson you will be learning about **collections**. There are three different types of collections in Python and they are: **Tuples, arrays/lists, and dictionaries**. Arrays and lists are pretty much the same in Python, but not all programming languages. You can create a new file and name it **Collections_Text.py**





Keep in mind you can always download the source code for this course if you need to from the Course Home page.

What are collections used for and how do we use them?

Until this point all of the variables that you have seen have only ever held one value at a time. This is fine, but if we want to **store multiple values** within a variable we are going to need to use **collections** for this. One of the benefits of doing this is that we can cart around multiple values in a single location and perform various operations on them. They are set up very similar to how we set up the variables in the previous lessons. We have the name, equals and then some value, except that we would have typically a list of values.

Tuples

Tuples are **non mutable**, which means that they **can store multiple values**, but we cannot modify them or we can't change individual values, and we can't add or remove elements from tuples.

Arrays or lists

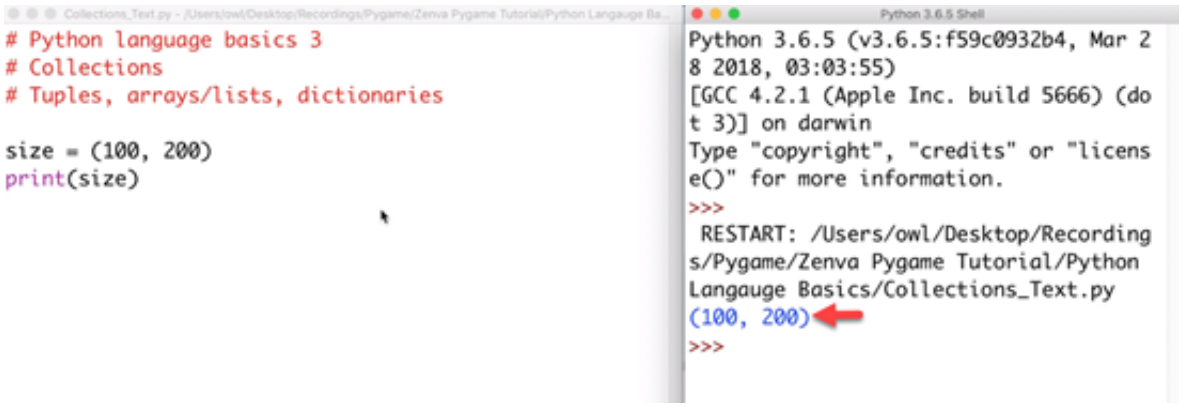
Arrays or **lists** are kind of like **tuples** except that we can add elements to them, we can remove elements from them, we can modify individual elements within them. When using **dictionaries** we do not care about the index of where the elements are store, we store values based on their key. With tuples, arrays, and lists we kind of access the elements based on their index. So if we want the first element, we access index zero. If we want the second element we access index one, and so on. With dictionaries its not really stored according to when you insert an element into the dictionary, its rather the key to which it was assigned.

Tuple Example

For this example lets say we are developing a 2D game. Everything within that game has to have some kind of width and height. So we could define two variables such as width of 100, and a height of 200, or we could define a size tuple and set this equal to 100 by 200.

```
size = (100, 200)
print(size)
```

Now if you run this you will get the tuple printed out, the hotkey for running is **F5:**



This way we can access multiple elements at once, but what if you wanted to access just the width or just the height? You can do so based off index, see the Python code below:

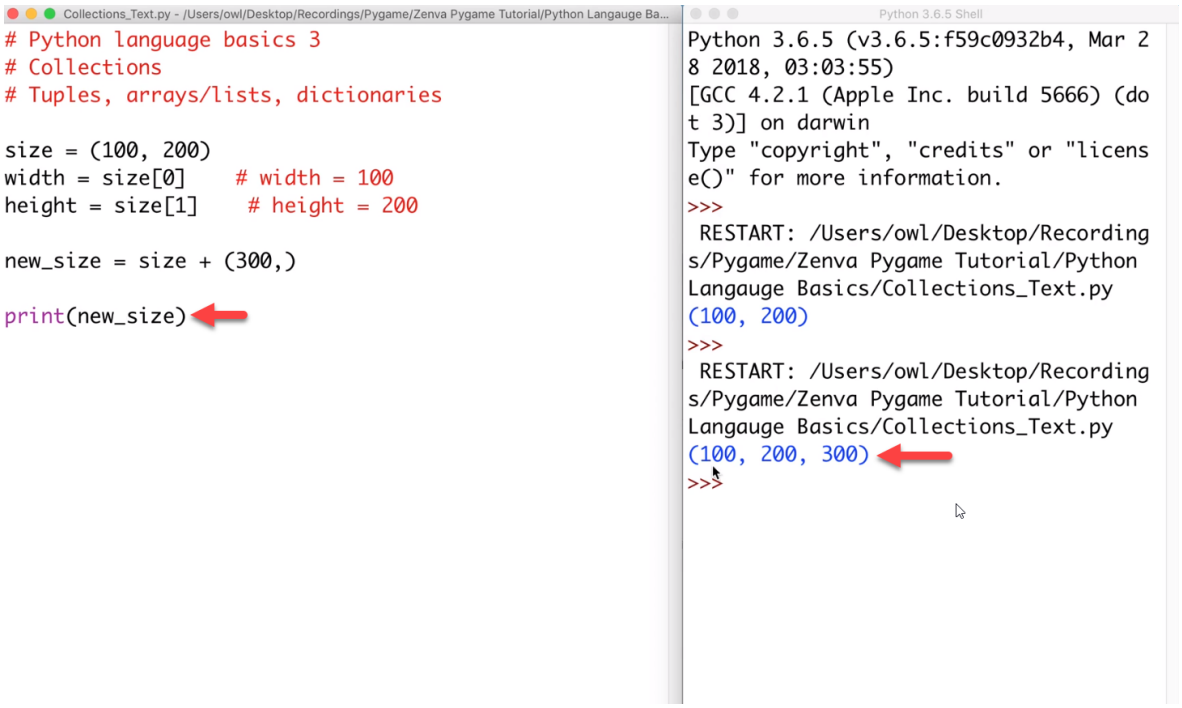
```
width = size[0]    # width = 100
height = size[1]    # height = 200
```

Elements cannot be added or removed from tuples, but what can be done is reassign a tuple. So if we wanted the tuple to have three dimensions, for example:

```
new_size = size + (300,)    # new_size = (100, 200, 300)
```

If you just have one element in your **tuple** you must add the comma. So if we used a print statement and then execute the code we would get:

```
print(new_size)
```



Now, if we wanted to remove elements, we cannot remove individual elements, or modify individual elements. What we would have to do instead is call on delete, new size, and at this point it would delete the entire **tuple.** See the Python code below:

```
del new_size    # new_size no longer exists
```

There are various built in operations that can be performed on tuples, arrays, lists, and dictionaries. If we wanted to find the length of a tuple, maximum length, and minimum size. See the Python code below:

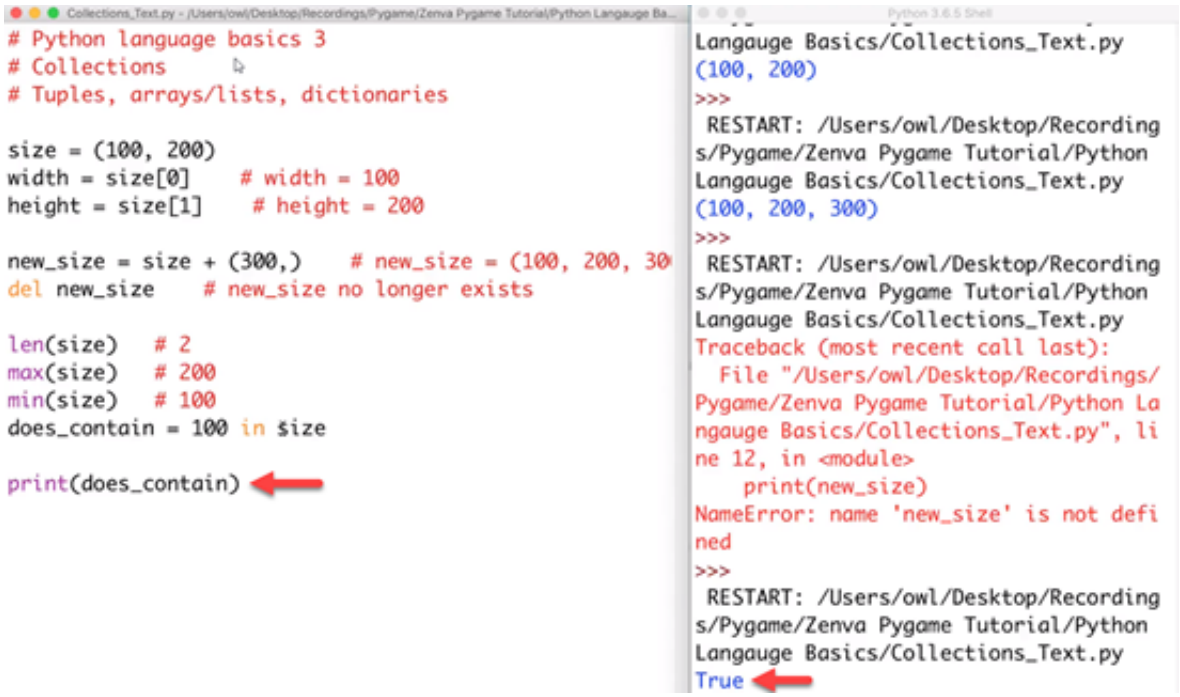
```
len(size)   # 2
max(size)   # 200
min(size)   # 100
does_contain = 100 in size   # does_contain = True
```

The last operator we will talk about in regards to tuples is the in operator. The in operator is used to check and see if a tuple, array, list, or dictionary contains an element. This will return a true or false value. See the Python code below:

```
does_contain = 100 in size   # does_contain = True
```

Then if we print it:

```
print(does_contain)
```

```
# Python language basics 3
# Collections
# Tuples, arrays/lists, dictionaries

size = (100, 200)
width = size[0]    # width = 100
height = size[1]    # height = 200

new_size = size + (300,)    # new_size = (100, 200, 30
del new_size    # new_size no longer exists

len(size)   # 2
max(size)   # 200
min(size)   # 100
does_contain = 100 in size

print(does_contain)
```

```
Langauge Basics/Collections_Text.py
(100, 200)
>>>
 RESTART: /Users/owl/Desktop/Recording
s/Pygame/Zenva Pygame Tutorial/Python
Langauge Basics/Collections_Text.py
(100, 200, 300)
>>>
 RESTART: /Users/owl/Desktop/Recording
s/Pygame/Zenva Pygame Tutorial/Python
Langauge Basics/Collections_Text.py
Traceback (most recent call last):
  File "/Users/owl/Desktop/Recordings/
Pygame/Zenva Pygame Tutorial/Python La
ngauge Basics/Collections_Text.py", li
ne 12, in <module>
    print(new_size)
NameError: name 'new_size' is not defi
ned
>>>
 RESTART: /Users/owl/Desktop/Recording
s/Pygame/Zenva Pygame Tutorial/Python
Langauge Basics/Collections_Text.py
True
```

This is the basics of tuples, and they are usually just used to store values, like if we are trying to retrieve the output from a function.

Using Arrays/Lists in Python

Arrays and **lists** are more flexible than using tuples because you can add or remove elements from them. We can also delete elements from them. In Python arrays and lists are the same thing, whereas in different programming languages they are not. For the arrays/lists example we will use some movement commands. See the Python code below:

```
movement = [5, -2, -3, 4, -1]
first_movement = movement[0]    # first_movement = 5
movement[0] = 7    # movement = [7, -2, -3, 4, -1]
movement.append(-5)    # movement = [7, -2, -3, 4, -1, -5]
movement.remove(-3)   # movement = [7, -2, 4, -1, -5]
```

The benefits of using arrays/lists in Python is that we can add, remove, modify, and delete elements. **Elements** are added by using the **append** command. **Elements** are removed by using the **remove** command. Arrays/lists are accessed by **index**, its important to remember this.

Using Dictionaries in Python

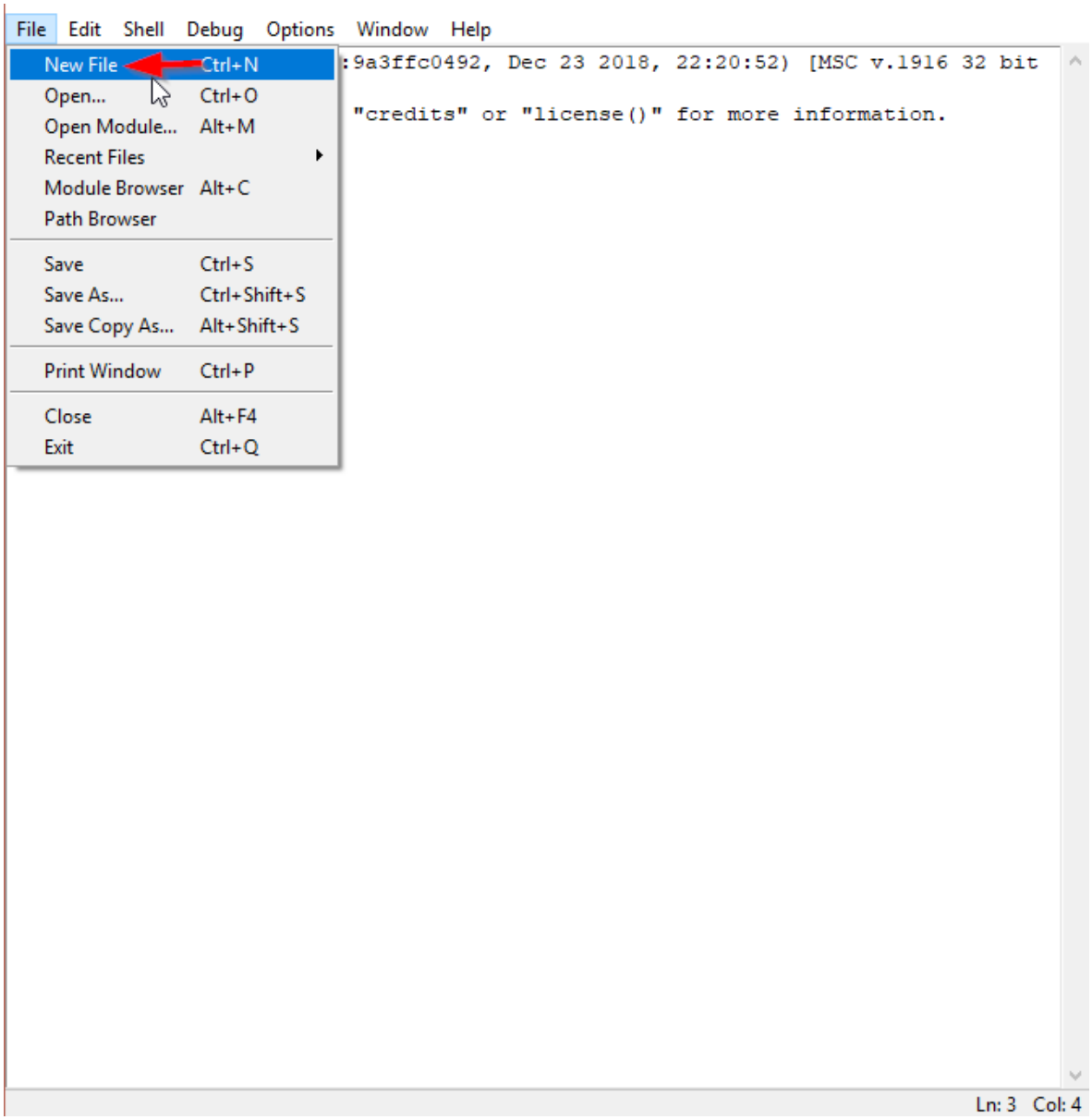
When using dictionaries in Python we store information based on key value pairing. So we don't store a value at index zero or index one in a dictionary. It is stored at a certain key. So for this example lets say we have a series of starting positions for our game setup, there are three characters in my game, see the Python code below:

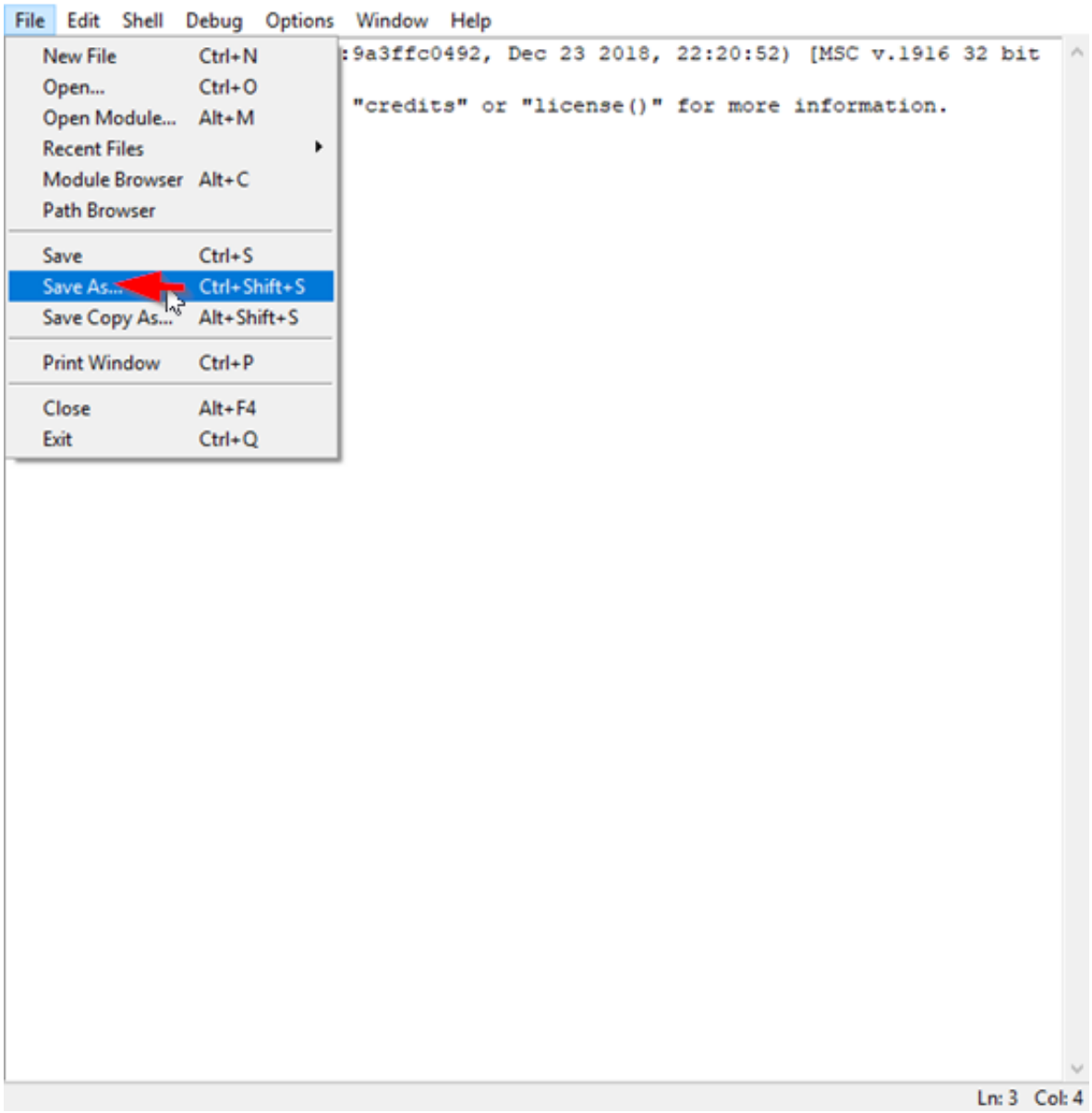
```
starting_positions = {'p_0': 50, 'p_1': 100, 'p_2': 150}
starting_positions['p_0']   # 50
starting_positions.keys()   # ['p_0', 'p_1', 'p_2']
```


If we wanted to append elements to dictionaries we have to do so by using the key value pair. If we wanted to remove elements to dictionaries we have to so by using the key value pair.

You may proceed to the next lesson in the course.

In this lesson you will be learning about how to control the flow of your projects using if statements. For this lesson you can start a new file and name it **“Control_Flow_Text.py”**





Controlling the Flow in Code

We use **control flow** to test some kind of condition, and then choose which piece of the code we want to execute based on the outcome of that test. These tests return true or false values. If the value is returned true, then some code is executed. If it is returned false then it's ignored, or a different block of code is executed. See the Python code example below for how to setup if statements:

```
if condition:
    code to execute if condition is true
```

We can use the game building example where we have some enemies, and we have our character, and we have to move form one side of the map to the other without colliding with the enemies. We might want to have some way to test when we move forward if we've collided with the enemy or not. We can use a simple if statement for this:


```
is_game_over = False
p_0_x_pos = 1
e_0_x_pos = 3

p_0_x_pos += 2    # p_0_x_pos = 2
if p_0_x_pos == e_0_x_pos:  # = False so skip code below
    is_game_over = True
```

Now if we had more than just the one enemy in the game:

```
is_game_over = False
p_0_x_pos = 1
e_0_x_pos = 3
e_1_x_pos = 5

p_0_x_pos += 2    # p_0_x_pos = 2
if p_0_x_pos == e_0_x_pos:  # = False so skip code below
    is_game_over = True
elif p_0_x_pos == e_1_x_pos:  # = False so skip code below
    is_game_over = True
else:   # Carried out if all above tests fail so execute code below
    e_0_x_pos += 1
    e_1_x_pos += 1
```

But, we can make this code more efficient, if you notice the outcome of both tests for the **if** and **elif** statements is true, so we are executing the same code here. What we can do is combine the two into just a single test and just have the test being performed in one go, and then have the else case as well. We can do this by using two **operators** called the **“and” “or”** see the Python code below:

```
# Only one of the two tests has to pass to execute the code
if p_0_x_pos == e_0_x_pos or p_0_x_pos == e_1_x_pos:
    is_game_over = True
else:
    e_0_x_pos += 1
    e_1_x_pos += 1
```

If we used the **“and”** operator instead of the **“or”** operator above in the code example, we would need both the tests to pass.

More Practice

It is highly recommended that you do some practice with control flow. It is very important to grasp the if statement concept before you learn about loops. Loops do use similar concepts, but can be a little more confusing.

Pay Attention to Indentation

One last thing for this lesson, it’s important to pay attention to the indentation in your Python code. If you come from another programming language, you might have seen that language use curly brackets. Python relies on colons and having the correct indentation.

In this lesson you will continue to learn about control flow, this lesson will focus on loops; while loops and for in loops.

It is important that you have grasped the concept of if statements. If you do not feel comfortable with if statements then go back to the previous lesson and get some more practice before beginning this lesson.

What are Loops?

Loops just provide us a way to execute the same statements or set of statements over and over again even though we've actually only typed it once, and we are running the loop.

Loops are very **useful** when you need to **iterate** through **arrays**, this is specific for **for in loops**, and **while loops** are mostly used to **update** the game.

It's very important hat you understand the big game loop, and how it works. The game loop can be used for every single frame in order to update stuff like player and enemy positions.

While Loops-How To Use Them

You can use the same code from the previous lesson, the file is called **"Control_Flow_Text.py"** Open this file up in the IDLE. We can add the code for the while loop example to what we already have in this script.

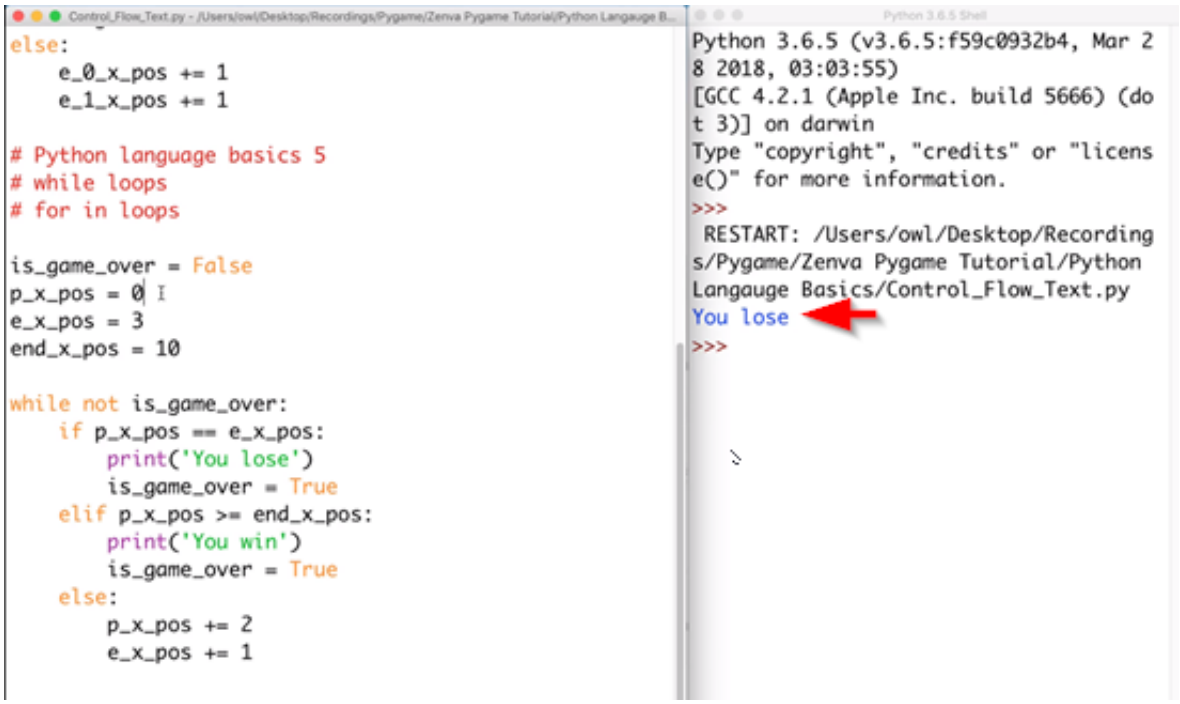
See the Python code below:

```
while not is_game_over:
```

There is some danger when using while loops that your program can get stuck in an **infinite loop**. This can happen if the condition you are testing for never ever fails. When designing while loops be careful to always have an exit condition. This means that at some point the loop will exit. otherwise it will cause your program to crash.

```
# while loops

is_game_over = False
p_x_pos = 0
e_x_pos = 3
end_x_pos = 10

while not is_game_over:
 if p_x_pos == e_x_pos:
        print('You lose')
        is_game_over = True
    elif p_x_pos >= end_x_pos:
        print('You win')
        is_game_over = True
    else:
        p_x_pos += 2
        e_x_pos += 1
```

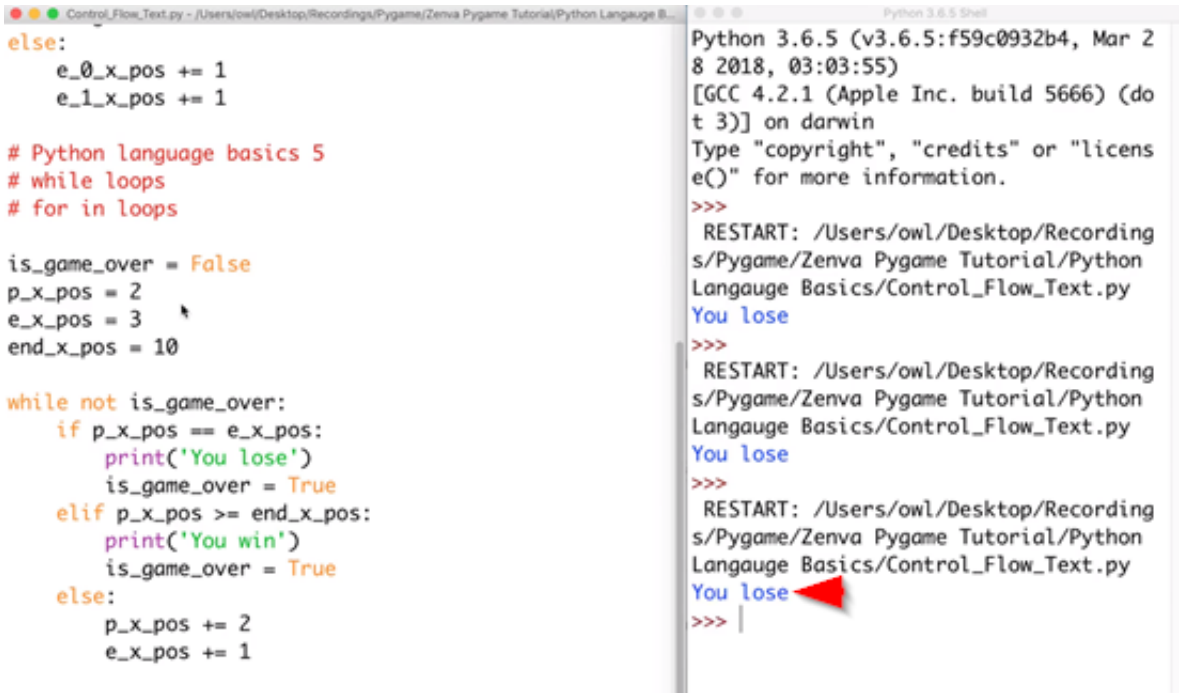

Save this and **run** the program.



Now we can try out some different x position variations to see if we can win, see the Python code below:

```
is_game_over = False
p_x_pos = 2
e_x_pos = 3
end_x_pos = 10

while not is_game_over:
    print(p_x_pos)
    print(e_x_pos)
    if p_x_pos == e_x_pos:
        print('You lose')
        is_game_over = True
    elif p_x_pos >= end_x_pos:
        print('You win')
        is_game_over = True
    else:
        p_x_pos += 2
        e_x_pos += 1
```

This is a lose as well:



Now we can increase the player position, see the Python code below:

```
is_game_over = False
p_x_pos = 2
e_x_pos = 3
end_x_pos = 10

while not is_game_over:
        if p_x_pos == e_x_pos:
        print('You lose')
        is_game_over = True
    elif p_x_pos >= end_x_pos:
        print('You win')
        is_game_over = True
    else:
        p_x_pos += 3
        e_x_pos += 1
```

Now this time when we ran the program we won:


```
    e_1_x_pos += 1

# Python language basics 5
# while loops
# for in loops

is_game_over = False
p_x_pos = 2
e_x_pos = 3
end_x_pos = 10

while not is_game_over:
    if p_x_pos == e_x_pos:
        print('You lose')
        is_game_over = True
    elif p_x_pos >= end_x_pos:
        print('You win')
        is_game_over = True
    else:
        p_x_pos += 3
        e_x_pos += 1
```

```
Type "copyright", "credits" or "licens
e()" for more information.
>>>
 RESTART: /Users/owl/Desktop/Recording
s/Pygame/Zenva Pygame Tutorial/Python
Langauge Basics/Control_Flow_Text.py
You lose
>>>
 RESTART: /Users/owl/Desktop/Recording
s/Pygame/Zenva Pygame Tutorial/Python
Langauge Basics/Control_Flow_Text.py
You lose
>>>
 RESTART: /Users/owl/Desktop/Recording
s/Pygame/Zenva Pygame Tutorial/Python
Langauge Basics/Control_Flow_Text.py
You lose
>>>
 RESTART: /Users/owl/Desktop/Recording
s/Pygame/Zenva Pygame Tutorial/Python
Langauge Basics/Control_Flow_Text.py
You win
```

Now you can print something out every iteration of the loop, see the Python code below:

```
is_game_over = False
p_x_pos = 2
e_x_pos = 3
end_x_pos = 10

while not is_game_over:
    print(p_x_pos)
    print(e_x_pos)
    if p_x_pos == e_x_pos:
        print('You lose')
        is_game_over = True
    elif p_x_pos >= end_x_pos:
        print('You win')
        is_game_over = True
    else:
        p_x_pos += 3
        e_x_pos += 1
```

Save this and **run** it, and you will see the actual positions printed out:

```
    e_1_x_pos += 1

# Python language basics 5
# while loops
# for in loops

is_game_over = False
p_x_pos = 2
e_x_pos = 3
end_x_pos = 10

while not is_game_over:
    print(p_x_pos)
    print(e_x_pos)
    if p_x_pos == e_x_pos:
        print('You lose')
        is_game_over = True
    elif p_x_pos >= end_x_pos:
        print('You win')
        is_game_over = True
    else:
        p_x_pos += 3
```

```
 RESTART: /Users/owl/Desktop/Recording
s/Pygame/Zenva Pygame Tutorial/Python
Langauge Basics/Control_Flow_Text.py
You lose
>>>
 RESTART: /Users/owl/Desktop/Recording
s/Pygame/Zenva Pygame Tutorial/Python
Langauge Basics/Control_Flow_Text.py
You win
>>>
 RESTART: /Users/owl/Desktop/Recording
s/Pygame/Zenva Pygame Tutorial/Python
Langauge Basics/Control_Flow_Text.py
2
3
5
4
8
5
11
6
You win
```

That was a basic while loop, it works very similar to the if statements from the previous lesson. Instead of only executing once and exiting, it executes over and over until the is_game_over evaluates to false.

You can also speed things up and move on to the next loop iteration by using a **continue** statement.

Or you can break out early from the loop by using a **break** statement.

For-In Loops-How to Use Them

For this example we will use the scenario where we have several movement commands.

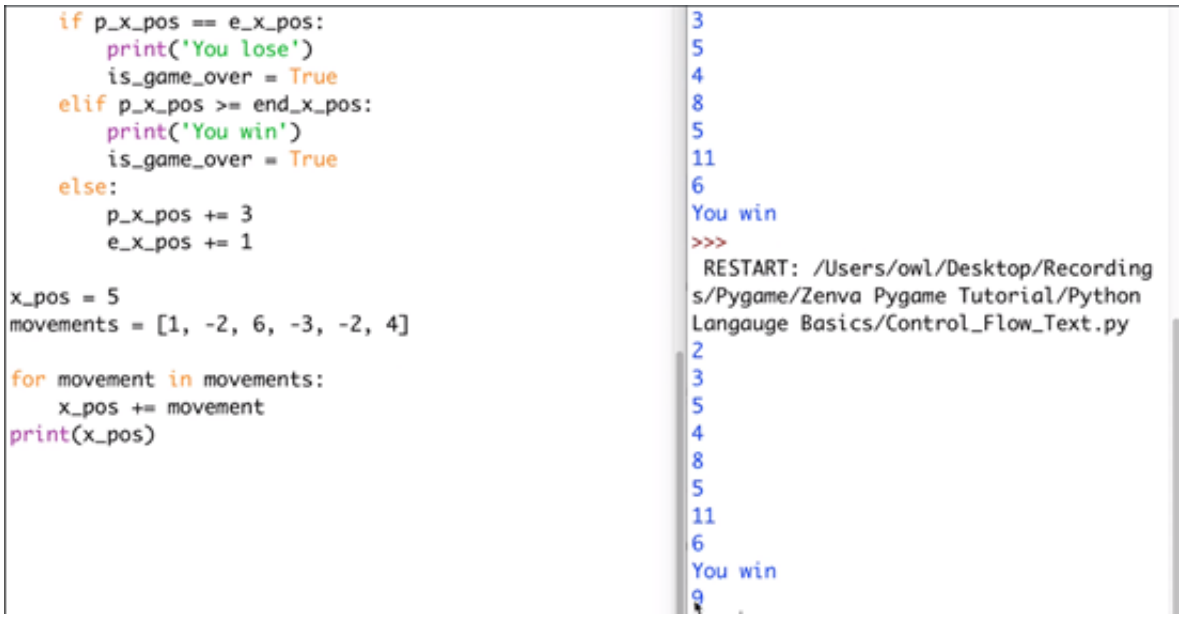
For in loops are built for working with ranges of numbers or with lists or arrays.

Its hard to go out of bounds with a for in loop.

See the Python code below:

```
x_pos = 5
movements = [1, -2, 6, -3, -2, 4]

for movement in movements:
    x_pos += movement
print(x_pos)
```

Save this and **run** it.



```
    if p_x_pos == e_x_pos:
        print('You lose')
        is_game_over = True
    elif p_x_pos >= end_x_pos:
        print('You win')
        is_game_over = True
    else:
        p_x_pos += 3
        e_x_pos += 1

x_pos = 5
movements = [1, -2, 6, -3, -2, 4]

for movement in movements:
    x_pos += movement
print(x_pos)
```

```
3
5
4
8
5
11
6
You win
>>>
 RESTART: /Users/owl/Desktop/Recording
s/Pygame/Zenva Pygame Tutorial/Python
Langauge Basics/Control_Flow_Text.py
2
3
5
4
8
5
11
6
You win
9
```

For in loops are very simple to use, its recommended that you use these with arrays, but really only when you want to visit every single element.

Sometimes you don't want to visit every element within an array. Sometimes you may want to start halfway through. That is the restriction when using for in loops, that it does visit every element, starting at the beginning and finishing at the very end.

More Practice!

Again, it's highly recommended that you take some time and practice with while and for in loops. Try some more examples on your own, and don't forget to use print statements to se the iterations.

Once you are comfortable with loops you can move on to the next lesson where you will learn about functions.

In this lesson you will be learning about functions. You will learn how to implement and call functions. How to pass in parameters and use them and retrieve return values from functions. Since this is a brand new topic you can start a new file and name it **“Functions_Text.py.”**

What are Functions and How Are They are Used?

Functions are the part of the code that actually do something, they perform some function, and typically contain a statement or a set of related statements that all work together to accomplish some kind of task. One of the benefits of using **functions** is that it helps to organize your **code** better. You can group together blocks of related functionality. You can choose exactly when and where you want to run that code simply by pulling up within a function, and then calling upon that function. See the Python code below for the example for function use:

```
x_pos = 0

# Function to increase x_pos by 1
def move():
    x_pos += 1    # changes the value of the global x_pos variable
```

So with this function we are taking the x position and its going to increase it by one. Now if we were to run this function nothing would happen, but if we printed the x position before and after we would get the same value for each. The value zero would be printed for both. This is because we have implemented the function but have not called upon it. In order to make a function execute you must call it.

Function Calling

Calling a function is done by calling on the function by name, and then passing any parameters, or just empty parenthesis if there are no parameters to be entered for the function. Once the function is called, and the function definition is found the code will run found within that specific function that was called. In order to run this move function we would need to call it, and you can see the Python code below for the function call example:

```
move()    # runs the code in the move function
# x_pos = 1
```

Global variables

A **global variable** is a variable that exists outside the scope of a function. The variable that we have declared above for the x_pos = 0 is a completely different variable from the one in the move function.

Local Variables

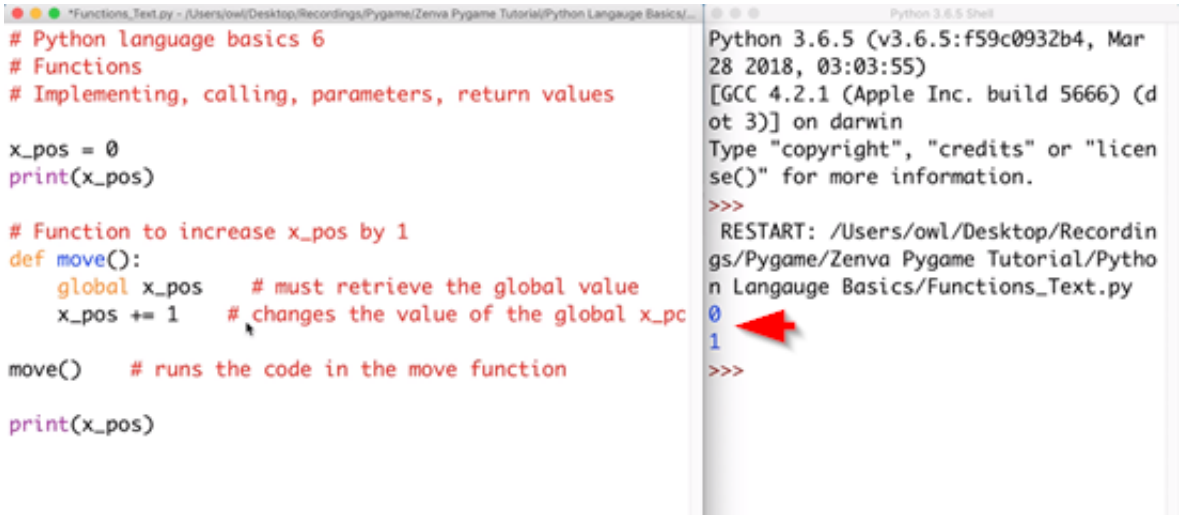
A **local variable** is a variable that exists only within a function. The variable x_pos that is inside the move function above is a local variable. This means that the x_pos variable inside this function does not exist outside the functions execution. See the Python code example below:

```
x_pos = 0
e_x_pos = 4
print(x_pos)
```



```
# Function to increase x_pos by 1
def move():
    global x_pos    # must retrieve the global value
    x_pos += 1    # changes the value of the global x_pos variable

move()    # runs the code in the move function
# x_pos = 1


# Function to increase x_pos by 'amount'
def move_by(amount):
    global x_pos
    x_pos += amount
```

Now if we save this and run it we will get the updated x_pos:



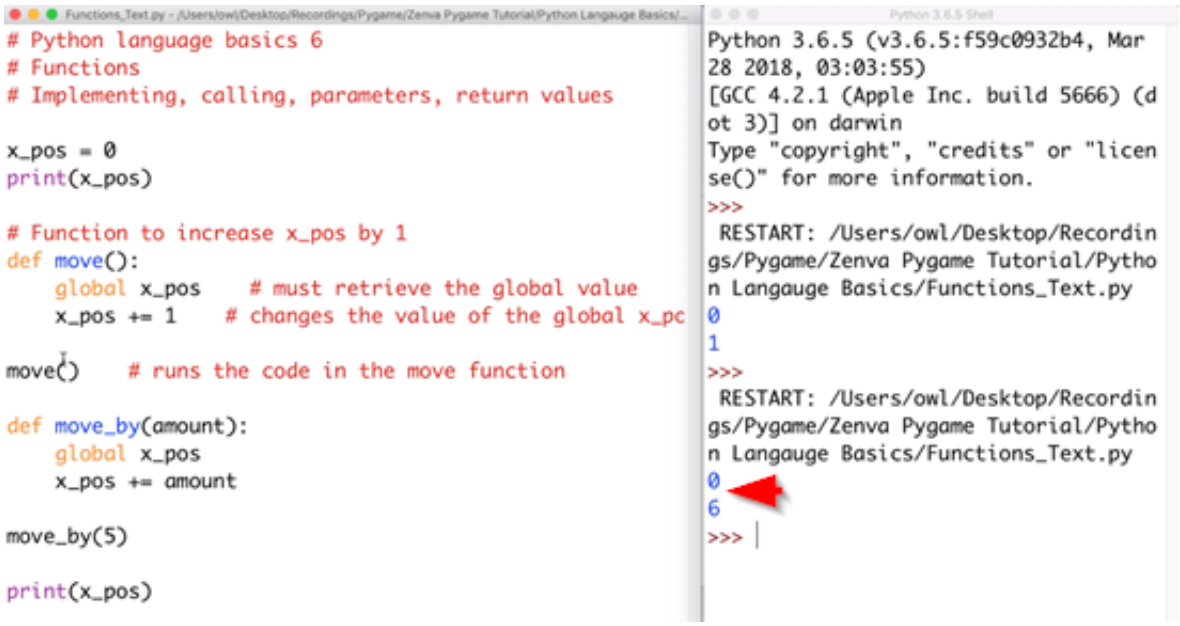
Now this move function is doing what we want it to, but its still not very good. The reason is because we are moving forward by a static one every time. What we want to do is modify this function above so that it takes in a parameter, which is going to allow us to determine how much we want to move forward or backwards. See the Python code below for the function that takes in the amount as a parameter for the move function:

```
# Function to increase x_pos by 'amount'
def move_by(amount):
    global x_pos
    x_pos += amount
```

So when we go to call this move function we will have to pass in some value that will set the amount.

```
move_by(5)    # need to pass in a value for 'amount'
```

So if you ran this you would see that the value of amount would print out:



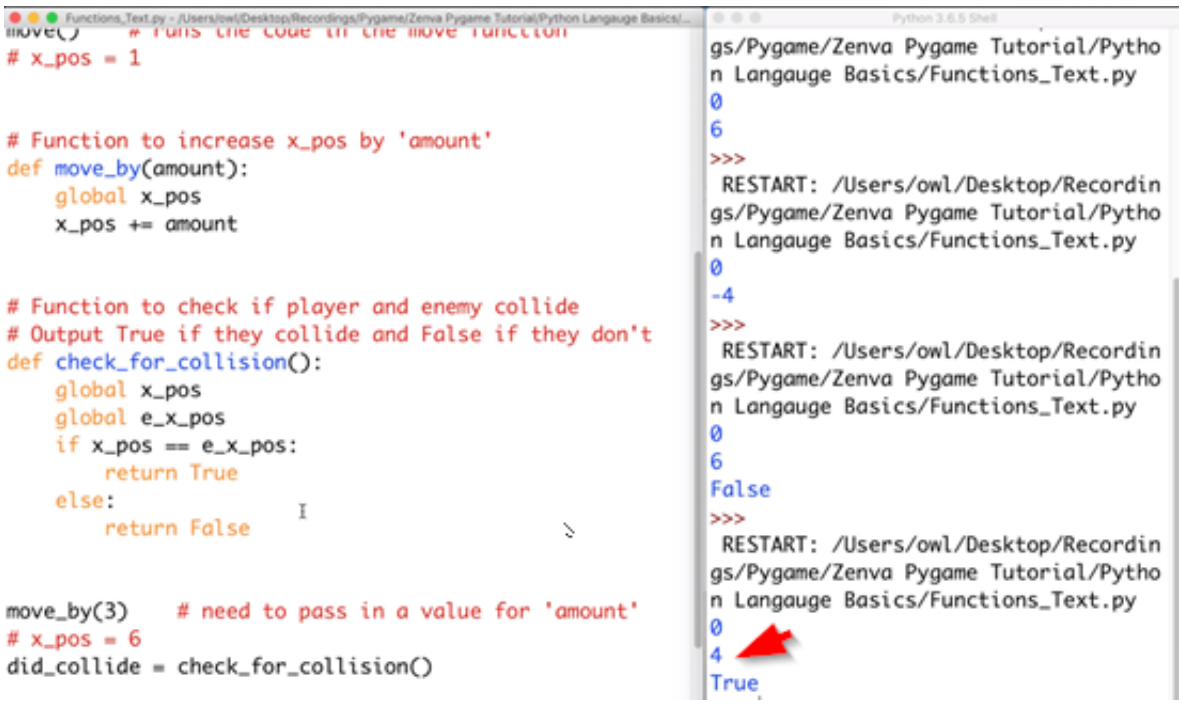
If you did not pass in a value for the amount parameter the function would not run properly. **Return values** are the opposite of parameters. **Parameters** are input into the function, but return values are output from the function. See the Python code below:

```
x_pos = 0
e_x_pos = 4
print(x_pos)

# Function to increase x_pos by 1
def move():
    global x_pos    # must retrieve the global value
    x_pos += 1    # changes the value of the global x_pos variable

move()    # runs the code in the move function
# x_pos = 1


# Function to increase x_pos by 'amount'
def move_by(amount):
    global x_pos
    x_pos += amount


# Function to check if player and enemy collide
# Output True if they collide and False if they don't
def check_for_collision():
    global x_pos
    global e_x_pos
    if x_pos == e_x_pos:
        return True
    else:
        return False


move_by(3)    # need to pass in a value for 'amount'
# x_pos = 6
```



```
did_collide = check_for_collision()

print(x_pos)
print(did_collide)
```

Give this a save and run it.



You now should understand function implementation, calling functions, passing in parameters, and returning values in functions. As always I would recommend you give some more practice to the use of functions and try working on some other examples using functions.

In this lesson we will be bringing together everything that we have covered so far. Please make sure you are comfortable with everything that you have learned so far in this course before you begin this lesson on classes and objects.

Classes and Objects

This is a brand new topic so start a new file and name it **“Objects_and_Classes_Text.py”**

In object oriented programming languages **objects** can be considered something within the code, some entity that has state and behavior that we might need to keep track of. State is represented through **fields**, or as we saw in the previous lesson **global variables**. The behavior is implemented through functions that actually do things. These are also called methods when they are inside of a class. We can create new instances of objects, **object instantiation** through the use of **constructors**. **Constructors** are a very special type of function or method that help to set up all of the different fields of a class. You can think about the object as being an instance of the class that keeps track of all the fields, methods, and constructors of that particular object. The **class** is kind of like the **blueprint**. Python is not strictly object oriented, which means we can actually write or run code without having to use classes and objects. See the Python code below for the example of a generic game character:

```
# A class is just a blueprint that defines and object's attributes and behaviours
class GameCharacter:

    # A field or global variable (assigned this value when class is instantiated
    speed = 5

    # Constructor creates a new class instance and sets up the defined fields
    def __init__(self, name, width, height, x_pos, y_pos):
        self.name = name
        self.width = width
        self.height = height
        self.x_pos = x_pos
        self.y_pos = y_pos

    # A method is just a function that typically modifies the classes fields
    def move(self, by_x_amount, by_y_amount):
        self.x_pos += by_x_amount
        self.y_pos += by_y_amount


# character_0 is a new instance of the GameCharacter class with the defined attribute
s
character_0 = GameCharacter('char_0', 50, 100, 100, 100)
character_0.name    # 'char_0'
character_0.name = 'char_1'
character_0.name    # 'char_1'

character_0.move(50, 100)
character_0.x_pos   # 150
character_0.y_pos   # 200
```

This covers just the basics of classes and objects, there is a lot more that can be covered. In the next

lesson we will other some other topics of subclassing and super classing.

This is the final lesson of the Python basics tutorials. After this lesson we will be able to apply everything that you have learned in the previous eight lessons towards game development.

In this lesson you will learn about **subclasses, superclasses, and inheritance**.

Because this lesson is just a continuation on classes and objects you can stick to using the same file we used in the previous lesson. So go ahead an open that filed named
“Objects_and_Classes_Text.py”

Subclasses inherit from superclasses, this means that if a subclass inherits from a superclass, it gets access to all of those attributes, all of the fields, all of the behaviors, and can also use the constructor from that superclass without us having to explicitly reprogram that stuff.

So for this example lets say the GameCharacter class we created in the previous lesson is the superclass and we make a subclass of this, maybe like a player character or a non player character, or some specific type of game character.

This would mean that the **subclass** would have a **speed, name, width, height, and X and Y position**.

It would also have access to the move function even if we did not explicitly program that.

Using subclasses gives us the benefit of not having to write more code, it minimizes the amount of code that needs to be written.

See the Python code below for the subclass that we will create from the GameCharacter superclass:

```
# Player character is a subclass of GameCharacter
# Player character has access to everything defined in GameCharacter but not vice ver
sa
class PlayerCharacter(GameCharacter):

    speed = 10

    # Should still provide a constructor/initializer
    def __init__(self, name, x_pos, y_pos):
        super().__init__(name, 100, 100, x_pos, y_pos)

    def move(self, by_y_amount):
        super().move(0, by_y_amount)


player_character = PlayerCharacter('P_character', 500, 500)
player_character.name   # 'P_character'
player_character.move(100)
print(player_character.x_pos)  # 600
print(player_character.y_pos)  # 600
```

Remember, subclasses inherit from superclasses, whihc means they gain all of the fields, attributes, behaviors, methods, and can use the constructor.

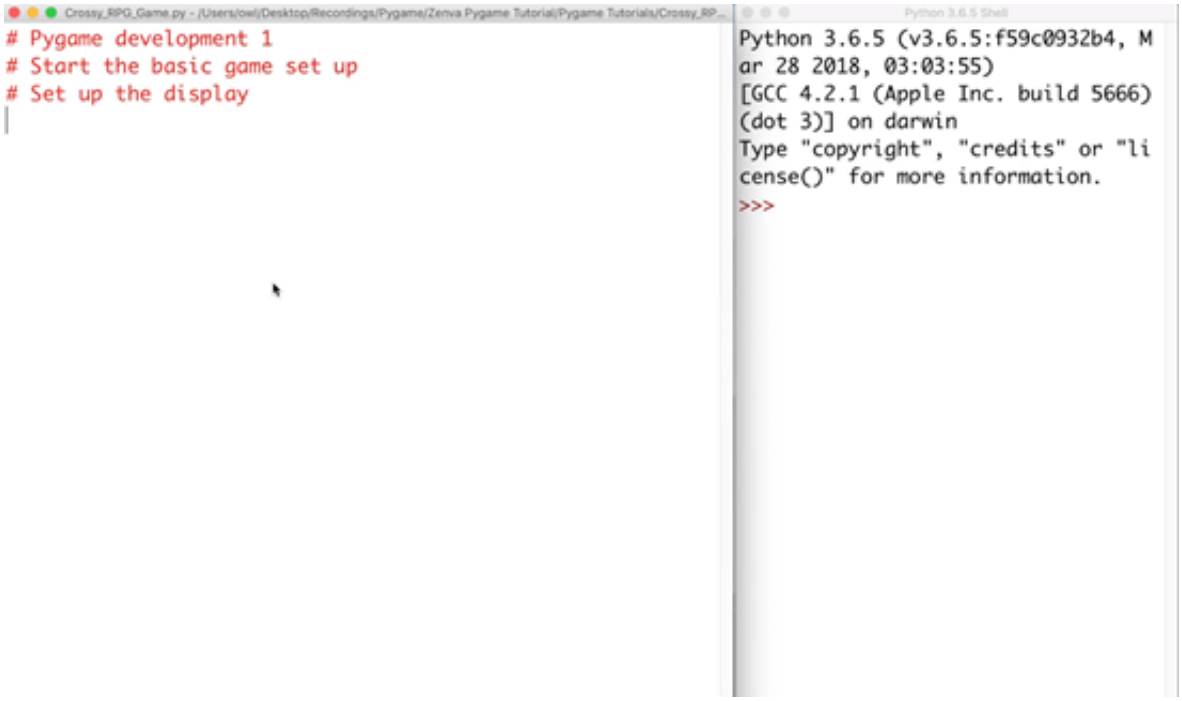
Review and Practice!

It is highly recommended that before you start the next part of this course which will teach you how to use Pygame that you completely understand the Python basics. If necessary go back and review the lessons and try doing some other examples that relate to game development. Once you feel comfortable with the basics go ahead an move on to the next lesson.

In this lesson we will begin working on building the crossy road game from scratch using Pygame. Go ahead and start a new file and name it **“Cross_RPG_Game.py”** the source code is included for this course, but it is highly recommended that you follow along by writing your own code. If you get stuck and need to consult the source code to locate a mistake you just cannot find on your own that is fine, feel free to consult the source code then. You can download all the source code for this course from the Course Home page.

Basic Game Setup

We will be starting this lesson off with the basic game set up, and this is going to be focusing mostly on actually working the display itself. My setup for the rest of the course will be having **IDLE** open and on the left side of the screen and then having the **Python shell** up and running on the ride side, like so:



Importing Pygame

The first thing that needs to be done is to import **Pygame**. You were showed right at the beginning of the course how to download and install Pygame. You can import Pygame by adding some code to the script we just started, see the Python code below:

```
# Gain access to the pygame library
import pygame
```

Pygame is just a library that contains all of the functions, classes, objects, etc. that will be used to build the crossy road game in this course.

Building up the Game Screen

We will have to specify a width and height. We will need a couple variables for this, one for **SCREEN_WIDTH** and the other for **SCREEN_HEIGHT**, see the Python code below:


```
# Size of the screen
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800

# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)
```

We now have all the components we need to build up the game screen window, but now we need to actually build the window itself. We can return this in a variable called something like **game_Screen.** This variable will represent the entire screen. See the Python code below:

```
# Create the window of specified size in white to display the game
game_screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
# Set the game window color to white
game_screen.fill(WHITE_COLOR)
```

Save the file. We have accomplished all of the tasks that we set out to finish for Pygame development one. We have the basic game setup and setup the basic display for the crossy game. There are going to many more steps to actually display something on the screen, so its not recommended to run this code we have so far because you will get some unintended behavior.

Basic Game Loop Creation

In order to run it properly we will need to create a very basic game loop to get started and we will have to use the **pygame.display.update()function** which will be used to render the graphics. In the next lesson we will talk about the game loop itself, focusing on displaying the screen, how to display items on the screen using this game loop and the display.update()function, and then we will talk about how to safely exit the game.

Review!

In this previous lesson we just setup the screen itself, and here is the Python code below for what you should have so far in your **“Crossy_RPG_Game.py file”**

```
# Gain access to the pygame library
import pygame

# Size of the screen
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800

# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)

# Create the window of specified size in white to display the game
game_screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
# Set the game window color to white
game_screen.fill(WHITE_COLOR)
```

We will now work on setting up the game loop to actually render the graphics. In this case it will just be rendering the screen that we have created.

The Basic Game Loop

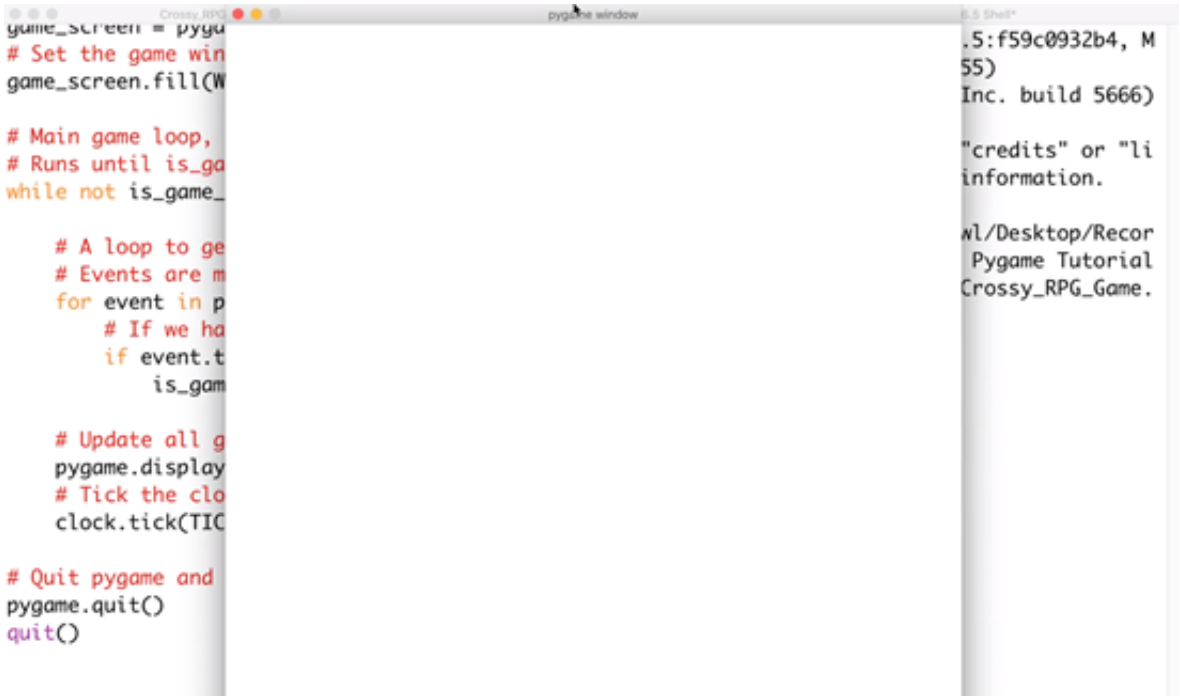
A game loop is a very simple while loop that will contain all of the game logic within it. This **while loop** is going to run over and over again until there is going to be some condition that maybe sets a variable to true or to false which will cause us to break out of the loop and then exit the game. We will be adding some code to the file we already have, so make sure its open to begin. See the Python code below:

```
# Gain access to the pygame library
import pygame

pygame.init()

# Size of the screen
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800

# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)

# Clock used to update game events and frames
clock = pygame.time.Clock()
# Typical rate of 60, equivalent to FPS
TICK_RATE = 60
is_game_over = False

# Create the window of specified size in white to display the game
game_screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
```



```
# Set the game window color to white
game_screen.fill(WHITE_COLOR)

# Main game loop, used to update all gameplay such as movement, checks, and graphics
# Runs until is_game_over = True
while not is_game_over:

            # A loop to get all of the events occuring at any given time
            # Events are most often mouse movement, mouse and button clicks, or exit
events
            for event in pygame.event.get():
              # If we have a quite type event (exit out) then exit out of the game lo
op
              if event.type == pygame.QUIT:
                is_game_over = True

              # Update all game graphics
              pygame.display.update()
              # Tick the clock to update everything within the game
              clock.tick(TICK_RATE)

# Quit pygame and the program
pygame.quit()
quit()
```

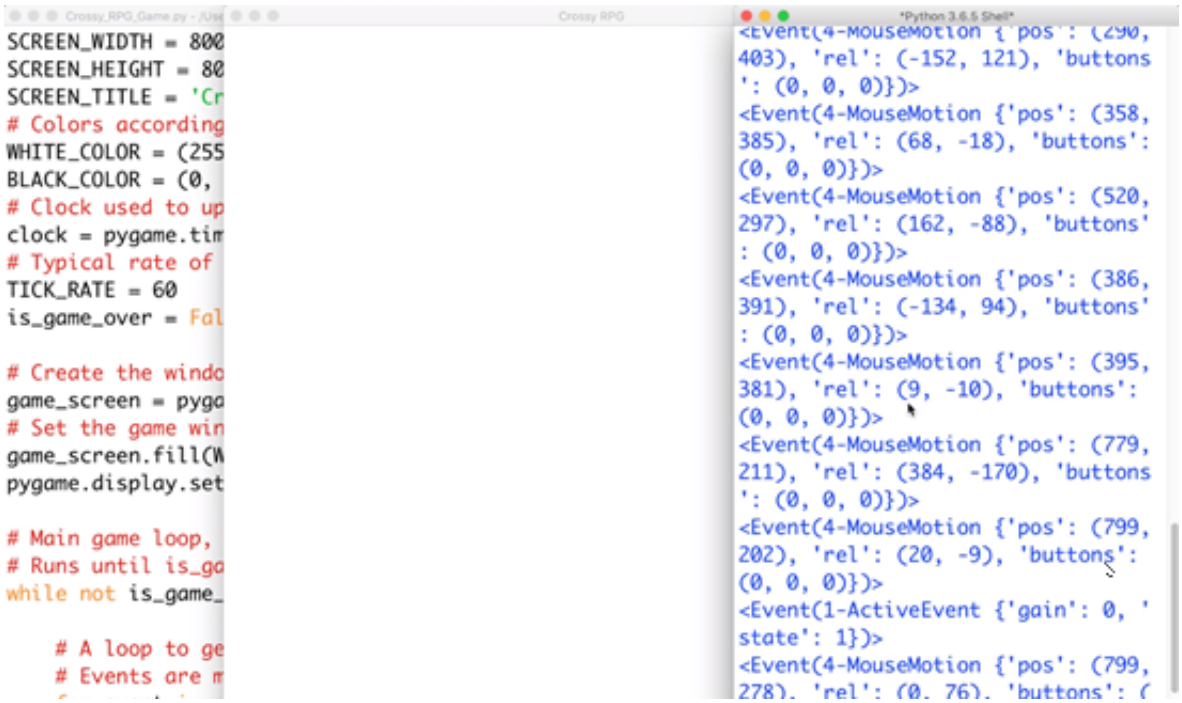
Save this and run it.



So this isn't very interesting but we have the 800 X 800 window displaying. We can add a title to the game and this will display instead of the **"pygame window"** text. Open the script back up and we will add some more code to it:


```
# Gain access to the pygame library
import pygame

pygame.init()

# Size of the screen
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)

# Clock used to update game events and frames
clock = pygame.time.Clock()
# Typical rate of 60, equivalent to FPS
TICK_RATE = 60
is_game_over = False

# Create the window of specified size in white to display the game
game_screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
# Set the game window color to white
game_screen.fill(WHITE_COLOR)
pygame.display.set_caption(SCREEN_TITLE)


# Main game loop, used to update all gameplay such as movement, checks, and graphics
# Runs until is_game_over = True
while not is_game_over:

            # A loop to get all of the events occuring at any given time
            # Events are most often mouse movement, mouse and button clicks, or exit
events
            for event in pygame.event.get():
                # If we have a quite type event (exit out) then exit out of the game
loop
                if event.type == pygame.QUIT:
                    is_game_over = True
                print(event)

            # Update all game graphics
            pygame.display.update()
            # Tick the clock to update everything within the game
            clock.tick(self.TICK_RATE)

# Quit pygame and the program
pygame.quit()
quit()
```

Save this and give it a run. You will see that the title has changed for the game window and you can see the events being printed in the Python shell.



When you exit the game window you will need to reopen the Python shell every time. In the next lesson we will be adding images to be displayed on the screen.

In this lesson we will be continuing where we left off with the code from the previous lesson. Make sure you open up your **“Crossy_RPG_Game.py”** file. We will be drawing objects to the screen such as squares and circles, and then we will move to loading images into those objects, and then displaying actual images on the screen. We will be using the draw function to draw the shapes we need to the screen. It is important to note that every time we run the game loop the graphics are going to be updated. So this means any type of graphics rendering needs to be actually done within the while loop. Sometimes the objects can be created outside the game loop, and then we just display them in the loop. We will be doing both in this case, but this may change a little bit later on.

Drawing the Objects

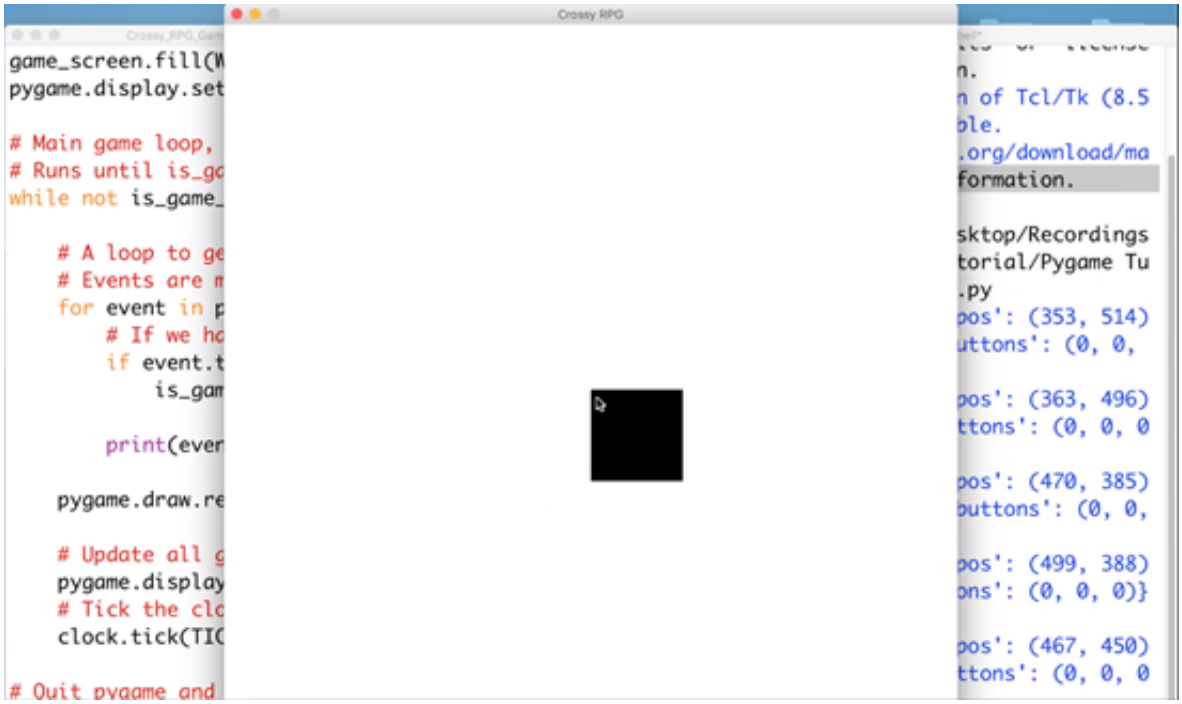
See the Python code below:

```
# Gain access to the pygame library
import pygame

pygame.init()

# Size of the screen
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)

# Clock used to update game events and frames
clock = pygame.time.Clock()
# Typical rate of 60, equivalent to FPS
TICK_RATE = 60
is_game_over = False

# Create the window of specified size in white to display the game
game_screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
# Set the game window color to white
game_screen.fill(WHITE_COLOR)
pygame.display.set_caption(SCREEN_TITLE)


# Main game loop, used to update all gameplay such as movement, checks, and graphics
# Runs until is_game_over = True
while not is_game_over:

            # A loop to get all of the events occuring at any given time
            # Events are most often mouse movement, mouse and button clicks, or exit
events
            for event in pygame.event.get():
                # If we have a quite type event (exit out) then exit out of the game
loop
                if event.type == pygame.QUIT:
                    is_game_over = True
                print(event)

            pygame.draw.rect(game_screen, BLACK_COLOR, [400, 400, 100, 100])
```



```
            # Update all game graphics
            pygame.display.update()
            # Tick the clock to update everything within the game
            clock.tick(self.TICK_RATE)

# Quit pygame and the program
pygame.quit()
quit()
```

Save this and give it a run. You can use **F5** to run the program:



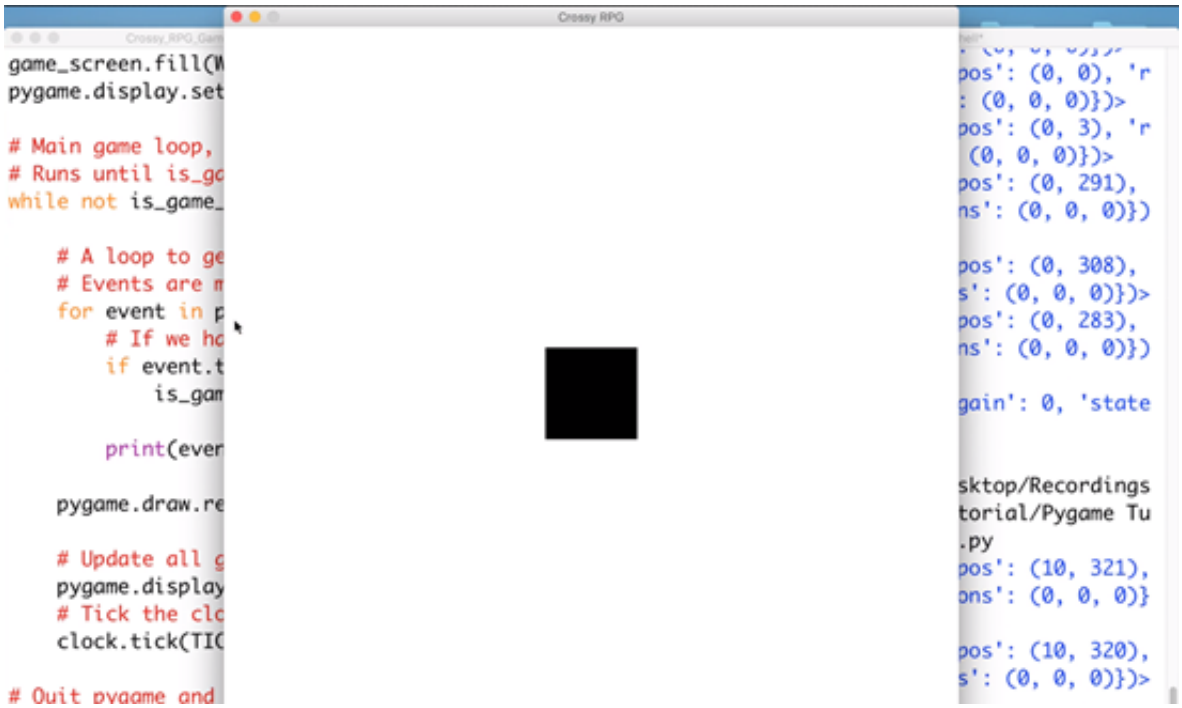
We have successfully drawn the square, but the square is actually slightly off-center. We need to shift the center point left by **50**, and up by **50** for this rectangle. Open the script back up so we can fix this:

```
# Gain access to the pygame library
import pygame

pygame.init()

# Size of the screen
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)

# Clock used to update game events and frames
```



```
clock = pygame.time.Clock()
# Typical rate of 60, equivalent to FPS
TICK_RATE = 60
is_game_over = False

# Create the window of specified size in white to display the game
game_screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
# Set the game window color to white
game_screen.fill(WHITE_COLOR)
pygame.display.set_caption(SCREEN_TITLE)


# Main game loop, used to update all gameplay such as movement, checks, and graphics
# Runs until is_game_over = True
while not is_game_over:

            # A loop to get all of the events occuring at any given time
            # Events are most often mouse movement, mouse and button clicks, or exit
events
            for event in pygame.event.get():
                # If we have a quite type event (exit out) then exit out of the game
loop
                if event.type == pygame.QUIT:
                    is_game_over = True
                print(event)

            pygame.draw.rect(game_screen, BLACK_COLOR, [350, 350, 100, 100])

            # Update all game graphics
            pygame.display.update()
            # Tick the clock to update everything within the game
            clock.tick(self.TICK_RATE)

# Quit pygame and the program
pygame.quit()
quit()
```

Save this and run the code again.



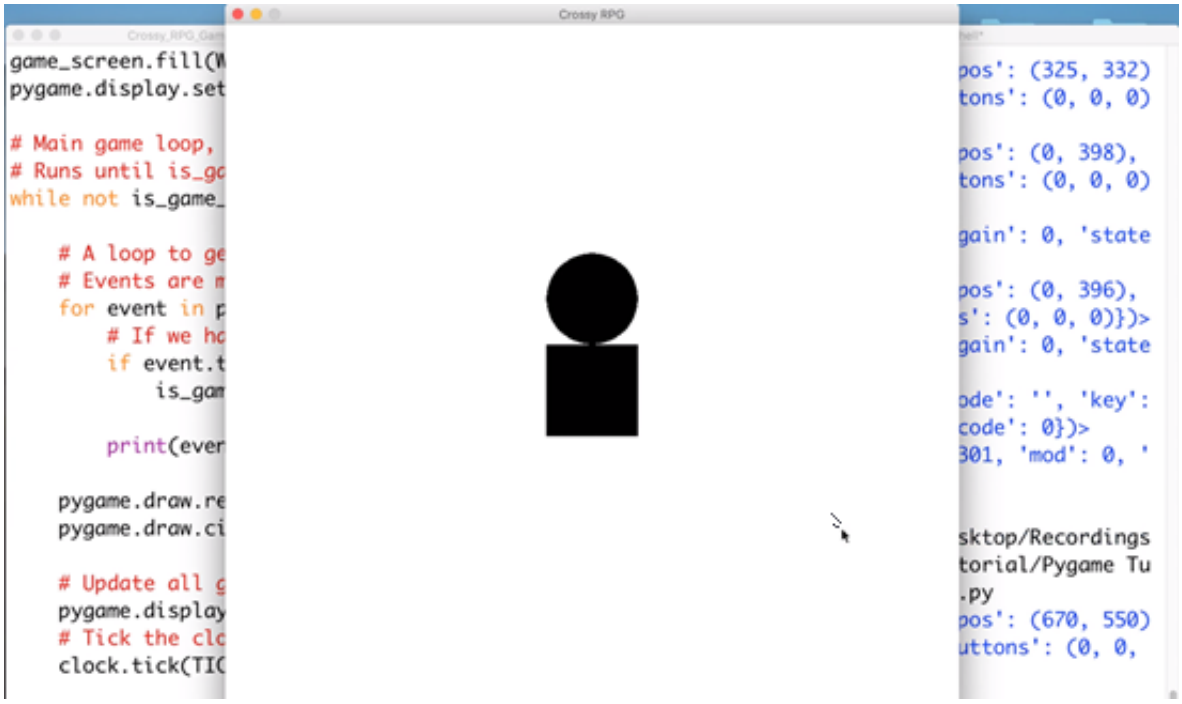
The square now looks centered. We can draw a circle on top of the rectangle we already have. Open the script back up and add the following code to it:

```
# Gain access to the pygame library
import pygame

pygame.init()

# Size of the screen
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)

# Clock used to update game events and frames
clock = pygame.time.Clock()
# Typical rate of 60, equivalent to FPS
TICK_RATE = 60
is_game_over = False

# Create the window of specified size in white to display the game
game_screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
# Set the game window color to white
game_screen.fill(WHITE_COLOR)
pygame.display.set_caption(SCREEN_TITLE)


# Main game loop, used to update all gameplay such as movement, checks, and graphics
# Runs until is_game_over = True
while not is_game_over:
```



```
            # A loop to get all of the events occuring at any given time
            # Events are most often mouse movement, mouse and button clicks, or exit
events
            for event in pygame.event.get():
                # If we have a quite type event (exit out) then exit out of the game
loop
                if event.type == pygame.QUIT:
                    is_game_over = True
                print(event)

            pygame.draw.rect(game_screen, BLACK_COLOR, [350, 350, 100, 100])
            pygame.draw.circle(game_screen, BLACK_COLOR, (400, 300), 50)

            # Update all game graphics
            pygame.display.update()
            # Tick the clock to update everything within the game
            clock.tick(self.TICK_RATE)

# Quit pygame and the program
pygame.quit()
quit()
```

Save this and give it a run using **F5:**



We have put the circle on top of the square.

Loading Images

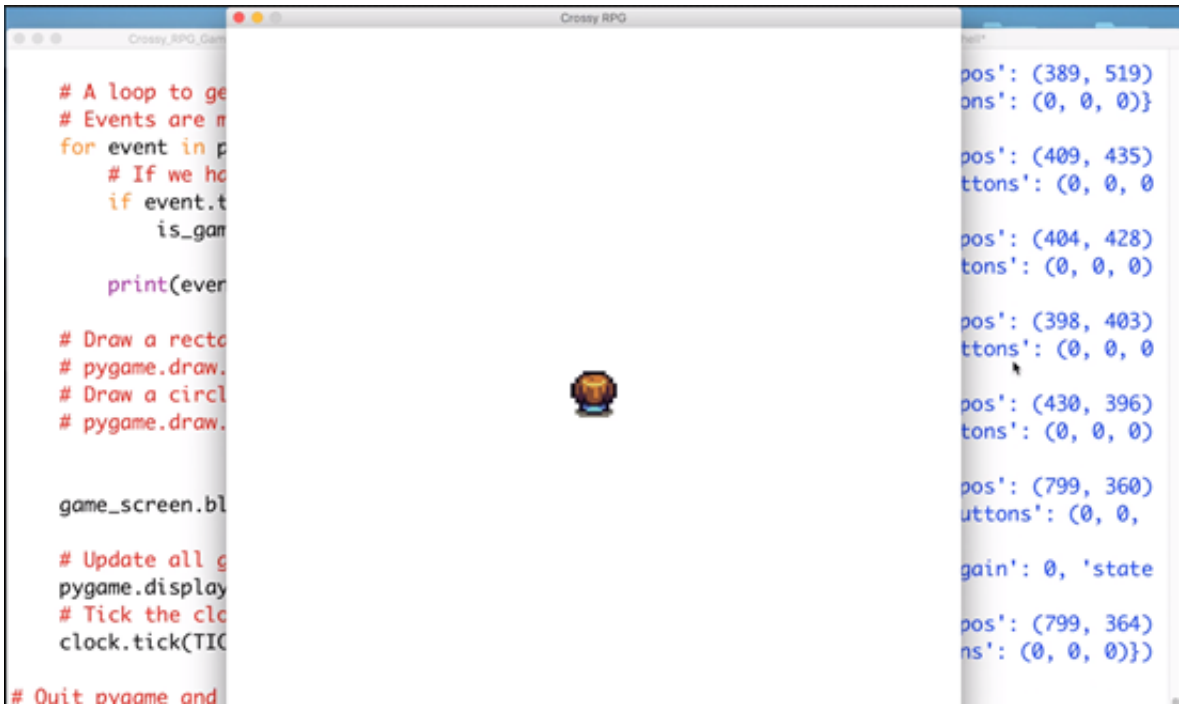
We are now going to be working with some images. I have commented the circle and rectangle drawing out of the code for now. Open the script back up and add the following Python code to it:


```
# Gain access to the pygame library
import pygame

pygame.init()

# Size of the screen
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)

# Clock used to update game events and frames
clock = pygame.time.Clock()
# Typical rate of 60, equivalent to FPS
TICK_RATE = 60
is_game_over = False

# Create the window of specified size in white to display the game
game_screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
# Set the game window color to white
game_screen.fill(WHITE_COLOR)
pygame.display.set_caption(SCREEN_TITLE)

# Load the player image from the file directory
# player_image = pygame.image.load('player.png')
# Scale the image up
# player_image = pygame.transform.scale(player_image, (50, 50))


# Main game loop, used to update all gameplay such as movement, checks, and graphics
# Runs until is_game_over = True
while not is_game_over:

            # A loop to get all of the events occuring at any given time
            # Events are most often mouse movement, mouse and button clicks, or exit
events
            for event in pygame.event.get():
                # If we have a quite type event (exit out) then exit out of the game
loop
                if event.type == pygame.QUIT:
                    is_game_over = True
                print(event)

            #pygame.draw.rect(game_screen, BLACK_COLOR, [350, 350, 100, 100])
            #pygame.draw.circle(game_screen, BLACK_COLOR, (400, 300), 50)

# Draw the player image on top of the screen at (x, y) position
game_screen.blit(player_image, (375, 375))

            # Update all game graphics
            pygame.display.update()
            # Tick the clock to update everything within the game
```



```
            clock.tick(self.TICK_RATE)

# Quit pygame and the program
pygame.quit()
quit()
```

Save this and give it a run using **F5:**



We have the player image in the center of the screen. In the next lesson we will begin making the code we have written so far more object oriented, this is also one of our major course goals, we will begin putting all of the code we have so far into a game class. Then making a new game object and using that.

In this lesson we will be making the code we have already written in the previous lessons more object oriented. We are going to introduce the notion of classes and objects into our specific context. We are only going to work with a first class for now and then I will get you to think about how we will build up the rest of it towards the end. One of the major course goals is to build up an object oriented game. Understanding how to build object oriented games is very important when you are developing games.

One Big Game Class

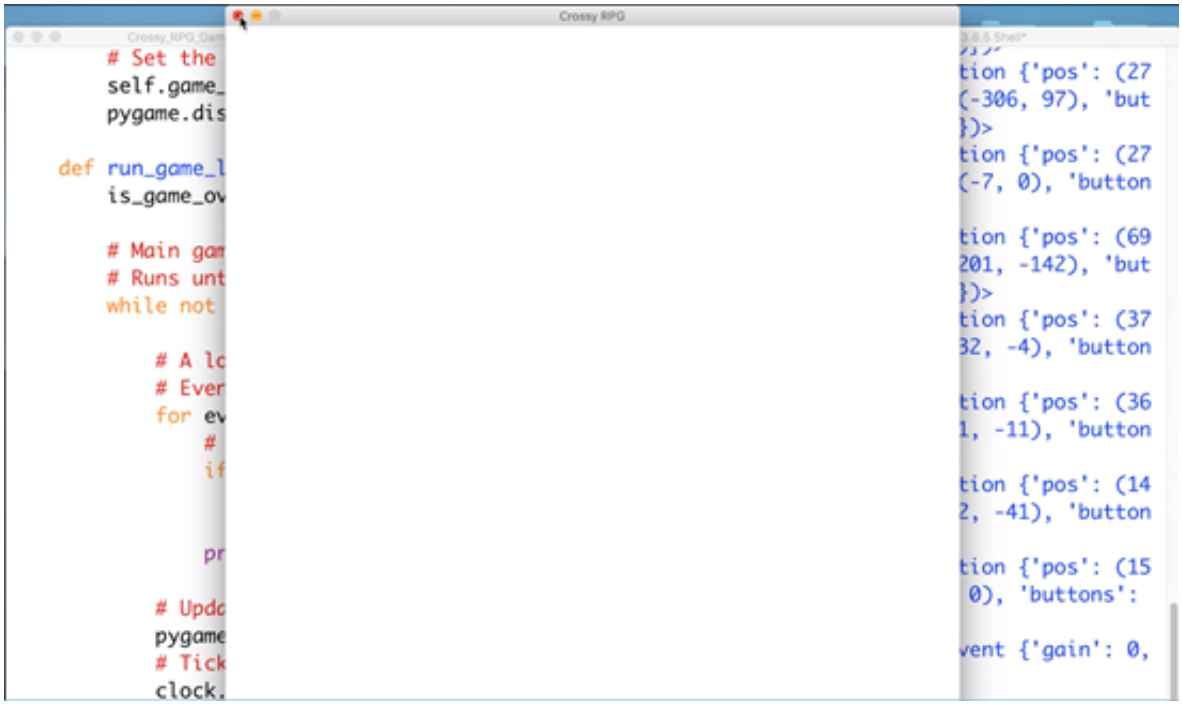
We will be putting all the code we have written already into one big game class. We will need to take the game loop and put this inside a function. This is important, especially for later on, but it will give us the ability to restart the game loop by calling on the function. Go ahead and open up your script and see the Python code below:

```
# Gain access to the pygame library
import pygame

# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)
# Clock used to update game events and frames
clock = pygame.time.Clock()
pygame.font.init()
font = pygame.font.SysFont('comicsans', 75)

class Game:

    # Typical rate of 60, equivalent to FPS
    TICK_RATE = 60

    # Initializer for the game class to set up the width, height, and title
    def __init__(self, title, width, height):
        self.title = title
        self.width = width
        self.height = height

        # Create the window of specified size in white to display the game
        self.game_screen = pygame.display.set_mode((width, height))
        # Set the game window color to white
        self.game_screen.fill(WHITE_COLOR)
        pygame.display.set_caption(title)

   def run_game_loop(self):
        is_game_over = False

         # Main game loop, used to update all gameplay such as movement, checks, and
graphics
         # Runs until is_game_over = True
        while not is_game_over:

         # A loop to get all of the events occuring at any given time
```



```
        # Events are most often mouse movement, mouse and button clicks, or exit eve
nts
         for event in pygame.event.get():
             # If we have a quite type event (exit out) then exit out of the game lo
op
               if event.type == pygame.QUIT:
                   is_game_over = True

               print(event)

        # Update all game graphics
        pygame.display.update()
        # Tick the clock to update everything within the game
        clock.tick(self.TICK_RATE)
pygame.init()

new_game = Game(SCREEN_TITLE, SCREEN_WIDTH, SCREEN_HEIGHT)
new_game.run_game_loop()

# Load the player image from the file directory
# player_image = pygame.image.load('player.png')
# Scale the image up
# player_image = pygame.transform.scale(player_image, (50, 50))

# Quit pygame and the program
pygame.quit()
quit()
```

Save the file and run it with **F5**.



We are not drawing anything to the screen, but the code is running and we can see the Crossy RPG title at the top of the window. In the next lesson we will start implementing a base class, which we can then use as a superclass.

In this lesson we will begin implementing our **game classes**, and we are going to start with a generic game object class. We will start writing the code we need for this lesson under the game class we already created in the previous lesson. Go ahead and open your script up and add the following Python code:

```
# Gain access to the pygame library
import pygame

# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)
# Clock used to update game events and frames
clock = pygame.time.Clock()
pygame.font.init()
font = pygame.font.SysFont('comicsans', 75)

class Game:

    # Typical rate of 60, equivalent to FPS
    TICK_RATE = 60

    # Initializer for the game class to set up the width, height, and title
    def __init__(self, title, width, height):
        self.title = title
        self.width = width
        self.height = height

        # Create the window of specified size in white to display the game
        self.game_screen = pygame.display.set_mode((width, height))
        # Set the game window color to white
        self.game_screen.fill(WHITE_COLOR)
        pygame.display.set_caption(title)

    def run_game_loop(self):
         is_game_over = False

# Main game loop, used to update all gameplay such as movement, checks, and graphics
# Runs until is_game_over = True
while not is_game_over:

         # A loop to get all of the events occuring at any given time
         # Events are most often mouse movement, mouse and button clicks, or exit eve
nts
         for event in pygame.event.get():
              # If we have a quite type event (exit out) then exit out of the game lo
op
                if event.type == pygame.QUIT:
                    is_game_over = True

                print(event)
```



```
# Update all game graphics
pygame.display.update()
# Tick the clock to update everything within the game
clock.tick(self.TICK_RATE)

# Generic game object class to be subclassed by other objects in the game
class GameObject:

    def __init__(self, image_path, x, y, width, height):
        object_image = pygame.image.load(image_path)
        # Scale the image up
        self.image = pygame.transform.scale(object_image, (width, height))

        self.x_pos = x
        self.y_pos = y

    # Draw the object by blitting it onto the background (game screen)
    def draw(self, background):
        background.blit(self.image, (self.x_pos, self.y_pos))

pygame.init()

new_game = Game(SCREEN_TITLE, SCREEN_WIDTH, SCREEN_HEIGHT)
new_game.run_game_loop()

# Load the player image from the file directory
# player_image = pygame.image.load('player.png')
# Scale the image up
# player_image = pygame.transform.scale(player_image, (50, 50))

# Quit pygame and the program
pygame.quit()
quit()
```

Save the file, and you may proceed to the next lesson in the course.

In this lesson we will begin adding in the code for the character class and the movement. We will be moving the character around the game screen using button presses. We will start adding the code we need for this lesson right below the GameObject class we wrote in the previous lesson. Open the "**Crossy_RPG_Game.py"** script, and begin adding the following Python code to it:

```
# Gain access to the pygame library
import pygame

# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)
# Clock used to update game events and frames
clock = pygame.time.Clock()
pygame.font.init()
font = pygame.font.SysFont('comicsans', 75)

class Game:

    # Typical rate of 60, equivalent to FPS
    TICK_RATE = 60

    # Initializer for the game class to set up the width, height, and title
    def __init__(self, title, width, height):
        self.title = title
        self.width = width
        self.height = height

        # Create the window of specified size in white to display the game
        self.game_screen = pygame.display.set_mode((width, height))
        # Set the game window color to white
        self.game_screen.fill(WHITE_COLOR)
        pygame.display.set_caption(title)

    def run_game_loop(self):
         is_game_over = False
         direction = 0

         player_character = PlayerCharacter('player.png', 375, 700, 50, 50)

         # Main game loop, used to update all gameplay such as movement, checks, and
graphics
         # Runs until is_game_over = True
         while not is_game_over:

         # A loop to get all of the events occuring at any given time
         # Events are most often mouse movement, mouse and button clicks, or exit eve
nts
           for event in pygame.event.get():
              # If we have a quite type event (exit out) then exit out of the game lo
op
                if event.type == pygame.QUIT:
```

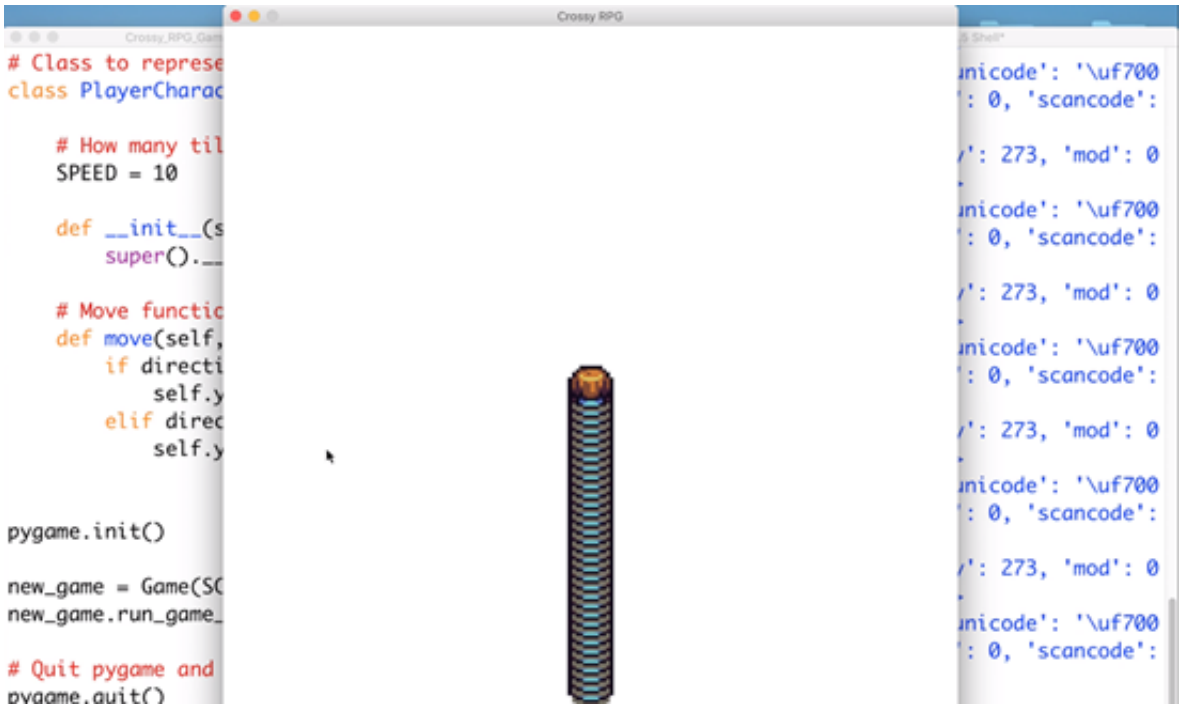


```
                is_game_over = True
            # Detect when key is pressed down
              elif event.type == pygame.KEYDOWN:
            # Move up if up key pressed
                if event.key == pygame.K_UP:
                    direction = 1
            # Move down if down key pressed
                elif event.key == pygame.K_DOWN:
                    direction = -1
            # Detect when key is released
              elif event.type == pygame.KEYUP:
            # Stop movement when key no longer pressed
                if event.key == pygame.K_UP or event.key == pygame.K_DOWN:
                    direction = 0

              print(event)

          # Update the player position
          player_character.move(direction)
          # Draw the player at the new position
          player_character.draw(self.game_screen)

        # Update all game graphics
        pygame.display.update()
        # Tick the clock to update everything within the game
        clock.tick(self.TICK_RATE)

# Generic game object class to be subclassed by other objects in the game
class GameObject:

    def __init__(self, image_path, x, y, width, height):
        object_image = pygame.image.load(image_path)
        # Scale the image up
        self.image = pygame.transform.scale(object_image, (width, height))

        self.x_pos = x
        self.y_pos = y

        self.width = width
        self.height = height

    # Draw the object by blitting it onto the background (game screen)
    def draw(self, background):
        background.blit(self.image, (self.x_pos, self.y_pos))

# Class to represent the character contolled by the player
class PlayerCharacter(GameObject):

    # How many tiles the character moves per second
    SPEED = 10

    def __init__(self, image_path, x, y, width, height):
        super().__init__(image_path, x, y, width, height)

# Move function will move character up if direction > 0 and down if < 0
```



```
    def move(self, direction):
        if direction > 0:
            self.y_pos -= self.SPEED
        elif direction < 0:
            self.y_pos += self.SPEED

pygame.init()

new_game = Game(SCREEN_TITLE, SCREEN_WIDTH, SCREEN_HEIGHT)
new_game.run_game_loop()

# Load the player image from the file directory
# player_image = pygame.image.load('player.png')
# Scale the image up
# player_image = pygame.transform.scale(player_image, (50, 50))

# Quit pygame and the program
pygame.quit()
quit()
```

Save this an run it using **F5:**



Okay this isn't quite working as we want it to because, its redrawing everything without updating the background. It's drawing everything on top of what is already there. What we want to do is draw the character and then in the next game loop iteration we want to draw a blank screen again over everything that happened here and then redraw the character. In our game class right before we draw the player character we need need to redraw our screen. Open the script back up and add the following Python code to it:

```
# Gain access to the pygame library
import pygame
```

```
# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)
# Clock used to update game events and frames
clock = pygame.time.Clock()
pygame.font.init()
font = pygame.font.SysFont('comicsans', 75)

class Game:

    # Typical rate of 60, equivalent to FPS
    TICK_RATE = 60

    # Initializer for the game class to set up the width, height, and title
    def __init__(self, title, width, height):
        self.title = title
        self.width = width
        self.height = height

        # Create the window of specified size in white to display the game
        self.game_screen = pygame.display.set_mode((width, height))
        # Set the game window color to white
        self.game_screen.fill(WHITE_COLOR)
        pygame.display.set_caption(title)

    def run_game_loop(self):
         is_game_over = False
         direction = 0

         player_character = PlayerCharacter('player.png', 375, 700, 50, 50)

# Main game loop, used to update all gameplay such as movement, checks, and graphics
# Runs until is_game_over = True
while not is_game_over:

         # A loop to get all of the events occuring at any given time
         # Events are most often mouse movement, mouse and button clicks, or exit eve
nts
         for event in pygame.event.get():
              # If we have a quite type event (exit out) then exit out of the game lo
op
                if event.type == pygame.QUIT:
                    is_game_over = True
              # Detect when key is pressed down
                elif event.type == pygame.KEYDOWN:
              # Move up if up key pressed
                    if event.key == pygame.K_UP:
                        direction = 1
              # Move down if down key pressed
                    elif event.key == pygame.K_DOWN:
```

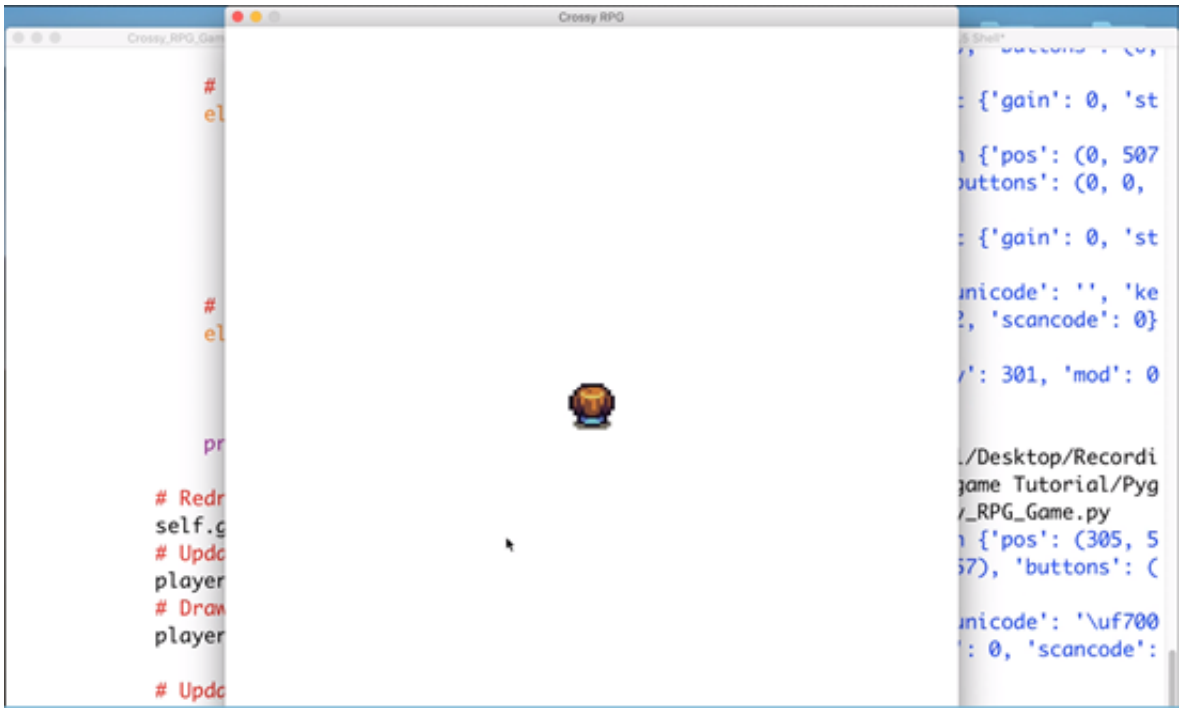


```
                    direction = -1
            # Detect when key is released
              elif event.type == pygame.KEYUP:
            # Stop movement when key no longer pressed
                  if event.key == pygame.K_UP or event.key == pygame.K_DOWN:
                      direction = 0

              print(event)

          # Redraw the screen to be a blank white window
          self.game_screen.fill(WHITE_COLOR)
          # Update the player position
          player_character.move(direction)
          # Draw the player at the new position
          player_character.draw(self.game_screen)

          # Update all game graphics
          pygame.display.update()
          # Tick the clock to update everything within the game
          clock.tick(self.TICK_RATE)

# Generic game object class to be subclassed by other objects in the game
class GameObject:

    def __init__(self, image_path, x, y, width, height):
        object_image = pygame.image.load(image_path)
        # Scale the image up
        self.image = pygame.transform.scale(object_image, (width, height))

        self.x_pos = x
        self.y_pos = y

        self.width = width
        self.height = height

    # Draw the object by blitting it onto the background (game screen)
    def draw(self, background):
        background.blit(self.image, (self.x_pos, self.y_pos))

# Class to represent the character contolled by the player
class PlayerCharacter(GameObject):

    # How many tiles the character moves per second
    SPEED = 10

    def __init__(self, image_path, x, y, width, height):
        super().__init__(image_path, x, y, width, height)

# Move function will move character up if direction > 0 and down if < 0
    def move(self, direction):
        if direction > 0:
            self.y_pos -= self.SPEED
        elif direction < 0:
            self.y_pos += self.SPEED
```



```
pygame.init()

new_game = Game(SCREEN_TITLE, SCREEN_WIDTH, SCREEN_HEIGHT)
new_game.run_game_loop()

# Load the player image from the file directory
# player_image = pygame.image.load('player.png')
# Scale the image up
# player_image = pygame.transform.scale(player_image, (50, 50))

# Quit pygame and the program
pygame.quit()
quit()
```

Save this and run it with **F5:**



So now we can move forwards and backwards. The movement isn't quite perfect with keeping up with the key presses. Right now the player character can also go off the screen, and we don't want that happening. In the next lesson we will implement some bounds checking and the enemy character class.

Code Correction in the Lesson Notes: The code in the video for this particular lesson has been updated, please see the lesson notes below for the corrected code.

In this lesson we will be implementing the enemy character class and bounds checking. We will need to make sure the enemy characters cannot move off the screen. The player character also needs to not be able to move off the screen. The enemy characters will be moving left and right across the screen. This is why the bounds checking is going to be so important for the enemies. The base of the enemy class is going to be very similar to what we already have for the PlayerCharacter class. Open up the script so we can add the following Python code to it**.**

The following code has been updated, and differs from the video:

```
# Gain access to the pygame library
import pygame

# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)
# Clock used to update game events and frames
clock = pygame.time.Clock()
pygame.font.init()
font = pygame.font.SysFont('comicsans', 75)

class Game:

    # Typical rate of 60, equivalent to FPS
    TICK_RATE = 60

    # Initializer for the game class to set up the width, height, and title
    def __init__(self, title, width, height):
        self.title = title
        self.width = width
        self.height = height

        # Create the window of specified size in white to display the game
        self.game_screen = pygame.display.set_mode((width, height))
        # Set the game window color to white
        self.game_screen.fill(WHITE_COLOR)
        pygame.display.set_caption(title)

    def run_game_loop(self):
         is_game_over = False
         direction = 0

         player_character = PlayerCharacter('player.png', 375, 700, 50, 50)
         enemy_0 = NonPlayerCharacter('enemy.png', 20, 600, 50, 50)

# Main game loop, used to update all gameplay such as movement, checks, and graphics
# Runs until is_game_over = True
while not is_game_over:
```



```
        # A loop to get all of the events occuring at any given time
        # Events are most often mouse movement, mouse and button clicks, or exit eve
nts
        for event in pygame.event.get():
            # If we have a quite type event (exit out) then exit out of the game lo
op
              if event.type == pygame.QUIT:
                  is_game_over = True
            # Detect when key is pressed down
              elif event.type == pygame.KEYDOWN:
            # Move up if up key pressed
                  if event.key == pygame.K_UP:
                      direction = 1
            # Move down if down key pressed
                  elif event.key == pygame.K_DOWN:
                      direction = -1
            # Detect when key is released
              elif event.type == pygame.KEYUP:
            # Stop movement when key no longer pressed
                  if event.key == pygame.K_UP or event.key == pygame.K_DOWN:
                      direction = 0

              print(event)

           # Redraw the screen to be a blank white window
           self.game_screen.fill(WHITE_COLOR)
           # Update the player position
           player_character.move(direction)
           # Draw the player at the new position
           player_character.draw(self.game_screen)

           # Move and draw the enemy character
           enemy_0.move(self.width)
           enemy_0.draw(self.game_screen)

           # Update all game graphics
           pygame.display.update()
           # Tick the clock to update everything within the game
           clock.tick(self.TICK_RATE)

# Generic game object class to be subclassed by other objects in the game
class GameObject:

    def __init__(self, image_path, x, y, width, height):
        object_image = pygame.image.load(image_path)
        # Scale the image up
        self.image = pygame.transform.scale(object_image, (width, height))

        self.x_pos = x
        self.y_pos = y

        self.width = width
        self.height = height

    # Draw the object by blitting it onto the background (game screen)
```



```
    def draw(self, background):
        background.blit(self.image, (self.x_pos, self.y_pos))

# Class to represent the character contolled by the player
class PlayerCharacter(GameObject):

    # How many tiles the character moves per second
    SPEED = 10

    def __init__(self, image_path, x, y, width, height):
        super().__init__(image_path, x, y, width, height)

# Move function will move character up if direction > 0 and down if < 0
    def move(self, direction):
        if direction > 0:
            self.y_pos -= self.SPEED
        elif direction < 0:
            self.y_pos += self.SPEED

# Class to represent the enemies moving left to right and right to left
class NonPlayerCharacter(GameObject):

    # How many tiles the character moves per second
    SPEED = 10

    def __init__(self, image_path, x, y, width, height):
        super().__init__(image_path, x, y, width, height)

    # Move function will move character right once it hits the far left of the
    # screen and left once it hits the far right of the screen
    def move(self, max_width):
        if self.x_pos <= 20:
            self.SPEED = abs(self.SPEED)
        elif self.x_pos >=  max_width - 20:
            self.SPEED = -abs(self.SPEED)
        self.x_pos += self.SPEED


pygame.init()

new_game = Game(SCREEN_TITLE, SCREEN_WIDTH, SCREEN_HEIGHT)
new_game.run_game_loop()

# Load the player image from the file directory
# player_image = pygame.image.load('player.png')
# Scale the image up
# player_image = pygame.transform.scale(player_image, (50, 50))

# Quit pygame and the program
pygame.quit()
quit()
```

Save this and run it with **F5:**

It looks pretty good, we have the enemy moving left and right. It does go off the screen a little bit when it moves to the right side. This may be because the width is not being computed properly. Open the script back up so we can fix the issue in the code.

The following code has been updated, and differs from the video:

```
# Gain access to the pygame library
import pygame

# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)
# Clock used to update game events and frames
clock = pygame.time.Clock()
pygame.font.init()
font = pygame.font.SysFont('comicsans', 75)

class Game:

    # Typical rate of 60, equivalent to FPS
    TICK_RATE = 60

    # Initializer for the game class to set up the width, height, and title
    def __init__(self, title, width, height):
        self.title = title
        self.width = width
        self.height = height

        # Create the window of specified size in white to display the game
```



```
        self.game_screen = pygame.display.set_mode((width, height))
        # Set the game window color to white
        self.game_screen.fill(WHITE_COLOR)
        pygame.display.set_caption(title)

    def run_game_loop(self):
         is_game_over = False
         direction = 0

         player_character = PlayerCharacter('player.png', 375, 700, 50, 50)
         enemy_0 = NonPlayerCharacter('enemy.png', 20, 600, 50, 50)

# Main game loop, used to update all gameplay such as movement, checks, and graphics
# Runs until is_game_over = True
while not is_game_over:

         # A loop to get all of the events occuring at any given time
         # Events are most often mouse movement, mouse and button clicks, or exit eve
nts
         for event in pygame.event.get():
              # If we have a quite type event (exit out) then exit out of the game lo
op
                if event.type == pygame.QUIT:
                    is_game_over = True
              # Detect when key is pressed down
                elif event.type == pygame.KEYDOWN:
              # Move up if up key pressed
                    if event.key == pygame.K_UP:
                        direction = 1
              # Move down if down key pressed
                    elif event.key == pygame.K_DOWN:
                        direction = -1
              # Detect when key is released
                elif event.type == pygame.KEYUP:
              # Stop movement when key no longer pressed
                    if event.key == pygame.K_UP or event.key == pygame.K_DOWN:
                        direction = 0

                print(event)

            # Redraw the screen to be a blank white window
            self.game_screen.fill(WHITE_COLOR)
            # Update the player position
            player_character.move(direction)
            # Draw the player at the new position
            player_character.draw(self.game_screen)

            # Move and draw the enemy character
            enemy_0.move(self.width)
            enemy_0.draw(self.game_screen)

            # Update all game graphics
            pygame.display.update()
            # Tick the clock to update everything within the game
            clock.tick(self.TICK_RATE)
```



```
# Generic game object class to be subclassed by other objects in the game
class GameObject:

    def __init__(self, image_path, x, y, width, height):
        object_image = pygame.image.load(image_path)
        # Scale the image up
        self.image = pygame.transform.scale(object_image, (width, height))

        self.x_pos = x
        self.y_pos = y

        self.width = width
        self.height = height

    # Draw the object by blitting it onto the background (game screen)
    def draw(self, background):
        background.blit(self.image, (self.x_pos, self.y_pos))

# Class to represent the character contolled by the player
class PlayerCharacter(GameObject):

    # How many tiles the character moves per second
    SPEED = 10

    def __init__(self, image_path, x, y, width, height):
        super().__init__(image_path, x, y, width, height)

# Move function will move character up if direction > 0 and down if < 0
    def move(self, direction, max_height):
        if direction > 0:
            self.y_pos -= self.SPEED
        elif direction < 0:
            self.y_pos += self.SPEED
# Make sure the character never goes past the bottom of the screen
        if self.y_pos >= max_height - 20:
            self.y_pos = max_height - 20

# Class to represent the enemies moving left to right and right to left
class NonPlayerCharacter(GameObject):

    # How many tiles the character moves per second
    SPEED = 10

    def __init__(self, image_path, x, y, width, height):
        super().__init__(image_path, x, y, width, height)

    # Move function will move character right once it hits the far left of the
    # screen and left once it hits the far right of the screen
    def move(self, max_width):
        if self.x_pos <= 20:
            self.SPEED = abs(self.SPEED)
        elif self.x_pos >=  max_width - 20:
            self.SPEED = -abs(self.SPEED)
        self.x_pos += self.SPEED
```



```
pygame.init()

new_game = Game(SCREEN_TITLE, SCREEN_WIDTH, SCREEN_HEIGHT)
new_game.run_game_loop()

# Load the player image from the file directory
# player_image = pygame.image.load('player.png')
# Scale the image up
# player_image = pygame.transform.scale(player_image, (50, 50))

# Quit pygame and the program
pygame.quit()
quit()
```

Save the file. In the next lesson we will be working on implementing the collision detection between the character, treasure, and enemies.

The video only shows the implementation of collision detection for one enemy. For multiple enemies, please consult the **Lesson Notes** available below the Video Player.

Code Correction in the Lesson Notes: The code in the video for this particular lesson has been updated, please see the lesson notes below for the corrected code.

In this lesson we will be implementing the collision detection. By the end of this lesson we will have completed most of the game logic for the Crossy RPG game. We are going to be setting up collision between the player character and the treasure, and the player character and the enemies.

Built-In Collision Detection

There are some built in collision detection functions in Pygame itself. They work best if you use the built-in Pygame classes, such as rectangles, and the built-in sprite classes. It's important to understand how collision detection works when developing games. It's always good to implement your own stuff before you start using built-in functions.

Tying Up Loose Ends

There was something that was left out of the previous lesson and needs to be added to the code. This was with the player character movement. We need to pass in the height. Open the script up and add the following Python code:

```
# Update the player position
player_character.move(direction, self.height)
```

Save this.

Increase the Bounds Checks

We need to increase the bounds checks because the player character is able to move down too far and leave the screens till. The enemy can also move too far to the right. Open the script up and the following code needs to be changed in the PlayerCharacter class:

```
# Make sure the character never goes past the bottom of the screen
if self.y_pos >= max_height - 40:
    self.y_pos = max_height - 40
```

Then make the change in the NonPlayerCharacter class:

```
 # Move function will move character right once it hits the far left of the
    # screen and left once it hits the far right of the screen
    def move(self, max_width):
        if self.x_pos <= 20:
            self.SPEED = abs(self.SPEED)
        elif self.x_pos >= max_width - 40:
            self.SPEED = -abs(self.SPEED)
        self.x_pos += self.SPEED
```

Save these changes.

PlayerCharacter Collision

We should now start to work on how we are going to start collision detection here. We have the player character, and we are not concerned at all about collision until we reach the same Y position as the enemy. As soon as we pass the enemy we will never collide with it. We will need to add some code the the PlayerCharacter class, open up the script and add the following Python code:

```
# Return False (no collision) if y positions and x positions do not overlap
    # Return True x and y positions overlap
    def detect_collision(self, other_body):
        if self.y_pos > other_body.y_pos + other_body.height:
            return False
        elif self.y_pos + self.height < other_body.y_pos:
            return False

        if self.x_pos > other_body.x_pos + other_body.width:
            return False
        elif self.x_pos + self.width < other_body.x_pos:
            return False

        return True
```

Save this. We now have implemented the player collision, and this is the way that required the least amount of code to be written.

Load the Treasure

Add the following code to the script, we need to load the treasure image and render it.

The following code has been updated, and differs from the video. The updated code is marked with an 'Update Code' comment:

```
treasure = GameObject('treasure.png', 375, 50, 50, 50)

# Draw the treasure
treasure.draw(self.game_screen)

 # End game if collision between enemy and treasure
            # Close game if we lose
            # Restart game loop if we win
            # Updated Code - check for collision with all enemies
            enemies = [enemy_0, enemy_1, enemy_2]
            if player_character.detect_collision(treasure):
                is_game_over = True
            else
              for enemy in enemies:
                 if player_character.detect_collision(enemy):
                   is_game_over = True
```

Save the file and run it using **F5**.

We have the treasure image being rendered on the screen at the very top and the collision with the enemy and the treasure is working properly. We now have most of the functionality of our game complete. In the next lesson we will implement winning and losing conditions along with the endgame logic.

In this lesson we will implementing the true end game conditions, this is going to be keeping specific track of whether or not we have won or lost. What we are going to do is just display some simple **“You Win”** or “**You Lose”** text on the screen, we then will restart the game. We will add some code in the game loop, open the script up and add the following Python code to it:

```
did_win = False

# End game if collision between enemy and treasure
# Close game if we lose
# Restart game loop if we win
            if player_character.detect_collision(enemy_0):
                is_game_over = True
                did_win = False
elif player_character.detect_collision(treasure):
                is_game_over = True
                did_win = True
```

Save this. We now need to initialize the font, so that we can display it on the screen when we win or lose.

The Pygame Font

Add the following Python code to the script for the font initialization and implementation:

```
pygame.font.init()
font = pygame.font.SysFont('comicsans', 75)

 # End game if collision between enemy and treasure
            # Close game if we lose
            # Restart game loop if we win
            if player_character.detect_collision(enemy_0):
                is_game_over = True
                did_win = False
                text = font.render('You lose! :(', True, BLACK_COLOR)
                self.game_screen.blit(text, (300, 350))
                pygame.display.update()
                clock.tick(1)
                break
            elif player_character.detect_collision(treasure):
                is_game_over = True
                did_win = True
                text = font.render('You win! :)', True, BLACK_COLOR)
                self.game_screen.blit(text, (300, 350))
                pygame.display.update()
                clock.tick(1)
                break

# Restart game loop if we won
        # Break out of game loop and quit if we lose
        if did_win:
            self.run_game_loop()
        else:
            return
```


Save this. We are now going to load in the background image and then have it display.

Loading and Displaying the Background Image

Open the script and add the following code to the **Initializer** for the game:

```
 # Initializer for the game class to set up the width, height, and title
    def __init__(self, image_path, title, width, height):
        self.title = title
        self.width = width
        self.height = height

pygame.init()

new_game = Game('background.png', SCREEN_TITLE, SCREEN_WIDTH, SCREEN_HEIGHT)

  # Load and set the background image for the scene
        background_image = pygame.image.load(image_path)
        self.image = pygame.transform.scale(background_image, (width, height))

 # Draw the image onto the background
         self.game_screen.blit(self.image, (0, 0))
```

Save this and run it with **F5:**

There we go, we have a nice background, and it looks a lot better now. The lose and win conditions both work correctly:

We now have a very simple Crossy RPG type of game. In the next lesson you will be shown ways in which you might be able to improve the game a little bit. This might be increasing the difficulty a little bit, this might be adding some more sprites and making them move faster.

In this lesson we will begin working on making the Crossy RPG game a little more interesting. We will add some more enemies and make them move a bit faster as we progress through levels. So if we win, we increase the speed of the enemies, and then when we win again we increase the speed again. This will make the game progressively difficult the longer it gets played.

The Speed Aspect

We will work on the speed aspect of making the game a little more difficult first. This is actually quite easy to do as well. We can make this change in the game loop, we will take in a parameter for the **run_game_loop function** called **level_speed**. Go ahead and open up the script and we will add some Python code to it:

```
def run_game_loop(self, level_speed):




# Speed increased as we advance in difficulty
enemy_0.SPEED *= level_speed



 # Restart game loop if we won
        # Break out of game loop and quit if we lose
        if did_win:
            self.run_game_loop(level_speed + 0.5)
        else:
            return


new_game.run_game_loop(1)
```

Save these changes to the script and run it with **F5:**

The speed of the enemy does get faster as we win. The **level_speed** functionality we implemented is working correctly.

Adding More Enemies

Another thing we are going to do to increase difficulty is add in some more enemies once we reach a certain level. We can add in the enemies right away, we will add in an enemy one and enemy two. Open the script back up and add the following code to it:

```
# Move and draw more enemies when we reach higher levels of difficulty
            if level_speed > 3:
                enemy_1.move(self.width)
                enemy_1.draw(self.game_screen)
            if level_speed > 5:
                enemy_2.move(self.width)
                enemy_2.draw(self.game_screen)
```

Save the changes and give it a run using **F5:**

The enemies are spawning once we get to the correct level as we win. What we want to do now is change the **level_speed** values from **3** and **5** to **2** and **4**. Go ahead and open the script up and change the following Python code in it:

```
# Move and draw more enemies when we reach higher levels of difficulty
            if level_speed > 2:
                enemy_1.move(self.width)
                enemy_1.draw(self.game_screen)
            if level_speed > 4:
                enemy_2.move(self.width)
                enemy_2.draw(self.game_screen)
```

Changing the values will just make it so that the player gets to encounter multiple enemies much sooner in the level progression. So for this we introduced a couple of ways to improve things for the Crossy RPG game. We increased the **level_ speed** as we progressively keep winning at the game. We also introduced two more enemies. In the next lesson we will just do a summary and wrap things up for this course.

This is the final lesson for the Pygame tutorials, and the final lesson for the entire course. All we are going to do here is run through a quick summary of what we did in the course.

Summary

1. We started by installing Python and Pygame.
2. We went over Python language basics: **Variables, collections, control flow, functions, and classes**.
3. We went over the Python basics we were able to actually begin the game development.
4. We started developing the Crossy RPG with Pygame.
5. We built up the Crossy RPG from scratch, we had nothing on the screen to start off with and now we have a fully working game.

The First Steps for Crossy RPG

We began the development of the game by how we would set up the game screen. We specified the **width**, **height**, and **title** for the game. Set the **color** of the background. Created the window of specified size in white to display the game.

```
# Create the window of specified size in white to display the game
self.game_screen = pygame.display.set_mode((width, height))
```

Set the game window color to white and the title of the game in the window.

```
# Set the game window color to white
self.game_screen.fill(WHITE_COLOR)
pygame.display.set_caption(title)
```

Introduced the Game Loop

We introduced thegame loop**,** and then ended up putting it in the function called **“run_game_loop”**

```
def run_game_loop(self, level_speed):
        is_game_over = False
        did_win = False
        direction = 0
```

We then used the while not **is_game_over** loop outside the function, the loop is used to do everything within the game. Its used for position, collision checks, player movement, enemy movement, update and render the graphics.

```
# Main game loop, used to update all gameplay such as movement, checks, and graphics
        # Runs until is_game_over = True
        while not is_game_over:

            # A loop to get all of the events occuring at any given time
            # Events are most often mouse movement, mouse and button clicks, or exit
events
            for event in pygame.event.get():
```



```
                # If we have a quite type event (exit out) then exit out of the game
loop
                if event.type == pygame.QUIT:
                    is_game_over = True
                # Detect when key is pressed down
                elif event.type == pygame.KEYDOWN:
                    # Move up if up key pressed
                    if event.key == pygame.K_UP:
                        direction = 1
                    # Move down if down key pressed
                    elif event.key == pygame.K_DOWN:
                        direction = -1
                # Detect when key is released
                elif event.type == pygame.KEYUP:
                    # Stop movement when key no longer pressed
                    if event.key == pygame.K_UP or event.key == pygame.K_DOWN:
                        direction = 0
                print(event)

            # Redraw the screen to be a blank white window
            self.game_screen.fill(WHITE_COLOR)
            # Draw the image onto the background
            self.game_screen.blit(self.image, (0, 0))

            # Draw the treasure
            treasure.draw(self.game_screen)

            # Update the player position
            player_character.move(direction, self.height)
            # Draw the player at the new position
            player_character.draw(self.game_screen)

            # Move and draw the enemy character
            enemy_0.move(self.width)
            enemy_0.draw(self.game_screen)

            # Move and draw more enemies when we reach higher levels of difficulty
            if level_speed > 2:
                enemy_1.move(self.width)
                enemy_1.draw(self.game_screen)
            if level_speed > 4:
                enemy_2.move(self.width)
                enemy_2.draw(self.game_screen)

            # End game if collision between enemy and treasure
            # Close game if we lose
            # Restart game loop if we win
            if player_character.detect_collision(enemy_0):
                is_game_over = True
                did_win = False
                text = font.render('You lose! :(', True, BLACK_COLOR)
                self.game_screen.blit(text, (275, 350))
                pygame.display.update()
                clock.tick(1)
                break
```



```
            elif player_character.detect_collision(treasure):
                is_game_over = True
                did_win = True
                text = font.render('You win! :)', True, BLACK_COLOR)
                self.game_screen.blit(text, (275, 350))
                pygame.display.update()
                clock.tick(1)
                break

            # Update all game graphics
            pygame.display.update()
            # Tick the clock to update everything within the game
            clock.tick(self.TICK_RATE)

        # Restart game loop if we won
        # Break out of game loop and quit if we lose
        if did_win:
            self.run_game_loop(level_speed + 0.5)
        else:
            return
```

Introduced the Drawing of Objects on the Screen

We introduced the notion of drawing objects on the screen, we actually have that code down at the bottom of the script and we then commented it out, we talked about how to “blit” images, so import images into the project, and then built them onto our screen, and then the actual image itself.

```
# Load the player image from the file directory
# player_image = pygame.image.load('player.png')
# Scale the image up
# player_image = pygame.transform.scale(player_image, (50, 50))

# Draw a rectangle on top of the game screen canvas (x, y, width, height)
            # pygame.draw.rect(game_screen, BLACK_COLOR, [350, 350, 100, 100])
            # Draw a circle on top of the game screen (x, y, radius)
            # pygame.draw.circle(game_screen, BLACK_COLOR, (400, 300), 50)

            # Draw the player image on top of the screen at (x, y) position
            # game_screen.blit(player_image, (375, 375))
```

More Object-Oriented Code

We worked on making the code we wrote more object oriented. We created the game class where we are using are initializer to setup the screen, windows setting, width and height, background image, and we put the run game loop inside of a function. We created the various classes, we started with a generic game object type class, which we then sub-classed with our Player Character and our Non Player Character classes.

```
# Generic game object class to be subclassed by other objects in the game
class GameObject:
```



```
    def __init__(self, image_path, x, y, width, height):
        object_image = pygame.image.load(image_path)
        # Scale the image up
        self.image = pygame.transform.scale(object_image, (width, height))

        self.x_pos = x
        self.y_pos = y

        self.width = width
        self.height = height

    # Draw the object by blitting it onto the background (game screen)
    def draw(self, background):
        background.blit(self.image, (self.x_pos, self.y_pos))



# Class to represent the character contolled by the player
class PlayerCharacter(GameObject):

    # How many tiles the character moves per second
    SPEED = 10

    def __init__(self, image_path, x, y, width, height):
        super().__init__(image_path, x, y, width, height)




# Class to represent the enemies moving left to right and right to left
class NonPlayerCharacter(GameObject):

    # How many tiles the character moves per second
    SPEED = 10

    def __init__(self, image_path, x, y, width, height):
        super().__init__(image_path, x, y, width, height)
```

We then worked on implementing movement into the move function.

```
# Move function will move character up if direction > 0 and down if < 0
    def move(self, direction, max_height):
        if direction > 0:
            self.y_pos -= self.SPEED
        elif direction < 0:
            self.y_pos += self.SPEED
        # Make sure the character never goes past the bottom of the screen
        if self.y_pos >= max_height - 40:
            self.y_pos = max_height - 40
```


Then we updated the graphics.

```
  # Redraw the screen to be a blank white window
            self.game_screen.fill(WHITE_COLOR)
            # Draw the image onto the background
            self.game_screen.blit(self.image, (0, 0))

            # Draw the treasure
            treasure.draw(self.game_screen)

            # Update the player position
            player_character.move(direction, self.height)
            # Draw the player at the new position
            player_character.draw(self.game_screen)

            # Move and draw the enemy character
            enemy_0.move(self.width)
            enemy_0.draw(self.game_screen)
```

We talked about bounds checking.

```
# Class to represent the enemies moving left to right and right to left
class NonPlayerCharacter(GameObject):

    # How many tiles the character moves per second
    SPEED = 10

    def __init__(self, image_path, x, y, width, height):
        super().__init__(image_path, x, y, width, height)

    # Move function will move character right once it hits the far left of the
    # screen and left once it hits the far right of the screen
    def move(self, max_width):
        if self.x_pos <= 20:
            self.SPEED = abs(self.SPEED)
        elif self.x_pos >= max_width - 40:
            self.SPEED = -abs(self.SPEED)
        self.x_pos += self.SPEED
```

Added the collision detection function.

```
  # Return False (no collision) if y positions and x positions do not overlap
    # Return True x and y positions overlap
    def detect_collision(self, other_body):
        if self.y_pos > other_body.y_pos + other_body.height:
            return False
        elif self.y_pos + self.height < other_body.y_pos:
```



```
            return False

        if self.x_pos > other_body.x_pos + other_body.width:
            return False
        elif self.x_pos + self.width < other_body.x_pos:
            return False

        return True
```

Winning and Losing Game Logic

We introduced and implemented the winning and losing conditions to the game.

```
  # Restart game loop if we won
        # Break out of game loop and quit if we lose
        if did_win:
            self.run_game_loop(level_speed + 0.5)
        else:
            return
```

Made the Game More Interesting

The final implementation for the Crossy RPG game was to make the game more interesting by adding more enemies.

```
  # Create another enemy
        enemy_1 = NonPlayerCharacter('enemy.png', self.width - 40, 400, 50, 50)
        enemy_1.SPEED *= level_speed

        # Create another enemy
        enemy_2 = NonPlayerCharacter('enemy.png', 20, 200, 50, 50)
        enemy_2.SPEED *= level_speed
```

Then making it progressively more difficult by increasing the speed at which the enemies are moving.

```
 def run_game_loop(self, level_speed):

# Speed increased as we advance in difficulty
        enemy_0.SPEED *= level_speed
```

Thanks!

Thank you so much for taking the course. You now have a fully functional Pygame made with Python and can make lots of great additions to it with your new knowledge. You can also start creating your

new games, and feel free to share the games with us as you create them.

The Script

Here is the entire script **“Crossy_RPG_Game.py”**

```
# Summary!
# Installing Python and Pygame
# Python language basics (variables, collections, control flow, functions, classes)
# Developing our game while learning about Pygame

# Gain access to the pygame library
import pygame

# Size of the screen
SCREEN_TITLE = 'Crossy RPG'
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 800
# Colors according to RGB codes
WHITE_COLOR = (255, 255, 255)
BLACK_COLOR = (0, 0, 0)
# Clock used to update game events and frames
clock = pygame.time.Clock()
pygame.font.init()
font = pygame.font.SysFont('comicsans', 75)

class Game:

    # Typical rate of 60, equivalent to FPS
    TICK_RATE = 60

    # Initializer for the game class to set up the width, height, and title
    def __init__(self, image_path, title, width, height):
        self.title = title
        self.width = width
        self.height = height

        # Create the window of specified size in white to display the game
        self.game_screen = pygame.display.set_mode((width, height))
        # Set the game window color to white
        self.game_screen.fill(WHITE_COLOR)
        pygame.display.set_caption(title)

        # Load and set the background image for the scene
        background_image = pygame.image.load(image_path)
        self.image = pygame.transform.scale(background_image, (width, height))

    def run_game_loop(self, level_speed):
        is_game_over = False
        did_win = False
        direction = 0

        player_character = PlayerCharacter('player.png', 375, 700, 50, 50)
        enemy_0 = NonPlayerCharacter('enemy.png', 20, 600, 50, 50)
        # Speed increased as we advance in difficulty
```



```
        enemy_0.SPEED *= level_speed

        # Create another enemy
        enemy_1 = NonPlayerCharacter('enemy.png', self.width - 40, 400, 50, 50)
        enemy_1.SPEED *= level_speed

        # Create another enemy
        enemy_2 = NonPlayerCharacter('enemy.png', 20, 200, 50, 50)
        enemy_2.SPEED *= level_speed

        treasure = GameObject('treasure.png', 375, 50, 50, 50)

        # Main game loop, used to update all gameplay such as movement, checks, and g
raphics
        # Runs until is_game_over = True
        while not is_game_over:

            # A loop to get all of the events occuring at any given time
            # Events are most often mouse movement, mouse and button clicks, or exit
events
            for event in pygame.event.get():
                # If we have a quite type event (exit out) then exit out of the game
loop
                if event.type == pygame.QUIT:
                    is_game_over = True
                # Detect when key is pressed down
                elif event.type == pygame.KEYDOWN:
                    # Move up if up key pressed
                    if event.key == pygame.K_UP:
                        direction = 1
                    # Move down if down key pressed
                    elif event.key == pygame.K_DOWN:
                        direction = -1
                # Detect when key is released
                elif event.type == pygame.KEYUP:
                    # Stop movement when key no longer pressed
                    if event.key == pygame.K_UP or event.key == pygame.K_DOWN:
                        direction = 0
                print(event)

            # Redraw the screen to be a blank white window
            self.game_screen.fill(WHITE_COLOR)
            # Draw the image onto the background
            self.game_screen.blit(self.image, (0, 0))

            # Draw the treasure
            treasure.draw(self.game_screen)

            # Update the player position
            player_character.move(direction, self.height)
            # Draw the player at the new position
            player_character.draw(self.game_screen)

            # Move and draw the enemy character
            enemy_0.move(self.width)
```



```
            enemy_0.draw(self.game_screen)

            # Move and draw more enemies when we reach higher levels of difficulty
            if level_speed > 2:
                enemy_1.move(self.width)
                enemy_1.draw(self.game_screen)
            if level_speed > 4:
                enemy_2.move(self.width)
                enemy_2.draw(self.game_screen)

            # End game if collision between enemy and treasure
            # Close game if we lose
            # Restart game loop if we win
            if player_character.detect_collision(enemy_0):
                is_game_over = True
                did_win = False
                text = font.render('You lose! :(', True, BLACK_COLOR)
                self.game_screen.blit(text, (275, 350))
                pygame.display.update()
                clock.tick(1)
                break
            elif player_character.detect_collision(treasure):
                is_game_over = True
                did_win = True
                text = font.render('You win! :)', True, BLACK_COLOR)
                self.game_screen.blit(text, (275, 350))
                pygame.display.update()
                clock.tick(1)
                break

            # Update all game graphics
            pygame.display.update()
            # Tick the clock to update everything within the game
            clock.tick(self.TICK_RATE)

        # Restart game loop if we won
        # Break out of game loop and quit if we lose
        if did_win:
            self.run_game_loop(level_speed + 0.5)
        else:
            return

# Generic game object class to be subclassed by other objects in the game
class GameObject:

    def __init__(self, image_path, x, y, width, height):
        object_image = pygame.image.load(image_path)
        # Scale the image up
        self.image = pygame.transform.scale(object_image, (width, height))

        self.x_pos = x
        self.y_pos = y

        self.width = width
        self.height = height
```



```
    # Draw the object by blitting it onto the background (game screen)
    def draw(self, background):
        background.blit(self.image, (self.x_pos, self.y_pos))

# Class to represent the character contolled by the player
class PlayerCharacter(GameObject):

    # How many tiles the character moves per second
    SPEED = 10

    def __init__(self, image_path, x, y, width, height):
        super().__init__(image_path, x, y, width, height)

    # Move function will move character up if direction > 0 and down if < 0
    def move(self, direction, max_height):
        if direction > 0:
            self.y_pos -= self.SPEED
        elif direction < 0:
            self.y_pos += self.SPEED
        # Make sure the character never goes past the bottom of the screen
        if self.y_pos >= max_height - 40:
            self.y_pos = max_height - 40

    # Return False (no collision) if y positions and x positions do not overlap
    # Return True x and y positions overlap
    def detect_collision(self, other_body):
        if self.y_pos > other_body.y_pos + other_body.height:
            return False
        elif self.y_pos + self.height < other_body.y_pos:
            return False

        if self.x_pos > other_body.x_pos + other_body.width:
            return False
        elif self.x_pos + self.width < other_body.x_pos:
            return False

        return True

# Class to represent the enemies moving left to right and right to left
class NonPlayerCharacter(GameObject):

    # How many tiles the character moves per second
    SPEED = 10

    def __init__(self, image_path, x, y, width, height):
        super().__init__(image_path, x, y, width, height)

    # Move function will move character right once it hits the far left of the
    # screen and left once it hits the far right of the screen
    def move(self, max_width):
        if self.x_pos <= 20:
            self.SPEED = abs(self.SPEED)
        elif self.x_pos >= max_width - 40:
            self.SPEED = -abs(self.SPEED)
```



```
        self.x_pos += self.SPEED

pygame.init()

new_game = Game('background.png', SCREEN_TITLE, SCREEN_WIDTH, SCREEN_HEIGHT)
new_game.run_game_loop(1)

# Quit pygame and the program
pygame.quit()
quit()


# Load the player image from the file directory
# player_image = pygame.image.load('player.png')
# Scale the image up
# player_image = pygame.transform.scale(player_image, (50, 50))

# Draw a rectangle on top of the game screen canvas (x, y, width, height)
            # pygame.draw.rect(game_screen, BLACK_COLOR, [350, 350, 100, 100])
            # Draw a circle on top of the game screen (x, y, radius)
            # pygame.draw.circle(game_screen, BLACK_COLOR, (400, 300), 50)

            # Draw the player image on top of the screen at (x, y) position
            # game_screen.blit(player_image, (375, 375))
```