Intro to Python

Intro to Python

Learn the basics of python in this interactive walkthrough room.

[Task 1] Introduction to Python

Welcome to the Introduction to Python room.

With this room, we aim to provide users with a basic understanding of the programming language Python, how it works and provide some ideas as to how you can use this to aid you.

Please note, that some tasks have questions to answer before completing in the section and you will have to type `truee` or `false`. This isn't a spelling mistake more a deterrent to just looking at the answer field and putting in the answer with that amount of characters!

Throughout this room, we will touch on subjects such as:
- **Variables**
- **Loops**
- **Functions**
- **Data Structures**
- **Libraries (PIP)**
- **Files**

Before you begin you will need:
- **Vim or a text editor of your choice. (Note you can use an IDE if you'd prefer)**
- **Python 3.7 Installed**

Just a foreword before continuing into the content within this room. You will be required to think about how certain things work within this room. It would be impossible to teach every tiny detail of a programming language so there will be some tasks that won't be taught that require you to think and test stuff out.
What's the worst that can happen? You get an error, well good news errors are easy to fix so let's dive into the content and crack on with the room.

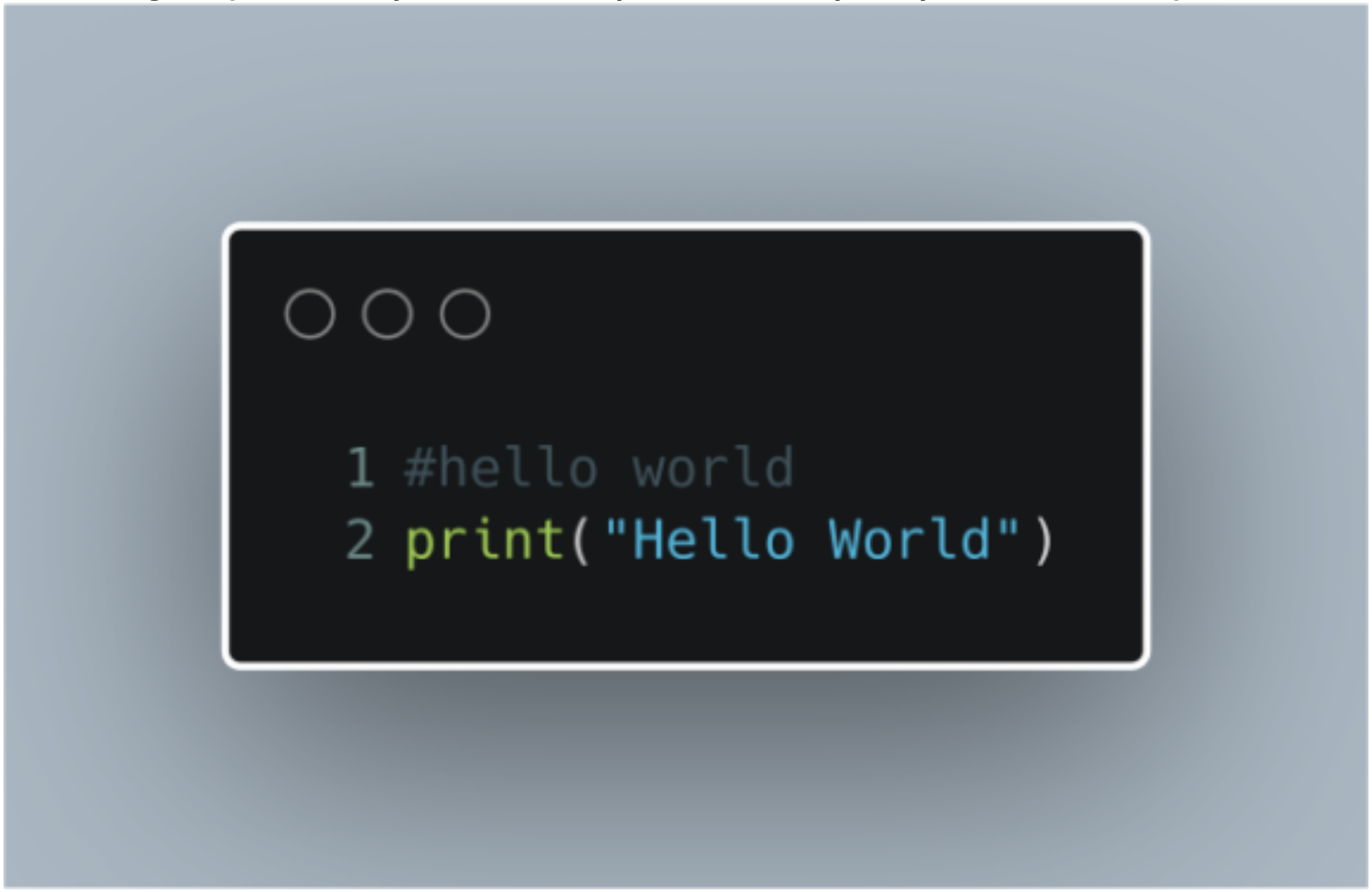
#1
Section Complete

No answer needed

[Task 2] Hello World

To begin we will create a simple hello world program as it's a staple for everyone when they are learning a new language.

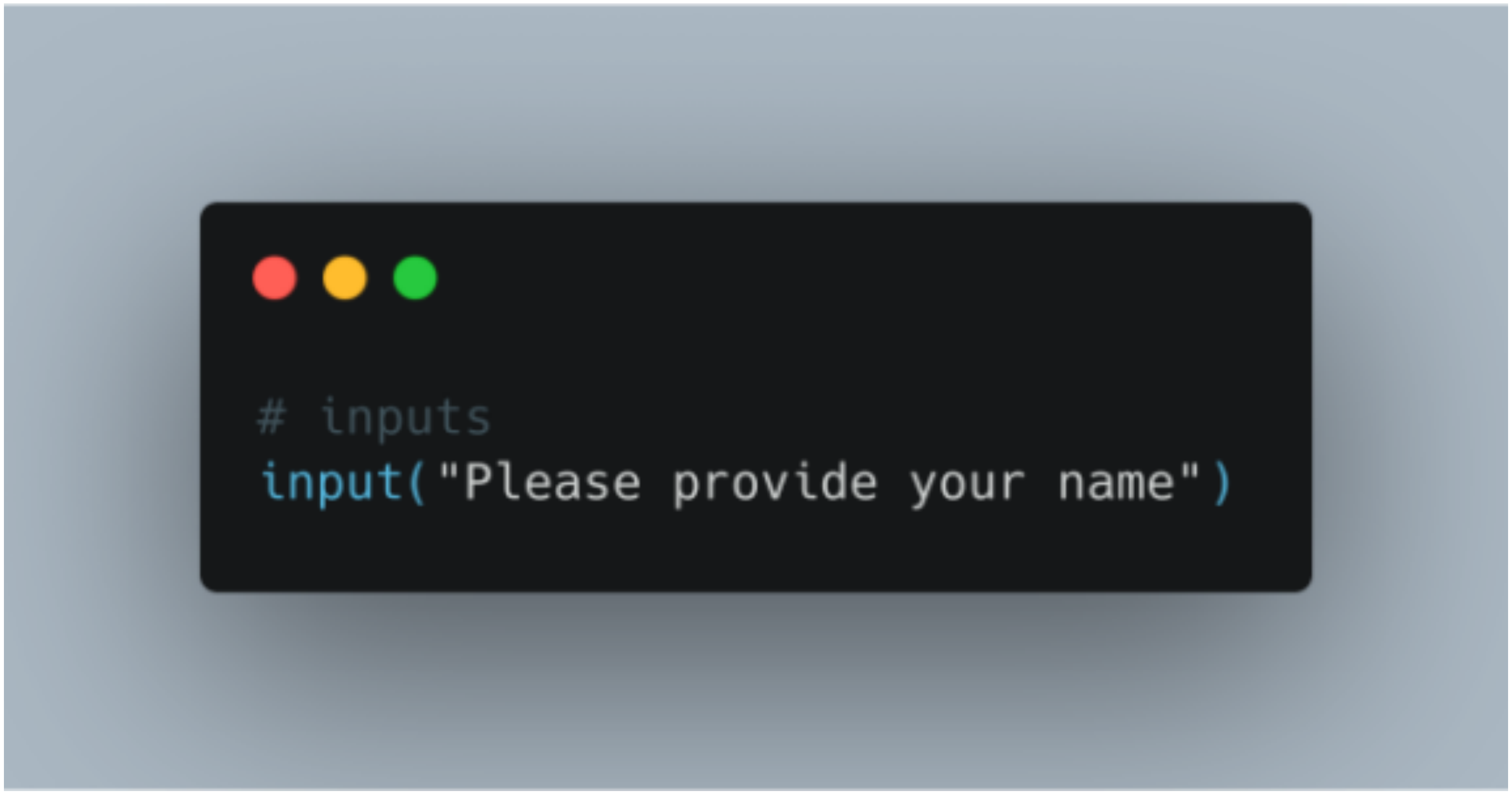
Now, this might surprise some of you but it is actually a lot easier than you may think. See the example below.



```python
#hello world
print("Hello World")
```

As you can see it's just one line and when we run, it will output `Hello World`. Now let's break this down.
The way we can control Python into printing something into our terminal is by using `print()` anything inside of the parenthesis () will be printed into the users terminal. However, because we are printing a string (More on these in task 3) we have to put them inside of quotations ""

Another key piece of syntax we will be using is `input()` This is the primary way we will be taking user input. See below for examples of how we would go about using input.



```python
# inputs
input("Please provide your name")
```

Anything that is placed within the parenthesis will be printed into the terminal, this is used as a common way to prompting the user what is expected to be entered.

#1
Section Complete

No answer needed

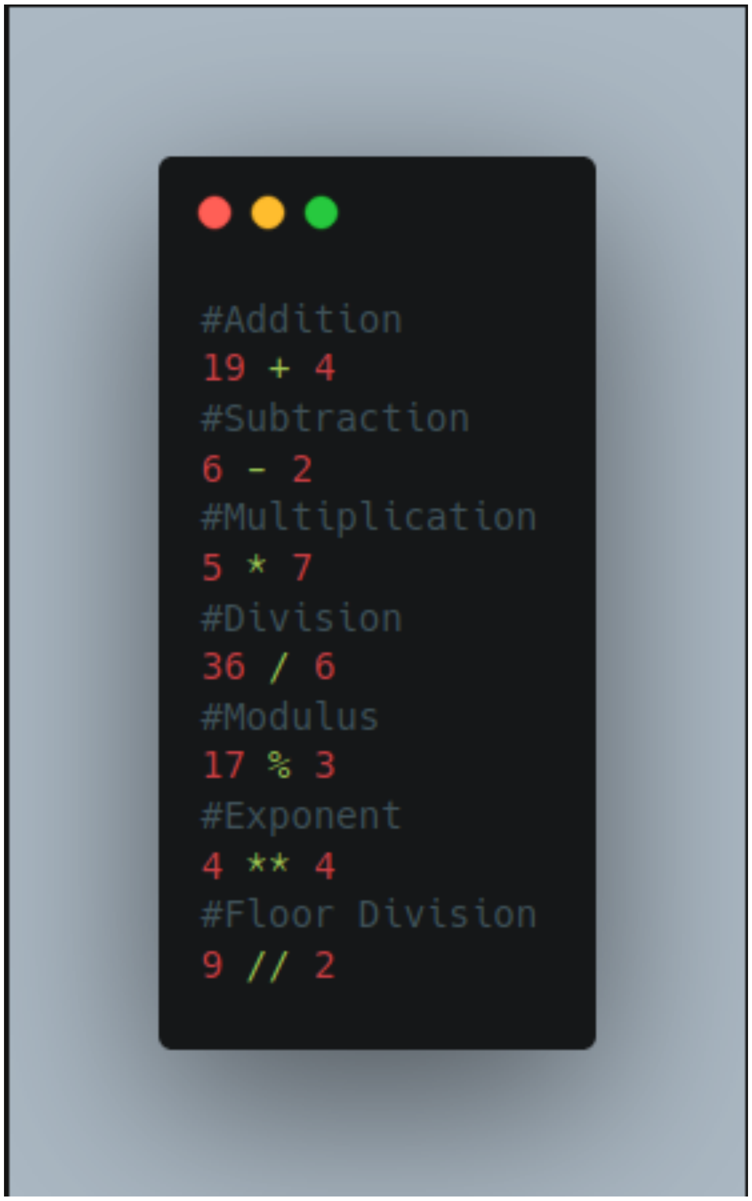
[Task 3] Mathematical Operators

In this section, we will be going over operators and how they can be applied to Python to increase the complexity of our scripts but also give us added control over what we are doing and how we check stuff.

Below in the table, we have the different types of mathematical logic that we can apply within the program.

click me	**click me**
Operator	Syntax
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus	%
Exponent	**
Floor Division	//

Now we know these we can start using them in our programs like so:



```
#Addition
19 + 4
#Subtraction
6 - 2
#Multiplication
5 * 7
#Division
36 / 6
#Modulus
17 % 3
#Exponent
4 ** 4
#Floor Division
9 // 2
```

It is key to note you can also store these in variables like so:

```
a = 10 + 5
```

Now we know basic mathematical operators let's move on to comparison operators, these play a big part in Python and will be built upon in the next topic when we look at loops and if statements.

click me	**click me**
Symbol	Syntax
Greater than	>
Less than	<
Equal to	==
Not Equal to	!=
Greater than or equal to	>=
Less than or equal	<=

We will cover these in much more detail in the next section as it is easier to show how these can be used with if statements.

#1
What is the name of >

greater than

#2
What is the name of !=

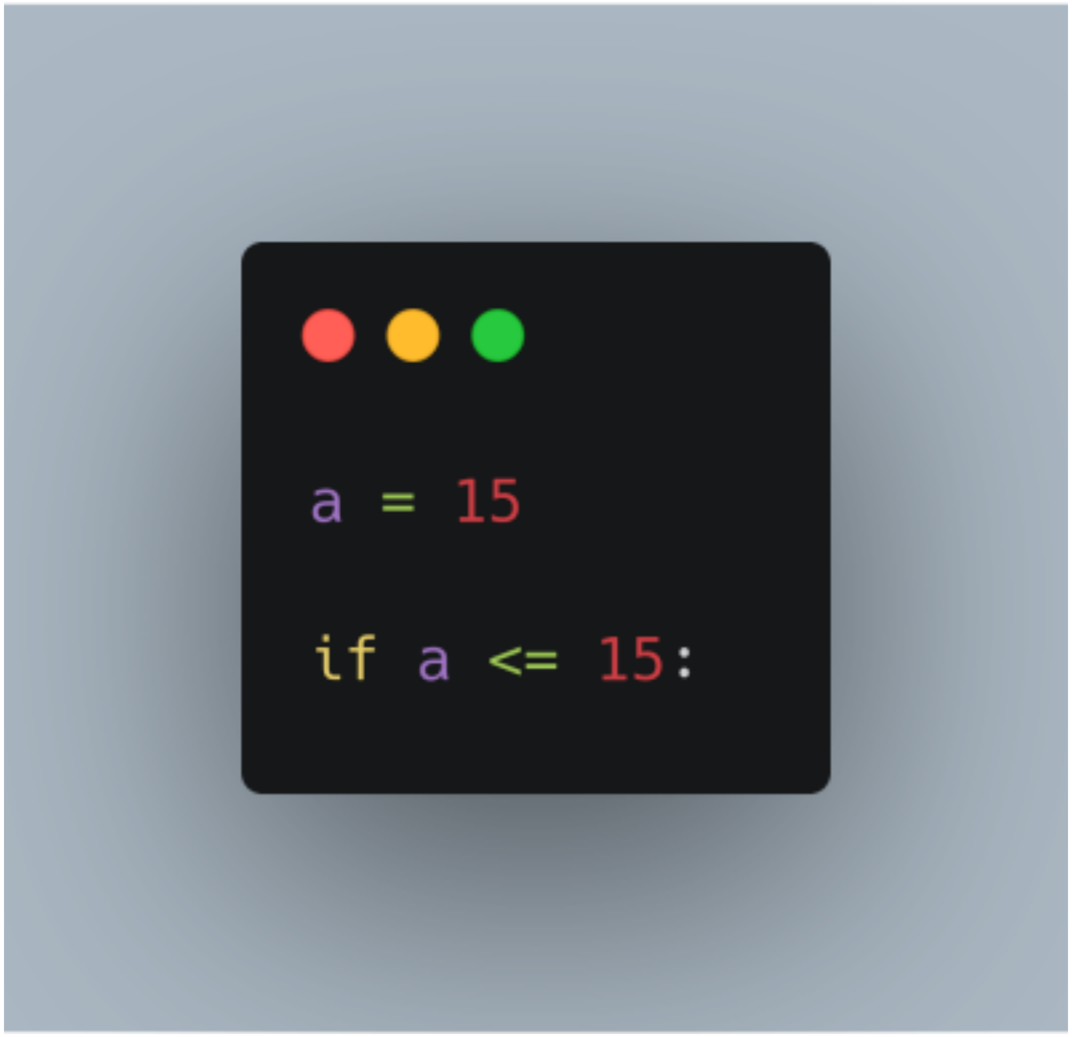
not equal to

#3
1 != 0 will this return true or false (T or F)

t

#4
What is the name of <=

less than or equal to



```
a = 15

if a <= 15:
```

#5
Will this sample code return truee or false

truee

#6
Section Complete

No answer needed

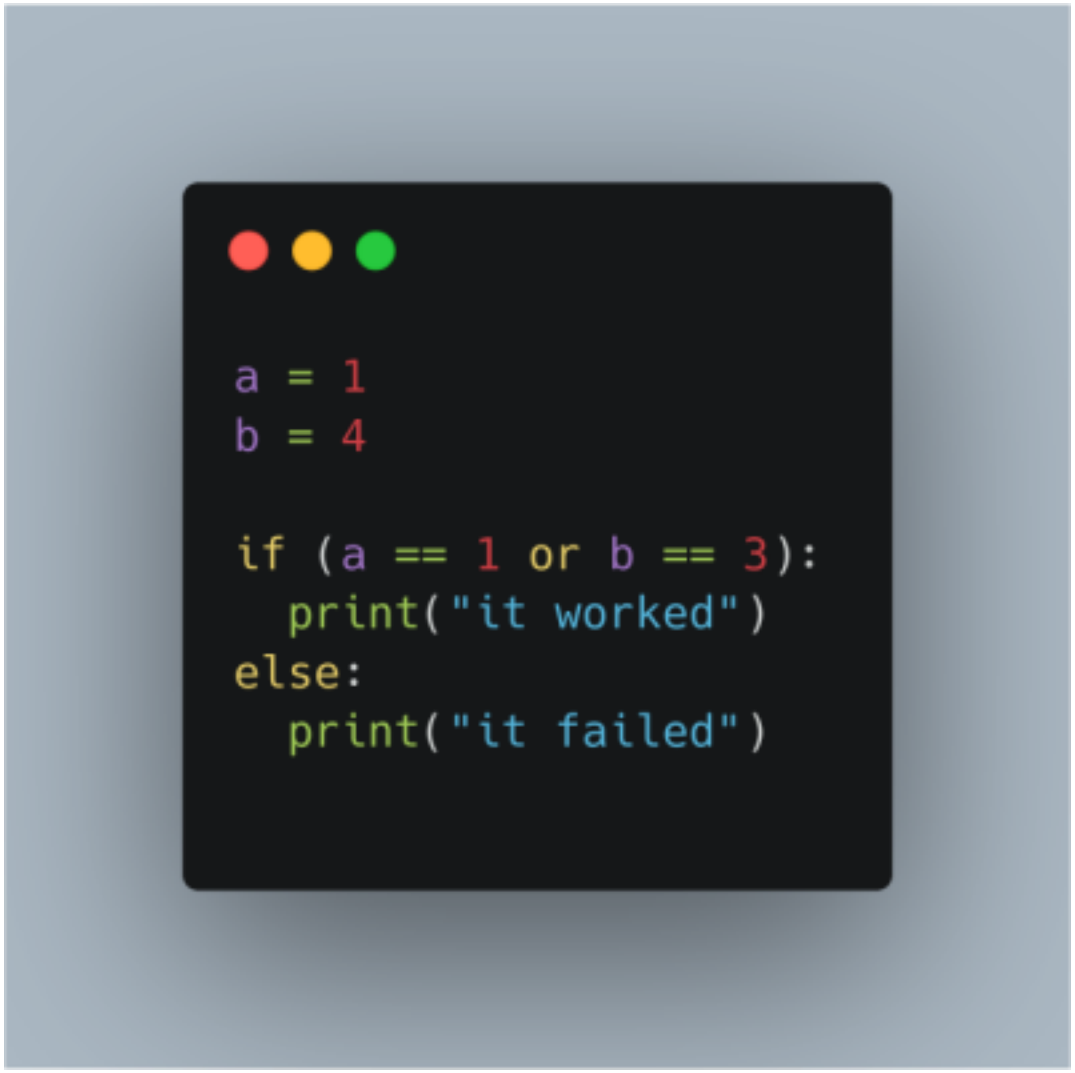
[Task 4] Logical Operators

Logical operators come in handy when we want to string sequences together for this I will show you an example of an or statement.

Please note that there are examples that include if statements which we will be covering in more detail in the next section!

Operator	Syntax
AND	and
OR	or
NOT	not
IS	is
IN	in

Let's break down this following piece of example code and explain how the OR statement works:

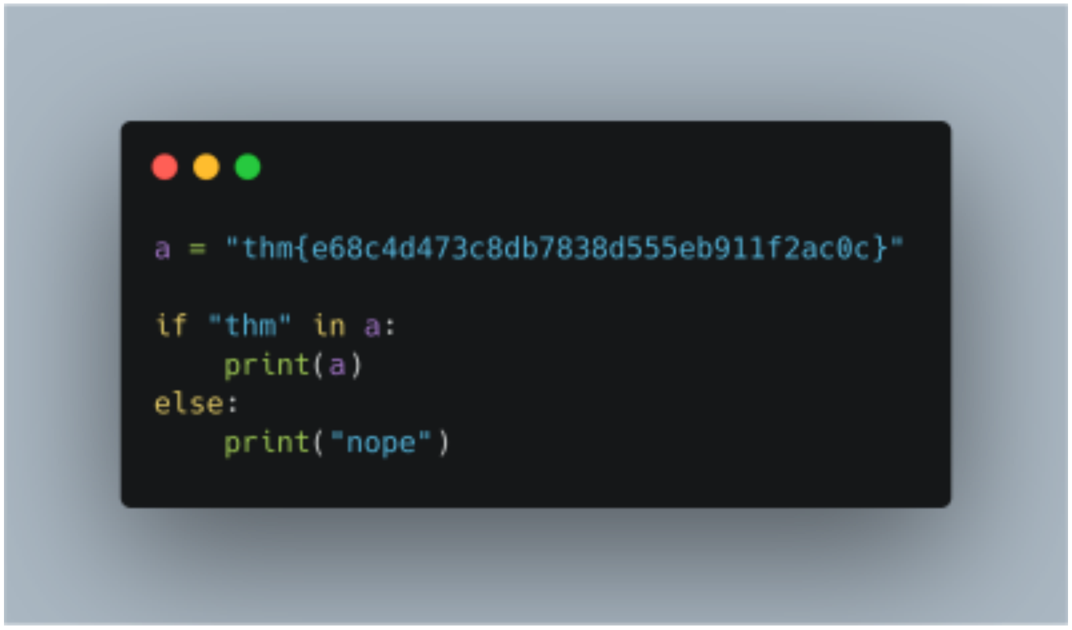


```
a = 1
b = 4

if (a == 1 or b == 3):
  print("it worked")
else:
  print("it failed")
```

So if we want to break this down. The statement will print "it worked", if a is equal to 1 or b is equal to 3. This will return True and so execute!

Note that if neither variable meets the requirements it will skip to the else statement which will print "it failed"

I won't be showing examples of all of these operators as I want you to go and explore these amazing pieces of syntax as I believe the best way to learn is through some form of trial and error!
If you're familiar with boolean logic AND, OR and NOT will come easily enough however IS and IN have some interesting features.
When we look at IN you can use this to check for something matching within a data type in this example we use a variable named a. If we run the following example it will search for "thm" within the given variable and if it finds it, will print the contents of a.



```
a = "thm{e68c4d473c8db7838d555eb911f2ac0c}"

if "thm" in a:
    print(a)
else:
    print("nope")
```

Now there are a number of applications for this however, this can be extremely useful when it comes to Capture the Flags and challenge rooms as you can create a program that once ran, will check through all files that you specify hunting for the flag prefix! It's always a pretty neat tool to have!

This pretty much covers logic, you probably noticed I didn't give examples for everything and that was intended. You can now go forth and research the missing links and develop your ability to research and gather information. If you need any help to be sure to head over to the discord and ask away, there are some really talented programmers in there!

#1
Section Complete

No answer needed

[Task 5] Variables and Data Types

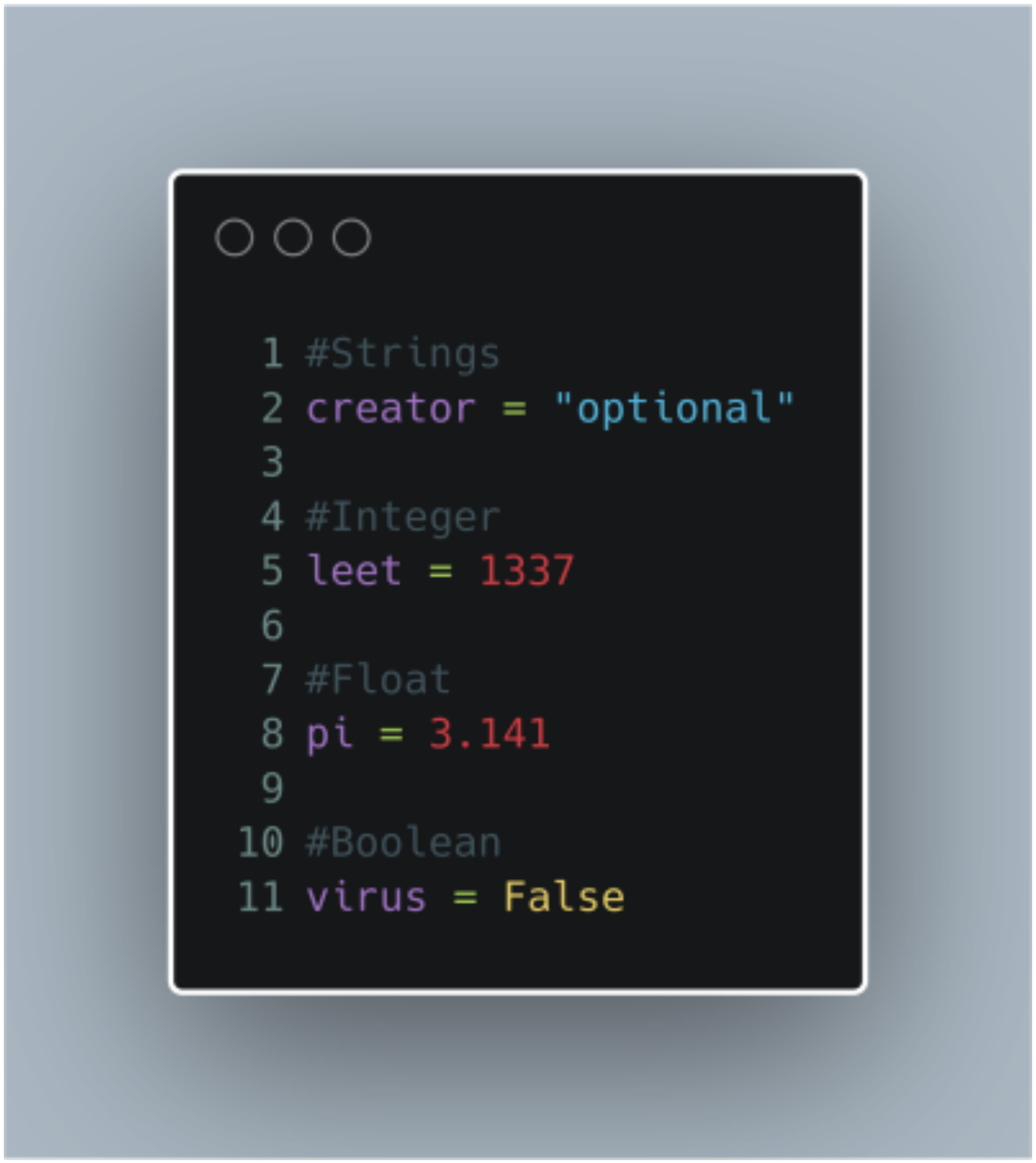
Data Types

One of the key things used within programming and Python is variables, which by definition are values that can be modified depending on conditions and information passed to the program. However, before we look into variables we need to understand data types and how each one differs.
The basic data types used in this room will be as follows:

Data Type	**Definition**	**Example**
Boolean	A value that can only be True or False.	True or False
String	Strings are used to define text instead of numbers	"Blu3 1snt b7r0k3n - DorkStar"
Integer	Solid numbers	55
Float	Decimal numbers such as 4.6	4.6
List	A series of different data types stored in a collection.	[1,2,3,4]
Dictionary	Similar to a list. However, Dictionaries have some interesting features we will discuss as you delve deeper into this section	{1:"one", 2:"two", 3:"three"}

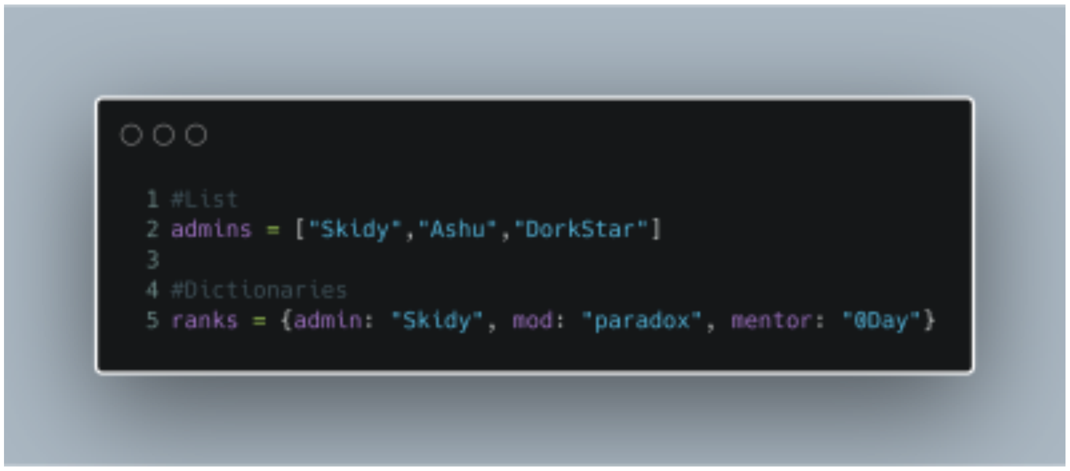
Variables
No matter what programming language you decide to learn, you will find variables are a key part of them.
To look into variables you will need to understand that there are two types of variables, those being; Global and local.
For now, we will only discuss local variables in theory however, we will discuss them further in the functions section of this room.

Global variables are any variable defined outside of a function and so can be accessed by any part of the program, compared to locals which are variables placed within a function. in turn, this also means that you can only access that variable within the function.
Please see below you'll see how we structure variables and how they look within our program and also how global and local variables will look.



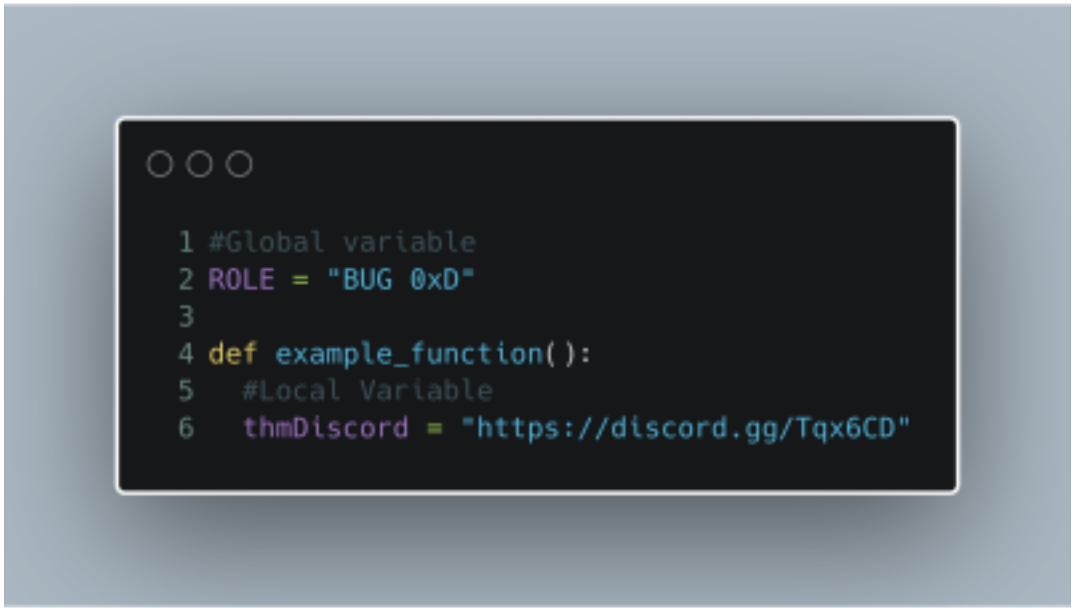
```python
#Strings
creator = "optional"

#Integer
leet = 1337

#Float
pi = 3.141

#Boolean
virus = False
```

String, integer and boolean examples



```python
#List
admins = ["Skidy","Ashu","DorkStar"]

#Dictionaries
ranks = {admin: "Skidy", mod: "paradox", mentor: "0Day"}
```

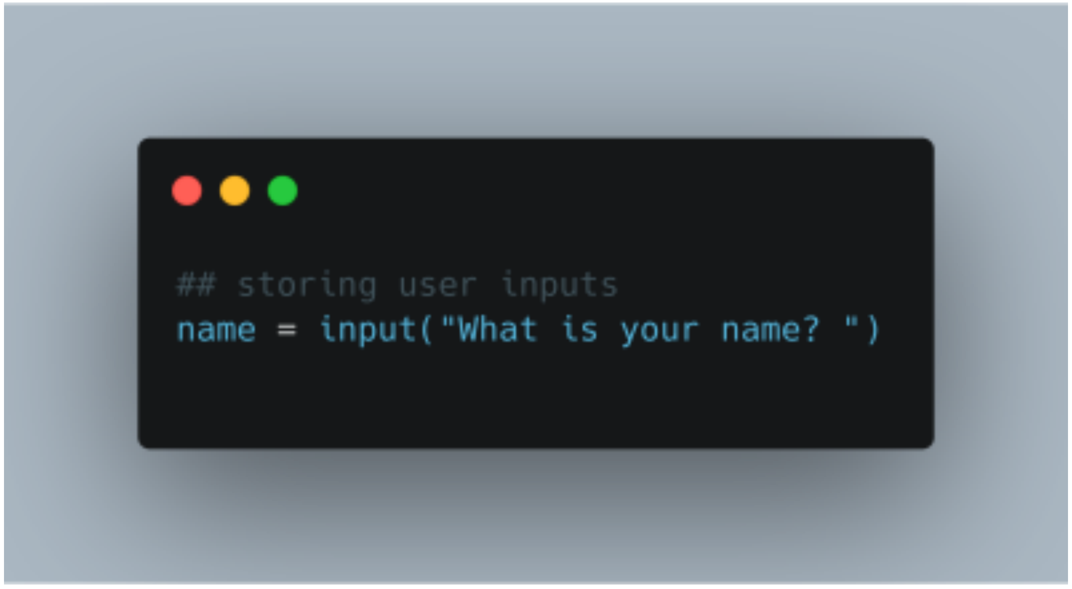
List and dictionary examples



```python
#Global variable
ROLE = "BUG 0xD"

def example_function():
  #Local Variable
  thmDiscord = "https://discord.gg/Tqx6CD"
```

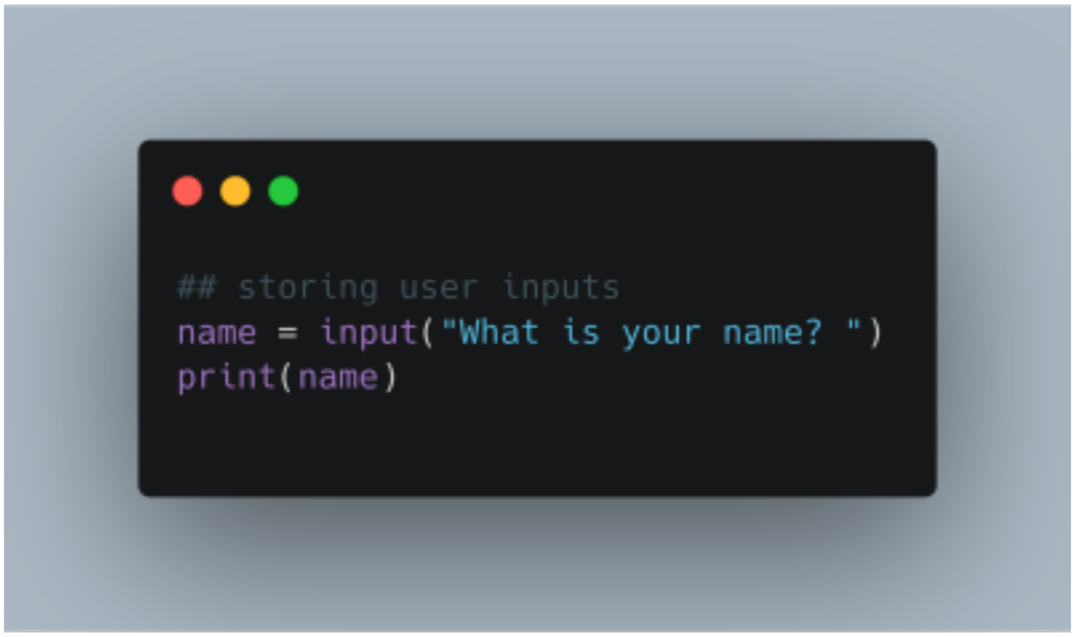
Example of global and local variables

However, as previously stated please don't look too much into local variables before the next task as that is when we will cover these in more detail and how they can be used.
Expanding on our last section we can take user input and store them in variables, this is in-fact one of the most useful features we have to take user input as seen below we can do it in one line.



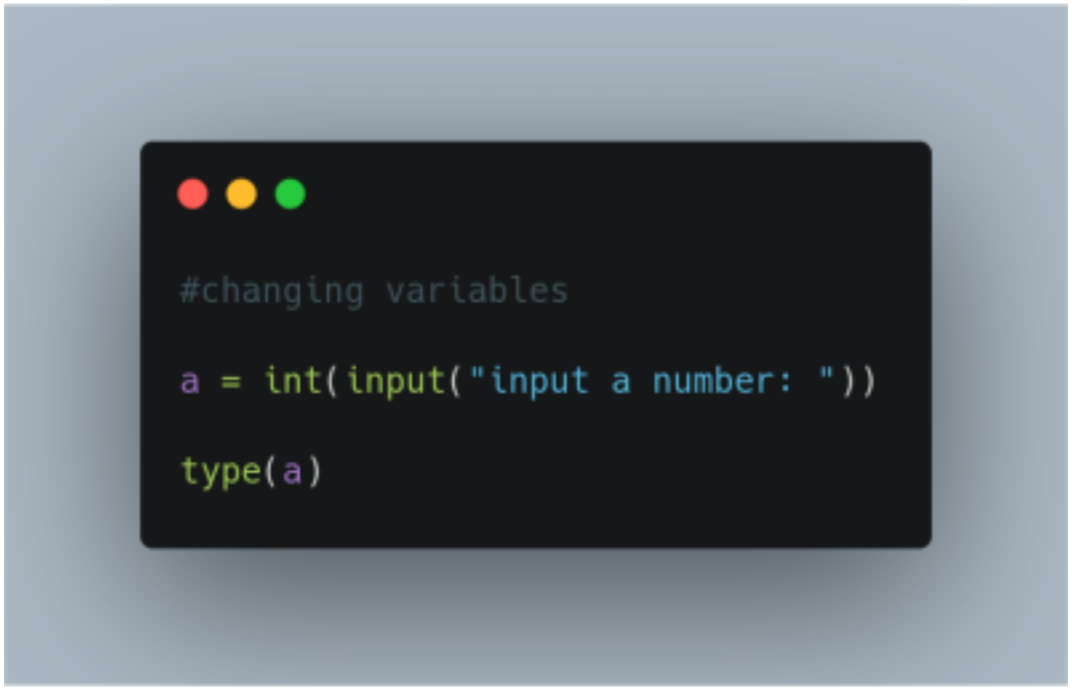
```python
## storing user inputs
name = input("What is your name? ")
```

To expand on this code, we can add `print(name)` to print the contents of a variable:



```
## storing user inputs
name = input("What is your name? ")
print(name)
```

One key thing to note with variables is that when we pass data to a variable from user input, it will by default be a string.
Note that we can change this quite simply with:



```
#changing variables

a = int(input("input a number: "))

type(a)
```

Note that type(a) is only going to return what data type this variable is. In this case, it will return as an integer!
We can change variables from one to another with ease with these following commands:
`int()` - Converts a given parameter to an integer data type
`str()` - Converts a given parameter to a string data type
`float()` - Converts a given parameter to a float data type
`bool()` - - Converts a given parameter to a boolean data type *(True or False)*

Whatever is passed inside the parathesis will be converted into that corresponding data type.
It is key to note that if you try to convert a string into an int or float, it will return an error!

#1
What data type is 13

integer

#2
What data type is "65"

string

#3
What data type is 62.193

float

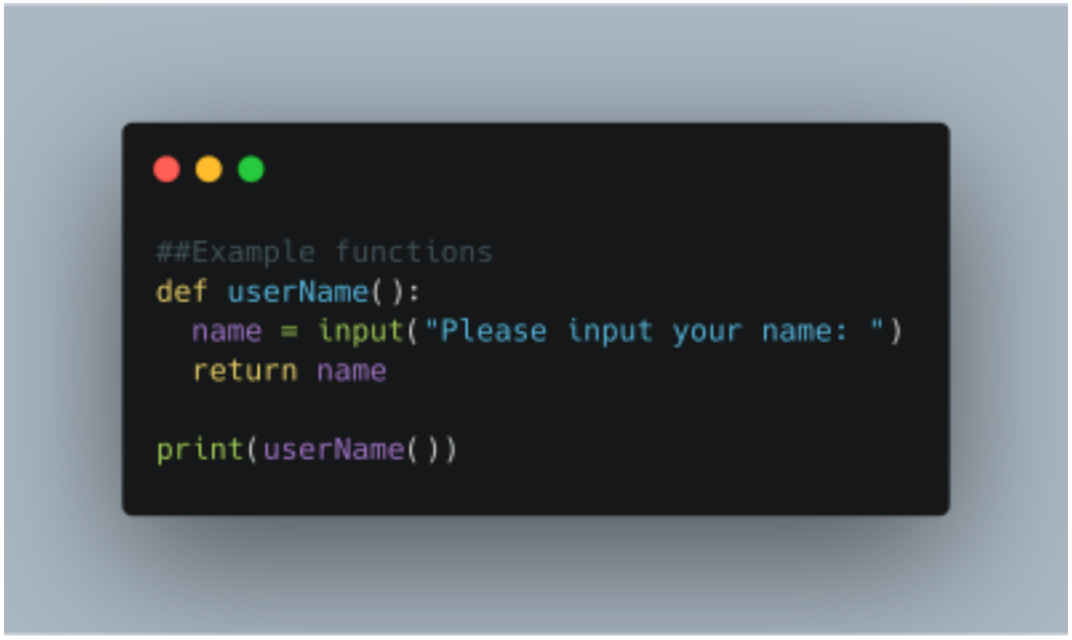
#4
Section Complete

No answer needed

[Task 6] Functions

Functions play a massive part when it comes to Python and many other languages, Python has a unique way of creating them as instead of them being called functions they are known as definitions, we would create one as follows:

`def exampleFunction():`
Here's a very basic function we have created that returns the name.

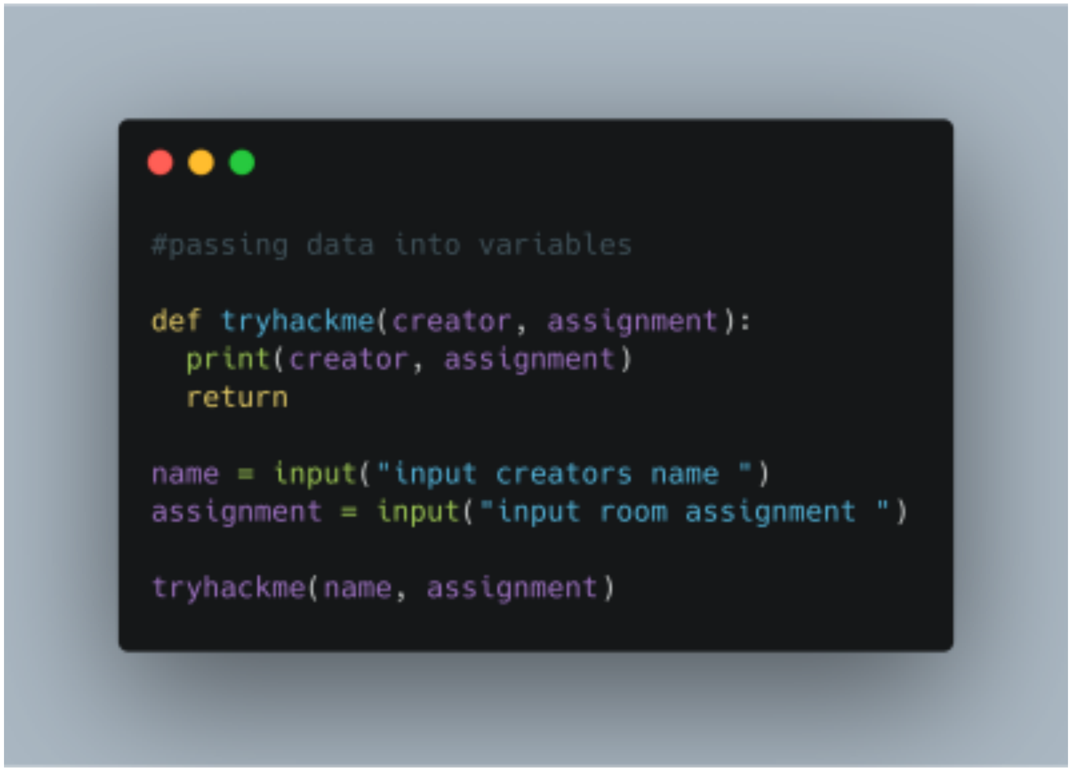


```
##Example functions
def userName():
  name = input("Please input your name: ")
  return name

print(userName())
```

As we can see we have set up our function so that when we execute it we are going to ask the user for a name. This will then be stored within the name variable.
To close off the name is given with a return, this is the standard syntax to tell the program that you want to exit the function.

It's also a good thing to mention now that the name variable within the userName function is known as a local variable as previously mentioned in the last task.
Reinforcing what we mentioned these variables are only usable within these functions as well.

<u>Parameters</u>
We can pass functions parameters when we are creating it for example if we wanted to create a basic room assignment function we could do
`def assign(creator, topic):`
As you can see our function is expecting two pieces of data when it is called, those being creator and topic.
As you can see in the image below, the data passed in using variables don't have to match when it's being called. These names are merely there for accessing the data within the function:



```
#passing data into variables

def tryhackme(creator, assignment):
  print(creator, assignment)
  return

name = input("input creators name ")
assignment = input("input room assignment ")

tryhackme(name, assignment)
```

In the image, we are passing name and assignment, even though the parameters are named creator and assignment.

This just about covers the basics of functions within Python. There are other uses for them and also ways that they can be used, though that can be extra reading for you.

#1
Section Complete

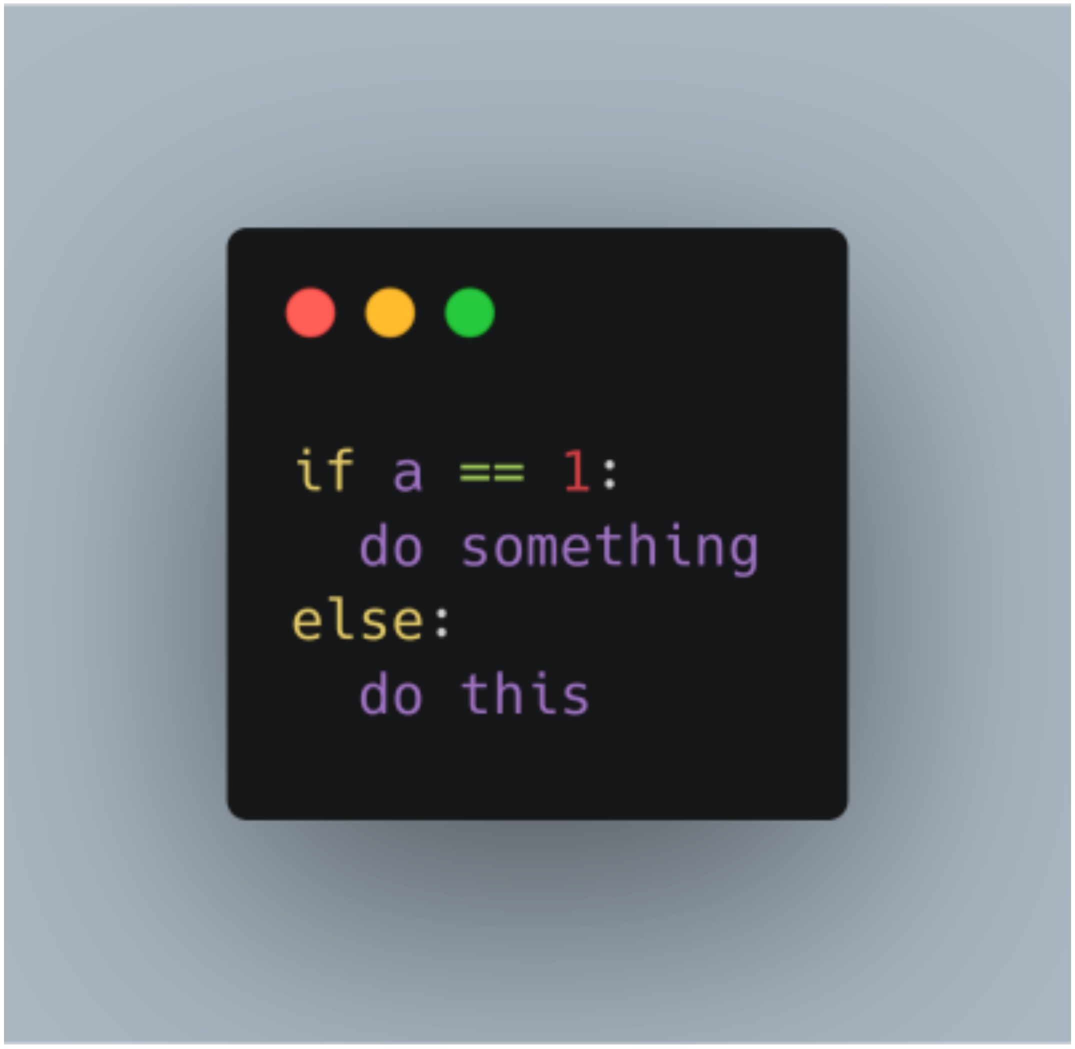
No answer needed

[Task 7] If Statements

So now we have covered quite a bit in the room. let's cover something that will take everything you've learnt and escalate it onto another level!
One of the most useful tools in any programming language is the ability to loop and having the ability to decide which part of code to execute at any given time. As you may have seen in past sections we have used examples of if statements, however, now we shall go over them in a bit more detail and break them down!

<u>If Statements</u>

This is possibly one of the most useful tools in our arsenal when we combine it with logic from the previous task.
To begin, if statements are structured like so:

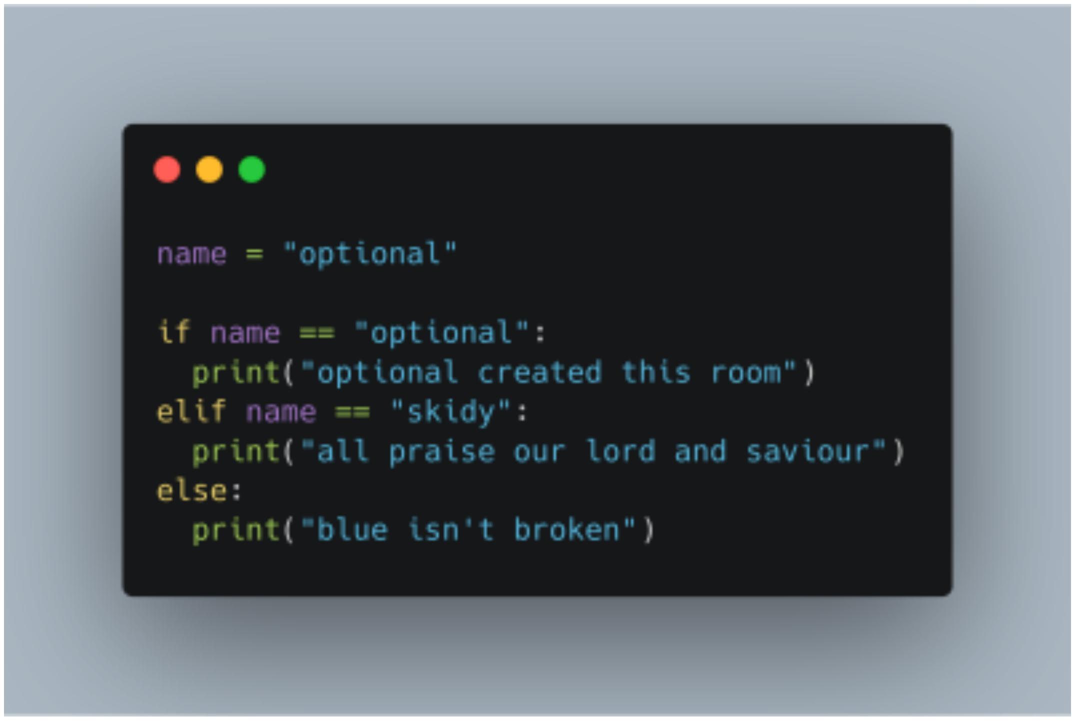


```
if a == 1:
  do something
else:
  do this
```

this is a basic if statement, as you can see we start with if then we have our criteria that if met and the output would be True will perform whatever we have in that code block, else we shall do what is under the else: statement

In this example, if a is equal to 1 it will "do something".
If a is not equal to 1 it will "do this"

This is a very basic intro to if statements as we saw in the last task we can do stuff that uses boolean logic to make these more complex, we can also make them more complex by adding more clauses into it. This can be done with an elif statement.
Let's take a look at a code snippet and break it down:

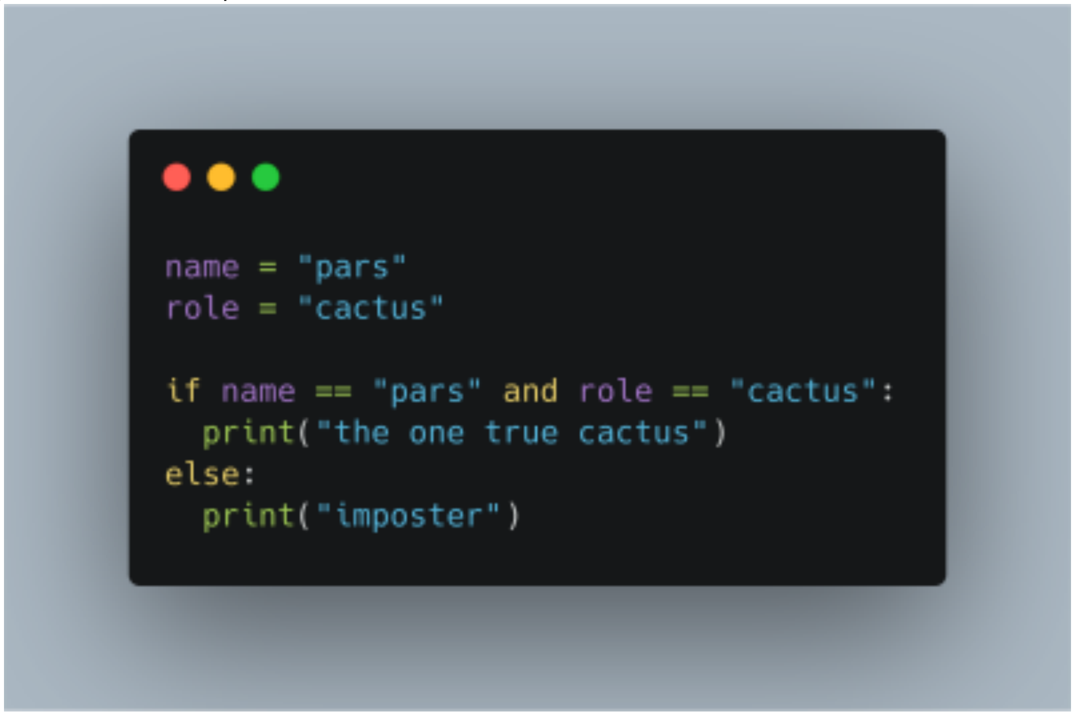


```python
name = "optional"

if name == "optional":
  print("optional created this room")
elif name == "skidy":
  print("all praise our lord and saviour")
else:
  print("blue isn't broken")
```

So to begin we have our `name` variable set to optional

if name is equal to optional it'll print out that "optional created this room".
The new piece of syntax that we introduced is `elif name == "skidy":` as you can see this adds an extra piece of complexity and allows us to determine what piece of code we will execute if we set name to skidy. It would print "all praise our lord and saviour". If the name doesn't match either, we simply get "blue isn't broken". *Which, as we all know, it is. Sorry Dark <3*

We can also implement boolean logic into our if statements with the use of `AND`, `OR` and `NOT`
Let's say for example we wanted to perform two checks before a block of code executed we would do this:



```python
name = "pars"
role = "cactus"

if name == "pars" and role == "cactus":
  print("the one true cactus")
else:
  print("imposter")
```

Here we have an example of `AND` logic. The only way we execute the print statement for "the one true cactus" is if the name is pars and role is cactus. Else we get imposter.
Hopefully, that clears up a little confusion you may have had if you did the last task and the if statement threw you out.

#1
Section Complete

No answer needed

[Task 8] Loops

Now we're moving onto loops, these are another fundamental part of this and equally as important as if statements as it allows us to iterate and perform actions a number of times.

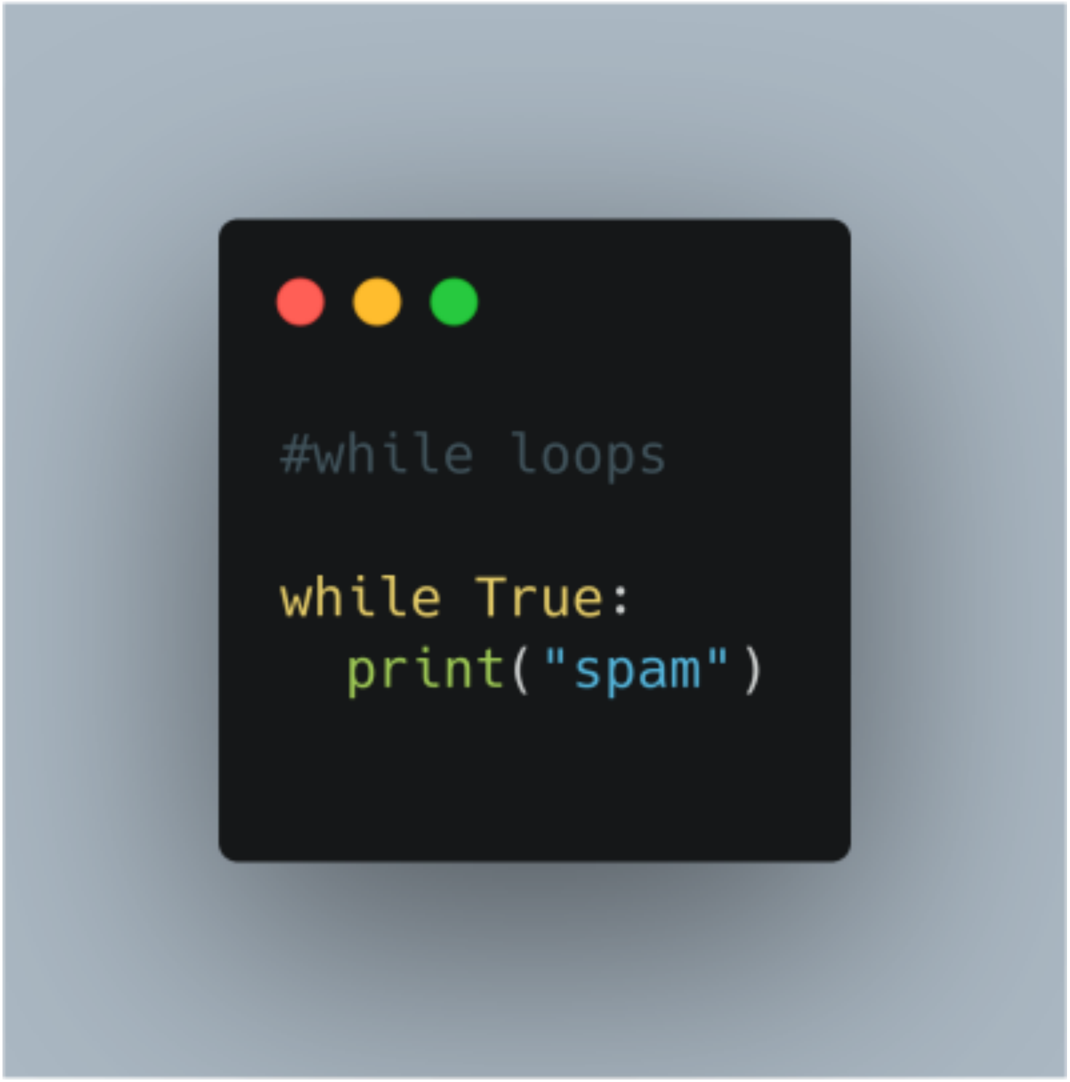
Within Python, there are two types of loops, those being `for` and `while` loops.

<u>While Loops</u>

Let's begin by looking at how we structure a while loop, to do this we need to decide whether we want the loop to run indefinitely or how many iterations we want to it to execute. To begin we will look at the basic syntax of an infinite loop.
Please note that this loop will only execute if you stop it manually! Just in case you get stuck, the shortcut is `ctrl + c`

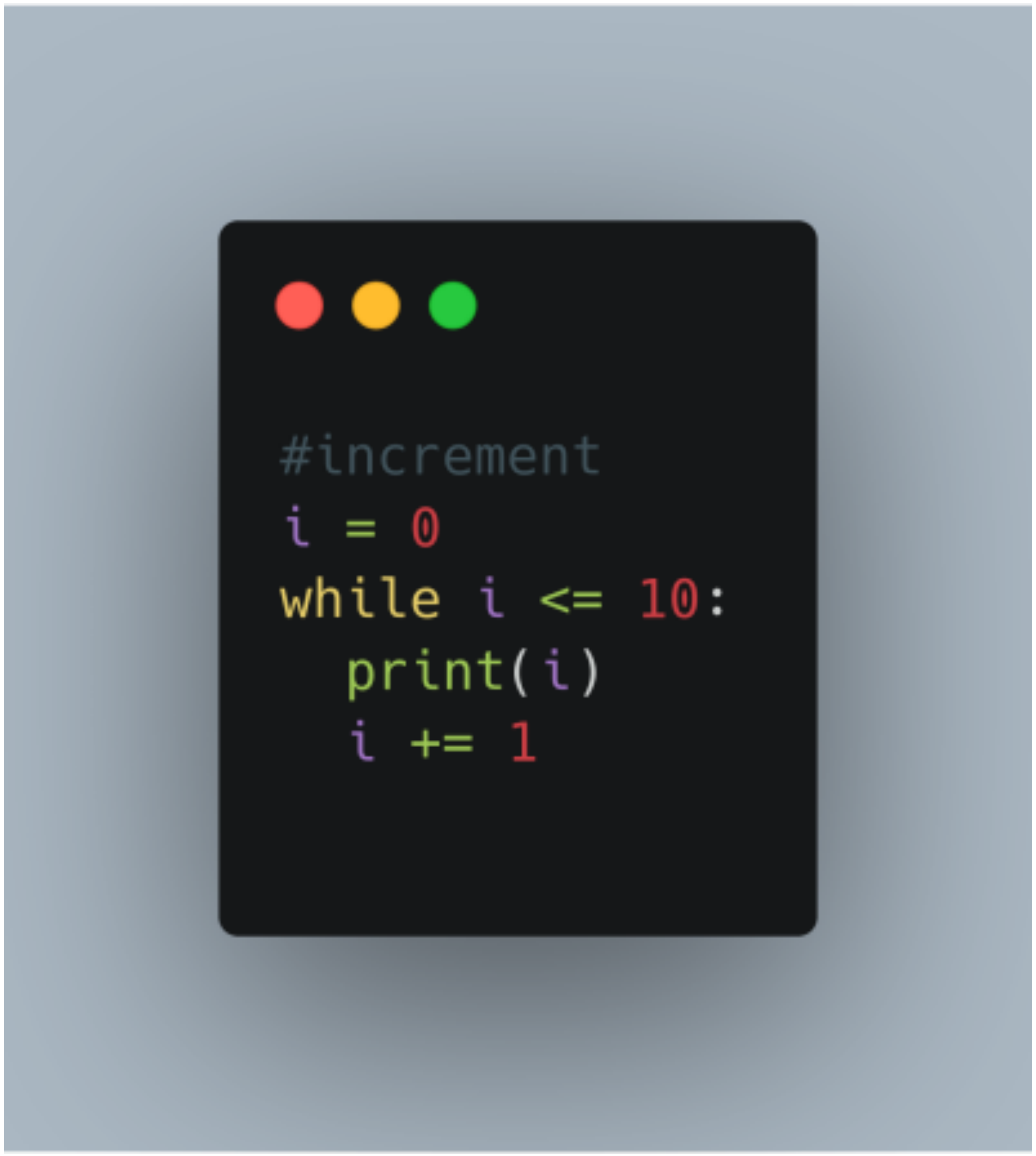
```
while True:
```

Here we have our example. To begin our `while` parameter.
Because we are providing True as our parameter, the loop will run infinitely, until terminated or unless we use `break:` to terminate the loop.



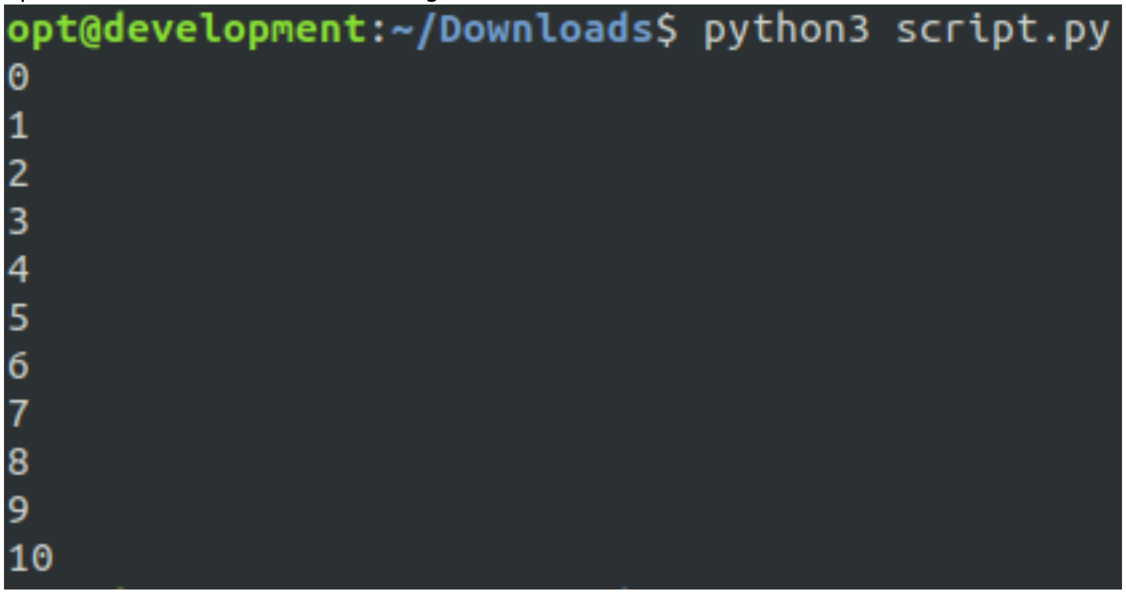
```
#while loops

while True:
  print("spam")
```

Let's take a look at how we could make it iterate for a number of times.
Just note this may look slightly more complicated as I have started to increase the complexity and added += to create our counter with less typing.
Note that doing i += 1 is the same as i = i + 1



```
#increment
i = 0
while i <= 10:
  print(i)
  i += 1
```

we have started by creating our variable i and setting it to 0. We then have:
`while i <= 10:`
So what this is saying is, if i is less than or equal to 10, it will execute what we have in our loop.
In this example, it will print out the value of i. Resulting in a list of numbers!

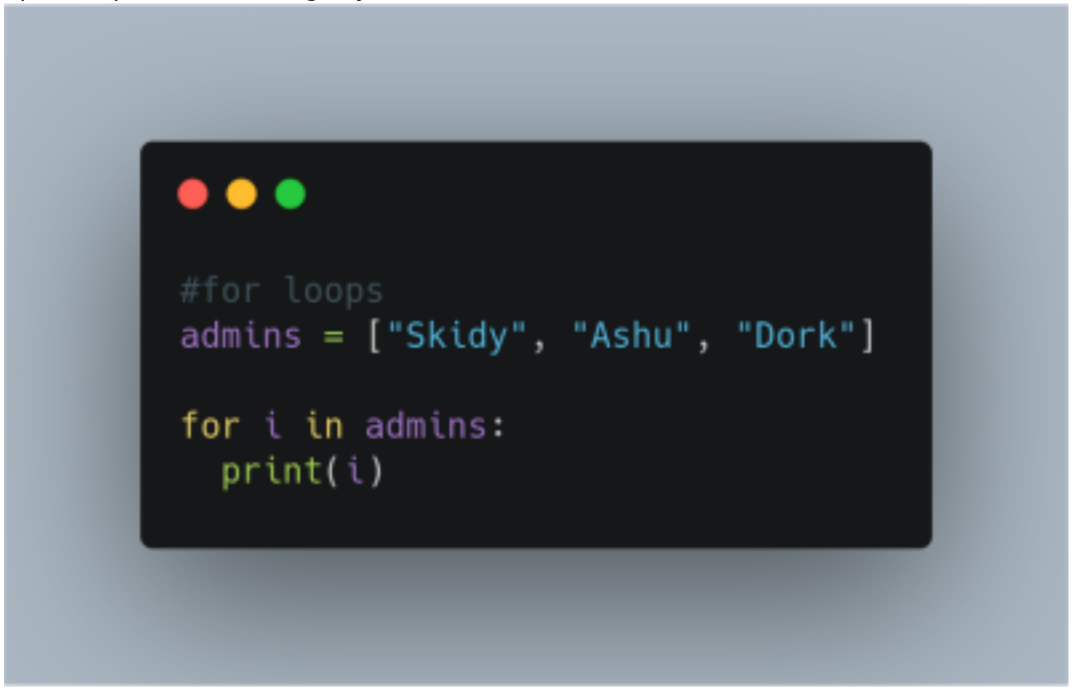


```
opt@development:~/Downloads$ python3 script.py
0
1
2
3
4
5
6
7
8
9
10
```

As you can see, because we have set our counter to 0 it started printing at 0, if we had `i += 1` before the print statement our print would end up printing 1-11 instead of 0-10. It's crucial to keep that in mind when working with counters and while loops. Note that if we wanted it to print out our counter, exactly 10 times instead of 11, we would just use a `less` than operator!
Now we've briefly covered while loops let's check out for loops.

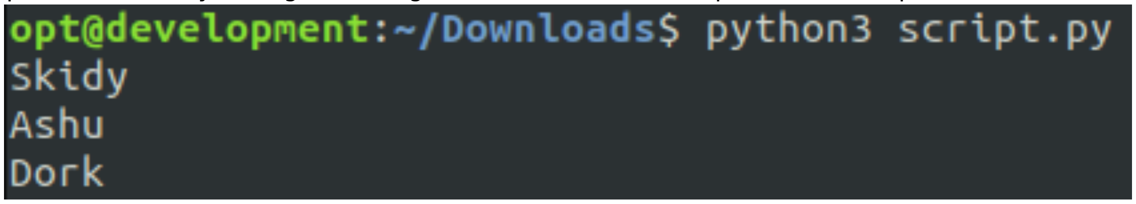
For Loops

Similar to while loops except these are slightly more structured. Let's take a look at how we use them with a basic show:



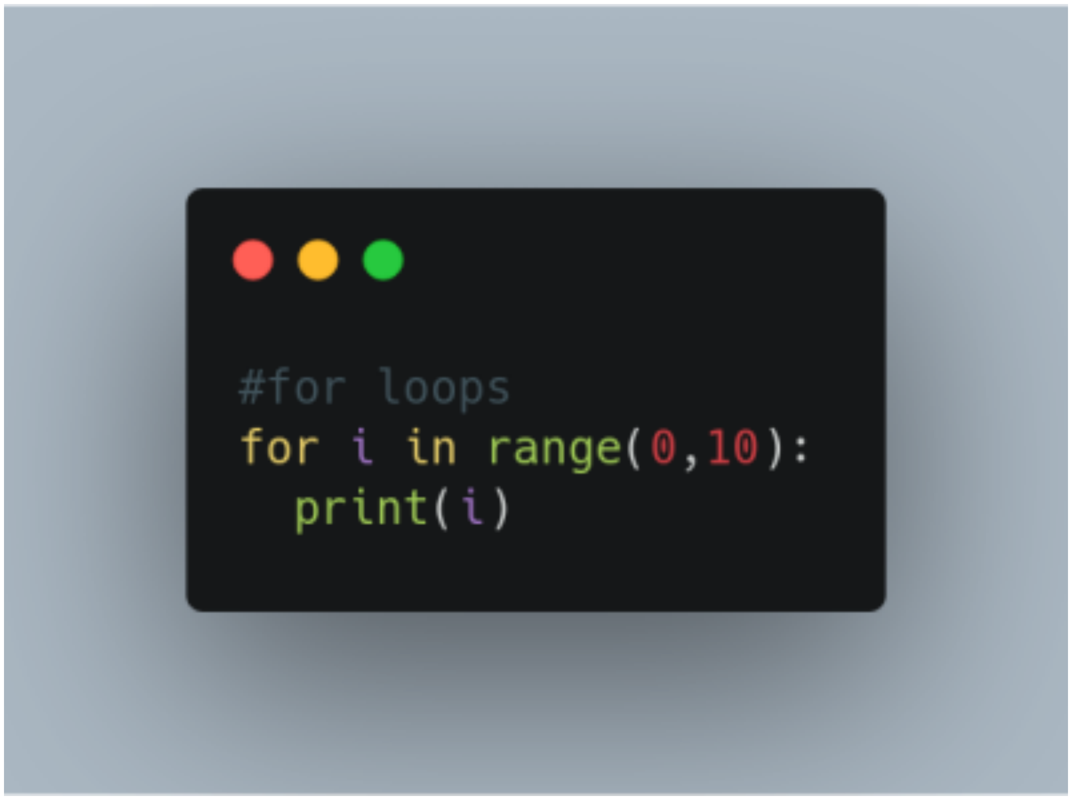
```
#for loops
admins = ["Skidy", "Ashu", "Dork"]

for i in admins:
  print(i)
```

For this, we will use a list as our example data type. We have specified three entries those being Skidy, Ashu and Dork. then we have our for loop. We structure it as `i in admins` which will iterate for every entry within that list and because of the `print(i)` it will print each entry as it goes along. See below for the output of this script.



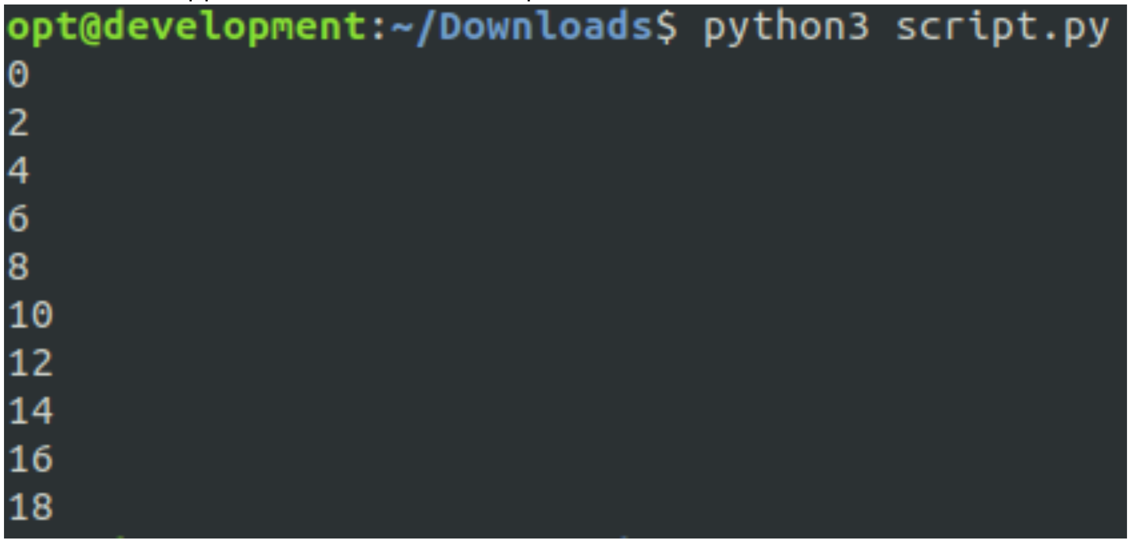
```
opt@development:~/Downloads$ python3 script.py
Skidy
Ashu
Dork
```

This is how we can go about iterating with datasets. However, a common use for this type of loop is iterating x amount of times which we can do with the following:



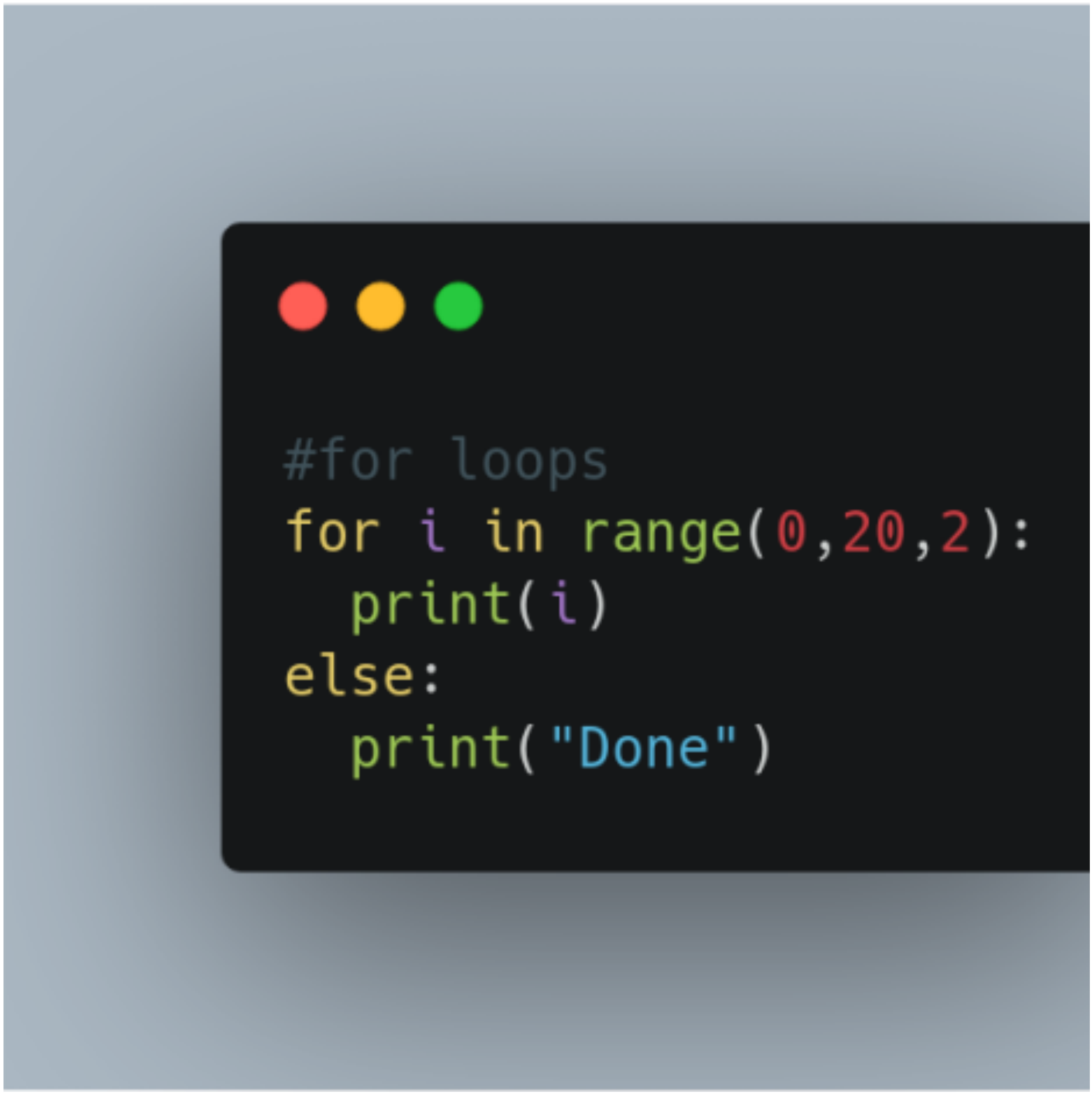
```
#for loops
for i in range(0,10):
  print(i)
```

When we use range(0,10) we are telling to program that we want to iterate it however many times it takes to reach the second field. Let's break it down:
`first parameter = start point`
`second parameter = end point`
Now, this is the key part. You can include a third parameter to tell the program the size of the increment. (By default this is 1)
`third parameter = how much to increment`
so you could do
`for i in range(0, 20, 2):`
and it will increment by two
If we run it in the previous code snippet this would be the output.



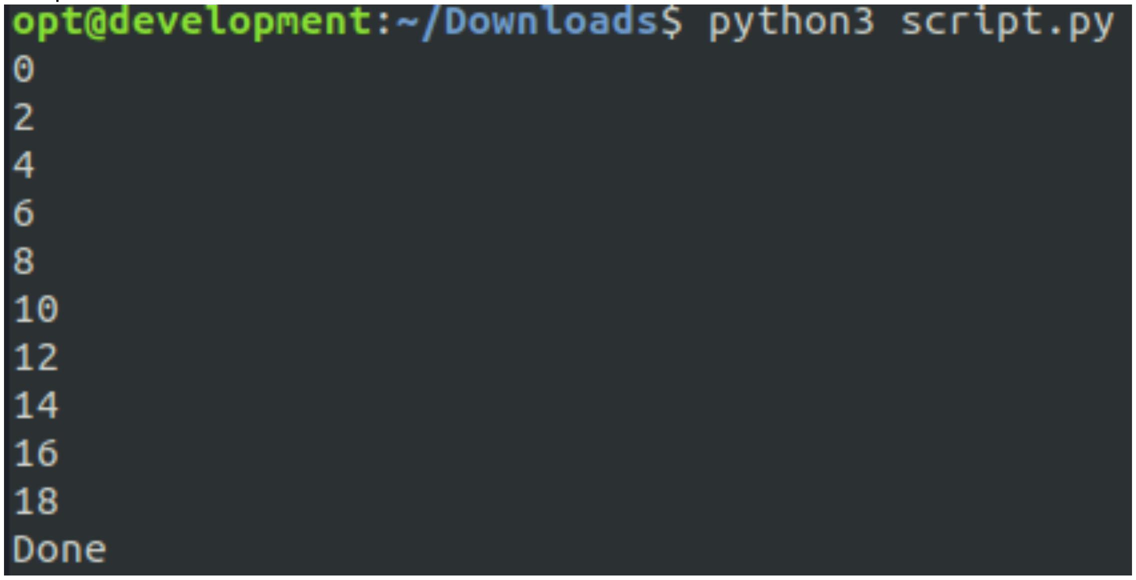
```
opt@development:~/Downloads$ python3 script.py
0
2
4
6
8
10
12
14
16
18
```

Lastly, before we finish this section, we can add else statements to this similar to an if statement. This is a basic way of implementing them into your code:



```python
#for loops
for i in range(0,20,2):
  print(i)
else:
  print("Done")
```

This code snippet would produce this result:



```
opt@development:~/Downloads$ python3 script.py
0
2
4
6
8
10
12
14
16
18
Done
```

There we go, that covers loops for now. There are more advanced methods to use these for, however, for now, you should be able to understand loops and if statements.

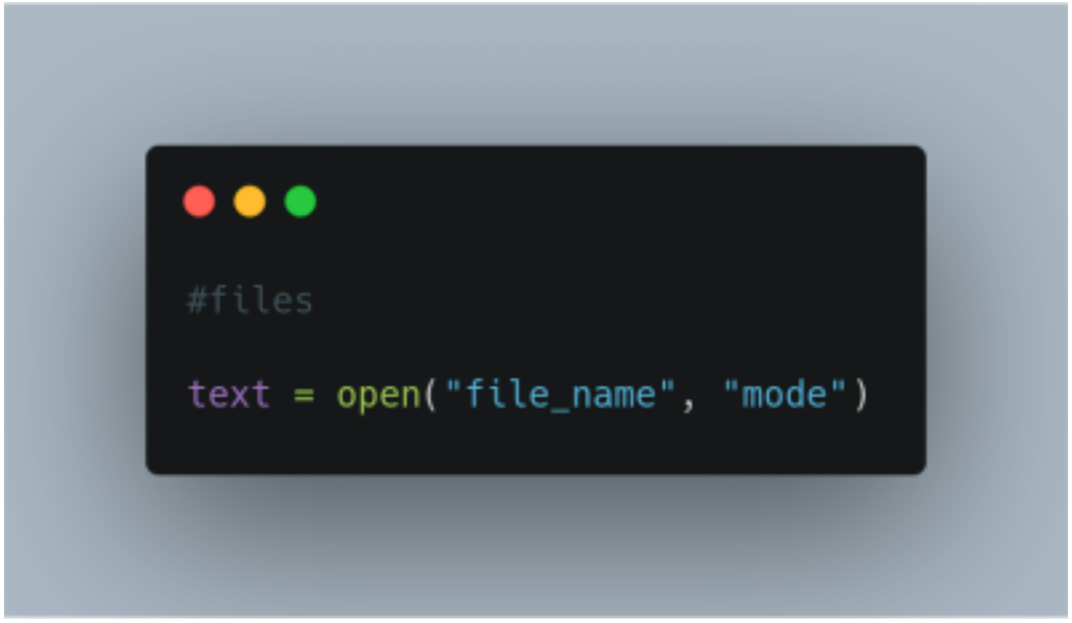
#1
Section Complete

No answer needed

[Task 9] Files

Well, you've come a long way so let's begin on learning how you can interact with files with Python. This feature is actually really useful and can come in handy with more than just text files.

Let's begin with how we want to tell the program what we want to do. Before we begin we need to create a variable to open a file. Take a look at this code snippet and we'll break it down after.



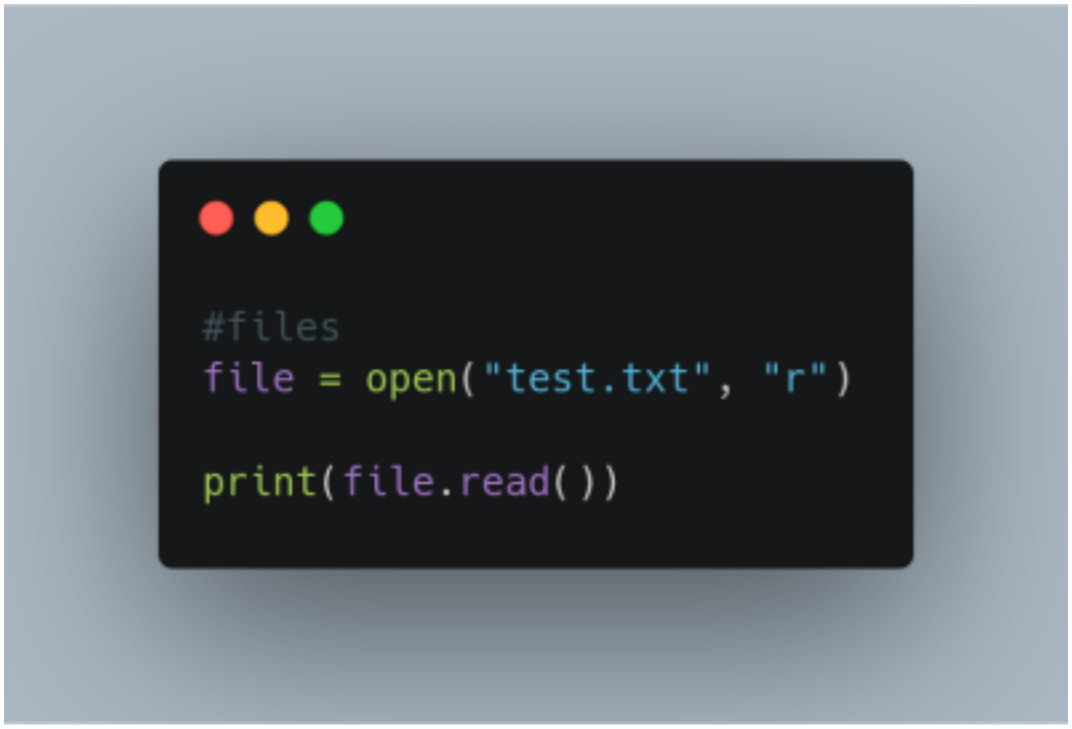
```
#files

text = open("file_name", "mode")
```

Right so as you can see it's quite simple. If the program is in the same directory as the file you wish to interact with you can simply put the name where we see `file_name`. However, if it isn't we can specify a path e.g `/home/optional/-file.txt`
The other part is the `mode`. Now, this is quite important so let's display this in a table. Because who doesn't love a good table right!

Mode	**Syntax**	**Definition**
Read	r	Read-only mode, cannot modify the file.
Write	w	This will write data to a file. Note that if the file already exists it will be erased!
Append	a	This will put data at the end of a file
Special Read + Write	r+	This mode has the ability to read and write to files

Please note there are other modes which you can find in the support information below

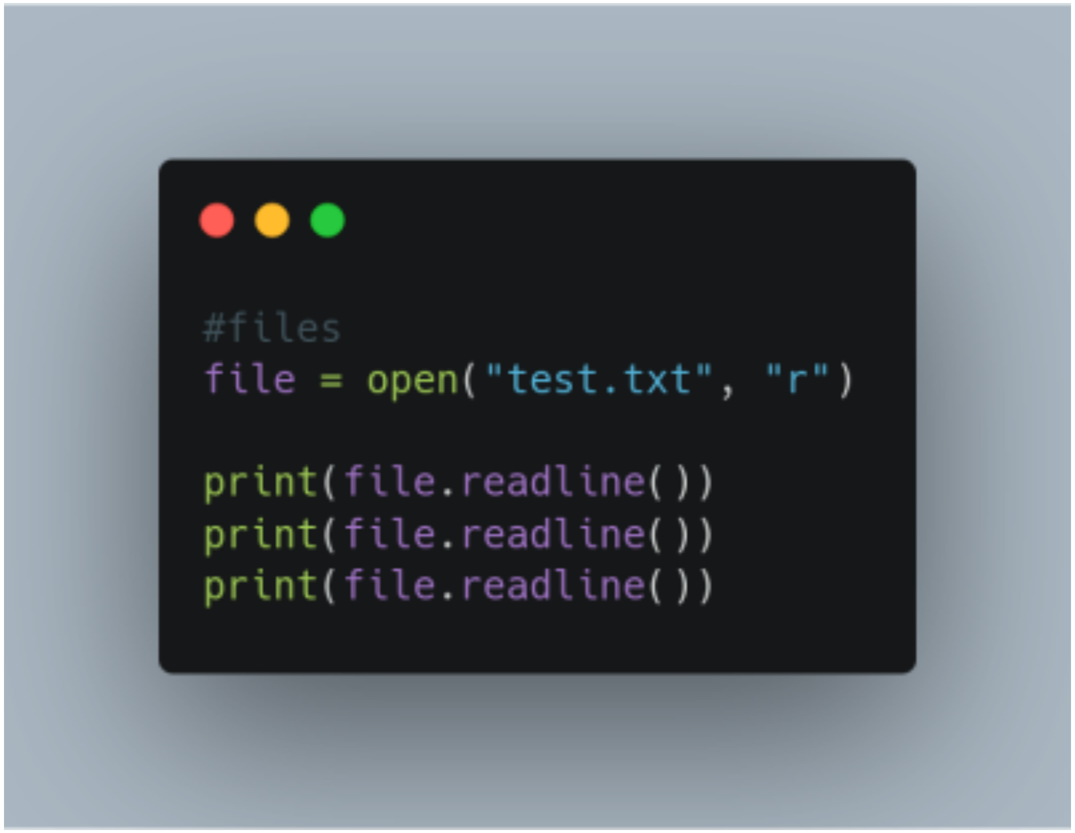
Right, so now we know the basics of how we open a file and specify which mode we want to use, let's look at some examples as to how this actually works. Let's take a look at reading a file.
This code snippet shows us that we will open a file named test.txt in read mode and then print the contents of the file.



```python
#files
file = open("test.txt", "r")

print(file.read())
```

We can also pass an integer value to the parameters within read() to print the specified number of characters.
Test.txt contains the following:
test1
test2
test3
test4

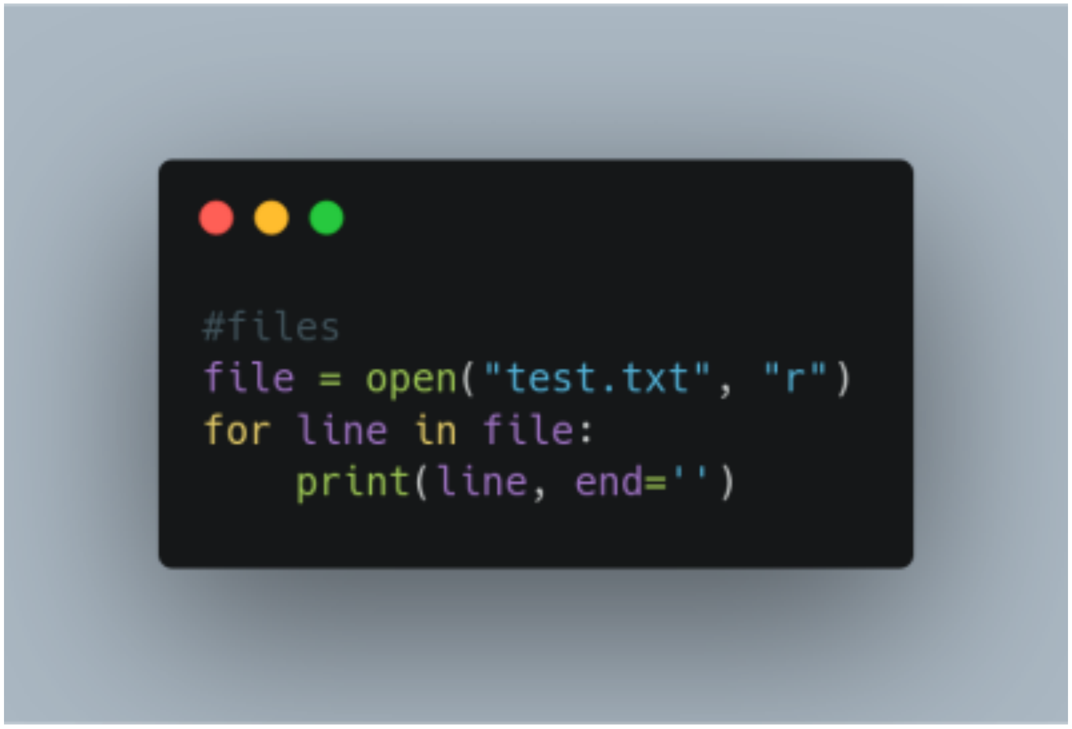
We can also tell it to read lines of the file, for this we can do
`file.readlines()` this will print out each line of the file that we have opened, or we can do `file.readline()` which will print a line every time it is called, this can be called multiple times, to print the subsequent lines. If we run the following code:



```python
#files
file = open("test.txt", "r")

print(file.readline())
print(file.readline())
print(file.readline())
```

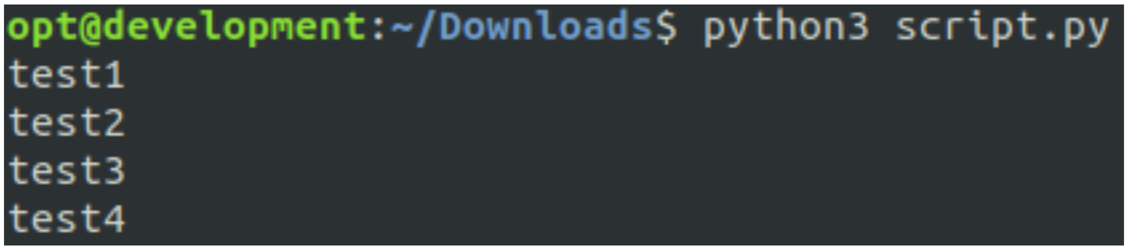
This will print out:
test1
test2
test3

As we are learning here this is fine. However, this will print the new lines too, so there will be an additional new-line. You will soon find that this is really inefficient so let's put this into a loop.
We can do this like so:



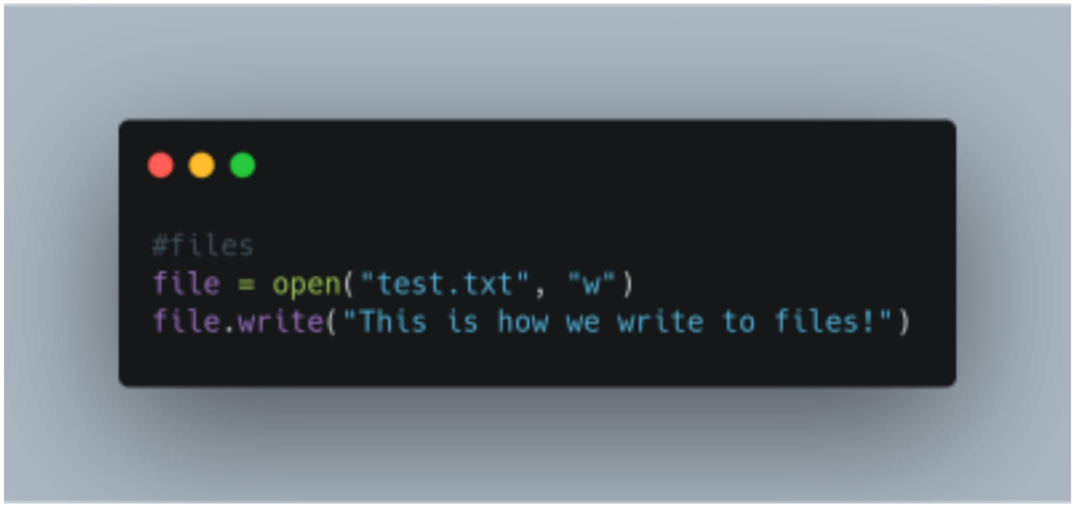
```python
#files
file = open("test.txt", "r")
for line in file:
    print(line, end='')
```

So as we learnt in the last section we provide a variable which the for loop will use. This will be called `line` each time it prints `line` and will end with `''`.
This is a nice, optimised way of printing out each line of the file and better yet, it's dynamic so will loop until every line is printed!
Here is how our output would look:



```
opt@development:~/Downloads$ python3 script.py
test1
test2
test3
test4
```

Right so let's check out how we can use the "w" mode to write files.
So we know that our test.txt contains test1 through 4. So let's write something to this file.



```
#files
file = open("test.txt", "w")
file.write("This is how we write to files!")
```

As you can see we simply replace .read() with .write() and enter what we want to output into the file.
The notable interaction here is that when we run this, the contents of test.txt will be overwritten and it will only contain our desired text!

Append doesn't really require any special notes except for when you use the .write() it will enter it at the end of the file and once you have written you should use
`file.close()`
To tell the program that you have finished modifying it.
We then have r+ which as we know from the start of the task will open in read and write. There isn't really much more to that after you know about read and write modes.

Anyway, there are byte modes which I have added some links below to some additional reading which will discuss it further.

https://docs.python.org/3/tutorial/inputoutput.html
https://www.tutorialsteacher.com/python/python-read-write-file

#1
Section Complete

No answer needed

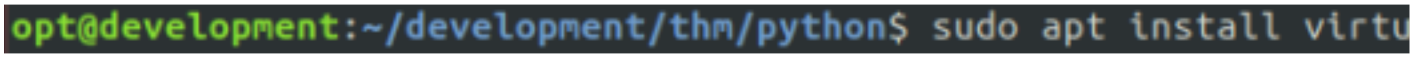
[Task 10] Virtual Environment

To begin we will introduce you to the Python virtual environment.
This is a great tool to use when developing larger projects using Python as it allows you to create an allocated space in which it will keep track of any packages you've installed using pip. (There will be more on pip and the uses it holds in the next section!)

For this room, we'll be working within a Python virtual environment as it's a good practice to get into.

To begin you may need to install virtualenv so please ensure you have installed with as shown below.
`sudo apt install virtualenv`

It's key to note that when installing this, it will also install the latest version of Python. So if you don't have it. You'll kill two birds with one stone.

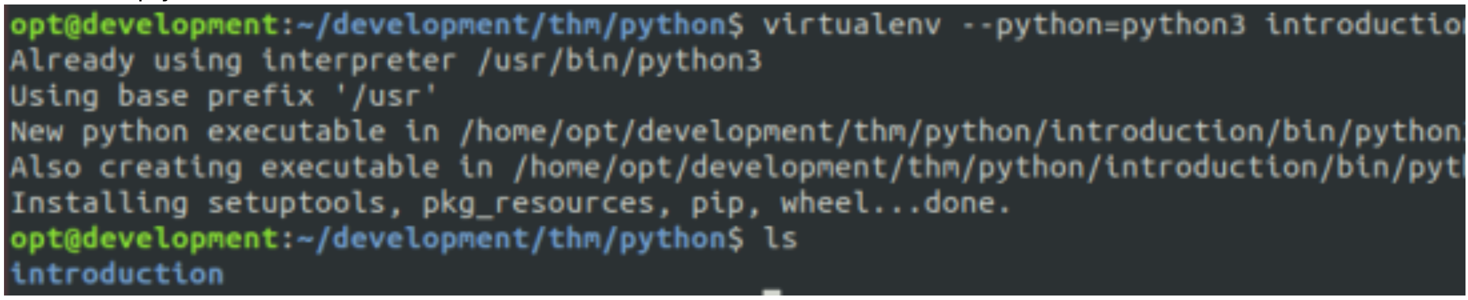


```
opt@development:~/development/thm/python$ sudo apt install virtu
```

Once the installation has finished up, we can start by creating our environment with the following command:
`virtualenv --python=python3 introduction`

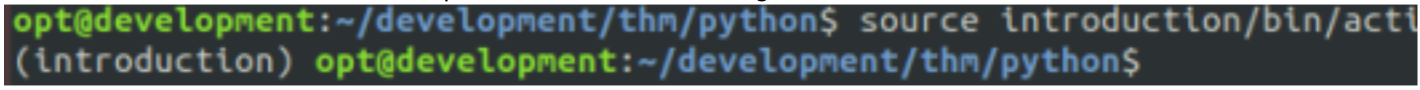
So let's break down what this command is doing:

- virtualenv - Telling the terminal what we want to run.
- --python=python3 - This is specifying which version of Python we want our environment to be in
- introduction - Simply a name for our environment



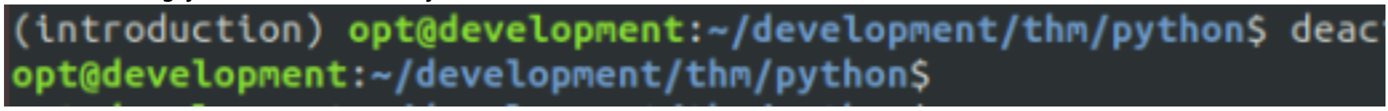
```
opt@development:~/development/thm/python$ virtualenv --python=python3 introductio
Already using interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in /home/opt/development/thm/python/introduction/bin/python
Also creating executable in /home/opt/development/thm/python/introduction/bin/pyt
Installing setuptools, pkg_resources, pip, wheel...done.
opt@development:~/development/thm/python$ ls
introduction
```

Now that we have our virtual environment set up we can activate it using source `introduction/bin/activate`



```
opt@development:~/development/thm/python$ source introduction/bin/acti
(introduction) opt@development:~/development/thm/python$
```

Now we are within our environment we can do what we like.
Once you have finished using your environment, you can use deactivate to close it.



```
(introduction) opt@development:~/development/thm/python$ deac
opt@development:~/development/thm/python$
```

deactivating our virtual env

#1
Section Complete

No answer needed

[Task 11] Libraries and PIP

This is the final section for the introduction to Python room, so let's use what we have learnt in the previous task and use virtualenv. However, before we progress forward, you will need to ensure you have pip3 installed, you can do this as follows:

```
sudo apt install python3-pip
```

Pip is what python uses to install packages, these are extremely useful and are utilised by some of the challenges on the TryHackMe platform will require you to install tools using pip! **cough cough* you know who you are!*
Let's start up our virtual environment, and begin by looking at the basic structure of a pip install:

```
pip3 install name
```

for this example, we'll use pwntools, you can install this like so:

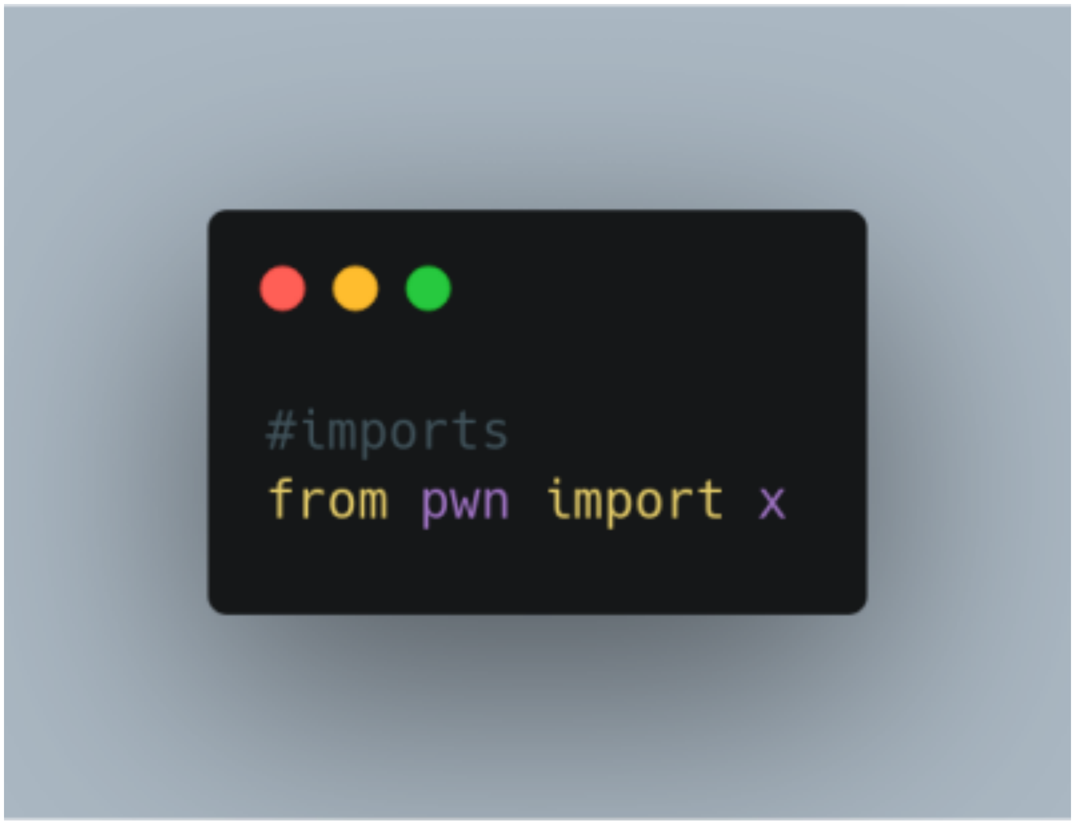
```
pip3 install pwntools
```

This will then install and we can find them within our virtual environment!
If you navigate into your bin directory within this environment, where you'll see that it has installed. This is the main reason we use virtual env, as it helps us keep track of our packages and they will only be available within the environment!

Using pip we can also install requirement files that may be supplied when working with tools from sites such as Github. We can install these with the following command:

```
pip3 install -r requirements.txt
```

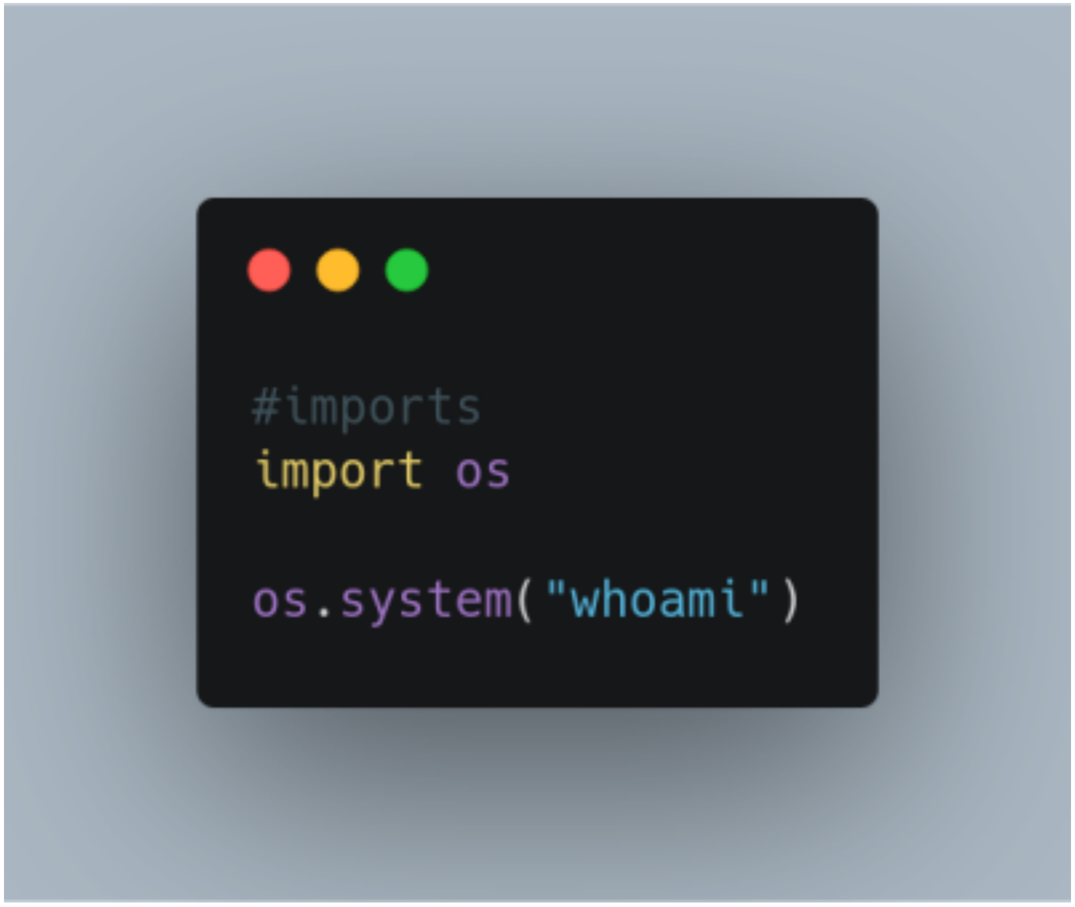
We can import these libraries using the syntax for import. As shown in the following code snippet we can specify which modules of these libraries we want to import:



```
#imports
from pwn import x
```

*Note, that we place x with what module we would like to import! You can always replace x with an * and it will import the entire package*

We can also choose to import a library without specifying any modules, this can be done as follows: `import os`



```
#imports
import os

os.system("whoami")
```

example of full import
This imports the os module and allows us to run system commands. In this example, we run whoami.

Note that because we simply imported the package, we have to specify which package it is from at the start.

If we re-wrote the import to look like this:
`from os import *`
You could simply run:
`system("whoami")`

This pretty much sums up the pip and libraries section of this room so thank you for reaching this point!
I hope you have enjoyed the room and if you have any questions or need any help, feel free to jump on the TryHackMe Discord and ask questions!

#1
Section Complete

No answer needed

[Task 12] Challenge Time!

Well, you didn't think I would leave you without a challenge?
You'll find a file attached to this task called encoded_flag.txt. Within this file, you will find some encoded information! This is your challenge as follows;
Using the base64 library within python. Can you decode this and retrieve the flag? Note this has been encoded a total of 15 times. Be sure to read from the file provided and the documentation for the base64 library. This will allow you to discover the syntax to aid with this challenge!
`from base64 import *`

5 times encoded using base 64
5 times encoded using base 32
5 times encoded using base 16!

Good luck and thank you for going through the room! I hope that you have learnt something with this room and if you have anything you believe should be added, reach out to me on the TryHackMe discord!
- **optional**

#1
Enter the decoded flag to complete the room

```python
import base64

f = open("encodedflag.txt", "r")

flag = f.read()

res = ''

for i in range(0, 5):
    res = base64.b16decode(flag)
    flag = res

for i in range(0, 5):
    res = base64.b32decode(flag)
    flag = res

for i in range(0, 5):
    res = base64.b64decode(flag)
    flag = res

print(res)
f.close()
```

THM{d431ff8fea00ad3f6b685b7182078e73}