Python Notes

By: Mildred Monsivais

September 20, 2019

Table of Contents

Python

Introduction to Python

Python is popular programming language across the world Python has a simple structure and uses indentation therefore syntax is easier to read. In addition python uses a huge amount of additional open source libraries that provide extra capability that is not built in the language. An advantage of using python is that developers can learn quickly and involves less code than languages. This language is designed to be simple however the disciplines can be elevated.

Jupyter Notebook
Anaconda : need to download the full anaconda in order to be able to install the newest version of the Jupyter notebook in order to run python 3 version (which is currently python 3.5)

Python 2 vs Python 3:
99% of external packages our on python 3. Later in the future python 2 will be discontinued in 2020. The Udemy bootcamp course is in python 3.

What are the topics covered in the course?

- Object & Data Structure
- Integers(int) = whole number, such as: 10,11,15
- Floating point(float) = 10.3,11.4,12,3
- Strings (str) = ordered sequence of characters: “hello” , “Sammy”
- Lists (list) = ordered sequence of objects[10,”hello” 200.3]
- Dictionaries (dict)= unordered key:value pairs (“mykey”: “value” ,”name” : “Frankie”)
- Tuples (tup) = ordered sequence of objects, (10, “hello”,200.3
- Sets(set) = unordered collection of unique objects “a”, “b”
- Booleans(bool) = logical value indicating True or False
- Integers(int) = whole number such as 3,300,200

Command Line
The command line also known as a terminal allows you to programmatically move through your computers directories. You run python scrips at your command line.

The section covers:

1) Find your current directory
2) Listing all files in a directory
3) How to change directory
4) How to clear the command line screen

For Mac OS
How to find your terminal?

1)put on spotlight search(Q) terminal
2)the command light interface allows you to move around different folders/files on your computer
3)with more experience you can make new directories basically terminal allows you to move around the computer

How to open your Jupyter Notebook?
Go to your terminal and open up the Jupyter notebook by typing in Jupyter notebook on your directory.
Purple text tells you what to type into the terminal.
Terminal commands for Mac:

1) pwd – tells you where your locates in the command line
2) Is – tells you where your files / folders are currently located in the terminal
3) cd- If you want to move to the folders/ files
4) cd~/Desktop – takes you to your desktop
5) clear – clears your terminal
6) cd.. – If you want to go up a directory or the line before

Quick Recap:
pwd: takes you to the current directory you are at Is:
tells you which files/ folders you are currently at
Clear: clears command / terminal screen
Cd Desktop: changes directory to a sub folder
Cd ____ which ever folder you need to go to
Cd desk -❼tab & If tab doesn't complete the rest of the word then it doesn't probably exist in a sub folder

Python setup

1)Command line basics
2)Installing python
3)Running python code
4)Getting the Jupyter notebook & the course materials
5)Git & Githut Overview

Running Python Code:
-There's 3 main types of environment
1) text editors: sublime text
2)full editors: IDE's
3) notebook environment: Jupyter Notebook

1)Text Editor

-general editors for any text file
-any variety if file types where you can work with
-can be customized with plugin & add-ons (that's why you have to get sublime text and Anaconda)

2)Full IDE's: (development Environment)
-it designed specifically for python
-usually used for companies because can support larger programs
-companies are hired to add more stuff to the environment that's why only community editions are free and pro version (web development) is where you have to pay for - designed specifically for python and there loss of extra functionalities

3)Notebook Environment: Jupyter Notebook
-great for learning
-get to see your input and output for code & see multiple inputs because notebooks use cells where you can see each output of the code
- it allows you to split your code up and to run a specific line of code
- unlike the text editor where you have to write the code and let it develop then run the code which is great when you have larger pieces of code, Jupyter Notebook allows you to see the output immediately
-for learning small pieces of code using notebook environment is perfect for beginners
-support in line markdown notes visualizations, videos and more

Comparison operators

1)equal:

```
In[1]: 2 == 2
Out[1]: True

In[2]: 1 == 0
Out[2]: False
```

== is a comparison operator while = is an assignment operator

Not equal:

```
In[3]: 2 !=1
Out[3]: True
```

Greater Than:

```
In[5]: 2 > 1
Out[5]: True

In[6]: 2 > 4
Out[6]: False
```

Less than:

```
In[7]: 2  < 4
Out[7]: True

In[8]: 2 < 1
Out[8]: False
```

Greater than or Equal to

```
In[9]: 2 >= 2
Out[9]: True

In[10]: 2 >= 1
Out[10]: True
```

Less than or Equal to

```
In[11]: 2 <= 2
Out[11]: True

In[12]: 2 <= 4
Out [12]: True
```

Chained Comparison Operators:
-ability to chain multiple comparison to perform more complex text
-can use (and) (or)compare the two values together

Ex:

```
In[1]: 1 <  2  <3
Out[1]: True

In[2]: 1<2 and 2<3
 Out[2]: True

In[3]: 1==2 or 2<3
Out[3]: True
```

Integers

-exactly what you plug into the computer Ex:

```
In: 2+1
Out: 3
```

-can do addition, subtraction, multiplication, division straight forward

```
7 % 4
```

-tells you if there's a remainder in the solution Ex:

```
In: 50 % 5
Out: 0
```

-this is because 50 divided by 5 is 10 and there's 0 remainders left in the solution
-another thing to remember is that if you PEMDOTS if you want to combine the order of the operations unless you put a parenthesis in the first order of operation you want

Introduction to Strings

Strings are ordered sequences that use indexing and slicing to grab sub-sections or the string. These actions use [] square brackets and a number index to indicate the position of you wish to grab. Strings are sequences of characters, using the syntax of either single or double quotes. If you use single or double quotation the output is still going to be "hello" or 'hello'. It's a personal choice to use a single or double quotation when writing code. Ex:

```
In:" 'hello'
Out: 'hello'

In: "hello"
Out: "hello"
```

Now what if you want to have two lines show up in the reading :

```
In: print ("hello world one")

    "hello world two"
```

-the print on top tells python that you want to print the top string in the output
OR you can put

```
        In:  print  ("hello world one")
        Print ("hello world two")
```

-its not very necessary to type the print on the bottom of the line because python is going to print the bottom line as well

What if you want to print two words on separate lines without having to print two lines?

```
In: print ('hello \n world')
Out: hello
          World
```

Printing two string with a space/ gap?

```
In: print ('hello \tworld')
Out: hello  world
```

Knowing the length of a string?

```
In: len('hello')
Out : 4
```

4 letters in the string

```
In: len (‘I am’)
Out: 4
```

-There’s 3 letters but a space counts for a letter which makes it 4

Indexing and Slicing with Strings:
What are you indexing in this string?:

Indexing:

```
In: mystring [0]:
Out: ‘H’
```

-depending on the position the character is at the first letter of indentation always starts at 0 until the end

reverse indexing:
Ex:

```
In: mystring[-2]
Out: ‘l’
```

However, reverse indexing starts at -1 unlike indexing in the positive direction which starts at 0.

Grabbing parts of the string
Always reference your string to something first then run that code in order to manipulate it later.
Setting a variable for a string For ex:

```
In: mystring =  ‘hello world’

In: mystring [0]
Out: h
```

Grabbing parts of the string:

```
In: mystring[2:]
Out: llo world
```

The semi colon means that your starting from the index position 2 all the way to the end of the string.

What if you need the beginning of the string?

```
In: mystring [:3]
Out: hel
```

You start at the beginning of the string and you end with the letter string 3 however this means that it goes up to the letter 3 but not including 3.

What if you want the middle of the string?

```
In: mystring[3:6]
Out: lw
```

-you don’t include index position 6 so the output will return 2 letters with a space.

Another alternative to grab the whole string is:

```
In: mystring[::]
Out: ‘hello world’
```

What about jumps of 3?

```
In: mystring[::3]
Out: hlo
```

-Start at 0 then do jumps of 3 strings

What about picking which string is in certain places?

```
In: mystring[2:7:2]
Out: ‘ceg ‘
```

-start, stop, step
Start at that position: stop at that position: jump / increase at the interval provided

How to reverse a string?:

```
In: mystring [::-1]
Out: dlrow olleh
```

-the string is backwards

String Immutability

Objects that are built in within python like integrals, floats, Booleans, strings, and tuples cannot be changed. Mutated objects in python are lists, dictionaries and sets.
Change the string Sam into Pam? -you
CANT do this directly

```
In: name = Sam
In:name[0] = ‘p’
```

In python you cannot grab a letter and use it as a string it doesn’t work like that.
Instead you have to add the p in the string and index the string directly. Ex:
In: #name[0]= ‘P’

The key is the #
-rewrite the name using indexing like
In: last_name = name[1:]
The new strings = the previous with a condition of indexing from 1: (1 to the end)

```
name = "Sam"
name[1:]
'am'
last_letter = ‘am’
```

Now that the string of ‘am’ has been grabbed from ‘Sam’ you can add the P to change the string to Pam. Ex:

```
In: ‘p’ + last letters
Out: ‘pam’
```

This is known as string concatenation which is merging or combining strings.
Theres two different ways of concatenation:
1)would be simply just adding a string with addition, subtraction, multiplication, division

```
In: 2+5
Out: 7
```

OR
2) adding numbers with quotes Ex:

```
In: ‘2’ + ‘3’
Out: 23
```

-this really does make python very flexible with adding different integers

Ex:

```
In: x= ‘hello world’
```

```
In: x + “it is beautiful outside!”
```

You can also add phases to the string

Can end up rewriting the string as

```
x  = x +  “ it is beautiful outside!”
x
‘hello world it is beautiful outside!’
```

Now: Let say that you want to capitalize a string?
Let’s just start with ‘hello world’

```
  x= ‘Hello World’
  x.upper()
O ‘HELLO WORLD’
```

Or just type in x.tab and by clicking on tab it allows you to get all the functions necessary.

Provides a lower case option

```
In: x.lower()
Out: hello world
```

What about the split method?

```
In: x = ‘Hi this is a string’

In: x.split()
Out: [‘Hi’ ,  ‘this’ ,  ‘is’  ,  ‘a’  , ‘string’  ]
```

This function splits each string separately.

What about splitting a string by each letter?

```
In: x.split( ‘I’ )
Out: [ ‘H’ , ‘th’ ‘s’ , ‘s a str’, ‘ng’ ]
```

What is occurring is that the code splits the string by the letter ‘i’.

Variable Assignment

Rules for variable names
-names cant start with a number
-There can be no spaces in the name use an underline (____)
-can use any of theses symbols
-considered best practice that names are lowercase
-avoid using words that have special meaning in python like “list” and “str” “list” = list
“str” = strings
If these variables are used in python they are usually highlighted
Pros
-Python has dynamic typing and very easy to work with
Faster developing time
Cons
May result in bugs for unexpected data types
-need to be aware of type() For example:

```
Int my_dog = 1
my_dog = “Sammy”
```

-this is called dynamic language and are not able to do this in any other programming language like C++

How to assign a variable?

```
a = 5
a
5
```

What if you reassign your variable and run it again?:

```
In: a
Out: 10
```

Every time you run the same code you are doubling the value of what you got before.
Input: type(a)
Output: int (integers any number with no remainders in it)

What if you have a remainder in your number?

```
a = 30.1

type(a)
float
```

Any integer with a remainder is called a float.

You can substituted integers with strings and multiple to get a value:

```
my income = 100
      Tax rate = 0.1
      My taxes = my income  *tax rate

my taxes
10.0
```

(this code could be done in all in one line together)

String Interpolation: injecting a variable by using curly braces directly into a string

Ex:

```
print('This is a string {} '.format('INSERTED'))
```

This is a string INSERTED

-no output your directly printing a string

```
print('The {} {} {}' .format('fox', 'brown', 'quick')
The fox brown quick
```

Each curly braces correspond to one variable and the whole point is to indexing the variables desired in the curly braces.

Ex:

```
print('The {2} {1} {0}' .format ('fox' , 'brown' , 'quick' ))
```

Useful when you want to play around with a string.

Ex:

```
In(2): print( 'The {0} {0} {0}' .format ('fox', 'brown', 'quick' ))   2.
The fox fox fox
```

Another method is to abbreviate certain words in the string and place them in the curly braces.

Ex:

```
In[3]: print ('The {q} {b} {f}' .format (f= 'fox' ,b= 'brown' , q= 'quick')
The quick brown fox
```

Float Formatting: having fractional values as the result

Ex:

```
result = 100/777
result
0.1287001287001287
print("The result was {r}" .format(r = result))
The result was 0.1287001287001287
```

.format (dot formatting)

Changing significant figures of the result of code:

```
print ("The result was {r:1.3f}" .format(r= result)
result 0.129
```

The r indicates that you start with the first decimal place and round the third decimal place holder.

```
name = "Jose"
print(f'Hello, his name is {name} ' )
Hello, his name is Jose
```

```
name  = "Sam"
            age = 3
print(f'{name} is {age} years old.')
Sam is 3 years old
```

In conclusion you can either use dot formatting or injecting variables in your string.

```
a = 5
type(a)
 int
```

int = integer

What if you change the variable?

```
a = 30.1
type (a)
float
```

The variable is an integer.

Setting the variable equal to an integer and inputting what type of variable it is.

Lists

You can use different variables in your list:

```
In: my_list = [ 'STRING', 100, 23.2 ]
```

To find the length of your string:

```
len(my list)
 3
```

This means that there are 3 variables in my_list.

What if you want to know the position of a variable in that list?:

```
my_list[0]
```

Manipulating a list by indexing certain variables

```
 my list[1:]
 [100, 23.2 ]
```

Combining different types of list by the addition of two strings:

```
 another_list = ['four', 'five']
 another_list + my_list
 [ 'four' , 'five' , 'STRING' , 100, 23.2 ]
```

Saving the new string created by naming a new variable:

```
new list
 [ 'four' , 'five' , 'STRING' , 100, 23.2 ]
```

Quick and easy way to replace a string by indexing:

```
1.  new_list[0] = ‘ONE ALL CAPS’
2.  new_list
3.  [‘ONE ALL CAPS’, ‘five’ , ‘STRING’ , 100, 23.2]
```

You can change the variable on a list by indexing a new variable

What about adding new variables at the end of a list?:

```
1.  new_list.append(‘six’)
2.  new_list
3.  [‘ONE ALL CAPS’, ‘five’ , ‘STRING’ , 100, 23.2, six]
```

start by .append('____ include what you want on the end of the list ') ex:

```
1.  new_list.append(‘seven’ )
2.  [‘ONE ALL CAPS’, ‘five’ , ‘STRING’ , 100, 23.2, six, seven]
```

Removing new variables from the end of a list:

```
1.  new list.pop()
2.  ‘seven’
```

-the seven has now been removed from the list permanently…

```
1.   new_list
2.   [‘ONE ALL CAPS’, ‘five’ , ‘STRING’ , 100, 23.2, six]
```

In conclusion append adds variables at the end of a list and pop removed items at the end of the list.
Removing a variable by index the first item in the lisr?

```
1.  new list.pop(0)
2.  ‘ONE ALL CAPS’
```

Now check your new list:

```
1.   new list
2.   [ ‘five’ , ‘STRING’ , 100, 23.2, six]
```

Negative indexing:

```
1.  new list.pop(-1)
2.  six
```

Remove the last variable on the list by using negative indexing and then check the new list.

```
new list
[ ‘five’ , ‘STRING’ , 100, 23.2]
```

One of the benefits of using a list instead of a tuple is that there are functions to be able to sort out a list.

Ex:

```
new list = [ ‘a’ , ‘e’ , ‘x’ , ‘b’ , ‘c’ ]
new list.sort()
new list
 [‘a’ , ‘b’ , ‘c’ , ‘e’ , ‘x’ ]
```

Dictionaries

Dictionaries are collection of unordered, changeable and indexed by a set of keys. Dictionaries are written in curly brackets separated with commas.

How to construct a dictionary: `my_dict = {‘key1”: ‘values1’ , ‘key2’ : ‘value’ 2 }`

1.Curly brackets
2. Key
3. Semi Colon

```
mv dict
{‘key1”: ‘values1’ , ‘key2’ : ‘value’ 2 }
In[3]: mv dict [ ‘key1’]
Out[3]: ‘value1’
```

[‘key1’] dictates that you only want to code what is stating key 1 not everything that is in the curly brackets.

Doesn’t have to state key it could be anything else that relates to the variable its representing. Python gives you the flexibility to change a variable name. Ex:

```
prices_lookup = {‘apple’:2.99, ‘oranges’: 1.99, ‘milk’:5.80}
```

Run the code

```
prices_lookup
{ ‘apple’:2.99, ‘oranges’: 1.99, ‘milk’:5.80}
```

only want one string to lookup:

```
prices lookup[‘apple]
 2.99
```

That’s why dictionary are used instead of a list because in a dictionary you can use key words to look up certain items on the list.

Abbreviations and adding more elements to a dictionary:

```
d = { ‘k1: 123, ‘k2’ : [0,1,2] , ‘k3’: {‘insidekey’:100}}
```

```
d[ ‘k2]
[0,1,2]
```

What if you want a certain integers from a dictionary?:

```
d[‘k2’] [2]
2
```

This is done by stating a key then indexing the value needed.

In: letter = mylist[2]

```
letter
‘c’
```

A dictionary is useful when adding variables to a dictionary:

```
d = {‘k1’:100 , ‘k2’:200}
d
{‘k1’: 100, ‘k2’:200}
d[‘k3’: 300
```

Check if the variable added to the dictionary:

```
d
{‘k1’:100 , ‘k2’:200 , ‘k3’: 300]
```

See the keys of the dictionary

```
d.kevs()
dict_keys([‘k1’ , ‘k2’ , ‘k3’ ])
```

See the value of the key inside the semi colon:

```
d.values()
dict_values([ ‘100’, ‘200’ , ‘300’ ])
```

Check the items on the dictionary:

```
d.items()
 (['k1':100 , 'k2':200 , 'k3': 300])
```

It always has to be a string itself
Tuples: similar to a list but tuples have immutability

```
t = (1,2,3)
mylist = [1,2,3]
```

Checking the type of function:

```
type(t)
tuple
```

You can check the length of the tuples:

```
len(t)
 3
```

You can check the type of variables that are in tuples:

```
t
(1,2,3)
```

Indexing and slicing:

```
 t = ('one', 2)
t[0]
'one'
t[-1]
2
```

How many variable of a specific variable a tuple contains:

```
 t = ('a' , 'a' , 'b')
t.count(a')
2
```

The output indicates that two 'a' show up in the tuple.

What about the position of a certain variable its at?

```
t.index('a')
0
```

There is an 'a' in position 0, the first variable in the tuple

Boolean values: are two constant objects that contain a False and True statement You can even double check with python

```
1.  type(False)
2.   bool
```

-Bool is short for Booleans
For inequality signs:

```
1.  1>2
2.  false
```

Equality sign:

```
1.  1 == 2
2.   true
```

If, Else, and Elif statements

If statements are test expressions that are executed only when the condition is held true. If the expression is held to be false then the else statement code will be executed. The elif is an optional statement that double checks if the expression is true and if no expression if found true the block of code will be executed. The general concept of an if, else and elif statement are usually used for the purpose of checking if more than one condition is true or false. The syntax part states that the boolean must be spaced inside the if and else statement function.

If, else and elif
If case l:
Perform action 1 Else:
Perform action 2 Elif
case 2:
Perform action 3
If the statement below is true then print ‘ITS TRUE’

Ex:

```
1.  if True:
2.      3 >2
3.      print('ITS TRUE')
4.  ITS TRUE
```

You don’t need to have a greater than or less than symbol to make a statement True instead substituting words with equally signs like below:

```
 hungry = True

          If hungry:
      Print('FEED ME!')

FEED ME!

 hungry = false

          If hungry:
      Print('FEED ME!')
          Else:
              Print('Im not hungry')
Im not hungry
```

The two key statements Else & If

```
In[5]: loc = 'Bank'

if loc == 'Auto shops'
    print("Cars are cool!")
else: print ('I do not know much')
```

The code is going to print the string 'I do not know much'. Since loc == doesn't state the statement of given then else statement code is going to be executed.

```
loc = 'Bank'

If loc == 'Auto Shop':
    Print ("Cars are cool!" )
Elif loc == 'Bank':
    Print ("Money is cool"!)
Elif loc == 'Store':
    Print ("Welcome to the store")
Else:
Print("I do not know much")

Money is cool !
```

The loc is the condition if the condition doesn't meet the string which is bank the its going to go to the elif statement and the elif statement doesn't meet the condition its going to the Else statement

Example of a condition where either the if or elif statement meet the condition so the else statement is the last resort to print a string :

```
Name = ‘Pam’

If name == ‘Frankie’:
    Print(‘Hello Frankie!’)
Elif name =
Print(“ hello Sammy”)
Else:
Print(“What is your name?”)
```

In summary if, elif, else statements verbally input to the computer ‘Hey if this case happens, perform some action’.

Chaining comparison operators

Using logical operators to combine comparisons of and or

```
1 < 2
True
```

Logical statements:

```
1<2<3
True
```

```
1 < 2 and 2 > 3
False
```

Even through the left statement is true the right statement is False so the whole statement output is False.

Equality signs:

```
1 == 1
True
```

```
not 1 == 1
False
```

not is asking for the opposite Booleans

```
400 > 5000
 False
```

```
not 400 > 5000
True
```

List comprehensions are used to create new list and consist of expressions inserted with brackets.

There I will show a short cut to do lists of strings:

Before we used for loops: Ex:

```
1.   mylist = []
2.            for letter in mystring:
3.                  mylist.append(letter)
4.   mylist
5.  ['h', 'e' , 'l' , 'l' 'o']
```

-this is called appending each letter of the list

An easier method to append each letter of the list would be do a list comprehension. Ex:

```
1.  mylist  = [letter for letter in mystring]
2.  In[4]: mylist
3.  Out[4]: ['h' ,'e' , 'l' , 'l' , 'o']
4.
5.  mylist = [x for x in 'word']
```

Same concept of using the .append function, in shorter terms of coding material

```
1.  mylist = [x for x in range(0,11) if x %2==0]
2.  mylist
3. [0, 2, 4, 6, 8, 10]
```

-%2 will show you the even numbers of the list

```
1. x **2 for x in range(0,11) if x%2==0
2. [0, 4, 16, 36, 64, 100]
```

** the square root of the even numbers

-doing more complex arithmetic's of inserted in x** like:

In[9]: celsius = [0,10,20,34.5]

Fahrenheit = [((9/5)*temp + 32) for temp in celcius] called a list comprehension

```
1.  fahrenheit
2.   [32.0, 50.0, 68.0, 94.1]
```

-0 degree celcius is 32 degrees in Celsius

Fahrenheit = [((9/5)*temp + 32) for temp in celcius] convert into a for loop -❼

```
1.                    Fahrenheit = []
2.                    For temp in celcius:
3.                    Fahrenheit.append(( (9/5)*temp + 32))
4.                    Fahrenheit
5.                    [32.0, 50.0, 68.0, 94.1]
```

How to use an if and elif statement in a list comprehension:
Cons of a list comprehension is that it quickly becomes unreadable or difficult to read when you come back a few months later to read the code. It will definitely take you longer to read code after a few months of not seeing a list comprehension. Ex:

```
results = [x if x%2==0 else 'ODD' for x in range (0,11)]
results
[0. 'ODD', 2, 'ODD' , 4, 'ODD' , 6, 'ODD' , 8, 'ODD' ,10]
```

Have a list of even and odd when its odd numbers in a list

Nested loop:

```
mylist = []

            For x in [2,4,6]:
                   For y in [100,200,300]:
                          Mylist.append(x*y)
mylist
[200, 400, 600, 400, 800, 1200, 600, 1200, 1800]
```

What the list is actually doing is that its taking the number 2 and multiplying it by each of the numbers then 4 multiplied by 100, 200, 300 = 400, 800,1200

❾Now lets see a list comprehension:

```
mylist = [x*y for x in [2,4,6] for y in [100,200,300]]
mylist [200, 400, 600, 400, 800, 1200, 600, 1200, 1800]
```

Its very possible to condense this but keep in mind that it might be harder to read in the future. Its suggested to always try to make the code you write readability first.
For loops
A for loop is used to iterates over the sequence several of time. Sequences that can be iterate are strings, lists, tuples, and even built in Iterables for dictionaries such as key or values

String Formatting

-often you want to " inject" a variable into a string for printing. For example
- my_name = "Jose"
-print ("Hello" + my_name)
-there are multiple ways to format string printing variables in them
There are 3 different ways to explore string interpolation which are string format % formatting and f strings.

String formatting
String formatting is a farirly new method in python that replaced the older method of using an % operator.

```
1.          name = 'World'
2.  program = 'Python'
3. print(f'Hello {name}! This is {program}')
```

% formatting:
Is an older operation used in python however the basic format is by using the %s to tell where the substituted value of a name as indicated in the string . Ex:

```
1.  '%x' % errno
2.  'badcoffee
3.  Hey %s, there is a 0x%x error! %(name, errno)
4.  Hey Bob, there is a 0xbadcoffee error!
```

String interpolation / f-string:
String interpolation is when you place a substituting value into a placeholder in a string.

```
1.  def greet(name, question):
2.  return f"Hello, {name}! How's it {question }?
3.
4.  greet ('Bob' , 'going')
5.  "Hello, Bob! How's it going?
```

Differences between a list & a dictionary

A list retrieve certain items from a list by indexing and can order the function on the list. Very importantly list are mutable type meaning that lists can be change over time. A dictionary are mapping and do not retain order but they are ordered. The reason why dictionaries don't retain order is because they their entry contains a key and a value store.

Function: creates block of code to actually make a simple function call that can be repeated constantly. Once you become a more advanced programmer a key part is being able to write code clean and effective. This is because when you write code you primary have two audiences your user and yourself.

What does a function start off by?
Def name_of_function(): -> -name_of_function should be all lower case Remember that the docstring explains the function.
For example:

Print('Hello" +name)

Next

```
1.  name of function(“Jose”)
2.  Hello Jose (print)
```

In this case we use the return button that send back the result of the function instead of just printing it out.

Ex:

def add_function(num1,num2): Return
num 1+ num2
>>result = add_function(1,2)
>>
>>print(result)
>>3

In[1]: def name_function(): The colon indicate that theres going to be an indended block of code
print(‘Hello’)

In[2]: name_function()
Hello

() : the parenthesis is key for the function to print ‘Hello’

The documentation in the string lets people know what your putting in the code
ex:

```
1.                  def name_function():
2.                  ‘’’
3.                  DOCSTRING: information about the function
4.                  INPUT: no input…
5.                  OUTPUT: hello…
6.                  ‘’’
7.                  Print(‘Hello..’)
```

Documentation strings helps people understand what the purpose of the function is:

```
1. def say_hello(name):
2.                 Print(‘hello’+name)
3.  say_hello(‘Jose’)
4.          hello
```

Getting an error when you didn't pass in a parameter and expect to have a positional argument. In order to resolve this issue you can use a default name to not get the error like this.

```
def say hello(name= 'NAME'):
          Print('hello'+name)

 say hello()
      Hello NAME
```

The 'NAME' function clears the default error. The default only occurs when you don't provide anything to the parenthesis function like: `say_hello()`

Ex:

```
 dog check('Dog ran away')
 False
```

Why does the statement show that its false if dog shows up in the documentation string?: The reason towards this issue is that dog is capitalized which would make the statement false. A simple syntax mistake like this can be the reason why the statement is false.

How can I fix the statement to show its true?

```
        def dog_check(mystring):
        if 'dog' in mystring.lower()
        return True    4.                        else:
                                 return False
```

What .lower() function does is that it makes the whole statement lower case which then allows it to take the string and run all the whole string has a lower case letter before checking for the word dog.

```
In[9]: dog check ('Dog ran away')
Out[9]: True
```

'dog' in mystring.lower() -> this is a Boolean itself theres a much simpler way to rewrite the code to show a true or false statement

```
'dog' in 'dog ran away'
True
```

Rewrite into a Boolean

```
def dog check(mystring):
            return ‘dog’ in mystring.lower()
 dog check( ‘Dog ran away’ )
 True

 ‘dog’ in ‘dog ran away’
 True
```

Pig Latin

Here are the basic rules about pig Latin. If a word starts with a vowel , add ‘ay’ to end. The second rule is that if word does not start with a vowel put first letter at the end then add ‘ay’.
For example: -word -> Ordway
-apple -> appleay

```
def pig latin = word[0]

       first_letter = word [0]
```

```
    # check if vowel        if
first_letter in ‘aeiou’:
Pig_word = word + ‘ay’
else:
        pig_word = word[1:] + first_letter + ‘ay’

    return pig_word
```

```
pig latin(‘apple’)
appleay’
pig latin(‘word’)
‘ordway’
```

Functions really improve capability.
The main function is that you have the def to start the function by.

Coding exercises

In python object and data structure
Print formatting by write an expression using any of the string formatting

Test:

```
input(“Type your age: “)
Type your age: 4
```

```
set('Mississippi')
 {'M' , 'i', 'p' , 's' }
```

-reminder that sets only show unique letter

```
a = int(input("Type your age: "))
Type your age: 5
```

```
a
5
```

```
I start = ["mildred"]
 start
 ['mildred']
```

Numbers

Two types of numerical information
1)integers which are whole numbers
2)floating point which are numbers with decimal value

Strings are ordered sequence of characters

Lists: ordered sequence of objects that are mutable you can change items in your list

Tuples: ordered sequence of objects (immutable) that you cant change objects Dictionaries:

key-value pairing that is unordered

Square root:

-most functions you have to import through libraries as python doesn't recognize certain functions

```
import math
math.sqrt(4)
2.0
```

Input values:

```
input("Type you age:")
Type your age: ____
```

Sets:

```
: set('Mississippi")
: { 'M', 'i' , 'p' , 's'}
```

Set only show the unique values in the string

List:

```
my list = [1, ‘two’ , 3.14159]
type(my list)
list
```

Type is a function source in python.

What about indexing to use which type of object are in the list. Simply just use the same concept has before:

```
In[3]: type(my_list[0])
```

String:

```
st = ‘Print only the words that start with s in this sentence’
```

-right here your assigning the variable st with a string

```
st.split()

'Print',
'only',
'the',
'words',
'that',
'start',
'with',
's',
'in',
'this',
'sentence']
```

Split the string one by one
What if you only want the word with no ‘’(curly braces) Ex:

```
for word in st.split():
        Print(word)
```

What if you want to print the ‘s’ word

```
for word in st.split():
   if word[0] == 's':
      print(word)
```

Prints all the words that start with the letter s:

```
start
s
sentence
```

Indexing also allows you to grab the first word by indexing .

What If you want to grab a upper case letter of all the ‘s’

```
  st = ‘Sam Print only the words that start with s in the sentence’

for word in st.split()
    if word[0].lower() == ‘s’:
       print(word)

Sam
start
s
sentence
```

-now python can be able to use all the words with s regardless if they are capitalized or not

Another method to grab either capitalized or lower case words with s would be:

```

for word in st.split():
    if word[0] == ‘s’ or word[0]== ‘S’
       print(word)
Sam
start
s
sentence
```

Using range() to print all the even numbers from 0 to 10

```
list(range(0,11,2))
[0,2,4,6,8,19]
```

What if you actually want to print the numbers:

```
for num in range(0,11,2):
print(num)

0
2
4
6
8
10
```

-For range the only thing to realize is that you have to go up to 11 and the step size is 2.

```
(0,11,2)
0= start
11 = stop
2= step size
```

-All for a range

Use a list comprehension to create a list of all numbers between 1 and 50 that are divisible by 3

```
In[8]: #code in this cell
       [x for x in range(1,51) if  x%3 == 0]
Out[8]: [3,6,9,12,15,18,21,23,27,30,36,39,42,45,48]
```

%3: makes it divisible by 3

Now go through the string below and if the length of a word is even print “even”!

```
                st = ‘Print every word in this sentence that has an even number of letters’
                for word in st.split():
                If len(word) %2 == 0:
                print(word+ ‘is even!’)
```

Homework set

Problem 1

Write a program that prints the integers from 1 to 100. But for multiples of three print “Fizz” instead of the number and for the multiples of five print ”Buzz”. For numbers which are multiple of both three and five print “FizzBuzz”

```
for num in range(1,101):
    if num%3 == 0 and num%5 == 0:
       print(‘FizzBuzz’)
    elif num%3 == 0:
         print(‘Fizz’)
    elif num%5 == 0:
         print(‘Buzz’)
    else:
         print(num)
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
Fizz
```

The main thing is to check 3 and 5 first so it doesn't complicate the elif statement

Problem 2

Use list comprehension to create a list of the first letters of every word in the string below:

```
In[10]: st = 'Create a list of the first letters of every word in this string'
In[11]: [word[0] for word in st.split()]
Out[11]: [ 'S', 'P' 'O', 't', 's', 'w' , 's' 'I' 't' ,'s']
```

Object Oriented Programming

Allows programmers to create their own objects that have methods and attributes

-it's a really great way to scale your code to organize your code

-like for string, list, dictionary or other objects you were able to call methods off of them as .method_name( ) syntax

Theses methods act as functions that use information about the object as well as the object itself to return results or change the current object

Ex: like when you did .sort( ) on a list it appended the list itself

-this would include appending to a list or counting the occurrences of an element in a tuple

OOP allows users to create their own objects

-allows you to create code that is repeatable and organized

-when you work with outside libraries you can see how they use object oriented programming and make it useful for coding

Brief Overview:

-the basic way to define an object is by the class keyword: class NameOfClass (): classes are the reserved upper case that's why for variable name and function names are in lower case because classes are the ones that are reserved for upper case words / naming schemes

def_init_ param 1 and param 2 are parameter that python expects you to pass def_init_(self, param1 , param2): self.param1 = param1 self.param2 = param2

def_init_(self, param1 , param2):
self.param2 = param2 #perform some action
Print(self.param1)

def some_method(self) --- pass in the self key word to let python know that this isn't just some function it's a method that is connected to this class

self.param2 = param2 (Were linking the self parameter)
def some_method(self) --- pass in the self key word to let python know that this isn't just some function it's a method that is connected to this class Object Oriented Programming Part One: (section 8)

Object Oriented Programming attributes and class Keyword

Introduction: python has objected oriented programming language has two concepts class & object. A class is the blueprint model for its object and the class is a model where to define the attributes and the behavior.
A class has two characteristics a attribute and a behavior. A
real life example:
Parrot is an object,

- name, age, color are attributes
- dancing , singing are behaviors

Add on to the real life example:
A class can be though about a class as a sketch of a parrots with multiple labels contains the details about the name, color, size and etc. Based on these details you can now study the parrot which would be the object.

How would you start programming the concept of an Object Oriented?:

```
In[1]: class Parrot:

        pass
```

`class` is a specific keyword to define an empty class, `Parrot`.
From a specified class the next step would be to construct an object (instance) . Once a class is defined only the description of an object can be defined. No memory or storage is located: ex:
Now the overall concept:

```
In[1]: class Parrot():

           #class attribute

           species = “bird”

           #instant attribute

            def __init__(self,name.age)

                self.name = name

                self.age = age

     #instantiate the Parrot class

In[2]:   blue = Parrot(“Blu”, 10)

In[3]:   woo = Parrot(“Woo”, 15)

    #now access the class attributes

In[4]: print(“Blu is a {}”.format(blue.__class__.species))

Out[4]: Blu is a bird

In[5]: print(“Woo is also a {}”.format(woo.__class__.species))

Out[5]: Woo is also a bird

     #acess the instance attributes

In[6]: print(“{} is {} years old”.format(blu.name, blue.age))

Out[6]: Blu is 10 years olf

In[7]: print(“{} is {} years old”.formate(woo.name, woo.age))

Out[7]: Woo is 15 years old
```

The concept of Object Oriented Programming follows some basic principles: inheritance: a process of using details from a new class without modifying exitsting class
encapsulation which is hiding the private details of a class from other objects
polymorphism: a concept of using common operation in different ways for different data input

Object Oriented Programming part one:

-methods: are objects you perform with the object you created
A function is a method inside a class and will work some type of way

Before we would use functions: Ex:

Class is our user defined object in other words class is the blue print that defines the nature of a future object and the classes are the instant of the object instant is the specific object of the nature of the class. For convention we use capital name by convention

```
In [12]: class SampleWord ():
```

You can see that we need pass as a placeholder if not it will just return back as error
-In addition you need to include the open and close parenthesis as well as the semi colons

```
In [14]: class Sample ():

             pass

In[15]: my_sample = Sample()

In[16]: type(my_sample)

Out[16]: __main__.Sample
```

```
In[1]:my_list = [1,2,3]

In[2]:myset = set()

In[3]:myset.add

Out[3]:<function set.add>

In[4]:type(myset)

Out[4]: set

In[5]:type(my_list)

Out[5]: list
```

The type function lets the programmer know that the it's a sample type class
Circle()
What line 14 tells you that we created a sample class and then defined a variable to the sample ❼ as my_sample = Sample() this right here is defining a variable name of my_sample to sample.
Line 16 checks the type of function . Now were going to create attributes:

```
            pass

In[13]: class SampleWord():

File “<ipython-input-13-72105eb0cdf6, line 2

SyntaxError: unexpected EOF while parsing
```

```
In[1]: class Dog():

            def __init__(self,breed):

                self.breed = breed

In[2]:my_dog = Dog(breed= ‘Lab’)

In[3]: type(my_dog)

Out[3]: __main__.Dog

In[4]: my_dog.breed

Out[4]: ‘Lab’
```

For line 1 __init__ is the constructor of the class self which means that it’s the instant of the object itself the breed , then the argument self . breed = to the attribute to the parameter mybreed which is the same as the parameter itself.
In short term taking the argument and then assigning it to the self.attribute ❼can be anything related to the class itself (in other words self.attribute_name) choose the name of the argument. You can use the same parameter for all three of three of these names.

More examples:

```
In[1]: class Dog():

    def __init__(self,breed,name,spots):

      self.breed = breed

      self.name = name

     # I Expect boolean True/False

      self.spots = spots
```

Attributes itself don’t actually have to be strings they can be Booleans, list, an integer or a floating point.

```
In[2]: my_dog = Dog(breed= ‘lab’ ,name= ‘Sammy’ ,spots=False

In[3]: type(my_dog)

Out[3]: __main__.Dog

In[4]:my_dog.breed

Out[4]: ‘Sammmy’

In[5]:my_dog.spots

Out[5]: False
```

Always try to add documentation to notify programmer seeing your code that your expecting strings for breed, strings for name and spots for Booleans.

Quick recap for object Oriented: class Dog(): capitalize def **init ,** the special init methods acts as a constructor def__init__(self, breed , anem , spots) self.breed ← anything in this position is are instant of the class self.name in other words are attributes that you pass based on the paramenters you defined as self breed. Simple terms parameter name and attribute name are the same and usually see it 3 times. The first time would be the parameter name, attribute name and when you pass the code.

Object Oriented Programming Part Two:

Methods are objects you perform with the object you created

```
In[1]: class Dog():

           species= ‘mammal’

            def __init__(self,breed,name):

#attributes

#Take in arguments

#Assignit using self.attribute_name

     self.breed= breed

     self.name= name

#operations/actions->methods

def bark(self,number):

print (‘WOOF! My name is {}”. format(self.name,number))

#methoddo need to be executesthat’s why you have open and close parenthesis

In[2]: my_dog= Dog(‘Lab’,‘Frankie’)
```

The difference between attributes and methods is that attributes don't have open and close parenthesis. In addition attributes aren't something that you need to execute its just information that is being called back. Methods need to be executed and do have open and close parenthesis .

```
In[3]: type(my_dog)

Out[3]: __main__.Dog

In[4]: my_dog.species

Out[4]: 'mammal'

In[5]: my_dog.bark(10)

WOOF!My name is Frankie
```

```
In[6]: my_dog.bark

Out[6]: <bound method Dog.bark of <__main__.Dog object at 0x10fe569e8>>

In[7]: my_dog.bark()

WOOF! My name is Frankie
```

Methods which is an action it can take

```
In[8]: class Circle():

        #Class obkect attribute

        Pi = 3.14

   def __init__(self,radius = 1):

       self.radius = radius

   def get_circumference(self):

        return self.radius * self.pi * 2

In[9]: my_circle = Circle()

In[10]: my_circle.pi

Out[10]: 3.14
```

You get 3.14 back because the attribute of pi is going to be the same regardless

```
In[11]: my_circle.radius

Out[11]: 1
```

How did you get the radius of 1 its because the radius is part of the default. How do you overwrite the default of the radius?

```
In[12]: my_circle = Circle(30)

In[13]: my_circle.raidus

Out[13]: 30
```

You set my_circle equal to the radius of the circle and include specific radius number you want.

```
In[14]: my_circle.get_circumference()

Out[14]: 188.4
```

The attribute doesn't actually have to be defined

Radius = is the argument we need to define the circle as and provided a default value of 1

Object Printed Programming Part Three:

```
In[1]: class Animal():

           def __init__(self):

               Print("ANIMAL CREATED")

            def who_am_i(self):

               print('I am an animal')

             def eat(self):

                 print("I am eating"

In[2]: myanimal = Animal()

          ANIMAL CREATED

In[3]:myanimal.eat()

I am eating

In[4]:myanimal.who_an_i()

I am an animal
```

This is a newly formed class

```
In[5]: class Dog(animal):

              def __init__(self):

                  Animal.__init__(self)

                   Print("Dog created")
```

Animal what animal in the parenthesis means is that you are deriving animal from the base class of the def init(self)

```
In[6]: mydog = Dog()

       ANIMAL CREATED

        Dog created

        mydog.eat()
```

What its doing is that its running the inhit method on dog
-you don't have to rewrite all the methods in order to have that certain function

```
In[7]: mydog.who_am_i()

       I am an animal
```

-you have to rerun the variable the dog in order to create the….

```
In[8]:mydog.bark()

       WOOF!
```

Inherit from the base line and now you have all the methods you want. In addition you can always add in methods in

Polymorphism: refers to the way that different object classes can refer to the same name and those methods can be called from the same . Can be carried out through inheritance with subclasses make use of base class methods or overriding the . A real life example of this would be when you see a bird that walks like a duck and swims like a duck and quacks like a duck I call that bird a duck. You check if the object quacks like a duck and walks like a duck rather than asking whether the object is a duck. Several classes can have the same method name but have different implementations for the Sam methods. A function will be able to run as theses polymorphic methods without having to know which classes they are.

Ex: Lets create two different types of classes: the first one a shark and the other one a clownfish

```
In[1]:class Shark():

          def swim(self):

            Print("Theshark is swimming.")

           def swim_backwards(Self):

             Print("Theshark cannot swim backwards,but can sink backwards.")

            def skeleton(Self):

                 Print("Thesharks skeleteonis made of cartilage."
In[2]: class Clownfish():

            def swim(self):

                Print("Theclownfishis swimming.")

           def swim_backwards(self):

          Print("Theshark cannot swim backwards,but can swim backwards.")

            def skeleton(Self):

                 Print("Theclownfish'sskeleteonis made of bone."
```

Above you can see that both the shark and clownfish have three methods with the same name in common the def swim. However the functionalities of the methods differ for each of the class. Each object behaves as expected because the output of the string is completely different. In conclusion different functions and methods through polymorphism make use of this python feature that provides greater flexibility for your object- oriented code.

```
In[9]:class Dog():

      def __init__(self,name):

          self.name = name

      def speak(self):

           return self.name + “says woof!”

In[10]: class Cat():

             def __init__(self,name):

                 self.name = name

              def speak(self):

                   return self.name + “says meow!”

In[11]: niko = Dog(“niko)

         felix = Cat(“felix”)

In[12]: print(niko.speak())

        Niko says woof!

In[13]: print(felix.speak())

        Felix says meow!
```

Here we have a dog class and a cat class each of them have a speak method. Each of them have a speak method that is unique to say meow and woof, the name is the unique part of each method. There are several ways of how to present a polymorphism.

1) Way is by loop:

```
In[14]: for pet in [niko,felix]:



    print(type(pet))

    print(pet.speak())
```

```
<class '__main__.Dog'>
niko says woof!
<class
'__main__.Cat'> felix
says meow!
```

Another way to show this is through interation which is a more common ay

```
In[15]: def pet_speak(pet):

            Print(pet.speak())

In[16]: pet_speak(niko)

 Niko says woof!

In[17]: pet_speak(felix)

Felix says meow!
```

The thing is that they share the same methods speak inherit through the message or pass the method through speak

```
In[18]: myanimal = Animal('fred')

In[19]: myanimal.speak()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last
)
<ipython-input-84-188713a00ef6> in <module>()
----> 1 myanimal.speak()

<ipython-input-82-0fdad9d8a130> in speak(self)
      5
      6     def speak(self):
```

```
----> 7         raise NotImplementedError("Subclass must implement this ab
stract method")
      8

NotImplementedError: Subclass must implement this abstract method
```

In th base class itself its not expected you to do anything but the class animal is expecting tod o overwrite the speak metho

```
In[20]: class Dog(animal):



           def speak(self):

               return self.name+ ‘says meow!”

In[21]: fido = Dog(“Fido”)

In{22]: isis = Cat(“Isis”)

In[23]: print(fido.speak())

Fido says meow!

In[24]: print(isis.speak())

Isis says meow!
```

A different way to use a polymorphism is to open a different name file like another class to open excel file another class for Microsoft words you would want them to share the same open method example of polymorphism is that you have some function.

Special Methods/Dunders

Theses special methods let you emulate the behavior of built-in types
Special methods: are using a length and print function on your own code created
All special methods are easy to recognize because they start and end with double underscores.

Ex:
__init__ or
__str__

```
In[1]: mylist = [1,2,3]

In[2]: len(mylist)

Out[2]: 3
```

Now what if you want to check the length of one of my own objects now:

```
In[3]: class Sample():

In[4]: mysample = Sample()

In[5]: len(mysample)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last
)
<ipython-input-5-97b353a42cb3> in <module>()
----> 1 len(mysample)

TypeError: object of type 'Sample' has no len()
```

Your going to get back an error because its an object itself

```
In[6]: print(mysample)

<__main__.Sample object at 0x10f023940>
```

If you print sample it tells you where your object is located

```
In[7]: print(mylist)

[1, 2, 3]
```

Now the question arises how am I suppose to use length and print function to use on my own to built in objects?
That's where the special methods comes in: class
Book():

```
In[8]: class Book():

        Def __init__(self, title, author,pages):



              Self.title = title

              Self.author = author

              self.pages = pages
In[9]: b = Book('Python rocks', 'Jose' , 200 )

In[10]: print(b)

<__main__.Book object at 0x10f0d4710>
```

What the print function does is that it ask what is the string representation of b

```
In[11]: str(b)


Out[11]: '<__main__.Book object at 0x10f0d4710>'
```

Instead what we can do is:

```
In[12]: class Book():

             def __init__(self,title, author,pages):

                 self.title = title #a
self.author = author #b
self.pages = pages #c
              def __str__(self):




                  return f"{self.title} by [self.author]"
```

```
   #this is super crucial in for the stirng and print to return

In[13]: b = Book(‘python rocks’ , ‘Jose’, 200)

In[14]: str(b)

Out[14]: ‘Python rockes by [self.author]’
```

```
In[15]: print (b)

Python rocks by [self.author]
```

It doesn’t matter if you use the string or print it will still return the string itself

```
In[16]: len(b)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last
)
<ipython-input-19-97d8916a185b> in <module>()
----> 1 len(b)

TypeError: object of type 'Book' has no len()
```

Still its doesn’t inform you about the length of the string

```
In[17]:  class Book():

          def __init__(self,title ,author,pages):

              self.title = title

              self.author = author

              self.author = pages

           def __str__(self):

               return f"{self.title} by {self.author}"

           def __len__(self):

                return self.pages

           def __del__(self):

                 print("A book object has been deleted")

In[18]: b = Book('Python rocks', 'Jose', 200)

In[19]: del b

A book object has been deleted

In[20]: len(b)
```

-With this function you can delete any of the necessary or required you just have to state what the variable name is and enter the special method
-you cant delete the variable twice because once it has been deletedthe next time the variable name b will not be defined.

Object Oriented Programming Homework

Homework Assignment:
Problem 2:

```
In[1]: class Cylinder():

         def __init__(self,height= 1, radius = 1):

          self.volume = 5

          self.surface_area = 9

In[2]: c = Cylinder()

In[3]: c.volume

Out[3]: 5

In[4]: c.surface_area

Out[4]: 9
```

Always have to to define what c is before you start using the . function in programming.

Problem 1:

```
In[1]: class Line:

        def __init__(self,coor1,coor2):

        def distance(self):

        def slope(self):


In[2]: coordinate1 = (3,2)

        coordinate2 = (8,10)

   li = Line
```

After you use tuple method you have to extract it someway by using tuple unpacking or indexing to extract the string.

Problem 2:

```
In[1]:  class Line():

              def __init__(self,coor1, coor2):

                  self.coor1 = coor1

                   self.coor2 = coor2

               def distance(self):

                   x1,y2 = self.coor1

                  x2,y2 = self.coor2

                  return((x2-x1)**2 +(y2-y1)**2)**0.5

               def slope(self):

                   x1,y1 = self.coor1

                   x2,y2 – self.coor2

                     return(y2-y1) / (x2-x1)
```

The position of return has to be in the same indentation of the slope if not python wills how an error in the code.

Modules and Packages

Pip Install and PyPi:

Up to this point we have only used libraries that come internally with python. There are many other libraries that are available that people have open-sourced and is shared on with PyPi. We can use pip install at the command line to install the packages necessary. By installing python through Anaconda distribution you have already installed pip. pip is a simple way to download packages at your command line directly from the PyPi repository. There are packages already created for almost any use case you can think of. Just by using a link to the Pypi page for the page -have to understand how to understand how to install external packages

(69) Modules and Packages: Now your own how to create your own modules and packages. To break it down modules are .py scripts where you use another .py script to complete it. Then packages are just a whole collection of modules.

Shows you how to create modules and packages:

Modular programming: two mechanism that make up modular programming is modules and packages. Some advantages in modularizing code in large applications:

1) simplicity: rather that focusing on the entire problem, you'll be able to focus on a small portion of the problem.
2) Maintainability: modules are designed to enforce logical boundaries between problems domains. In the sense that you can make changes to a module without having any knowledge of the application outside the module.) This makes the large application more collaborative for a team of programmers.
3) Reusability: Functionality in a single module can be easily reused by other pieces of the application which leads for no need to duplicate a code.
4) Scoping: Modules have a separate namespace which helps with any confusion in different areas of a program.

Accessing Modules from Another Directory:
-Modules can be used in any programming project and should be keep in a particular directory that's is tied to a specific project.

Lets see how to use a module from a location other than the same directory where your man program is:

Appending Paths: first method of obtain a module from another directory
This is done by invoking the path of the module via the programming files that use the module. Your path(which is the list of directories Python goes through to search for a module or a file is being stored in th system module. Since the path is a list you can use the append method to add new directories to the path. How to append a path of a module to another programming files.
First youre going to start by importing the sys module.
sys is part of the python Standard Library

How to install packages:

1) Open terminal
2) pip install request
   -open the colormap library in the

This is just to install the program once its already installed
User: >python
>>> from colora import init
>>> init
>>> from colorama import Fore >>>
print (Fore.Red + "some red text")
Some red text
>> >print(Fore.Green + "Switch to green")
Switch to green
>>> quit()

C: \Users\Marcial>
How to work with excel files and how to use it

Goggle search on Python Package for excel
Just google search whatever youre looking for _____
Try to use th python .org search and if you go to the download page it tells you about the documentation link and usually hosted by .readthedocs.io

Go back to the terminal and type in pip install openpyxl

69: Modules and Packages:
Now that we know how to understand how to install external packages lets explore how to create our own modules and packages
-Modules are just .py scripts that you call another .py script
A module is a single file or could be files that are imported under one import and used. A package is a collection of modules in directories that give a package hierarchy. Any python file is a module its name being the files. You can use any IDE (like subline text editors) lets get started by open in subline text file. Module is a .py scribe being brought in a file. To open:

Namespace Package:

- it is extremely important that every distribution uses the namespace package omits the __init__.py
-If any __init__.py here
-module.py
-module.py
-form setuptools import setup, find find_namespace_packages
-setup(name = 'mynamespace-subpackage-a',
Packages= find_namespace_package(include=['mynamespace.*]))

Creating a package with several subpackages in Python:

-The really package/subpackage2 need to be able to cal
-package/tests calls both subpackages(using pytests)

Section 70_name_ == "__main__":
-Something when you are importing from a module you would like to know whether a module function is being used as an import or if you are using a .py file of that module? - the difference between python and other languages is that python all the line in subline text get run regardless what they are doing

-in python there isa built in __name__ function
What python does is that in the back of the run it sets __name__ == “__main__
What it does is that it states if the build in function equals to each other then its going to go ahead and run

One.PY is the main script and will print

```
In[1]: #one.py

In[2]:

In[3]: def func():

IN[4]:      print(“FUNC() IN ONE.PY”)

In[5]:

In[6]:       print(“TOP LEVEL IN ONE.PY)

In[7]:

In[8]:if__name__ ==’__main__’:

In[9]:   print(‘ONE.PY is being run directy!’)

In[10]: else:

In[11]: print(‘ONE.PY has been imported!’)
```

```
Desktop — -bash — 80×14
Mildreds-Air:~ mildredmonsivais$ python one.py
python: can't open file 'one.py': [Errno 2] No such file or directory
Mildreds-Air:~ mildredmonsivais$ cd Desktop
Mildreds-Air:Desktop mildredmonsivais$ python one.py
python: can't open file 'one.py': [Errno 2] No such file or directory
Mildreds-Air:Desktop mildredmonsivais$ python one.py
ONE.PY is being run directly!
Mildreds-Air:Desktop mildredmonsivais$ python two.py
ONE.PY has been imported!
TOP LEVEL IN TWO.PY
FUNC() IN ONE.PY
TOP LEVEL IN ONE.PY
TWO.PY is being run directly
Mildreds-Air:Desktop mildredmonsivais$
```

The built in function of __name__ ==_main__ has been assigned to the string and has been called directly. Now you can see that two.pyf ile has been imported from a different file. You have manipulated the built in function of(__name__== __main__) to see which file gets imported or which one is being runned directly from the computer.

You have all the logic of the code in the beginning of the lines then the function of the code running towards the end of the lines. This is the very basic function when starting to make your own modules and packages.

Errors and Exception Handling

A user may write to a file that was the only opened in mode =’r’ if there is any error in your code the entire script will stop. How to fix this is that we use error handling to let the script continue with other code even if there is an error Three key words:
Try: block of code that is attempted (but could lead to an error)
Except: block of code will execute in case there is an error in the try block
Finally: a final block of code is the block of code that will be executed regardless of an error

```
In[1]:def add(n1,n2):

          print(n1+n2)

In[2]: add(10,20)

       30

In[3]: result = 20

In[4]: number1 = 10

In[5]: number2 = input(“Please provide a number:”)

Provide provide a number:

In[6]:       add(number1, number2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last
)
<ipython-input-20-ad6929989779> in <module>()
----> 1 add(number1, number2)

<ipython-input-16-48bddae7e371> in add(n1, n2)
      1 def add(n1,n2):
----> 2     print(n1+n2)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
Python error is basically telling you that you cant add the integer and string together or you
can cancadinate two strings together, but you cant work with the two data types.

In[7]: add(number1, number 2)

       Print(“Something happened!”)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last
)
```

```
<ipython-input-21-c15b1ab87eee> in <module>()
----> 1 add(number1,number2)
      2 print("Something happened!")

<ipython-input-16-48bddae7e371> in add(n1, n2)
      1 def add(n1,n2):
----> 2     print(n1+n2)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

-No matter what block of code you will still get an error because you didn't fix the error above so python is not letting you continue it and nothing else if going to get executed.

```
In[8]:

try:

    #want to attempt this code

    #may have an error

     result = 10 +'10'

except:

      print("Hey it looks like you aren't adding correctly!!")

Hey it looks like you aren't adding correctly!

In[9]: result

Out[9]: 20
```

Now the block of code is being executed as you can see above, generally you can put except in any area of code in order to not get the error code

```
In[10]: try:

            #WANT TO ATTEMPT THIS CODE

             #MAY HAVE AN ERROR

             Result = 10 + '10'

Except:

       Print("Hey it look like you aren't adding corrrecly!")

Else:

    Print('Add went well!")
```

```
    Print(result)

Hey it looks like you aren't adding correctly!

#this is known as the try or else except statement

In[11]: try:

        f = open('testfile', 'r'

        f.write("Write a test line")

        except TypeError:

             print("Write was a type error!)

          print(result)

Hey you have an OS Error

I always run

#Right now were getting back I always run which means that youre only running the last script
on the code

#the difference between except statement and the finally block is that the except statement
will only run for the specific block of code but the finally function codew ill run for the
code of the finally and except statement

In[12]: try:

       f = open('testfile','r')

       f.write("Write a test line ")

except TypeError:

print('There was a type error!")

except:

print('Hey you have an OS Errror')

finally:

print('I always run")

Hey you have an OS Error

I always run

-even if you have an OS error then you will still have thee xcept and the finally code run
regardless of what you put in front of the except register code
```

Errors and Exception Homework

Problem 1:

```
In[1]:
try:


    for  i in [‘a’,’b’, ‘c’]:
result =(i**2)


                    :

except
                    print(“There
                    is an
                    error”) is
                    an error
There
```

Problem 2:

Handle the exception thrown by the code below by using `try` and `except` blocks. Then use a `finally` block to print 'All Done:

```
In[2]: x = 5

       Y = 0

       Z = x/y
```

Problem 3: Write a function that asks for an integer and prints the square of it. Use a `while` loop with a `try`, `except`, `else` block to account for incorrect inputs.

```
In[3]: def ask():

          While True:

              Try:

                 n = int(input(“Enter a number”))

              except:

                   print(“Please try again! \n”)

                   continue
```

```
            else:

            break:

        print("Your number squared is:")

        print(n**2)

In[4]: ask()

Enter a number 10

Your number squared is:

2500
```

-the break statement terminates the loop and resumes execution to the next statement; remember that the break statement can be used in while loops and for loops -else means that its going to occur each time but finally enters continuely
-while true is depended on break

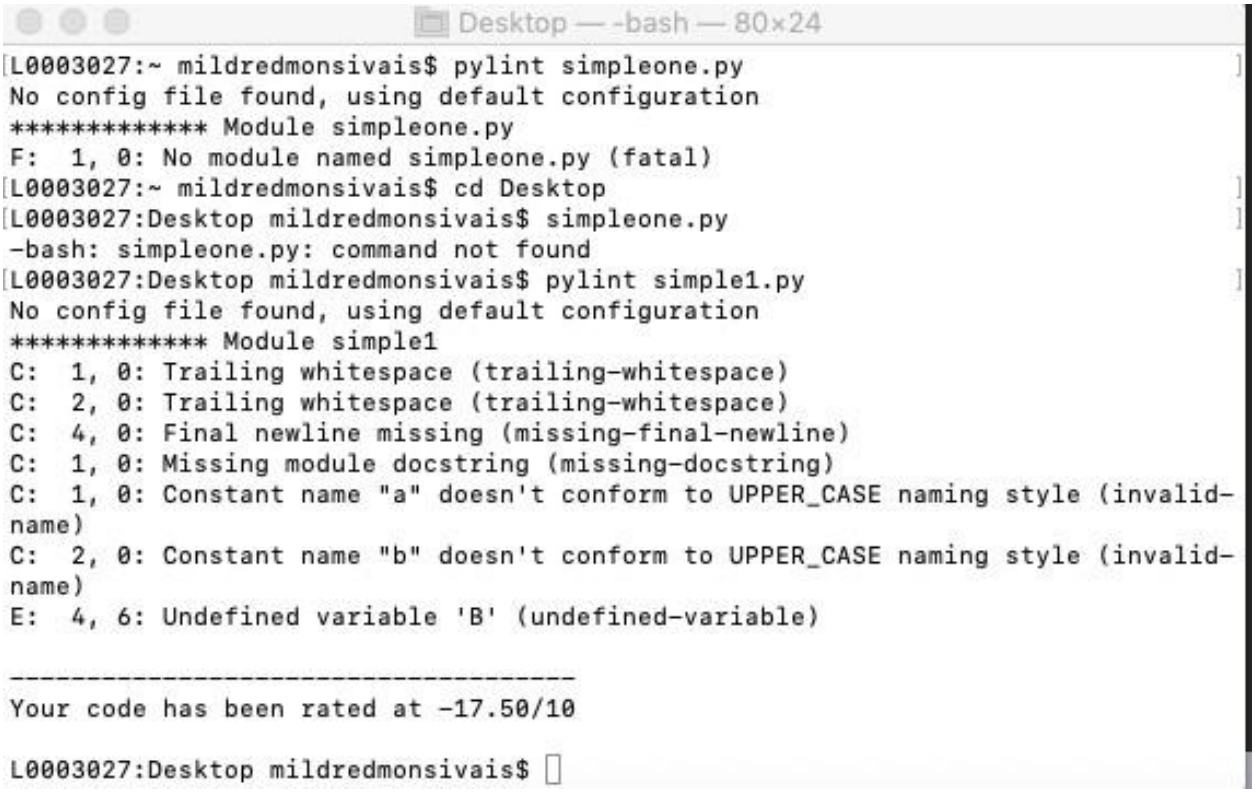
pylint overview

unit testing:part 1

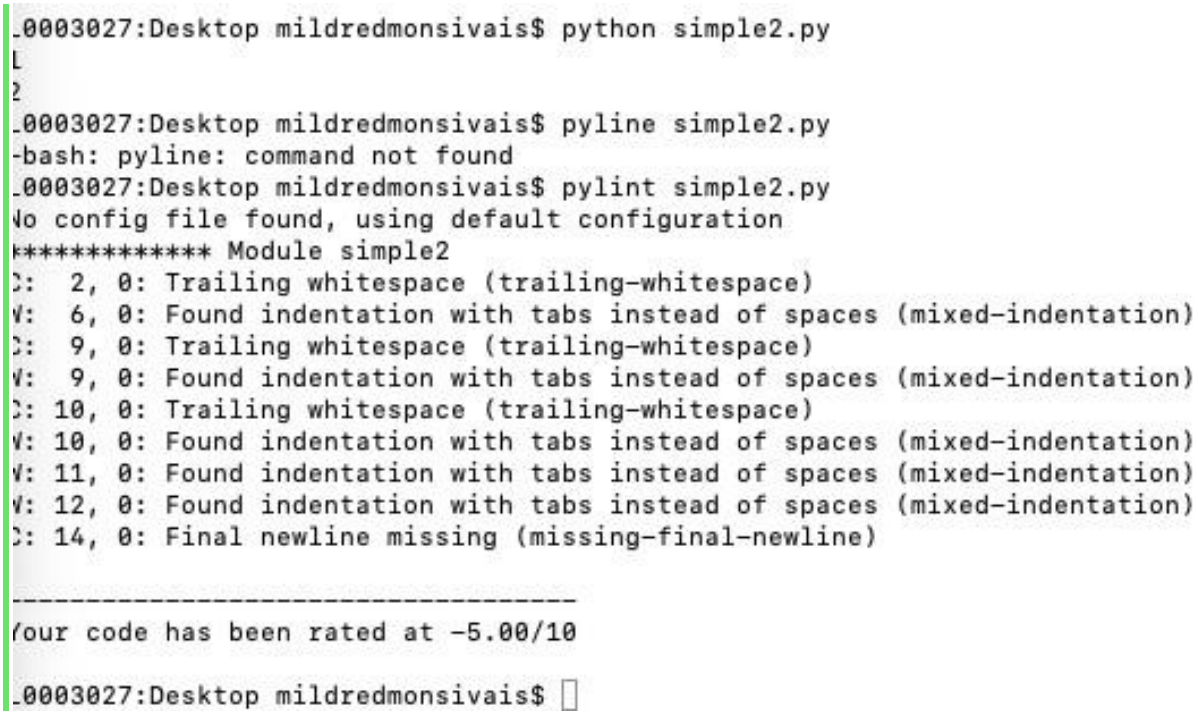
As beginning to expand to larger multi-file projects it becomes important to have tests in place, and this way you can make changes/ update any test fils to make sure code still runs as excepted
Two testing tools focus on:

1)Pylint: this is a library that looks at your code and reposts back possible issues
2)unittest: this is a built in library that will allow to test your own programs and check that your desired outputs are successful.
Going to write the code under .py scripts in sublime Step
by step what is occurring:
1.type in code in sublime text
2. save the code under your desktop
3. open terminal make sure your at your desktop(use cd Desktop to get there)
4. type in pyline _____(your file name that you saved on sublime text)

*on your terminal its going to give you the rating of your code as viewed below:



*Tells you the error you got on your code like line E: tells you an undefined E variable



Reran a different code and ended up getting -5.00/ 10 for the simple2.py code, there was no issue runninng the code but pylint was giving feedback on the format issue. More specifically spacing issue , like indentation.

Pylint is usually used when you have a large set of code or when your working with a group that wants specific styling methods not really useful for a single rating

unit testing: part 2

-unit test always you to write your own program and the goal is to send the specific data to your program and analyze the return results with the end goal to see if it sends you the specific result.
-since you are going to write your own program your going to have to use object oriented programing. When writing test function is always best to start simple then get more complex

-capitalize actually capitalizes the first word of the string or you can run.
You have a script you can call func or classes inside the testing scripts your doing to import any scripts you've been working on and create a class you've inherited and set whatever variables you need and then insert equal and to the expected result
-be aware of the self.assertEqual script and just to start off with create the assert Equal statement

Decorators: allow you to "decorate" a function, For
example:
def simple_func():
#what if you want to add more stuff ?
You have two options to "decorate" this function:
1.add that extra code(functionality) to your old function
2.Create a brand new function that contains the old code and then add new code to that. First you have to understand what a function is: a function returns a value based on the given argument / statement

- Functions need some information about the environment in which it has been called. The function consist of variables which are called parameters. From using the parameters its possible to use all kind of objects from "outside" inside a function. How the syntax is used to pass the parameters of the function depends on the programming language.

The idea of decoration is pretty abstract in practice and you have to go through steps of manually building out a decorator in order to show the @ operator is doing.

```
In[1]: def func():

           return 1

In[2]: func()

Out[2]: 1

In [3]: func

Out[3]: <function __main__.func()>
```

Basic review that func without parenthesis returns what you currently have doesn't actually return the argument of the function

```
In [4]: def hello():

            Return "Hello"

In[5]: hello()

Out[5]: 'Hello!'

In[6]: hello

Out[6]: <function __main__.hello()>
In[7]: greet = hello

In [8]: greet()

Out[8]: 'Hello!'
```

Python Generator

Generator are functions that allow us to write a function that can send back a value and then resume to pick up where it left off. In other words they are function that are can be paused and the resumed very quickly. Generator allow us to generate a sequence of values over time. The main difference in syntax will be the use of a yield statement. Generator are a function that is compiled that become an object that supports an iteration protocol. Meaning that generators don't actually return a value and then exit instead they automatically suspend and then resume their execution and state around the last point of value that is being generated. The advantage is that the generator will compute one value then waits for the next value until its called for. For

example lets say that you want to generate value from 1-100 a for loop is going to start from 1,2,3,4,5,6,7..etc while a generator slows picks off a number from memory.

- A generator is a simple way to create iterators(any type that can be used in a for loop.

Python Debugger

Python built in debugger and it includes features to allow to do do step by step and it's a super powerful tool to have

Lecture Regular Expression:

Datetime: