Learn Python for Students
- Setup of Developer environment
- Getting Started with Python Code
- Introduction to setting up your machine to write Python Code
- Mac OS install and Setup of Python
- Windows OS install and Setup of Python
- Creating a Python file in your editor
- Commenting in Python Code
- Variables -
- Python Data Types
- User Input
- Exercise Create a simple Calculator
- Favorite Number Checker Mini App
- Conditions and Logic If .. Else Statements
- Bouncer Exercise
- Loops while loop
- For Loop for lists and dictionary items
- Exercise Number Guessing Game
- Functions core part of programming
- Lambda
- Function scope
- Python built in Methods
- Python Modules
- Import Modules Python
- String Functions Python

Projects with Python
- Create a countdown timer
- Dice game in Python
- Rock Paper Scissors Game

Learn Python for Students

How to get setup and ready to code using Python, setup your development environment.

This is for all students who aspire to be a coder in the future and start building smart python apps. No prerequisite required as long as you have a desktop or a laptop to practice the projects.

Introduction to coding with Python designed for students. Equipped with several Python projects and loaded with code examples.

Setup of Developer environment

Python is perfect for both small and large scale projects. Designed to help programmers write clear and logical code made to be human readable. Use of whitespace and indentations to separate blocks of code. Code that is written is easy to read because of the use of indentation and object oriented approach.

Common uses for Python include, task automation, data analysis and visualization and development web applications. Due to the ease of getting started with Python it also gets commonly adopted by many non-programmers to help with tasks.
Python code runs using an interpreter, and there are two major versions of Python. Python version 2 is backwards compatible to previous versions. In 2008 Python 3 came along, the codebase was overhauled and updated. Which meant that version 3 was no longer compatible with earlier versions of Python. Package libraries written in 2 are not compatible with Python 3.

One of the most popular programming languages. Small learning curve, you can get started with Python quickly. Powerful and straightforward.

Most PCs and Mac will come with Python installed already. You will also need an editor to write the code and a terminal to run the code. The editor that I will be using within this course is Visual Studio Code https://code.visualstudio.com/ It comes with a built-in terminal so that you can write the code and also run the code in the same application.

Getting Started with Python Code

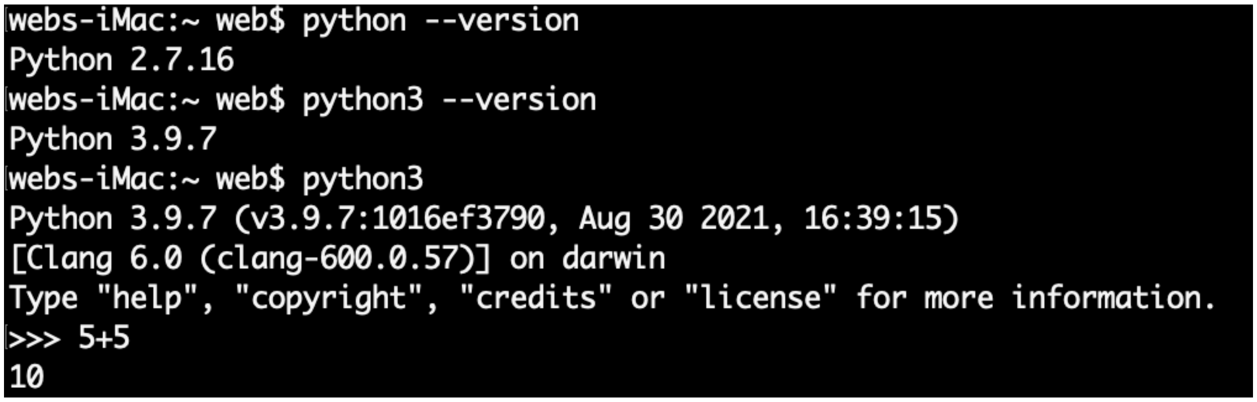
Walk through on the first steps to set up your computer to be ready to write Python code. How to install Python on a Mac and Windows Machine. How to set up and prepare your code editor for writing Python Code.

Introduction to setting up your machine to write Python Code

- Download and install Python3 latest version from https://www.python.org/downloads/
- Download and install code editor https://code.visualstudio.com/

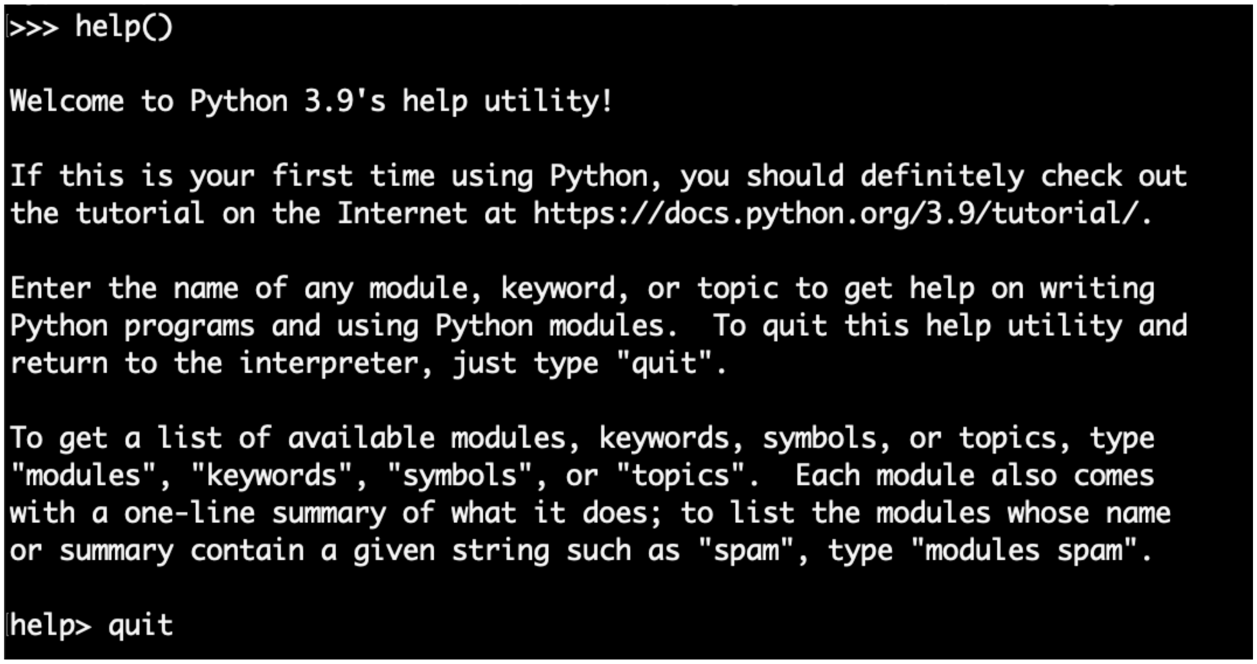
Mac OS install and Setup of Python

- open terminal with CMD+Spacebar for the spotlight then type Terminal
- Once terminal is open type python --version which will show the default python install typically Version 2
- Type python3 --version which will show the version of Python 3 installed
- Type python3 to open shell interpreter for Python code
- Type 5+5 and note the result



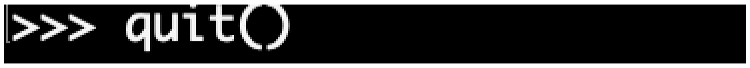
```
webs-iMac:~ web$ python --version
Python 2.7.16
webs-iMac:~ web$ python3 --version
Python 3.9.7
webs-iMac:~ web$ python3
Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 5+5
10
```

- type help() to open the Python help utility
- Enter quit anytime to exit the help utility



```
>>> help()

Welcome to Python 3.9's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.9/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> quit
```

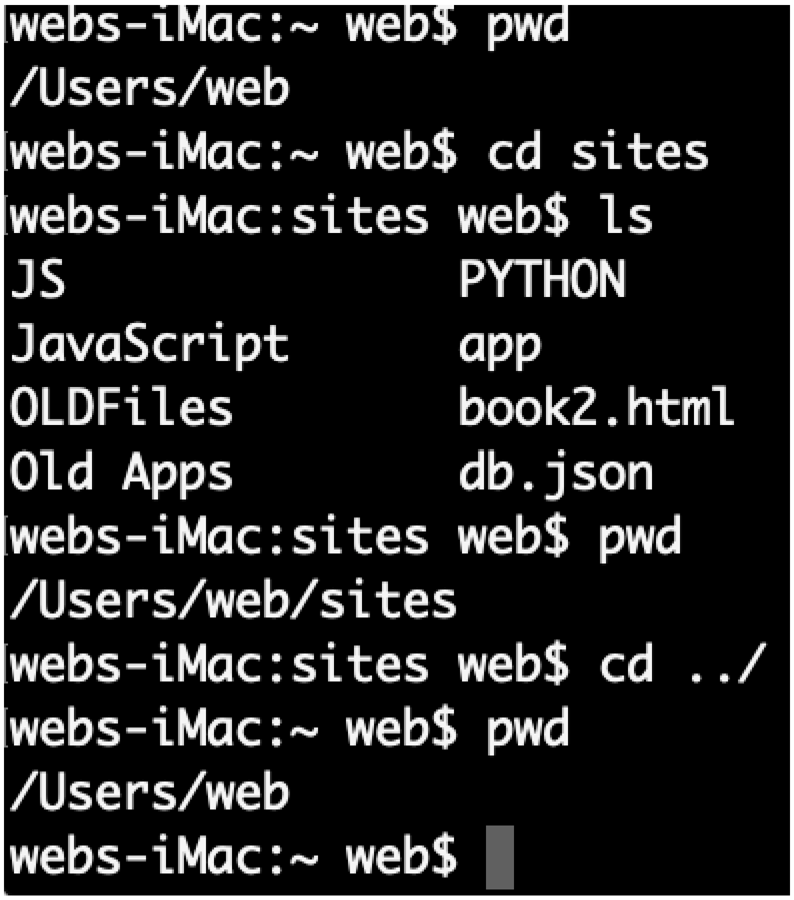
- to exit the shell interpreter type either quit() or exit() this will bring you back to the terminal command prompt



```
>>> quit()
```

Simple Command Prompt Commands on a MAC

- **pwd** shows current directory path
- **cd** changes to a directory within the current folder
- **ls** lists all files within the current directory
- **cs ../** moves down one level from the current directory

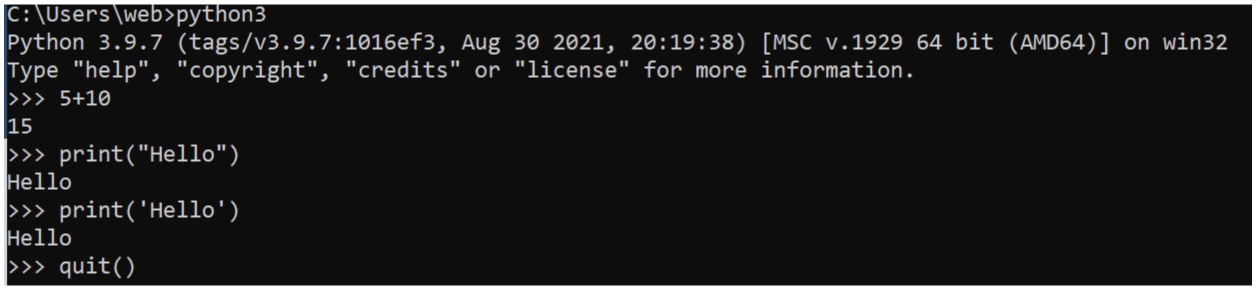


```
webs-iMac:~ web$ pwd
/Users/web
webs-iMac:~ web$ cd sites
webs-iMac:sites web$ ls
JS                  PYTHON
JavaScript          app
OLDFiles            book2.html
Old Apps            db.json
webs-iMac:sites web$ pwd
/Users/web/sites
webs-iMac:sites web$ cd ../
webs-iMac:~ web$ pwd
/Users/web
webs-iMac:~ web$
```

Windows OS install and Setup of Python

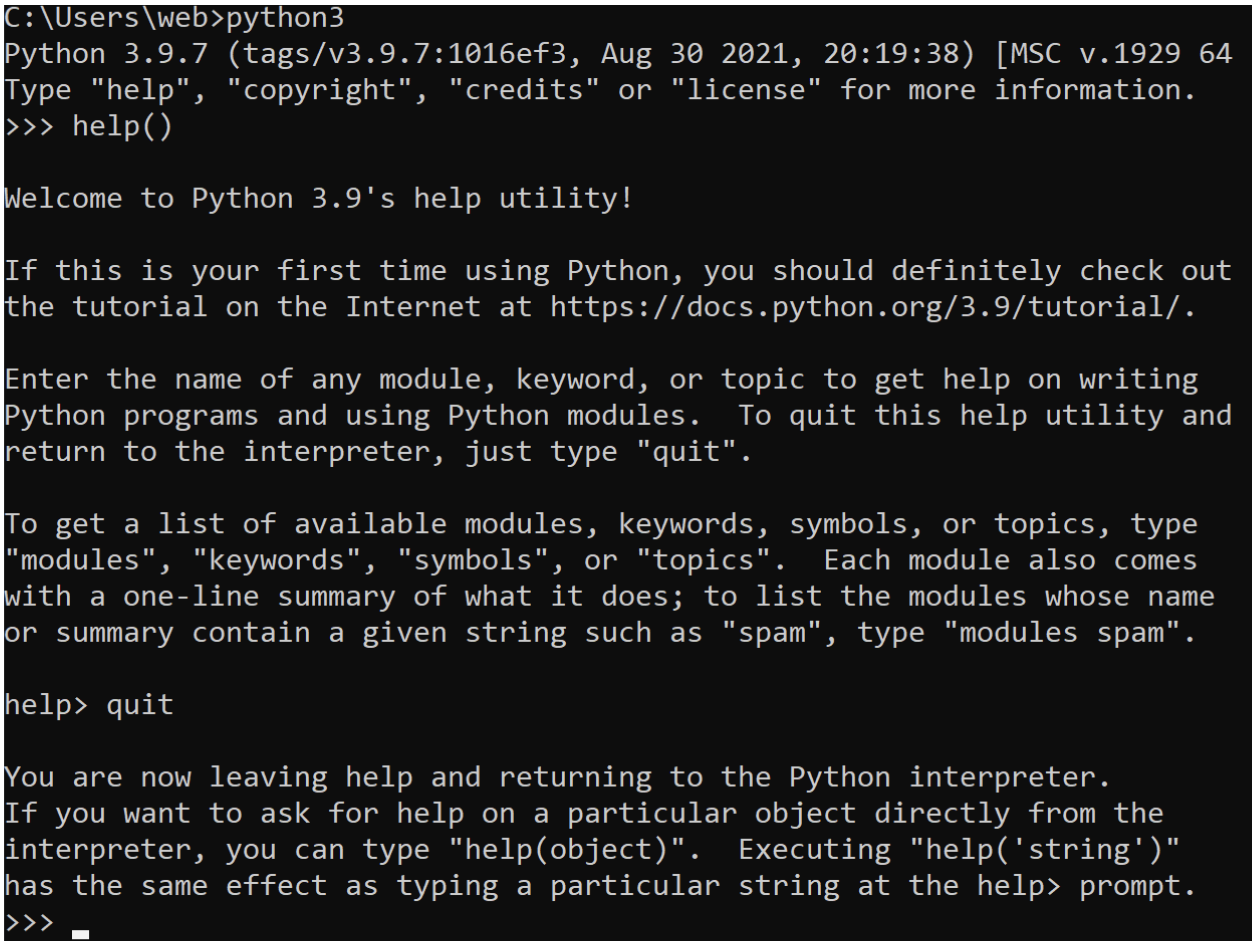
- open search and type command prompt
- Once terminal is open type python --version which will show the default python install typically Version 2
- Type python3 --version which will show the version of Python 3 installed

- Type python3 to open shell interpreter for Python code
- Type 5+10 and note the result
- to exit the shell interpreter type either quit() or exit() this will bring you back to the terminal command prompt



```
C:\Users\web>python3
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 5+10
15
>>> print("Hello")
Hello
>>> print('Hello')
Hello
>>> quit()
```

- type help() to open the Python help utility
- Enter quit anytime to exit the help utility

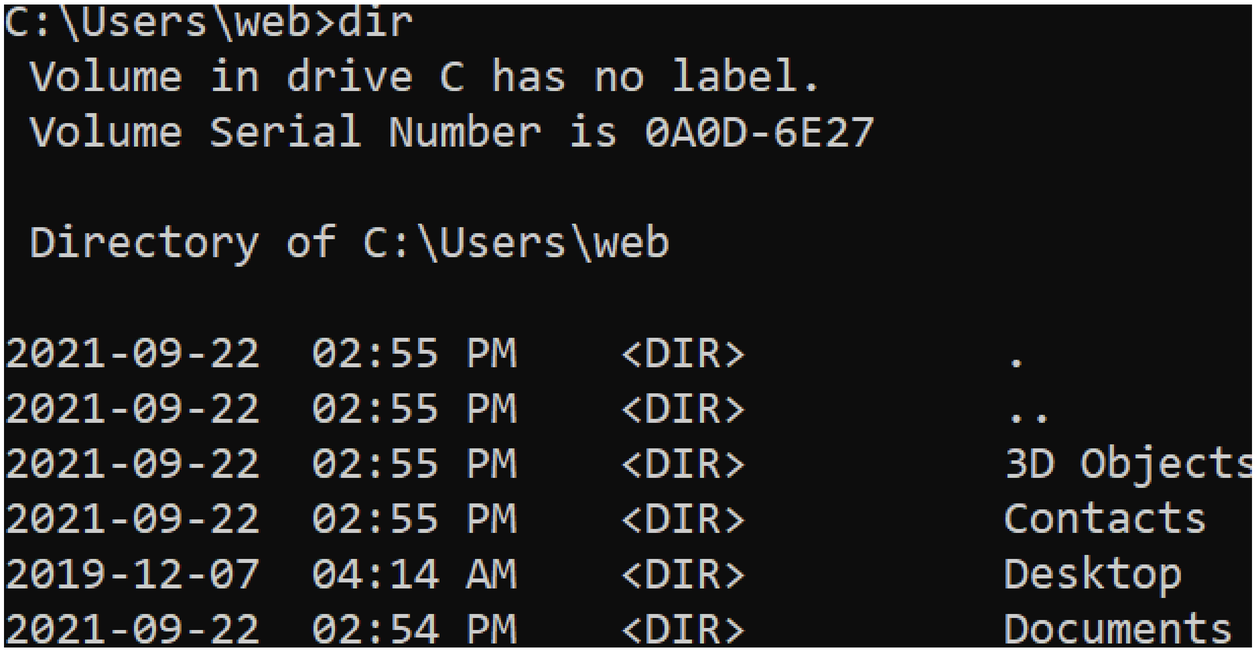


```
C:\Users\web>python3
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64
Type "help", "copyright", "credits" or "license" for more information.
>>> help()

Welcome to Python 3.9's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.9/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> quit

You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>> _
```

Simple Command Prompt Commands on a Windows PC

- **cd** changes to a directory within the current folder
- **dir** lists all files within the current directory

- **cs ../** moves down one level from the current directory

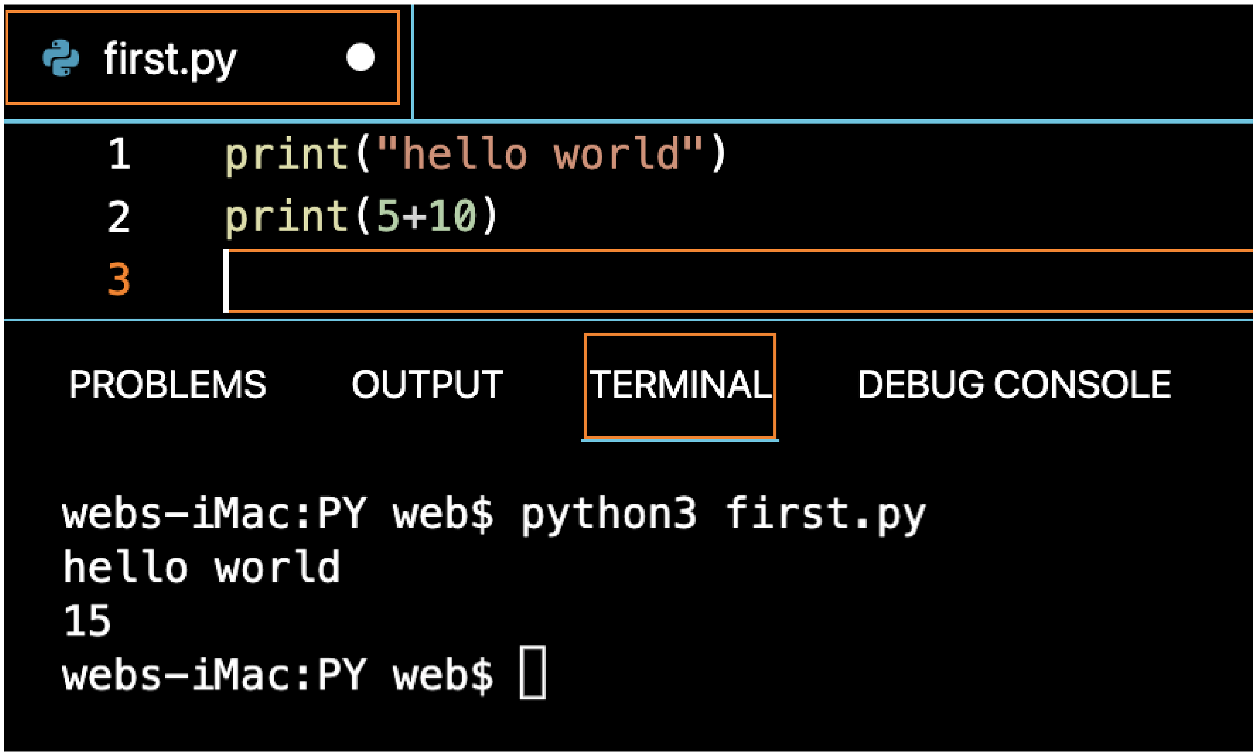


```
C:\Users\web>dir
 Volume in drive C has no label.
 Volume Serial Number is 0A0D-6E27

 Directory of C:\Users\web

2021-09-22  02:55 PM    <DIR>          .
2021-09-22  02:55 PM    <DIR>          ..
2021-09-22  02:55 PM    <DIR>          3D Objects
2021-09-22  02:55 PM    <DIR>          Contacts
2019-12-07  04:14 AM    <DIR>          Desktop
2021-09-22  02:54 PM    <DIR>          Documents
```

Creating a Python file in your editor

Using print create a string value that you want displayed on the screen. print(“hello”)
Save the file as filename.py
To run the python file you can type **python filename.py**
also to run in python3 you suffix the word python with 3 **python3 filename.py**

```
print("hello world")
print(5+10)
```



```python
print("hello world")
print(5+10)

```

```
webs-iMac:PY web$ python3 first.py
hello world
15
webs-iMac:PY web$ 
```

Python uses indentation and how it works is that the indentation indicates a block of code.   **In Python indentation is very important.**

Commenting in Python Code

Comments can explain code, making it more readable and also great for testing when you want lines of code not to execute.   You can also provide information that can be used later and referred to at a later date by yourself or other developers.

```python
print("hello world")
#print(5+10)
#print(5+10)
```

```
"""
print(5+10)
print(5+10)
"""
print(5+10)
print(5+10) #adding
```

Variables -

One of the most important concepts of programming. Variables are at the core of programming. They are containers that can hold information, which can then be easily referenced within the code and returned when needed. Think of a variable as a way to reference something that might change, like a box where you can put items into it and return back the contents of that box by selecting the box label.

- Strings can be single or double quote
- Variable names are case-sensitive

Rules -

1. Must start with letter or _ underscore
2. Can use a-zA-Z0-9_ alpha numeric characters within the variable name
3. Case sensitive
4. Use Camel case to indicate words in variable name

Get type of data
type("variable")

How to assign multiple values in variables by comma separation of variable names and assigning to comma separated values. Comma separate for multiple variables in one line
You can also assign the same value to all variable names using the = sign
Using a collection you can assign those values to a number of variables within one statement

```
# Start with letter or _ cannot start with a number
# a-Z0-9_
a = 55
```

```
message = "Hello World"
print(a+b)
print(5+10)
print(message)
a = str(a)
print(type(a))
a = int(a)
print(type(a))
print(a)
str1 = "hello world"
str2 = 'hello'
str3 = "hello' world"
Str1 = "hello"
print(str1)
print(Str1)
outputMessage = "hello"
a, b, c = 5 , 10 , 15
print(c)
d = e = f = 50
print(e)
data = ["Laurence","John","Linda"]
g,h,i = data
print(h)
```

Python Code Operators

```
a = 5
b = 10
a += 3 # a = a + 3
print(a+b)
print(b-6)
print(4*5)
print(50/10)
```

print(53%10)
print(5 == 2)
print(5 > 2 and 10 < 5)
print(5 > 2 or 10 < 5)
print(not(50 > 5))

Python Data Types

More Data type within Python
Boolean can be either True or False - must be capitalized
Variables can also store multiple items in various collections.
Four built in data types to store multiple items are Tuple, Set, List and Dictionary.

```
a = True
#print(type(a))
b = False
#print(type(b))
#print(5>10)
#List Type
testList = [50,"100",True,50,"50"]
#print(testList[1])
#print(len(testList))
#print(type(testList))
#Tuple Type
testTuple = (50,"100",True,50,"50")
#print(testTuple)
#print(len(testTuple))
#print(type(testTuple))
testSet = {50,"100",True,50,"50",50,100,100,"a"}
#print(testSet)
#print(len(testSet))
#print(type(testSet))
```

```
testDic =
{"first":50,"second":"100","third":True,"a":50,"a":"LAST"}
#No Dups Ordered
print(testDic)
print(len(testDic))
print(type(testDic))
```

Python List

Lists - created with square brackets testList = [50,"100",True,50,"50"]

- have defined order and the order does not change when in the list
- You can add new items and remove items from the list. The list can be changed
- Duplicate values are allowed within the list.
- Use index value to identify and retrieve values from within the List. Indexes are zero based, values start at zero.
- Use the len() function to determine the number of items within the list.

Get Values -
tempList = [“one”,”two”,”three”,”four”]]
Use the index value example 1 to return the value in a list.
tempList[1] will return “two”
tempList[1] = “NEW” - will assign a new value of NEW to index item 1
tempList[-1] will return “four” last item in the list
tempList[2:3] will return ,”three”,”four”] - if no end value is provided will return all the items until the end.
tempList[:3] will return all expect the item with index 3
tempList.index(“two”) - gets the index value of the item
check if in list print("two" in testList) will return a boolean result
tempList.append(“last”)
tempList.insert(1,“last”)
tempList.remove(“last”)
tempList.pop(1) no index removes the last item in the list
del tempList[0]
del tempList - removes the entire list
tempList.clear() - empties list
tempList.sort() list sorts ascending and alphanumerically - change sort order
.sort(reverse = True)

copyList = tempList.copy()

```
testList = ["Laurence","World","Hello","World"]
#a,b,c,d,e,f,g = testList
testList.insert(3,"Svekis")
#testList.append("End")
#testList.remove(100)
testList[0] = "New"
print(testList)
#val = testList.pop(3)
#del testList[1]
#del testList
#testList.clear()
testList.sort()
print(testList)
testList.reverse()
copyList = testList.copy()
copyList.append("NEW")
print(copyList)
print(testList)
#print(testList[2])
#print(c)
#print(len(testList))
#print(testList[-2])
#print(testList[1:2])
#print(testList[:3])
#print(testList.index(100))
#val = ("ASvekis" in testList)

#print(val)
```

Python Tuple

Tuple - created using rounded brackets testTuple = (50,"100",True,50,"50")

- Does allow duplicates since just like lists they are indexed.
- ordered is defined and does not change
- **cannot change or remove items from the tuple after it's created**
- Zero based and can use functions like len() to get the number of items
- Allowed data types are strings, integers, and booleans

Get Values -
Using the index just like lists.

```
testTuple =
("Hello","Laurence","Svekis",100,True,"Svekis","Svekis")
val = len(testTuple)
print(testTuple)
#testTuple[2] = "NEW"
val = testTuple[2]
print(val)
```

Python Set

Sets - created using curly brackets testSet = {50,"100",True,50,"50",50,100,100,"a"}

- no order they are unordered. They have no set order that they appear in.
- values cannot be changed after its set although you can add to it and remove from it
- Allowed data types are strings, integers, and booleans

Get Values
check if it exists using (“value” in testSet)
testSet.add(“Last”)
testSet.remove(“Last”)
testSet.pop() removes random item

```
testSet = {"Svekis",100,True,"Svekis",100,"Laurence"}
testSet.add("New")
testSet.remove(100)
testSet.pop()
val = ("Svekis" in testSet)
print(testSet)
print(val)
```

Dictionary - named key value pairs using curly brackets testDic = {"first":50,"second":"100","third":True,"a":50,"a":"LAST"}

- order does not change as they are not indexed and accessed using the key value
- Can be changed and updated as needed referencing the key value. Add and remove new items into the dictionary
- Cannot have more than one item using the key name
- Dictionaries can contain nested dictionaries in a child parent format

Get Values
testDic[“first”] or to change use testDic[“first”]=
testDic.keys() - gets all the keys
testDic[“newKey”] = “to add new items”
testDic.pop(“first”) to remove items
testDic.clear to clear all items

```
testDic = {"first":"Hello
World","key":{"one":"test","two":"test2"},"num":100,"num":200,"n
um":300,"boo":True}
#print(testDic)
testDic["first"] = "test value"
val = testDic["first"]
val = testDic["num"]
val = testDic["key"]["two"]
testDic["test"] = 10000
#testDic.pop("key")
#testDic.clear()
```

```
val = testDic.keys()
val = testDic.values()

test2 = {
   "first" : 100,
   "second" : "Values",
   "third" : True
}
print(test2)


#print(testDic)
print(val)
#print(testDic["test"])
```

Get User Input in terminal assign to a Variable

Within your code you can request the user provide an input. The input will be a string data type by default. You can assign the response of the user input to a variable that can be used within your code.
String methods can be helpful in checking to see if they can be converted to an integer. You can use a condition to check and run a block of code depending on the boolean value of a variable.

The isnumeric() method returns True if all the characters in the string value are numeric (0-9), otherwise it will return a value of False.

Exercise Create a simple Calculator

Using the input, ask the user for two numbers, take the numbers and convert them to integers so that you can add them together. Create an output message to the user with the total and the equation for the two numbers added together.

```
num1 = int(input("First Number : "))
```

```
num2 = int(input("Second Number : "))
total = num1 + num2
cal = str(num1) + "+" + str(num2) + "=" + str(total)
message = "Your Total is " + str(total)
print(cal)
print(message)
```

Favorite Number Checker Mini App

Create a mini application from the content of this section the user their favourite number and then check if its a valid number that can be converted into a integrer and used within your code.

```
val = input("What is your favorite number 1-9: ")
boo = val.isnumeric()
print(boo)
message = "Sorry not a number "
if(boo):
   num = int(val)
   print(type(num))
   message = "Great your number is " + val
print(message)
```

Conditions and Logic If .. Else Statements

Conditions allow us to apply logic into our Python code.
Indentation is important as it help Python understand the block of code that needs to run if the condition is true.
Exploring shorthand methods that can be written in one statement

```
boo = False
a = 500
b = 1500
#print(a < b)
```

```
if boo: print("boo is True")
print("Hello") if boo else print("world")
print("Equal") if a==b else print("B bigger") if b > a else
print("A must be bigger")

if a < b:
   a = 1000
   print(a)
   print("True")
elif a > b:
   print("a is greater")
else:
   print("a and b must be the same equal")
```

Bouncer Exercise Project - Code bouncer allowed in or not?

Create an application that will ask the user their age. Depending on the age, first check if the response is a number. If its not a number reject them and provide a message back. If its a number then check to see if the value is allowed in and can drink which is 21+, if they are 18-20 they can come in but not drink and lastly if they are 17 or less they should not be allowed in. Create the application to follow these rules!

```
age = input("How old are you? ")
boo = age.isnumeric()
if boo:
   print("Thank you I am checking if you can come in...")
   val = int(age)
   if(val >= 21):
       print("Great you are allowed in and can drink")
   elif(val >= 18):
       print("Come in but can't drink")
   else:
       print("You are not allow in.  Not old enough")
else:
   print("We need your age!")
```

Loops while loop

While loop allows us to run blocks of code multiple times. There are options using the keyword continue to skip an iteration as well as using the keyboard break to leave and stop the loop. In addition to the loop there is also an option to provide an alternative output once the loop condition is no longer true and run an else block of code.

```
val = 0
while val <= 10:
   print(val)
   val += 1
   if val == 2:
       continue
   #print(".")
   if val == 15:
       break
else:
   print("val is no longer less than 11")
```

For Loop for lists and dictionary items

Using a for loop can output the contents of a list or a dictionary.

```
testList = ["Laurence","Hello","World",55,100,True]
for item in testList :
   print(item)
myStr = "Hello World"
```

```
for letter in myStr :
   if letter == "l":
       continue
   if letter == "x":
       break
   print(letter)
else:
   print("Word done!")

testDic = {
   "first" : "Laurence",
   "last" : "Svekis",
   "allowed" : "True"
}
for key in testDic:
   print(key + ":" + testDic[key])
for key in testDic:
   print(testDic[key])
for val in testDic.values():
   print(val)
for val in testDic.keys():
   print(val)
for key,val in testDic.items():
   print(key,val)
```

Exercise Number Guessing Game

Using a while loop create a game that allows the user to make a guess at the hidden value of a number.

1. Start off create secret number in a variable
2. Set a variable that will limit the number of guesses
3. Create a variable to hold the number of guesses made by the user.

4. Setup a while loop that compares guess value to the secret number
5. Print provide the user how many guesses they have left
6. Capture the input from the users guess as a integer
7. If the users guess is correct set the final message as a WIN
8. Provide feedback to the user if their guess incorrect guess was too high or too low
9. Increment the counter and check if the count is at the max guesses, if it is break from the while loop
10. Output the WIN or LOSE message to the player

```python
num = 10
limit = 6
cnt = 0
playGame = False
guess = 0
while guess != num :
   print("You have "+str(limit - cnt)+" left")
   guessFirst = input("Enter a Number : ")
   if(guessFirst.isnumeric()):
       guess = int(guessFirst)
       if guess == num : playGame = True
       if(guess > num):
           print("Wrong too high")
       elif(guess < num):
           print("wrong too low")
   cnt+=1
   if((limit-cnt) == 0 ):
       break
else:
   playGame = True

if(playGame):
   print("You Got it!")
else:
    print("You ran out of guesses it was:"+str(num))
```

Functions core part of programming

You can reuse blocks of code invoking them anytime within the code to run the function block of code. Function can have arguments that are values passed and assigned to the variable names used within the function arguments.   These values can then be used within the function. You can also return values with functions, those values can then be assigned to variables and used within your Python Code.

```python
def fun1(val):
    print(val*val)
    #print("Hello")

fun1(1)
fun1(5)
fun1(6)
fun1(7)

def fun2(first,last):
    print("Hi, " + first + " " + last)
    return first + " " + last

fun2("Laurence","Svekis")
fun2("Linda","Jones")
fun2("Mike","Smith")
myName = fun2("Laurence","Svekis")
print(myName)

def fun3(val1,val2):
    total = val1 + val2
    print(str(val1) + " + " + str(val2) + " = " + str(total))
    return total
```

```python
num1 = fun3(6,7)
num2 = fun3(126,2317)
print(num1, num2)
```

Lambda

Shorter anonymous functions written as one statement

```python
val = lambda a : a * 5
num1 = val(10)
print(num1)

val1 = lambda a,b : a * b
num2 = val1(10,20)
print(num2)

print(val1(3,9))
```

Function scope

Values can be accessed locally with the block of code that is assigning the values. Each block of code has its own scope, the main block of code is called the global scope. You can use variables from the parent block scope but you cannot update the variable values within the child scope. There are keywords to access and update the global scope, you can use global to reference the variable you want to use. That opens the variable up to be updated within the child scope. You can also use the keyword nonlocal to access the parent scope variables to update them within a child local scope.

```python
a = "test"
b = 0
def fun1(val):
   global b
   b = b + 1
   #a = "hello"
   def fun2():
       global b
       nonlocal val
       b += 5
       val += 1000
       print(val)
   fun2()
   print(a)

fun1(100)
print(b)
```

Python built in Methods

Built in functions within Python
https://docs.python.org/3/library/functions.html

```
print(5 * 8)
val1 = min(4,6,4354,345,23,1000,54,6,433,6665,3434)
print(val1)
val2 = max(4,6,4354,345,23,1000,54,6,433,6665,3434)
print(val2)

val3 = round(4.6)
print(val3)

val4 = round(4.1)
print(val4)

val5 = round(4.5)
print(val5)
```

Python Modules

You can use functions and data from other python files in your main file. Allows you to import functionality referencing the module

```
import mod1
import data1
import data2 as nn
from mod1 import myName

mod1.welcome(myName["first"],myName["last"])
mod1.welcome(data1.myName["first"],data1.myName["last"])
mod1.welcome(nn.myName["first"],nn.myName["last"])
total = mod1.adder(5,8)
print(total)
```

Mod1.py

```
def welcome(first,last):
   fullName = first + " " + last
   print("Welcome to my page, " + fullName)



def adder(a,b):
   return a + b

myName = {
   "first" : "John",
   "last" : "Vekis",
   "status" : True
}
```

data1.py
```
myName = {
   "first" : "Laurence",
   "last" : "Svekis",
   "status" : True
}
```

Import Modules Python

datetime is a stand module that you can use to get date values

```
#import datetime
from datetime import datetime
today = datetime.now()
days = ["Mon","Tues","Wed","Thurs","Fri","Sat","Sun"]
print(today)
print(today.day)
print(today.month)
print(today.weekday())
print(days[today.weekday()])
```

String Functions Python

We can update and manipulate the string values with useful built in string methods that can be applied to strings

```
name = "     Laurence Svekis    "
val = len(name)
findThis = "SDe"
val = (findThis in name)
val = (findThis not in name)
val = name[0:5]
val = name[:5]
val = name[6:]
val = name.upper()
val = name.lower()
val = name.strip()
val = name.replace("e","").strip().upper()
val = name.strip().split(" ")
print(name)
val1 = '----'.join(val)
print(val)
print(val1)
```

Projects with Python

Create a countdown timer

1. create a function that will decrease a val
2. Using Modulus and Floor Division calculate the minutes and seconds
3. Create a string output value using format to structure it as minutes and seconds
4. Using the time module import the sleep method slowing the output with a 1 second delay
5. Subtract from the total seconds
6. Once the while loop completes print the time is up and invoke the start function again
7. Create a starting function that gets the users inputs for the countdown timer in total seconds. Check if the input is numeric if it is send the total second value to the countdown function
8.

```
import time
def test():
   val = 0
   while val < 10:
       val+=1
       time.sleep(1)
       print(val)

def countdown(t):
   while t:
       mins = t // 60
       secs = t % 60
       output = "Your Time Left is
{:02d}:{:02d}".format(mins,secs)
       print(output)
       time.sleep(1)
       t -= 1
   print("Time is up!!")
   start()
def start():
   t = input("Enter the number of seconds : ")
   if t.isnumeric() :
       countdown(int(t))
   else:
       print("Was not a Number")
       start()
start()
```

Dice game in Python

Play against the computer, role the dice get a random value and see who scores more. The highest role wins the game. Perfect game to practice and learn more about using random in game logic, and applying conditions to check for a winner

1. Import the random module
2. setup default variables for score and set the values of the roles from min to max.

3. Create a function to house the game coding
4. Setup a while loop that will run the block of game code
5. Game code generates random values for both the player and computer.
6. Apply conditions to check who wins
7. output the feedback and results to the player. Keeping score of the rounds.
8. Allow player to exit and end game

```python
import random
min = 1
max = 6
computerScore = 0
playerScore = 0
inPlay = True

x = 20
while x < 10:
   test = random.randint(min,max)
   print(test)
   x+=1

def gamePlay():
   global inPlay
   global computerScore
   global playerScore
   while inPlay:
       player = random.randint(min,max)
       computer = random.randint(min,max)
       print(f"You Got {player} vs {computer}")
       if(player == computer):
           print("Tie Game")
       elif (player > computer):
           print("Player Wins")
           playerScore += 1
       elif (player < computer):
            print("Computer Wins")
            computerScore += 1
        inPlay = input("Roll Again ? ")
        if inPlay == "exit" :
            break
gamePlay()
print("Game Over")
print(f"Computer Score : {computerScore } vs Player Score :
{playerScore }")
```

Rock Paper Scissors Game

Rock paper scissors, played between the player and the computer. Setup the game make a selection and see who wins. Rock beats Scissors and crushes them, Paper covers the rock and wins, Scissors can cut up the paper to win.

1. import the random module
2. setup the default values and array
3. create the gameplay function
4. create a loop within the gameplay function
5. Get the user selection and generate a random selection for the computer
6. Apply logic to see who wins
7. Let the player exit the game or go for another round.

```
import random
computerScore = 0
playerScore = 0
arr = ["Rock","Paper","Scissors"]
inPlay = True
def gamePlay():
   global inPlay
   global computerScore
   global playerScore
   while inPlay:
       player = input("Rock Paper or Scissors ? ").capitalize()
```

```
        computer = random.choice(arr).capitalize()
        print(f"You Selected : {player} vs Computer Selected :
{computer}")
        if player ==  computer :
            print("Tie Game")
        elif player == "Rock":
            if(computer == "Paper"):
                print("You Lose")
                computerScore+=1
            else:
                print("You win")
                playerScore += 1
        elif player == "Paper":
            if(computer == "Scissors"):
                print("You Lose")
                computerScore+=1
            else:
                print("You win")
                playerScore += 1
        elif player == "Scissors":
            if(computer == "Rock"):
                print("You Lose")
                computerScore+=1
            else:
                print("You win")
                playerScore += 1
        print(f"You ({playerScore}) vs
Computer({computerScore})")
        inPlay = input("Play Again ? ")
    print("Game Over")
    print(f"Your Score ({playerScore}) vs
Computer({computerScore})")
```

```
gamePlay()
```