**What is Python:**

* It’s a Programming Language. It’s Interpreted, Object-oriented, High-Level language.
* It supports Procedural as well as Object Oriented
* It’s also called General Purpose Language
* It’s older than Java;
* Easier than other languages; that’s why scientist choose it for AI, machine learning because they don’t want to waste much time to learn any language
* Author: Guido Van Rossum; why named ‘python’: author was a big fan of british comedy television show “Monty Python's Flying Circus”
* Python program is complied and interpreted
* Python source code gets complied by compiler -> converts into a byte code format -> then interpreter interprets the byte code -> converts into machine/binary code which is then executed by a machine
* Interpreter works on Python virtual machine (PVM) like JVM (java virtual machine)
* By that way, you can create a program in any machine and run it on another machine (platform independent)

**Where we can use Python:**

* Machine learning
* GUI development
* Web Development
* Software Development

**Installation:**

* Download from <https://www.python.org/downloads/> and install (it will install Interpreter)
* Install IDE for python i.e. Pycharm from <https://www.jetbrains.com/pycharm/>
* Python has it’s own IDLE to write simple programs. It gets installed when you install Python
* Python gets installed at C:\Users\DELL\AppData\Local\Programs\Python by default
* Note that PIP gets installed when you install Python. PIP is the standard package manager for python. It allows you to install and manage additional packages that are not part of python standard library.
* **For Selenium,**
* you just have to run below command in command prompt to get install Selenium for Python. That’s it. Selenium package will get injected into python folder.

C:/> **pip install selenium**

If you wanted to see installed selenium version and location, use below command

C:/> **pip show selenium**

If you wanted to upgrade installed selenium, use below command

C:/> **pip install -U selenium**

* PIP stands for either "Pip Installs Packages" or "Pip Installs Python
* When you create a Project in Pycharm IDE, by default it uses it’s own python environment (and not the one which we install at C:\Users\DELL\AppData\Local\Programs\Python)
* For each project, Pycharm creates it’s own python environment at project level. E.g. ‘F:\Python Programs’ is my project path, then you can see python.exe at “F:\Python Programs\venv\Scripts\python.exe”.
* To check this, click on pycharm version displayed on Pycharm IDE right bottom > Interpreter Settings
* Here, you will not see installed Selenium package since it gets installed in different location; not in the project.
* So, if you wanted to use the python environment where selenium package is installed then
* Python Interpreter Setting screen, click config button displayed next the default path > show All.
* You might see your python path; then you can select it.
* If you are not able to see your python path, then click + icon > Virtualenv Environment > Existing Environment > select or navigate your base python environment ("C:\Users\DELL\AppData\Local\Programs\Python\Python38-32\python.exe") > Select ‘make available to all projects’ > OK > Select it in Python Interpreter list > OK
* Now your base python environment has been set to all projects and selenium package will be displayed on Interpreter Settings screen.
* Now, whatever the package you will install from command prompt using PIP command, that package will be available for your project.
* Note, even though you don’t set base python environment for your project in pycharm, when you use selenium jargons, pycharm will show error and give suggestions to install it. By clicking on it, you can install selenium package in your pycharm python environment.

**Data Types:**

* You will get variables Data type using type() in python

>>>num = 2.5

>>>**type(**num**)**

🡺<class ‘float’>

* We have following data types:
* None
  + A variable with no value; it’s data type will be none
* Numeric
  + int
  + float
  + complex
  + bool (Boolean)

|  |  |  |
| --- | --- | --- |
| Operations | Output |  |
| a = 5.7 type(a) | <class 'float'> | gives datatype of a variable |
| b = **int(**a**)** print(b) | 5 | converts into INT |
| c = **float(**b**)** print( c) | 5.0 | converts into Float |
| d = 8 e = **complex(**b, d**)** print(e) | (5+8j) | converts into Complex |
| f = 7 > 5  print(f) | True |  |
| type( f ) | <class 'bool'> |  |
| **int(** True **)** | 1 |  |
| **int(** False **)** | 0 |  |
| g = ‘swara’  type( g ) | <class 'str'> | Gives same output for single char  i.e. for g = ‘s’ |

* Sequence
  + String
  + List
  + Tuple
  + Set
  + Range

|  |  |  |
| --- | --- | --- |
| Operations | Output |  |
| range(10) | range(0, 10) | tells the range from 0 to 10 |
| list( range(10) ) | [0,1,2,3,4,5,6,7,8,9] | prints the all elements in the range in a list format |
| list( range(3,10,2) ) | [3, 5, 7, 9] | prints elements in the range 3 to 10 having difference of 2 |
| type( range(10) ) | <class 'range'> |  |

* Dictionary (Map)

**Operators:**

* Arithmetic operators (+ , - , \*, / , %)
* Assignment operators ( =, +=, -=, \*=, /=, %=, //=, \*\*=, &=, |=, ^=, >>=, >>= )
* Relational/comparison operators (<, >, ==, <=, >=, !=) i.e. for comparing the objects
* Logical operators (and, or, not) i.e. to combine more conditions
* Unary operators ( - ) e.g. -7, -10 i.e. to make a negative values
* Identity operators (is, is not)
* Membership operators (in, not in)
* Bitwise operators (&, |, ^, ~, <<, >>). Details are given below

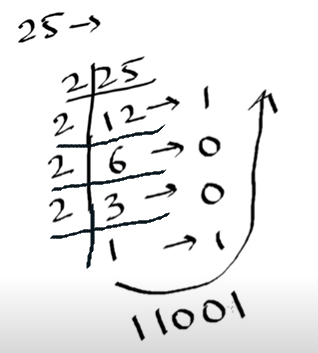
|  |  |  |
| --- | --- | --- |
| Operations | Output |  |
| 7 + 3 | 10 |  |
| 7 - 3 | 4 |  |
| 7 \* 3 | 21 |  |
| 7 \*\* 3 | 343 | (Exponent/power) 7 rest to 3 i.e. 7 \* 7 \* 7 |
| 7 / 3 | 2.33 |  |
| 7 // 3 | 2 | (Floor Division)  integer value of (7/3); skip decimals |
| 7 % 3 | 1 | (Modulus) gives remainder of (7/3) |
| 7 + 3 \* 5 | 105 |  |
| (7 + 3) \* 5 | 50 |  |
| X = 7  X \*= 3  Print(X) | 21 | i.e. X = X \* 3 |
| X = 7  X += 3  Print(X) | 10 | i.e. X = X + 3 |
| **a, b = 5, 6**  print(a)  print(b) | 5  6 | Assignment of multiple variables in one line |
| n = 7  print( **-**n ) | -7 | (-) sign is a **Unary** operator |
| 12 > 5 **and** 15 > 8 | True | Logical operator |
| 12 > 5 **and** 15 < 8 | False |  |
| 12 > 5 **or** 15 < 8 | True |  |
| **not** (12 > 5) | False |  |
| x **is** y | False | If x = 5, y = 7. Returns True if both variables are the same object |
| x **is not** y | True | Returns True if both variables are not the same object |
| 33  **in** p | True | If p = [22,33,44]. Returns True if a sequence with the specified value is present in the object |
| 77 **not in** p | True | Returns True if a sequence with the specified value is not present in the object |

**Bitwise Operators:**

* In Python, bitwise operators are used to perform bitwise calculations on integers.
* The integers are first converted into binary and then operations are performed on bit by bit, hence the name bitwise operators. Then the result is returned in decimal format.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operator | Description |  | Operations | Output |  |
| & | Bitwise AND | Returns 1 if both the bits are 1 else 0 | x & y | 12 | 0011 1100 = 60 = x 0000 1101 = 13 = y --------------- 0000 1100 = 12 |
| | | Bitwise OR | Returns 1 if one of the bit is 1 else 0 | x | y | 61 | 0011 1100 0000 1101 --------------- 0011 1101 |
| ~ | Bitwise NOT (complement) | Inverts all the bits | ~x | -61 | 0011 1100  -------------------  1100 0011 (2’s complement of -61)?? |
| ^ | Bitwise XOR | Sets each bit to 1 if only one of two bits is 1 | x ^ y | 49 | 0011 1100 0000 1101 --------------- 0011 0001 |
| >> | Bitwise Right Shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off | x>> | 7 | 0011 1100 ----------------- 0011 1**100 ==> 0000 0111 = 7** |
| << | Bitwise Left Shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off | x<< | 480 | 0011 1100 ----------------- 0011 1100**000 = 480** |

**Number system:**

* Binary
  + Base is 2 numbers (i.e. 0 & 1)
  + Starts with ‘0b’; to indicate it’s binary format
  + E.g. Binary of 25 is “0b11001”
  + 
  + We are converting in Binary format, that’s why here we are dividing the number by **2**
* Decimal
  + Base is 10 numbers (i.e. contains 0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
* Octal
  + Base is 8 numbers (i.e. 0, 1, 2, 3, 4, 5, 6, 7)
  + Starts with ‘0o’; to indicate it’s octal format
  + E.g. Binary of 25 is “0o31”
* HexaDecimal
  + Base is 9 numbers (i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9) and 6 chars (i.a. a, b, c, d, e, f )
  + Starts with ‘0x’; to indicate it’s hexadecimal format
  + E.g. Binary of 125 is “0x7d”

|  |  |  |
| --- | --- | --- |
| Operations | Output |  |
| bin( 25 ) | '0b11001' | converts to binary format |
| print(0b11001) | 25 | Decimal format |
| oct( 25 ) | '0o31' | converts to octal format |
| hex( 125 ) | '0x7d' | converts to hexadecimal format |

**String Operations:**

|  |  |  |
| --- | --- | --- |
| String Operations | Output |  |
| print('swara') | swara |  |
| print("swara") | swara |  |
| print("swara's laptop") | swara's laptop |  |
| print('swara "laptop"') | swara "laptop" |  |
| print('swara\'s "laptop"') | swara's "laptop" |  |
| 'swara' + 'swara' | ‘swaraswara’ |  |
| Print('swara' + 'swara') | swaraswara |  |
| 5 \* 'swara' | ‘swaraswaraswaraswaraswara’ | 5 times ‘swara’ |
| Print(5 \* 'swara') | swaraswaraswaraswaraswara | 5 times ‘swara’ |
| print('c:\docs**\n**avin') | c:\docs avin | \n considered as new line char here |
| print(**r**'c:\docs\navin') | c:\docs\navin | here 'r' converts string into raw string (print as it is) |

**Math Operations:**

* To access some inbuilt math functions/methods, we have to use import them as

>>> **import math**

And then you can use math functions as below:

>>> print( **math.sqrt(25)** )

>>> 5.0

|  |  |  |
| --- | --- | --- |
| operations | Output |  |
| math.sqrt(25) | 5 |  |
| math.floor(2.9) | 2 |  |
| math.ceil(2.1) | 3 |  |
| math.pow(8,3) | 512 | 8\*8\*8 |
| math.pi | 3.14 |  |
| math.e | 2.71 |  |

* You can use allies for packages/libraries as below

>>> **import math as m**

>>> print( **m.**sqrt(25) )

>>>5.0

* Also, instead of importing complete package/libraries, you can import specific functions

>>> **from math import sqrt, pow**

>>> print( **sqrt(**25**)** )

>>> 5.0

Here, we don’t need to use **math.** as we imported specific methods.

How to swap numbers (a = 6, b = 5; make it to a = 5, b = 6):

* Using 3rd variable:

temp = a

a = b

b = temp

* Without using 3rd variable

a = a + b //a = 11

b = a – b //b = 11 – 5 = 6

a = a – b //a = 11 – 6 = 5

* Using cap operator (or ‘xor’ operator or ‘bitwise exclusive-or’ operator)

a = a **^** b //a = 3

b = a **^** b //b = 6

a = a **^** b //a = 5

* Using python’s method

**a,b = b,a**

Note: it will not work if you write in two different statement as

a = b

b = a

Why “a,b = b,a” swaps the value in python:

Because right side part gets into stack and the stack position gets assigned to left side variables.

**Variables:**

* Don’t need to define variable type
* Variable auto detect type of data whether it’s integer, float or string

|  |  |  |
| --- | --- | --- |
| Variables | Output |  |
| x = 3 x + 5 | 8 |  |
| \_ + 10 | 18 | '\_' takes output of previous operations |
| name = 'youtube' name + ' channel' | 'youtube channel' |  |
| name[0] | 'y' |  |
| name[6] | 'e' |  |
| name[-1] | 'e' | start from ending chars |
| name[-7] | 'y' |  |
| name[0:2] | 'yo' | stars at 0th index till 2nd index; i.e. ends at 1st index |
| name[1:4] | 'out' | stars at 1st & ends at 3rd index |
| name[3:1] | ' ' |  |
| name[3:3] | ' ' |  |
| name[3:5] | 'tu' | stars at 3rd & ends at 4th index |
| name[2:] | 'utube' | stars at 2nd index till end |
| name[:4] | 'yout' | stars at 0th & ends at 3rd index |
| name[3:10] | 'tube' | stars at 3rd & ends at end; won't give error here |
| 'my ' + name[3:] | 'my tube' |  |
| name[0:3] = 'my' | ERROR | cannot change letters in a string with simple assignment |

More info about variables:

* As you know, each variable has it’s own unique address (memory location)

Example: var = 10

Here variable ‘var’ may have memory address something like 88379297

* We can print this address using

>>> **id(**var**)**

🡺88379297

* Another example

Var1 = 10

Var2 = Var1

Here, Var1 and Var2 will point to same memory address in Python; unlike in other languages

>>> **id (**Var1**)**

🡺58711245

>>> **id (**Var2**)**

🡺58711245

Also, if you fetch the address if 10, it will point to same memory location

>>>**id (**10**)**

🡺58711245

* So, Python is memory efficient language
* Now, if you change the value of ‘Var1’, memory address of it will be changed

Var1 = 8

>>> **id (**Var1**)**

**🡺**7754128754

>>> **id (**8**)**

**🡺**7754128754

>>> **id (**Var2**)**

🡺58711245 //memory address of ‘Var2’ will remain as existing

>>>print(Var2)

🡺10

>>>**id (**10**)**

🡺58711245

* In Python, we cannot define as ‘Constant’ variable as such; however you can define a variable with capital letters, so that readers will understand that this variable value is used as constant in the program and we don’t have to change it.

**Lists:**

|  |  |  |
| --- | --- | --- |
| List Operations | Output |  |
| num=[25,10,11,29,35] print(num) | [25, 10, 11, 29, 35] |  |
| num[0] | 25 |  |
| num[4] | 35 |  |
| num[-1] | 35 |  |
| num[-5] | 25 |  |
| num[2:] | [11,29,35] | similar to string operations |
| num[2:3] | 11 |  |
| values=[20.5,'swara', 35] print(values) | [20.5, 'swara', 35] | no Error; can assign different type of values to a list |
| myList=[num,values] print(myList) | [[25, 10, 11, 29, 35],  [20.5, 'swara', 35]] | |
| num.append(45) print(num) | [25, 10, 11, 29, 35, 45] | appends a value at the end |
| num.insert(2,77) print(num) | [25, 10, 77, 11, 29, 35, 45] | inserts value 77 at 2nd index |
| num.remove(29) print(num) | [25, 10, 77, 11, 35, 45] | removes element of value 29 |
| num.pop(1) print(num) | [25, 77, 11, 35, 45] | removes element at 1st index |
| num.pop() | 45 | removes and print last element |
| print(num) | [25, 77, 11, 35] | printed after pop() operation |
| del num[2:] print(num) | [25, 77] | delete elements from 2nd index till end |
| num.clear() print(num) | [ ] | clears the list |
| num.extend([23,77,47,49,50]) print(num) | [23,77,47,49,50] | adds multiple values in a list |
| num.sort() print(num) | [23, 47, 49, 50, 77] | sorting of a list |
| min(num) | 23 | minimum value from a list |
| max(num) | 77 | max value from a list |
| sum(num) | 246 | sum of all numbers in a list |

**Tuple:**

* It’s similar to List; however values inside it are unchangeable, we cannot extend or append the values on Tuple
* Tuple example:

myTup = **(**23,45,33,67,33**)**

* Only we can use count() and index() on tuple.

>>> myTup**.count**(33)

🡺 2

i.e. it shows how many times 33 has occurred into the tuple list

>>> myTup**.index**(67)

🡺 3

i.e. it shows index of the number

* When you don’t want to change elements in the list, you can use Tuple
* Also, operations with Tuple can be faster than List

**Set:**

* Set is collection of unique elements
* Set example:

>>>mySet = **{**23,45,11,67,44,11**}**

>>>print(mySet)

🡺{67,11,44,23,45}

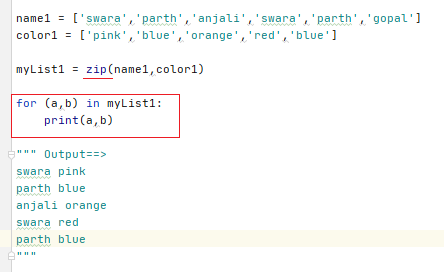
* Set prints elements in any order; does not keep stored sequence
* Also, set prints unique values; here, 11 printed only once
* Indexing is not supported in Set
* Most the other methods can be used like in List; such as .update, .extend, .count etc

**Dictionary:**

* Stores data on key-value pair

|  |  |  |
| --- | --- | --- |
| Dictionary operations | Output |  |
| myDict = {1:'Swara', 2:'Parth', 4:'Anjali'} print(myDict) | {1: 'Swara', 2: 'Parth', 4: 'Anjali'} |  |
| myDict**[1]** | 'Swara' |  |
| myDict[4] | 'Anjali' |  |
| myDict[3] | ERROR |  |
| myDict**.get(**1**)** | 'Swara' | Also get() can be used to fetch the value by providing key |
| myDict.get(4) | 'Anjali' |  |
| myDict.get(3) | <<BLANK>> | nothing will be displayed as output |
| print(myDict.get(3)) | None | "None" will be displayed as output |
| myDict**.get(**1, 'Not Found'**)** | 'Swara' | if value is not found for a given key, given message will be displayed |
| myDict.get(3, 'Not Found') | 'Not Found' |  |
|  |  |  |
| myKeys = ['sp', 'pp', 'ap'] myValues = ['Swara P', 'Parth P', 'Anjali P'] myDict = **dict**(**zip**(myKeys, myValues)) print(myDict) | {'sp': 'Swara P', 'pp': 'Parth P', 'ap': 'Anjali P'} | here, defined separate list of Keys and Values. Then zipped keys and values together. Then converted the zip into Dictionary |
| myDict['gp'] = 'Gopal P' print(myDict) | {'sp': 'Swara P', 'pp': 'Parth P', 'ap': 'Anjali P', 'gp': 'Gopal P'} | Adds key-value in the Dictionary |
| myDict**.keys()** | dict\_keys([‘sp’, ’pp’, ‘ap’, ‘gp’]) | Prints keys from the dictionary |
| myDict**.values()** | dict\_values([‘Swara P’, ’Parth P’, ‘Anjali P’, ‘Gopal P’]) | Prints values from the dictionary |
| **del** myDict['gp'] print(myDict) | {'sp': 'Swara P', 'pp': 'Parth P', 'ap': 'Anjali P'} | Deletes key-value pair from Dictionary |
| myDict['sp'] = 'Gopal P' print(myDict) | {'sp': 'Gopal P', 'pp': 'Parth P', 'ap': 'Anjali P'} | if key already presents, then it will update the value |
|  |  |  |
| myDict = {'JS': 'Atom', 'CS': 'VS',   'Python': ['Pycharm', 'Sublime'],  'Java': {'JSE': 'Netbeans', 'JEE': 'Eclipse'}} |  | we can define list and dictionaries inside a Dictionary as shown here |
| print(myDict) | {'JS': 'Atom', 'CS': 'VS',  'Python': ['Pycharm', 'Sublime'], 'Java': {'JSE': 'Netbeans', 'JEE': 'Eclipse'}} |  |
| myDict['JS'] | 'Atom' |  |
| myDict['Python'] | ['Pycharm', 'Sublime'] |  |
| myDict['Python'][1] | 'Sublime' |  |
| myDict['Java'] | {'JSE': 'Netbeans', 'JEE': 'Eclipse'} |  |
| myDict['Java']['JEE'] | 'Eclipse' |  |

**Zip() function example:**

* Zip function combine two set of lists
* 
* 
* 
* 

**Input:**

* Use **input()** method to read the values from users input/console

>>> x = input()

5

>>> print(x)

>>> ‘5’

* NOTE: Input() always converts the provided value in a string. So, here ‘5’ is printed as String.
* You can convert it in Int as below

>>> x = **int( input(**5**) )**

>>> 5

>>> x = int( input("Enter a number"))

Enter a number 8

>>> 8

* Read and print single character

>>> ch = input(“enter a char”)**[0]**

pqr

>>> p //when you enter pqr as input, only p will be assigned to ‘ch’

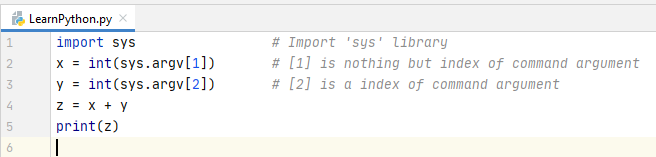
* Read and calculate mathematical expression

>>> result = **eval(** input(“enter a expression”) **)**

enter a expression 5+7-1

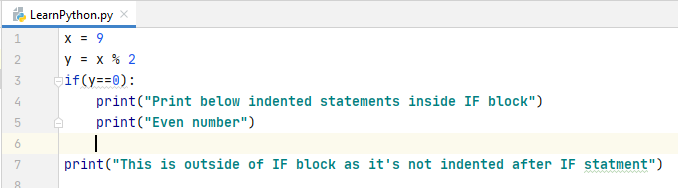
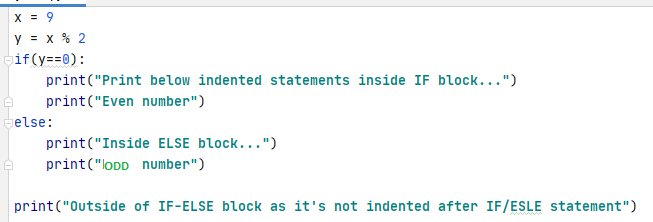
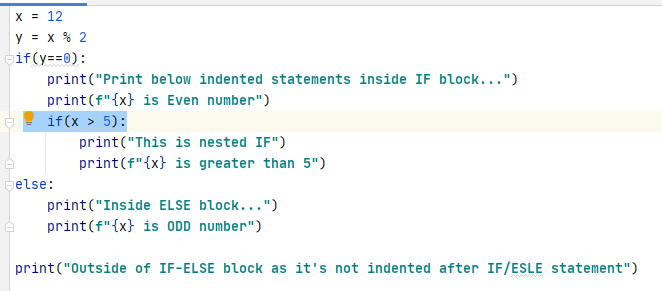
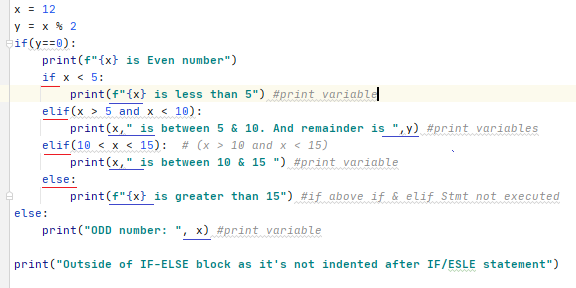
>>> 11

* When you have to run the program from command prompt with multiple inputs, we can use **sys.argv[]**

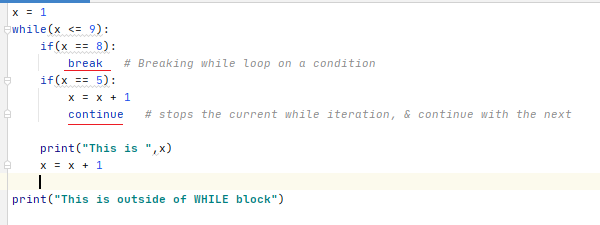


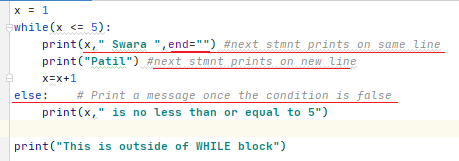


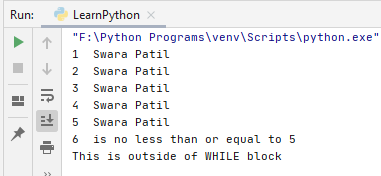
**If-Else statement:**

* 
* 
* 
* 

**While loop:**

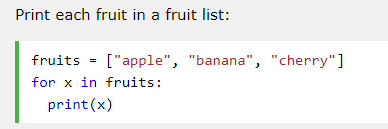
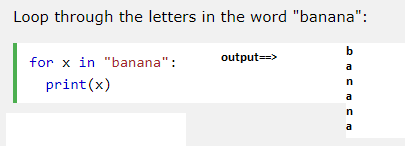
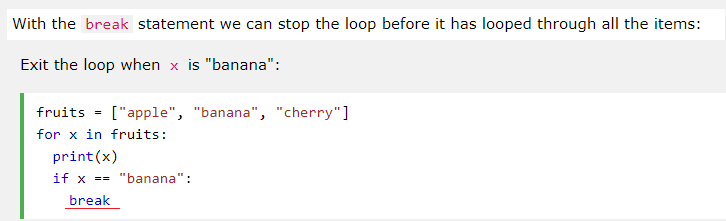
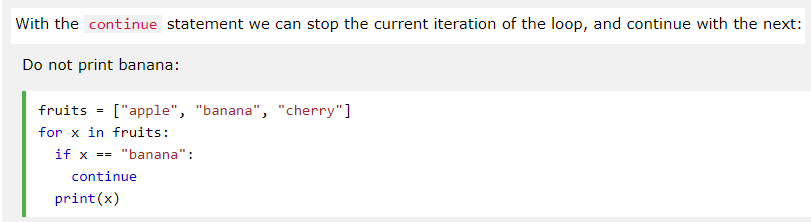
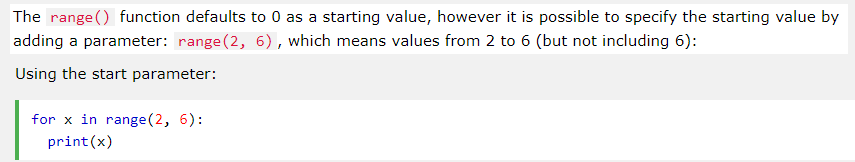
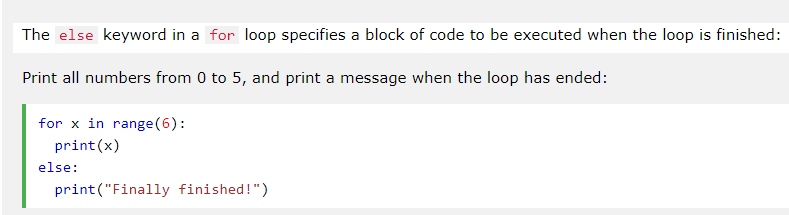
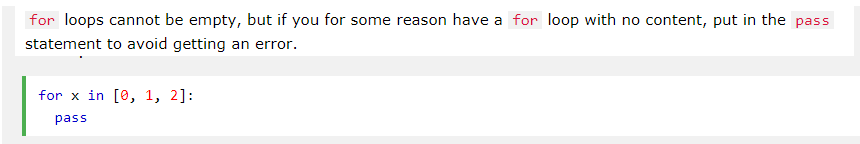
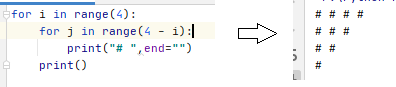
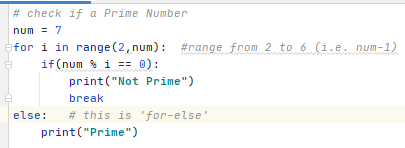




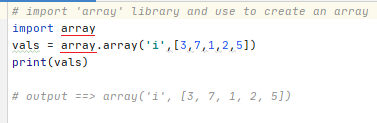
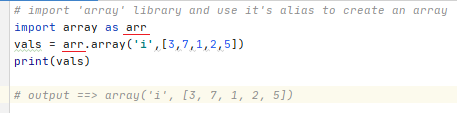
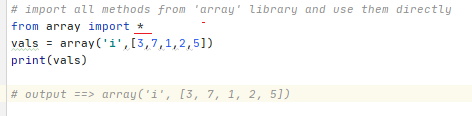
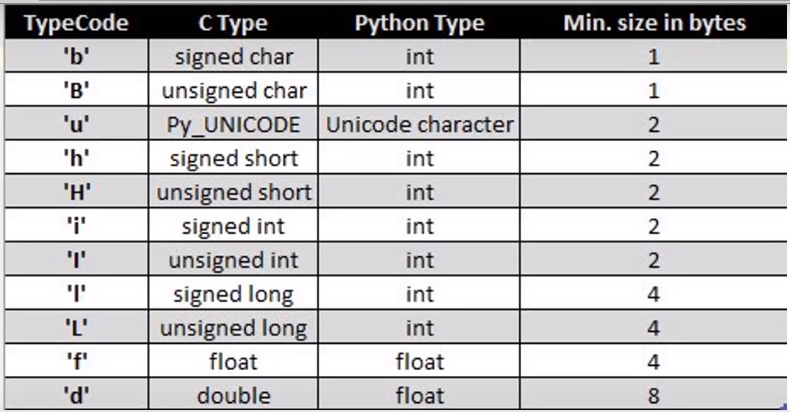
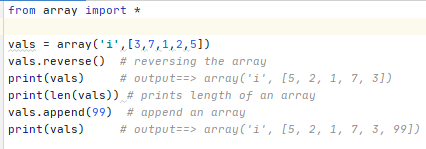
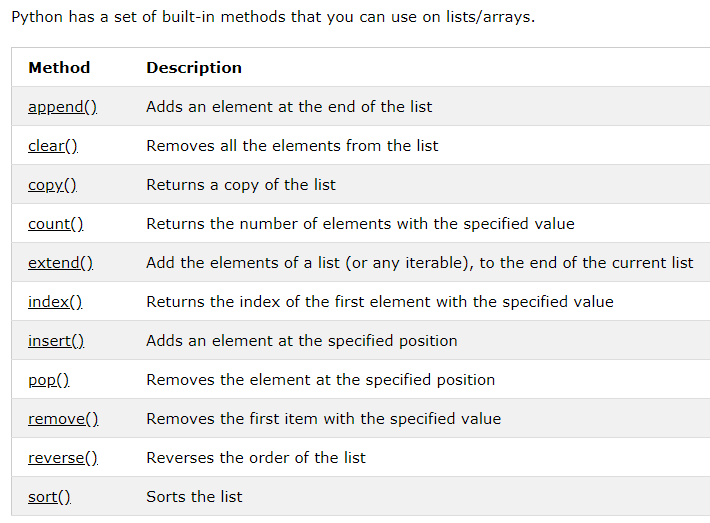
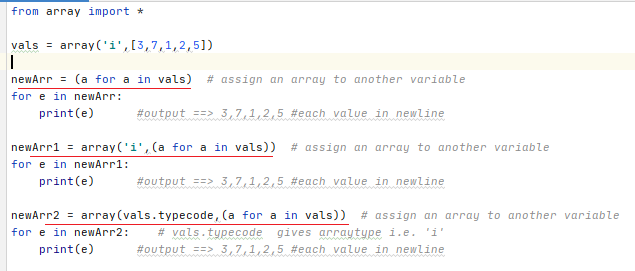
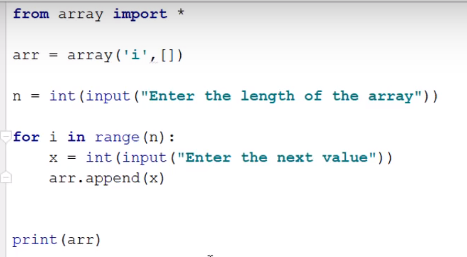


* NOTE: If there is ‘break’ statement inside while loop and ‘break’ is executed, then ‘else’ part will not be executed.

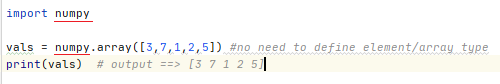
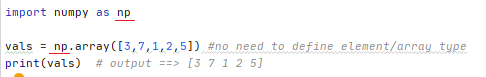
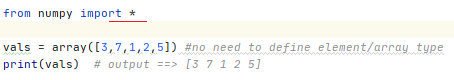
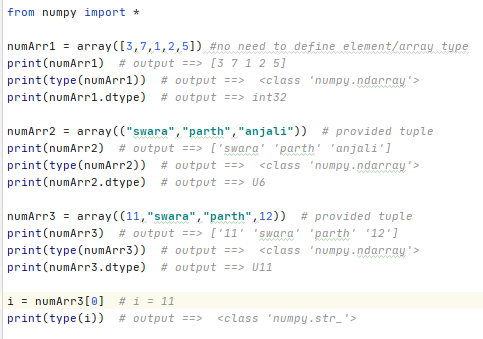
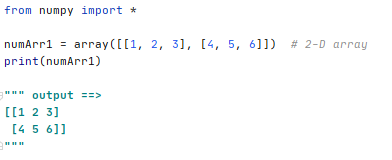
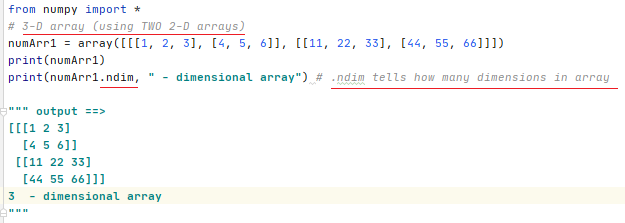
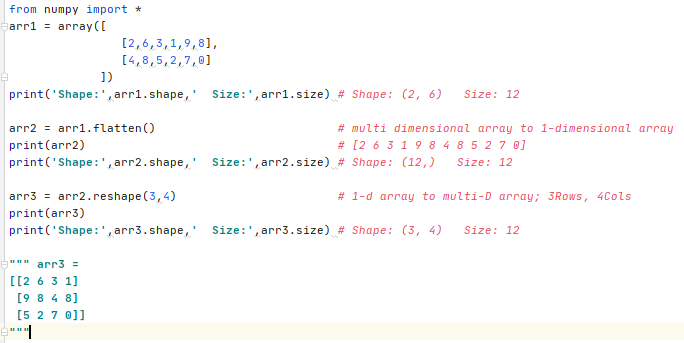
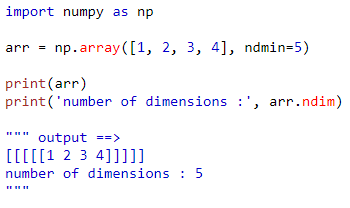
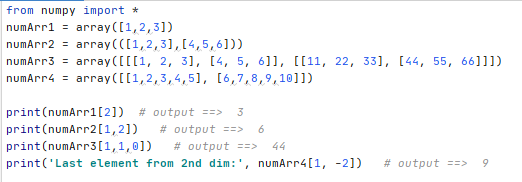
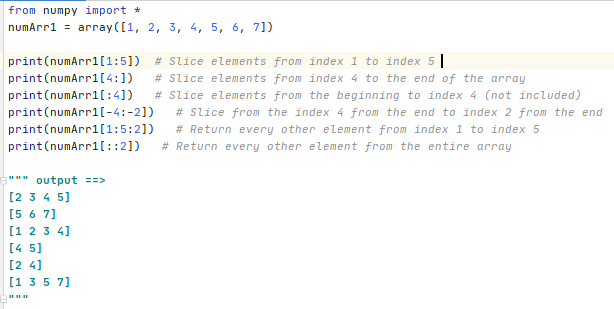
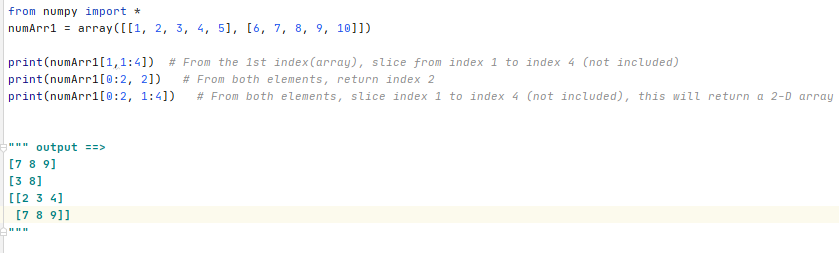
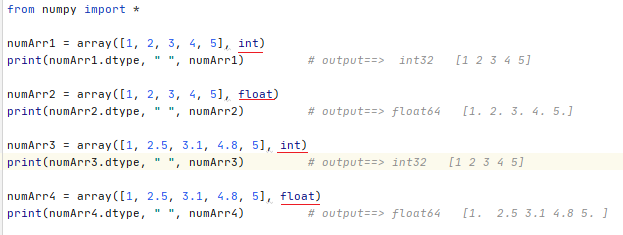
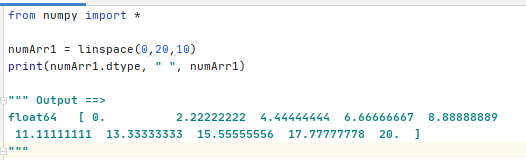
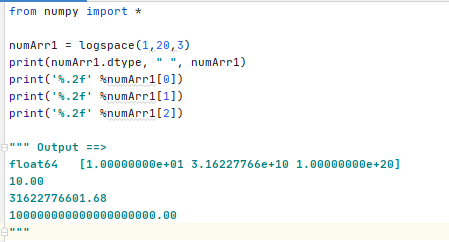
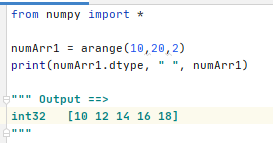
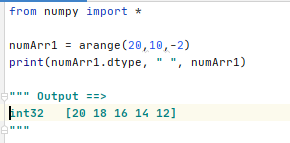
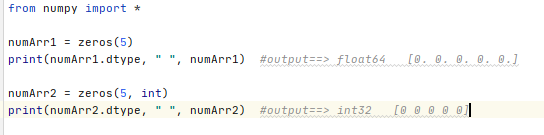
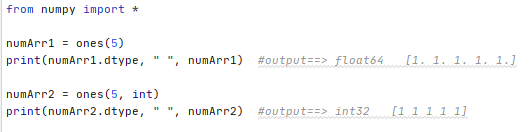
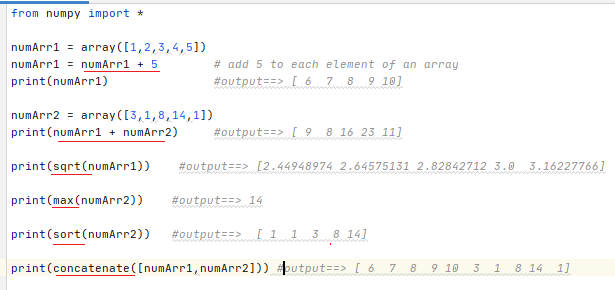
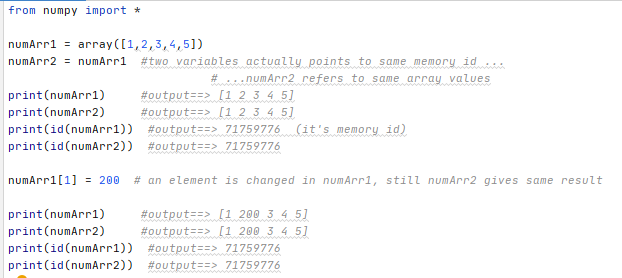
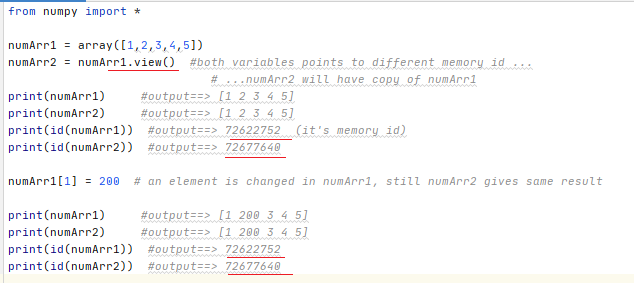
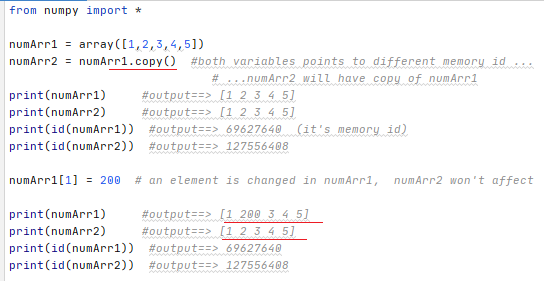
**For Loop:**

* A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, range or a string)
* 
* 
* 
* 
* 
* 
* NOTE: If there is ‘break’ statement inside for loop and ‘break’ is executed, then ‘else’ part will not be executed.
* 
* NOTE: “pass” statement can be used in any block (if, for, while, function, class, etc) where you don’t have any statement to execute for now and just keep it blank.
* Print a pattern:
* 
* Prime Number:
* 

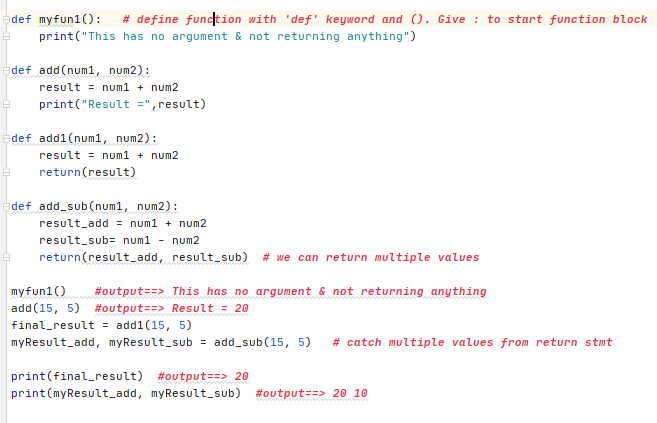
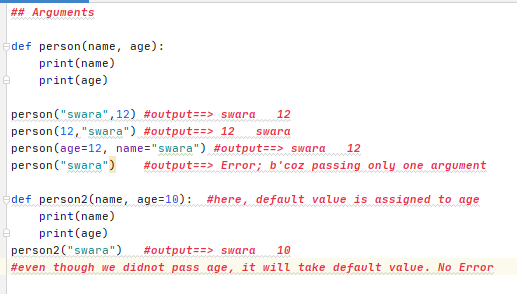
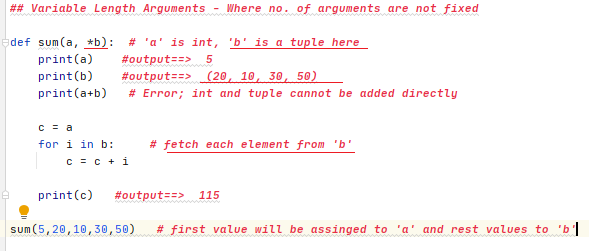
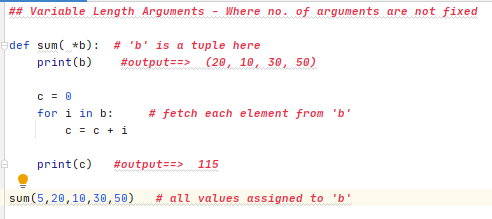
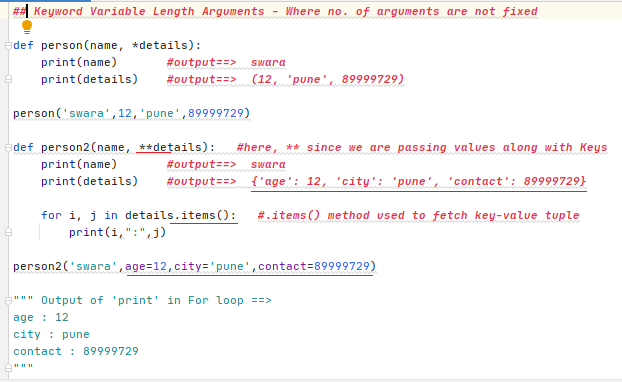
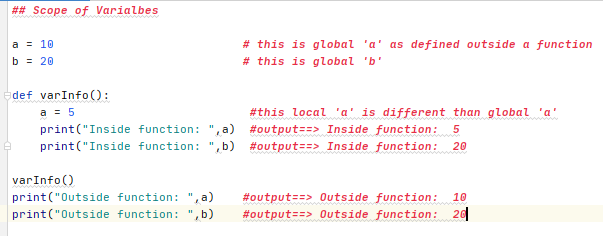
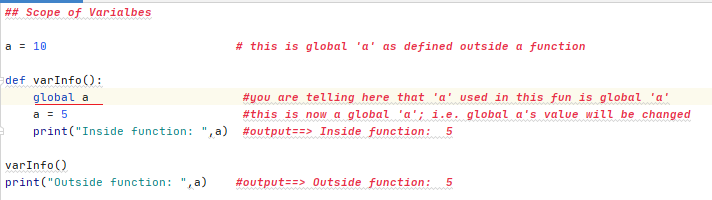
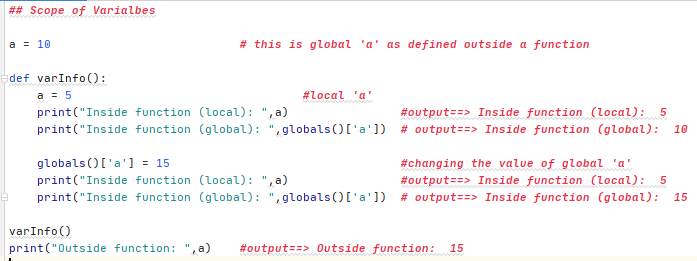
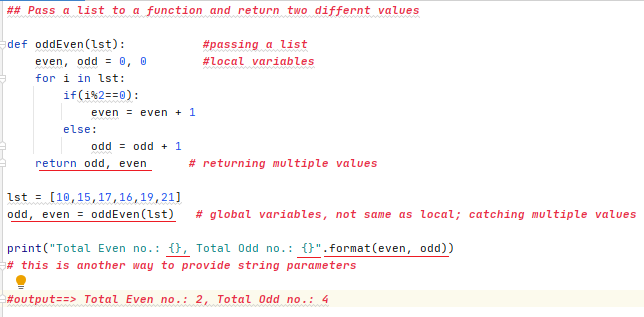
**Arrays:**

* Arrays are similar to list, tuple which holds more values at a time
* But in array you have to enter all elements of same type.
* In list, we can enter multiple type of elements such as String, int, float in same list.
* We have to import ‘array’ library: we can do it three ways
  + 
  + 
  + 
* In above examples, ‘i’ is used to define type of array i.e. int type.
* We can use followings:
* 
* Array with some more methods:
* 
* 
* Assign an Array to another variable
* 
* Inserting values in an array
* 

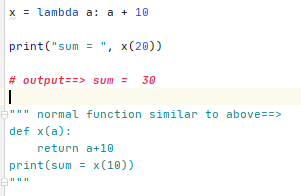
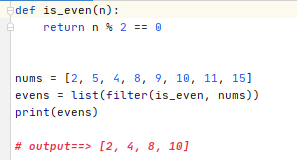
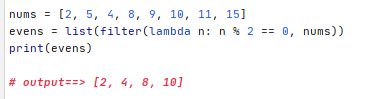
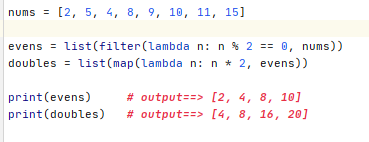
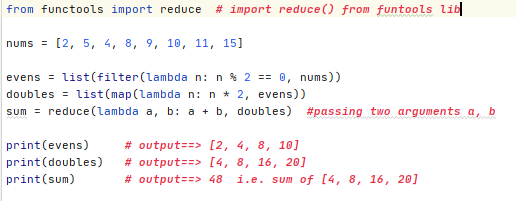
**NumPy:**

* NumPy is a python library (3rd party lib) used for working with arrays.
* It also has functions for working in domain of linear algebra, fourier transform, and matrices.
* NumPy stands for Numerical Python
* It’s faster than lists, arrays
* NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
* This behavior is called locality of reference in computer science
* If you have [Python](https://www.w3schools.com/python/default.asp) and [PIP](https://www.w3schools.com/python/python_pip.asp) already installed on a system, then installation of NumPy is very easy.
* Install it using this command:
* C:\Users\ > **pip install numpy**
* In PyCharm, goto Files > Settings > Project:xxxx > Python Interpreter > click + sign (top right) > search for ‘numpy’ > install package
* Same like array, we have to import numpy in the code.
* 
* 
* 
* NumPy is used to work with arrays. The array object in NumPy is called **ndarray**.
* We can create a NumPy **ndarray** object by using the **array()** function.
* 
* **Multi-dimensional array:**
* Into arrays, defining multiple dimensional array is not possible; So here we can use NumPy.
* 
* 
* 
* you can define the number of dimensions by using the ndmin argument
* Create an array with 5 dimensions and verify that it has 5 dimensions
* 
* **Indexing:** You can access an array element by referring to its index number.
* 
* **Slicing:**  taking elements from one given index to another given index
* 
* 
* NumPy has a big topic. You can refer <https://www.w3schools.com/python/numpy_intro.asp> section to learn more about NumPy
* You can define, data type into numpy array
* 
* There are different ways to create an array in NumPy
  + **array()** 🡺 we saw in above examples
  + **linspace()**
    - here, 0 is starting digit, 20 is ending digit, 10 is total equal parts from 0 to 20 (i.e. start and ending number).
    - Dtype is always float
    - If you don’t mention 3rd element (here 10), then it consider 50 equal parts by default
    - 
  + **logspace()**
    - here, 1 is (101 ), 20 is (1020) and 3 is total three parts (with equal difference) starting from 101 to 1020
    - 
  + **arange()**
    - here, 10 is starting number, 20 is ending number (not included), 2 is steps i.e. array contains the numbers starting with 10 till 20 incrementing by 2
    - 
    - 
  + **zeros()**
    - Creates an array of Zeros. By-default it’s float.
    - Here, it will create an array of 5 zeros
    - 
  + **ones()**
    - Creates an array of Ones. By-default it’s float.
    - Here, it will create an array of 5 ones
    - 
* Some more operations with an array:
* 
* **Assigning of an array (=)**:
* Two variables points to same memory id; numArr2 will refer to numArr1 values or array element
* If you update something in numArr1, it would reflect into numArr2
* 
* **Assigning of an array (.view())**:
* Two variables points to different memory id; numArr2 will have a copy of numArr1 values or array element
* If you update something in numArr1, it would reflect into numArr2 or vice versa
* It’s also called ‘shallow copy’
* ****
* **Assigning of an array (.copy())**:
* Two variables points to different memory id; numArr2 will have a copy of numArr1 values or array element
* If you update something in numArr1, it won’t reflect into numArr2 or vice versa
* It’s also called ‘deep copy’
* ****

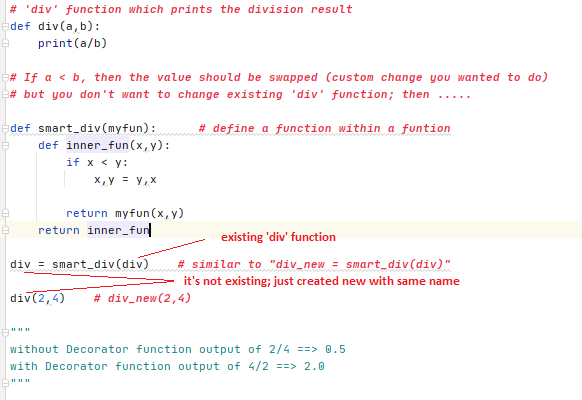
**Functions:**

* ****
* ****
* ****
* ****
* ****
* **Scope of variables:**
* ****
* Here, Note that local ‘a’ will not affect to global ‘a’ i.e. global ‘a’s value will not be changed
* If you wanted to change global ‘a’s value inside a function then see below code
* 
* If you wanted to use same variable name as local and global inside same function, see below code
* 
* Passing a list to the function
* 

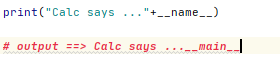
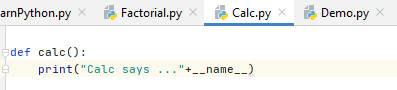
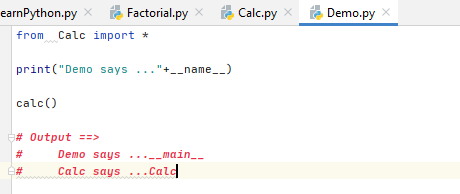
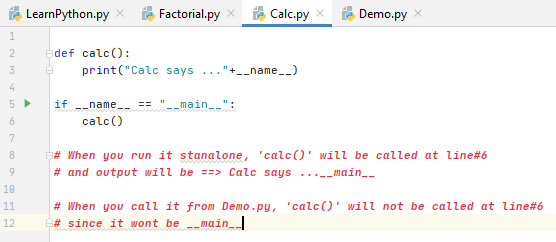
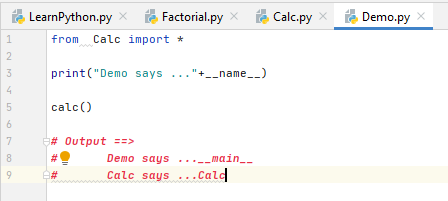
**Lambda functions:**

* Python Lambda function is known as the anonymous function that is defined without a name
* Lambda functions can accept any number of arguments, but they can return only one value in the form of expression
* The anonymous function contains a small piece of code like a inline (one line code)
* 
* **Lambda function with filter():**
* The Python built-in **filter() function** accepts a function and a list as an argument. It provides an effective way to filter out all elements of the sequence. It returns the new sequence in which the function evaluates to **True**.
* Using normal function:
* 
* Using Lambda function:
* 
* **Lambda function with map():**
* The **map() function** in Python accepts a function and a list.
* It gives a new list which contains all modified items returned by the function for each item.
* 
* **Lambda function with reduce():**
* The **reduce() function** in Python takes in a function and a list as argument.
* The function is called with a lambda function and a list and a new reduced result is returned.
* This performs a repetitive operation over the pairs of the list.
* 

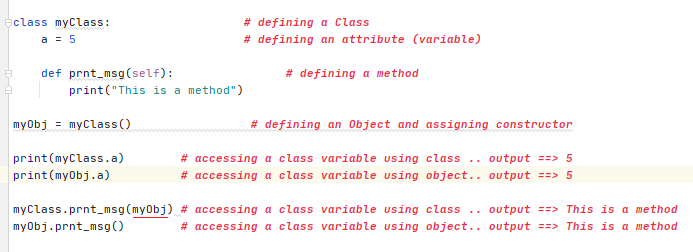
**Decorators**

* Decorators meaning Changing the behavior of an existing functions
* If the existing function is not accessible to you or you don’t want to change the code, then how to do that
* Here, we create a function (here, ‘smart\_div()’) and provide a function as a parameter (here, ‘myfun’)
* Then define another function inside (here, ‘inner\_fun()’) with the same number of parameters of existing function i.e. div(a,b).
* E.g. 

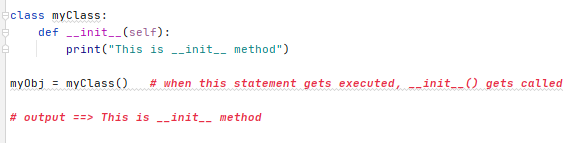
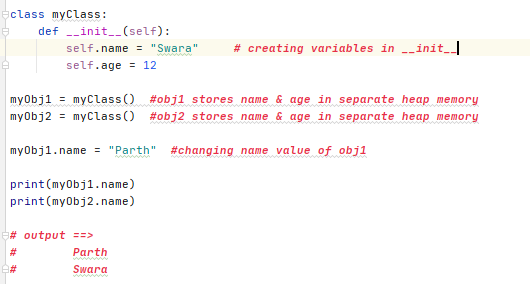
**\_\_name\_\_**

* This is a special variable which prints the current module name or “\_\_main\_\_” if it’s first executing module
* E.g. if you run simple python file (single module), as below
* 
* Here \_\_name\_\_ variable prints “\_\_main\_\_” since it’s starting module for execution.
* To elaborate it more, we will create two different modules or .py files and import a module into another
* 
* 
* Another example:
* 
* 

**Class**

* Object Oriented ‘Class’ concept
* 

**\_\_init\_\_**

* \_\_init\_\_() is a constructor
* Even though you don’t mention it into code, it gets called when you create an class object.
* In below example, we have added \_\_init\_\_(); but did not call it explicitly; still it’s getting executed
* 
* Creating variables in \_\_init\_\_:
* 
* When you write myObj1 = myClass(), object ‘myObj1’ gets pass thru as parameter in myClass() constructor and which gets caught by ‘**self’** keyword in the \_\_init\_\_(**self**) and the variables inside the constructor gets assigned or stored into that Object.
* The **self** parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
* It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class

**Variables**

**Instance(object) variables**

* Instance variables are assigned to an object separately
* E.g in above example, ‘name’ and ‘age’ are instance variables. These variables are gets assigned and saved differently on both objects myObj1 and myObj2. You can change the instance variable value for any of the object, which won’t affect the stored value of same variable in another object. Here, ‘name’ gets changed for myObj1 and it remains original for myObj2.
* If you define a variable inside \_\_init\_\_(), it becomes an instance variable

**Class(static) variables**

* Class variables are the common variables for all it’s objects. If you change the it’s value, it will change the value for all it’s object.
* If you define a variable outside \_\_init(), it becomes a class variable
* If you want to change it’s value, you have to access the variable using class name
* E.g. variable ‘ city = “Pune” ’ is defined as class variable in above example and you have to change it’s value somewhere in the program, you can access it as

myClass.city = “Mumbai”

**Methods**

* Unlike a function, methods are called on an object;

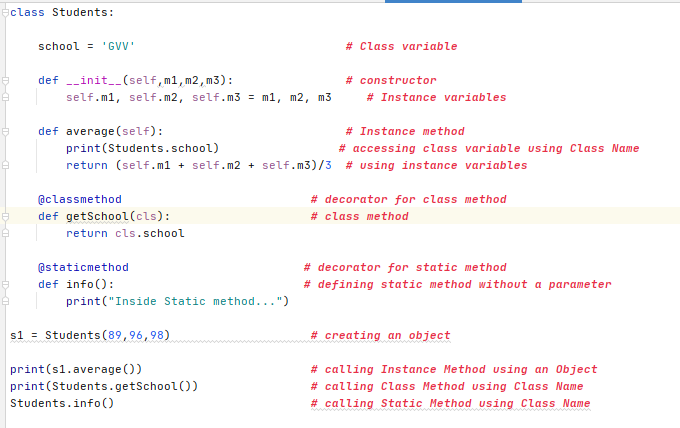
**Instance methods**

* Instance method uses instance variables
* Instance method uses ‘**self**’ parameter

**Class methods**

* Class method uses class variables
* Class method uses ‘**cls**’ parameter
* Also, we have to use **@classmethod** before the defining a class method

**Static methods**

* Class methods and Static methods are different; not same as variables (class variables and static variables are same thing)
* 
* Output🡺

GVV

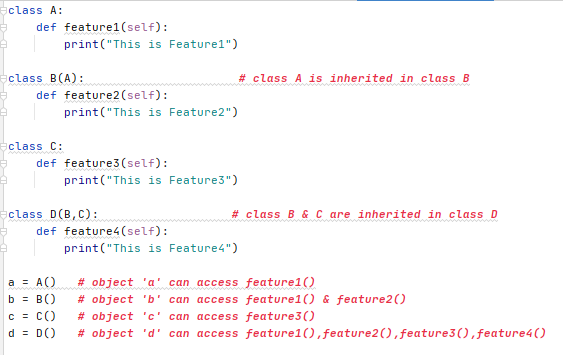
94.33333333333333

GVV

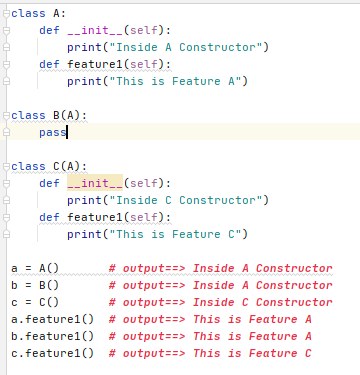
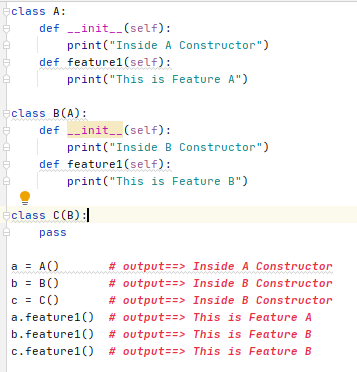
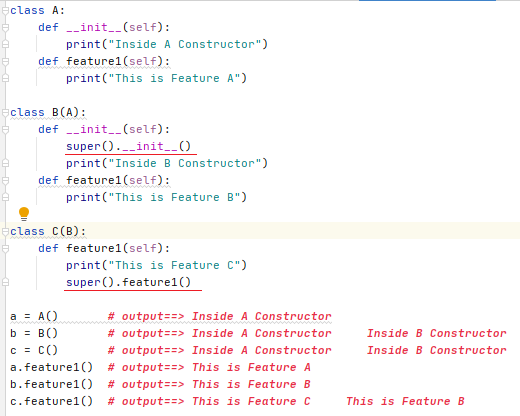
Inside Static method...

**Inner Class**

**Inheritance**

* 

**Constructor in Inheritance:**

* 
* Here, when constructor B() gets called, it will check it’s own \_\_init\_\_() and if not found then it executes from it’s parent.
* Similar for the method having same name; here, feature1().
* 
* Now, if you wanted to run own constructor as well as parent’s constructor then use **super().** as below.
* Same for methods of same name in different classes
* 

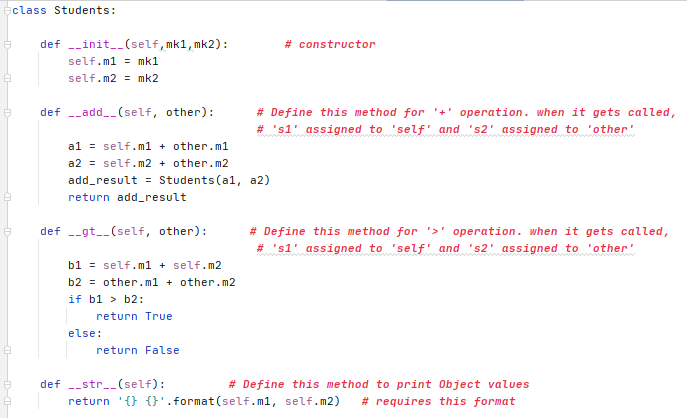
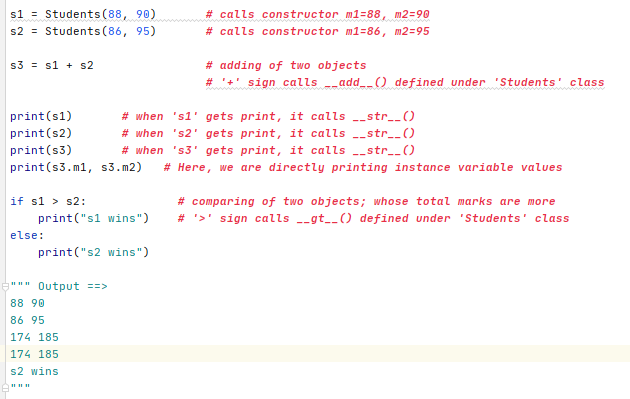
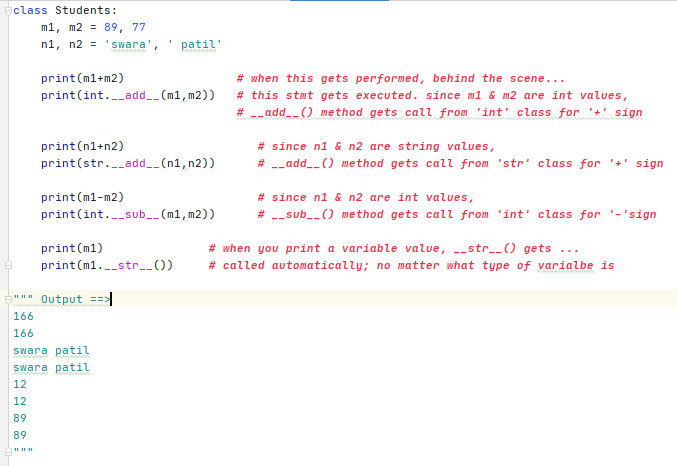
**Polymorphism**

* There are different ways to implement Polymorphism in python
  + Duck Typing
  + Operator Overloading
  + Method Overloading
  + Method Overriding

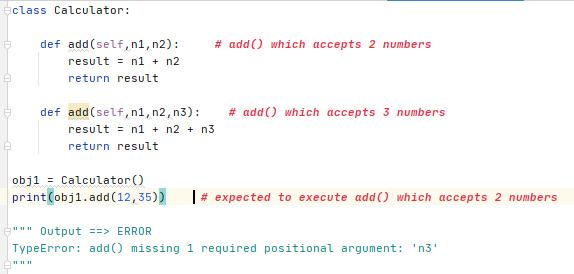
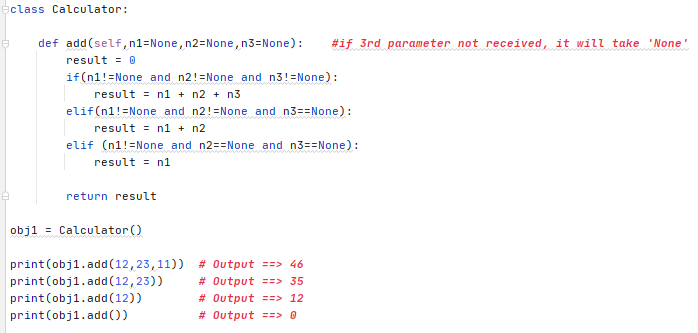
**Duck Typing**



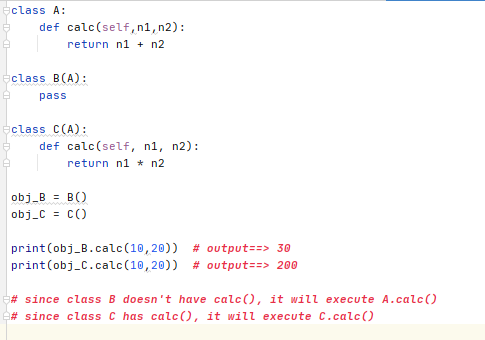
**Operator Overloading**

* Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because ‘+’ operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called Operator Overloading.Performing
* different operations on same objects
* 
* 
* In above example, we have defines \_\_add\_\_(), \_\_gt\_\_(), \_\_str\_\_() methods since we wanted to perform addition, greater than and printing values on class Objects.
* When you perform such operations (or use such operators) for normal values or variables, these predefined methods get called from out of the box from corresponding predefined class
* Example, when you perform addition of two integer values, \_\_add\_\_() gets call from predefined ‘int’ class. When you use + operator to concatenate two strings, \_\_add\_\_() gets call from predefined ‘str’ class. See below
* 

**Method Overloading**

* When multiple methods have same name in a class and accepts different number of arguments and/or different types of arguments, it’s called Method Overloading
* However, it will not work in python similar to other language like C++, Java
* E.g. if you write below code, it will get an error in python
* 
* Then how to handle it in Python; as below
* 

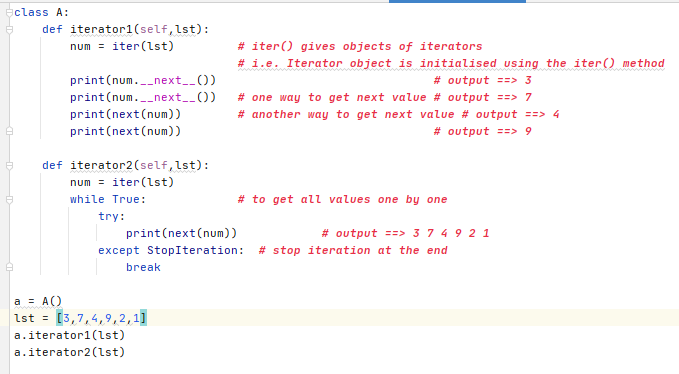
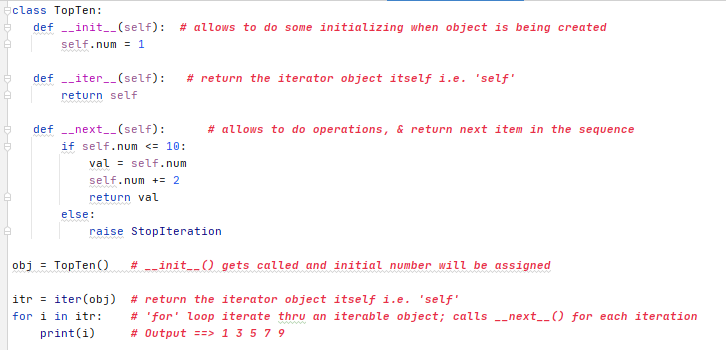
**Method Overriding**

* When multiple methods have same name and same number of parameters, it’s called Method Overriding
* Can we have multiple methods of same name and parameters in a class?
* Not exactly; But we may have it in different class and if these classes are uses ‘Inheritance’ then child class will have multiple methods of same name and parameters (b’coz one method inherited from parent); it’s called Method Overriding
* 

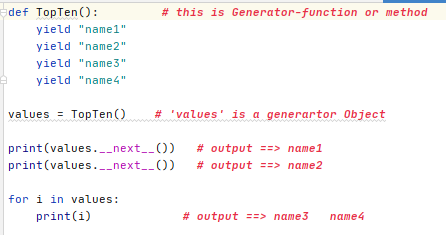
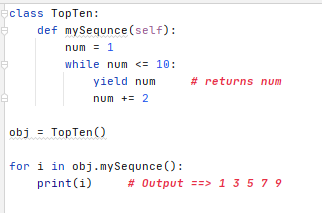
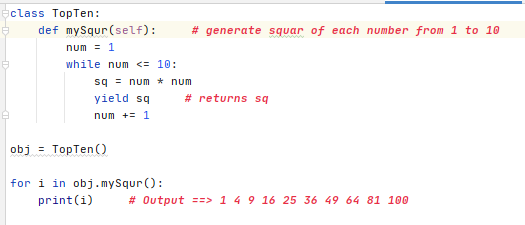
**Abstract Class and Abstract Method**

* A class which has abstract methods, called an Abstract Class
* A method which has no body/definition, only has name or declaration, it’s call Abstract method
* Python directly not supports Abstract method; for that we have import a module ‘ABC’ (Abstract Base Class)
* We cannot create an object of an Abstract Class; python will give error
* Add examples and more details

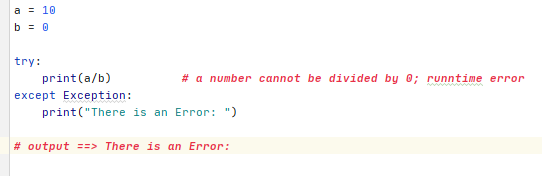
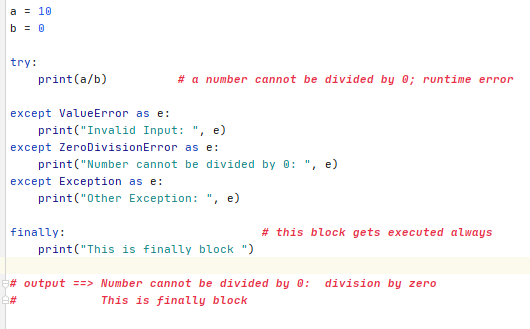
**Iterators**

* Iterator in python is an object that is used to iterate over iterable objects like lists, tuples, dicts and sets. Iterator object is initialized using the **iter()**method. It uses the **next()** method for iteration.
* **iter(iterable)\_\_** method that is called for initialization of an iterator. This returns an iterator object
* **next ( \_\_next\_\_ in Python 3)** The next method returns the next value for the iterable. When we use a for loop to traverse any iterable object, internally it uses the iter() method to get an iterator object which further uses next() method to iterate over. This method raises a **StopIteration** to signal the end of the iteration.
* 
* Build your own interator methods to display alternate numbers in 1 to 10 numbers
* 

**Generators**

* For iterators, we need iter() and next() methods to get next value; whereas in Generators, we don’t need these two methods
* Generators works similar to Iterators without iter() and next()
* When a method contains ‘yield’ statement, it means it’s ‘Generator’ method.
* When a method contains ‘return’ statement, it’s normal method (normal method may not contain ‘return’)
* In another words, A **generator-function** is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return. If the body of a def contains **yield**, the function automatically becomes a generator function.
* Generator functions return a **Generator-Object**. Generator objects are used either by calling the next method on the generator object or using the generator object in a **“for in” loop**
* Generators are more fast and consume less memory than Iterator since Iterator object holds all values and iterate it one by one; whereas Generator create a value one by one i.e. holds one value at a time
* We can use multiple ‘yield’ statement in a Generator method; below example of ‘yield’
* 
* See below example of Generator
* 
* Another example of Generator
* 

**Exception Handling**:

* We use **‘try-except’** block to handle runtime errors. See below examples:
* Multiple ‘except’ block can be written for one try
* ‘**finally’** block also can be written after ‘except’ block. ‘finally’ block will get executed in both cases i.e. if ‘try’ block gets successfully executed and in case of error, after ‘except’ block gets executed
* 
* 
* Generally, we have to put statements in ‘finally’ block which need to be executed in any case e.g. closing the DB connection or closing the already opened file

**File Handling**

* We use open () function in Python to open a file in read or write mode. As explained above, open ( ) will return a file object. To return a file object we use open() function along with two arguments, that accepts file name and the mode, whether to read or write. So, the syntax being: open(filename, mode). There are three kinds of mode, that Python provides and how files can be opened:

“ r “, for reading.

“ w “, for writing.

“ a “, for appending.

“ r+ “, for both reading and writing

“rb”, for reading binary file (e.g. Image files)

“wb”, for writing binary file

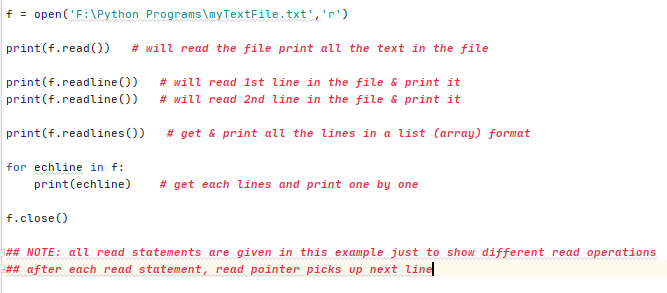
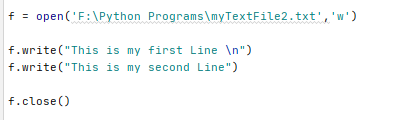
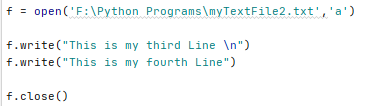
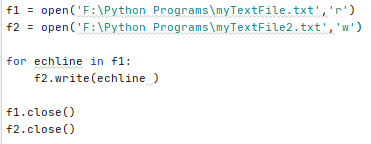
* One must keep in mind that the mode argument is not mandatory. If not passed, then Python will assume it to be “ r ” by default.
* Let’s say there is .txt file with below lines:

This is a Test File Line 1

This is a Test File Line 2

This is a Test File Line 3

This is a Test File Line 4

* Reading of file:
* 
* Writing into a file:
* 
* ‘write’ operation deletes all the data in the file and write into it i.e. overwrite it
* I file is not present at the given location, file gets created.
* Appending a file:
* 
* Copying a file to another file
* 

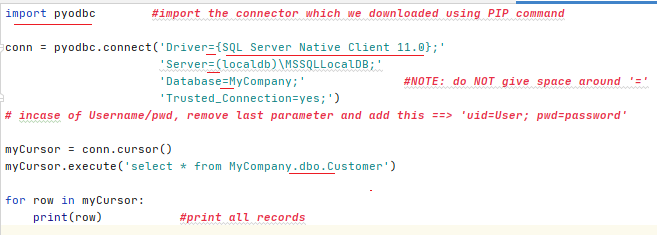
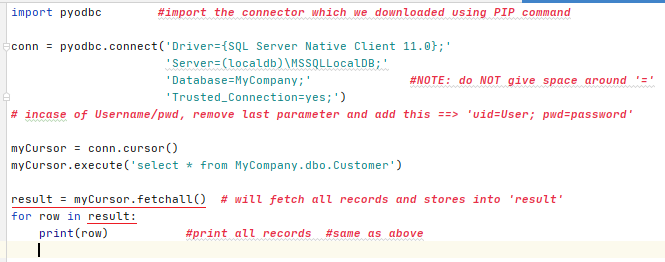
Multi-Threading



**Database Connection**

* We need to install SQL server connector in python using package installer.
* Run below command in command prompt

C:/> **pip install pyodbc**

* Obtain DB Server Name, Database Name, table name of SQL server instance
* Connect python with SQL server
* 
* 
* 