**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”

**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/>

**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

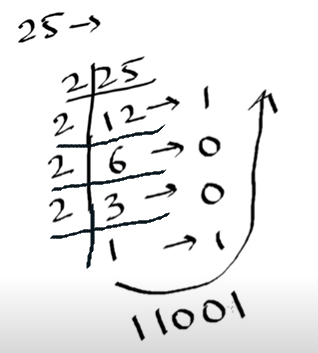
|  |  |  |
| --- | --- | --- |
| Operations | Output |  |
| 7 + 3 | 10 |  |
| 7 - 3 | 4 |  |
| 7 \* 3 | 21 |  |
| 7 \*\* 3 | 343 | (Exponent/power) 7 rest to 3 i.e. 7 \* 7 \* 7 |
| 7 / 3 | 2.33 |  |
| 7 // 3 | 2 | (Floor Division)  integer value of (7/3); skip decimals |
| 7 % 3 | 1 | (Modulus) gives 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' |  |

**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’

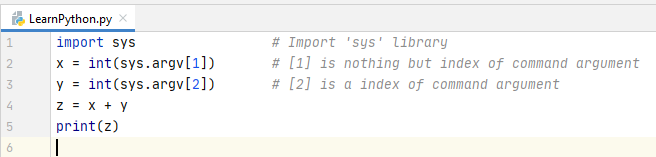
* Read and calculate mathematical expression

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

enter a expression 5+7-1

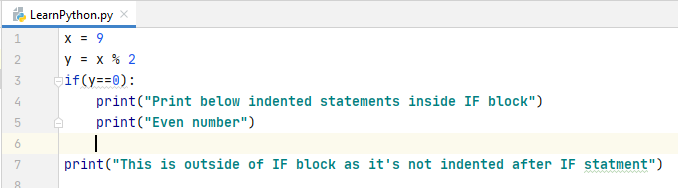
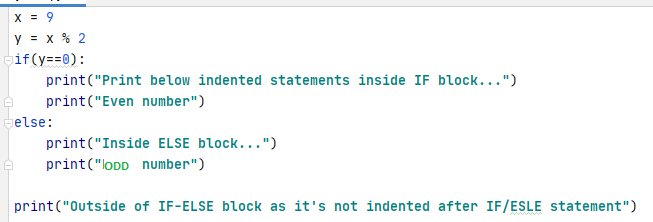
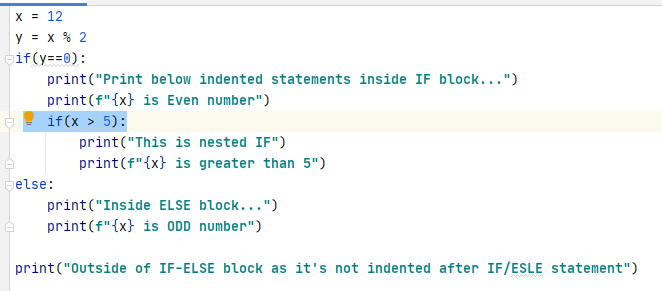
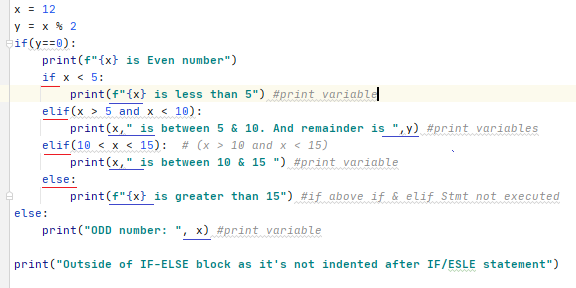
>>> 11

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

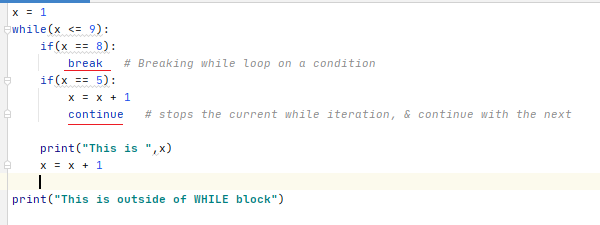


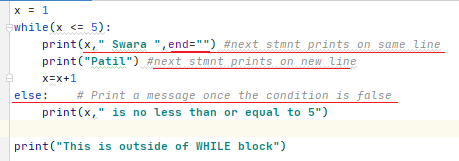


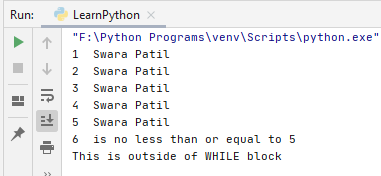
**If-Else statement:**

* 
* 
* 
* 

**While loop:**

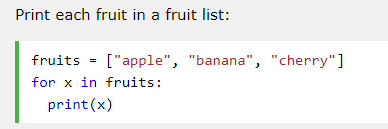
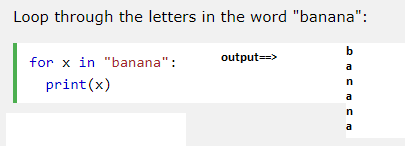
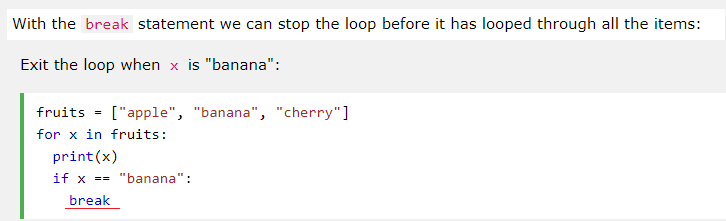
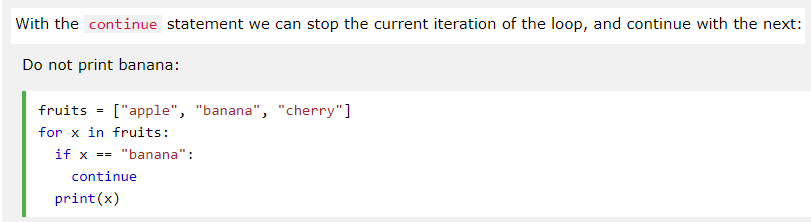
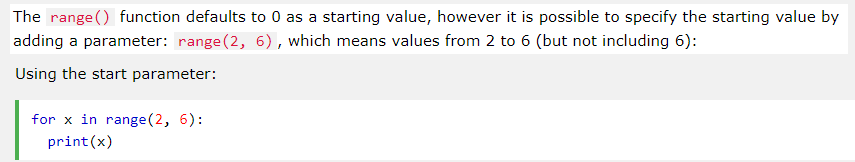
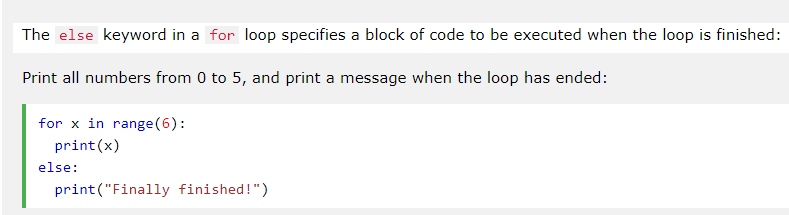
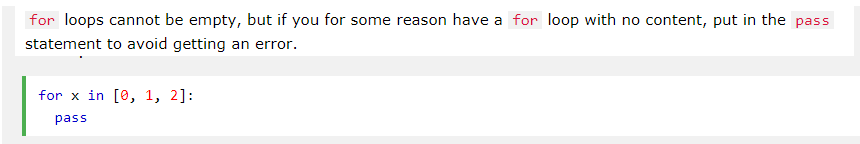
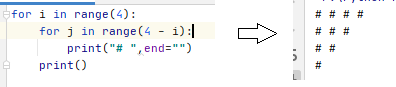
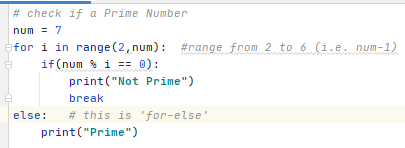




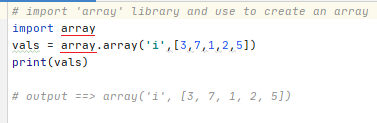
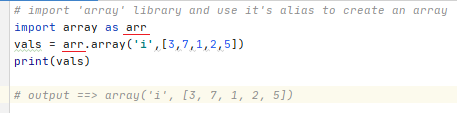
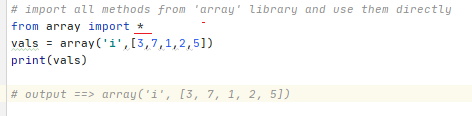
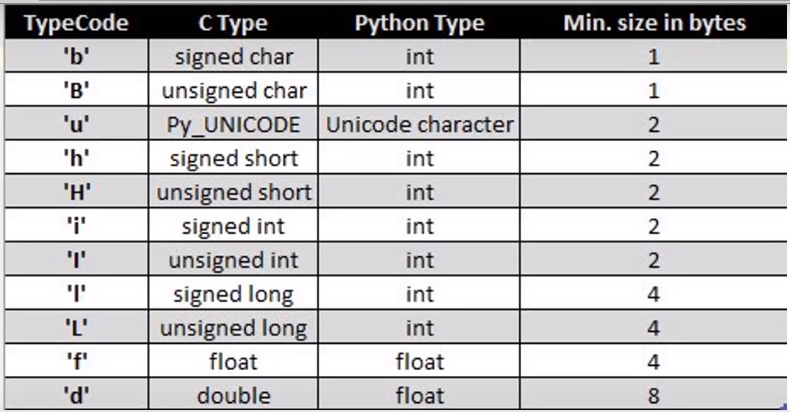
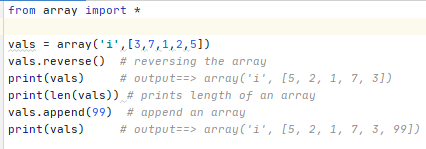
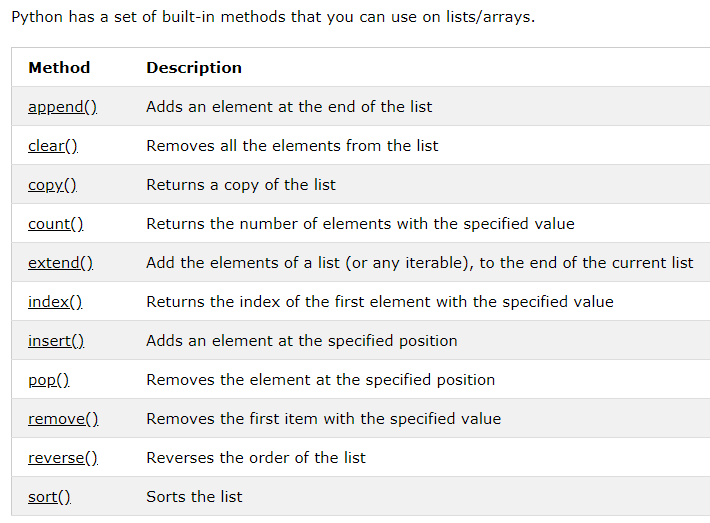
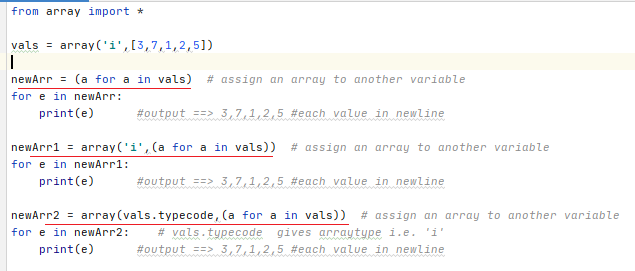
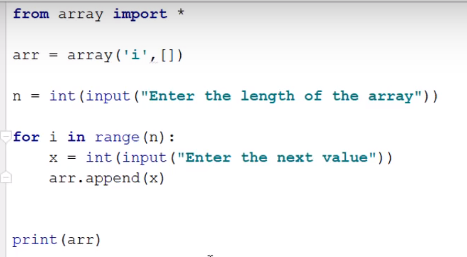


* NOTE: If there is ‘break’ statement inside while loop and ‘break’ is executed, then ‘else’ part will not be executed.

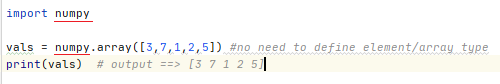
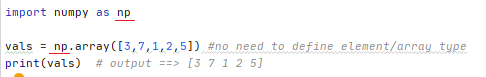
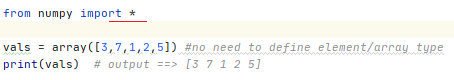
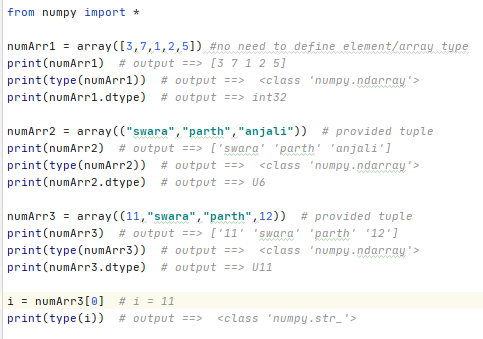
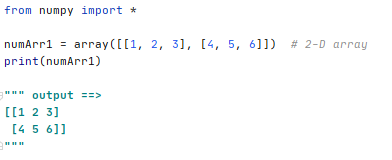
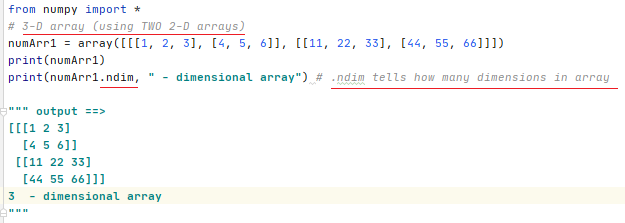
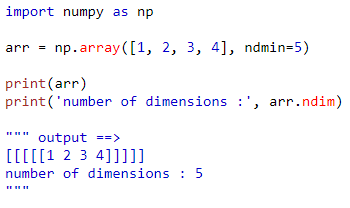
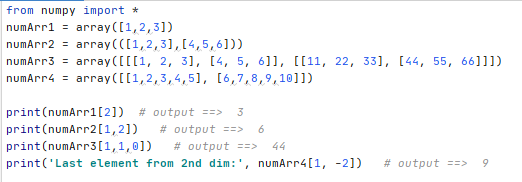
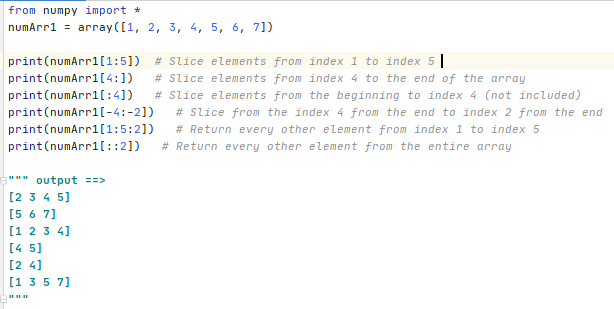
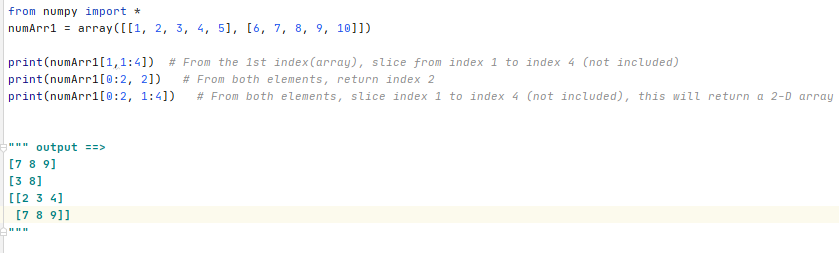
**For Loop:**

* 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