Python Programming for Beginners

Python print()

> Python print() is used to print the output on the screen.

print "hello world" using print()

```
print("hello world")
```

Now lets print your name in my case "Nishant Tiwari"

```
print("Nishant Tiwari")
```

Add, sub, multiply, and divide two number together and print the output

```
print(10+10)
print(20-10)
print(30*20)
print(100/6)
```

> 💡.py is the file extension for python file

Python Comments

> Python Comments are used to explain your python code and make your code more readable.

There are two types of Python Comments

1) Single Line Comments

Single Line Comments starts with #

```
print(10+20) #example of single line
```

💡 If you use Python Comments on a line then python will ignore the rest of the line

2) Multi Line Comments

Multi Line Comments starts and ends with ''' or """

```
'''
example of muti-line comment
print(20+20)
'''
#or
"""
```

```
print(20-10)
"""
```

Python Variables

> Variables are the space or a location used to store data in memory

Assign the value 10 to a variable

```
a = 10 #here a is a variable
print(a) #output: 10
```

Reassigning value to a variable

```
a = 20 #now new value is 20 to variable a
print(a) #output: 20
```

Now let us take three variable and print the values of each

```
a = 30
b = 40
c = 50
#print in a single line
print(a,b,c) #output: 30 40 50
print(a) #output: 30
print(b) #output: 40
print(c) #output: 50
```

Assign multiple value to multiple variables

```
a,b,c = 30,40,50 #same as above
print(a,b,c) #output: 30,40,50
```

Assign single value to multiple variables

```
a = b = c = 10
print(a,b,c) #output: 10 10 10
```

Rules and Naming Convention for Variables

There are some following rules for variables :

- **Variables always starts with Capital Letter, Small Letter or an Underscore** (_)

```
a = "hello"
A = "world"
_ = "dx4iot here"
print(a) #output: hello
print(A) #output: world
print(_) #output: dx4iot here
```

- **Variables always start with Capital Letter, Small Letter, or an Underscore (_)**

```
% = "hello"
1 = "hello"
print(%) #output: Error
print(1) #output: Error
```

- **If there is a variable of two words then separate it using _ (snake case) or use camelCase**

```
#snake_case
snake_case = "hello" #output: hello
#camelCase
camelCase = "world" #output: world
```

Python Data Types

> Variable can store data of different types such as int, float ,list, tuple ,set, etc

There are different data types such as:

Numbers

There are three types of data type under numbers

1) **Integer (int) :** integer values such as (-1,0,1,2)

```
a = 10 #int (integer)
print(type(a)) #output: <class 'int'>
```

💡 type() is used to check the type of data

2) **Float :** a floating-point number (10.3, 13.001, 21.223)

```
b = 10.5 #float
print(type(b)) #output: <class 'float'>
```

3) **Complex:** expressed in the form x + yj (x is the real part and y is the imaginary part)

```
c = 1 + 2j #complex (1+2j => x + yj)
print(type(c)) #output: <class 'complex'>
```

List

List is used to store multiple elements of different data types. List is an ordered sequence of mutable elements.

Example of a simple list

```
d = [10,10.5,20]
print(d) #output: [10, 10.5, 20]
print(type(d)) #output: <class 'list'>
```

print the first element of a list

```
d = [10,10.5,20]
print(d[0]) #output: 10
print(d[1]) #output: 10.5
```

Insert an element to the first position of a list

```
fruit = ["apple","mango","grapes"]
fruit.insert(0,"banana")
print(fruit) #output: ['banana', 'apple', 'mango', 'grapes']
```

💡 List is always mutable (changeable) that's why we add an element to add

Tuple

Tuple is the same as list it is also used to store multiple elements of different data types but the difference is tuple is immutable.

Example of a simple tuple

```
months = ("jan","feb","march")
print(months) #output: ('jan', 'feb', 'march')
print(type(months)) #output: <class 'tuple'>
```

print the first element of a tuple

```
months = ("jan","feb","march")
print(months[0]) #output: jan
```

💡 Tuple is immutable so if we try to insert an element it will throw an error

```
months = ("jan","feb","march")
months.insert(0,"april") #ERROR
```

String

String is a sequence of characters that are immutable

Example of a string

```
a = "hello world"
```

Example of a multi line string

```
e = """nishant
tiwari
age: 19
"""
```

Set

Set is used to store unique elements (set automatically removes duplicate elements).

Example of a set

```
f = {1,5,6,9,2,4,4}
print(f) #output: {1, 2, 4, 5, 6, 9}
print(type(f)) #output: <class 'set'>
```

Dictionary

Dictionary is a collection of key-value pairs

Example of a simple dictionary

```
#dictionary : key/value
name = {1:"nishant",2:"kartik"}
print(type(name)) #output: <class 'dict'>
print(name[1]) #output: nishant
```

Python Type Conversion

> Converting the value of one data type to another is known as type conversion

There are two types of type conversion

1) **Implicit Type Conversion:** In Implicit Type Conversion python automatically convert the value of data type to another

Example of Implicit Type Conversion

```
a = 10 #int
b = 20.3 #float
c = a + b #python automatically convert to float
'''
print(type(a)) #output: <class 'int'>
print(type(b)) #output: <class 'float'>
print(type(c)) #output: <class 'float'>
```

💡 **Remember this table**

```
int + int = int
int + float = float
float + float = float
float + int = int
str + int = error
str + float = error
```

2) **Explicit Type Conversion:** In Explicit Type Conversion user convert one data type to the required datatype

Example of Explicit Type Conversion

```
a = 100 #int
b = "40" #str
c = int(b) #convert to int
d = a + c #int + int = int
print(type(d)) #output: <class 'int'>
```

Python Input()

> Input() statement is used to take input from the user

Example of a Input()

```
a = input() #take input from the user and store it to a variable
```

Take two numbers from user

```
a = int(input("Enter first num: ")) #user input: 10
b = int(input("Enter second num: ")) #user input: 20
c = a + b #int
print(a+b) #Output: 30
```

Python Operator

> Operators are used to perform operations (eg: addition, compare two value, assign a value, etc)

Some types of Operators are :

- **Arithmetic Operator :** For mathematical operations **(**+, -, /, %)
- **Comparison Operator:** Compare two values (x > y, x < y, x != y)
- **Logical Operator:** For logical operations (and, or, not)
- **Assignment Operator:** Assign values to variables (a= 10, x = 20)

Python Operator: Arithmetic Operator

> Arithmetic Operator are used to perform mathematical operations **(addition, subtraction, multiplication, etc)**

Example of Arithmetic Operator

```
#take two variable and assign some values to it
a = 3 #first variable
b = 2 #second variable

#Addition
print(a+b) #output: 5

#Subtraction
print(a-b) #output: 1

#Multiplication
print(a*b) #output: 6

#Division
print(a/b) #output: 1.5

#Modulus: remainder of the division
print(a%b) #output: 1

#Floor division: result in whole number
print(a//b) #output: 1

#Exponent: power
print(a**b) #output: 9
```

Python Operator: Comparison Operators

> Comparison Operator are used to compare two values **(x is greater than y, x is less than y, etc)**

Example of Comparison Operators

```
#take two variable and assign some values to it
x = 10 #first variable
y = 20 #second variable

#x is greater than y
print(x > y) #output: false

#x is less than y
print(x < y) #output: true

#x is equal to y
print(x == y) #output: false

#x is not equal to y
print(x != y) #output: true

#x is greater than or equal to y
print(x >= y) #output: false

#x is less than or equal to y
print(x <= y) #output: true
```

Python Operator: Logical Operators

> Logical Operator are and, or, not operators and are used on conditional statements

💡 Truth table for and operator

```
True  and True  = True
True  and False = False
False and True  = False
False and False = False
```

💡 Truth table for or operator

```
True  or True  = True
True  or False = True
False or True  = True
False or False = False
```

💡 **Truth table for not operator**

```
True  = False
False = True
```

Example of Logical Operators

```
#take two variable and assign some values to it
x = True
y = False
print(x and y) #output: False
print(x or y) #output: True
print(not y) #output: True
```

Python Operator: Assignment Operator

> Assignment Operator are used to assign a value to a variables (x = 10, y = 20, etc)

Example of Assignment Operators

```
a = 5
print(a) #output: 5

#Reassigning value to a variable
a = a + 5
print(a) #output: 10

a = a - 5
print(a) #output: 5
```

💡 You can also write `a = a + 5` as `a += 5` and same for subtraction , multiplication ,division etc.

```
a = 5

#Reassigning value to a variable
a += 5
print(a) #output: 10

a -= 2
print(a) #output: 8
```

Python if/else statement

> If and else statement is used for decision making (If one statement isn't true then runs the next one)

Simple Example of if/else statement

```
#take two variable and assign some values to it
a = 20
b = 10
#first condition is false because a is less than 2
if a<b:
  print("a is less than b")
#second condition is true because a is greater than 2
elif a>b:
  print("a is greater than b")
#third condition is false because a is not equal to 2
else:
  print("a is equal to b")
```

💡 `elif` is short for else if (used for multiple expressions)

💡 `else` if no statement is true then at last else statement is executed

Python Program to Check if a number is Odd or Even (IF/ELSE)

This python program checks whether the number is even or odd. To check even/odd numbers we need to use the if/else statement. In the if condition we need to make sure the `input number` % `2` is always `0` (even) and for the else condition all the numbers are odd.

```
#create a variable and assign input()
num = int(input("Enter a number: "))
if (num % 2) == 0:
    print(num,"is an Even Number")
else:
    print(num,"is an Odd Number")
```

Python for loop

> For loop is used to iterate over a sequence of numbers, string, list, etc

Simple Example of for loop

```
#print the value from 0 to 4, and when the loop end, it prints hello world
for x in range(5):
    print(x)
print("hello world")
#output:
'''
0
1
2
3
4
hello world
'''

#print the value from 5 to 10, and when the loop end, it prints hello world
for x in range(5,21):
    print(x)
print("hello world")
#output:
'''
5
6
```

```
7
8
9
hello world
'''
```

Print the table of any number

```
#create a varaible and assign input()
num = int(input("Enter a num: ")) #user input: 2
#runs 1 to 10 times
for i in range(1,11):
    print(num*i)  #mutiply num by i
#output:
'''
2
4
6
8
10
12
14
16
18
20
'''
```

Iterating over lists with for loop

```
#create a list of fruits
fruit = ["apple","banana","grapes"]
#iterating each elements of list
for i in fruit:
    print(i)
#output:
'''
apple
banana
grapes
'''
```

💡 Generally, if you need to print an element of a list you need to use print() each time for each element. And, with the help of for loop, you can just directly iterating each element of a list.

Iterating over string with for loop

```
helloWorld = "hello world"
#iterating each characters of string
for i in helloWorld:
    print(i)
print("hello world")
#output:
'''
h
e
l
l
o

w
o
r
l
d
hello world
'''
```

Python while loop

> While loop is same as for loop, it runs until the condition is True

Simple Example of while loop

```
a = 1
#loop runs until the value of a is not greater than 9
while a < 10:
    print(a) #print the value of a each time until loop end
    a = a + 1 #adding the current value of a with 1
print("hello world") #when the loop end then (hello world) prints
#output:
'''
1
2
3
4
5
6
```

```
7
8
9
hello world
'''
```

Example of infinite loop

If the condition of while loop is always True, we get an infinite loop

```
#since while loop is always true
while True:
    print("hello world")
#output: infinite times hello world
```

💡If the loop is always False then nothing happens because while loop runs until the condition is true

```
#since while loop is always true
while False:
    print("hello world")
#output: nothing happen
```

Python Break & Continue Statement

> Break statement breaks the current loop containing it

Example of Break Statement

First, we iterate each character of a string inside a for loop then if the character (x) is equal to "o" it breaks the condition.

```
#python break
a = "hello"
for x in a:
```

```
  if x == "o":
    break
  print(x)
print("hello world")
#output:
'''
h
e
l
l
hello world
'''
```

> Continue Statement begin the next iteration of the loop

Example of Continue Statement

First, we iterate each character of a string inside a for loop then if the character (x) is equal to "l" it just continues to the next iteration.

```
#python continue
a = "hello"
for x in a:
  if x == "l":
    continue
  print(x)
print("hello world")
#output:
'''
h
e
o
hello world
'''
```

Python Numbers

There are three types of data type under numbers

1) **Integer (int) :** integer values such as (-1,0,1,2)

2) **Float :** a floating-point number (10.3, 13.001, 21.223)

3) **Complex:** expressed in the form x + yj (x is the real part and y is the imaginary part)

Example of integer

```
a = 5 #int (integer)
print(type(a)) #output: <class 'int'>
```

Example of float

```
b = 5.5 #float
print(type(b)) #output: <class 'float'>
```

Example of complex

```
c = 5 + 3j #complex (1+2j => x + yj)
print(type(c)) #output: <class 'complex'>
```

max() method

max() method return the highest value

Example of max() method

If we have three numbers then the min() method return the lowest number

```


print(max(80,90,100)) #output: 100
print(max(1000,-20,300)) #output: 1000
print(max(30,0,-1)) #output: 30
#negative numbers
print(max(-1,-2,-10)) #output: -1
#floating numbers
print(max(3.0,3.1,3.2)) #output: 3.2
'''
```

min() method

min() method return the lowest value

Example of max() method

If we have three numbers then the min() method return the lowest number

```
print(min(80,80,100)) #output: 80
print(min(30,40,50)) #output: 30
print(min(-1,-2,-3)) #output: -30
```

Python List

> List is used to store multiple elements of different data types. List is an ordered sequence of mutable elements.

Example of a empty list

```
a_list = []
print(a_list) #output: []
```

Example of list of integer values

```
a_int = [1,2,3,4,5]
print(a_int) #output: [1,2,3,4,5]
```

Example of list of float values

```
a_float = [1.0,2.2,3.4,3.5]
print(a_float) #output: [1.0,2.2,3.4,3.5]
```

Example of list of string values

```
a_str = ["apple","mango","grapes"]
print(a_str) #output: ["apple","mango","grapes"]
```

Example of list with mixed data types

```
a_mix = [1,1.0,"apple"]
print(a_mix) #output: [1,1.0,"apple"]
```

accessing the elements of a list

```
a_mix = [1,1.0,"apple"]
print(a_mix[0]) #output:1
print(a_mix[1]) #output: 1.0
print(a_mix[2]) #output: apple
```

Nested list

A nested list means list within a list

Example of Nested list

```
nest_list = ["hello",[1,2,3]]
#print the 1 of 0 element
print(nest_list[1][0]) #output: 1
#print the 1 of 1 element
print(nest_list[1][1]) #output: 2
#print the 1 of 2 element
print(nest_list[1][2]) #output: 3
```

Negative Indexing

Negative Indexing means index starts from the end

Example of Negative Indexing

```
#negative indexing
a = [1,2,3,4,5]
#print the last element of the list
print(a[-1]) #output: 5
```

```
#print the second last element of the list
print(a[-2]) #output: 4
```

find the length of a list

len() method prints the length of a list

```
#to find length of a list
a = [1,2,3,4,5,6,7,8,9,10]
#there are 10 elements in a list so length of a list will be 10
print(len(a)) #output: 10
```

Python List Methods

There are different types of list methods :

append()

append() is used to add the elements to the end of the list

Example of append()

```
fruit = ["apple","mango","grapes"]
print(fruit) #output: ["apple","mango","grapes"]
#append "banana" to the end of the list
fruit.append("banana")
print(fruit) #output: ["apple","mango","grapes","banana"]
```

extend()

extend() is used to merge two list

Example of extend()

```
lang = ["English","Hindi"]
lang1 = ["French","German"]
#add elements of lang1 to the end of lang list
lang.extend(lang1)
print(lang) #output: ['English', 'Hindi', 'French', 'German']
```

index()

index() is used to find the position of the element in a list

Example of index()

```
lang = ['English', 'Hindi', 'French', 'German']
#print the position of "French"
print(lang.index("French")) #output: 2
```

insert()

insert() is used to insert an element at a specified position

Example of insert()

```
lang = ["English","Hindi","French"]
#inserting "German" on 2 position
lang.insert(2,"German")
print(lang) #output: ['English', 'Hindi', 'German', 'French']
```

count()

Example of count()

count() is used to calculates the total existence of a given element of List.

```
list_int = [1,2,3,1,2,1,1,5]
#print the total occurrence of 1 in a list
print(list_int.count(1)) #output: 4
```

remove()

remove() is used to remove an element according to their element name

Example of remove()

```
lang = ["English","Hindi","French"]
#remove "French" from above list
lang.remove("French")
print(lang) #output: ["English","Hindi"]
```

pop()

pop() is also used to remove an element but according to their index position

Example of pop()

```
lang = ["English","Hindi","French"]
#remove "Hindi" from above list
lang.pop(1)
print(lang) #output: ['English','French']
```

reverse()

reverse() is used to reverse the element of a list

```
lang = ["English","Hindi","French"]
#reverse the element
lang.reverse()
print(lang) #output: ['French', 'Hindi', 'English']
```

sort()

sort() is used to sort the elements of a list in ascending order (by default)

Example of sort()

```
list_int = [1,5,4,7,8,3,2,1]
#sort in ascending order
list_int.sort()
print(list_int) #output: [1, 1, 2, 3, 4, 5, 7, 8]
```

Sort in descending order

```
list_int = [1,5,4,7,8,3,2,1]
#sort in descending order
list_int.sort(reverse=True)
print(list_int) #output: [8, 7, 5, 4, 3, 2, 1, 1]
```

Python Tuple

> Tuple is the same as list it is also used to store multiple items of different data types but the difference is tuple is immutable.

Example of a empty tuple

```
e_tuple = ()
print(e_tuple) #output: ()
```

Example of a simple list

```
months = ("jan","feb","march")
print(months) #output: ("jan","feb","march")
```

accessing the elements of a tuple

```
months = ("jan","feb","march")
print(months[0]) #output: jan
print(months[1]) #output: feb
print(months[2]) #output: march
```

Nested Tuple

Nested Tuple is same as Nested List

Example of Nested Tuple

```
n_tuple = ("hello",(1,2,3))
print(n_tuple[0])
#print the 1 of 1 element
print(n_tuple[1][1])
#print the 1 of 2 element
print(n_tuple[1][2])
```

Negative Indexing

Negative Indexing means index starts from the end
Example of Negative Indexing

```
e_tuple = (1,2,3)
#print the last element of the tuple
print(e_tuple[-1])
#print the second last element of the tuple
print(e_tuple[-2])
```

Python Set

> Set is used to store unique elements (set automatically removes duplicate elements)

Example of a empty set

```
my_set = {}
print(my_set) #output: {}
```

Example of a simple set

```
my_set = {1,2,9,6,8}
print(my_set) #output: {1, 2, 6, 8, 9}
```

Example of list with mixed data types

```
my_set = {1,1.0,"hello world",2,2}
print(my_set) #output: {'hello world', 1, 2}
```

Set does not support Indexing

```
a = {"a","b","c"}
print(a[0]) #Error: not support indexing
```

Add a element to a set

```
my_set = {1,2,9,6,8}
my_set.add(3)
print(my_set) #output: {1, 2, 3, 6, 8, 9}
```

Add a multiple element to a set

```
my_set = {1,2,9,6,8}
my_set.update([10,11,12])
print(my_set) #output: {1, 2, 6, 8, 9, 10, 11, 12}
```

Remove a element from a set

```
my_set = {1,2,9,6,8}
my_set.remove(2)
print(my_set) #output: {1, 6, 8, 9}
```

Set Operation

The set operations are performed on two or more sets to obtain a combination of elements

- **Union :** all elements which are in either A or B (or both)

```
a = {0,1,2}
b = {3,4,5}
print(a|b) #output: {0, 1, 2, 3, 4, 5}
```

- **Intersection :** all elements which are in both A and B

```
a = {0,1,2,3,4}
b = {0,2,4,9,8,7}
print(a&b) #output: {0, 2, 4}
```

Python Dictionary

> Dictionary is a collection of key-value pairs

Example of a simple dictionary

```
my_dict = {"fruit":"apple","veg":"carrot"}
print(my_dict) #output: {'fruit': 'apple', 'veg': 'carrot'}
```

accessing values of a dictionary

```
my_dict = {"fruit":"apple","veg":"carrot"}
print(my_dict["fruit"]) #output: apple
print(my_dict["veg"]) #output: carrot
```

print dictionary keys only

```
my_dict = {"fruit":"apple","veg":"carrot"}
#with the help of keys()
print(my_dict.keys()) #output: dict_keys(['fruit', 'veg'])
```

print dictionary values only

```
my_dict = {"fruit":"apple","veg":"carrot"}
#with the help of values()
print(my_dict.values()) #output: dict_values(['apple', 'carrot'])
```

add multiple values to a single key

```
my_new = {"fruit":["apple","mango","grapes"],"veg":["carrot","potato","tomato"]}
print(my_new)
#output: {'fruit': ['apple', 'mango', 'grapes'], 'veg': ['carrot', 'potato', 'tomato']}
```

accessing multiple values

```
my_new = {"fruit":["apple","mango","grapes"],"veg":["carrot","potato","tomato"]}
print(my_new["fruit"][0]) #output: apple
print(my_new["fruit"][1]) #output: mango
print(my_new["fruit"][2]) #output: grapes
```

Python Function

> Function are the block of code and it only runs when it called

Example of simple Function

```
def hello():
  print("my name is nishant")
#output: no output because function is not called
```

calling a function

```
def hello():
    print("my name is nishant")
hello() #function called
#output: my name is nishant
```

Adding numbers inside a function

```
#create a addnum function
def addnum():
    print(10+20+30)
addnum()
#output: 60
```

Passing values to a function

```
def hello(f,l):
    print(f,l)
fname = "Nishant"
lname = "Tiwari"
#passing fname and lname as a parameter to a function
hello(fname,lname)
#output: Nishant Tiwari
```

Python Types of Function

There are two types of function:

- **built-in function:** there are many function which are already available on python are known as built-in function. For example:

  `min()`, `max()`, `int()`, `float()`, `type()`, `len()`

- **user-defined function:** which are created by an user are known as user-defined function

  Example of user-defined function

```
def hello():
print("hello world")
hello()
```

Python Global and Local Variables

Global variable can be used anywhere in the program

Example of a Global Variable

```
x = 10 #old value: 10
def hello():
   #use of global variable inside a function
   global x
   print(x)
   #reassign a value to x
   x = 20 #new value: 20
hello() #first print the old value
print(x) # then the new value
#output:
'''
10
20
'''
```

Local variable are use within there scopes

Example of Local Variable

```
def hello():
  x = "hello" #local variable
  print(x)
hello() #output is hello
print(x) #error because local variable are use within there scopes
#output:
'''
hello
'x' is not defined
'''
```

Python Lambda & Anonymous Function

> Lambda function is also known as Anonymous Function are those which are defined without a name

Example of a simple Lambda Function

```
num = lambda x:x #return x
print(num(10)) #output: 10
```

Multiply a number using Lambda

```
multi = lambda x:x*2 #return x*2
print(multi(10)) #output: 20
```

Add two numbers using Lambda

```
#using lambda
add = lambda x,y:x+y #return x+y
print(add(2,55)) #output: 57

#using function
def add(a,b):
    return a+b
print(add(5,6)) #output: 57
```

Python Program to Check if a number is Odd or Even using Lambda

```
oddeven = lambda num: True if num%2 == 0 else False
print(oddeven(55)) #output: False
```

Python File Handling

> File handling is used to create, read, update, and delete files

opening a text file in python

With the help of the open() method, you can open any file in python

```
f = open("hello.txt")
```

reading a text file in python

With the help of the read() method, you can read any data of a file

```
#hello.txt
'''
hello world
'''
f = open("hello.txt","r") #"r" : Opens a file for reading
print(f.read()) #Output: hello world
f.close() #closes the opened file
```

💡 If you don't need to use `f.close()` you can also open a file as following :

```
#same as above example
with open("hello.txt","r") as f:
    print(f.read())
```

writing a text file in python

With the help of the write() method, you can write any data to a file

```
f = open("test.txt","w") #ppens a file for writing
f.write("hello world my name is nishant\n")
f.write("hello world\n")
f.close()
```

```
#This program will create a new file named hello.txt in the current directory
#hello.txt
'''
hello world my name is nishant
hello world
'''
```

Python Try Except

> Try Except statement is used to handle the error

Example of try except statement

```
#If try condition is false then except condition runs
try:
    print("hello")
except:
    print("hello world")
```

Example of else statement with try except

```
#If try condition is true then only else condition will run
try:
    print("hello")
except:
    print("hello world")
else:
    print("hello")
#output:
'''
hello
hello
'''

#If try condition is false then only except codition will run
try:
    print(x) #x is not define
except:
    print("hello world")
else:
    print("hello")
#output: hello world
```

Example of finally with try except

```
#Finally runs in each case either try is true or false
try:
   print("hello")
except:
   print("hello world")
finally:
   print("hello")
#output:
'''
hello
hello
'''
#If try condition is false then except and finally wil run
try:
    print(x) #x is not define
except:
    print("hello world")
finally:
    print("hello")
#output:
'''
hello world
hello
'''
```

Raise an exception

`raise` keyword is used to throw an exception for a condition if a condition occurs

```
#raise an exception
x = 5
if x < 11:
    raise Exception("Sorry, no number below 11")
output:
'''
Exception: Sorry, no number below 11
'''
```