## ***Python***

**Python** is a general-purpose language. It has wide range of applications from Web development (like: Django and Bottle), scientific and mathematical computing (Orange, SymPy, NumPy) to desktop graphical user Interfaces (Pygame, Panda3D).

There are two major Python versions, Python 2 and Python 3. Python 2 and 3 are quite different

For example, one difference between Python 2 and 3 is the print statement. In Python 2, the "print" statement is not a function, and therefore it is invoked without parentheses. However, in Python 3, it is a function, and must be invoked with parentheses

## ***Python Indentation***

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

A code block (body of a [function](https://www.programiz.com/python-programming/function), [loop](https://www.programiz.com/python-programming/for-loop) etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.

## ***Python Comments***

Any line starting with # in Python programming is a comment.

#This is a comment

#print out Hello

print('Hello')

## ***Multi-line comments***

Another way of doing this is to use triple quotes, either ''' or """.

These triple quotes are generally used for multi-line strings.

"""This is also a

perfect example of

multi-line comments"""

## ***Assigning multiple values to multiple variables***

a, b, c = 5, 3.2, "Hello"

print (a)

print (b)

print (c)

If we want to assign the same value to multiple variables at once, we can do this as:

x = y = z = "same"

print (x)

print (y)

print (z)

## ***Constants***

A constant is a type of variable whose value cannot be changed.

Example 3: Declaring and assigning value to a constant

Create a constant.py

PI = 3.14

GRAVITY = 9.8

Create a main.py

import constant

print(constant.PI)

print(constant.GRAVITY)

## ***Rules and Naming convention for variables and constants***

1. Create a name that makes sense. Suppose, vowel makes more sense than v.
2. Use camelCase notation to declare a variable. It starts with lowercase letter. For example:
   1. myName
   2. myAge
   3. myAddress
3. Use capital letters where possible to declare a constant. For example:
   1. PI
   2. G
   3. MASS

## ***Rules:***

Assignment uses = and comparison uses ==.

• For numbers + - \* / % \*\*,// are as expected.

• Special use of + for string concatenation.

• Special use of % for string formatting (as with printf in C)

• Logical operators are words (and, or, not)

not symbols

• The basic printing command is print.

• The first assignment to a variable creates it.

• Variable types don’t need to be declared.

• Python figures out the variable types on its own.

## ***Basic Datatypes:***

• Integers (default for numbers)

z = 5 / 2 # Answer is 2, integer division.

• Floats

x = 3.456

• Strings

• Can use “” or ‘’ to specify.

“abc” ‘abc’ (Same thing.)

• Unmatched can occur within the string.

“matt’s”

• Use triple double-quotes for multi-line strings or strings than contain both ‘

and “ inside of them:“““a‘b“c”””

## ***Whitespace:***

Whitespace is meaningful in Python: especially

indentation and placement of newlines.

• Use a newline to end a line of code.

• Use \ when must go to next line prematurely.

• No braces { } to mark blocks of code in Python…

Use consistent indentation instead.

• The first line with less indentation is outside of the block.

• The first line with more indentation starts a nested block

• Often a colon appears at the start of a new block.

(E.g. for function and class definitions.)

## ***Comments:***

• Start comments with # – the rest of line is ignored.

• Can include a “documentation string” as the first line of any

new function or class that you define.

• The development environment, debugger, and other tools use

it: it’s good style to include one.

def my\_function(x, y):

“““This is the docstring. This

function does blah blah blah.”””

# The code would go here...

## ***Assignment:***

• Binding a variable in Python means setting a name to hold a

reference to some object.

• Assignment creates references, not copies

• Names in Python do not have an intrinsic type. Objects have

types.

• Python determines the type of the reference automatically based on the

data object assigned to it.

• You create a name the first time it appears on the left side of

an assignment expression:

! x = 3

• A reference is deleted via garbage collection after any names

bound to it have passed out of scope.

## ***Multiple Assignment***

• You can also assign to multiple names at the same time.

>>> x, y = 2, 3

>>> x

2

>>> y

3

**In Python, the datatypes integer, float, and string (and tuple) are**

“immutable.”

• This doesn’t mean we can’t change the value of x, i.e. change

what x refers to …

• For example, we could increment x:

>>> x = 3

>>> x = x + 1

>>> print x

4

If we increment x, then what’s really happening is:

1. The reference of name x is looked up.

2. The value at that reference is retrieved.

3. The 3+1 calculation occurs, producing a new data element 4 which is

assigned to a fresh memory location with a new reference.

4. The name x is changed to point to this new reference.

5. The old data 3 is garbage collected if no name still refers to it.

Tuples are defined using parentheses (and commas).

>>> tu = (23, ‘abc’, 4.56, (2,3), ‘def’)

• Lists are defined using square brackets (and commas).

>>> li = [“abc”, 34, 4.34, 23]

• Strings are defined using quotes (“, ‘, or “““).

>>> st = “Hello World”

>>> st = ‘Hello World’

>>> st = “““This is a multi-line

string that uses triple quotes.”””

## ***Functions:***

• def creates a function and assigns it a name

• return sends a result back to the caller

• Arguments are passed by assignment

• Arguments and return types are not declared

def <name>(arg1, arg2, ..., argN):

<statements>

return <value>

def times(x,y):

return x\*y

## ***Example 1:***

x = 3

y = 5

print(’The sum of’, x, ’plus’, y, ’is’, x+y)

## ***The input Function***

## ***Example 2:***

hello\_you.py.

-------------------

person = input(’Enter your name: ’)

print(’Hello’, person)

## ***Example 3:***

A Code Sample

x = 34 – 23 # A comment.

y = “Hello” # Another one.

z = 3.45

if z == 3.45 or y == “Hello”:

x = x + 1

y = y + “ World” # String concat.

print x

print y

## ***Example 4:***

applicant = input("Enter the applicant’s name: ")

interviewer = input("Enter the interviewer’s name: ")

time = input("Enter the appointment time: ")

print(interviewer, "will interview", applicant, "at", time)

## ***Example 5:***

Consider the following problem: Prompt the user for two numbers, and then print out a sentence stating the sum. For

instance if the user entered 2 and 3, you would print ‘The sum of 2 and 3 is 5.’

You might imagine a solution like the example file addition1.py, shown below. There is a problem. Can you

figure it out before you try it? Hint: 4

’’’Error in addition from input.’’’

x = input("Enter a number: ")

y = input("Enter a second number: ")

print(’The sum of ’, x, ’ and ’, y, ’ is ’, x+y, ’.’, sep=’’) #error

End up running it in any case.

We do not want string concatenation, but integer addition. We need integer operands. Briefly mentioned in Whirlwind

Introduction To Types and Functions (page 12) was the fact that we can use type names as functions to convert types.

One approach would be to do that. Further variable names are also introduced in the example addition2.py file

below to emphasize the distinctions in types. Read and run:

’’’Conversion of strings to int before addition’’’

xString = input("Enter a number: ")

x = int(xString)

yString = input("Enter a second number: ")

y = int(yString)

print(’The sum of ’, x, ’ and ’, y, ’ is ’, x+y, ’.’, sep=’’)

Needing to convert string input to numbers is a common situation, both with keyboard input and later in web pages.

While the extra variables above emphasized the steps, it is more concise to write as in the variation in example file,

addition3.py, doing the conversions to type int immediately:

’’’Two numeric inputs, with immediate conversion’’’

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

y = int(input("Enter a second number: "))

print(’The sum of ’, x, ’ and ’, y, ’ is ’, x+y, ’.’, sep=’’)

The simple programs so far have followed a basic programming pattern: input-calculate-output. Get all the data first,

calculate with it second, and output the results last. The pattern sequence would be even clearer if we explicitly create

a named result variable in the middle, as in addition4.py

’’’Two numeric inputs, explicit sum’’’

x = int(input("Enter an integer: "))

y = int(input("Enter another integer: "))

sum = x+y

print(’The sum of ’, x, ’ and ’, y, ’ is ’, sum, ’.’, sep=’’)

## ***MORE CONTROL FLOW TOOLS***

Besides the while statement just introduced, Python knows the usual control flow statements known from other

languages, with some twists.

## ***if Statements***

Perhaps the most well-known statement type is the if statement. For example:

>>> x = int(input("Please enter an integer: "))

Please enter an integer: 42

>>> if x < 0:

... x = 0

... print(’Negative changed to zero’)

... elif x == 0:

... print(’Zero’)

... elif x == 1:

... print(’Single’)

... else:

... print(’More’)

...

More

There can be zero or more elif parts, and the else part is optional. The keyword ‘elif‘ is short for ‘else if’, and

is useful to avoid excessive indentation. An if ... elif ... elif ... sequence is a substitute for the switch or case

statements found in other languages.

## ***for Statements***

The for statement in Python differs a bit from what you may be used to in C or Pascal. Rather than always iterating

over an arithmetic progression of numbers (like in Pascal), or giving the user the ability to define both the iteration

step and halting condition (as C), Python’s for statement iterates over the items of any sequence (a list or a string), in

the order that they appear in the sequence. For example (no pun intended):

>>> # Measure some strings:

... a = [’cat’, ’window’, ’defenestrate’]

>>> for x in a:

... print(x, len(x))

...

cat 3

window 6

defenestrate 12

It is not safe to modify the sequence being iterated over in the loop (this can only happen for mutable sequence types,

such as lists). If you need to modify the list you are iterating over (for example, to duplicate selected items) you must

iterate over a copy. The slice notation makes this particularly convenient:

>>> for x in a[:]: # make a slice copy of the entire list

... if len(x) > 6: a.insert(0, x)

...

>>> a

[’defenestrate’, ’cat’, ’window’, ’defenestrate’]

## ***The range() Function***

If you do need to iterate over a sequence of numbers, the built-in function range() comes in handy. It generates

arithmetic progressions:

>>> for i in range(5):

... print(i)

...

0

1

2

3

4

The given end point is never part of the generated sequence; range(10) generates 10 values, the legal indices for

items of a sequence of length 10. It is possible to let the range start at another number, or to specify a different

increment (even negative; sometimes this is called the ‘step’):

range(5, 10)

5 through 9

range(0, 10, 3)

0, 3, 6, 9

range(-10, -100, -30)

-10, -40, -70

To iterate over the indices of a sequence, you can combine range() and len() as follows:

>>> a = [’Mary’, ’had’, ’a’, ’little’, ’lamb’]

>>> for i in range(len(a)):

... print(i, a[i])

...

0 Mary

1 had

2 a

3 little

4 lamb

In most such cases, however, it is convenient to use the enumerate() function, see Looping Techniques.

A strange thing happens if you just print a range:

>>> print(range(10))

range(0, 10)

In many ways the object returned by range() behaves as if it is a list, but in fact it isn’t. It is an object which returns

the successive items of the desired sequence when you iterate over it, but it doesn’t really make the list, thus saving

space.

We say such an object is iterable, that is, suitable as a target for functions and constructs that expect something from

which they can obtain successive items until the supply is exhausted. We have seen that the for statement is such an

iterator. The function list() is another; it creates lists from iterables:

>>> list(range(5))

[0, 1, 2, 3, 4]

Later we will see more functions that return iterables and take iterables as argument.

## ***Lists:***

Lists are very similar to arrays. They can contain any type of variable, and they can contain as many variables as you wish. Lists can also be iterated over in a

very simple manner.

mylist = []

mylist.append(1)

mylist.append(2)

mylist.append(3)

print(mylist[0]) # prints 1

print(mylist[1]) # prints 2

print(mylist[2]) # prints 3

# prints out 1,2,3

for x in mylist:

print(x)

mylist = [1,2,3]

print(mylist[1])

## ***break and continue Statements, and else Clauses on Loops***

count = 0

while True:

print(count)

count += 1

if count >= 5:

break

# Prints out only odd numbers - 1,3,5,7,9

for x in range(10):

# Check if x is even

if x % 2 == 0:

continue

print(x)

## ***Defining Functions***

Functions in python are defined using the block keyword "def", followed with the function's name as the block's name.

For example:

def my\_function():

print("Hello From My Function!")

def sum\_two\_numbers(a, b):

return a + b

# print(a simple greeting)

my\_function()

# after this line x will hold the value 3!

x = sum\_two\_numbers(1,2)

We can create a function that writes the Fibonacci series to an arbitrary boundary:

>>> def fib(n): # write Fibonacci series up to n

... """Print a Fibonacci series up to n."""

... a, b = 0, 1

... while a < n:

... print(a, end=’ ’)

... a, b = b, a+b

... print()

...

>>> # Now call the function we just defined:

... fib(2000)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597

The keyword **def** introduces a function definition. It must be followed by the function name and the parenthesized list

of formal parameters. The statements that form the body of the function start at the next line, and must be indented.

## ***Examples:***

1.0

a=5

b=2

c=int(a)\*\*2

d=int(b)\*\*3

print("Square of a is:",c) #Square of a is: 25

print("Cube of b is:",d)#Cube of b is: 8

e=int(a)/int(b)

print("e is:",e)#e is: 2.5

f=int(a)//int(b)

print("f is:",f)#f is: 2

1.1

x = 5

y = 8

if x > 55:

print('x is greater than 55')

else:

print('x is not greater than’)

1.2

x = 5

y = 10

if x > y:

print('x is greater than y')

1.3

x=16;

y=9;

if x>y:

print("x is greater than y");

else:

print("y is greater than x");

2.

x = 5

y = 10

z = 22

if x > y:

print(“x is greater than y”);

elif x > z:

print(“x is greater than z”);

else:

print(“if and elif never ran...”);

3.

if x == 3:

print (“X equals 3.”);

elif x == 2:

print (“X equals 2.”);

else:

print (“X equals something else.”);

print (“This is outside the ‘if’.”);

-------------------------------------------------------------------------

4.

x = 3

while x < 10:

if x > 7:

x += 2

continue

x = x + 1

print (“Still in the loop.”);

if x == 8:

break

print (“Outside of the loop.”);

4.1

a=2

b=4

c=8

d=int(a)+int(b)+int(c)

print("Sum is:",d)

if int(a)>int(b)and int(a)>int(c) :

print("a is biggest")

elif int(b)>int(a)and int(b)>int(c) :

print("b is biggest")

else:

print("c is biggest")

5:

exampleList = [1,5,6,6,2,1,5,2,1,4]

for x in exampleList:

print(x)

5.1:

for x in range(10):

if x > 7:

x += 2

continue

x = x + 1

print (“Still in the loop.”);

if x == 8:

break

print (“Outside of the loop.”);

5.2:

**For loop Syntax:**

# Prints out the numbers 0,1,2,3,4

for x in range(5):

print(x)

# Prints out 3,4,5

for x in range(3, 6):

print(x)

# Prints out 3,5,7

for x in range(3, 8, 2):

print(x)

5.3:

**while loop Syntax:**

While loops repeat as long as a certain boolean

Condition met:

For example:

"while" loops:

count = 0

while count < 5:

print(count)

count += 1 # This is the same as count = count + 1

## ***Import Uses:***

**Calculate the perimeter of a circle**

>>> from math import \*

>>> diameter = 5

>>> perimeter = 2 \* pi \* diameter

>>> perimeter

31.41592653589793

**Calculate the amplitude of a sine wave:**

>>> from math import \*

>>> Ueff = 230

>>> amplitude = Ueff \* sqrt(2)

>>> amplitude

325.2691193458119

**Source Code: Add Two Numbers Provided by The User**

# Store input numbers

num1 = input('Enter first number: ')

num2 = input('Enter second number: ')

# Add two numbers

sum = float(num1) + float(num2)

# Display the sum

print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))

**Output**

Enter first number: 1.5

Enter second number: 6.3

The sum of 1.5 and 6.3 is 7.8

# **Python Program to Calculate the Area of a Triangle**

# Python Program to find the area of triangle

a = 5

b = 6

c = 7

# Uncomment below to take inputs from the user

# a = float(input('Enter first side: '))

# b = float(input('Enter second side: '))

# c = float(input('Enter third side: '))

# calculate the semi-perimeter

s = (a + b + c) / 2

# calculate the area

area = (s\*(s-a)\*(s-b)\*(s-c)) \*\* 0.5

print('The area of the triangle is %0.2f' %area)

# **Python Program to Find the Largest Among Three Numbers**

# Python program to find the largest number among the three input numbers

# change the values of num1, num2 and num3

# for a different result

num1 = 10

num2 = 14

num3 = 12

# uncomment following lines to take three numbers from user

#num1 = float(input("Enter first number: "))

#num2 = float(input("Enter second number: "))

#num3 = float(input("Enter third number: "))

if (num1 >= num2) and (num1 >= num3):

largest = num1

elif (num2 >= num1) and (num2 >= num3):

largest = num2

else:

largest = num3

print("The largest number between",num1,",",num2,"and",num3,"is",largest)

## Source Code: Simple Caculator by Making Functions

# Program make a simple calculator that can add, subtract, multiply and divide using functions

# This function adds two numbers

def add(x, y):

return x + y

# This function subtracts two numbers

def subtract(x, y):

return x - y

# This function multiplies two numbers

def multiply(x, y):

return x \* y

# This function divides two numbers

def divide(x, y):

return x / y

print("Select operation.")

print("1.Add")

print("2.Subtract")

print("3.Multiply")

print("4.Divide")

# Take input from the user

choice = input("Enter choice(1/2/3/4):")

num1 = int(input("Enter first number: "))

num2 = int(input("Enter second number: "))

if choice == '1':

print(num1,"+",num2,"=", add(num1,num2))

elif choice == '2':

print(num1,"-",num2,"=", subtract(num1,num2))

elif choice == '3':

print(num1,"\*",num2,"=", multiply(num1,num2))

elif choice == '4':

print(num1,"/",num2,"=", divide(num1,num2))

else:

print("Invalid input")

**Output**

Select operation.

1.Add

2.Subtract

3.Multiply

4.Divide

Enter choice(1/2/3/4): 3

Enter first number: 15

Enter second number: 14

15 \* 14 = 210

# **Python Program to Find the Factorial of a Number**

# Python program to find the factorial of a number provided by the user.

# change the value for a different result

num = 7

# uncomment to take input from the user

#num = int(input("Enter a number: "))

factorial = 1

# check if the number is negative, positive or zero

if num < 0:

print("Sorry, factorial does not exist for negative numbers")

elif num == 0:

print("The factorial of 0 is 1")

else:

for i in range(1,num + 1):

factorial = factorial\*i

print("The factorial of",num,"is",factorial)

# **Python Program to Check Prime Number**

# Python program to check if the input number is prime or not

num = 407

# take input from the user

# num = int(input("Enter a number: "))

# prime numbers are greater than 1

if num > 1:

# check for factors

for i in range(2,num):

if (num % i) == 0:

print(num,"is not a prime number")

print(i,"times",num//i,"is",num)

break

else:

print(num,"is a prime number")

# if input number is less than

# or equal to 1, it is not prime

else:

print(num,"is not a prime number")

# **Python Program to Print all Prime Numbers in an Interval**

# Python program to display all the prime numbers within an interval

# change the values of lower and upper for a different result

lower = 900

upper = 1000

# uncomment the following lines to take input from the user

#lower = int(input("Enter lower range: "))

#upper = int(input("Enter upper range: "))

print("Prime numbers between",lower,"and",upper,"are:")

for num in range(lower,upper + 1):

# prime numbers are greater than 1

if num > 1:

for i in range(2,num):

if (num % i) == 0:

break

else:

print(num)

# **Python Program to Find the Sum of Natural Numbers**

# Python program to find the sum of natural numbers up to n where n is provided by user

# change this value for a different result

num = 16

# uncomment to take input from the user

#num = int(input("Enter a number: "))

if num < 0:

print("Enter a positive number")

else:

sum = 0

# use while loop to iterate un till zero

while(num > 0):

sum += num

num -= 1

print("The sum is",sum)

# **Python Program to Find Armstrong Number in an Interval**

lower = 100

upper = 2000

# To take input from the user

# lower = int(input("Enter lower range: "))

# upper = int(input("Enter upper range: "))

for num in range(lower, upper + 1):

# order of number

order = len(str(num))

# initialize sum

sum = 0

# find the sum of the cube of each digit

temp = num

while temp > 0:

digit = temp % 10

sum += digit \*\* order

temp //= 10

if num == sum:

print(num)

# **Python Program to Check Armstrong Number**

# Python program to check if the number provided by the user is an Armstrong number or not

# take input from the user

# num = int(input("Enter a number: "))

# initialize sum

sum = 0

# find the sum of the cube of each digit

temp = num

while temp > 0:

digit = temp % 10

sum += digit \*\* 3

temp //= 10

# display the result

if num == sum:

print(num,"is an Armstrong number")

else:

print(num,"is not an Armstrong number")

# **Python Program to Count the Number of Each Vowel**

# Program to count the number of each vowel in a string

# string of vowels

vowels = 'aeiou'

# change this value for a different result

ip\_str = 'Hello, have you tried our turorial section yet?'

# uncomment to take input from the user

#ip\_str = input("Enter a string: ")

# make it suitable for caseless comparisions

ip\_str = ip\_str.casefold()

# make a dictionary with each vowel a key and value 0

count = {}.fromkeys(vowels,0)

# count the vowels

for char in ip\_str:

if char in count:

count[char] += 1

print(count)

# **Python Program to Check Whether a String is Palindrome or Not**

# Program to check if a string

# is palindrome or not

# change this value for a different output

my\_str = 'aIbohPhoBiA'

# make it suitable for caseless comparison

my\_str = my\_str.casefold()

# reverse the string

rev\_str = reversed(my\_str)

# check if the string is equal to its reverse

if list(my\_str) == list(rev\_str):

print("It is palindrome")

else:

print("It is not palindrome")

# **Python Program to Print the Fibonacci sequence**

# Program to display the Fibonacci sequence up to n-th term where n is provided by the user

# change this value for a different result

nterms = 10

# uncomment to take input from the user

#nterms = int(input("How many terms? "))

# first two terms

n1 = 0

n2 = 1

count = 0

# check if the number of terms is valid

if nterms <= 0:

print("Please enter a positive integer")

elif nterms == 1:

print("Fibonacci sequence upto",nterms,":")

print(n1)

else:

print("Fibonacci sequence upto",nterms,":")

while count < nterms:

print(n1,end=' , ')

nth = n1 + n2

# update values

n1 = n2

n2 = nth

count += 1

# **Python Program to Find the Sum of Natural Numbers**

# Python program to find the sum of natural numbers up to n where n is provided by user

# change this value for a different result

num = 16

# uncomment to take input from the user

#num = int(input("Enter a number: "))

if num < 0:

print("Enter a positive number")

else:

sum = 0

# use while loop to iterate un till zero

while(num > 0):

sum += num

num -= 1

print("The sum is",sum)

**Output**

The sum is 136

# **Python Program to Check if a Number is Positive, Negative or 0**

num = float(input("Enter a number: "))

if num > 0:

print("Positive number")

elif num == 0:

print("Zero")

else:

print("Negative number")

num = float(input("Enter a number: "))

if num >= 0:

if num == 0:

print("Zero")

else:

print("Positive number")

else:

print("Negative number")

# **Python Program to Check if a Number is Odd or Even**

num = int(input("Enter a number: "))

if (num % 2) == 0:

print("{0} is Even".format(num))

else:

print("{0} is Odd".format(num))

# [**Python Program to Find the Square Root**](https://www.programiz.com/python-programming/examples/square-root)

## For positive numbers using exponent \*\*

# Python Program to calculate the square root

# Note: change this value for a different result

num = 8

# uncomment to take the input from the user

#num = float(input('Enter a number: '))

num\_sqrt = num \*\* 0.5

print('The square root of %0.3f is %0.3f'%(num ,num\_sqrt))

# **Python Program to Convert Celsius To Fahrenheit**

# Python Program to convert temperature in celsius to fahrenheit

# change this value for a different result

celsius = 37.5

# calculate fahrenheit

fahrenheit = (celsius \* 1.8) + 32

print('%0.1f degree Celsius is equal to %0.1f degree Fahrenheit' %(celsius,fahrenheit))

# **Python Program to Check Leap Year**

# Python program to check if the input year is a leap year or not

year = 2000

# To get year (integer input) from the user

# year = int(input("Enter a year: "))

if (year % 4) == 0:

if (year % 100) == 0:

if (year % 400) == 0:

print("{0} is a leap year".format(year))

else:

print("{0} is not a leap year".format(year))

else:

print("{0} is a leap year".format(year))

else:

print("{0} is not a leap year".format(year))

# **Python Program to Add Two Matrices**

# Program to add two matrices using nested loop

X = [[12,7,3],

[4 ,5,6],

[7 ,8,9]]

Y = [[5,8,1],

[6,7,3],

[4,5,9]]

result = [[0,0,0],

[0,0,0],

[0,0,0]]

# iterate through rows

for i in range(len(X)):

# iterate through columns

for j in range(len(X[0])):

result[i][j] = X[i][j] + Y[i][j]

for r in result:

print(r)

# **Python Program to Transpose a Matrix**

# Program to transpose a matrix using nested loop

X = [[12,7],

[4 ,5],

[3 ,8]]

result = [[0,0,0],

[0,0,0]]

# iterate through rows

for i in range(len(X)):

# iterate through columns

for j in range(len(X[0])):

result[j][i] = X[i][j]

for r in result:

print(r)

## ***1.Python Object Oriented Programming:***

**Class Definition Syntax**

The simplest form of class definition looks like this:

class ClassName:

<statement-1>

.

.

.

<statement-N>

class MyClass:

"""A simple example class"""

i = 12345

def f(self):

return ’hello world’

Class instantiation uses function notation. Just pretend that the class object is a parameterless function that returns a

new instance of the class. For example (assuming the above class):

x = MyClass()

creates a new instance of the class and assigns this object to the local variable x.

When a class defines an \_\_init\_\_() method, class instantiation automatically invokes \_\_init\_\_() for the

newly-created class instance.

## ***Example1:***

class Employee:

id=1;

name="sss";

def display(self):

print("id is",self.id);

print("Name is",self.name);

emp= Employee();

emp.display();

## ***Example2:***

# define the Vehicle class

class Vehicle:

name = "";

kind = "car";

color = "";

value = 100.00;

def description(self):

desc\_str = "%s is a %s %s worth $%.2f." % (self.name, self.color, self.kind, self.value);

return desc\_str;

# your code goes here

car1=Vehicle();

car2=Vehicle();

# test code

print(car1.description());

print(car2.description());

In python, the method double underscore (\_\_\_) \_\_init\_\_ simulates the constructor of the class. This method is called when

the class is instantiated. We can pass any number of arguments at the time of creating the class object, depending upon

\_\_init\_\_ definition. It is mostly used to initialize the class attributes. Every class must have a constructor, even if it

simply relies on the default constructor.

## ***2.Constructor:***

## ***Python Non-Parameterized Constructor Example:***

class Student:

def \_\_init\_\_(self):

print("This is non parameterized constructor")

def display(self,name):

print("Name is::",name);

ss=Student();

ss.display("Sanjoy");

## ***Python Parameterized Constructor Example:***

## ***Example 1:***

class Student:

def \_\_init\_\_(self,id,name):

print("This is parameterized constructor")

self.id=id;

self.name=name;

def display(self):

print("Id is::",self.id);

print("Name is::",self.name);

ss=Student(1,"Sanjoy");

pp=Student(2,"Pradip");

ss.display();

pp.display();

## ***Example 2:***

class Rectangle:

def \_\_init\_\_(self, length,breadth):

self.length = length

self.breadth = breadth

def area(self):

print("Area is,int(self.length) \* int(self.breadth)")

x = Rectangle(10,20)

x.area()

## ***Python Inheritance:***

## ***Single level Inheritance:***

class A:

def m1(self):

print("Calling m1 method of A Class");

class B(A):

def m2(self):

print("Calling m2 method of B Class");

bb=B();

bb.m1();

bb.m2();

## ***Multilevel Inheritance:***

class A:

def m1(self):

print("Calling m1 method of A Class");

class B(A):

def m2(self):

print("Calling m2 method of B Class");

class C(B):

def m3(self):

print("Calling m3 method of C Class");

cc=C();

cc.m1();

cc.m2();

cc.m3();

## ***Method Overiding:***

class T:

def display(self):

print("XXX is called");

class R(T):

def display(self):

print("YYY is called");

rr=R();

rr.display();