**INTRODUCTION**

**BRIEF ON RELEVANCE OF PYTHON:**

**Relevance of Python Programming:**

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python was designed to be highly readable which uses English keywords frequently whereas other languages use punctuation and it has fewer syntactical constructions than other languages.

**Python is Interpreted -** It means that it is processed at runtime by the interpreter and you do not need to compile your program before executing it.

**Python is Interactive -** It means that you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

**Python is Object-Oriented -** It means that Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

**Python is Beginner's Language -** Python is a great language for the beginner programmers and supports the development of a wide range of applications from simple text processing to www browsers to games.

**Portable -** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

**Extendable -** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

**Databases -** Python provides interfaces to all major commercial databases.

**GUI Programming -** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh and the X Window system of Unix.

**Scalable -** Python provides a better structure and support for large programs than shell scripting.

**HOW TO INSTALL AND EXECUTE PYTHON:**

**To install python:**

Installing Python on Windows takes a series of few easy steps.

## **Step 1 − Select Version of Python to Install**

Python has various versions available with differences between the syntax and working of different versions of the language. We need to choose the version which we want to use or need. There are different versions of Python 2 and Python 3 available.

## **Step 2 − Download Python Executable Installer**

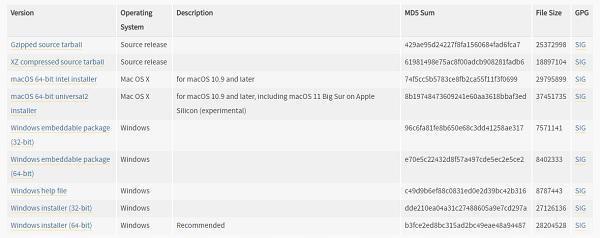
On the web browser, in the official site of python ([www.python.org](https://www.tutorialspoint.com/www.python.org)), move to the Download for Windows section.

All the available versions of Python will be listed. Select the version required by you and click on Download. Let suppose, we chose the Python 3.9.1 version.



On clicking download, various available executable installers shall be visible with different operating system specifications. Choose the installer which suits your system operating system and download the instlaller. Let suppose, we select the Windows installer(64 bits).

The download size is less than 30MB.



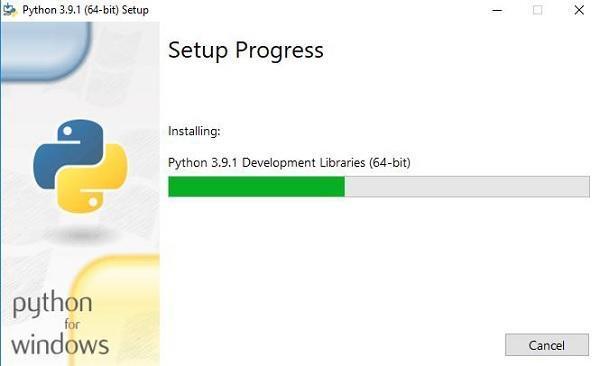
## **Step 3 − Run Executable Installer**

We downloaded the Python 3.9.1 Windows 64 bit installer.

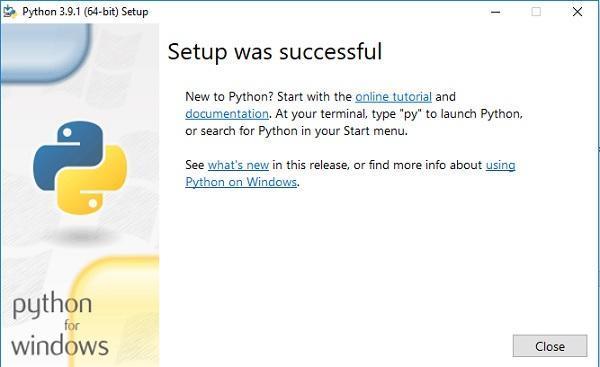
Run the installer. Make sure to select both the checkboxes at the bottom and then click Install New.



On clicking the Install Now, The installation process starts.



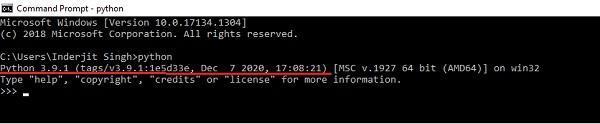
The installation process will take few minutes to complete and once the installation is successful, the following screen is displayed.



## **Step 4 − Verify Python is installed on Windows**

To ensure if Python is succesfully installed on your system. Follow the given steps −

* Open the command prompt.
* Type ‘python’ and press enter.
* The version of the python which you have installed will be displayed if the python is successfully installed on your windows.

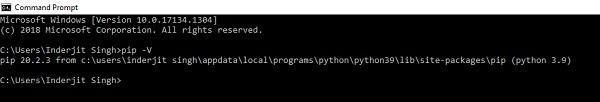


## **Step 5 − Verify Pip was installed**

Pip is a powerful package management system for Python software packages. Thus, make sure that you have it installed.

To verify if pip was installed, follow the given steps −

* Open the command prompt.
* Enter pip –V to check if pip was installed.
* The following output appears if pip is installed successfully.



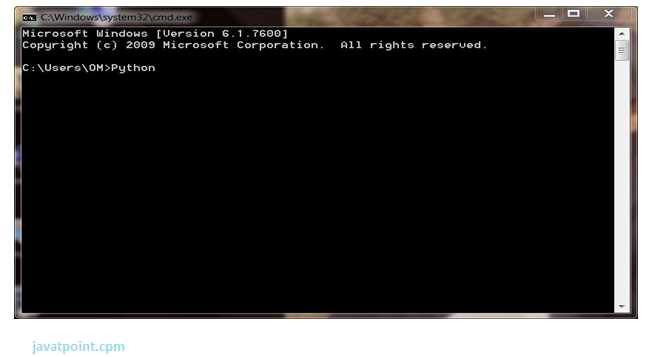
We have successfully installed python and pip on our Windows system.

**To execute python code:**

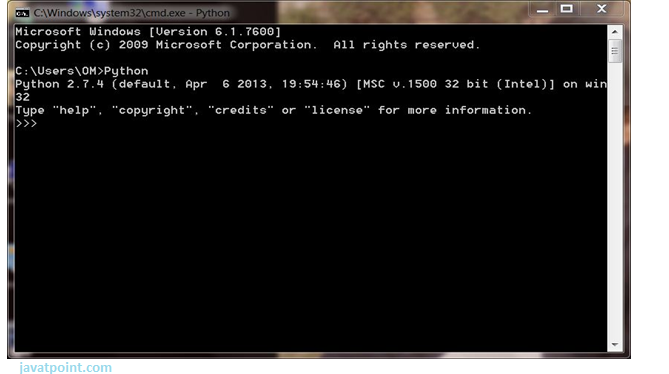
To execute Python code, we can use any approach that are given below.

## **1) Interactive Mode**

Python provides Interactive Shell to execute code immediatly and produce output instantly. To get into this shell, write python in the command prompt and start working with Python.

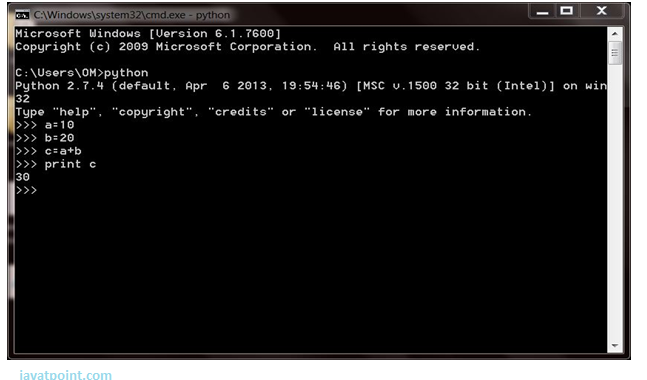


Press Enter key and the Command Prompt will appear like:



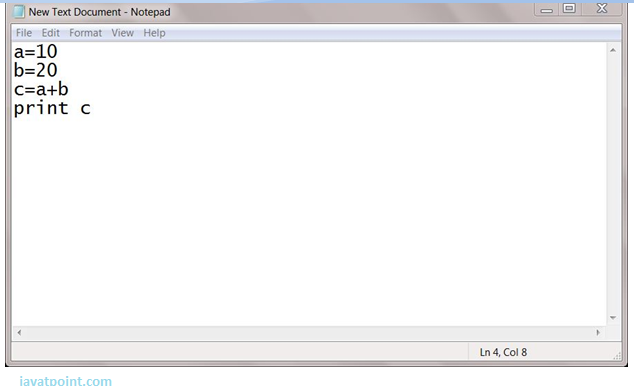
Now we can execute our Python commands.

**Eg:**

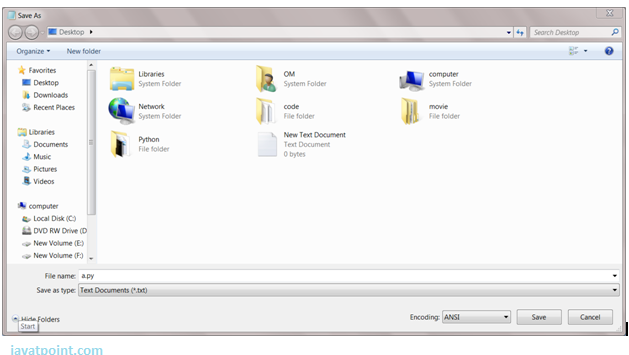
****

## **2) Script Mode**

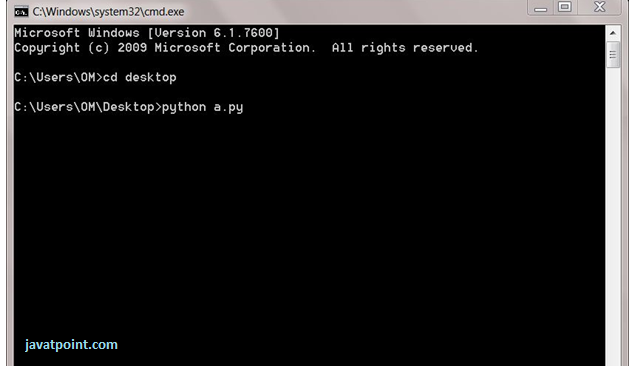
## Using Script Mode, we can write our Python code in a separate file of any editor in our Operating System.



Save it by **.py** extension.



Now open Command prompt and execute it by :



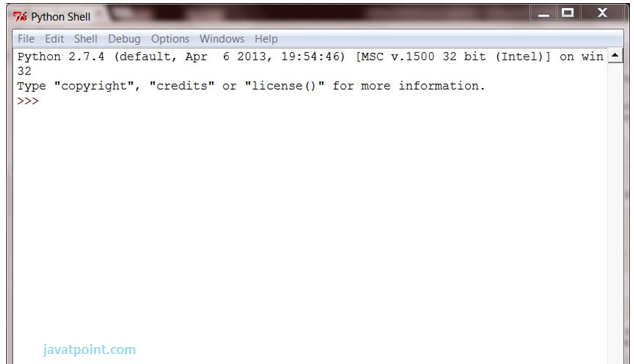
NOTE: Path in the command prompt should be the location of the saved file.where you have saved your file. In the above case the file should be saved at the desktop.

## 3) **Using IDE (Integrated Development Environment)**

We can execute our Python code using a Graphical User Interface (GUI).

All you need to do is:

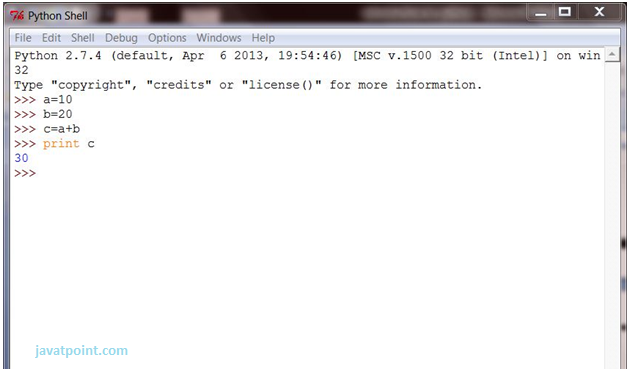
Click on Start button -> All Programs -> Python -> IDLE(Python GUI)



We can use both Interactive as well as Script mode in IDE.

**1) Using Interactive mode:**

Execute our Python code on the Python prompt and it will display result simultaneously.

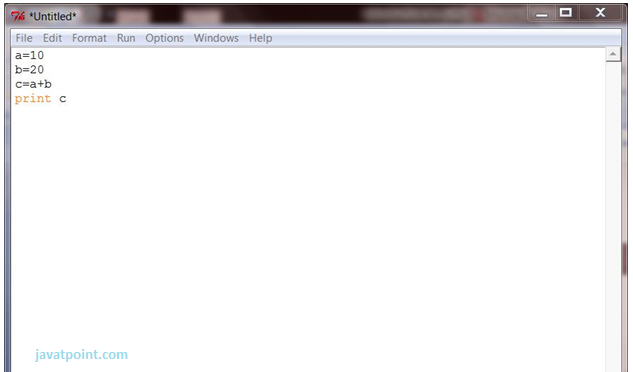


**2) Using Script Mode:**

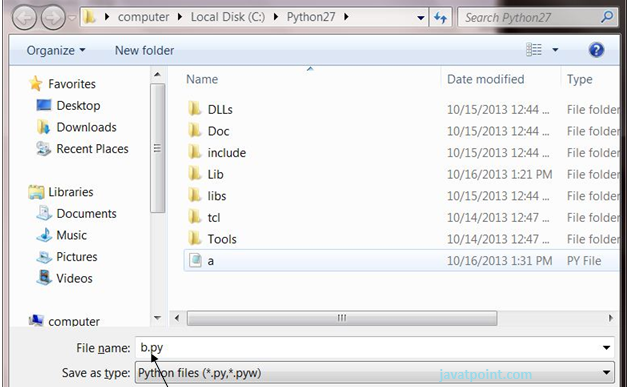
i) Click on Start button -> All Programs -> Python -> IDLE(Python GUI)

ii) Python Shell will be opened. Now click on File -> New Window.

A new Editor will be opened. Write our Python code here.



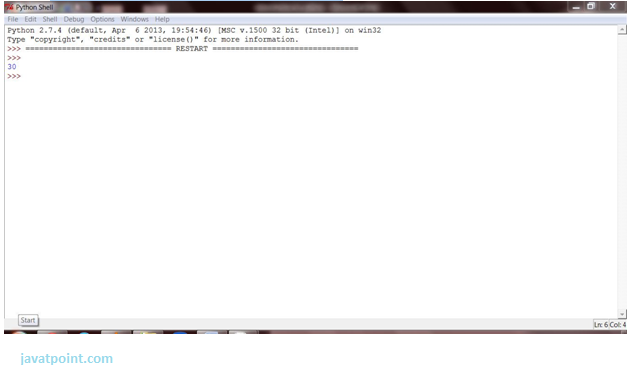
Click on file -> save as



Run code by clicking on Run in the Menu bar.

Run -> Run Module

Result will be displayed on a new Python shell as:



**DATA TYPE AND OPERATORS IN PYTHON PROGRAMMING :**

**Datatypes in python:**

Datatypes represents the type useful that tells what operations are often performed on specific data.

Python has six standard Data Types:-

### **Numeric Data Type:-**

In Python, numeric data type represents the data that has a numeric value. The numeric value can be an integer, floating number, or even complex number. These values are defined as int, float, and complex classes in Python.

Integers – This data type is represented with the help of int class. It consists of positive or negative whole numbers (without fraction or decimal). In Python, there’s no limit to how long integer values are often.

Example:-

a = 2

print(a, "is of type", type(a))

Output: 2 is of type

Float – This type is represented by the float class. It is a true number with floating-point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer could even be appended to specify scientific notation.

Example:-

b = 1.5

print(b, "is of type", type(b))

Output: 1.5 is of type

Complex Numbers – Complex numbers are represented by complex classes. It is specified as (real part) + (imaginary part)j, For example – 4+5j.

Example:-

c = 8+3j

print(c, "is a type", type(c))

Output: (8+3j) is a type

### **String Data Type:-**

The string is a sequence of Unicode characters. A string may be a collection of 1 or more characters put during a quotation mark, double-quote, or triple quote. It can be represented using an str class.

Example:-

string1= “Hello World”

print(string1)

output: Hello World

### We can perform several operations in strings like Concatenation, Slicing, and Repetition.

Concatenation: It includes the operation of joining two or more strings together.

Example:-

String1 = "Hello"

String2 ="World"

print(String1+String2)

Output: Hello World

Slicing: Slicing is a technique for extracting different parts of a string.

Example:-

String1 = "Hello"

print(String1[2:4])

Output: llo

Repetition:

It means repeating a sequence of instructions a certain number of times.

Example:-

Print(String1\*5)

Output: HelloHelloHelloHelloHello

### **List Data Type:-**

A list is formed(or created) by placing all the items (elements) inside square brackets [ ], separated by commas.

It can have any number of items and they may or may not be of different types (integer, float, string, etc.).

A list is mutable, which suggests we will modify the list

Example:

List1 = [3,8,7.2,"Hello"]

print("List1[2] = ", List[2])

Output: List1[2] = 7.2

print("List1[1:3] = ", List[1:3])

Output: List1[1:3] = [8, 7.2]

Updating the list:- we can update the list.

List1[3] = "World"

#If we print the whole list, we can see the updated list.

print(List1)

Output: [3, 8, 7.2, ‘World’]

### **Tuple Data Type:-**

A tuple is defined as an ordered collection of Python objects. The only difference between tuple and list is that tuples are immutable i.e. tuples can’t be modified after it’s created. It is represented by tuple class. we can represent tuples using parentheses ( ).

Example:

Tuple = (25,10,12.5,"Hello")

print("Tuple[1] = ", Tuple[1])

Output: Tuple[1] = 10

print("Tuple[0:3] =", Tuple[0:3])

Output: Tuple[0:3] = (25,10,12.5)

### **Set Data Type:-**

A set is an unordered collection of items. Every set element is exclusive (no duplicates) and must be immutable (cannot be changed).

Example:

Set = {4,3,6.6,"Hello"}

print(Set)

Output: {‘Hello’, 3, 4, 6.6}

As the set is an unordered collection, indexing will be meaningless. Hence the slicing operator [ ] doesn’t work.

Set[1] = 12

Output: TypeError

### 

### **Dictionary Data Type:-**

In Python, Dictionary is an unordered collection of data values, which is used to store data values like a map, which, unlike other Data Types that hold only a single value as an element, a Dictionary consists of key-value pair. Key-value is provided within the dictionary to form it more optimized. In the representation of a dictionary data type, each key-value pair during a Dictionary is separated by a colon: whereas each key’s separated by a ‘comma’.

Syntax:

Key:value

Example:

Dict1 = {1:'Hello',2:5.5, 3:'World'}

print(Dict1)

Output: {1: ‘Hello’, 2: 5.5, 3: ‘World’}

We can retrieve the value by using the following method:

Example:

print(Dict[2])

Output: 5.5

We can update the dictionary by following methods as well:

Example:

Dict[3] = 'World'

print(Dict)

Output:

{1: ‘Hello’, 2: 5.5, 3: ‘World’}

**Operators in Python:-**

Operators are the constructs which can manipulate the value of operands.

## **Python Arithmetic Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + Addition | Adds values on either side of the operator. | a + b = 30 |
| - Subtraction | Subtracts right hand operand from left hand operand. | a – b = -10 |
| \* Multiplication | Multiplies values on either side of the operator | a \* b = 200 |
| / Division | Divides left hand operand by right hand operand | b / a = 2 |
| % Modulus | Divides left hand operand by right hand operand and returns remainder | b % a = 0 |
| \*\* Exponent | Performs exponential (power) calculation on operators | a\*\*b =10 to the power 20 |
| // | Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) − | 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0 |

## **Python Comparison Operators**

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | If the values of two operands are equal, then the condition becomes true. | (a == b) is not true. |
| != | If values of two operands are not equal, then condition becomes true. | (a != b) is true. |
| <> | If values of two operands are not equal, then condition becomes true. | (a <> b) is true. This is similar to != operator. |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. | (a > b) is not true. |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. | (a < b) is true. |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | (a >= b) is not true. |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. | (a <= b) is true. |

## **Python Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Assigns values from right side operands to left side operand | c = a + b assigns value of a + b into c |
| += Add AND | It adds right operand to the left operand and assign the result to left operand | c += a is equivalent to c = c + a |
| -= Subtract AND | It subtracts right operand from the left operand and assign the result to left operand | c -= a is equivalent to c = c - a |
| \*= Multiply AND | It multiplies right operand with the left operand and assign the result to left operand | c \*= a is equivalent to c = c \* a |
| /= Divide AND | It divides left operand with the right operand and assign the result to left operand | c /= a is equivalent to c = c / a |
| %= Modulus AND | It takes modulus using two operands and assign the result to left operand | c %= a is equivalent to c = c % a |
| \*\*= Exponent AND | Performs exponential (power) calculation on operators and assign value to the left operand | c \*\*= a is equivalent to c = c \*\* a |
| //= Floor Division | It performs floor division on operators and assign value to the left operand | c //= a is equivalent to c = c // a |

## 

## **Python Bitwise Operators**

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands −

a = 0011 1100

b = 0000 1101

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

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

There are following Bitwise operators supported by Python language

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & Binary AND | Operator copies a bit to the result if it exists in both operands | (a & b) (means 0000 1100) |
| | Binary OR | It copies a bit if it exists in either operand. | (a | b) = 61 (means 0011 1101) |
| ^ Binary XOR | It copies the bit if it is set in one operand but not both. | (a ^ b) = 49 (means 0011 0001) |
| ~ Binary Ones Complement | It is unary and has the effect of 'flipping' bits. | (~a ) = -61 (means 1100 0011 in 2's complement form due to a signed binary number. |
| << Binary Left Shift | The left operands value is moved left by the number of bits specified by the right operand. | a << 2 = 240 (means 1111 0000) |
| >> Binary Right Shift | The left operands value is moved right by the number of bits specified by the right operand. | a >> 2 = 15 (means 0000 1111) |

## **Python Logical Operators**

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and Logical AND | If both the operands are true then condition becomes true. | (a and b) is true. |
| or Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is true. |
| not Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is false. |

## 

## 

## **Python Membership Operators**

Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| in | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y, here in results in a 1 if x is a member of sequence y. |
| not in | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y, here not in results in a 1 if x is not a member of sequence y. |

## **Python Identity Operators**

Identity operators compare the memory locations of two objects. There are two Identity operators explained below −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| is | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y, here is results in 1 if id(x) equals id(y). |
| is not | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. | x is not y, here is not results in 1 if id(x) is not equal to id(y). |

## 

## **Python Operators Precedence**

The following table lists all operators from highest precedence to lowest.

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | \*\*  Exponentiation (raise to the power) |
| 2 | ~ + -  Complement, unary plus and minus (method names for the last two are +@ and -@) |
| 3 | \* / % //  Multiply, divide, modulo and floor division |
| 4 | + -  Addition and subtraction |
| 5 | >> <<  Right and left bitwise shift |
| 6 | &  Bitwise 'AND' |
| 7 | ^ |  Bitwise exclusive `OR' and regular `OR' |
| 8 | <= < > >=  Comparison operators |
| 9 | <> == !=  Equality operators |
| 10 | = %= /= //= -= += \*= \*\*=  Assignment operators |
| 11 | is is not  Identity operators |
| 12 | in not in  Membership operators |
| 13 | not or and  Logical operators |

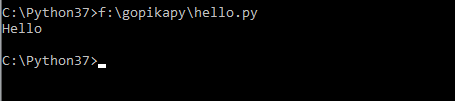
**Program : 1**

Aim : Print Hello

Code:

Print(“Hello”)

**Output :**



**Program : 2**

Aim : Add two numbers program(takes values as default)

Code:

a=5

print("first number is:")

print(a)

b=3

print("second number is :")

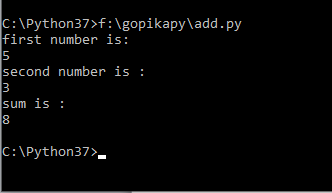
print(b)

sum=a+b

print("sum is :")

print(sum)

**Output :**



**Program : 3**

Aim : Program to add two numbers by user interactive input

Code:

a=input('Enter first number : ')

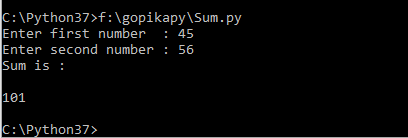
b=input('Enter second number : ')

sum=int(a)+int(b)

print("Sum is :\n")

print(sum)

**Output :**



**Program : 4**

Aim : Program to swap two numbers using temporary variable

Code:

x=input("Enter the value for x :")

y=input("Enter the value for y :")

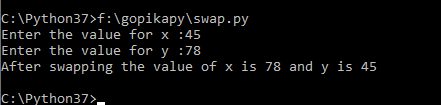
temp=x

x=y

y=temp

print("After swapping the value of x is {} and y is {}".format(x,y))

**Output :**



**Program : 5**

Aim : Program to swap two numbers without using temporary variables

Code :

a=input("Enter first number : ")

b=input("Enter second number : ")

sum=int(a)+int(b)

a=int(sum)-int(a)

print("First number after swapping : ")

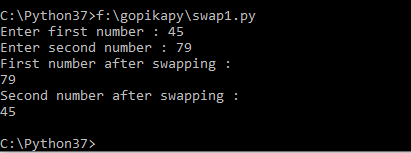
print(a)

b=int(sum)-int(b)

print("Second number after swapping : ")

print(b)

**Output**



**Program : 6**

Aim :Program to compute the roots of quadratic equation

Code:

import cmath

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

b=float(input("Enter b : "))

c=float(input("Enter c : "))

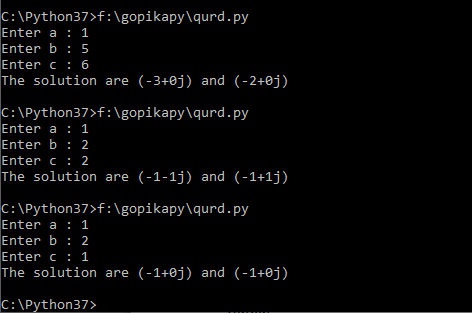
d=(b\*\*2)-(4\*a\*c)

sol1=(-b-cmath.sqrt(d))/(2\*a)

sol2=(-b+cmath.sqrt(d))/(2\*a)

print('The solution are {0} and {1}'. format(sol1,sol2))

**Output:**



**Program : 7**

Aim : Program to compute the area of a triangle

Code :

a=float(input("Enter first side : "))

b=float(input("Enter second side : "))

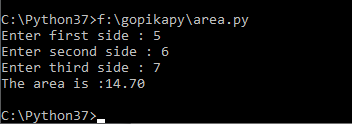
c=float(input("Enter third side : "))

s=(a+b+c)/2

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

print("The area is :%0.2f"%area)

**Output :**



**Program : 8**

Aim : Program to check leap year or not

Code :

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

if(year%4==0):

if(year%100==0):

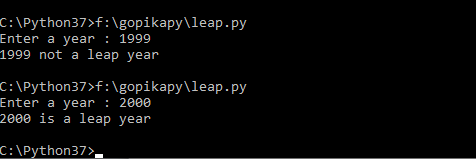
if(year%400==0):

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

else:

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

**Output:**



**Program : 9**

Aim :Program to stimulate calculator(without functions)

Code:

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

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

op=input(" Enter operator +,-,\*,/,^ : \n")

if(op=="+"):

print("sum is",a+b)

elif(op=="-"):

print("difference is",a-b)

elif(op=="\*"):

print("product is",a\*b)

elif(op=="/"):

print("quotient is",a//b)

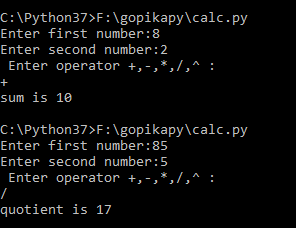
elif(op=="^"):

print("power is",a\*\*b)

else:

print("Not recognised operator")

**Output :**



**Program : 10**

Aim:Program to check whether the given number is Armstrong or not

Code

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

sum=0;

temp=num;

while(temp>0):

digit=temp%10

sum+=digit\*\*3

temp=temp//10

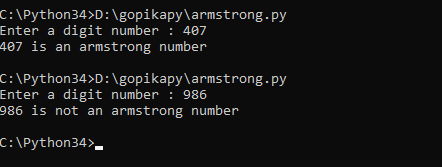
if (num==sum):

print(num,"is an armstrong number")

else:

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

**Output :**



**Program : 11**

Aim:To display multiplication table of a number

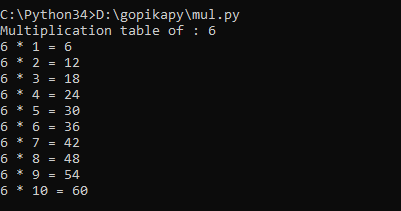
Code

n=int(input("Multiplication table of : "))

for i in range(1,11):

print(n,"\*",i,"=",n\*i)

**Output :**



**Program : 12**

Aim:Program to check perfect number or not

Code:

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

sum=0

for i in range(1,n):

if(n%i==0):

print(i,"is a factor of",n)

sum=sum+i

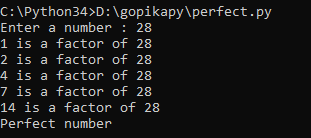
if(sum==n):

print("Perfect number")

else:

print("Not a perfect number")

**Output :**

****

**Program : 13**

Aim:Program to check whether prime or not

Code:

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

if(n>1):

c=0

for i in range(2,n):

if(n%i==0):

c=c+1

break

else:

continue

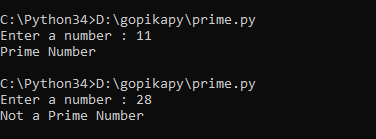
if(c==1):

print("Not a Prime Number")

else:

print("Prime Number")

**Output :**



**Program : 14**

Aim: Program to print prime in an interval

Code

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

high=int(input("Enter higher range :"))

for num in range(low,high+1):

if(num>1):

for i in range(2,num):

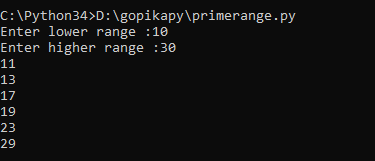
if((num%i)==0):

break

else:

print(num)

**Output :**



**Program : 15**

Aim : To convert a binary number to decimal

Code

def conv(n):

s=0

i=0

while(n>0):

r=n%10

s=s+r\*(2\*\*i)

i=i+1

n=n//10

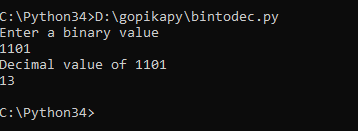
print(s)

f=int(input("Enter a binary value\n"))

print("Decimal value of",f)

conv(f)

**Output:**



**Program : 16**

Aim : To find sum till n number using function

Code :

def sum(n):

s=0

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

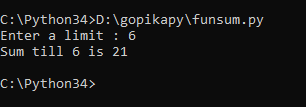
s=s+i

return s

n=int(input("Enter a limit : "))

print("Sum till",n,"is",sum(n))

**Output :**



**Program : 17**

Aim : To print fibinocci series of n numbers using function

Code :

def fibo(num):

a=0

b=1

for i in range(2,num):

c=a+b

print(c)

a=b

b=c

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

if(num==1):

print(0)

elif(num==2):

print(0,1)

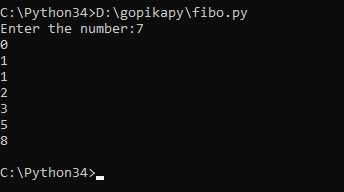
else:

print(0)

print(1)

fibo(num)

**Output :**



**Program : 18**

Aim : To find whether a number is palindrome or not

Code :

def pal(n):

s=0

temp=n

while(n>0):

r=n%10

s=s\*10+r

n=n//10

print("The reverse is ",s)

if(temp==s):

print("Number is palindrome")

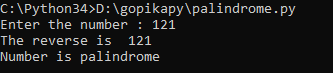
else:

print("Number is not palindrome")

f=int(input("Enter the number : "))

pal(f)

**Output :**



**Program : 19**

Aim : To compute exponential series

Code :

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

n=int(input("Enter n : "))

s=1

def fac(n):

f=1

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

f=f\*i

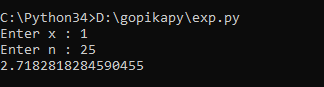
return(f)

for i in range(1,n):

s=s+(x\*\*i)/fac(i)

print(s)

**Output:**



**Program : 20**

Aim : To stimulate a calculator using function

Code :

def add(x,y):

return x+y

def sub(x,y):

return x-y

def mul(x,y):

return x\*y

def div(x,y):

return x/y

def pow(x,y):

return x\*\*y

def mod(x,y):

return x%y

print("Select Operations\n1-Add\n2-Subtract\n3-Multiply\n4-Division\n5-Power\n6-Modulus")

choice=input("Enter choice(1/2/3/4/5/6) : ")

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,"=",sub(num1,num2))

elif choice=='3':

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

elif choice=='4':

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

elif choice=='5':

print(num1,"^",num2,"=",pow(num1,num2))

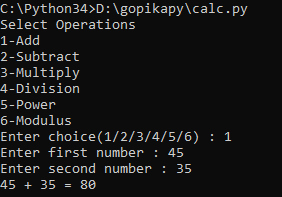
elif choice=='6':

print(num1,"%",num2,"=",mod(num1,num2))

else:

print("Invalid input")

**Output :**



**Program : 21**

Aim : To find factorial of a number using recursion

Code :

def rec(n):

if(n==1):

return 1

else:

return n\*rec(n-1)

num=int(input("Enter a number \n"))

if(num<0):

print("Factorial doesnot exist")

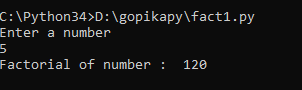
elif(num==0):

print("Factorial of num is 1 ")

else:

print("Factorial of number : ",rec(num))

**Output :**



**Program : 22**

Aim :

def fib(n):

if(n<=1):

return(n)

elif(n>0):

return(fib(n-1)+fib(n-2))

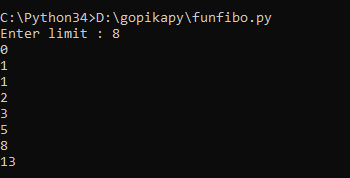
num=int(input("Enter limit : "))

for i in range(0,num):

r=fib(i)

print(r)

**Output :**



**Program : 23**

Aim : To find power of a number using recursion

Code :

def pow(x,n):

if n==0:

return(1)

else:

return(x\*pow(x,n-1))

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

b=int(input("Enter the power : "))

r=pow(a,b)

print("Power of the number is ",r)

**Output :**

