

INTRODUCTION TO PYTHON

Introduction

Python is a high-level, general-purpose, interpreted, interactive, object-oriented dynamic programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). Python is named after a TV Show called 'Monty Python's Flying Circus' and not after Python-the snake.

Python 3.0 was released in 2008. Although this version is supposed to be backward incompatibles, later on many of its important features have been back ported to be compatible with version 2.7

Installing Python

Virtually all modern programming languages make us of an IDE, or Integrated Development Environment, which allows the creation, editing, testing, and saving of programs and modules.

For Python programming you need a working Python installation and a text editor. Python comes with its own editor, IDLE, which is quite nice and totally sufficient for the beginning.

The Python download page is <http://www.python.org/download>. The most recent version is Python 3.5.3 (as of January 2017);

Windows users

Download the appropriate Windows installer (the x86 MSI installer, if you do not have a 64-bit AMD or Intel chip). Start the installer by double-clicking it and follow the prompts.

See <https://docs.python.org/3/using/windows.html#installing-python> for information.

Mac users

Starting from Mac OS X Tiger, Python ships by default with the operating system, but you will need to update to Python 3 until OS X starts including Python 3 (check the version by starting `python3` in a command line terminal). Also IDLE (the Python editor) might be missing in the standard installation. If you want to (re-)install Python, get the MacOS installer from the Python download site.

Linux, BSD, and UNIX users

You are probably lucky and Python is already installed on your machine. To test it type `python3` on a command line. If you see something like what is shown in the following section, you are set.

IDLE may need to be installed separately, from its own package such as `idle3` or as part of `python-tools`.

If you have to install Python, first try to use the operating system's package manager or go to the repository where your packages are available and get Python 3. Python 3.0 was released in December 2008; all distributions should have Python 3 available, so you may not need to compile it from scratch. Ubuntu and Fedora do have Python 3 binary packages available, but they are not yet the default, so they need to be installed specially.

Roughly, here are the steps to compile Python from source code in Unix (If these totally don't make sense, you may want to read another introduction to *nix, such as Introduction to Linux):

- ❓ Download the .tgz file (use your Web browser to get the gzipped tar file from <https://www.python.org/downloads/release/python-343>)
- Uncompress the tar file (put in the correct path to where you downloaded it):

```
$ tar -xvzf ~/Download/Python-3.4.3.tgz ...  
list of files as they are uncompressed
```

- Change to the directory and tell the computer to compile and install the program

```
cd Python-3.4/  
$ ./configure --prefix=$HOME/python3_install  
... lots of output. Watch for error messages here ...  
$ make  
... even more output. Hopefully no error messages ...  
$ make install
```

- Add Python 3 to your path. You can test it first by specifying the full path. You should add `$HOME/python3_install/bin` to your `PATH` bash variable.

```
$ ~/python3_install/bin/python3
Python 3.4.3 (... size and date information ...)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for more
information. >>>
```

The above commands will install Python 3 to your home directory, which is probably what you want, but if you skip the `--prefix=$HOME/python3_install`, it will install it to `/usr/local`. If you want to use the IDLE graphical code editor, you need to make sure that the tk and tcl libraries, together with their development files, are installed on the system. You will get a warning during the make phase if these are not available.

Configuring your PATH environment variable

The PATH environment variable is a list of folders, separated by semicolons, in which Windows will look for a program whenever you try to execute one by typing its name at a Command Prompt. You can see the current value of your PATH by typing this command at a Command Prompt:

```
echo %PATH%
```

The easiest way to permanently change environment variables is to bring up the built-in environment variable editor in Windows. How you get to this editor is slightly different on different versions of Windows.

On Windows 8 or Windows 10:

Press the Windows key and type *Control Panel* to locate the Windows Control Panel. Once you've opened the Control Panel, select View by: Large Icons, then click on *System*. In the window that pops up, click the *Advanced System Settings* link, then click the *Environment Variables...* button.

On Windows 7 or Vista:

Click the Start button in the lower-left corner of the screen, move your mouse over *Computer*, right-click, and select *Properties* from the pop-up menu. Click the *Advanced System Settings* link, then click the *Environment Variables...* button.

On Windows XP:

Right-click the *My Computer* icon on your desktop and select *Properties*. Select the *Advanced* tab, then click the *Environment Variables...* button.

Once you've brought up the environment variable editor, you'll do the same thing regardless of which version of Windows you're running. Under *System Variables* in the bottom half of the editor, find a variable called *PATH*. If there is one, select it and click *Edit...* Assuming your Python root is *C:\Python34*, add these two folders to your path (and make sure you get the semicolons right; there should be a semicolon between each folder in the list):

```
: \Python34  
C:\Python34\Scripts
```

Python programming – modes

Python has two basic modes: normal and interactive. The normal mode is the mode where the scripted and finished *.py* files are run in the Python interpreter. Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory. As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole.

Interactive mode is a good way to play around and try variations on syntax.

On macOS or linux, open a terminal and simply type "python". On Windows, bring up the command prompt and type "py", or start an interactive Python session by selecting "Python (command line)", "IDLE", or similar program from the task bar / app menu. IDLE is a GUI which includes both an interactive mode and options to edit and run files. Python should print something like this:

```
$ python  
Python 3.0b3 (r30b3:66303, Sep  8 2008, 14:01:02) [MSC v.1500 32 bit  
(Intel)] on win32  
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>>
```

The `>>>` is Python's way of telling you that you are in interactive mode. In interactive mode what you type is immediately run. Try typing `1+1` in. Python will respond with `2`.

Interactive mode allows you to test out and see what Python will do. If you ever feel the need to play with new Python statements, go into interactive mode and try them out. A sample interactive session:

```
>>> 5
5
>>> print(5*7)
35
>>> "hello" * 4
'hellohellohellohello'
>>> "hello".__class__
<type 'str'>
```

However, you need to be careful in the interactive environment to avoid confusion. For example, the following is a valid Python script:

```
if 1:
    print("True")
print("Done")
```

If you try to enter this as written in the interactive environment, you might be surprised by the result:

```
>>> if 1:
...     print("True")
...     print("Done")
File "<stdin>", line 3
    print("Done")
    ^
SyntaxError: invalid syntax
```

What the interpreter is saying is that the indentation of the second print was unexpected. You should have entered a blank line to end the first (i.e., "if") statement, before you started writing the next print statement. For example, you should have entered the statements as though they were written:

```
if 1:
    print("True")

print("Done")
```

Which would have resulted in the following:

```
>>> if 1:
...     print("True")
...
True
>>> print("Done")
Done
>>>
```

Interactive mode

Instead of Python exiting when the program is finished, you can use the -i flag to start an interactive session. This can be **very** useful for debugging and prototyping.

```
python -i hello.py
```

Ex.No.1**COMPUTE GCD OF TWO NUMBERS**

Date:

Aim:

To Write a Python program to compute the GCD of two numbers.

Algorithm:

Step 1: Start

Step 2: Take two numbers from the user as input

Step 3: Calculate the remainder $d1 \% d2$

Step 4: While the remainder is not equal to 0

Step 5: $d1 = d2$

Step 6: $d2 = \text{remainder}$

Step 7: $\text{remainder} = d1 \% d2$

Step 8: print the GCD

Step 9: Stop

Program:

```
def gcd(a,b):  
    if(b==0):  
        return a  
    else:  
        return gcd(b,a%b)  
a=int(input("Enter first number:"))  
b=int(input("Enter second number:"))  
print (gcd(a,b))
```

Output:

Enter the first number:15

Enter the second number:5

The GCD of the two numbers is 5

Result:

Thus the Python program to compute the GCD of two numbers was created and executed successfully.

Ex.No.2 FIND THE SQUARE ROOT OF NUMBER(NEWTONS METHOD)

Date:

Aim:

To Write a Python program to find the square root of a number (Newton's method)

Algorithm:

Step 1: Start

Step 2: The first parameter is the value whose square root will be approximated.

Step 3: The second is the number of times to iterate the calculation yielding a better result from the user.

Step 4: Calculate better = $\frac{1}{2} * (\text{approx.} + n / \text{approx.})$

Step 5: Print the final result

Step 6: Stop

Program:

```
n=int(input("Enter a number:"))
newton_squareroot= 0.5*((0.5*n)+n/(0.5*n))
print (newton_squareroot)
```

Output:

Enter a number:4

2.0

Result:

Thus the Python program to find the square root of a number (Newton's method) was created and executed successfully

Ex.No.3**EXPONENTIATION (POWER OF A NUMBER)**

Date :

Aim:

To Write a Python program to find the exponentiation (Power of a number)

Algorithm:

Step 1: Start

Step 2: $r \rightarrow n$

Step 3: Read Base Number n

Step 4: Read Exponent Number e

Step 5: FOR $i \rightarrow 1$ to e

 Step 5.1: Compute $r \rightarrow n * r$

Step 6: Print 'Exponent' r

Step 7: Stop

Program:

```
n=int(input("Enter a number:"))
e=int(input("Enter a number:"))
r=n
for i in range(1,e):
    r=n*r
print ("Exponent",r)
```

Output:

Enter a number:3

Enter a number:2

Exponent 9

Result:

Thus the a Python program to find the exponentiation (Power of a number) was created and executed successfully

Ex.No.4**FIND THE MAXIMUM OF A LIST OF NUMBERS**

Date :

Aim:

To write a Python program to find the maximum of a list of numbers.

Algorithm:

Step 1: Start

Step 2: Take n the number of elements and store it in a variable.

Step 3: Take the elements of the list one by one.

Step 4: Print the last element of the list

Step 5: Stop

Program

```
n=int(input("Enter the list size:"))
a=[]
for i in range(n):
    num=int(input("Enter the number"))
    a.append(num)
print (a)
max=a[0]
for i in range(n):
    if(max<a[i]):
        max=a[i]
print ("maximum",max)
```

Output:

Enter the list size:3

Enter the number1

Enter the number2

Enter the number3

[1, 2, 3]

maximum 3

Result:

Thus a Python program to find the maximum of a list of numbers was created and executed successfully.

Ex.No.5a

LINEAR SEARCH

Date :

Aim:

To write a python program to search an element in an array using Linear search technique.

Algorithm:

- Step 1: Start
- Step 2: Get list of elements from the user
- Step 3: Get the number to be searched
- Step 4: Set i to 1
- Step 5: if $i > n$ then go to step 7
- Step 6: if $A[i] = x$ then go to step 6
- Step 7: Set I to $i+1$
- Step 8: Go to Step 2
- Step 9: Print Element x found at index I and step 8
- Step 10: Print element not found
- Step 11: Stop

Program

```
n=int(input("Enter the no. of element"))
i=0
a=[i for i in range(n)]
for i in range(0,n):
    a[i]=int(input("enter the array element"))
for i in range(0,n):
```

```
print (a[i])

key=int(input("Enter the key element to be searched"))
for i in range(0,n):
    if a[i]==key:
        print ("key found")
```

Output:

```
Enter the no. of element5
enter the array element10
enter the array element20
enter the array element30
enter the array element40
enter the array element50
10
20
30
40
50
Enter the key element to be searched40
key found
```

Result:

Thus a python program to search an element was created using linear search technique and executed successfully

Ex.No.5b

BINARY SEARCH

Date:

Aim:

To write a Python program to search an element in a list of elements using Binary search technique.

Algorithm:

```
Step 1: Start
Step 2: Get list of elements from the user
Step 3: Get the number to be searched
Step 4: Found<-False
Step 5: While not found and first <=top
    if List[Midpoint]=ItemSought Then
        ItemFound=True
    Elif first>=Last Then
        SearchFailed=True
    Elif List[Midpoint]>ItemSought Then
        Last=Midpoint-1
    Else
        First=Midpoint+1
    End if
End While
Step 7: Stop
```

Program:

```
def binary_search(a,n,key):
    low=0
    high=n
    while(low<=high):
        mid=int((low+high)/2)
        if(key==a[mid]):
            return mid
        elif(key<a[mid]):
            high=mid-1
        else:
            low=mid+1
    return -1

n=int(input("enter the no of element"))
a=[i for i in range(n)]
for i in range(0,n):
    a[i]=int(input("enter the array element"))
k=int(input("Enter the key element to be searched"))
position=binary_search(a,n,k)
if(position!=-1):
    print ("present in ",(position))
else:
    print ("not present")
```

Output:

```
enter the no of element5
enter the array element10
enter the array element20
enter the array element30
enter the array element40
enter the array element50
Enter the key element to be searched30
present in 2
```

Result:

Thus a python program to search an element using Binary search technique was created and executed successfully

Ex.No.6a

SELECTION SORT

Date:

Aim:

To write a Python program to sort the elements using selection sort.

Algorithm:

- Step 1: Start
- Step 2: Get the elements from the user
- Step 3: Set MIN to location 0
- Step 4: Search the minimum element in the list.
- Step 5: Swap with value at location MIN
- Step 6: Increment MIN to point to next element
- Step 7: Repeat until list is sorted.
- Step 8: Stop

Program:

```
def selectionSort(a):  
    for i in range(len(a)):  
        least=i  
        for k in range(i+1,len(a)):  
            if a[k]<a[least]:  
                least=k  
        temp=a[least]  
        a[least]=a[i]  
        a[i]=temp  
a=[50,30,10,20,40,70,60]  
print ("Original list",a)
```

```
selectionSort(a)    #calling to the function  
print("Selection Sort:",a)
```

Output:

Original list [50, 30, 10, 20, 40, 70, 60]

Selection Sort: [10, 20, 30, 40, 50, 60, 70]

Result:

Thus a Python programs to sort elements using selection sort method was created and executed successfully.

Ex.No.6b**INSERTION SORT****Date:****Aim:**

To write a Python program to sort the elements using insertion sort .

Algorithm:

Step 1: Start

Step 2: Get the elements from the user

Step 3a: The second element in the list is compared with the elements that appear before it (only first element in this case).

Step 3b: If the second element is smaller than first element, second element is inserted in the position of first element. After first step, first two elements of an array will be sorted.

Step 3c: Pick next element

Step 4: Repeat step 3 until all the elements in the list is sorted

Step 5: Stop

Program:

```
def insertion(a):  
    for i in range(1,len(a)):  
        currentvalue=a[i]  
        position=i  
        while position>0 and a[position-1]>currentvalue:  
            a[position]=a[position-1]  
            position=position-1  
        a[position]=currentvalue  
a=[]    #define empty list  
n=int(input("Enter the upper limit"))
```

```
for i in range (n):  
    a.append(int(input()))  
insertion(a)  
print ("Insertion list",a)
```

Output:

```
Enter the upper limit7  
10  
09  
05  
04  
12  
16  
18  
Insertion list [4, 5, 9, 10, 12, 16, 18]
```

Result:

Thus a Python program to sort the elements using selection sort method was created and executed successfully

Ex.N0.7**MERGE SORT****Date:****Aim:**

To write a Python program to sort elements using merge sort method.

Algorithm:

Step 1: Start

MERGE-SORT(A,p,r)

Step 2: if $r > p$

Step 3: Find the middle point to divide the array into two halves:

then $q = \text{FLOOR} [(p+r)/2]$ //Divide step

Step 4: Call merge Sort for first half:

Call MERGE (A,p,q)

Step 5: Call merge Sort for second half:

Call MERGE (A,q+1,r)

Step 6: Merge the two halves sorted in step 2 and 3:

Call MERGE (A,p,q,r) //Conquer step

Step 7: Stop

Program:

```
def mergeSort(a):  
    print("Splitting ",a)  
    if len(a)>1:  
        mid = len(a)//2  
        lefthalf = a[:mid]  
        righthalf = a[mid:]  
        mergeSort(lefthalf)  
        mergeSort(righthalf)  
        i=0  
        j=0  
        k=0
```



```

while i < len(lefthalf) and j < len(righthalf):
    if lefthalf[i] < righthalf[j]:
        a[k]=lefthalf[i]
        i=i+1
    else:
        a[k]=righthalf[j]
        j=j+1
    k=k+1
while i < len(lefthalf):
    a[k]=lefthalf[i]
    i=i+1
    k=k+1
while j < len(righthalf):
    a[k]=righthalf[j]
    j=j+1
    k=k+1
print("Merging ",a)
a = [54,26,93,17,77,31,44,55,20]
mergeSort(a)
print(a)

```

Output:

```

Splitting [54, 26, 93, 17, 77, 31, 44, 55, 20]
Splitting [54, 26, 93, 17]
Splitting [54, 26]
Splitting [54]
Merging [54]
Splitting [26]
Merging [26]
Merging [26, 54]

```

Splitting [93, 17]
Splitting [93]
Merging [93]
Splitting [17]
Merging [17]
Merging [17, 93]
Merging [17, 26, 54, 93]
Splitting [77, 31, 44, 55, 20]
Splitting [77, 31]
Splitting [77]
Merging [77]
Splitting [31]
Merging [31]
Merging [31, 77]
Splitting [44, 55, 20]
Splitting [44]
Merging [44]
Splitting [55, 20]
Splitting [55]
Merging [55]
Splitting [20]
Merging [20]
Merging [20, 55]
Merging [20, 44, 55]
Merging [20, 31, 44, 55, 77]
Merging [17, 20, 26, 31, 44, 54, 55, 77, 93]

Result:

Thus a Python programs to sort the element using merge sort method was created and executed successfully

Ex.NO.8**FIND N PRIME NUMBERS****Date:****Aim:**

To Write a Python program to find the first n prime numbers.

Algorithm:

Step 1: Start

Step 2: Take the number to be checked and store it in a variable.

Step 3: Initialize the count variable to 0

Step 4: Let the for loop range from 2 to half number (excluding 1 and the number itself).

Step 5: Then find the number of divisors using the if statement and increment the count variable each time.

Step 6: If the number of divisors is lesser than or equal to 0, the number is prime.

Step 7: Print the final result

Step 8: Stop

Program:

```
n=int(input("Enter the limit:"))
for num in range(1,n): #iterate between 1 to n
    for i in range(2,num): #iterate factors of the num
        if num%i==0:
            r=num/i
            break
    else:
        print ("Prime number:",num)
```

Output:

Enter the limit:25

Prime number: 1

Prime number: 2

Prime number: 3

Prime number: 5

Prime number: 7

Prime number: 11

Prime number: 13

Prime number: 17

Prime number: 19

Prime number: 23

Result:

Thus a Python program to find the first n prime numbers was created and executed successfully.

Ex.No.9**MULTIPLY MATRICES****Date:****Aim:**

To write a Python program to multiply two matrices.

Algorithm:

Step 1: Start

Step 2: Declare variables and initialize necessary variables

Step 3: Enter the element of matrices by row wise using loops

Step 4: Check the numbers of rows and column of first and second matrices

Step 5: If number of rows of first matrix is equal to the number of columns of second matrix, go to Otherwise, print matrix multiplications are not possible and go to step 3

Step 6: Multiply the matrices using nested loops

Step 7: Print the product in matrix form as console output.

Step 8: Stop

Program:

```
A = [[12,7,3],
```

```
      [4 ,5,6],
```

```
      [7 ,8,9]]
```

```
# 3x4 matrix
```

```
B = [[5,8,1,2],
```

```
      [6,7,3,0],
```

```
      [4,5,9,1]]
```

```
# result is 3x4
```

```
C = [[0,0,0,0],
```

```
      [0,0,0,0],
```

```
[0,0,0,0]]
```

```
# iterate through rows of A
for i in range(len(A)):
    #iterate through columns of B
    for j in range(len(B[0])):
        # iterate through rows of B
        for k in range(len(B)):
            C[i][j] += A[i][k] * B[k][j]
for r in C:
    print(r)
```

Output:

```
[114, 160, 60, 27]
```

```
[74, 97, 73, 14]
```

```
[119, 157, 112, 23]
```

Result:

Thus a Python program to multiply two matrices was created and executed successfully.

Ex.No.10 PROGRAMS THAT TAKE COMMAND LINE ARGUMENTS (WORD COUNT)

Date:

Aim:

To write a Python program to take command line arguments(word count)

Algorithm:

Step 1: Start

Step 2: Take the file name from the user

Step 3: Read each line from a file and split a lines to form a list of words

Step 4: Find the length of items in a list and print it

Step 5: Stop

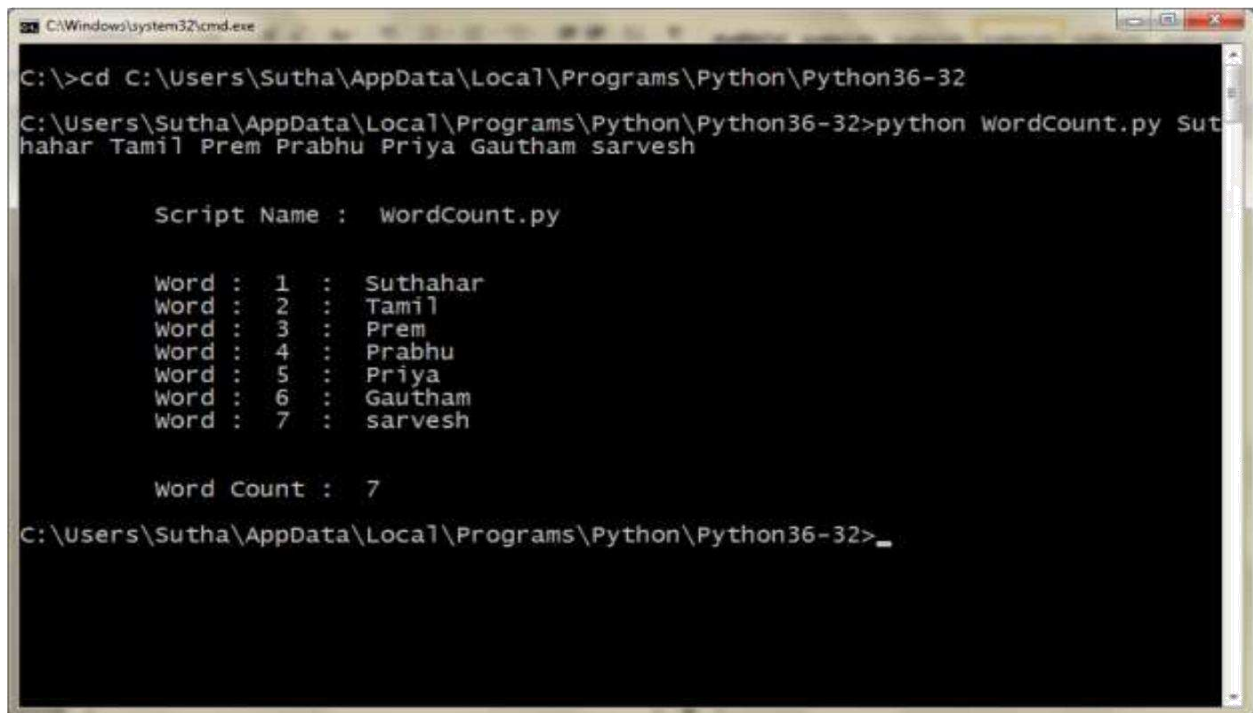
Program 1:

```
import sys
print("\n\n\t Script Name : ",sys.argv[0])
print("\n")
le=len(sys.argv)
for i in range(1,le):
    print("\t Word : ",i," : ", sys.argv[i])
print("\n\n\t Word Count : ", len(sys.argv)-1)
```

Output:

Number of arguments 1 arguments

Argument list ['C:\Users\Sutha\AppData\Local\Programs\Python\Python36-32\WordCount.py']



```
C:\Windows\system32\cmd.exe
C:\>cd C:\Users\Sutha\AppData\Local\Programs\Python\Python36-32
C:\Users\Sutha\AppData\Local\Programs\Python\Python36-32>python WordCount.py Sut
hahar Tamil Prem Prabhu Priya Gautham sarvesh

    Script Name :  WordCount.py

    Word :  1  :  Suthahar
    Word :  2  :  Tamil
    Word :  3  :  Prem
    Word :  4  :  Prabhu
    Word :  5  :  Priya
    Word :  6  :  Gautham
    Word :  7  :  sarvesh

    Word Count :  7
C:\Users\Sutha\AppData\Local\Programs\Python\Python36-32>
```

Result:

Thus a Python program for word count was created and executed successfully

Ex.No.11

Date:

**FIND THE MOST FREQUENT WORDS IN A TEXT READ
FROM A FILE**

Aim:

To write a Python program to find the most frequent words in a text read from a file

Algorithm:

Step 1: Start

Step 2: Create a file in a notepad and save it with .py extension.

Step 2: Take the file name and letter to be counted from the user

Step 2: Read each line from the file and split the line to form a list of words.

Step 3: Use a for loop to traverse through the words in the list and another for loop to traverse through the letters in the word

Step 4: Check if the letter provided by the user and the letter encountered over iteration is equal and if they are, increment the letter count. Step 5: stop

Program:

```
from collections import Counter
def frequent(name):
    with open(name) as file:
        return Counter(file.read().split())
print ("Most frequent word")
print ("File name")
name=input()
print (frequent(name))
```

Test.txt

Sri Sairam Institute of Technology

Sai Leo Nagar

West Tambaram

Sairam

sai Leo Nagar

Chennai

Output:

Most frequent word

File name : Test.txt

```
Counter({'Sairam': 2, 'Leo': 2, 'Nagar': 2, 'Sri': 1, 'Institute': 1, 'of': 1, 'Technology': 1, 'Sai': 1, 'West': 1, 'Tambaram': 1, 'sai': 1, 'Chennai': 1})
```

Result:

Thus a Python program to find the most frequent words in a text read from a file was created and successfully

Ex.No.12**SIMULATE ELLIPTICAL ORBITS IN PYGAME**

Date:

Aim:

To write a Python program to simulate elliptical orbits in pygame.

Algorithm:

Step 1: Start

Step 2: Import necessary mathematical function library from numpy package

Step 3: Import plot library

Step 4: Create the necessary window and

Step 5: draw elliptical orbit by using the equation $ay^2 + bxy + cx + dy + e = x^2$

Step 6: Plot the points using Scatter plot functions and draw orbit

Step 7: Stop

Program:

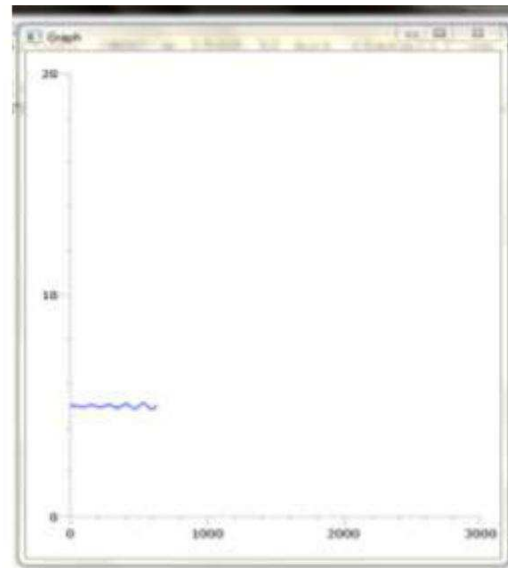
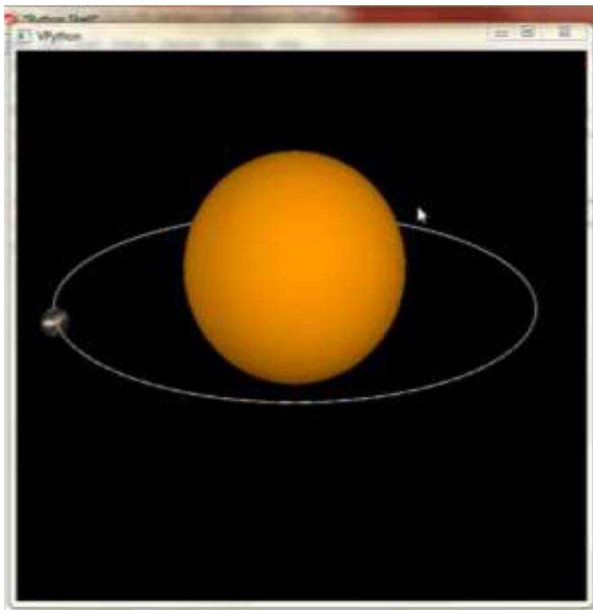
```
from visual import *
from visual.graph import *
scene=display(width=600,height=600,center=(0,5,0))
Sun=sphere(pos=(0,5,0),radius=100,color=color.orange)
earth=sphere(pos=(-200,0,0),radius=10,material=materials.earth,make_trial=true)
earthv=vector(0,0,5)
gd=gdisplay(x=800,y=0,width=600,height=600,
            foreground=color.black,background=color.white,
            xmax=3000,xmin=0,ymax=20,ymin=0)
f1=gcurve(color=color.blue)
t=0
for i in range(1000):
    rate(50)
```

```

earth.pos=earth.pos+earthv
dist=(earth.x**2 + earth.y**2 + earth.z**2)**0.5
RadialVector=(earth.pos - Sun.pos)/dist
Fgrav=-10000*RadialVector/dist **2
earthv=earthv+Fgrav
earth.pos+=earthv
f1.plot(pos=(t,mag(earthv)))
t+=1
if dist<=Sun.radius: break

```

Output:



Result:

Thus a Python program to simulate elliptical orbits in pygame was created and executed successfully.

Ex.No.13**SIMULATE BOUNCING BALL USING PYGAME**

Date :

Aim:

To write a Python program to simulate bouncing ball using pygame.

Algorithm:

Step 1: Start

Step 2: Import necessary GUI for simulation

Step 3: Set the window coordinates and the ball coordinates

Step 4: Assign various colors for the ball and set the base color

Step 5: Fix the color, shape and bouncing speed randomly

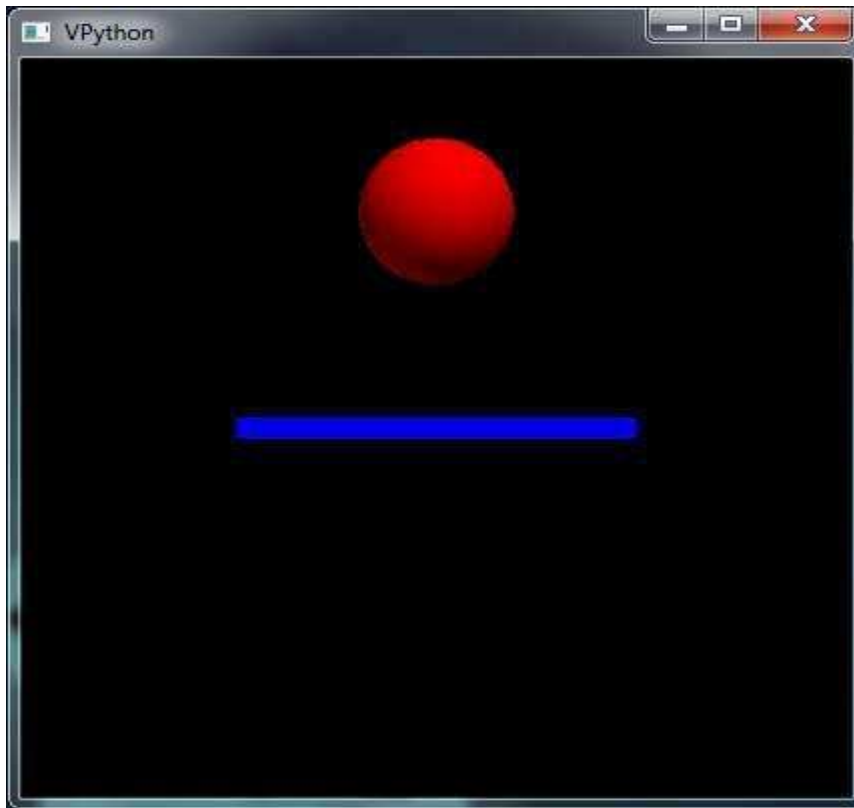
Step 6: Write a function to move the base according to the ball position

Step 7: Stop

Program:

```
from visual import *
floor = box(length=4, height=0.25, width=4, color=color.blue)
ball = sphere(pos=(0,4,0), color=color.red)
ball.velocity = vector(0,-1,0)
dt = 0.01
while 1:
    rate(100)
    ball.pos = ball.pos + ball.velocity*dt
    if ball.y < 1:
        ball.velocity.y = -ball.velocity.y
    else:
        ball.velocity.y = ball.velocity.y - 9.8*dt
```

Output:



Result:

Thus a Python program to simulate bouncing ball using pygame was simulated successfully

Ex.No.14.A

PYTHON MYSQL

Date :

Aim:

To simulate Python MySQL.

MySQL connector/Python to access MySQL databases:

MySQL connector/Python is a standardized database driver provided by MySQL. To work with MySQL Python connector, you need to download and install it in local system.

Verifying MySQL Connector/Python installation:

After installing the MySQL Python connector, you need to test it to make sure that it is working correctly and you are able to connect to MySQL database server without any issues. To verify the installation, use the following steps:

1. Open Python command line
2. Type the following code

After installing the MySQL Python connector, you need to test it to make sure that it is working correctly and you are able to connect to MySQL database server without any issues. To verify the installation, use the following steps:

1. Open Python command line
2. Type the following code

```
1 >>>>> import mysql.connector
2 mysql.connector.connect(host='localhost',database='mysql',user='root',password='')
```

If the output is shown as below, you have been successfully installing the MySQL Python connector in your system.

Connecting to MySQL databases from Python using MySQL Connector/Python API:

```
1 <mysql.connector.connection.MySQLConnection object at 0x0187AE50>
```

1. Prepare a sample database :To create a new database, you can launch the MySQL Workbench or mysql client tool, and use the CREATE DATABASE statement as follows

```
1 CREATE DATABASE python_mysql;
```

2. Load the sample data into the python_mysql database from python_mysql.sql file.
3. Connect to MySQL database using connect() function

```
import mysql.connector
from mysql.connector import Error
def connect():
    """ Connect to MySQL database """
    try:
        conn = mysql.connector.connect(host='localhost',
                                       database='python_mysql',
                                       user='root',
                                       password='secret')

        if conn.is_connected():
            print('Connected to MySQL database')
    except Error as e:
        print(e)
    finally:
        conn.close()
if __name__ == '__main__':
    connect()
```

4. Test the python_mysql_connect1.py module, you use the following command:

```
1 >python python_mysql_connect1.py
2 Connected to MySQL database
```


To query data in a MySQL database from Python:

1. [Connect to the MySQL Database](#), you get a MySQLConnection object.
2. Instantiate a MySQLCursor object from the the MySQLConnection object.
3. Use the cursor to execute a query by calling its execute() method.
4. Use fetchone() , fetchmany() or fetchall() method to fetch data from the result set.
5. Close the cursor as well as the database connection by calling the close() method of the corresponding object.

- **Querying data with fetchone:**

The fetchone() method returns the next row of a query result set or None in case there is no row left.

```
from mysql.connector import MySQLConnection,
Error from python_mysql_dbconfig import
read_db_config def query_with_fetchone():
    try:
        dbconfig = read_db_config()
        conn = MySQLConnection(**dbconfig)
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM books")
        row = cursor.fetchone()
        while row is not None:
            print(row)
            row = cursor.fetchone()
    except Error as e:
        print(e)
    finally:
        cursor.close()
        conn.close()
if __name__ == '__main__':
```

query_with_fetchone()

1. First, we connected to the database by create a new MySQLConnection object
2. Second, from the MySQLConnection object, we instantiated a new MySQLCursor object
3. Third, we executed a query that selects all rows from the books table.
4. Fourth, we called fetchone() method to fetch the next row in the result set. In the while loopblock, we printed out the content of the row and move to the next row until all rows are fetched.
5. Fifth, we closed both cursor and connection objects by invoking the close() method of the corresponding object.

- **Querying data with fetchall**

In case the number of rows in the table is small, you can use the fetchall() method to fetch all rows from the database table.

```
from mysql.connector import MySQLConnection, Error
from python_mysql_dbconfig import read_db_config
def query_with_fetchall():
    try:
        dbconfig = read_db_config()
        conn = MySQLConnection(**dbconfig)
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM books")
        rows = cursor.fetchall()
        print("Total Row(s):", cursor.rowcount)
        for row in rows:
            print(row)
    except Error as e:
        print(e)
    finally:
```

```
cursor.close()
conn.close()

if __name__ == '__main__':
    query_with_fetchall()
```

The logic is similar to the example with the `fetchone()` method except for the `fetchall()` method call part. Because we fetched all rows from the books table into the memory, we can get the total rows returned by using the `rowcount` property of the cursor object.

- **Querying data with `fetchmany`**

MySQL Connector/Python provides us with the `fetchmany()` method that returns the next number of rows (n)

- First, we develop a generator that chunks the database calls into a series of `fetchmany()` calls as follows:

```
1 def iter_row(cursor, size=10):
2     while True:
3         rows = cursor.fetchmany(size)
4         if not rows:
5             break
6         for row in rows:
7             yield row
```

- Second, we can use the `iter_row()` generator to fetch 10 rows at a time as shown below:

```
def query_with_fetchmany():
    try:
        dbconfig = read_db_config()
```

```

conn = MySQLConnection(**dbconfig)
cursor = conn.cursor()
cursor.execute("SELECT * FROM books")
for row in iter_row(cursor, 10):
    print(row)
except Error as e:
    print(e)
finally:
    cursor.close()
    conn.close()

```

To insert new rows into a MySQL table follow the steps below:

1. Connect to the MySQL database server by creating a new MySQLConnection object.
2. Initiate a MySQLCursor object from the MySQLConnection object.
3. Execute the INSERT statement to insert data into the intended table.
4. Close the database connection.

The following code inserts a new book into the `books` table:

```

1 from mysql.connector import MySQLConnection, Error
2 from python_mysql_dbconfig import read_db_config
3
4 def insert_book(title, isbn):
5     query = "INSERT INTO books(title,isbn) " \
6         "VALUES(%s,%s)"
7     args = (title, isbn)

```

```

8
9  try:
10     db_config = read_db_config()
11     conn = MySQLConnection(**db_config)
12     cursor = conn.cursor()
13     cursor.execute(query, args)
14     if cursor.lastrowid:
15         print('last insert id', cursor.lastrowid)
16     else:
17         print('last insert id not found')
18     conn.commit()
19 except Error as error:
20     print(error)
21 finally:
22     cursor.close()
23     conn.close()
24 def main():
25     insert_book('A Sudden Light','9781439187036')
26 if __name__ == '__main__':
27     main()

```

To delete rows in a MySQL table from Python, you need to do the following steps:

1. Connect to the database by creating a new MySQLConnection object.
2. Instantiate a new cursor object and call its execute() method. To commit the changes, you should always call the commit() method of the MySQLConnection object after calling the execute() method.
3. Close the cursor and database connection by calling close() method of the corresponding objects.

The following example shows you how to delete a book specified by book id

```
1 from mysql.connector import MySQLConnection, Error
2 from python_mysql_dbconfig import read_db_config
3
4 def delete_book(book_id):
5     db_config = read_db_config()
6     query = "DELETE FROM books WHERE id = %s"
7     try:
8         # connect to the database server
9         conn = MySQLConnection(**db_config)
10        # execute the query
11        cursor = conn.cursor()
12        cursor.execute(query, (book_id,))
13        # accept the change
14        conn.commit()
15    except Error as error:
16        print(error)
17    finally:
18        cursor.close()
19        conn.close()
20 if __name__ == '__main__':
21     delete_book(102)
```

Result:

Thus the connection of MySQL is simulated using Python successfully.

Ex.No.14.B SOCKET PROGRAM TO TRANSFER A FILE FROM

Date: SERVER/CLIENT USING PYTHON

(NETWORKING – OVER TCP)

Aim:

To write a Socket Program to transfer a file from Server/Client using Python.

Algorithm:

Server:

1. Import all the necessary packages.
2. Create a client and server socket and accept client connection.
3. Transfer files that user requested to transfer using output stream.
4. Close the socket and trace the output.

Client:

1. Start, import all the necessary packages.
2. Create a client socket and connect to server using port & IP address.
3. Send a file request to server.
4. Then server responses are retrieved using input & output streams.
5. Close the socket.

Result:

Thus a Socket Program to transfer a file from Server/Client using Python is successfully verified.

Ex.No.15.a

ODD OR EVEN

Date:

Aim:

To write a Python program to find given number is odd or even.

Algorithm:

1. Read the value of n.
2. If $n > 1$
 If $n \% 2 == 0$
 Print the value of n, "Even number"
 Else
 Print the value of n, "Odd number"
3. Else
4. Print "invalid number"

Program:

```
n=int (input ("Enter the numbers :"))  
if n>1:  
    if(n%2)==0:  
        print(n,"Even number")  
    else:  
        print(n,"Odd number")  
else:  
    print ("Invalid number")
```


Output:

Enter the number: 16

16 Even number

Enter the number: 13

13 Odd number

Result:

Thus the Python program to find the given number is odd or even is executed successfully and the output is verified.

Ex.No.15.b**SIMPLE CALCULATOR****Date:****Aim:**

To write a Python program to develop a simple calculator.

Algorithm:

1. Read the value of a and b.
2. Print the choices.
3. Read the choice as n.
4. if (n>0) and(n<5)
 - If n==1
 - Print the sum of a and b.
 - elif n==2
 - Print the difference of a and b.
 - elif n==3
 - Print the product of a and b.
 - else
 - Print the quotient of a divided by b.
- 5.else
 - Print Invalid choice.

Program:

```
a=float(input("Enter a:"))
b= float(input("Enter b:"))
print ("To add -1\n To Subtract -2\n To Multiplly -3\n To divide-4)
n=int(input("Enter your choice:"))
if (n>0) and (n<5):
    if n==1:
        print(a+b)
```

```
elif n==2:
    print(a-b)
elif n==3:
    print(a*b)
else:
    print(a/b)
else:
    print("Invalid Choice")
```

Output:

```
Enter a:10
Enter b:5
To add -1
To Subtract -2
To Multiply -3
To divide-4
Enter your choice:4
2
```

Result:

Thus the Python program to develop a simple calculator is executed successfully and the output is verified.

Ex.No.15.c**LEAP YEAR OR NOT****Date:****Aim:**

To write a Python program to find given year is leap or not.

Algorithm:

1. Read the value of year
2. If $\text{year} \% 4 == 0$
 - 2.1 Print "It is a leap year"
3. Else
 - 3.1 Print " It is not a leap year"
4. End the program.

Program:

```
Year =int (input("Enter the year :"))  
if (Year%4 )==0 :  
    print ("It is a leap year")  
else:  
    print ("It is not a leap year")
```

Output:

```
Enter the year: 2016  
It is a leap year
```

Result:

Thus the Python program to find the given year is leap or not is executed successfully and the output is verified.

Ex.No.15.d

FACTORIAL

Date:

Aim:

To write a Python program to find the factorial of the given number.

Algorithm:

1. Initialize the value of fact=1
2. Read the value of n
3. For i in range of n+1
4. Print (fact , “ is factorial ’)
5. End of the program.

Program:

```
n =int(input("Enter the number :"))
fact=1
for i in range (1,n+1):
    fact=fact+i
print("Factorial is ', fact)
```

OUTPUT:

Enter the number:5

Factorial is 120

Result:

Thus the Python program to find the factorial of a number is executed successfully and the output is verified.

Ex.No.15.e

FIBONACCI SERIES

Date:

Aim:

To write a Python program to evaluate the Fibonacci series for n terms.

Algorithm:

1. Intialize the value of x=0
2. Intialize the value of y =1
3. Read the value of n
4. For I in range of n+1
 - 4.1 .z=x+y
 - 4.2 Print x
 - 4.3 x=y
 - 4.4 y =z
5. End of the program.

Program:

```
x=0
y=1
n = int(input("Enter the terms:"))
for i in range (n,+1):
    z=x+y
    print(x)
    x=y
    y=z
```

OUTPUT:

Enter the numbers:10

0
1
1
2
3
5
8
13
21
34

Result:

Thus the Python program to evaluate Fibonacci series for n terms is executed successfully and the output is verified.

Ex.No.15.f

ARMSTRONG NUMBER

Date:

Aim:

To write a python program to find the Armstrong Number.

Algorithm:

1. Read num value
2. Initialize Sum=0
3. Temp=num
4. If num>0 Else goto step 6
5. While (temp>0):
 - 5.1 Rem=temp%10
 - 5.2 temp=temp//10
6. If num==sum
 - 6.1 : Print(num,"is an armstron number")
7. Else
 - 7.1 : Print(num,"is not an armstong number")
8. End of the program.

Program:

```
num=int(input("enter the value"))
sum=0
temp=num
while temp>0:
    d=temp%10
    sum=sum+d **3
    temp=temp//10
```

```
if num==sum:  
  
    print("Armstrong number is:",num)  
else:  
    print("not an Armstrong number is:",num)
```

Output:

```
Enter the value:153  
Armstrong number is:153
```

Result:

Thus the Python program to find the given number is Armstrong or not is executed successfully and the output is verified.

Ex.No.16.a**CIRCULATING N VALUES****Date:****Aim:**

To write a Python program to find circulating 'n' values.

Algorithm:

1. Define a List named a[]
2. Get the input 'n' value to rotate the values.
3. Perform the following step till
 $b = a[n:] + a[:n]$,using slice operator
4. Print the rotated values.

Program:

```
a=[1,2,3,4,5,6,7,8,9,10]
n=int(input())
b=a[n:]+a[:n]
print(b)
```

Output:

```
Enter the number:6
[7, 8, 9, 10, 1, 2, 3, 4, 5, 6]
```

Result:

Thus the Python programs to circulate the N Values are executed successfully and the output is verified.

Ex.No.16.b**SUM OF N NUMBERS****Date:****Aim:**

To write a Python program to find Sum Of N Numbers.

Algorithm:

1. Import the module 'array'
2. Create an array with name Y
3. Enter the limit of the array 'a'
4. Get the elements in the array until a
5. Assign the initial value as b=0
6. Add all the elements of the array with initial value.
7. Print the sum of the elements in the array.

Program:

```
from array import*
y=array('I',[])
a = int(input("ENTER LIMIT:"))
for i in range(0,a):
    x=int(input("ELEMENTS:"))
    y.append(x)
b = 0
for i in y:
    b+= i
print ("The Sum of n elements is" ,b)
```

OUTPUT

ENTER LIMIT:3

ELEMENTS:4

ELEMENTS:5

ELEMENTS:6

The Sum of n elements is 15

Result:

Thus the Python programs to sum of N Numbers are executed successfully and the output is verified.

Ex.No.16.c

EXCHANGE THE VALUES OF TWO VARIABLES

Date:

Aim:

To write a Python program to perform Swapping of two numbers.

Algorithm:

1. Define a function swap and call the function.
2. Get the first number
3. Get the second number
4. Print the numbers before swapping
5. Assign the first number with temporary variable 'temp'
6. Store the second number to the first number
7. Reassign the second number to the temporary variable
8. Print the Swapped values.

Program:

```
def swap():  
    a= int(input("enter first number:"))  
    b=int(input("enter the second number:"))  
    print("the numbers before swaping",a,b)  
    temp=a  
    a=b  
    b=temp  
    print("the numbers after swaping",a,b)  
swap()
```

OUTPUT:

Enter first number:4

Enter the second number:7

The numbers before swapping 4 7

The numbers after swapping 7 4

Result:

Thus the Python programs to exchange the values between two variables are executed successfully and the output is verified.

Ex.No.16.d DISTANCE BETWEEN TWO POINTS

Date:

Aim:

To write a Python program to find the Distance between Two Points

Algorithm:

1. Import the module 'math'
2. Define a function distance with four coordinates and call the function.
3. Calculate the difference of the 'x' coordinates as 'a'.
4. Calculate the difference of the 'y' coordinates as 'b'
5. Calculate the sum of 'x' and 'y' coordinates;
i. $c=(a**2)+(b**2)$
6. Calculate the square root of 'c'
7. Print the distance between two points.

Program:

```
import math
def distance(x1,y1,x2,y2):
    a=x2-x1
    b=y2-y1
    c=(a**2)+(b**2)
    d=math.sqrt(c)
    return d
a=int(input("enter the no:"))
b=int(input("enter the no:"))
c=int(input("enter the no:"))
d=int(input("enter the no:"))
e=distance(a,b,c,d)
print("the distance between two points is ",e)
```


OUTPUT:

Enter the no:4

Enter the no:2

Enter the no:6

Enter the no:4

The distance between two points is 2.8284271247461903

Result:

Thus the Python programs to distance between two points are executed successfully and the output is verified.