SWAMI KESHVANAND INSTITUTE OF TECHNOLOGY, MANAGEMENT AND GRAMOTHAN, JAIPUR

Hands on Lab Guide
(Lab Manual)

6CS4-23 PYTHON LAB

(III Year B.Tech. VI Sem)

Session 2021-22

Department of Computer Science and Engineering

INTRODUCTION TO PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level 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 a language with a simple syntax, and a powerful set of libraries. It is an interpreted language, with a rich programming environment, including a robust debugger and profiler. While it is easy for beginners to learn, it is widely used in many scientific areas for data exploration. The key advantages of learning Python:

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** −We can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

1.1 Characteristics of Python

- Following are important characteristics of **Python Programming** −
- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

1.2 Python Features

Python's features include −

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.

- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **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.

Apart from the above-mentioned features, Python has good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Different Ways of Invoking Python:

- Python GUI
- Python command line
- Command prompt from windows

Use Python Shell (using command line) and IDLE – Interactive development environment.

- To evaluate expression
- To create a script.

1.3 Using IDLE

IDLE is the standard Python development environment. Its name is an acronym of "Integrated DeveLopment Environment". It works well on both Unix and Windows platforms. It has a Python shell window, which gives you access to the Python interactive mode. It also has a file editor that lets you create and edit existing Python source files.

1.3.1 Interactive Python shell

When you start up IDLE, a window with an interactive Python shell will pop up:

You can type Python code directly into this shell, at the '>>>' prompt. Whenever you enter a complete code fragment, it will be executed. For instance, typing: >>> print "hello world" and pressing ENTER, will cause the following to be displayed: hello world

IDLE can also be used as a calculator:

```
>>> 4+4 8

>>> 8**3 512
```

Addition (+), subtraction (-), multiplication (*), division (/), modulo (%) and power (**) operators are built into the Python language. This means you can use them right away. If you want to use a square root in your calculation, you can either raise something to the power of 0.5 or you can import the math module Below are two examples of square root calculation:

```
>>> 16**0.5            #4.0

>>> import math

>>> math.sqrt(16)        #4.0
```

The math module allows you to do a number of useful operations:

```
>>> math.log(16, 2)            #4.0

>>> math.cos( 0 )               #1.0
```

Note that you only need to execute the import command once after you start IDLE; however you will need to execute it again if you restart the shell, as restarting resets everything back to how it was when you opened IDLE.

1.3.2 Creating scripts

1. save your hello.py program in the ~/pythonpractice folder.

2. Open up the terminal program.

3. Type cd ~/pythonpractice to change directory to your pythonpractice folder, and hit Enter.

4. Type chmod a+x hello.py to tell Linux that it is an executable program.

5. Type ./hello.py to run your program!

Program to add two integers. Take input from user.

```
number1 = input(" Please Enter the First Number: ")
 number2 = input(" Please Enter the second number: ")
# Using arithmetic + Operator to add two numbers
 sum = float(number1) + float(number2)
print('The sum of {0} and {1} is {2}'.format(number1, number2, sum))
```

1.4 Python Comments

Comments starts with a #,""" and Python will ignore them:

```
# this is a comment
print("Python Program")
```

1.5 Variables

Containers for storing data values. Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it. Variables do not need to be declared with any particular type and can even change type after they have been set.

```
x = 4           # x is of type int
x = "python" # x is now of type str
print(x)
```

1.6 Built-in Data Types

Variables can store data of different types, and different types can do different things. Python has the following data types built-in by default, in these categories:

Text Type:	Str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	Dict
Set Types:	set, frozenset
Boolean Type:	Bool

Binary Types: bytes, bytearray, memoryview

We can get the data type of any object by using the type() function:

```
x = 1            # int
y = 2.8          # float
z = 1j         # complex
print(type(x))    #output 'int'
```

Example

- integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

```
x = int(1)       # x will be 1
y = int(2.8)       # y will be 2
z = int("3")      # z will be 3
```

1.7 Python Operators

Operators are used to perform operations on variables and values. Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

DECISION MAKING & CONTROL STATEMENTS

Decision making statements in programming languages decides the direction of flow of program execution. Decision making statements available in python are:

- if statement
- if..else statements
- nested if statements
- if-elif ladder

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

```
for val in sequence:

        Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

```
F=["ap", "ba", "ch"]
for x in fruits:
  print(x)
```

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax of while Loop in Python

```
while test_expression:

   Body of while
```

```
i= 1
while i< 6:
  print(i)
  if i== 3:
    break
  i += 1
```

FUNCTIONS

In Python, function is a group of related statements that perform a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable. Furthermore, it avoids repetition and makes code reusable.

Syntax of Function

```
def function_name(parameters):
        """docstring"""
        statement(s)
```

```
def my_function():
  print("Hellofromafunction")
```
my_function()

function definition which consists of following components.

1. Keyword def marks the start of function header.
2. A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
7. An optional return statement to return a value from the function.

map()

The python **map()** function is used to return a list of results after applying a given function to each item of an iterable(list, tuple etc.)

```
map(function, iterables)
```

STRING

String is a sequence of characters treated as a single unit in python. String is a object of str class. This string class has many constructors. String is surrounded by either single quotation marks, or double quotation marks, or " " ". +(concatenation) operator,*(repetition)operator ,in and not in operators are used in strings. You can assign a multiline string to a variable by using three quotes:

```
>>>a = " " "This is my first program.
           of python lab" " "
>>>S1=str()                    #empty String
>>>S2=str("hello")              # create a string object for hello
>>>S2[0]                          #print 'h'

S2[-1]                     #negative indexing and print 'o'
s2[0]="u"                  #gives error becouse string is a immutable object
S2[1:3:2]                    # print 'el' and string slicing syntax is S1[start index:end index:step]
```

String Function

capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
len()	Returns a length of string
count()	Returns the number of times a specified value occurs in a string
endswith()	Returns true if the string ends with the specified value
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
islower()	Returns True if all characters in the string are lower case
join()	Joins the elements of an iterable to the end of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rstrip()	Returns a right trim version of the string

split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
Swapcase()	Convert the case of string
startswith()	Returns true if the string starts with the specified value
trip()	Returns a trimmed version of the string

LIST

A list is a collection which is ordered and changeable. List class defines lists. In Python lists are written with square brackets. Concatenation(+) operator, *(repetition)operator, in, not in, is operators are used in strings.

```
L1=list()                    #empty list

L2=list([10,20,"abc"])       #lists's constructor to create a list

l3=list(range(0,6))          #list creates with 0 to 5 element

L2[0]                        #print '10'

L2[-1]                       #negative indexing and print 'abc'
L2[0]="u"                    #List is a mutable object.so change the value.
L2[:]                        # print '[10,20,"abc"]' and list slicing syntax is L1[start index:end index:step]
```

List Function

append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
Max()	Returns the element with the greatest value
Min()	Returns the element with the lowest value
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

TUPLE

Python Tuple is used to store the sequence of immutable python objects. Tuple is similar to lists since the value of the items stored in the list can be changed whereas the tuple is immutable and the value of the items stored in the tuple can not be changed. . concatenation(+) operator, *(repetition)operator are used in lists.Tuple don't support all methods supported by lists.

A tuple can be written as the collection of comma-separated values enclosed with the small brackets. A tuple can be defined as follows.

```
    T1 = (1, "python", 22)
    T2 = (10,20,30,40,50)
    T3=()                        # Empty tuple
   T4=("Pyhton")                 #tuple function with String

  T2[-1]            #negative indexing and print '50'
  T2[0]="u"       #tuplet is a immutable object.so cannot change the value.
  T2[:]         # print '(10,20,30,40,50)' and tuple slicing syntax is tuple_name[start index:end index:step]
```

Inbuilt Function

Function	**Description**
cmp(tuple1, tuple2)	Compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.
len(tuple)	Calculates the length of the tuple.
max(tuple)	Returns the maximum element of the tuple.
min(tuple)	Returns the minimum element of the tuple.
tuple(seq)	Converts the specified sequence to the tuple.
Index(x)	Returns index of element x
count(x)	Returns number of occurrences of element x
sum(tuple)	Returns the sum of all the elements in a tuple

List v/s Tuple

List	**Tuple**
The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
The List is mutable.	The tuple is immutable.
The List has the variable length.	The tuple has the fixed length.
The list provides more functionality than tuple.	The tuple provides less functionality than the list.
The list Is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items can not be changed. It can be used as the key inside the dictionary.

Python zip() Function

Python zip() function returns a zip object, which maps a similar index of multiple containers. It takes iterables (can be zero or more), makes it an iterator that aggregates the elements based on iterables passed, and returns an iterator of tuples. it takes item in sequence from number of collections to make a list of tuples,where each tuple contains one item from each container.

```
zip(iterator1, iterator2, iterator3 ...)
```

```
L1=[1,2,3]
L2=[3,4,5]
list(zip(L1,L2))        #[(1,3),(2,4),(3,5)]
```

Inverse zip(*) Function

* operator is used within the zip() function. The * operator unpacks a sequence into positional arguments.for ex

```
# transpose a matrix
matrix=[(1,2),(3,4),(5,6)]
x=zip(*matrix)
tuple(x)          # output  : ((1,3,5),(2,4,6))
```

SET

A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed). The order of elements in a set is undefined though it may consist of various elements. However, the set itself is mutable. We can add or remove items from it. Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.
The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.

Sets can be created by using the built-in **set()** function with an iterable object or a sequence by placing the sequence inside curly braces, separated by ‘comma’. It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary, as its element.

```
set1 = set()            # empty set
set 2= {1, 2, 3}
type(set2)                 # ‘set’
```

Python contains the following methods to be used with the sets.

add(item)	It adds an item to the set. It has no effect if the item is already present in the set
clear()	It deletes all the items from the set.
copy()	It returns a shallow copy of the set.
difference_update(....)	It modifies this set by removing all the items that are also
discard(item)	It removes the specified item from the set.
intersection()	It returns a new set that contains only the common elements of both the sets. (all the sets if more than two are specified).
pop()	Remove and return an arbitrary set element that is the last element of the set. Raises KeyError if the set is empty.
remove(item)	Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.
union(....) are in either set.)	Return the union of sets as a new set.(i.e. all elements that
update()	Update a set with the union of itself and others.

DICTIONARY

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

```
Thisdict={
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

You can access the items of a dictionary by referring to its key name, inside square brackets:

```
x = thisdict["model"]
```

clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

RTU Syllabus Programs:
Experiment 1

Aim: Write a program to demonstrate basic data type in python.
Code:

```
a = 5
print("Type of a: ", type(a))
b = 5.0
print("\nType of b: ", type(b))
c = 2 + 4j
print("\nType of c: ", type(c))

# Creating a String
String1 = 'Welcome to the Geeks World'
print("String with the use of Single Quotes: ")
print(String1)

# Creating a List with
List = ["Geeks", "For", "Geeks"]
print("\nList containing multiple values: ")
print(List[0])
print(List[2])

# Creating a Tuple with the use of list
list1 = [1, 2, 4, 5, 6]
print("\nTuple using List: ")
print(tuple(list1))

# Creating a Set
set1 = set()
set1.add(8)
set1.add(9)
set1.add((6, 7))
print("\nSet after Addition of Three elements: ")
print(set1)
```

Output:

```
Type of a:  <class 'int'>
Type of b:  <class 'float'>
Type of c:  <class 'complex'>
String with the use of Single Quotes:
Welcome to the Geeks World
```

```
List containing multiple values:
Geeks
Geeks

Tuple using List:
(1, 2, 4, 5, 6)

Set after Addition of Three elements:
{8, 9, (6, 7)}
```

Experiment 2

Aim: (a) Write a program to compute distance between two points taking input from the user.

Code:

```
import math
a=int(input("Enter first value"))
b=int(input("Enter second value"))
c=math.sqrt(a**2+b**2)
print("Distance=",c)
```

Output:

```
Enter first value5

Enter second value6
Distance= 7.810249675906654
```

Aim: (b) Write a program add.py that takes 2 numbers as command line arguments and prints its sum.

Code:

```
import sys
a=int(sys.argv[1])
b=int(sys.argv[2])
c=a+b
print("Sum=",c)
```

Output:

```
python add.py 4 5
Sum= 9
```

Experiment 3

Aim: (a) Write a Program for checking whether the given number is an even number or not.

Code:

```
num = int(input("Enter a number: "))
if(num%2==0):
    print("This is an even number.")
else:
    print("This is an odd number.")
```

Output:

```
Enter a number: 4
This is an even number.
```

Aim: (b) Using a for loop, write a program that prints out the decimal equivalents of 1/2, 1/3, 1/4, . . . , 1/10.

Code:

```
for i in range(1,11):
    print ("Decimal equivalent value for 1/",i," is",1/float(i))
```

Output:

```
Decimal equivalent value for 1/ 1  is 1.0
Decimal equivalent value for 1/ 2  is 0.5
Decimal equivalent value for 1/ 3  is 0.3333333333333333
Decimal equivalent value for 1/ 4  is 0.25
Decimal equivalent value for 1/ 5  is 0.2
Decimal equivalent value for 1/ 6  is 0.16666666666666666
Decimal equivalent value for 1/ 7  is 0.14285714285714285
Decimal equivalent value for 1/ 8  is 0.125
Decimal equivalent value for 1/ 9  is 0.1111111111111111
Decimal equivalent value for 1/ 10  is 0.1
```

Experiment 4

Aim: (a) Write a Program to demonstrate list in python (We are given an array of n distinct numbers, the task is to sort all even-placed numbers in increasing and odd-place numbers in decreasing order. The modified array should contain all sorted even-placed numbers followed by reverse sorted odd-placed numbers.)

Code:

```
def evenOddSort(input):

    # separate even odd indexed elements list
    evens = [ input[i] for i in range(0,len(input)) if i%2==0 ]
    odds = [ input[i] for i in range(0,len(input)) if i%2!=0 ]

    # sort evens in ascending and odds in descending using sorted() method
    print (sorted(evens) + sorted(odds,reverse=True))

input = [0, 1, 2, 3, 4, 5, 6, 7]
evenOddSort(input)
```

Output:

```
[0, 2, 4, 6, 7, 5, 3, 1]
```

Aim: (b) Write a Program to demonstrate tuple in python (Given a list of tuples, Write a Python program to remove all the duplicated tuples from the given list).

Code:

```
def removeDuplicates(lst):

    return [t for t in (set(tuple(i) for i in lst))]

# Driver code
lst = [(1, 2), (5, 7), (3, 6), (1, 2)]
print(removeDuplicates(lst))
```

Output:

```
[(1, 2), (5, 7), (3, 6)]
```

Aim: (c) Write a program using a for loop that loops over a sequence.

Code:

```
players=["kohli", "dhoni", "sachin", "sehwag", "Dravid"]
for i in players:
    print (i)
```

Output:

```
kohli
dhoni
sachin
sehwag
Dravid
```

Aim: (d) Write a program using a while loop that asks the user for a number, and prints a countdown from that number to zero.

Code:

```
n=int(input("Enter the number for countdown: "))
while (0<=n):
    print (n, end=" ")
    n=n-1
```

Output:

```
Enter the number for countdown: 15
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

Experiment 5

Aim: (a) Find the sum of all the primes below two million.

Code:

```
n = 2000000
prime = [True for i in range(n+1)]
p = 2
while (p * p <= n):
    if (prime[p] == True):
        for i in range(p * p, n+1, p):
            prime[i] = False
    p += 1
sum=0
for p in range(2, n):
    if prime[p]:
        sum=sum+p
print("sum=", sum)
```

Output:

```
sum= 142913828922
```

Aim: (b) By considering the terms in the Fibonacci sequence whose values do not exceed four million, WAP to find the sum of the even-valued terms.

Code:

```
limit = 4000000
if (limit < 2):
    print("Sum=0")

else:
    ef1 = 0
    ef2 = 2
    sm= ef1 + ef2

    while (ef2 <= limit):
        ef3 = 4 * ef2 + ef1
        if (ef3 > limit):
            break
        ef1 = ef2
        ef2 = ef3
        sm = sm + ef2
    print("Sum=",sm)
```

Output: Sum= 4613732

Experiment 6

Aim: (a) Write a program to count the numbers of characters in the string and store them in a dictionary data structure.

Code:

```
def char_frequency(str1):
    dict = {}
    for n in str1:
        keys = dict.keys()
        if n in keys:
            dict[n] += 1
        else:
            dict[n] = 1
    return dict
print(char_frequency('google.com'))
```

Output:

```
 {'c': 1, 'e': 1, 'g': 2, 'm': 1, 'l': 1, 'o': 3, '.': 1}
```

Aim: (b) Write a program to use split and join methods in the string and trace a birthday of a person with a dictionary data structure.

Code:

```
dob={"mothi":"12-11-1990","sudheer":"17-08-1991","vinay":"31-08-1988"}
str1=input("which person dob you want: ")
l=str1.split()
birth=""
for i in l:
    if i in dob.keys():
        name=i
print (" ".join([name,"Birthday is",dob[name]]))
```

Output:

```
which person dob you want: i want vinay dob
vinay Birthday is 31-08-1988
```

Experiment 7

Aim: Write a program to count frequency of characters in a given file. Can you use character frequency to tell whether the given file is a Python program file, C program file or a text file?

Code:

```
import os
f=open("deepa.py")
count=dict()
for line in f:
  for ch in line:
    if ch in count:
      count[ch]=count[ch]+1
    else:
      count[ch]=1
print (count)
filename,file_extension=os.path.splitext("deepa.py");
print("file_extension==",file_extension);
if(file_extension=='.py'):
  print("its python program file");
elif(file_extension==".txt"):
  print("its a txt file");
elif(file_extension==".c"):
  print("its a c program file");
f.close()
```

deepa.py:

```
my name is deepa modi
```

Output:

```
{'m': 3, 'y': 1, ' ': 4, 'n': 1, 'a': 2, 'e': 3, 'i': 2, 's': 1, 'd': 2, 'p': 1, 'o': 1}
file_extension== .py
its python program file
```

Experiment 8

Aim: (a) Write a program to print each line of a file in reverse order.

Code:

```
filename=input("Enter the filename: ")
f=open(filename,"r")
for line in f:
  line2=""
  for ch in range(len(line)-1,-1,-1):
    line2=line2+line[ch]
  print(line2)
f.close()
```

deepa.py:

```
my name is deepa modi
i am a cool person
```

Output:

```
Enter the filename: deepa.py
idom apeed si eman ym
nosrep looc a ma i
```

Aim: (b) Write a program to compute the number of characters, words and lines in a file.

Code:

```
filename=input("Enter the filename: ")
f=open(filename,"r")
l=w=c=0
for line in f:
  words=line.split()
  l=l+1
  for word in words:
    w=w+1
    for ch in word:
      c=c+1
print("No. of lines",l)
print("No. of words",w)
print("No. of characters",c)
f.close()
```

deepa.py:

```
my name is deepa modi
i am a cool person
```

Output:

```
Enter the filename: deepa.py
No. of lines 2
No. of words 10
No. of characters 31
```

Experiment 9

Aim: (a) Write a function nearly equal to test whether two strings are nearly equal. Two strings a and b are nearly equal when a can be generated by a single mutation on.

Code:

```
def mutate(word):
  out_list = []
  letters = 'abcdefghijklmnopqrstuvwxyz'
  #insert a character
  for i in range(len(word) + 1):
    for j in range(26):
      out_list.append(word[:i] + letters[j] + word[i:])
  #deleting a character
  for i in range(len(word)):
    out_list.append(word[:i] + word[i + 1:])
  #replace a character
  for i in range(len(word)):
    for j in range(26):
      out_list.append(word[:i] + letters[j] + word[i + 1:])
  #swapping a characters
  current_word = []
  out_word = ''
  for i in range(len(word) - 1):
    for j in range(i + 1, len(word)):
      #converting string into list
      cword = list(word)
      #Swapping of characters in a list
      cword[i], cword [j] = cword [j], cword [i]
      #converting list into string
      str1="".join(current_word)
      out_list.append(str1)
  return out_list

def nearly_equal(word1, word2):
  if len(word1)<len(word2):
    word1,word2=word2,word1
    return word1 in mutate(word2)
  else:
    return word1 in mutate(word2)

a=input("Enter First Word: ")
b=input("Enter Second Word: ")
print(nearly_equal(a,b))
```

Output:

```
Enter First Word: deepa
Enter Second Word: dipa
False

Enter First Word: welcome
Enter Second Word: welcme
True
```

Aim: (b) Write function to compute gcd, lcm of two numbers. Each function shouldn't exceed one line.

Code:

```
def gcd(x,y):
  return x if y==0 else gcd(y,x%y)
def lcm(x,y):
  return (x*y)//gcd(x,y)
print ("gcd is",gcd(54,24))
print ("lcm is",lcm(54,24))
```

Output:

```
gcd is 6
lcm is 216
```

Experiment 10

Aim: (a) Write a program to implement Merge sort.

Code:

```
def mergeSort(nlist):
    #print("Splitting ",nlist)
    if len(nlist)>1:
        mid = len(nlist)//2
        lefthalf = nlist[:mid]
        righthalf = nlist[mid:]

        mergeSort(lefthalf)
        mergeSort(righthalf)
        i=j=k=0
        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i] < righthalf[j]:
                nlist[k]=lefthalf[i]
                i=i+1
            else:
                nlist[k]=righthalf[j]
                j=j+1
            k=k+1

        while i < len(lefthalf):
            nlist[k]=lefthalf[i]
            i=i+1
            k=k+1

        while j < len(righthalf):
            nlist[k]=righthalf[j]
            j=j+1
            k=k+1
    #print("Merging ",nlist)

nlist = [14,46,43,27,57,41,45,21,70]
mergeSort(nlist)
print(nlist)
```

Output:

```
[14, 21, 27, 41, 43, 45, 46, 57, 70]
```

Aim: (b) Write a program to implement Selection sort.

Code:

```
def selectionSort(nlist):
   for fillslot in range(len(nlist)-1,0,-1):
       maxpos=0
       for location in range(1,fillslot+1):
           if nlist[location]>nlist[maxpos]:
               maxpos = location

       temp = nlist[fillslot]
       nlist[fillslot] = nlist[maxpos]
       nlist[maxpos] = temp

nlist = [14,46,43,27,57,41,45,21,70]
selectionSort(nlist)
print(nlist)
```

Output:

```
[14, 21, 27, 41, 43, 45, 46, 57, 70]
```

Aim: (c) Write a program to implement Insertion sort.

Code:

```
def insertionSort(nlist):
   for index in range(1,len(nlist)):

     currentvalue = nlist[index]
     position = index

     while position>0 and nlist[position-1]>currentvalue:
         nlist[position]=nlist[position-1]
         position = position-1

     nlist[position]=currentvalue

nlist = [14,46,43,27,57,41,45,21,70]
insertionSort(nlist)
print(nlist)
```

Output:

```
[14, 21, 27, 41, 43, 45, 46, 57, 70]
```

Beyond Syllabus Programs:

Experiment 1

Aim : Sort a list according to the second element in sublist.

Code:

```
def Sort(sub_li):
    sub_li.sort(key = lambda x: x[1])
    return sub_li

sub_li =[['rishav', 10], ['akash', 5], ['ram', 20], ['gaurav', 15]]
print(Sort(sub_li))
```

Output:

```
[['akash', 5], ['rishav', 10], ['gaurav', 15], ['ram', 20]]
```

Experiment 2

Aim: Split the Even and Odd elements into two different lists.

Code:

```
def Split(mix):
    ev_li = []
    od_li = []
    for i in mix:
        if (i % 2 == 0):
            ev_li.append(i)
        else:
            od_li.append(i)
    print("Even lists:", ev_li)
    print("Odd lists:", od_li)

# Driver Code
mix = [2, 5, 13, 17, 51, 62, 73, 84, 95]
Split(mix)
```

Output:

```
Even lists: [2, 62, 84]
Odd lists: [5, 13, 17, 51, 73, 95]
```

Experiment 3

Aim: Given an array of n integers where each value represents number of chocolates in a packet. Each packet can have variable number of chocolates. There are m students, the task is to distribute chocolate packets such that:

1. Each student gets one packet.
2. The difference between the number of chocolates in packet with maximum chocolates and packet with minimum chocolates given to the students is minimum.

Code:

```
import sys;
def findMinDiff(arr, n, m):
    if (m==0 or n==0):
        return 0
    arr.sort()
    if (n < m):
        return -1
    min_diff = sys.maxsize
    first = 0
    last = 0
    i=0
    while(i+m-1<n ):
         diff = arr[i+m-1] - arr[i]
        if (diff < min_diff):
            min_diff = diff
            first = i
            last = i + m - 1
        i+=1
    return (arr[last] - arr[first])

arr = [12, 4, 7, 9, 2, 23, 25, 41,30, 40, 28, 42, 30, 44, 48, 43, 50]
m = 7 # Number of students
n = len(arr)
print("Minimum difference is", findMinDiff(arr, n, m))
```

Output:

```
Minimum difference is 10
```

Experiment 4

Aim: Given a value N, if we want to make change for N cents, and we have infinite supply of each of S = { S1, S2, .. , Sm} valued coins, how many ways can we make the change? The order of coins doesn\’t matter.

For example, for N = 4 and S = {1,2,3}, there are four solutions: {1,1,1,1},{1,1,2},{2,2},{1,3}. So output should be 4.

Code:

```
def count(S, m, n):
    table = [0 for k in range(n+1)]
    table[0] = 1
    for i in range(0,m):
        for j in range(S[i],n+1):
            table[j] += table[j-S[i]]

    return table[n]

arr = [1, 2, 3]
m = len(arr)
n = 4
x = count(arr, m, n)
print (x)
```

Output:

```
4
```

Experiment 5

Aim: Python Program for Extended Euclidean algorithms.

Code:

```
def gcdExtended(a, b, x, y):
    # Base Case
    if a == 0 :
        x = 0
        y = 1
        return b

    x1 = 1
    y1 = 1 # To store results of recursive call
    gcd = gcdExtended(b%a, a, x1, y1)

    x = y1 - (b/a) * x1
    y = x1

    return gcd


x = 1
y = 1
a = 35
b = 15
g = gcdExtended(a, b, x, y)
print("gcd(", a , "," , b, ") = ", g)
```

Output:

```
gcd( 35 , 15 ) =  5
```

Experiment 6

Aim: Python Program for GCD of more than two (or array) numbers.

Code:

```
def find_gcd(x, y):
    while(y):
        x, y = y, x % y

    return x

l = [2, 4, 6, 8, 16]

num1=l[0]
num2=l[1]
gcd=find_gcd(num1,num2)

for i in range(2,len(l)):
    gcd=find_gcd(gcd,l[i])

print(gcd)
```

Output:

```
2
```

Experiment 7

Aim: Python Program for Check if all digits of a number divide itself.

Code:

```
def checkDivisibility(n, digit) :
    return (digit != 0 and n % digit == 0)
def allDigitsDivide( n) :
    temp = n
    while (temp > 0) :

        # Taking the digit of
        # the number into digit
        # var.
        digit = n % 10
        if ((checkDivisibility(n, digit)) == False) :
            return False
        temp = temp // 10
    return True

n = 128
if (allDigitsDivide(n)) :
    print("Yes")
else :
    print("No" )
```

Output:

```
Yes
```

Experiment 8

Aim: Python program to check if a string contains all unique characters.

Code:

```
def isUniqueChars(st):
    if len(st) > 256:
      return False
    char_set = [False] * 128

    for i in range(0, len(st)):
        val = ord(st[i])
        if char_set[val]:
            return False
        char_set[val] = True
        return True

st = "abcd"
print(isUniqueChars(st))
```

Output:

```
True
```

Experiment 9

Aim: Find all close matches of input string from a list

Code:

```
from difflib import get_close_matches

def closeMatches(patterns, word):
     print(get_close_matches(word, patterns))

# Driver program
if __name__ == "__main__":
    word = 'appel'
    patterns = ['ape', 'apple', 'peach', 'puppy']
    closeMatches(patterns, word)
```

Output:

```
['apple', 'ape']
```

Experiment 10

Aim: Permutation of a given string using inbuilt function

Code:

```
from itertools import permutations

def allPermutations(str):

     # Get all permutations of string 'ABC'
     permList = permutations(str)

     # print all permutations
     for perm in list(permList):
         print (''.join(perm))

# Driver program
if __name__ == "__main__":
    str = 'ABC'
    allPermutations(str)
```

Output:

```
ABC
ACB
BAC
BCACAB
CBA
```

Experiment 11

Aim: Find common elements in three sorted arrays by dictionary intersection.

Code:

```
# Function to find common elements in three
# sorted arrays
from collections import Counter

def commonElement(ar1,ar2,ar3):
    # first convert lists into dictionary
    ar1 = Counter(ar1)
    ar2 = Counter(ar2)
    ar3 = Counter(ar3)

    # perform intersection operation
    resultDict = dict(ar1.items() & ar2.items() & ar3.items())
    common = []

    # iterate through resultant dictionary
    # and collect common elements
    for (key,val) in resultDict.items():
        for i in range(0,val):
            common.append(key)

    print(common)

# Driver program
if __name__ == "__main__":
   ar1 = [1, 5, 10, 20, 40, 80]
   ar2 = [6, 7, 20, 80, 100]
   ar3 = [3, 4, 15, 20, 30, 70, 80, 120]
   commonElement(ar1,ar2,ar3)
```

Output:

```
[80, 20]
```

Experiment 12

Aim: Python program to convert time from 12 hour to 24 hour format.

Code:

```
# Python program to convert time
# from 12 hour to 24 hour format

# Function to convert the date format
def convert24(str1):

    # Checking if last two elements of time
    # is AM and first two elements are 12
    if str1[-2:] == "AM" and str1[:2] == "12":
        return "00" + str1[2:-2]

    # remove the AM
    elif str1[-2:] == "AM":
        return str1[:-2]

    # Checking if last two elements of time
    # is PM and first two elements are 12
    elif str1[-2:] == "PM" and str1[:2] == "12":
        return str1[:-2]

    else:

        # add 12 to hours and remove PM
        return str(int(str1[:2]) + 12) + str1[2:8]

# Driver Code
print(convert24("08:05:45 PM"))
```

Output:

```
20:05:45
```

Some Useful Links:

1. https://docs.python.org/3/tutorial/
2. https://www.w3schools.com/python/
3. https://www.tutorialspoint.com/python/index.htm
4. https://www.javatpoint.com/python-tutorial
5. https://www.geeksforgeeks.org/python-programming-language/