

YSPM's

Yashoda Technical Campus,

Faculty of Engineering, Satara.

Department of Computer Science and Engineering

Python Programming

Lab Manual

For S.Y. (CSE) – Semester IV

Prepared By:

Prof. Asma S. Mulla

Assistant Professor (CSE Dept.)

YSPM's
Yashoda Technical Campus,

Faculty of Engineering, Satara.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Python Programming
(Dr. Babasaheb Ambedkar Technological University, Lonere Subject Code:

For S.Y. (CSE) – Semester IV

Teaching Scheme

Lecture (L)	Tutorial (T)	Practical (P)	Total
1	-	8	9

Examination Scheme (Practical)

MSE	CA	ESE	Credit
-	60	40	3

Course Outcomes (COs)

YSPM's

Yashoda Technical Campus, Faculty of Engineering, Satara.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

Python Programming

(BATU University Lonere Subject Code: BTCOL406) For S.Y. (CSE) – Semester IV

INDEX

Sr. No.	Description	Page No.
1.	Program to calculate area of triangle, rectangle, circle.	
2.	Program to find the union of two list	
3.	Program to find the intersection of two lists	
4.	Program to remove the "i" th occurrence of the given word in a list where words repeat.	
5.	Program to count the occurrences of each word in a given string sentence	
6.	Program to check if a substring is present in a given string.	
7.	Program to map two lists into a dictionary.	
8.	Program to count the frequency of words appearing in a string using a dictionary	
9	Program to create a dictionary with key as first character and value as words starting with that character.	
10	Program to find the length of a list using recursion	
11	compute the diameter, circumference, and volume of a sphere using class	
12	Program to read a file and capitalize the first letter of every word in the file.	

Experiment No: 1

Aim: Program to calculate area of triangle, rectangle, circle.

Objectives:

1) To learn Arithmetic Operators.
2) To learn Structure of Python Program.
3) To learn Basic programming in Python

Hardware/Software Required:

1. Python 3

Prerequisites:

1. Python Installation.
2. Working of Variables, input Function and Arithmetic Operators.

Theory:

Variables-

A Python variable is a symbolic name that is a reference or pointer to an object. Once an object is assigned to a variable, you can refer to the object by that name. But the data itself is still contained within the object.

Python Variable is containers that store values. Python is not "statically typed". We do not need to declare variables before using them or declare their type. A variable is created the moment we first assign a value to it. A Python variable is a name given to a memory location. It is the basic unit of storage in a program.

For example:

```
>>> n = 300
```

Input function-

Python has an input function which lets you ask a user for some text input. You call this function to tell the program to stop and wait for the user to key in the data. In Python 2, you have a built-in function raw_input(), whereas in Python 3, you have input(). The program will resume once the user presses the ENTER or RETURN key. Look at this example to get input from the

keyboard using Python 2 in the interactive mode. Your output is displayed in quotes once you hit the ENTER key.

For example:

>>>raw_input()

I am learning at YSPM (This is where you type in)

'I am learning at YSPM ' (The interpreter showing you how the input is captured.)

print() Function-

The print() method prints the given object to the console or to the text stream file.

print() syntax:

print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

Parameters:

1. objects: One or more objects to be printed, seperated by a space ' ' by default.
2. Sep: (Optional) If multiple objects passed, they are separated by the specified separator. Default is ' '.
3. end: (Optional) The last value to print. Default is '\n'.
4. file: (Optional) Must be an object with write(string) method. Default is sys.stdout.
5. flush: (Optional) The stream is forcibly flushed if buffered. Default is False.

The following example demonstrates the print() function.

Example:

```
print("Learning Python")
name = 'John'
print("My name is",name)
```

Output-

Learning Python

My name is John

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

Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

To calculate area of rectangle -

To calculate area of rectangle we have to accept two inputs from user which are breadth (b) and length (l). After getting input from user we have to declare another variable (triangle_area) to store the area calculated using some arithmetic operators (*). Lastly, we can display the output using print() function. To calculate area of triangle we have to use formula-

rectangle_area=b*l

To calculate area of Triangle-

To calculate area of triangle we have to accept two inputs from user which are base (b) and higth (h). After getting input from user we have to declare another variable (triangle_area) to store the area calculated using some arithmetic operators (*, /). Lastly, we can display the output using print() function. To calculate area of triangle we have to use formula-

triangle_area=(b*h)/2

To calculate area of Circle-

To calculate area of circle we have to accept one input from user which is radius (r). After getting input from user we have to declare another variable (circle_area) to store the area calculated using some arithmetic operators (*). Lastly, we can display the output using print() function. To calculate area of triangle we have to use formula-

circle_area=3.14*r*r

Program:

```
l=int(input("Enter value of length"))
b=int(input("Enter value of breadth"))
rectangle_area=b*l
print("Area of rectangle=", rectangle_area)

h=int(input("Enter value of height"))
b=int(input("Enter value of base"))
triangle_area=(b*h)/2
print("Area of triangle =",triangle_area)

r=int(input("Enter value of radius"))
circle_area=3.14*r*r
print("Area of circle =",circle_area)
```

Output:

Enter value of length: 3

Enter value of breadth: 4

Area of rectangle=12

Enter value of height: 3

Enter value of base: 4

Area of triangle =6

Enter value of height: 3

Area of circle =28.27

Conclusion: In this way We have Learned the Implementation of Variables, Input function and Arithmetic operators in Python

Experiment No: 3

Aim: Program to find intersection of two lists

Objectives:

Hardware/Software Required:

1. Python 3

Prerequisites:

Theory:

Def function-

A function is a collection of related assertions that performs a mathematical, analytical, or evaluative operation. A collection of statements called Python Functions returns the particular task. Python functions are simple to define and essential to intermediate-level programming. The exact criteria hold to function names as they do to variable names. The goal is to group up certain often performed actions and define a function. We may call the function and reuse the code contained within it with different variables rather than repeatedly creating the same code block for different input variables.

Intersection-

A **list** is an arranged collection of elements. It is used to store collections of data. It can contain a list of various types of data objects with a comma separated and enclosed within a square bracket.

The intersection of two lists contains the elements that the two lists have in common. The returned list contains only items that exist in both lists, or in all lists, if the comparison is done with more than two lists. There are various ways through which we can perform the intersection of two lists. Here we have mentioned most of them.

Intersection of two list means we need to take all those elements which are common to both of the initial lists and store them into another list. Now there are various ways in Python, through which we can perform the Intersection of the lists.

Python is known for its excellent built-in data structure. Python list is one of the famous and valuable built-in data types of Python. It can store the various data-types value in sorted order. However, there is no built-in function for lists like sets.

Python provides the many ways to perform the intersection of the lists. Let's see the following scenario.

Examples:

Input:

```
lst1 = [15, 9, 10, 56, 23, 78, 5, 4, 9]
lst2 = [9, 4, 5, 36, 47, 26, 10, 45, 87]
```

Output:

[9, 10, 4, 5]

Input:
lst1 = [4, 9, 1, 17, 11, 26, 28, 54, 69]
lst2 = [9, 9, 74, 21, 45, 11, 63, 28, 26]

Output:

[9, 11, 26, 28]

Program:

```
L1= []
Num1=int (input ("Enter size of list 1:"))
for n in range (Num1):
   a=int (input ("Enter number:"))
   L1. append(a)

L2= []
Num2=int (input ("Enter size of list2:"))
for  m in range (Num2):
   b=int (input ("Enter number:"))
   L2. append(b)

def Intersection (L1, L2):
   return set(L1). intersection(L2)
print ("Interaction of two lists =", Intersection (L1, L2))
```

Output:

```
Enter size of list 1:4
Enter number:1
Enter number:2
Enter number:3
Enter number:4
Enter size of list2:5
Enter number:3
Enter number:4
Enter number:5
Enter number:6
Enter number:7
Interaction of two lists = {3, 4}
```

Experiment no. 4

Aim: Program to remove the ‘i’th occurrence of the given word in a list where words repeat.

Theory:

List -

Python lists are both mutable and dynamic. Many types in Python are immutable. Integers, floats, strings, and (as you’ll learn later in this course) tuples are all immutable. Once one of these objects is created, it can’t be modified, unless you reassign the object to a new value.
The list is a data type that is mutable. Once a list has been created:

- Elements can be modified.
- Individual values can be replaced.
- The order of elements can be changed.
- Lists are also dynamic. Elements can be added and deleted from a list, allowing it to grow or shrink:

Example:
>>> a = ['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
>>> a
['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']

List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

Example

Lists allow duplicate values:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

List Length

To determine how many items a list has, use the `len()` function:

Example

Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

A list can contain different data types:

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]
```

Experiment no. 5

Aim: Program to count the occurrences of each word in a given string sentence.

Theory:

Python String-

A string is a data structure in Python that represents a sequence of characters. It is an immutable data type, meaning that once you have created a string, you cannot change it. Strings are used widely in many different applications, such as storing and manipulating text data, representing names, addresses, and other types of data that can be represented as text.

Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes.

Creating strings is as simple as assigning a value to a variable.

Example:
'welcome', 'thank you', etc

Python String split() Method:

The `split ()` method splits a string into a list.

You can specify the separator, default separator is any whitespace.

Python for loop with string-

In the Python programming language, for loops are also called "definite loops" because they perform the instruction a certain number of times.
This is in contrast to while loops, or indefinite loops, which execute an action until a condition is met and they are told to stop.
For loops are useful when you want to execute the same code for each item in a given sequence. With a for loop, you can iterate over any iterable data such as lists, sets, tuples, dictionaries, ranges, and even strings.
You can iterate over string as shown below:

```
name = "YSPM"
for letter in name:
    print(letter)
```

This will print all the letters in the string individually:

```
# Output:
# Y
# S
# P
# M
```

Program code-

```
string=input("Enter string:")
a=[]
```

```
b=[]
c=[]
count=0
a=string.split(" ")
for i in range(len(a)):
 if a[i] in c:
    b.append(a[i])
 else:
    count=0
    if(a[i]!=" "):
       c.append(a[i])
       for j in range (0,len(a)):
          if(a[i]==a[j]):
             count=count+1
       print("Occurrence of ",a[i],"=",count)
```

Output-

```
Enter string: Satara is My Dream City is Satara
Occurrence of  Satara = 2
Occurrence of  is = 2
Occurrence of  My = 1
Occurrence of  Dream = 1
Occurrence of  City = 1
```

Experiment no. 6

Aim: Program to check if a substring is present in a given string.

Theory:

'in' operator in Python-

In Python, the in operator determines whether a given value is a constituent element of a sequence such as a string, array, list, or tuple.

When used in a condition, the statement returns a Boolean result of True or False. The statement returns True if the specified value is found within the sequence. When it is not found, we get a False.

In this article, we will show how the in operator works on a list in different ways using Python. Here we will see 3 different scenarios −

- To find a single element/object in a single list/flat list.
- To find multiple elements list in a nested list.
- Usage of in operator with if statement.

Assume we have taken a list containing random elements.

Method 1: To find a single element/object in a single list/flat list

Algorithm (Steps)

Following are the Algorithm/steps to be followed to perform the desired task −

- Create a variable to store the input list and give it some random values.
- Check if the element is present in list or not using in operator and print it.

The following program checks whether the single element is present in the flat list or not using the in operator −

```
lst = ["Hello", 10, "CSE", 20, "python", "code"]
print("CSE" in lst)
print("bigdata" in lst)
```

Output

On executing, the above program will generate the following output −

```
True
False
```

To begin, we have filled a list lst with random values. The in operator is then used to determine whether or not some values are part of the previous sequence.

As we can see from the output above, "CSE" in the list evaluates to True. This indicates that the value "CSE" can be found within the list.

The term "bigdata" in the list evaluates to False. This means that the value "bigdata" was not found in the list.

Method 2: To find multiple elements list in a nested

Algorithm (Steps)

Following are the Algorithm/steps to be followed to perform the desired task −

- Create a variable to store the input nested list and give it some random list values.
- Check if the list is present in the nested list or not using in operator and print it.

```
lst = [["Hello", 10], ["CSE", 20], ["python", "code"]]

print(["CSE",20] in lst)

print(["CSE","code"] in lst)
```

Output

On executing, the above program will generate the following output −

```
True
False
```

Method 3: Usage of in operator with if statement.

Algorithm (Steps)

Following are the Algorithm/steps to be followed to perform the desired task −

Create a variable to store the input nested list and give it some random list values.

Check if the element is present in the list or not using the in operator and if statement.

The if statement executes only if the result returned by the in operator is true i.e if the element is present in the list.

Print the result tuple after conversion from a list into a tuple.

Print the data type of the result tuple using the type() function for verification. Write the else statement if the element is not present in the list.

```
lst = ["Hello", 10, "CSE", 20, "python", "code"]
if "CSEt" in lst:
   print('{CSE} Element is in the given list')
if "bigdata" in lst:
   print('{bigdata} Element is in the given list')
else:
   print('{bigdata} Element is not present in the given list')
```

Output
On executing, the above program will generate the following output −

{CSE} Element is in the given list
{bigdata} Element is not present in the given list

Program Code-

```
x=input("Enter Any String = ")
y=input("Enter Any Substring = ")
if y in x:
        print(y,"is present in string",x)
else:
        print(y,"is not present in string",x)
```

Output-

```
Enter Any String = Satara
Enter Any Substring = tara
tara is present in string Satara
```

Experiment no. 7

Aim: Program to map two lists into a dictionary.

Theory-

Append() Function-

The append() function in Python takes a single item as an input parameter and adds it to the end of the given list. In Python, append() doesn't return a new list of items; in fact, it returns no value at all. It just modifies the original list by adding the item to the end of the list.

After executing append() on a list, the size of the original list increases by one. The item in the list can be a string, number, dictionary, or even another list (because a list is an object too). When a list is appended onto the original list, it is added as a single object. The addition of the appended list happens, as usual, at the end of the original list.

Syntax

```
list.append(item)
```

list.append(item)

- **Parameters:** item is the only parameter append() takes, and it is the item to be added at the end of the list.
- **Returns:** append() doesn't return any value. It just adds the item to the end of the list.

Example-

```
stringList = ['mon', 'tue', 'wed', 'thu']
print(stringList)
stringList.append('fri')

print(stringList)
```

Ouput-

```
['mon', 'tue', 'wed', 'thu']

['mon', 'tue', 'wed', 'thu', 'fri']
```

Update() Function-

Python update() method updates the dictionary with the key and value pairs. It inserts key/value if it is not present. It updates key/value if it is already present in the dictionary.

It also allows an iterable of key/value pairs to update the dictionary. like: update(a=10,b=20) etc.

Signature and examples of this method are given below.

Signature

update([other])

Parameters

other: It is a list of key/value pairs.

Return

It returns None.

Example-

```
einventory = {'Fan': 200, 'Bulb':150, 'Led':1000}
print("Inventory:",einventory)
```

```
        # Calling Method
        einventory.update({'cooler':50})
        print("Updated inventory:",einventory)
```

Output:

```
Inventory: {'Fan': 200, 'Bulb': 150, 'Led': 1000}
Updated inventory: {'Fan': 200, 'Bulb': 150, 'Led': 1000, 'cooler': 50}
```

Program code-

```
#getting the size of lists from user
size=int(input('Enter the size of lists--'))
list1=[]
list2=[]
dict={}
#getting elements of 1st list from user
for i in range (size):
        i=input('Enter element for 1st list=')
        list1.append(i)

#getting elements of 2st list from user
for i in range(size):
        i=input('Enter element for 2nd list=')
        list2.append(i)

#Mapping lists in dictionary
for i in range (size):
        dict.update({list1[i]:list2[i]})

print(dict)
```

Output-

```
Enter the size of lists--3
Enter element for 1st list=Roll no
Enter element for 1st list=name
Enter element for 1st list=Address
Enter element for 2nd list=5
Enter element for 2nd list=ABC
Enter element for 2nd list=Satara
{'Roll no': '5', 'name': 'ABC', 'Address': 'Satara'}
```

Experiment No. 8

Aim: Program to count the frequency of words appearing in a string using a dictionary.

Theory:

Split() method-
The split() method splits a string at the specified separator and returns a list of substrings.
Example

```
cars = 'BMW-Telsa-Range Rover'
# split at '-'
print(cars.split('-'))
```

Output: ['BMW', 'Tesla', 'Range Rover']

Syntax of String split()
The syntax of split() is:
str.split(separator, maxsplit)

---

split() Parameters
The split() method takes a maximum of 2 parameters:

- **separator** (optional)- Delimiter at which splits occur. If not provided, the string is splitted at whitespaces.
- **maxsplit** (optional) - Maximum number of splits. If not provided, there is no limit on the number of splits.

---

split() Return Value
The split() method returns a list of strings.

Example 1: How split() works in Python?

```
text= 'Love thy neighbor'
# splits at space
print(text.split())
grocery = 'Milk, Chicken, Bread'
# splits at ','
print(grocery.split(', '))
# Splits at ':'
print(grocery.split(':'))
```

Output

```
['Love', 'thy', 'neighbor']
```

['Milk', 'Chicken', 'Bread']
['Milk, Chicken, Bread']
Here,

- text.split() - splits string into a list of substrings at each space character
- grocery.split(', ') - splits string into a list of substrings at each comma and space character
- grocery.split(':') - since there are no colons in the string, split() does not split the string.

Count()-

Python is a high-level, interpreted programming language that has gained immense popularity in data science, machine learning, and web development. One of Python's most useful built-in functions is the count() function, which allows you to count the number of occurrences of a particular element in a python list count or a tuple. In this article, we will learn how to use the count function in Python and explore its practical applications.

Python's count() function is a built-in function that allows you to count the number of times an element appears in a list or tuple. This function can be handy when dealing with large datasets or when you need to perform calculations based on the frequency of certain elements

Zip()-

Python’s zip() function creates an iterator that will aggregate elements from two or more iterables. You can use the resulting iterator to quickly and consistently solve common programming problems, like creating dictionaries.

Python’s zip() function is defined as zip(*iterables). The function takes in iterables as arguments and returns an iterator. This iterator generates a series of tuples containing elements from each iterable. zip() can accept any type of iterable, such as files, lists, tuples, dictionaries, sets, and so on.

Passing n Arguments
If you use zip() with n arguments, then the function will return an iterator that generates tuples of length n. To see this in action, take a look at the following code block:

```
>>> numbers = [1, 2, 3]
>>> letters = ['a', 'b', 'c']
>>> zipped = zip(numbers, letters)
>>> zipped  # Holds an iterator object
<zip object at 0x7fa4831153c8>
>>> type(zipped)
<class 'zip'>
>>> list(zipped)
[(1, 'a'), (2, 'b'), (3, 'c')]
```

Here, you use zip(numbers, letters) to create an iterator that produces tuples of the form (x, y). In this case, the x values are taken from numbers and the y values are taken from letters. Notice how the Python zip() function returns an iterator. To retrieve the final list object, you need to use list() to consume the iterator.

Program Code-

```
my_string = input("Enter the string :")
```

```
my_list=[]
my_list=my_string.split()
word_freq=[my_list.count(p) for p in my_list]
print("The frequency of words is ...")
print(dict(zip(my_list, word_freq)))
```

Output-

```
Enter the string :Hii Alexa how are you Alexa
The frequency of words is ...
{'Hii': 1, 'Alexa': 2, 'how': 1, 'are': 1, 'you': 1}
```

Experiment No. 9

Aim: Program to create a dictionary with key as first character and value as words starting with that character.

Theory:

Membership Operators:

Python's in and not in operators allow you to quickly determine if a given value is or isn't part of a collection of values. This type of check is common in programming, and it's generally known as a membership test in Python. Therefore, these operators are known as membership operators.

Arguably, the natural way to perform this kind of check is to iterate over the values and compare them with the target value. You can do this with the help of a for loop and a conditional statement.

Append()

It is easy to append elements to the existing dictionary using the dictionary name followed by square brackets with a key inside it and assigning a value to it.

Here's an example:

```
my_dict = {"username": "ABC", "email": "abc@gmail.com", "location":"Gurgaon"}
my_dict['name']='Nick'
print(my_dict)
```

The output is:

{‘username’: ‘ABC’, ’email’: ‘abc@gmail.com’, ‘location’: ‘Gurgaon’, ‘name’: ‘Nick’}

Items ()

A dictionary in Python is an unordered collection of items. As in a real-life dictionary, each word is associated with its meaning, in the Python dictionary, each key is paired with its respective value. Thus, the data in the dictionary is stored in pairs of key and value.
Syntax to create a dictionary: name of dictionary = {key:value}

The items, that is, the data that is stored in the dictionary are unordered and mutable. Thus, they can be changed after the creation of the dictionary. However, the items cannot be duplicated within the same dictionary. These items can be of any data type, including numeric and characters such as integers, strings, floats, complex numbers. Boolean type, etc.

Items () Parameters

The items () method in the dictionary is used to return each item in a dictionary as tuples in a list. Thus, the dictionary key and value pairs will be returned in the form of a list of tuple pairs.

Syntax of items () method: *dictionary_name.items ()*

The items () method does not take any parameters.

Return Value from items ()

When you use the items () method on a dictionary, the key and value pair stored in it will be displayed in the form of tuples in a list. Note that the returned list is a view of the items stored in the dictionary. The method does not change a dictionary into a list. Also, all the changes that will be done in the dictionary will be shown in the list view as well.

Program Code-

```
my_string=input("Enter the string :")
split_string = my_string.split()
my_dict={}
for elem in split_string:
  if(elem[0] not in my_dict.keys()):
        my_dict[elem[0]]=[]
          my_dict[elem[0]].append(elem)
  else:
           if(elem not in my_dict[elem[0]]):
           my_dict[elem[0]].append(elem)
print("The dictionary created is: ",my_dict)
```

Output-

Enter the string :Hello Everyone Welcome to YSPM Thank you

The dictionary created is: {'H': ['Hello'], 'E': ['Everyone'], 'W': ['Welcome'], 't': ['to'], 'Y': ['YSPM'], 'T': ['Thank'], 'y': ['you']}

Experiment No. 10

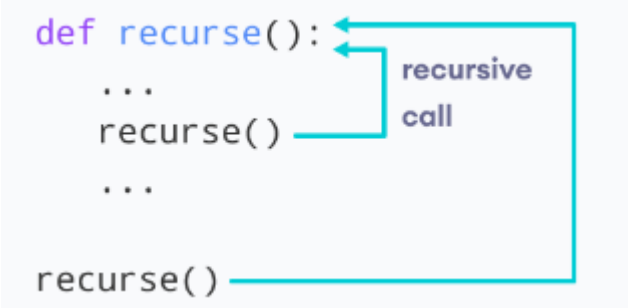
Aim: Program to find the length of the given list using recursion.

Theory:

Python Recursion—

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

The following image shows the working of a recursive function called recurse.



Following is an example of a recursive function to find the factorial of an integer.

Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is `1*2*3*4*5*6 = 720`.

Example of a recursive function

```
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))


num = 3
print("The factorial of", num, "is", factorial(num))
```

Run Code

Output

```
The factorial of 3 is 6
```

In the above example, `factorial()` is a recursive function as it calls itself.

When we call this function with a positive integer, it will recursively call itself by decreasing the number.

Each function multiplies the number with the factorial of the number below it until it is equal to one.

Advantages of Recursion

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

Program Code-

```
def list_length(my_list):
  if not my_list:
    return 0
  return 1 + list_length(my_list[1::2]) + list_length(my_list[2::2])
my_list = [1, 2, 3, 11, 34, 52, 78]
print("The list is :")
print(my_list)
print("The length of the list is : ")
print(list_length(my_list))
```

Output—

```
The list is :
[1, 2, 3, 11, 34, 52, 78]
The length of the list is :
7
```

Experiment No. 11

Aim: Compute the diameter, circumference and volume of a sphere using class.

Theory:

CLASSES-

Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

Compared with other programming languages, Python's class mechanism adds classes with a minimum of new syntax and semantics. It is a mixture of the class mechanisms found in C++ and Modula-3. Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Objects can contain arbitrary amounts and kinds of data. As is true for modules, classes partake of the dynamic nature of Python: they are created at runtime, and can be modified further after creation.

FORMULAS OF A SPHERE-

Diameter of a Sphere $D = 2\ r$
Surface Area of a Sphere $A = 4\ \pi\ r2$
Volume of a Sphere $V = (4/3)\ \pi\ r3$

FUNCTION-

Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Some Benefits of Using Functions-
Increase Code Readability
Increase Code Reusability

Program Code-

```
class sphere():
  def __init__(self,radius):
        self.radius=radius

  def diameter(self):
    diameter=2*radius
    print("Diameter=",diameter)
```

```
  def circumference(self):
    circumference=6.2832*radius
    print("circumference=",circumference)

  def volume(self):
    volume=1.33*3.14*radius*radius*radius
    print("Volume=",volume)


radius=float(input("Enter the radius of the sphere--"))
obj=sphere(radius)
obj.diameter()
obj.circumference()
obj.volume()
```

OUTPUT-

```
Enter the radius of the sphere--5
Diameter= 10.0
circumference= 31.416
Volume= 522.0250000000001
```

Experiment No. 2

Aim: Program to find the union between two lists**.**

Theory:

List:

List are used to store multiple items in a single variable. List are one of 4 built-in data types in python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage list items can be of any data tuple. List are created using square brackers:

```
Thislist 1=[“apple”,”banana”,”cherry”]
Thislist 2=[“apple”,2,”cherry”]
Print(thislist 1)
Print(thislist 2)
```

The list is a most versatile datatype available in python which can be written as a list of comma separated values(items) between square brackets. Important things about a list is that items in a list need not be of the same type.

List items are ordered, changeable and allow duplicate values. List items are index , the first item has index [0] , the second item has index [1] etc. when we say that list are ordered , it means that the item have a defined order , and that order will not change. If you add new item to a list, the new item will be placed at the end of the list.

Length function :

To determine how many items a list has, use the len() function:

For example:
Print the number of items in the list:

This list =[“apple”, “banana”, “cherry”]
Print(len(thislist))

Output = 3

Updating List:

You can update single or multiple elements of lists by giving the slice on the left hand side of the assignment operator , and you can add to elements in a list with the append() method.
For example:

List =[‘physics’, ‘chemistry’ ,19997,2000]
Print(“value available at index 2”)
Print list [2]
List [2] = 2001;
Print(“new value available at index 2”)
Print list[2]

Union function:

The union of the two or more sets is the set of all distinct elements present in all the sets.

For Example:
A={1,2}
B={2,3,4}
C={5}

Then,
A U B=B U A = {1,2,3,4}
A U C=C U A ={1,2,5}
B U C=C U B ={2,3,4,5}

A U B U C ={1,2,3,4,5}

Program Code :

```
L1=[]
Num1=int(input("Enter size of list 1:"))
for n in range (Num1):
   a=int(input("Enter number:"))
L1.append(a)

L2=[]
Num2=int(input("Enter size of list2:"))
for n in range(Num2):
   b=int(input("Enter number:"))
L1.append(b)

union=list(set().union(L1,L2))
print("The union of two list is :",union)
```