ACAD-DI-86	Lab Manual	Academic Year: 2024-25
Rev : 01		Semester: II
Date: 02.04.2021		

Subject: Python Programming Lab (BTCOL406)

List of Experiments

Exp. No.	Name of Experiments
1.	Program to calculate area of triangle, rectangle, circle
2.	Program to find the union of two lists.
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: 01

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

Objectives: To demonstrate basic Python programming concepts

Theory: Theory: Python is a widely used general-purpose, high-level programming language. Python is a popular programming language. It was created by Guido van Rossum in 1991 and released.

Features in Python:

1. **Free and Open Source**: Since it is open-source, this means that the source code is also available to the public. So it is freely available for everyone. It has a large community across the world that is dedicatedly working towards making new Python modules and functions.
2. **Expressive Language**: Python can perform complex tasks using a few lines of code. A simple example: the hello world program. You simply type print("Hello World"). It will take only one line to execute, while Java or C takes multiple lines.
3. **Interpreted Language:** Python is an interpreted language; it means the Python program is executed one line at a time. The advantage of being an interpreted language is that it makes debugging easy and portable.
4. **Cross-platform Language**: Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language. It enables programmers to develop software for several competing platforms by writing a program only once.
5. **Object-oriented Language:** Python supports object-oriented language concepts of classes and objects. It supports inheritance, polymorphism, and encapsulation. The object-oriented approach helps programmers write reusable code and develop applications with less code.
6. **Large Standard Library:** It provides a vast range of libraries for various fields such as machine learning and web development, and also for scripting. There are various machine learning libraries such as TensorFlow, Pandas, NumPy, Keras, and PyTorch.
7. **GUI Programming Support:** Graphical User Interface (GUI) support for developing desktop applications. PyQt5, Tkinter, and Kivy are the libraries used for developing web applications.

8. **Integrated:** It can be easily integrated with languages like C, C++, and Java. Python runs code line by line like C or Java, making it easy to debug the code.

Advantages:

1. Presence of third-party modules.
2. Extensive support libraries (NumPy for numerical calculations, Pandas for data analytics, etc.)
3. User-friendly data structures.
4. High-level language.
5. Dynamically typed language (no need to mention data type on the value assigned).
6. Portable and interactive.
7. Ideal for prototypes - provide more functionality with less coding.
8. Versatile, easy to read, learn, and write.

Code:

```
import math

def square_area(side): # Area of Square

    return side ** 2

def rectangle_area(length, width): # Area of Rectangle

    return length * width

def circle_area(radius): # Area of Circle

    return math.pi * radius ** 2

def triangle_area1(base, height): # Area of Triangle 01

    return 0.5 * base * height

def triangle_area2(a,b,c): # Area of Triangle 02

    s = float((a + b + c) / 2)

    return (s * (s - a) * (s - b) * (s - c)) ** 0.5

while True:
    print("Select the shape for which you want to calculate the
area:")

    print("1. Square")

    print("2. Rectangle")

    print("3. Circle")

    print("4. Triangle (if base and height are given)")

    print("5. Triangle (if three sides of triangle are given)")

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

    if choice == 1:

          side = float(input("Enter the length of a side of the
square: "))

        print("Area of the square:", square_area(side))

    elif choice == 2:

         length = float(input("Enter the length of the rectangle:
"))

          width = float(input("Enter the width of the rectangle:
"))

          print("Area of the rectangle:", rectangle_area(length,
width))

    elif choice == 3:

          radius = float(input("Enter the radius of the circle:
"))

        print("Area of the circle:", circle_area(radius))

    elif choice == 4:

        base = float(input("Enter the base of the triangle: "))
```

```
         height = float(input("Enter the height of the triangle: "))

            print("Area of the triangle:", triangle_area1(base, height))

    elif choice == 5:

         a = float(input("Enter the first side of the triangle: "))

         b = float(input("Enter the second side of the triangle: "))

         c = float(input("Enter the third side of the triangle: "))

        print("Area of the triangle:", triangle_area2(a, b, c))

    else:

        print("Invalid choice!")

    cont = input("Do you want to continue? (yes/no): ")

    if cont.lower() != 'yes':

        break
```

Output:

```
Select the shape for which you want to calculate the area:

1. Square

2. Rectangle

3. Circle

4. Triangle (if base and height are given)

5. Triangle (if three sides of triangle are given)
```

```
Enter your choice (1/2/3/4/5):  1

Enter the length of a side of the square:  4

Area of the square: 16.0

Do you want to continue? (yes/no):  yes

Select the shape for which you want to calculate the area:

1. Square

2. Rectangle

3. Circle

4. Triangle (if base and height are given)

5. Triangle (if three sides of triangle are given)

Enter your choice (1/2/3/4/5):  2

Enter the length of the rectangle:  5

Enter the width of the rectangle:  6

Area of the rectangle: 30.0

Do you want to continue? (yes/no):  yes

Select the shape for which you want to calculate the area:

1. Square

2. Rectangle

3. Circle

4. Triangle (if base and height are given)

5. Triangle (if three sides of triangle are given)

Enter your choice (1/2/3/4/5):  7

Invalid choice!

Do you want to continue? (yes/no):  yes
```

```
Select the shape for which you want to calculate the area:

1. Square

2. Rectangle

3. Circle

4. Triangle (if base and height are given)

5. Triangle (if three sides of triangle are given)

Enter your choice (1/2/3/4/5):  3

Enter the radius of the circle:  6

Area of the circle: 113.09733552923255

Do you want to continue? (yes/no):  yes

Select the shape for which you want to calculate the area:

1. Square

2. Rectangle

3. Circle

4. Triangle (if base and height are given)

5. Triangle (if three sides of triangle are given)

Enter your choice (1/2/3/4/5):  4

Enter the base of the triangle:  45

Enter the height of the triangle:  85

Area of the triangle: 1912.5

Do you want to continue? (yes/no):  yes

Select the shape for which you want to calculate the area:

1. Square

2. Rectangle
```

```
3. Circle

4. Triangle (if base and height are given)

5. Triangle (if three sides of triangle are given)


Enter your choice (1/2/3/4/5):  5

Enter the first side of the triangle:  5

Enter the second side of the triangle:  8

Enter the third side of the triangle:  12

Area of the triangle: 14.523687548277813

Do you want to continue? (yes/no):  no
```

Conclusion:

Python is a versatile, powerful, and user-friendly programming language known for its readability, extensive library support, and portability across multiple platforms. It facilitates rapid development and integration with other languages, making it ideal for a wide range of applications from simple scripts to complex machine learning projects.

Experiment No: 02

Aim: Program to find the union of two lists.

Objectives:

1. Understand the Definition and Usage of Lists in Python
2. Create and Manipulate and Combine Lists

Theory :

Definition: Lists are used to store multiple items in a single variable. Lists are one of four built-in data types in Python used to store collections of data; the other three are Tuple, Set, and Dictionary, all of which have different qualities and uses. Lists are created using square brackets.

- **Example of List:** Create a list:

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

- **Output:**

```
["apple", "banana", "cherry"]
```

Append Function in List: To add an item to the end of the list, use the `append()` method.

- **Example:**

```
this_list = ["apple", "banana", "cherry"]

this_list.append("orange")

print(this_list)
```

- **Output:**

```
["apple", "banana", "cherry", "orange"]
```

Extend Function in List: To append elements from another list to the current list, use the `extend()` method.

- **Example**: Add the elements of `tropical` to the list:

```
this_list = ["apple", "banana", "cherry"] tropical =
["mango", "pineapple", "papaya"]

this_list.extend(tropical)

print(this_list)
```

- **Output:**

```
["apple", "banana", "cherry", "mango", "pineapple",
"papaya"]
```

Union of Two Lists: It means taking all the elements from list A and list B (there can be more than two lists) and putting them inside a single new list. There are various ways to combine the lists, and we can maintain the repetition or remove the repeated elements in the final list.

- **Example:**

```
list1 = [23, 15, 2, 14, 16, 20, 52]

list2 = [2, 48, 15, 12, 26, 32, 47, 54]

combined_list = list1 + list2

print(combined_list)
```

- **Output:**

```
[23, 15, 2, 14, 16, 20, 52, 2, 48, 15, 12, 26, 32, 47, 54]
```

Set Union Method: The `union()` method returns a set that contains all items from the original set and all items from the specified set(s).

Syntax:

```
set.union(set1, set2)
```

- **Example:**

```
x = {'a', 'b', 'c'}

y = {'f', 'd', 'a'}

z = {'c', 'd', 'e'}

result = x.union(y, z)
```

```
print(result)
```

- **Output:**

```
{'a', 'f', 'b', 'e', 'c', 'd'}
```

Conclusion:

In this practical lab, we have shown the union of two lists. To show the union of two lists, we used `list1` and `list2`. Among the lists, we combined `list1` and `list2` to write the union of the two lists.

Experiment No: 03

Aim: Program to find the intersection of two lists.

Objectives:

1. Understand the Concept of Sets
2. To use different methods of finding intersection between two lists

Theory:

A set can contain unique values and cannot maintain order. Repetition is removed from the sets.

Intersection of Sets: The set contains only items that exist in both sets, or in all sets if the comparison is done with more than two sets.

- The `intersection()` method returns a set that contains the similarity between two or more sets.

Syntax:

```
set.intersection(set1, set2, etc)
```

AND (&) Operator: We can also get intersections using the `&` operator.

```
set1 = {2, 4, 5, 6, 7}

set2 = {5, 4, 6, 17, 18, 4}

set3 = {1, 0, 12, 3}

print(set1 & set2)

print(set1 & set3)
```

Output:

```
{4, 5, 6}
```

Programs:

Program 1:

```
list1 = ["ncer", 1, 4, 2]

list2 = ["abc", 1, 5, 16, "ncer"]

new_list = []

for element in list1:

     if element in list2:

           new_list.append(element)

print(new_list)
```

Output:

```
["ncer", 1]
```

Explanation: In this Python program for the intersection of two lists, we use a for loop and `append()` function. Here, we created a new list to print the intersection of two sets using `append()`, which appends the elements that are common in list1 and list2.

Program 2:

```
list1 = [1, 2, 3, 4, 5, 6, 11, 27]

list2 = ["abc", 1, 2, 3, 4]

s = set(list1)

si = set(list2)

print(s)

print(si)

new_set = s & si

print(new_set)
```

Output:

```
{1, 2, 3, 4, 5, 6, 11, 27}

{1, 2, 3, 4, "abc"}

{1, 2, 3, 4}
```

Explanation: In this example, we have created a list of elements using square brackets enclosing the elements. One element is repeated in the list, implying that list elements are ordered, changeable, and allow duplicate values. In the output, all duplicate values are removed because of typecasting from list to set.

Program 3:

```
def intersection(list1, list2):

     temp = set(list2)

     list3 = [value for value in list1

     if value in temp]

          return list3  l

     List1 = [9, 9, 7, 4, 21, 45, 11, 63]

     list2 = [4, 9, 11, 26, 28, 25, 26, 66, 91]


print(intersection(list1, list2))
```

Output:

```
[9, 4, 11]
```

Explanation: The intersection of two lists means getting all the familiar elements in both initial lists. Python is known for its excellent built-in data structures. Python lists are one of the famous and valuable built-in data types in Python. They can store various data types in a sorted order.

Intersection of Two Sets Using Hybrid Approach:

- **Code :**

  ```
  set1 = {"apple", "banana", "cherry"} set2 = {"google",
  "microsoft", "apple"} new_set = set1.intersection(set2)
  print(new_set)
  ```

- **Output:**

  ```
  {"apple"}
  ```

- **Explanation:** In the above program, we perform the intersection operation on two sets (`set1` and `set2`). We created a new set using `set1.intersection(set2)` and assigned the common elements of `set1` and `set2` to the new set.

Conclusion:

In this experiment, we performed the intersection operation on two lists and two sets using different methods and functions like `append()`, `set()`, and `intersection()`. We also demonstrated converting lists to sets.

Experiment No: 04

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

Objectives:

1. Understand the Concept of Sets
2. To use different methods of finding intersection between two lists

Theory:

Definitions:

1. **List:**
   - A list is a collection of items in Python that is ordered and mutable. Lists can contain elements of different types and allow duplicate values. They are created using square brackets.
2. **Occurrence:**
   - Occurrence refers to the instance or frequency of an element appearing within a list. For example, in the list ['apple', 'banana', 'apple'], the word 'apple' has two occurrences.
3. **Element Removal:**
   - The process of removing specific elements from a list based on certain criteria. In this case, it involves removing a particular occurrence of a specified element.

Steps of Program Execution:

1. **Input the List:**
   The user is prompted to enter elements of the list separated by spaces. These elements are then converted into a list format for further processing.
2. **Specify Element to Remove:**
   The user is asked to input the element they wish to remove and specify which occurrence of that element should be removed from the list.
3. **Count Occurrences:**
   The program iterates through the list, counting the occurrences of the specified element. Each time the element is encountered, the count is incremented.
4. **Remove the Specified Occurrence:**
   When the count matches the specified occurrence, the element is added to a temporary list (list2) of items to be removed. If the count does not match, the element is added to another list (list3) which represents the modified list.

5. **Update and Display Results:**
   After processing, the program displays the list of removed items and the updated list after the deletion.

Code :

```
print("Enter elements of the list separated by spaces:")

list1_input = input().split()

list1 = []

# Append each item from input to list1

for item in list1_input:

    list1.append(item)

print("Original list:", list1)

# Input the element to be removed and the occurrence

item = input("Enter the element to be removed: ")

occurrence = int(input("Enter the occurrence to be removed: "))

count = 0

list2 = []  # List to store removed items

list3 = []  # List to store updated list

# Process the list to remove the specified occurrence

for i in list1:

    if i == item:

        count += 1

        if count == occurrence:

            list2.append(i)

        else:

            list3.append(i)
    else:

        list3.append(i)

# Output results

if count >= occurrence:

    print("Items removed are:", list2)

    print("List after deletion:", list3)

else:

    print("Element entered is not available!")
```

Output : -

```
Enter elements of the list separated by spaces:

pranav prathamesh stephen shantanu nikhil stephen

Original list: ['pranav', 'prathamesh', 'stephen', 'shantanu',
'nikhil', 'stephen']

Enter the element to be removed: stephen

Enter the occurrence to be removed: 1

Items removed are: ['stephen']

List after deletion: ['pranav', 'prathamesh', 'shantanu',
'nikhil', 'stephen']
```

Explanation: This program provides a way to manage lists by selectively removing occurrences of specified elements. It helps in understanding how to manipulate lists by removing specific instances of elements based on user input.

Conclusion

1. The program removes a specified occurrence of an element from a list.
2. It handles user input and processes the list to update it accordingly.
3. Results are clearly displayed, showing removed items and the updated list.
4. It checks for cases where the element or occurrence does not exist.

Experiment No: 05

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

Objectives:

1. Understand string manipulation and methods in Python.
2. Learn how to count occurrences of each word in a given string.

Theory:

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.

Example:

```
s = 'hi'

print(s[1])  # Output: i

print(len(s))  # Output: 2

print(s + ' there')  # Output: hi there
```

String Methods: Here are some of the most common string methods. A method is like a function, but it runs "on" an object. If the variable s is a string, then the code s.lower() runs the lower() method on that string object and returns the result (this idea of a method running on an object is one of the basic ideas that make up Object-Oriented Programming, OOP). Here are some of the most common string methods:

- s.lower(), s.upper() -- returns the lowercase or uppercase version of the string
- s.strip() -- returns a string with whitespace removed from the start and end
- s.isalpha(), s.isdigit(), s.isspace() -- tests if all the string chars are in the various character classes
- s.startswith('other'), `s.endswith('other') -- tests if the string starts or ends with the given other string
- s.find('other') -- searches for the given other string (not a regular expression) within s, and returns the first index where it begins or -1 if not found
- s.replace('old', 'new') -- returns a string where all occurrences of 'old' have been replaced by 'new'

- s.split('delim') -- returns a list of substrings separated by the given delimiter. The delimiter is not a regular expression, it's just text. 'aaa,bbb.ccc'.split(',') -> ['aaa', 'bbb', 'ccc']. As a convenient special case s.split() (with no arguments) splits on all whitespace chars.
- s.join(list) -- opposite of split(), joins the elements in the given list together using the string as the delimiter. e.g. '---'.join(['aaa', 'bbb', 'ccc']) -> aaa---bbb---ccc

Python Program to Count the Occurrences of Each Word in a Given String Sentence: The program takes a string and counts the occurrence of each word in the given sentence.

Problem Solution:

1. Take a string and a word from the user and store it in separate variables.
2. Initialize a count variable to 0.
3. Split the string using space as the reference and store the words in a list.
4. Use a for loop to traverse through the words in the list and use an if statement to check if the word in the list matches the word given by the user and increment the count.
5. Print the total count of the variable.
6. Exit.

Code :

```
# Step 1: Take a string and a word from the user and store it in
separate variables.

input_string = input("Enter a string: ")

word_to_count = input("Enter the word to count: ")

# Step 2: Initialize a count variable to 0.

count = 0

# Step 3: Split the string using space as the reference and
store the words in a list.

words_list = input_string.split()

# Step 4: Use a for loop to traverse through the words in the
list and use an if statement to check if the word in the list
matches the word given by the user and increment the count.

for word in words_list:
    if word == word_to_count:

        count += 1


# Step 5: Print the total count of the variable.

print("The word '{}' appears {} times in the given
string.".format(word_to_count, count))
```

Output :

```
Enter a string: this is a test. this test is simple. this is
only a test.

Enter the word to count: test

The word 'test' appears 3 times in the given string.
```

Conclusion:

1. The experiment demonstrates how to manipulate strings and use string methods in Python.
2. The program effectively counts and displays the occurrences of each word in a given string.

Experiment No: 06

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

Objectives:

1. Understand various methods to check for the presence of a substring in a string.
2. Learn how to use string methods like find(), index(), count(), and split().

Theory:

Python string contains another string or a substring in Python. Given two strings, check if a substring is there in the given string or not. Does Python have a string containing the substring method? Yes, checking a substring is one of the most used tasks in Python. Python uses many methods to check a string containing a substring like find(), index(), count(), etc. The most efficient and fast method is by using an "in" operator, which is used as a comparison operator. Here are different approaches to check if a substring is present in a given string:

1. Using the if...in operator:
2. Using the split() method:
3. Using the find() method:
4. Using the count() method:
5. Using the index() method:
6. Using the __contains__ magic method:
7. Using regular expressions:

Some of the Methods are shown below :

Method 1: Check substring using the if...in operator:

Code :

```
# Take input from users

MyString1 = "A geek in need is a geek indeed"

if "need" in MyString1:

    print("Yes! it is present in the string")

else:
    print("No! it is not present")
```

Output :

```
Yes! it is present in the string
```

Method 2: Check substring using the find() method:

We can iteratively check for every word, but Python provides us an inbuilt function find() which checks if a substring is present in the string, which is done in one line. find() function returns -1 if it is not found, else it returns the first occurrence, so using this function this problem can be solved.

Code :

```
# Function to check if small string is there in big string

def check(string, sub_str):

    if (string.find(sub_str) == -1):

        print("NO")

    else:

        print("YES")

# Driver code

string = "ncer talegaon pune"

sub_str = "pune"

check(string, sub_str)
```

Output :

```
YES
```

Method 3: Check substring using the count() method:

You can also count the number of occurrences of a specific substring in a string, then you can use the Python count() method. If the substring is not found, then "NO" will be printed; otherwise, "YES" will be printed.

Code :

```
def check(s2, s1):

    if (s2.count(s1) > 0):

        print("YES")

    else:

        print("NO")

s2 = "Allow the User to input the Width and Height"

s1 = "new"

check(s2, s1)
```

Output :

```
NO
```

Method 4: Check substring using the index() method:

The .index() method returns the starting index of the substring passed as a parameter. Here "substring" is present at index 16.

Code :

```
any_string = "Allow the User to input the Width and Height"

start = 0

end = 1000

print(any_string.index('substring', start, end))
```

Output:

```
ValueError: substring not found
```

Method 6: Using slicing:

This implementation uses a loop to iterate through every possible starting index of the substring in the string, and then uses slicing to compare the current substring to the substring argument. If the current substring matches the substring argument, then the function returns True. If the substring is not found after checking all possible starting indices, then the function returns False.

Code :

```
def is_substring(string, substring):

    for i in range(len(string) - len(substring) + 1):

        if string[i:i+len(substring)] == substring:

            return True

    return False

string = "Allow the User to input the Width and Height"

substring = " User "

print(is_substring(string, substring))
```

Output:

```
True
```

Conclusion :

Multiple methods exist to check for a substring in a given string in Python:

1. if...in statement: Simple and efficient.
2. find() method: Returns the first occurrence index or -1 if not found.
3. count() method: Counts occurrences, returns "YES" if >0.
4. index() method: Returns the starting index of the substring.
5. Slicing and loop: Iterates through possible starting indices and compares slices to the substring.

Experiment No: 07

Aim: Program to map two lists into a dictionary.

Objectives:

1. Understand various methods to check for the presence of a substring in a string.
2. Learn how to use string methods like find(), index(), count(), and split().

Theory:

A list is an arranged collection of elements used to store collections of data. It can contain various types of data objects, separated by commas and enclosed within square brackets. Lists are mutable, meaning we can change the order of elements and replace individual elements even after the list has been created. This mutability makes lists widely used, despite their higher memory consumption in small projects.

A dictionary is an ordered collection of data values, accessible by key. It is a mapping of keys and values. The dict() constructor is used to create instances of the class dict. The order of elements in a dictionary is undefined. We can iterate over keys, values, or key-value pairs.

Mapping Two Lists into a Dictionary

There are different ways to map two lists into a dictionary. One common method is using the zip() function.

Using the zip() Function

The zip() function in Python creates an iterator that aggregates elements from at least two lists. To map two lists together, we can use the zip() function, which allows us to combine two lists. We can use one list as the keys for the dictionary and the other as the values.

Example:

In the given example, we have two lists: one containing a list of students and the other containing their marks. The requirement is to create a single dictionary that stores the name of a student along with their marks. We can use the following solution to accomplish this task:

Code :

```
students = ['Smith', 'John', 'Priska', 'Abhi']

marks = [89, 53, 92, 86]
```

```
students_dict = dict(zip(students, marks))

print(students_dict)
```

Output :

```
{'Smith': 89, 'John': 53, 'Priska': 92, 'Abhi': 86}
```

Conclusion :

1. Lists are mutable, ordered collections of elements in Python, allowing for versatile data storage and modification, though they consume more memory.
2. Dictionaries are unordered collections of unique key-value pairs, and the zip() function can be used to map two lists into a dictionary, with the shorter list determining the length and the last value being used if keys are repeated.

Experiment No: 08

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

Objectives:

1. To understand the use of dictionaries in Python.
2. To learn how to split a string into individual words.

Theory: In Python, a dictionary is a mutable, unordered collection of data values, used to store data values such as a map. Unlike other data types that hold only a single value as an element, a dictionary holds key

pairs. Key-value is provided in the dictionary to make it more optimized. The keys in a dictionary are unique and can be any immutable data type, such as strings, numbers, or tuples.

The objective of this program is to count the frequency of each word in a given string and store the result in a dictionary. This allows us to efficiently keep track of how many times each word appears in the string. The approach is to split the string into individual words, iterate through these words, and use a dictionary to maintain a count of each word's occurrences.

1. **Dictionary**: A collection of key-value pairs where keys are unique and used to access values.
2. **split():** A method that splits a string into a list where each word is a list item.
3. **in:** A keyword used to check if a key exists in a dictionary.

Code :

```
# Prompt the user to enter a sentence

sent1 = input("Enter the sentence: ")

# Split the sentence into words

list1 = sent1.split()

# Initialize an empty dictionary

dict1 = dict()

# Print the list of words

print(list1)
```

```
# Iterate through the list of words

for i in list1:

    # If the word is already in the dictionary, increment its
count

    if i in dict1:

        dict1[i] = int(dict1[i]) + 1

    # If the word is not in the dictionary, add it with a count
of one

    else:

        dict1[i] = 1

# Print the dictionary containing word counts

print(dict1)
```

Output :

```
Enter the sentence: the quick brown fox jumps over the lazy dog
the quick brown fox

['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy',
'dog', 'the', 'quick', 'brown', 'fox']

{'the': 3, 'quick': 2, 'brown': 2, 'fox': 2, 'jumps': 1, 'over':
1, 'lazy': 1, 'dog': 1}
```

Explanation of the Program:

1. Input the Sentence: The user is prompted to enter a sentence.
2. Splitting the Sentence: The sentence is split into individual words using the split() method.
3. Initialize the Dictionary: An empty dictionary is created to store the word counts.
4. Count Word Frequency: Iterate through each word in the list. If the word is already in the dictionary, increment its count. Otherwise, add the word to the dictionary with a count of one.
5. Print the Dictionary: Finally, print the dictionary containing the words and their respective counts.

Conclusion:

1. Word Frequency Count: The program successfully counts the frequency of each word in a given string and stores the results in a dictionary.
2. Efficient Data Management: Using a dictionary allows for efficient storage and retrieval of word counts, demonstrating the utility of dictionaries in handling frequency counts and other similar tasks.

Experiment No: 09

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

Objectives:

1. To understand the usage of dictionaries in Python.
2. To learn how to split a string into individual words.

Theory:

In Python, a dictionary is a powerful data structure used to store data in key-value pairs. This allows for efficient data retrieval and management. In this experiment, we aim to create a dictionary where each key is the first character of words in a given sentence, and the corresponding value is a list of words starting with that character.

The objective of this program is to practice manipulating strings and dictionaries. By doing this, we can efficiently group words by their starting character, which can be useful for various text processing tasks.

1. **Dictionary:** A mutable, unordered collection of key-value pairs. Keys must be unique.
2. **split():** A method that splits a string into a list of words based on whitespace.
3. **append()**: A method used to add an element to the end of a list.

Code :

```
# Initialize an empty dictionary

dict = {}

# Predefined sentence

a = "The cat sat on the mat and then the cat jumped off the mat"

# Print the sentence

print(a)

# Split the sentence into words

list1 = a.split()

# Print the list of words
```

```
print(list1)

# Iterate through the list of words

for i in list1:

    # If the first character of the word is not a key in the
dictionary

    if i[0] not in dict.keys():

        # Add the first character as a key and initialize with a
list containing the word

        dict[i[0]] = []

        dict[i[0]].append(i)

    else:

        # If the first character is already a key, append the
word to the list

        if i[0] not in dict[i[0]]:

            dict[i[0]].append(i)

# Print the list of words

print(list1)

# Print the dictionary

print(dict)
```

Output:

```
The cat sat on the mat and then the cat jumped off the mat

['The', 'cat', 'sat', 'on', 'the', 'mat', 'and', 'then', 'the',
'cat', 'jumped', 'off', 'the', 'mat']

['The', 'cat', 'sat', 'on', 'the', 'mat', 'and', 'then', 'the',
'cat', 'jumped', 'off', 'the', 'mat']
```

```
{'T': ['The'], 'c': ['cat'], 's': ['sat'], 'o': ['on', 'off'],
't': ['the', 'then', 'the', 'the'], 'm': ['mat'], 'a': ['and'],
'j': ['jumped']}
```

Explanation of the Program:

1. Initialize the Dictionary: An empty dictionary is created to store the results.
2. Input Sentence: A predefined sentence is used as the input string.
3. Split Sentence into Words: The sentence is split into individual words using the split() method.
4. Iterate through Words: Each word in the list is examined.
5. Update Dictionary: For each word, check if the first character is already a key in the dictionary:
   - o If not, add the first character as a key and initialize the value with a list containing the word.
   - o If the first character is already a key, append the word to the list associated with that key.
6. Print Results: The program prints the list of words and the resulting dictionary.

Conclusion :

1. **Grouping Words**: The program successfully groups words by their starting character using a dictionary.
2. **Efficient Data Management:** This approach demonstrates the efficient use of dictionaries for text processing tasks, enabling quick retrieval and organization of data based on specific criteria.

Experiment No: 10

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

Objectives:

1. To understand the concept of recursion.
2. To learn how to use recursion to solve problems involving repetitive tasks.
3. To implement a recursive function to find the length of a list.

Theory:

Recursion is a method where the solution to a problem depends on solutions to smaller instances of the same problem. In Python, recursion involves a function calling itself. This technique can be used to perform repetitive tasks in a compact manner. In this experiment, we aim to find the length of a list using recursion. This is a valuable exercise to understand the concept of recursion and how it can be applied to solve problems iteratively.

Key Terms:

1. **Recursion:** A method of solving a problem where the function calls itself as a subroutine.
2. **Base Case**: The condition under which the recursive function stops calling itself.
3. **Global Variable**: A variable declared outside of a function which can be accessed globally.

Code :

```
# Method 1: Using a global variable

len = 0

def length1(a):

    global len

    if a:

        len = len + 1

        length1(a[1:])

    return len
```

```
my_list1 = [1, 2, 3, 4, 5, 6]

len1 = length1(my_list1)

print("The length of the list is:", len1)


# Method 2: Without using a global variable

def length2(a):

    if not a:

        return 0

    else:

        return 1 + length2(a[1:])

my_list2 = [7, 8, 9, 10]

len2 = length2(my_list2)

print("The length of the list is:", len2)
```

Output :

```
#Method 01

The length of the list is: 6

#Method 02

The length of the list is: 4
```

Explanation of the Program:

Method 1: This method uses a global variable to keep track of the length of the list.

- **Base Case:** If the list is empty, the function stops calling itself.
- **Recursive Case:** If the list is not empty, increment the length by 1 and call the function with the rest of the list (excluding the first element).

Method 2: This method calculates the length without using a global variable.

- **Base Case**: If the list is empty, return 0.

- **Recursive Case:** If the list is not empty, return 1 plus the length of the rest of the list (excluding the first element).

Conclusion :

1. **Recursion Application:** The program demonstrates how recursion can be applied to find the length of a list.
2. **Understanding Base and Recursive Cases**: By implementing these methods, we understand the importance of defining base and recursive cases in recursive functions.

Experiment No: 11

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

Objectives:

1. To understand the basics of object-oriented programming in Python.
2. To learn how to define and use classes and objects in Python.
3. To implement methods for calculating the diameter, circumference, and volume of a sphere.

Theory:

Object-oriented programming (OOP) is a paradigm that uses "objects" to design applications and computer programs. Python, being an OOP language, allows us to create classes and objects. In this experiment, we create a class Sphere to calculate the diameter, circumference, and volume of a sphere given its radius.

1. **Class:** A blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods).
2. **Object:** An instance of a class. When a class is defined, no memory is allocated until an object of that class is instantiated.
3. **Method:** A function that is associated with an object. In Python, methods are defined within a class.
4. **init method:** A special method in Python classes, also known as the constructor, which initializes an object's attributes.
5. **Self**: Represents the instance of the class. It is used to access variables that belong to the class.

Code :

```
class Sphere:

    def __init__(self):

        radius = float(input("Enter the Radius of the Sphere: 
"))

        self.radius = radius
    def diameter(self):
        dia = self.radius * 2
        print(f"Diameter of the sphere is {dia} meters")


    def circumference(self):
        circum = 2 * 3.141592653589793 * self.radius
        print(f"Circumference of the sphere is {round(circum,
3)} meters")


    def volume(self):
        vol = (4/3) * 3.141592653589793 * (self.radius**3)
        print(f"Volume of the sphere is {round(vol, 3)} cubic
meters")
s1 = Sphere()
s1.diameter()
s1.circumference()
s1.volume()
```

Output:

```
Enter the Radius of the Sphere: 5
Diameter of the sphere is 10.0 meters
Circumference of the sphere is 31.416 meters
Volume of the sphere is 523.598 cubic meters
```

Conclusion:

1. **Object-Oriented Programming Application**: The program demonstrates the use of classes and objects to encapsulate data and methods for computing geometric properties of a sphere.
2. **Mathematical Calculations in OOP**: By implementing methods within the Sphere class, we efficiently calculate the diameter, circumference, and volume of a sphere, showcasing the practical application of OOP in mathematical computations.

Experiment No: 12

Aim: Program to read a file and capitalize the first letter of every word in the file.

Objectives:

1. To read and display the contents of a file.
2. To manipulate the content by capitalizing the first letter of each word.

Theory:

In this experiment, we aim to create a Python program that reads the content of a file and capitalizes the first letter of every word. This task involves basic file handling operations and string manipulation techniques in Python.

Explanation of Terms:
- **File Handling:** Refers to the operations performed on files such as reading from and writing to them.
- **String Manipulation:** Involves modifying or analyzing strings of text. For this task, we will manipulate strings to capitalize the first letter of each word.
- **Exception Handling:** Allows a program to handle errors gracefully without crashing. The `try-except` block is used to manage situations where the file may not be found.

File Handling in Python

File handling is the process of managing files in a program, including tasks such as opening, reading, writing, and closing files. In Python, file handling is straightforward and involves several built-in functions and methods. Proper file handling ensures that resources are managed efficiently and data is processed correctly.

Different Ways of File Handling in Python

Python provides multiple ways to handle files, which are generally categorized into the following operations:

1. **Opening Files**

- Use the `open()` function to open a file. This function returns a file object, which provides methods and attributes to interact with the file.

```
file = open("example.txt", "r")  # Open file in read mode
```

Modes for Opening Files:

- `"r"`: Read mode (default). Opens the file for reading. The file must exist.

- `"w"`: Write mode. Opens the file for writing. Creates a new file or truncates an existing file.

- `"a"`: Append mode. Opens the file for writing at the end of the file. Creates a new file if it does not exist.

- `"b"`: Binary mode. Used to read or write binary files. Combined with other modes (e.g., `"rb"` or `"wb"`).

- `"x"`: Exclusive creation. Creates a new file but fails if the file already exists.

2. **Reading Files**

- Use methods like `read()`, `readline()`, or `readlines()` to read the contents of a file.

```
content = file.read()  # Read the entire file
  line = file.readline()  # Read a single line
  lines = file.readlines()  # Read all lines into a list
```

3. **Writing to Files**

- Use `write()` or `writelines()` methods to write data to a file.

```
file = open("example.txt", "w")  # Open file in write mode
  file.write("Hello, World!")  # Write a string to the file
  file.writelines(["Line 1\n", "Line 2\n"])  # Write multiple
lines
```

4. **Closing Files**

- Always close a file using the `close()` method to free up system resources.

```
file.close()
```

---

5. Using the `with` Statement

- The `with` statement simplifies file handling by automatically closing the file after the block is executed, even if an error occurs.

```
with open("example.txt", "r") as file:

    content = file.read()

    print(content)
```

6. File Object Methods

- `seek(offset, whence)`: Moves the file pointer to a specific position.

- `tell()`: Returns the current file pointer position.

- `flush()`: Flushes the internal buffer to the file.

```
with open("example.txt", "r+") as file:

    file.seek(0)  # Move to the start of the file

    print(file.tell())  # Get the current position (0)

    file.write("New content")

    file.flush()  # Ensure data is written to the file
```

7. Reading Binary Files

- Use binary modes (`"rb"` for reading and `"wb"` for writing) to handle non-text files such as images or executables.

```
with open("example.jpg", "rb") as file:

    data = file.read()
```

Code :

```
try:
    # Open the file in read mode
    with open("College.txt", "r") as file:
        # Read the entire content of the file
        content = file.read()
    # Capitalize the first letter of every word
    capitalized_content = content.title()
    # Print the original content
    print("Original Content:")
    print(content)
    # Print the capitalized content
    print("\nCapitalized Content:")
    print(capitalized_content)
except FileNotFoundError:
    # Print an error message if the file is not found
    print("Error: File 'College.txt' not found.")
```

Output:

```
Assuming `College.txt` contains the following text:
this is a sample file for testing.
The output of the program will be:
#Original Content:
this is a sample file for testing.
#Capitalized Content:
This Is A Sample File For Testing.
```

Conclusion:

1. The program successfully reads the contents of a specified file and capitalizes the first letter of every word using Python's string manipulation capabilities.
2. This experiment demonstrates the fundamental concepts of file handling, string manipulation, and exception handling in Python.