JSPM's

RAJARSHI SHAHU COLLEGE OF ENGINEERING

TATHAWADE, PUNE-33

(An Autonomous Institute Affiliated to Savitribai Phule Pune University,Pune)

LAB MANUAL

OF

PROGRAMMING LAB - I

(IT2214)

S.Y.B.Tech (2024-25)

INFORMATION TECHNOLOGY

Authors: 1) Dr. Pallavi Tekade

2) Prof. Ashika Hirulkar

Creation Date: - June 2024

Version: - 1

INDEX

1. How to Use This Manual

This manual assumes that the facilitators are aware of collaborative learning methodologies.

This manual will provide a tool to facilitate the session on Digital Communication modules in collaborative learning environment.

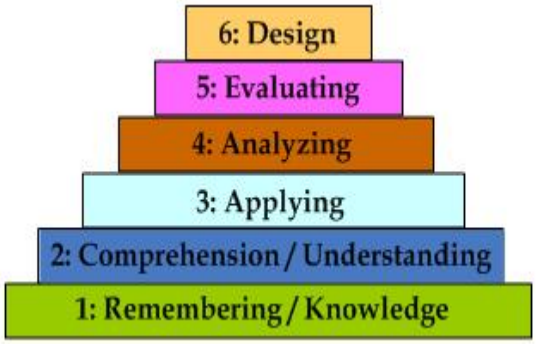
The facilitator is expected to refer this manual before the session.

Icon of Graduate Attributes

K Applying Knowledge	**A** Problem Analysis	**D** Design & Development	**I** Investigation of problems
M Modern Tool Usage	**E** Engineer & Society	**E** Environment Sustainability	**T** Ethics
T Individual & Team work	**O** Communication	**M** Project Management & Finance	**I** Life Long Learning

Disk Approach- Digital Blooms Taxonomy

1: Remembering / Knowledge
2: Comprehension / Understanding
3: Applying
4: Analyzing
5: Evaluating
6: Creating / Design



This Manual uses icons as visual cues to the interactivities during the session.

	Blooms Taxonomy
	Remembering
	Understanding
	Applying
	Analyzing
	Evaluating
	Creating

☞	This icon is used to indicate instructions for faculties.
	This icon is used to indicate a statement to be made by faculty.
	This icon is used to indicate a list of additional resources.
	This icon indicates an activity to be conducted.
?	This icon indicates questions to be asked by faculty.

VISION of Institute

To satisfy the aspirations of youth force, who want to lead the nation towards prosperity through techno economic development.

MISSION of Institute

To provide, nurture and maintain an environment of high academic excellence, research and entrepreneurship for all aspiring students, which will prepare them to face global challenges maintaining high ethical and moral standards.

VISION of Department

To create quality information technology professionals through superior academic environment.

MISSION of Department

To incorporate the IT fundamentals in students to be successful in their career. To motivate the students for higher studies, research and entrepreneurship. To provide IT services to society.

PSO's of Department

PSO1: Demonstrate the ability to apply discrete principles of mathematics along with programming paradigms to expedite solution building in the IT domain.

PSO2: Apply computational techniques using core aspects of network and system programming to deliver secured applications in the arena of analytics and computing.

PSO3: Demonstrate project management skills to handle multidisciplinary complex tasks proficiently and utilize these skills for entrepreneurship.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

PO 1: Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO 2: Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO 3: Design/Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO 4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO 5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO 6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO 7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO 8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO 11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO 12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Prerequisites: Programming Fundamentals, Problem solving skills.

Laboratory Outcomes

LO1: Demonstrate the use of built-in data structures and sequences in data slicing

LO2: Expertise in high-order functions and functional programming

LO3: Perform data analysis, manipulation, and visualization.

LO4: Analyze the computational complexity of Python data structures.

2.1 Laboratory Objectives

Upon completion of Python Programming Lab students should able to:

1. To understand different types of python in-built data structures, sequences, high-order functions such as lambda, map, its applications, and complexity analysis.

2. To learn different object-oriented features of Python Programming.

2. Lab Details

2.1 Laboratory Equipment list

Hardware requirement:

Hardware	Specification
Processor	Intel Pentium 4 or Higher
Hard Drive	64 GB
Memory	4 GB Or Higher
Internet Browser	Microsoft Internet Explorer 7.0 or Higher, Google Chrome or Mozilla Firefox.
Software	Python 3.7 and above

2.2 Laboratory Plan

Exp. No.	Title of Experiment
1.	Study and implement a Basic Input /Output Programs.
2.	Study and implement Python programs using various types of Loops
3.	Study and implement Python programs using Functions.
4.	Study and implement Python programs using String.
5.	Write a Python program to create user-defined lists and execute its operations.
6.	Write a Python program to create user defined tuple and execute its operations.
7.	Write a Python program that reads the Particulars.txt file containing the elements in the string format and perform given operations on the myWallet dictionary
8.	Write a Python program to construct Python built-in data structure Set.
9.	Study and implement Python programs on Object Oriented Programming
10.	Mini Project
End semester assessment **Mock Practical Examination**	

EXPERIMENT NO.1

Study and implement a Basic Input /Output Programs.

Practical Session Plan

Time (min)	Content	Learning Aid / Methodology	Faculty Approach	Typical Student Activity	Skill / Competency Developed
20	Explanation of Experiment	Chalk & Talk , Presentation	Introduces, Facilitates, Explains	Listens	Knowledge, Communication, intrapersonal, Application
10	Preparation of Algorithm and Logic Development for Program	Paper Work	Introduces, Facilitates, Explains	Discuss,	Knowledge, Communication, intrapersonal, Application
60	Coding of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
20	Testing of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
10	Results and conclusions	Keywords	Lists, Facilitates	Listens, Participates, Discusses	Knowledge, Communication, intrapersonal, Comprehension

Title: Study and implement a Basic Input /Output Programs.

Write a **Python** program:

1: Accept numbers from a user
2: Display three string “Name”, “Is”, “James” as “Name**Is**James”
3: Convert Decimal number to octal using print() output formatting
4: Display float number with 2 decimal places using print()
5: Demonstrate basic data types
6: Implicit Type Conversion
7: Create a variable inside a function, with the same name as the global variable
8: Swap Two Numbers
9: To display the current date and time
10: Calculate the Area of a circle and triangle.

Objective:- To Study python in-built input/output operations

Aim/ Problem Statement: Write a Python Program to understand Basic Input/Output programs.

Theory:

Python - Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Identifiers:

1) A Python identifier cannot starts with a digit/integer.

2) An identifier can be a combination of a letters **(A to Z, a to z)** or an **underscore (_)** followed by zero or more letters, underscores and digits **(0 to 9)**

3) Keywords cannot be used as identifiers.

4) Python does not allow/use special symbols/ characters like **(~, !, @, $, %, ^, &, *)** within identifier.

5) Python is a case sensitive language; eg: num1 and Num1 are two different identifier

6) An identifier can be of any length.

7) We can start a variable name by **underscore (_).**

Comments in Python

A # sign is used before the text which we want to make comment line

Example: # print(“Hi Students!”)

What is Variable?

Let's see a statement age=10;

What is age in this statement? age is the name of the variable or we can say the identifier. Then what is variable? Variable is nothing but reserved memory locations associated with the variable name to store value and this value can change. As in this case value of age will change every year.

What is a constant ?

Consider another example: pi=3.14 In this case our identifier which is pi is storing a standard constant value which will never change. So we call it a constant.

Lets take another example where we take an identifier to store a name of a person: name="Advik"

In this case our identifier name is constant as value kept in it will never change for a person.

Data Types of Python:

In programming, data type is an important concept.

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:

Numeric Types: int, float, complex

Text Type: str

Sequencial Types: list, tuple, range

Mapping Type: dict

Set Types: set, frozenset

Boolean Type: bool

Binary Types: bytes, bytearray, memoryview

Reserved words:

and, exec, not, assert, finally, or, break, for, pass, class, from, print, continue, global, raise, def, if, return, del, import, try, elif, in, while, else, is, with, except, lambda, yield.

Questions:

1) What is Python, and what are its key features that make it popular among developers?

2) What is the purpose of using the input() function in Python, and how does it handle user input in different data types?

3) What are variables in Python, and how do you assign a value to a variable?

4) What is the significance of indentation in Python? How does it affect the structure of the code?

5) What are Python's basic data types, and how are they used in programming?

Conclusion-

In studying and implementing basic Input/Output programs in Python, we learned how to take user input with the input() function and display results using print(). We also explored formatting and managing data types, which are key skills for creating interactive Python programs.

EXPERIMENT NO.2

Study and implement Conditional Statements and Loop Programs

Practical Session Plan

Time (min)	Content	Learning Aid / Methodology	Faculty Approach	Typical Student Activity	Skill / Competency Developed
20	Explanation of Experiment	Chalk & Talk , Presentation	Introduces, Facilitates, Explains	Listens	Knowledge, Communication, intrapersonal, Application
10	Preparation of Algorithm and Logic Development for Program	Paper Work	Introduces, Facilitates, Explains	Discuss,	Knowledge, Communication, intrapersonal, Application

60	Coding of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
20	Testing of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
10	Results and conclusions	Keywords	Lists, Facilitates	Listens, Participates, Discusses	Knowledge, Communication, intrapersonal, Comprehension

Title: Study and implement Conditional Statements and Loop Programs

Write a **Python** program to:

1:Demonstrate Python operators
2: Find maximum of two numbers
3: Check if a number is even or odd
4: Count the total number of digits in a number
5: Determine the grade of a student based on their score.
6: Display numbers from -10 to -1 using for loop
7: Use the else block to display a message "Done" after successful execution of for loop
8: Write a program to display all prime numbers within a range
9: Display the Fibonacci series up to 10 terms
10: Find the factorial of a given number
11: Reverse a given integer number
12: Use a loop to display elements from a given list present at odd index positions
13: Calculate the cube of all numbers from 1 to a given number
14: Find the sum of the series upto n terms

Objective:- To Study conditional statements & looping in python.

Aim/ Problem Statement: Write a Python Program on conditional statements and loops

Theory:

In Python, **conditional statements** and **loops** are key elements of control flow in programming. **Conditional statements** allow a program to make decisions and execute certain sections of code

based on whether a condition is true or false. The most basic form of a conditional statement is the `if` statement, which checks a condition and runs a block of code if the condition is met. Additionally, `elif` (else if) and `else` statements can be used to check multiple conditions or to define an alternative action if no conditions are met.

Loops enable repetitive execution of a block of code as long as a specified condition is true or for a predetermined number of iterations. **For loops** are used to iterate over sequences like lists or strings, executing a block of code for each element in the sequence. **While loops** continue to execute as long as a certain condition remains true. Loops are powerful tools for automating repetitive tasks, reducing the need for manual coding, and making programs more efficient. Both conditional statements and loops are essential for writing flexible, responsive, and efficient code.

Operators in Python

1. Arithmetic operators

2. Assignment operators

3. Comparison operators

4. Logical operators

5. Identity operators

6. Membership operators

7. Bitwise operators

1) Arithmetic Operators:

i. Addition(+) x + y

ii. Subtraction(-) x - y

iii. Multiplication(*) x * y

iv. Division(/) x / y

v. Modulus(%) x % y

vi. Exponentiation(**) x ** y

vii. Floor division(//) x // y

2) Assignment Operators:

i. = x=1

ii. += (x+=5) is equivalent to x=x+5

iii. -= (x-=5) is equivalent to x=x-5

iv. *= (x*=5) is equivalent to x=x*5

v. /= (x/=5) is equivalent to x=x/5

vi. %= (x%=5) is equivalent to x=x%5

vii. **= (x**=5) is equivalent to x=x**5 (exponent operator)

viii. //= (x//=5) is equivalent to x=x//5 (floor division)

3) Comparison Operators:

i. Equal = = (x = = y)

ii. Not equal != (x != y)

iii. Greater than > (x > y)

iv. Less than < (x < y)

v. Greater than or equal to >= (x >= y)

vi. Less than or equal to <= (x <= y)

4) Logical Operators:

i. and

ii. or

iii. not

5) Identity Operators:

i. is

ii. is not

6) Membership operators:

i. in

ii. not in

7) Bitwise operators:

i. & (binary and)

ii. | (binary or)

iii. ^ (binary xor)

iv. ~ (negation)

v. << (left shift)

vi >> (right shift)

& (binary and)

• a = 1010 (Binary of 10)

• b = 0100 (Binary of 4)

• a & b =

1010

&

0100

= 0000 = 0 (Decimal)

| (binary or)

• a = 1010 (Binary of 10)

• b = 0100 (Binary of 4)

• a | b =

1010

|

0100

= 1110 = 14 (Decimal)

^ (binary xor)

• a = 1010 (Binary of 10)

• b = 0100 (Binary of 4)

• a ^ b =

1010

^

0100

= 1110 = 14 (Decimal)

~ (negation)

a= 1010 (Binary of 10)

~a = ~1010

= -(1010 + 1)

= -(1011)

= -11 (Decimal)

<< (left shift)

a= 0000 0101 (Binary of 5)

a << 1 = 0000 1010

= 10 (decimal)

a << 2

= 0001 0100

= 20 (decimal)

>> (right shift)

• a=1010 (Binary of 10)

• a>>2 1010>>2

1	0	1	0

		1	0

10 (Binary) = 2 in Decimal

Python For Loops :

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Loop Control Statements:

```
for x in "Students":
  print(x)
```

The break Statement:

With the break statement we can stop the loop before it has looped through all the items:

Ex-

```
for i in range(5):
    if i==3:
        break
    print(i)
```

```
for i in range(5):
    print(i)
    if i==3:
        break
```

The continue Statement:

With the continue statement we can stop the current iteration of the loop, and continue with the next:

Ex-

```
for i in range(5):
    if i==3:
        continue
    print(i)


for i in range(5):
    print(i)
    if i==2:
        continue
```

The range() Function:

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Ex-

```
for i in range(3):
    print(i)
```

Else in For Loop:

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

```
for i in range(3):
  print(i)
else:
  print("Bye!")
```

If statement:

An "if statement" is written by using the if keyword.

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Elif statement:

The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

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

Questions-

1) What is a conditional statement in Python, and how does it control the flow of a program?
2) Explain the difference between if, elif, and else statements in Python and when to use each.
3) What are loops in Python, and how do for and while loops differ in terms of usage and functionality?
4) Describe how the break and continue statements alter the normal flow of loops in Python.
5) Explain the significance of the loop control variable in a for loop and how it influences the loop's execution.

Conclusion:

In studying and implementing Conditional Statements and Loop Programs, we learned how to make decisions and repeat tasks in Python. Conditional statements let us control what the program does based on conditions, while loops help us perform actions multiple times. These are essential tools for writing effective and dynamic code.

EXPERIMENT NO.3

Study and implement Functions in Python.

Practical Session Plan

Time (min)	Content	Learning Aid / Methodology	Faculty Approach	Typical Student Activity	Skill / Competency Developed
20	Explanation of Experiment	Chalk & Talk , Presentation	Introduces, Facilitates, Explains	Listens	Knowledge, Communication, intrapersonal, Application
10	Preparation of Algorithm and Logic Development for Program	Paper Work	Introduces, Facilitates, Explains	Discuss,	Knowledge, Communication, intrapersonal, Application
60	Coding of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
20	Testing of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
10	Results and conclusions	Keywords	Lists, Facilitates	Listens, Participates, Discusses	Knowledge, Communication, intrapersonal, Comprehension

Title: Study and implement Functions in Python

Write a **Python** program to:

1: Create a function in Python
2: Create a function with variable length of arguments
3: Return multiple values from a function
4: Create a function with a default argument
5: Create an inner function to calculate the addition in the following way
6: Create a recursive function
7: Assign a different name to function and call it through the new name

Objective:- To Study python functions

Aim/ Problem Statement: Write a Python Program to understand functions.

Theory:

In Python, **functions** are reusable blocks of code that perform a specific task. Functions help organize code, making it more modular, readable, and easier to debug. You define a function using the `def` keyword followed by the function name and parentheses. Inside the parentheses, you can specify parameters, which are variables that the function can accept as input.

Once a function is defined, you can "call" it anywhere in your program by using its name followed by parentheses. If the function was defined with parameters, you can pass arguments to it when you call it. Functions can also return a value using the `return` statement, which allows the function to send back a result to the place where it was called.

Functions are crucial for reducing code repetition and implementing complex logic in a clean and organized way. By breaking down tasks into smaller, manageable functions, you can build more scalable and maintainable programs.

Structure of a Python Function:

```
def function_name(parameters):

    # Function body (code that runs when the function is called)

    return result  # Optional: to return a value
```

Key Concepts:

Function Definition:

Functions are created using the def keyword, followed by the function's name and a colon :

```
def function():

    print("Hello, World!")
```

Calling a Function:

To execute a function, you simply call its name followed by parentheses.

```
function()  # Output: Hello, World!
```

Parameters and Arguments:

Functions can take input values called parameters. These inputs are passed when calling the function.

```
def function(name):
    print("Hello, " + name)
function("Advik")  # Output: Hello, Advik
```

Return Statement:

Functions can return values using the return keyword.

```
def add(a, b):
    return a + b
result = add(3, 4)  # result = 7
```

Default Arguments:

You can provide default values for parameters, so if no argument is passed, the function uses the default.

```
def function(name="Advik"):
    print("Hello, " + name)
function()  # Output: Hello, Advik
```

Keyword Arguments:

When calling a function, you can specify parameters by name to avoid confusion, especially with many parameters.

```
def student_info(name, age):
    print(f"Name: {name}, Age: {age}")
student_info(age=20, name="Advik")  # Output: Name: Advik, Age: 20
```

EXAMPLE:

```
def function():
    print("Hello Students")
function() #function call
```

FUNCTION PARAMETERS:

• The name of the function while calling and the number of arguments must match with the function definition.

• If there is a mismatched in parameters passed and declared, then an error will be returned.

• If the data type of the parameters passed does not match with that of the function, then an error is generated.

TYPES OF FUNCTIONS IN PYTHON:

1) Built-in functions
2) User-Defined Functions
3) Anonymous functions (lambda functions)

LAMBDA OR ANONYMOUS FUNCTION:

Lambda functions are not declared using def keyword. Instead, **lambda** keyword is used. Lambda functions are required at place where they have been created in the program. Any number of arguments ca be supplied to lambda function, but it must a single expression.

Lambda functions have no name. Lambda function cannot access variables other than those in their parameter list. Lambda function can take N number of arguments. Lambda function does not have any return statement.

SYNTAX:

lambda arguments : expression

z = lambda x, b : x* b print(z(7, 18))

x = lambda m, n, c: m + n + c print(x(6, 2, 7))

GLOBAL AND LOCAL VARIABLE:

• Global variables are defined in the main body of the program and can be used throughout the program. They are also accesible by all the function in the program.

• Local Variables are defined within the function and their scope is within that function only. They are not related with the same name variable defined outside the function.

Questions-

1) What is a function in Python? Explain its purpose.
2) How do you define a function in Python? What keyword is used?
3) What is the difference between a built-in function and a user-defined function?
4) Explain the concept of arguments and parameters in Python functions.
5) What is the difference between a function with return values and one without return values?

Conclusion:

Functions in Python are used to group code that performs a specific task, making programs easier to manage and reuse. They can take inputs, called arguments, and return results. Python has different types of functions, like user-defined, built-in, and recursive functions. Features like default arguments and the ability to handle varying numbers of inputs make functions flexible and powerful for writing efficient code.

EXPERIMENT NO.4

Study and implement Python programs using String.

Practical Session Plan

Time (min)	Content	Learning Aid / Methodology	Faculty Approach	Typical Student Activity	Skill / Competency Developed
20	Explanation of Experiment	Chalk & Talk , Presentation	Introduces, Facilitates, Explains	Listens	Knowledge, Communication, intrapersonal, Application
10	Preparation of Algorithm and Logic Development for Program	Paper Work	Introduces, Facilitates, Explains	Discuss,	Knowledge, Communication, intrapersonal, Application
60	Coding of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
20	Testing of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
10	Results and conclusions	Keywords	Lists, Facilitates	Listens, Participates, Discusses	Knowledge, Communication, intrapersonal, Comprehension

Title: Study and implement Python programs using String.

Write a Python program to:

1: Create a string made of the first, middle and last character
2: Append new string in the middle of a given string
3: Create a new string made of the first, middle, and last characters of each input string
4: Arrange string characters such that lowercase letters should come first
5: Count all letters, digits, and special symbols from a given string
6: Find all occurrences of a substring in a given string by ignoring the case
7: Calculate the sum and average of the digits present in a string
8: Write a program to count occurrences of all characters within a string
9: Reverse a given string
10: Find the last position of a given substring

Objective:- To Study python in-built data structures

Aim/ Problem Statement: Write a Python Program to create user define Lists and Execute Different Operation on List.

Theory:

A string is a collection of characters. Strings are used to use text in python. In python string can be used in the following ways:

– Using Single Quotes: 'Welcome'

– Using Double Quotes: "Welcome"

– Using Triple Quotes: Triple quotes are used to

specify multiple line of strings.

for example:

''' Hello everyone,

How are you all ?

Let's play a game'''

Strings are Arrays :

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"
print(a[1])
```

Indexing :

• Subscript operator [ ] is used to access the individual characters in a string.

• The value or the expression inside the [ ] brackets is called index.

• This index value is used to access a particular character from a string.

• The index of the first character in a string is 0 (zero) and that of the last character is n-1 where n is the total number of character in a string

.• For example:

Consider a string : “PYTHON”

• Total number of characters in the string is 6

(six) i.e. n=6.

IndexConcatenation

• Concatenation means joining.

• In python two strings can be joined together using + operator.

• For example:

first_str = “New”

second_str= “Delhi”

first_str + second_str

Appending

• Append is used to add a string at the end of another string.

• The operator used for append operation is +=

Example

```
fname = "Hello"

lname = "Students"

fname += lname
```

print(fname)

Output:

HelloStudents

String Formatting

• format() function is used.

• Syntax:

– format(value, format specifier)

< is used for left justify

> Is used for right justify

^ is used for centre alignment

String slices, function and methods:

Slice:

• Slice can be used to return a range of characters from a string.

• This is done by specifying the start index and the end index, separated by a colon, to return a part of the string.

Example

str='spyder'

print(str[2:5])

Output: yde

Example

str='spyder'

print(str[3:-1])

Output: de

Example

```
str='spyder'

print(str[:4])
```

Output: spyd

String Methods :

• upper() : To upper case

• lower(): To lower case

• endswith(): Returns true if the string ends with the specified value.

• split(): Splits the string at the specified separator, and returns a list.• find(): Searches the string for a specified value and returns the position of where it was found.

• strip(): Returns a trimmed version of the string.

• count(): Returns the number of times a specified value occurs in a string.

• isalnum(): Returns True if all characters in the string are alphanumeric

Questions:

1) What is a string in Python, and how can you create one?
2) Explain string immutability with an example. Why are strings immutable in Python?
3) How do you concatenate two strings in Python? Write a program to demonstrate this.
4) What is slicing in Python? Provide an example where you extract a substring from a given string.
5) Write a Python program to count the number of vowels in a given string.
6) How do the split() and join() methods work in Python?
7) How can you reverse a string in Python using slicing?
8) What is the difference between upper(), lower(), and swapcase() methods in Python? Provide examples for each.
9) How would you remove leading and trailing whitespaces from a string?

Conclusion:

In this lab, we explored various string manipulation techniques in Python, including concatenation, slicing, and the use of built-in methods like split(), replace(), and others. We learned that strings in Python are immutable, meaning any modification results in the creation of a new string. These operations are fundamental for efficient text processing and form a critical part of programming tasks involving textual data. Mastering these concepts is essential for handling real-world applications involving strings.

EXPERIMENT NO.5

Write a Python program to create user-defined lists and execute its operations.

Practical Session Plan

Time (min)	Content	Learning Aid / Methodology	Faculty Approach	Typical Student Activity	Skill / Competency Developed
20	Explanation of Experiment	Chalk & Talk , Presentation	Introduces, Facilitates, Explains	Listens	Knowledge, Communication, intrapersonal, Application
10	Preparation of Algorithm and Logic Development for Program	Paper Work	Introduces, Facilitates, Explains	Discuss,	Knowledge, Communication, intrapersonal, Application
60	Coding of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
20	Testing of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
10	Results and conclusions	Keywords	Lists, Facilitates	Listens, Participates, Discusses	Knowledge, Communication, intrapersonal, Comprehension

Title: Write a Python program to create user-defined lists and execute its operations

A. myList= [10, 40, 50, 20, 30,10,40,10]
B. yourList = ['saw','small','foxes','he','six']

Use built in methods to perform following operations on the list:

1) Append integer 60 into myList
2) Insert 70 on 2nd Position
3) Sort myList in ascending and descending order.
4) Sort yourList in ascending and descending according to length of strings.
5) Add float value 3.5 into yourList.
6) Use POP and remove method to remove 3.5
7) Create ourList by merging myList and yourList
8) Find sum of elements in mylist.
9) Find smallest, largest and second largest number in a myList.
10) Count occurrences of all element in a list
11) Perform Data slicing to display string elements from ascending sorted yourList as: -

a. Display - 'saw','six','small'
b. Use negative indices to display - . 'six','small', 'foxes'
c. All elements after mid of the list (In both directions).

Alternate elements in both direction middle of list..

Objective:- To Study python program to create user-defined lists and execute its operations**.**

Aim/ Problem Statement: Write a Python Program to create user define Lists and Execute Different Operation on List.

Theory:

In Python, lists are versatile, user-defined data structures that allow storing multiple elements of different data types in an ordered and mutable collection. Lists can include numbers, strings, or even other lists, making them a powerful tool for managing collections of items.

You can create a list by enclosing elements within square brackets ([ ]), separated by commas. Python provides several operations for manipulating lists.

List is a dynamic data type available in python. List is used to store multiple items in a single variable. The elements are stored in a list as a comma-seperated values between the square bracket. The most important feature of List is that it can store elements elements of different data types.

Syntax for defining a list

List_variable=[val1, val2, val3,….]

For example:

list_X= [10, 11, "Advik", 'a', 3.14]

Example

```
first_list= [1,2,3,4,5]

print (first_list)

second_list=['Delhi','b','c',10,20,30,3.14]

print(second_list)
```

• In a List each element has a specific index which starts from zero and is auto incremented for next value. Therefore it can also be considered as an ordered set.

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

Example of Index

```
List_city = ["Delhi", "Mumbai", "Kolkata"]

print("\n List Items: ")

print(List_city[0]) # Index 0

print(List_city[2]) # index 2
```

Multi-Dimensional List

```
List = [[1, 2] , [1]]

print(List[0][1])

print(List[1][0])
```

len function # len function determine how many items a list has.

```
List_num = [100, 200, 114,12,34]

print(len(List_num))
```

List Slicing:

```
List_A = ['A','B','C','D','E','F','G']

print(List_A[2:5])
```

In the above list note that the range is from 2(Inclusive) to 5(exclusive). Therefore values at index position 2,3 and 4 will be printed. Another point to keep in mind is that a the first element of the list has index 0 (zero).

Therefore the output will be: ['C', 'D', 'E']

INDEX	0	1	2	3	4	5	6
ELEMENTS	A	B	C	D	E	F	G

List_A = ['A','B','C','D','E','F','G']

print(List_A[:4])

In the above list note that the range will begin from 0th index to the 3rd index. Therefore values at index position 0,1,2, and 3 will be printed.

Therefore the output will be: ['A', 'B', 'C', 'D']

Negative Indexing in a List

Negative indexing means start from the end of the list. Therefore an index value of [-1] refers to the last item of the list.

Similarly an index value of [-2] refers to the second last item of the list and so on.

Example

List = [1, 2, 'a', 4, 'b', 6, 'c']

print(List[-1]) # prints c

print(List[-3]) # prints b

print(List[-5]) # prints a

remove method

List = [1, 2, 3, 4, 5, 6, 7]

print(List)

 # Removing elements from List using remove() method

List.remove(5)

List.remove(6)

print("\n List after Removal of two elements: ")

print(List)

Removing elements from List using iterator method

```
List=[1,2,3,4,5,6,7,8,9]
for i in range(1,5):
List.remove(i)
print("\n List after Removing a range of elements: ")
print(List)
```

list operation:

Converts an iterable (tuple ,string, set, dictionary) to a list.

For example consider the following code.

```
list1=list("MORNING")
print(list1)
```

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
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
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

Adding elements to list using append

```
List_new = [ ]
print("Initial blank List: ")
print(List_new)
```

 # Addition of Elements

```
List_new.append(10)
List_new.append(20)
List_new.append(30)
print("\nList after Addition of Three elements: ")
print(List_new)
```

Few List Methods

```
ListA = [2, 4, 2, 6, 7, 2]
print(ListA.count(2)) #counts no. of times element appears
ListA.insert(2,50) #Insert object at specific index
```

```
print(ListA)
print(ListA.pop(2)) #Removes an element at the specified index
ListA.remove(2) #removes or deletes and element from the list
print(ListA)
```

reverse and sort method

```
ListA = [2, 4, 2, 6, 7, 2]
ListA.reverse() #Reverses the elements in the list
print(ListA)
ListA.sort() #Sorts the elements in the list
print(ListA)
```

extend method

```
ListA = [2, 4, 2, 6, 7, 2]
ListB = [100,200,300]
ListA.extend(ListB) #adds elements in a list to the end of another list
print(ListA)
List_A = [1, 3, 6, 2] + [5, 4, 7] #Concatenation print (List_A)
print ("Max =", max(List_A)) #prints maximum value
print ("Min =", min(List_A)) #prints minimum value
print ("Sum =", sum(List_A)) #sum operation add the values in the list of numbers
```

Questions:

1) What is a list in Python, and how is it different from a tuple?
2) How do you create a user-defined list in Python? Provide an example.
3) Write a Python program to add elements to a list using the append() method.
4) How can you remove an element from a list? Explain with an example.
5) What are the differences between remove(), pop(), and del in Python lists?
6) How do you access elements from a list using indexing and slicing?
7) Write a Python program to find the length of a list using the len() function.
8) How can you concatenate two lists in Python? Provide a program to demonstrate it.
9) Explain the difference between sort() and sorted() when working with Python lists.
10) How would you check if an element exists in a list using Python?
11) Write a Python program to find the maximum and minimum values from a list.

Conclusion:-

Hence we conclude, user-defined lists in Python provide a flexible and powerful way to store, access, and manipulate collections of data. Through various operations such as adding, removing, sorting, and slicing, Python lists can handle a wide range of tasks efficiently. Understanding how to implement these operations is fundamental for managing data and solving real-world problems. Lists also allow for dynamic modifications, making them essential for developing complex programs in Python.

EXPERIMENT NO.6

Write a Python program to create user defined tuple and execute its operations.

Practical Session Plan

Time (min)	Content	Learning Aid / Methodology	Faculty Approach	Typical Student Activity	Skill / Competency Developed
20	Explanation of Experiment	Chalk & Talk , Presentation	Introduces, Facilitates, Explains	Listens	Knowledge, Communication, intrapersonal, Application
10	Preparation of Algorithm and Logic Development for Program	Paper Work	Introduces, Facilitates, Explains	Discuss,	Knowledge, Communication, intrapersonal, Application
60	Coding of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application

20	Testing of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
10	Results and conclusions	Keywords	Lists, Facilitates	Listens, Participates, Discusses	Knowledge, Communication, intrapersonal, Comprehension

Title: Write a Python program to create user defined tuple and execute its operations.

A. myTuple= (10, 20, 30)
B. yourTuple = ("Pune", 'Mumbai', "Delhi")
C. mixTuple= ('Foo',[1,2,3],'True')
D. nestedTuple=((' **Wes McKinney**',' **Python for Data Analysis**',' **O'Reilly Media**'),
('**Mark Lutz**',' **Programming Python**',' **O'Reilly Media**'),
('**Charles Severance**',' **Python for Everybody**',' **Blumenberg**'))

Use built in methods to perform following operations on the tuple:

1) Merge myTuple and yourTuple into ourTuple.
2) Convert myTuple into list myList and reverse.
3) Unpack yourTuple values into three variables - District, State, and National.
4) Display all elements of mixTuple.
5) Add 4 into list element of mixTuple
6) Perform algebraic operations addition and multiplication on myTuple and yourTuple.
7) Access information from nestedTuple and display the information as:
Name of Author = ' **Wes McKinney**',
Name of Book = ' **Python for Data Analysis**',
Name of Publisher= ' **O'Reilly Media**'

Objective:- To Study python in-built data structures

Aim/ Problem Statement: Write a Python Program to create user define Lists and Execute Different Operation on List.

Theory:

Tuples are used to store multiple items in a single variable. Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage. A tuple is a collection which is ordered and unchangeable. Tuples are written with round brackets **( )**.

Example:

Create a Tuple:

```
tup=("Advik", "Aarya", "Yash")

print(tup)   # output- ('Advik','Aarya','Yash')
```

Tuple Items:

Tuple items are ordered, unchangeable, and allow duplicate values. Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered:

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

Unchangeable:

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow Duplicates:

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

Example:

Tuples allow duplicate values:

```
tup = ("apple", "banana", "cherry", "apple", "cherry")
print(tup)  # output- ('apple', 'banana', 'cherry', 'apple', 'cherry')
```

Tuple Length:

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

```
tup = ("apple", "banana", "cherry", "apple", "cherry")
print(len(tup))  # output- 5
```

Tuple Items - Data Types:

Tuple items can be of any data type:

Example:

String, int and boolean data types:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

Access Tuple Items:

You can access tuple items by referring to the index number, inside square brackets:

Example:

Print the second item in the tuple:

```
tup = ("Advik", "Aarya", "Yash")
print(tup[1])  # output- Aarya
```

Negative Indexing:

Negative indexing means start from the end.

[-1 refers to the last item, -2 refers to the second last item etc.]

```
tup=("apple","banana","cherry")
print(tup[-1])  # output- cherry
```

Change Tuple Values:

Once a tuple is created, we cannot change its values. Tuples are unchangeable, or immutable. We can convert the tuple into a list, change the list, and convert the list back into a tuple.

Convert the tuple into a list for changing it:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tup(y)
print(x)   # output- ("apple", "kiwi", "cherry")
```

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

1. What is a tuple in Python?
2. How do you create a tuple in Python?
3. What is the difference between a tuple and a list in Python?
4. How can you access elements of a tuple?
5. Can you modify elements of a tuple after it is created?
6. How do you concatenate two tuples?
7. How do you check if an element exists in a tuple?
8. What operations can be performed on tuples?

Conclusion:-

In this lab, we explored how to create and manipulate user-defined tuples in Python. Tuples, being immutable, provide a way to store a collection of items in a fixed structure, which ensures data integrity. Through various operations like accessing elements, slicing, concatenation, and using built-in methods like count() and index(), students gained practical experience with tuple functionalities. This understanding is crucial for handling immutable data efficiently in Python applications.

EXPERIMENT NO.7

Write a Python program that reads the Particulars.txt file containing the elements in the string format and perform given operations on the myWallet dictionary.

Practical Session Plan

Time (min)	Content	Learning Aid / Methodology	Faculty Approach	Typical Student Activity	Skill / Competency Developed

20	Explanation of Experiment	Chalk & Talk , Presentation	Introduces, Facilitates, Explains	Listens	Knowledge, Communication, intrapersonal, Application
10	Preparation of Algorithm and Logic Development for Program	Paper Work	Introduces, Facilitates, Explains	Discuss,	Knowledge, Communication, intrapersonal, Application
60	Coding of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
20	Testing of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
10	Results and conclusions	Keywords	Lists, Facilitates	Listens, Participates, Discusses	Knowledge, Communication, intrapersonal, Comprehension

Title: Write a Python program that reads the Particulars.txt file containing the elements in the string format and perform given operations on the myWallet dictionary.

Particular.txt : Diary CCards DCards VCards CCards DCards VCards VCards VCards VCards

Create a dictionary myWallet by reading the elements and get() method.

myWallet={'Diary': 1, 'CCards': 2, 'DCards': 2, 'VCards': 5}

Perform following operations on myWallet dictionary:

1) A new credit card is added in myWallet
2) Check that any Photograph available in myWallet or not in True or False output.
3) Add four Photographs in myWallet.

4) Remove Photographs using del() method and pop() method.
5) Represent the particulars of dictionary in the form of tuple.
6) Sort the item of myWallet in ascending order based on items.

Sort the items of myWallet in the ascending order based on item quantity

Objective:- To Study python in-built data structures

Aim/ Problem Statement: Write a Python Program to create user define Dictionary and Execute Different Operation on Dictionary.

Theory:

Dictionary is a built-in Python Data Structure and are used to store data values in key:value pairs. Each key is separated from its value by a colon (:). A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

Dictionaries are not indexed by a sequence of numbers but indexed based on keys.

The keys in the dictionary must be unique and of immutable data type i.e. strings, numbers or

tuples. The value doesn't have any such restrictions. Dictionary are case-sensitive i.e. two keys with similar name but different case will be treated differently. The elements within the dictionary are accessed with the help of the keys rather than its relative position.

Syntax:

dictionary_name = {key_1: value_1, key_2: value2, key_3: value_3}

Example:

```
d={
  "name": "Advik",
  "age": 6,
  "address": "Pune"
}
print(d)
```

Dictionary Functions:

1. Access Dictionary:

You can access dictionary values by using their keys.

Syntax:

dictionary[key]

Example:

```
d = {'name': 'Advik', 'age': 6, 'City': 'Pune'}

print(d['name'])

# Output: Advik
```

If the key does not exist, Python raises a KeyError. You can use the get() method to avoid this, which returns None or a default value if the key is not found.

Example using get():

```
print(d.get('name'))

# Output: Advik

print(d.get('job', 'N/A'))

# Output: N/A (if the key doesn't exist)
```

2. Change Dictionary:

You can update the value of a key in a dictionary by assigning a new value to it.

Syntax:

```
dictionary[key] = new_value
```

Example:

```
d['age'] = 30

print(d)

# Output: {'name': 'Advik', 'age': 30, 'City': 'Pune'}
```

3. Add Dictionary:

To add a new key-value pair to a dictionary, you can simply assign a value to a new key.

Syntax:

```
dictionary[new_key] = value
```

Example:

```
d['job'] = 'Engineer'

print(d)
```

Output: {'name': 'Advik', 'age': 30, 'City': 'Pune', 'job': 'Engineer'}

4. Remove Dictionary Elements:

You can remove key-value pairs from a dictionary using several methods.

Methods:

pop(key): Removes the item with the specified key and returns the value.

del: Deletes the key-value pair.

popitem(): Removes the last inserted key-value pair.

clear(): Removes all the items from the dictionary.

Example:

 # Using pop():

removed_value = d.pop('age')

print(removed_value)

Output: 30

print(d) # Output: {'name': 'Advik', 'City': 'Pune', 'job': 'Engineer'}

 # Using del:

del d['city']

print(d) # Output: {'name': 'Advik', 'job': 'Engineer'}

 # Using popitem():

d={'name': 'Advik', 'job': 'Engineer'}

d.popitem()

print(d) # Output: ('name': 'Advik')

5. Copy Dictionary:

You can create a copy of a dictionary using either the copy() method or the dict() constructor.

Syntax:

new_dict = d.copy()# or

new_dict = dict(d)

Example:

d_copy = d.copy()

print(d_copy) # Output: {'name': 'Advik', 'job': 'Engineer'}

Questions:

1) Define a Python dictionary. How does it differ from other data structures like lists and tuples?
2) Explain how to access elements in a dictionary using keys. What is the difference between accessing a value using square brackets [] and using the get() method?
3) How can you change the value of an existing key in a dictionary? Give an example.
4) How do you add a new key-value pair to a dictionary? Explain with an example.
5) What are the different methods available for removing elements from a dictionary? Explain the difference between pop(), del, and popitem().

Conclusion:- Python dictionaries provide a flexible way to store and manipulate data as key-value pairs. The dictionary is a very versatile data structure, allowing easy access, modification, and management of large datasets.Hence We studied and Implemented Dictionary in Python.

EXPERIMENT NO.8

Write a Python program to construct Python built-in data structure Set.

Practical Session Plan

Time (min)	Content	Learning Aid / Methodology	Faculty Approach	Typical Student Activity	Skill / Competency Developed
20	Explanation of Experiment	Chalk & Talk , Presentation	Introduces, Facilitates, Explains	Listens	Knowledge, Communication, intrapersonal, Application

10	Preparation of Algorithm and Logic Development for Program	Paper Work	Introduces, Facilitates, Explains	Discuss,	Knowledge, Communication, intrapersonal, Application
60	Coding of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
20	Testing of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
10	Results and conclusions	Keywords	Lists, Facilitates	Listens, Participates, Discusses	Knowledge, Communication, intrapersonal, Comprehension

Title: Write a Python program to construct Python built-in data structure Set.

1) Create empty set 'Engineers' and 'Managers'.
2) Using input method add elements in 'Engineers' and 'Managers':

Engineers={'Jane', 'John', 'Janice', 'Jack'}
Managers ={'Jane', 'Jack', 'Susan', 'Zack'}

3) Display all engineers in this format : "Name of Engineer is --- " Jane
4) Copy all managers and construct a tuple myManagers =('Jane', 'Jack', 'Susan', 'Zack')
5) Copy all engineers and construct a list myEngineers ={'Jane', 'John', 'Janice', 'Jack'}
6) Add new manager 'Jenifer'
7) Create a third set Engineer_Manager by merging both Engineers and Managers sets.
8) Display the name of engineers who are not managers
9) Display the name of engineers who also serving as managers. Display the name of person who is either engineer or manager only but not performing both jobs

Objective:- To Study python in-built data structures

Aim/ Problem Statement: Write a Python Program to create user define Set and Execute Different Operation on Set.

Theory:

Sets are used to store multiple items in a single variable. Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage. A set is a collection which is unordered, unchangeable*, and unindexed. To create a set, you can use curly braces {} or the set() function for an empty set.

Syntax:

```
# Using curly braces

set_name = {element1, element2, element3}

# Using set() function (for an empty set or type casting)

set_name = set()
```

Example:

```
s = {'Advik', 'Aarya', 'Yash', 'Paras'}

print(s) # Output: {'Advik', 'Aarya', 'Yash', 'Paras'}
```

1. Access Set Items:

You cannot access set items by index, as sets are unordered. However, you can check if a specific element is present in a set using the in keyword or loop through the set.

Syntax:

```
element in set_name  # Returns True if element exists in the set
```

Example:

```
my_set = {'Advik', 'Aarya', 'Yash', 'Paras'}

print('Omkar' in my_set)  # Output: False
```

2. Add Set Items:

You can add elements to a set using the add() method for a single item or the update() method to add multiple items at once.

Syntax:

```
set_name.add(element)        # Adds a single element

set_name.update([elem1, elem2])  # Adds multiple elements
```

Example:

```
my_set = {'Advik', 'Aarya', 'Yash', 'Paras'}

my_set.add('Omkar')

print(my_set)  # Output: {'Advik', 'Aarya', 'Yash', 'Paras', 'Omkar'}
```

3. Remove Set Items:

You can remove items from a set using the remove() or discard() methods. The remove() method raises an error if the element is not found, while discard() does not. You can also use the pop() method to remove and return an arbitrary element.

Syntax:

```
set_name.remove(element)  # Raises an error if the element doesn't exist

set_name.discard(element)  # Does not raise an error if the element doesn't exist

set_name.pop()  # Removes a random element
```

Example of remove:

```
my_set = {'Advik', 'Aarya', 'Yash', 'Paras'}

my_set.remove('Yash')

print(my_set)  # Output: {'Advik', 'Aarya', 'Paras'}
```

Example of discard:

```
my_set.discard('Omkar')   # No error if element is not found

print(my_set)
```

Example of pop:

```
removed_item = my_set.pop()  # Removes a random element

print(removed_item)
```

4. Join Sets:

You can join two or more sets using set operations such as union and intersection. The union() method returns a new set containing all unique elements from both sets, while the intersection() method returns only the elements common to both sets.

Syntax:

set1.union(set2) # Returns a set containing all elements from set1 and set2

set1.intersection(set2) # Returns a set with only elements found in both sets

Example:

set1 = {'Advik', 'Aarya', 'Paras'}

set2 = {'Yash', 'Paras'}

 # Union of sets:

combined_set = set1.union(set2)

print(combined_set) # Output: {'Advik', 'Aarya','Yash', 'Paras'}

 # Intersection of sets:

common_set = set1.intersection(set2)

print(common_set) # Output: {'Paras'}

Python has a set of built-in methods that you can use on sets:

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set

remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_ update()	Inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Questions:

1) What is a set in Python, and how does it differ from lists and tuples?
2) Explain the key features of sets, particularly the concept of unordered and unique elements.
3) How can you check if an element exists in a set? Write the syntax and explain?
4) Explain the difference between the remove() and discard() methods when deleting an element from a set.
5) What are the various set operations available in Python (e.g., union, intersection, difference)? Explain any two operations with examples.

Conclusion:- Sets in Python are a versatile data structure that store unique, unordered elements. They are ideal for tasks like removing duplicates and performing operations such as union, intersection, and difference. Unlike lists, sets do not maintain order, making them useful when the sequence of elements is not important. Mastering sets helps students efficiently handle and compare large datasets in Python. Hence We studied and Implemented Sets in Python.

EXPERIMENT NO.9

Study and implement Python programs on File Handling.

Practical Session Plan

Time (min)	Content	Learning Aid / Methodology	Faculty Approach	Typical Student Activity	Skill / Competency Developed

20	Explanation of Experiment	Chalk & Talk , Presentation	Introduces, Facilitates, Explains	Listens	Knowledge, Communication, intrapersonal, Application
10	Preparation of Algorithm and Logic Development for Program	Paper Work	Introduces, Facilitates, Explains	Discuss,	Knowledge, Communication, intrapersonal, Application
60	Coding of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
20	Testing of program	Keyboard, mouse, computers.	Introduces, Facilitates, Explains	Discuss, participates	Knowledge, Communication, intrapersonal, Application
10	Results and conclusions	Keywords	Lists, Facilitates	Listens, Participates, Discusses	Knowledge, Communication, intrapersonal, Comprehension

Title: Study and implement Python programs on File Handling.

Programs to demonstrate File handling in Python:
1: Program to read the contents from a text file and display the same on screen.
2: Program to count the number of lines, words and characters from a text file.
3: Program to read first n lines from a text file
4: Program to read lines from a text file and find the length of the longest line.
5: Program to read last n lines of a file
6: Program to count the frequency of words in a file.
7: Program to count lines starting with a word "The"
8: Program to replaces all special characters by space

9: Program to count occurrences of a word in a file

Objective:- To Study File Handling in python.

Aim/ Problem Statement: Write a Python Program for File Handling.

Theory:

The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists.

Syntax:

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

Open file:

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file.

Example:

```
f = open("demofile.txt", "r")
print(f.read())
```

Read Lines:

You can return one line by using the readline() method:

Example:

Read one line of the file:

```
f = open("demofile.txt", "r")
print(f.readline())
```

Write to an Existing File:

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Example:

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Create a New File:

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

Example:

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Delete a File:

To delete a file, you must import the OS module, and run its os.remove() function:

Example:

Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

Delete Folder:

To delete an entire folder, use the os.rmdir() method:

Example:

Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```

Question:

1) What is file handling in Python, and why is it important?
2) Explain the purpose of the open() function in Python. What are the different file modes available?
3) What is the difference between reading a file using the read(), readline(), and readlines() methods? Provide examples.
4) Why is it important to close a file after performing file operations? What is the role of the close() function?
5) Explain the difference between the modes 'r', 'w', 'a', and 'r+' in file handling. In which situations would you use each mode?

Conclusion:- File Handling in Python can easily read, write, and manage data in files. Using functions like open(), read(), and write(), you can work with text and binary files. The with statement ensures files are handled safely, preventing errors. Understanding file handling is important for tasks like data processing and logging in real-world projects.Hence We studied and Implemented File Handling in Python.