Dr B R Ambedkar National Institute of Technology

Jalandhar,Punjab(144011)

(Center for Artificial Intelligence)

Python Programming Lab File

Submitted to :

Dr. Diksha

Submitted By:

Satyartha Shukla

24901321

CONTENTS

S. NO.	EXPERIMENT	DATE OF SUBMISSION	PAGE NO.	REMARKS
1.	Install Python and write basic programs to explore its syntax and functionality	13-Aug-2024	3-5	
2.	Demonstrate python operators and develop code for given problem statements	20-Aug-2024	6-9	
3.	Demonstrate conditional and loop statements and develop code for given problem statements	27-Aug-2024	10-17	
4.	Demonstrate list operations and develop code for given problem statements	03-Sept-2024	18-21	
5.	Demonstrate arrays and tuples and develop code for given problem statements	10-Sept-2024	22-28	
6.	Demonstrate functions and modules and develop code for given problem statements	24-Sept-Nov	29-32	
7.	Demonstrate Sets and develop code for given problem statements	01-Oct-2024	33-36	
8.	Demonstrate Dictionaries and develop code for given problem statements	22-Oct-2024	37-40	
9.	Demonstrate Strings and develop code for given problem statements	05-Nov-2024	41-48	
10.	Demonstrate File Handling and develop code for given problem statements	12-Nov-2024	49-51	

11.	Demonstrate Class & Objects and develop code for given problem statements	19-Nov-2024	52-56	
12.	Demonstrate Inhertitance and develop code for given problem statements	26-Nov-2024	57-64	

EXPERIMENT 1

OBJECTIVE: Install Python and write basic programs to explore its syntax and functionality.

THEORY:

Python is a high-level, interpreted programming language created by Guido van Rossum in 1991. Known for its simplicity and readability, Python uses indentation for defining code blocks, making it beginner-friendly. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is dynamically typed and has an extensive standard library that simplifies complex tasks like file handling, data manipulation, and web development. Its versatility makes it widely used in various fields such as web development, data science, artificial intelligence, automation, and game development. Python's large community and open-source nature further enhance its adaptability and resource availability.

INSTALLATION STEPS:

- **Download Python**:
  - Go to python.org.
  - Download the latest Python 3 version for your OS.
- **Install Python**:
  - Run the installer.
  - Select *"Add Python to PATH"* and click **Install Now**.
- **Verify Installation**:
  - Open a terminal and type:
    - `python --version` (Windows/macOS/Linux).
  - Confirm the version is displayed.
- **Optional**: Install an IDE like **PyCharm**, **VS Code**, or use **IDLE** (bundled with Python).

CODE:

```
# Simple Program to perform arithmetic operations on two numbers

a = 7

b = 2

print("sum: ", a+b)

print("diff: ", a-b)

print("mult: ", a*b)
```

print("div: ", a/b)

print("mod: ", a%b)

print("floor: ", a//b)

print("power: ", a**b)

RESULTS:

[2]:
```
a = 7
b = 2
print("sum: ", a+b)
print("diff: ", a-b)
print("mult: ", a*b)
print("div: ", a/b)
print("mod: ", a%b)
print("floor: ", a//b)
print("power: ", a**b)
```

```
sum:  9
diff:  5
mult:  14
div:  3.5
mod:  1
floor:  3
power:  49
```

EXPERIMENT 2

OBJECTIVE: Demonstrate python operators and develop code for given problem statements:

1) Datatype Conversion:
   a. convert char to int, and find octal, hex value of given value
   b. convert string to tuple, set and list
2) Types of operators:
   a. perform arithmetic operations on 2 numbers
   b. demonstrate use of comparison, logical, identity, membership operators

THEORY:

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

- Arithmetic operators - Arithmetic operators are used with numeric values to perform common mathematical operations
- Assignment operators - Assignment operators are used to assign values to variables
- Comparison operators - Comparison operators are used to compare two values
- Logical operators - Logical operators are used to combine conditional statements
- Identity operators - Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location
- Membership operators - Membership operators are used to test if a sequence is presented in an object
- Bitwise operators - Bitwise operators are used to compare (binary) numbers

CODE:

1. Convert char to int, and find octal, hex value of given value

```
# Convert char to int
a = '4'
b = ord(a)
print(b)
print(type(b))

# Find hex value of given int
b = hex(56)
print(b)
print(type(b))

# Convert int to octal
b = oct(56)
print(b)
print(type(b))
```

2. Convert string to tuple, set and list

```
x = 'javaTpoint'
y=tuple(x)
print("after converting the string to a tuple: ", end="")
print(y)

y = set(x)
print("after converting the string to a set: ", end="")
print(y)

y = list(x)
print("after converting the string to a list: ", end="")
print(y)
```

3. Perform arithmetic operations on 2 numbers

```
# Arithmetic operators in python
a = 7
b = 2
print("sum: ", a+b)
print("diff: ", a-b)
print("mult: ", a*b)
print("div: ", a/b)
print("mod: ", a%b)
print("floor: ", a//b)
print("power: ", a**b)
```

4. Demonstrate use of comparison, logical, identity, membership operators

```
# Comparison Operators
a=5
b=2
print(a==b)
print(a!=b)
print(a>b)
print(a<b)
print(a<=b)
print(a>=b)

# Logical Operators
a=5
b=6
print((a>2) and (b>=6))
print((a>2) or (b>=6))

# Identity operators
```

```
x1=5
y1=5
x2='Hello'
y2='Hello'
x3=[1,2,3]
y3=[1,2,3]
print(x1 is not y1)
print(x2 is y2)
print(x3 is y3)
```

RESULTS:

1. Convert char to int, and find octal, hex value of given value

```
# Convert char to int
a = '4'
b = ord(a)
print(b)
print(type(b))
```

```
52
<class 'int'>
```

```
# Find hex value of given int
b = hex(56)
print(b)
print(type(b))
```

```
0x38
<class 'str'>
```

```
<class 'str'>
```

```
[10]: # Convert int to octal
b = oct(56)
print(b)
print(type(b))
```

```
0o70
<class 'str'>
```

2.Convert string to tuple, set and list

```
x = 'javaTpoint'
y=tuple(x)
print("after converting the string to a tuple: ", end="")
print(y)
```

```
after converting the string to a tuple: ('j', 'a', 'v', 'a', 'T', 'p', 'o', 'i', 'n', 't')
```

```
y = set(x)
print("after converting the string to a set: ", end="")
print(y)
```

```
after converting the string to a set: {'p', 'T', 'n', 'i', 'a', 'j', 't', 'o', 'v'}
```

```
y = list(x)
print("after converting the string to a list: ", end="")
print(y)
```

```
after converting the string to a list: ['j', 'a', 'v', 'a', 'T', 'p', 'o', 'i', 'n', 't']
```

3. Perform arithmetic operations on 2 numbers

[18]:
```
a = 7
b = 2
print("sum: ", a+b)
print("diff: ", a-b)
print("mult: ", a*b)
print("div: ", a/b)
print("mod: ", a%b)
print("floor: ", a//b)
print("power: ", a**b)
```

```
sum:  9
diff:  5
mult:  14
div:  3.5
mod:  1
floor:  3
power:  49
```

4. Demonstrate use of comparison, logical, identity, membership operators

:
```
# Comparison Operators
a=5
b=2
print(a==b)
print(a!=b)
print(a>b)
print(a<b)
print(a<=b)
print(a>=b)
```

```
False
True
True
False
False
True
```

:
```
# Logical Operators
a=5
b=6
print((a>2) and (b>=6))
print((a>2) or (b>=6))
```

```
True
True
```

```
# Identity operators
x1=5
y1=5
x2='Hello'
y2='Hello'
x3=[1,2,3]
y3=[1,2,3]
print(x1 is not y1)
print(x2 is y2)
print(x3 is y3)
```

```
False
True
False
```

EXPERIMENT 3

OBJECTIVE: Demonstrate conditional and loop statements and develop code for given problem statements:

1. Conditional Statements –
    1) WAP to take input from a user and then check whether it is a number or a character. If it is a char, determine whether it is Upper case or lower case
    2) WAP that displays the user to enter a number between 1 to 7 and then displays the corresponding day of the week
2. Looping -
    1) Demonstrate nested looping
        i. Nested loop to print given pattern

        *

        * *

        * * *

        * * * *

    2) Demonstrate while loop inside for loop
    3) WAP to print the pattern

        1

        2 2

        3 3 3

        4 4 4 4

        5 5 5 5 5

    4) WAP using for loop to calculate factorial of a number
    5) WAP that displays all leap years from 1900 to 2101
    6) WAP to sum the series numbers - 1 + 1/2 + ... + 1/n using for loop

THEORY:

Conditional Statements

Conditional statements allow a program to execute a specific block of code based on whether a condition is true or false. In Python, conditional statements include:

if: Executes a block if the condition is true.

elif: Adds multiple conditions.

else: Executes a block if all previous conditions are false.

Loops

Loops allow repeated execution of a block of code:

For Loop: Iterates over a sequence (e.g., list, range).

While Loop: Repeats a block as long as a condition is true.

Nested loops involve placing one loop inside another, enabling multi-level iteration (e.g., matrix operations).

CODE:

```
# WAP to take input from a user and then check whether it is a number or a character.
# If it is a char, determine whether it is Upper case or lower case

inp = input("Enter the input: ")
''' USING IN-BUILT LIBRARIES '''
print()
print("*** USING IN-BUILT LIBRARIES ***")
if (inp.isalpha()):
    print("It's a Char")
    if inp.isupper():
        print("and in upper case")
    elif inp.islower():
        print("and in lower case")
    else:
        print("and has both cases")
elif(inp.isnumeric()):
    print("It's a number")
else:
    print("Invalid Input")


''' ALTERNATE APPROACH '''
print()
print("*** USING CODE ***")
l1 = [0,0,0] #It will have 3 elements. First is No. of upper case char, second is no. of lower case chars, third is no. of integers
len1 = len(inp)
flag = 0
for i in inp:
    in_ascii = ord(i)
    if in_ascii in range(65,91) or in_ascii in range(97, 123):
        flag = 1
        if in_ascii in range(65,91):
            l1[0] +=1
```

```
        else:
            l1[1] +=1
    elif in_ascii in range(48, 58):
        flag = 2
        l1[2] +=1
if flag == 1:
    if l1[0] == len1:
        print("It's a Char")
        print("and in upper case")
    elif l1[1] == len1:
        print("It's a Char")
        print("and in lower case")
    elif l1[0]+l1[1] == len1:
        print("It's a Char")
        print("and has both cases")
    else:
        print("Invalid Input")
elif flag == 2:
    if l1[2] == len1:
        print("It's a number")
    else:
        print("Invalid Input")
else:
        print("Invalid Input")

# WAP that displays the user to enter a number between 1 to 7 and then displays the corr day
of the week
print("*** Program that displays the user to enter a number between 1 to 7 and then displays
the corr day of the week ***")
num = int(input("Enter the number: "))
if num >= 1 and num <= 7:
    if num == 1:
        print ("Monday")
    if num == 2:
        print ("Tuesday")
    if num == 3:
        print ("Wednesday")
    if num == 4:
        print ("Thursday")
    if num == 5:
        print ("Friday")
    if num == 6:
        print ("Saturday")
    if num == 7:
        print ("Sunday")
else:
    print("Incorrect number")
```

```
# Nested loop to print pattern
for i in range(1,6):
    for j in range(1, i+1):
        print("*", end = " ")
    print()


# While loop inside for loop
names = ["Kelly", "Jessa", "Emma"]
for name in names:
    count = 0
    while(count<5):
        print(name, end=' ')
        count+=1
    print()


# WAP to print the pattern
for i in range(1, 6):
    for k in range(1, 6-i):
        print(" ", end=" ")
    for j in range(1,i+1):
        print(i, " ", end=" ")

    print()

# Alternate approach
n=5
for i in range(1, n+1):
    for k in range(n, i, -1):
        print(" ", end=" ")
    for j in range(1,i+1):
        print(i, " ", end=" ")
    print()

# Calculating factorial
fact = 1
for i in range(2,n+1):
    fact *= i
print("Factorial is: ", fact)


# WAP that displays all leap years from 1900 to 2101
year = int(input("Enter the year (1900-2101) to check whether leap year: "))
if year%100 == 0:
    if year%400 == 0:
```

```
        print("Leap year")
    else:
        print("Not leap year")
else:
    if year%4 == 0:
        print("Leap year")
    else:
        print("Not leap year")
```

```
# WAP to sum the series numbers - 1 + 1/2 + ... + 1/n using for loop
n = int(input("Enter the number: "))
s = 0
for i in range(1, n+1):
    s += (1/i)
print("Sum of series is: ", s)
```

RESULTS:

[6]:
```
# WAP to take input from a user and then check whether it is a number or a character.
# If it is a char, determine whether it is Upper case or lower case

inp = input("Enter the input: ")
''' USING IN-BUILT LIBRARIES '''
print()
print("*** USING IN-BUILT LIBRARIES ***")
if (inp.isalpha()):
    print("It's a Char")
    if inp.isupper():
        print("and in upper case")
    elif inp.islower():
        print("and in lower case")
    else:
        print("and has both cases")
elif(inp.isnumeric()):
    print("It's a number")
else:
    print("Invalid Input")
```

```
Enter the input:  B

*** USING IN-BUILT LIBRARIES ***
It's a Char
and in upper case
```

```
''' ALTERNATE APPROACH '''
print()
print("*** USING CODE ***")
l1 = [0,0,0] #It will have 3 elements. First is No. of upper case char, second is no. of lower case chars, third is no. of integers
len1 = len(inp)
flag = 0
for i in inp:
    in_ascii = ord(i)
    if in_ascii in range(65,91) or in_ascii in range(97, 123):
        flag = 1
        if in_ascii in range(65,91):
            l1[0] +=1
        else:
            l1[1] +=1
    elif in_ascii in range(48, 58):
        flag = 2
        l1[2] +=1
if flag == 1:
    if l1[0] == len1:
        print("It's a Char")
        print("and in upper case")
    elif l1[1] == len1:
        print("It's a Char")
        print("and in lower case")
    elif l1[0]+l1[1] == len1:
        print("It's a Char")
        print("and has both cases")
    else:
        print("Invalid Input")
elif flag == 2:
    if l1[2] == len1:
        print("It's a number")
    else:
        print("Invalid Input")
else:
        print("Invalid Input")
```

```
*** USING CODE ***
It's a Char
and in upper case
```

[10]:
```
# WAP that displays the user to enter a number between 1 to 7 and then displays the corr day of the week
print("*** Program that displays the user to enter a number between 1 to 7 and then displays the corr day of the week ***")
num = int(input("Enter the number: "))
if num >= 1 and num <= 7:
    if num == 1:
        print ("Monday")
    if num == 2:
        print ("Tuesday")
    if num == 3:
        print ("Wednesday")
    if num == 4:
        print ("Thursday")
    if num == 5:
        print ("Friday")
    if num == 6:
        print ("Saturday")
    if num == 7:
        print ("Sunday")
else:
    print("Incorrect number")
```

```
*** Program that displays the user to enter a number between 1 to 7 and then displays the corr day of the week ***
Enter the number:  1
Monday
```

```
# Nested loop to print pattern
for i in range(1,6):
    for j in range(1, i+1):
        print("*", end = " ")
    print()
```

```
*
*  *
*  *  *
*  *  *  *
*  *  *  *  *
```

```
# While loop inside for loop
names = ["Kelly", "Jessa", "Emma"]
for name in names:
    count = 0
    while(count<5):
        print(name, end=' ')
        count+=1
    print()
```

```
Kelly Kelly Kelly Kelly Kelly
Jessa Jessa Jessa Jessa Jessa
Emma Emma Emma Emma Emma
```

```
# WAP to print the pattern
for i in range(1, 6):
    for k in range(1, 6-i):
        print(" ", end=" ")
    for j in range(1,i+1):
        print(i, " ", end=" ")

    print()
```

```
    1
  2   2
3   3   3
4   4   4   4
5   5   5   5   5
```

```
# Alternate approach
n=5
for i in range(1, n+1):
    for k in range(n, i, -1):
        print(" ", end=" ")
    for j in range(1,i+1):
        print(i, " ", end=" ")
    print()
```

```
    1
  2   2
3   3   3
4   4   4   4
5   5   5   5   5
```

```python
# Calculating factorial
fact = 1
for i in range(2,n+1):
    fact *= i
print("Factorial is: ", fact)
```

```
Factorial is:  120
```

```python
# WAP that displays all leap years from 1900 to 2101
year = int(input("Enter the year (1900-2101) to check whether leap year: "))
if year%100 == 0:
    if year%400 == 0:
        print("Leap year")
    else:
        print("Not leap year")
else:
    if year%4 == 0:
        print("Leap year")
    else:
        print("Not leap year")
```

```
Enter the year (1900-2101) to check whether leap year:  2100
Not leap year
```

```python
# WAP to sum the series numbers - 1 + 1/2 + ... + 1/n using for loop
n = int(input("Enter the number: "))
s = 0
for i in range(1, n+1):
    s += (1/i)
print("Sum of series is: ", s)
```

```
Enter the number:  5
Sum of series is:  2.283333333333333
```

EXPERIMENT 4

OBJECTIVE: Demonstrate list operations and develop code for given problem statements:

1. Demonstrate list slicing and list cloning
2. Demonstrate use of list methods- insert, append, extend, reverse, reversed, remove, pop
3. List comprehension
4. Looping in lists
5. WAP to print index of values in a list
6. Sum and average of elements in list

THEORY:

Lists in Python

A list is an ordered, mutable collection of items in Python. It is widely used for storing sequences of data, and its functionality enables efficient manipulation and traversal of elements.

List Slicing and Cloning

Slicing allows extracting parts of a list using **list[start:end:step].**

Cloning creates an independent copy of the list using slicing: **cloned_list = original_list[:].**

List Methods

Python provides a variety of built-in methods to modify and manipulate lists, such as:

- **insert(index, value)**: Adds a value at a specific position.
- **append(value)**: Adds a value to the end of the list.
- **extend(iterable)**: Extends the list by appending elements from another iterable.
- **reverse()**: Reverses the list in place.
- **reversed()**: Returns a reversed iterator (does not modify the original list).
- **remove(value)**: Removes the first occurrence of the specified value.
- **pop(index)**: Removes and returns the element at a given index (default is the last).

List Comprehension

A compact way to create or modify lists using syntax: `[expression for item in iterable if condition]`

Looping in Lists

Use loops (e.g., for or while) to iterate over lists for processing or data transformation.

CODE:

```
# List slicing

list1 = ['physics', 'chem', 1997, 2000]
```

```
list2 = [1,2,3,4,5,6,7,8]
print(list2[1:5])

# List methods- insert, append, extend, reverse, reversed, remove, pop, slicing,
List = ['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']
print(List)
Sliced_list = List[:-6]
print("Sliced: ", Sliced_list)
l2 = List[-6:-1]
print(l2)
l3 = List[::-1]
print(l3)

# List Comprehension
# Syntax - [expression(element) for element in oddList if condition]
l1 = [x**2 for x in range(1,11) if x%2 == 1]
print(l1)

# List Comprehension
# Syntax - [expression(element) for element in oddList if condition]
l1 = [x**2 for x in range(1,11) if x%2 == 1]
print(l1)

# Looping in lists
ls = [1,'a',"abc",[2,3,4,5],8.9]
i = 0
while i < (len(ls)):
    print(ls[i])
    i+=1
# Program to print index of values in a list
l1 = [1,2,3,4,5]
for i in range(len(l1)):
```

```
    print("index: ", i)


# Sum and average of list items
l1 = [1,2,3,4,5,6,7,8,9,10]
s = 0
for i in l1:
    s+=i
print("Sum = ", s)
print("Avg = ", s/len(l1))
```

RESULTS:

1. Demonstrate list slicing and list cloning

```
# List slicing
list1 = ['physics', 'chem', 1997, 2000]
list2 = [1,2,3,4,5,6,7,8]
print(list2[1:5])
```

```
[2, 3, 4, 5]
```

2. Demonstrate use of list methods- insert, append, extend, reverse, reversed, remove,pop

```
# List methods- insert, append, extend, reverse, reversed, remove, pop, slicing,
List = ['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']
print(List)
Sliced_list = List[:-6]
print("Sliced: ", Sliced_list)
l2 = List[-6:-1]
print(l2)
l3 = List[::-1]
print(l3)
```

```
['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']
Sliced:  ['G', 'E', 'E', 'K', 'S', 'F', 'O']
['R', 'G', 'E', 'E', 'K']
['S', 'K', 'E', 'E', 'G', 'R', 'O', 'F', 'S', 'K', 'E', 'E', 'G']
```

3. List comprehension

```
# List Comprehension
# Syntax - [expression(element) for element in oddList if condition]
l1 = [x**2 for x in range(1,11) if x%2 == 1]
print(l1)

# List Comprehension
# Syntax - [expression(element) for element in oddList if condition]
l1 = [x**2 for x in range(1,11) if x%2 == 1]
print(l1)
```

```
[1, 9, 25, 49, 81]
[1, 9, 25, 49, 81]
```

4. Looping in lists

```
# Looping in lists
ls = [1,'a',"abc",[2,3,4,5],8.9]
i = 0
while i < (len(ls)):
    print(ls[i])
    i+=1
```

```
1
a
abc
[2, 3, 4, 5]
8.9
```

5. WAP to print index of values in a list

```
# Program to print index of values in a list
l1 = [1,2,3,4,5]
for i in range(len(l1)):
    print("index: ", i)
```

```
index:  0
index:  1
index:  2
index:  3
index:  4
```

6. Sum and average of elements in list

```
# Sum and average of list items
l1 = [1,2,3,4,5,6,7,8,9,10]
s = 0
for i in l1:
    s+=i
print("Sum = ", s)
print("Avg = ", s/len(l1))
```

```
Sum =  55
Avg =  5.5
```

EXPERIMENT 5

OBJECTIVE: Demonstrate arrays and tuples and develop code for given problem statements:

1. Operations in array - Create array in python, Demonstrate functions in arrays - insert(), append(), Slicing in array, updating elements in array
2. Create an empty tuple, create tuple using string, create tuple using list, and create a tuple with mixed datatypes
3. Write a program to demonstrate use of nested tuples. Also, WAP that has a nested list to store toppers details. Edit the details and reprint the details.
4. Creating a tuple using Loop
5. WAP to swap two values using tuple assignment
6. WAP using a function that returns the area and circumference of a circle whose radius is passed as an argument
7. WAP that scans an email address and forms a tuple of username and domain

THEORY:

Arrays in Python

An array is a collection of elements that are stored in a sequential manner. Arrays are commonly used when you need to perform operations on a large set of data efficiently.

In Python, arrays can be created using the array module. Arrays differ from lists in that they can only store elements of the same type.

Tuples in Python

A tuple is a collection of objects that are ordered and immutable. Once created, the elements in a tuple cannot be modified.

Tuples are commonly used when you need a collection of items that should not be altered.

CODE:

```
#1. Creating array in python
import array as arr
a = arr.array('i', [1,2,3])
print(a)
for i in range(0,3):
    print(a[i], end=" ")

# Demonstrate the functions in arrays like insert(), append()
a = arr.array('i', [1,2,3])
print("Array of integers (Before): ", a)
a.insert(1,4)
```

```
print("Array of integers (After Inserting): ",a)
b = arr.array('d', [1,2,3])
print("Array of floats (Before): ", b)
b.append(4.4)
print("Array of floats (After appending): ", b)

# Slicing
import array as arr
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
a = arr.array('i', l)
print("Initial Array: ")
for i in (a):
    print(i, end = " ")
sliced_array = a[3:8]
print("\nSlicing elements in a range 3-8: ")
print(sliced_array)
sliced_array = a[5:]
print("\nElements sliced from 5th element till the end: ")
print(sliced_array)
sliced_array=a[:]
print("\nPrinting all elements using slice operartion: ")
print(sliced_array)

# Array Updation
import array
arr = array.array('i', [1,2,3,1,2,5])
for i in range(0,6):
    print(arr[i], end = " ")
print("\nAfter updation")
arr[2]=6
for i in range(0,6):
    print(arr[i], end=" ")

# Create empty tuple:
tuple1 = ()
print(tuple1)

# Create tuple using string:
tuple1 = ('Hello', 'Sam')
print(tuple1)

# Create tuple using list:
list1 = ['Hello', 'Sam']
print(tuple(list1))

# Create a tuple using built-in function:
tuple1 = tuple('Sam')
```

```
print(tuple1)

# Creating a tuple with mixed datatypes
tuple1 = (5, 'aiojdio', 7, 'JFidsof')
print(tuple1)

# Nested tuples

t1 = (1,2,3)
t2 = ('a', 'b', 'c')
t3 = (t1, t2)
print(t3)

# Program to demonstrate use of nested tuples

Toppers = (("arav", 97, "B.Sc."), ("raghav", 87, "BCA"))
for i in Toppers:
    print(i)

# WAP that has a nested list to store toppers details. Edit the details and reprint the details.
# Eg - l1 = ["Arav", "MSC", 92]

l1 = [["Arav", "MSC", 92], ["Student2", "MBA", 99], ["Student3", "MTech", 94],
["Student4", "BSC", 95]]

print("The original list of toppers is: ", l1)
print("Enter the metadata you wish to edit: ")
print("\nChoose the name of the student you wish to edit the details for. Press")
for i in range(len(l1)):
    print(f'{i}. To edit the details of student {l1[i][0]}')
ch1 = int(input("Enter your choice: "))

print("Press\n1. To edit the name\n2. To edit the branch\n3. To edit the marks")
ch2 = int(input("Enter your choice (1/2/3): "))

if ch1 not in range(len(l1)):
    print("Wrong Student index chosen!")
else:

    if ch2 == 1:
        new_name = input("Enter the new name: ")
        l1[ch1][0] = new_name
    elif ch2 == 1:
        new_name = input("Enter the new branch: ")
        l1[ch1][1] = new_name
    elif ch2 == 1:
        new_name = input("Enter the new marks: ")
```

```
        l1[ch1][2] = new_name
    else:
        print("Wrong choice entered!")

    print("New list is: ", l1)


# Creating a tuple using Loop
t1 = ('Sam')
n = 5
for i in range(int(n)):
    t1 = (t1,)
    print(t1)

# WAP to swap two values using tuple assignment
t1 = (2,3)
print("Tuple is: ", t1)
print("Before swap: ")
a, b = t1
print(f'Value of a is {a} and value of b is {b}')
print("After swap: ")
(a, b) = (b, a)
print(f'Value of a is {a} and value of b is {b}')


# WAP using a function that returns the area and circumference of a circle whose radius is passed as an argument
import math
def func1(r):
    area = math.pi * r * r
    circum = 2 *math.pi *r
    return (area, circum)
rad = int(input("Enter radius: "))
(ar, circum) = func1(rad)
print("Area is: ", ar)
print("Circumference is: ", circum)

# WAP that scans an email address and forms a tuple of username and domain
email = input("Enter the email address: ")
email = email.split("@")
email_tuple = tuple(email)
print(email_tuple)
```

RESULTS:1.a

1. Operations in array - Create array in python, Demonstrate functions in arrays - insert(), append(), Slicing in array, updating elements in array

[2]:
```
# Creating array in python
import array as arr
a = arr.array('i', [1,2,3])
print(a)
for i in range(0,3):
    print(a[i], end=" ")
```

```
array('i', [1, 2, 3])
1 2 3
```

[4]:
```
# Demonstrate the functions in arrays like insert(), append()
a = arr.array('i', [1,2,3])
print("Array of integers (Before): ", a)
a.insert(1,4)
print("Array of integers (After Inserting): ",a)
b = arr.array('d', [1,2,3])
print("Array of floats (Before): ", b)
b.append(4.4)
print("Array of floats (After appending): ", b)
```

```
Array of integers (Before):  array('i', [1, 2, 3])
Array of integers (After Inserting):  array('i', [1, 4, 2, 3])
Array of floats (Before):  array('d', [1.0, 2.0, 3.0])
Array of floats (After appending):  array('d', [1.0, 2.0, 3.0, 4.4])
```

1.b

```
print("\nSlicing elements in a range 3-8: ")
print(sliced_array)
sliced_array = a[5:]
print("\nElements sliced from 5th element till the end: ")
print(sliced_array)
sliced_array=a[:]
print("\nPrinting all elements using slice operartion: ")
print(sliced_array)

# Array Updation
import array
arr = array.array('i', [1,2,3,1,2,5])
for i in range(0,6):
    print(arr[i], end = " ")
print("\nAfter updation")
arr[2]=6
for i in range(0,6):
    print(arr[i], end=" ")
```

```
Initial Array:
1 2 3 4 5 6 7 8 9 10
Slicing elements in a range 3-8:
array('i', [4, 5, 6, 7, 8])

Elements sliced from 5th element till the end:
array('i', [6, 7, 8, 9, 10])

Printing all elements using slice operartion:
array('i', [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
1 2 3 1 2 5
After updation
1 2 6 1 2 5
```

2. Create an empty tuple, create tuple using string, create tuple using list, and create a tuple with mixed datatypes

```
print(tuple1)

# Create tuple using string:
tuple1 = ('Hello', 'Sam')
print(tuple1)

# Create tuple using list:
list1 = ['Hello', 'Sam']
print(tuple(list1))

# Create a tuple using built-in function:
tuple1 = tuple('Sam')
print(tuple1)

# Creating a tuple with mixed datatypes
tuple1 = (5, 'aiojdio', 7, 'JFidsof')
print(tuple1)

# Nested tuples

t1 = (1,2,3)
t2 = ('a', 'b', 'c')
t3 = (t1, t2)
print(t3)
```

```
()
('Hello', 'Sam')
('Hello', 'Sam')
('S', 'a', 'm')
(5, 'aiojdio', 7, 'JFidsof')
((1, 2, 3), ('a', 'b', 'c'))
```

3. Write a program to demonstrate use of nested tuples. Also, WAP that has a nested list to store toppers details. Edit the details and reprint the details.

```
        new_name = input("Enter the new name: ")
        l1[ch1][0] = new_name
    elif ch2 == 1:
        new_name = input("Enter the new branch: ")
        l1[ch1][1] = new_name
    elif ch2 == 1:
        new_name = input("Enter the new marks: ")
        l1[ch1][2] = new_name
    else:
        print("Wrong choice entered!")

    print("New list is: ", l1)
```

```
('arav', 97, 'B.Sc.')
('raghav', 87, 'BCA')
The original list of toppers is:  [['Arav', 'MSC', 92], ['Student2', 'MBA', 99], ['Student3', 'MTech', 94], ['Student4', 'BSC', 95]]
Enter the metadata you wish to edit:

Choose the name of the student you wish to edit the details for. Press
0. To edit the details of student Arav
1. To edit the details of student Student2
2. To edit the details of student Student3
3. To edit the details of student Student4
Enter your choice:  0
Press
1. To edit the name
2. To edit the branch
3. To edit the marks
Enter your choice (1/2/3):  1
Enter the new name:  Shubham
New list is:  [['Shubham', 'MSC', 92], ['Student2', 'MBA', 99], ['Student3', 'MTech', 94], ['Student4', 'BSC', 95]]
```

4.&5. Creating a tuple using Loop & WAP to swap two values using tuple assignment

[12]:
```
# Creating a tuple using Loop
t1 = ('Sam')
n = 5
for i in range(int(n)):
    t1 = (t1,)
    print(t1)
```

```
('Sam',)
(('Sam',),)
((('Sam',),),)
((((('Sam',),),),),)
(((((('Sam',),),),),),)
```

[14]:
```
# WAP to swap two values using tuple assignment
t1 = (2,3)
print("Tuple is: ", t1)
print("Before swap: ")
a, b = t1
print(f'Value of a is {a} and value of b is {b}')
print("After swap: ")
(a, b) = (b, a)
print(f'Value of a is {a} and value of b is {b}')
```

```
Tuple is:  (2, 3)
Before swap:
Value of a is 2 and value of b is 3
After swap:
Value of a is 3 and value of b is 2
```

6.&7. WAP using a function that returns the area and circumference of a circle whose radius is passed as an argument

WAP that scans an email address and forms a tuple of username and domain

```
# WAP using a function that returns the area and circumference of a circle whose radius is passed as an argument
import math
def func1(r):
    area = math.pi * r * r
    circum = 2 *math.pi *r
    return (area, circum)
rad = int(input("Enter radius: "))
(ar, circum) = func1(rad)
print("Area is: ", ar)
print("Circumference is: ", circum)
```

```
Enter radius:  4
Area is:  50.26548245743669
Circumference is:  25.132741228718345
```

```
# WAP that scans an email address and forms a tuple of username and domain
email = input("Enter the email address: ")
email = email.split("@")
email_tuple = tuple(email)
print(email_tuple)
```

```
Enter the email address:  satyarthas.24.ai@nitj.ac.in
('satyarthas.24.ai', 'nitj.ac.in')
```

EXPERIMENT 6

OBJECTIVE: Demonstrate functions and modules and develop code for given problem statements:

1. Create a function to return the square of the number
2. Demonstrate Pass by Reference and Pass by value
3. WAP that subtracts two numbers using a function
4. WAP using functions and return statements to check whether a number is even or odd
5. WAP to calculate simple interest. Suppose the customer is a Senior citizen and is being offered 12% ROI. For all other customers, ROI is 10%.
6. Program to find certain power of a number using recursion

THEORY:

- **Functions**: Functions in Python are blocks of code that perform a specific task. They are defined using the `def` keyword. Functions can take parameters, return values, and be called multiple times to avoid code repetition. Python supports recursion, where a function calls itself.

- **Modules**: A module is a file containing Python code that can define functions, variables, and classes. It helps organize code into separate namespaces. Modules can be built-in (e.g., `math`, `random`) or custom (created by the user). You can import entire modules or specific functions from them using the `import` statement.

CODE:

```
# Defining the function
def square(num):
#     Returns the square of the number
    return num**2

obj = square(6)
print(obj)

# Pass by Reference and Pass by value

def square(item_list):
#     Returns the square of the number
    squares = []
    for i in item_list:
        squares.append(i**2)
    return squares

# Pass by reference
num = [1,2,3,4,5]
```

```
obj = square(num)
print(obj)

# Pass by value
obj = square([1,2,3,4,5])
print(obj)

# WAP that subtracts two numbers using a function
def func(a,b):
    return a - b
a = int(input("Enter num1: "))
b =  int(input("Enter num1: "))
print("num1 - num2 = ", func(a,b))

# WAP using functions and return statements to check whether a number is even or odd
def func(a):
    if (a%2 == 0):
        return "Even"
    else:
        return "Odd"
a = int(input("Enter num1: "))
print("Number is", func(a))

# WAP to calculate simple interest.
# Suppose the customer is a Senior citizen and is being offered 12% ROI. For all other customers, ROI is 10%.
age = int(input("Enter age of person: "))
principal = float(input("Enter principal amount: "))
time = int(input("Enter time in years: "))
if age>=60:
    r=12
else:
    r=10
si = principal*r*time/100
print("Simple Interest is: ", si)

# Program to find certain power of a number using recursion
def func1(n,i):
    if i == 0:
        return 1
    else:
        return n*func1(n,i-1)
func1(2,6)
```

RESULTS:

1. Create a function to return the square of the number

```
# Defining the function
def square(num):
#     Returns the square of the number
    return num**2

obj = square(6)
print(obj)
```

```
36
```

2. Demonstrate Pass by Reference and Pass by value

```
# Pass by Reference and Pass by value

def square(item_list):
#     Returns the square of the number
    squares = []
    for i in item_list:
        squares.append(i**2)
    return squares

# Pass by reference
num = [1,2,3,4,5]
obj = square(num)
print(obj)

# Pass by value
obj = square([1,2,3,4,5])
print(obj)
```

```
[1, 4, 9, 16, 25]
[1, 4, 9, 16, 25]
```

3. WAP that subtracts two numbers using a function

```
# WAP that subtracts two numbers using a function
def func(a,b):
    return a - b
a = int(input("Enter num1: "))
b =  int(input("Enter num1: "))
print("num1 - num2 = ", func(a,b))
```

```
Enter num1:  5
Enter num1:  2
num1 - num2 =  3
```

4. WAP using functions and return statements to check whether a number is even or odd

```
# WAP using functions and return statements to check whether a number is even or odd
def func(a):
    if (a%2 == 0):
        return "Even"
    else:
        return "Odd"
a = int(input("Enter num: "))
print("Number is", func(a))
```

```
Enter num:  56
Number is Even
```

5. WAP to calculate simple interest. Suppose the customer is a Senior citizen and is being offered 12% ROI. For all other customers, ROI is 10%.

```
# WAP to calculate simple interest.
# Suppose the customer is a Senior citizen and is being offered 12% ROI. For all other customers, ROI is 10%.
age = int(input("Enter age of person: "))
principal = float(input("Enter principal amount: "))
time = int(input("Enter time in years: "))
if age>=60:
    r=12
else:
    r=10
si = principal*r*time/100
print("Simple Interest is: ", si)
```

```
Enter age of person:  48
Enter principal amount:  10000
Enter time in years:  5
Simple Interest is:  5000.0
```

6. Program to find certain power of a number using recursion

```
# Program to find certain power of a number using recursion
def func1(n,i):
    if i == 0:
        return 1
    else:
        return n*func1(n,i-1)
func1(2,6)
```

```
64
```

EXPERIMENT 7

OBJECTIVE: Demonstrate Set operations and develop code for given problem statements:

1. Set Operations - Create set, Add items in set, Add items from another set into this set, Add elements of a list to the set, Remove item, Remove item using discard()
2. WAP that creates 2 sets squares and cubes in range 1 to 10. Demonstrate the use of update, pop, remove and clear function
3. WAP that creates two sets one of even numbers in the range 1 to 10 and the other as all composite numbers in range 1 to 20. Demonstrate the use of all(), issuperset(), len() and sum() on the sets.

THEORY: **Sets in Python**

A set in Python is an unordered collection of unique elements. Sets are useful when you need to store multiple items, but the order of those items doesn't matter, and you want to ensure that no duplicates are allowed.

Key Features of Sets:

Unordered: The elements in a set do not maintain any specific order.

Unique Elements: A set automatically removes duplicate items.

Mutable: You can add or remove elements from a set after its creation.

No Indexing: Sets do not support indexing, slicing, or other sequence-like behavior

CODE:

```
# SETS

thisset = {"apple", "banana", "cherry"}
print(type(thisset))
print("banana" in thisset)

# Add items in set
thisset.add("orange")
print(thisset)

# Add items from another set into this set
tropical = {"mango", "papaya"}
thisset.update(tropical)
print(thisset)

# Add elements of a list to the set
l1 = ["mango2", "papaya2"]
thisset.update(l1)
```

```
print(thisset)

# Remove item
thisset.remove("mango2")
print(thisset)

# Remove item using discard()
thisset.discard("banana")
print(thisset)

# WAP that creates 2 sets squares and cubes in range 1 to 10. Demonstrate the use of update, pop, remove and clear function

set1 = set()
set2 = set()
for i in range(1, 11):
    set1.add(i*i)
    set2.add(i*i*i)
print("Set1 after adding squares: ", set1)
print("Set2 after adding cubes: ", set2)

print("\nDemonstrating the use of update function: ")
set3 = {"mango"}
set1.update(set3)
print("Set1 after update: ", set1)

print("\nDemonstrating the use of pop function: ")
print(set1.pop())

print("\nDemonstrating the use of remove function: ")
set1.remove("mango")
print(set1)

print("\nDemonstrating the use of clear function: ")
set1.clear()
print(set1)


# WAP that creates two sets one of even numbers in the range 1 to 10 and the other as all composite numbers in range 1 to 20
# Demonstrate the use of all(), issuperset(), len() and sum() on the sets.

set1 = {i for i in range(1, 11) if i % 2 == 0 }
print("Set of even numbers: ",set1)

set2 = set()
```

```
c = 0
for i in range(2, 21):
    for j in range(2, i):
        if i%j ==0:
            c+=1
    if c!=0:
        set2.add(i)
    c = 0
print("Set of composite numbers: ", set2)

# all() function returns True if all elements are True, else returns False
print("\nDemonstrating use of all() function: ")
print(all(set1))

set1.remove(2)
print("\nRemoving '2' from set1: ", set1)

print("\nDemonstrating use of issuperset() function: ")
print(set2.issuperset(set1))

print("\nDemonstrating use of len() function: ")
print(len(set2))

print("\nDemonstrating use of sum() function: ")
print("Sum of elements of set1: ", sum(set1))
```

RESULTS:

1. Set Operations - Create set, Add items in set, Add items from another set into this set, Add elements of a list to the set, Remove item, Remove item using discard()

```
# SETS
thisset = {"apple", "banana", "cherry"}
print(type(thisset))
print("banana" in thisset)
# Add items in set
thisset.add("orange")
print(thisset)
# Add items from another set into this set
tropical = {"mango", "papaya"}
thisset.update(tropical)
print(thisset)
# Add elements of a list to the set
l1 = ["mango2", "papaya2"]
thisset.update(l1)
print(thisset)
# Remove item
thisset.remove("mango2")
print(thisset)
# Remove item using discard()
thisset.discard("banana")
print(thisset)
```

```
<class 'set'>
True
{'orange', 'apple', 'cherry', 'banana'}
{'orange', 'apple', 'papaya', 'mango', 'cherry', 'banana'}
{'orange', 'apple', 'papaya2', 'papaya', 'mango', 'cherry', 'mango2', 'banana'}
{'orange', 'apple', 'papaya2', 'papaya', 'mango', 'cherry', 'banana'}
{'orange', 'apple', 'papaya2', 'papaya', 'mango', 'cherry'}
```

2.WAP that creates 2 sets squares and cubes in range 1 to 10. Demonstrate the use of update, pop, remove and clear function

```
# WAP that creates 2 sets squares and cubes in range 1 to 10. Demonstrate the use of update, pop, remove and clear function
set1 = set()
set2 = set()
for i in range(1, 11):
    set1.add(i*i)
    set2.add(i*i*i)
print("Set1 after adding squares: ", set1)
print("Set2 after adding cubes: ", set2)

print("\nDemonstrating the use of update function: ")
set3 = {"mango"}
set1.update(set3)
print("Set1 after update: ", set1)

print("\nDemonstrating the use of pop function: ")
print(set1.pop())

print("\nDemonstrating the use of remove function: ")
set1.remove("mango")
print(set1)

print("\nDemonstrating the use of clear function: ")
set1.clear()
print(set1)
```

```
Set1 after adding squares:  {64, 1, 4, 36, 100, 9, 16, 49, 81, 25}
Set2 after adding cubes:  {64, 1, 512, 8, 1000, 343, 216, 729, 27, 125}

Demonstrating the use of update function:
Set1 after update:  {64, 1, 4, 36, 100, 9, 16, 49, 81, 'mango', 25}

Demonstrating the use of pop function:
64

Demonstrating the use of remove function:
{1, 4, 36, 100, 9, 16, 49, 81, 25}

Demonstrating the use of clear function:
set()
```

3.WAP that creates two sets one of even numbers in the range 1 to 10 and the other as all composite numbers in range 1 to 20. Demonstrate the use of all(), issuperset(), len() and sum() on the sets.

```
# Demonstrate the use of all(), issuperset(), len() and sum() on the sets.

set1 = {i for i in range(1, 11) if i % 2 == 0 }
print("Set of even numbers: ",set1)

set2 = set()

c = 0
for i in range(2, 21):
    for j in range(2, i):
        if i%j ==0:
            c+=1
    if c!=0:
        set2.add(i)
    c = 0
print("Set of composite numbers: ", set2)

# all() function returns True if all elements are True, else returns False
print("\nDemonstrating use of all() function: ")
print(all(set1))

set1.remove(2)
print("\nRemoving '2' from set1: ", set1)

print("\nDemonstrating use of issuperset() function: ")
print(set2.issuperset(set1))

print("\nDemonstrating use of len() function: ")
print(len(set2))

print("\nDemonstrating use of sum() function: ")
print("Sum of elements of set1: ", sum(set1))
```

```
Set of even numbers:  {2, 4, 6, 8, 10}
Set of composite numbers:  {4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20}

Demonstrating use of all() function:
True

Removing '2' from set1:  {4, 6, 8, 10}

Demonstrating use of issuperset() function:
True

Demonstrating use of len() function:
11

Demonstrating use of sum() function:
Sum of elements of set1:  28
```

EXPERIMENT 8

OBJECTIVE: Demonstrate dictionary operations and develop code for given problem statements:

1. Dictionary Operations –
    a. Accessing values in a Dictionary, Updating a dict, adding new values, Delete particular entries, Clear whole dict, Delete whole dict
    b. Dictionary methods – len(), copy(), dictionary to string, Fromkeys(), get(), items(), setdefault(), Update(), values()
2. WAP to merge two dictionaries with a third one
3. Iterating through a dictionary
4. WAP to Sort dictionary by values

THEORY:

A dictionary in Python is an unordered collection of key-value pairs. Each key in a dictionary is unique, and it maps to a value. Dictionaries are mutable, which means you can add, update, or remove elements after the dictionary is created.

Key Features of Dictionaries:

- Unordered: The key-value pairs are not stored in any specific order.
- Key-Value Pairs: Each element is a pair consisting of a key (unique) and a corresponding value.
- Mutable: You can change the dictionary by adding, updating, or removing key-value pairs.
- Indexed by Keys: You access values using their corresponding keys, not numeric indices.

CODE:

```
# Accessing values in a Dictionary
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print(dict1['Name'])
print(dict1['Age'])

# Updating a dict
dict1['Age'] = 8
print(dict1)

# Add a new entry
dict1['School'] = 'DPS'
print(dict1)

# Delete entries
del dict1['Name']
```

```
print(dict1)

# Clear whole dict
dict1.clear()
print(dict1)

# Delete whole dict
del dict1
print(dict1)

# WAP to merge two dictionaries with a third one
a = {'Name': 'Zara', 'Age': 10}
b = {'Gender': 'Female'}
c = {'Senior_Citizen': 'No'}
c.update(b)
c.update(a)
print(c)

# Iterating through a dictionary
dict1 = {"a": "time", "b": "money", "c": "value"}
for key, values in dict1.items():
    print(key, " ", values)
print()
for i in dict1.keys():
    print(i)
for i in dict1.values():
    print(i)

# Sort dictionary by values
dict1 = {"a": 23, "b": 91038, "c": 1, "d": 20, "e": 55}
# print(sorted(dict1, key = dict1.values))
print(dict1)
ls = sorted(dict1.values())
print(ls)
dict2 = {}
for i in ls:
    for j in dict1.keys():
        if dict1.get(j) == i:
            dict2[j] = i
print(dict2)
```

RESULTS:

1. Dictionary Operations –
    a. Accessing values in a Dictionary, Updating a dict, adding new values, Delete particular entries, Clear whole dict, Delete whole dict

```
# Accessing values in a Dictionary
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print(dict1['Name'])
print(dict1['Age'])

# Updating a dict
dict1['Age'] = 8
print(dict1)

# Add a new entry
dict1['School'] = 'DPS'
print(dict1)

# Delete entries
del dict1['Name']
print(dict1)

# Clear whole dict
dict1.clear()
print(dict1)

# Delete whole dict
del dict1
print(dict1)
```

```
Zara
7
{'Name': 'Zara', 'Age': 8, 'Class': 'First'}
{'Name': 'Zara', 'Age': 8, 'Class': 'First', 'School': 'DPS'}
{'Age': 8, 'Class': 'First', 'School': 'DPS'}
{}
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[6], line 24
     22 # Delete whole dict
     23 del dict1
---> 24 print(dict1)

NameError: name 'dict1' is not defined
```

    b. Dictionary methods – len(), copy(), dictionary to string, Fromkeys(), get(), items(), setdefault(), Update(), values()

```
# len() - Returns the number of key-value pairs in the dictionary
my_dict = {'a': 1, 'b': 2}
print(len(my_dict))  # Output: 2

# copy() - Creates a shallow copy of the dictionary
new_dict = my_dict.copy()
print(new_dict)  # Output: {'a': 1, 'b': 2}

# Dictionary to String - Convert a dictionary to a string
print(str(my_dict))  # Output: "{'a': 1, 'b': 2}"

# fromkeys() - Creates a new dictionary with keys from an iterable, assigning a default value
keys = ['a', 'b']
new_dict = dict.fromkeys(keys, 'default')
print(new_dict)  # Output: {'a': 'default', 'b': 'default'}

# get() - Returns the value of the key, or a default value if not found
print(my_dict.get('a'))  # Output: 1
print(my_dict.get('c', 'Not Found'))  # Output: 'Not Found'

# items() - Returns a view object of the dictionary's key-value pairs
print(my_dict.items())  # Output: dict_items([('a', 1), ('b', 2)])

# setdefault() - Returns the value of the key, and if not found, inserts it with a default value
print(my_dict.setdefault('c', 3))  # Output: 3
print(my_dict)  # Output: {'a': 1, 'b': 2, 'c': 3}

# update() - Updates the dictionary with key-value pairs from another dictionary or iterable
my_dict.update({'d': 4})
print(my_dict)  # Output: {'a': 1, 'b': 2, 'c': 3, 'd': 4}

# values() - Returns a view object of the dictionary's values
print(my_dict.values())  # Output: dict_values([1, 2, 3, 4])
```

```
2
{'a': 1, 'b': 2}
{'a': 1, 'b': 2}
{'a': 'default', 'b': 'default'}
1
Not Found
dict_items([('a', 1), ('b', 2)])
3
{'a': 1, 'b': 2, 'c': 3}
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
dict_values([1, 2, 3, 4])
```

2. WAP to merge two dictionaries with a third one

```
# WAP to merge two dictionaries with a third one
a = {'Name': 'Zara', 'Age': 10}
b = {'Gender': 'Female'}
c = {'Senior_Citizen': 'No'}
c.update(b)
c.update(a)
print(c)
```

```
{'Senior_Citizen': 'No', 'Gender': 'Female', 'Name': 'Zara', 'Age': 10}
```

3. Iterating through a dictionary

```
# Iterating through a dictionary
dict1 = {"a": "time", "b": "money", "c": "value"}
for key, values in dict1.items():
    print(key, " ", values)
print()
for i in dict1.keys():
    print(i)
for i in dict1.values():
    print(i)
```

```
a    time
b    money
c    value

a
b
c
time
money
value
```

4. WAP to Sort dictionary by values

```
# Sort dictionary by values
dict1 = {"a": 23, "b": 91038, "c": 1, "d": 20, "e": 55}
# print(sorted(dict1, key = dict1.values))
print(dict1)
ls = sorted(dict1.values())
print(ls)
dict2 = {}
for i in ls:
    for j in dict1.keys():
        if dict1.get(j) == i:
            dict2[j] = i
print(dict2)
```

```
{'a': 23, 'b': 91038, 'c': 1, 'd': 20, 'e': 55}
[1, 20, 23, 55, 91038]
{'c': 1, 'd': 20, 'a': 23, 'e': 55, 'b': 91038}
```

EXPERIMENT 9

OBJECTIVE: Demonstrate strings and its related operations and develop code for given problem statements:

1) Slicing – WAP to Get the characters from o in “World” to but not included d in "World"
2) WAP to display powers of number without using formatting characters
3) String methods and functions –
    i. capitalize(), center(), count(), endswith(), startswith(), find(), index(), rfind(), rindex(), isalnum(), isalpha(), isdigit(), islower(), isupper(), len(), etc.
    ii. WAP to print following pattern

        A
        AB
        ABC
        ABCD
        ABCDE
        ABCDEF

    iii. WAP using while loop to iterate a given string
    iv. WAP that encrypts a message by adding a key value to every character
    v. WAP that uses split function to split a multi-line string
    vi. WAP that accepts a string from user and re-displays the same string after removing vowels
4) Regular Expressions
    i. WAP to find patterns that begin with one or more characters followed by space and followed by one or more digits
    ii. WAP that uses a regex to match strings which start with sequence of digits (atleast 1) followed by a blank and after this add arbitrary characters

THEORY:**Strings in Python**

A string is a sequence of characters enclosed within single ('), double (") or triple (''' or """) quotes. Strings in Python are immutable, meaning their content cannot be changed after creation. They are one of the most commonly used data types in Python and are extensively used for text manipulation.

Key Features of Strings

Immutable: Once created, a string cannot be modified.

Indexed: Strings are indexed, allowing individual character access using indices.

Sliceable: Substrings can be obtained using slicing.

Built-in Methods: Python provides many methods for string manipulation, like split(), join(), replace(), etc.

CODE:

```
a = "HelloWorld"
# Get the characters from o in World to but not included d in "World"
print(a[-4:-1])

# WAP to display powers of number without using formatting characters
i=1
while i<=5:
   print(i**1, "\t", i**2, "\t",  i**3, "\t",  i**4)
   i+=1
print()
print()

i=1
while i<=5:
   print("%d\t%d\t%d\t%d"%(i**1, i**2, i**3,  i**4))
   i+=1
print()
print()

i = 1
print("%-4s%-5s%-6s"%('i', 'i**2', 'i**3'))
print()
print()

i = 1
while i<=5:
   print("%-4d%-5d%-6d"%(i, i**2, i**3))
   i+=1


# Built-in string methods and functions
s =  "hello"
print(s.capitalize())

s = "hello"
print(s.center(10, '*'))

msg = 'he'
str1 = "hellohello"
print(str1.count(msg, 0, len(str1)))

msg = "she is my best friend"
print(msg.endswith("end", 0, len(msg)))
```

```
str1 = "the world is beautiful"
print(str1.startswith("th", 0, len(str1)))

msg = "she is my best my friend"
print(msg.find("my", 0, len(msg)))
print(msg.find("mine", 0, len(msg)))

try:
    print(msg.index("mine", 0, len(msg)))
except:
    print("substring not found")

# rfind searches from end
msg = "is this your bag?"
print(msg.rfind("is", 0, len(msg)))

print(msg.rindex("is"))
try:
    print(msg.rindex("z"))
except:
    print("substring not found")

msg = "jamesbond007"
print(msg.isalnum())

print(msg.isalpha())
msg = "jamesbond"
print(msg.isalpha())

msg = "007"
print(msg.isdigit())

msg = "Hello"
print(msg.islower())

msg = "    "
print(msg.isspace())

msg = "Hello"
print(msg.isupper())

print(len(msg))

s = "Hello"
print(s.ljust(10,'%'))

print(s.rjust(10,'*'))
```

```
print(s.rjust(10))

s = "-1234"
print(s.zfill(10))

s = "  Hello  "
print('abc' + s.lstrip() + 'zyx')

print('abc' + s.rstrip() + 'zyx')

print('abc' + s.strip() + 'zyx')

s = "Hello friends"
print(max(s))

s = "Hello Hello Hello"
print(s.replace("He", "Fo"))
print(s.replace("He", "Fo", 2))

s = "The world is beautiful"
print(s.title())

s = "hEllO WorLD"
print(s.swapcase())

s = "abc, def, ghi, jkl"
print(s.split(','))

# WAP to print the pattern
for i in range(1, 7):
    ch = 'A'
    print()
    for j in range(1, i+1):
        print(ch, end="")
        ch = chr(ord(ch)+1)

# WAP using while loop to iterate a given string
s = "Welcome to Python"
i = 0
while i < len(s):
    print(s[i], end="")
    i+=1

# WAP that encrypts a message by adding a key value to every character
s = input("Enter the string: ")
key = int(input("Enter the encryption key: "))
new_s = ""
```

```
for i in s:
    new_s += chr(ord(i)+key)
print(new_s)

# WAP that uses split function to split a multi-line string
s = '''Dear Students, I am pleased to inform you that, there is a workshop on Python in college tomorrow.
Everyone should come and there will also be a quiz in Python, whosoever wins will win a gold medal.'''

print(s.split('\n'))

# WAP that accepts a string from user and re-displays the same string after removing vowels
vowels = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']
s = input("Enter the string: ")
for i in s:
    if i not in vowels:
        print(i, end="")

pattern = r"[a-zA-Z]+\s+\d+"
# Patterns that begin with one or more characters followed by space and followed by one or more digits
matches = re.finditer(pattern, "LXI 2013,VXI 2015,VDI 20104,Maruti Suzuki Cars available with us")
for match in matches:
    print(match.start(), match.end(), match.span())

# WAP that uses a regex to match strings which start with sequence of digits (atleast 1) followed by a blank and after this add arbitrary characters

pat = r"^\d+\s*"
pat = r"^[0-9]+ .*"
if re.match(pat, "123 adij"):
    print("Good")
```

RESULTS:

1.Slicing – WAP to Get the characters from o in “World” to but not included d in "World"

```
[2]: a = "HelloWorld"
     # Get the characters from o in World to but not included d in "World"
     print(a[-4:-1])

     orl
```

2.WAP to display powers of number without using formatting characters

```
# WAP to display powers of number without using formatting characters
i=1
while i<=5:
    print(i**1, "\t", i**2, "\t",  i**3, "\t",  i**4)
    i+=1
print()
print()

i=1
while i<=5:
    print("%d\t%d\t%d\t%d"%(i**1, i**2, i**3,  i**4))
    i+=1
print()
print()

i = 1
print("%-4s%-5s%-6s"%('i', 'i**2', 'i**3'))
print()
print()

i = 1
while i<=5:
    print("%-4d%-5d%-6d"%(i, i**2, i**3))
    i+=1
```

```
1       1       1       1
2       4       8       16
3       9       27      81
4       16      64      256
5       25      125     625

1       1       1       1
2       4       8       16
3       9       27      81
4       16      64      256
5       25      125     625

i   i**2 i**3

1   1    1
2   4    8
3   9    27
4   16   64
5   25   125
```

3.String methods and functions –

i.) capitalize(), center(), count(), endswith(), startswith(), find(), index(), rfind(), rindex(), isalnum(), isalpha(), isdigit(), islower(), isupper(), len(), etc.

```
msg = "is this your bag?"
print(msg.rfind("is", 0, len(msg)))

print(msg.rindex("is"))
try:
    print(msg.rindex("z"))
except:
    print("substring not found")

msg = "jamesbond007"
print(msg.isalnum())

print(msg.isalpha())
msg = "jamesbond"
print(msg.isalpha())

msg = "007"
print(msg.isdigit())

msg = "Hello"
print(msg.islower())

msg = "   "
print(msg.isspace())

msg = "Hello"
print(msg.isupper())

print(len(msg))
```

```
Hello
**hello***
2
True
True
7
-1
substring not found
5
5
substring not found
True
False
True
True
False
True
False
5
```

```
s = "Hello"
print(s.ljust(10,'%'))

print(s.rjust(10,'*'))
print(s.rjust(10))

s = "-1234"
print(s.zfill(10))

s = "   Hello   "
print('abc' + s.lstrip() + 'zyx')

print('abc' + s.rstrip() + 'zyx')

print('abc' + s.strip() + 'zyx')

s = "Hello friends"
print(max(s))

s = "Hello Hello Hello"
print(s.replace("He", "Fo"))
print(s.replace("He", "Fo", 2))

s = "The world is beautiful"
print(s.title())

s = "hEllO WorLD"
print(s.swapcase())

s = "abc, def, ghi, jkl"
print(s.split(','))
```

```
Hello%%%%%
*****Hello
     Hello
-000001234
abcHello   zyx
abc   Hellozyx
abcHellozyx
s
Follo Follo Follo
Follo Follo Hello
The World Is Beautiful
HeLLo wORld
['abc', ' def', ' ghi', ' jkl']
```

ii.) WAP to print following pattern

A
AB
ABC
ABCD
ABCDE
ABCDEF

```
# WAP to print the pattern
for i in range(1, 7):
    ch = 'A'
    print()
    for j in range(1, i+1):
        print(ch, end="")
        ch = chr(ord(ch)+1)
```

```
A
AB
ABC
ABCD
ABCDE
ABCDEF
```

iii.) WAP using while loop to iterate a given string

```
# WAP using while loop to iterate a given string
s = "Welcome to Python"
i = 0
while i < len(s):
    print(s[i], end="")
    i+=1
```

```
Welcome to Python
```

iv.) WAP that encrypts a message by adding a key value to every character

```
# WAP that encrypts a message by adding a key value to every character
s = input("Enter the string: ")
key = int(input("Enter the encryption key: "))
new_s = ""
for i in s:
    new_s += chr(ord(i)+key)
print(new_s)
```

```
Enter the string:  Satya
Enter the encryption key:  5
Xfy~f
```

v.)WAP that uses split function to split a multi-line string

```
# WAP that uses split function to split a multi-line string
s = '''Dear Students, I am pleased to inform you that, there is a workshop on Python in college tomorrow.
Everyone should come and there will also be a quiz in Python, whosoever wins will win a gold medal.'''

print(s.split('\n'))
```

```
['Dear Students, I am pleased to inform you that, there is a workshop on Python in college tomorrow.', 'Everyone should come and there will a
lso be a quiz in Python, whosoever wins will win a gold medal.']
```

vi.)WAP that accepts a string from user and re-displays the same string after removing vowels

```
# WAP that accepts a string from user and re-displays the same string after removing vowels
vowels = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']
s = input("Enter the string: ")
for i in s:
    if i not in vowels:
        print(i, end="")

pattern = r"[a-zA-Z]+\s+\d+"
```

```
Enter the string:  SATYA
STY
```

4.Regular Expressions

i.)WAP to find patterns that begin with one or more characters followed by space and followed by one or more digits

```
# Patterns that begin with one or more characters followed by space and followed by one or more digits
import re
matches = re.finditer(pattern, "LXI 2013,VXI 2015,VDI 20104,Maruti Suzuki Cars available with us")
for match in matches:
    print(match.start(), match.end(), match.span())
```

```
0 8 (0, 8)
9 17 (9, 17)
18 27 (18, 27)
```

ii.)WAP that uses a regex to match strings which start with sequence of digits (atleast 1) followed by a blank and after this add arbitrary characters

```
# WAP that uses a regex to match strings which start with sequence of digits (atleast 1) followed by a blank and after th

pat = r"^\d+\s*"
pat = r"^[0-9]+ .*"
if re.match(pat, "123 adij"):
    print("Good")
```

```
Good
```

EXPERIMENT 10

OBJECTIVE: Demonstrate file handling and develop code for given problem statements:

1) WAP that copies first 10 bytes of a binary file into another
2) WAP that accepts a file name as an input from the user. Open the file and count the number of times a character appears in the file
3) WAP to create a new directory in the current directory, WAP that changes current directory to newly created directory new_dir, WAP to delete new_dir
4) WAP to print the absolute path of a file using os.path.join

THEORY:

File Handling in Python

File handling is a mechanism in Python that allows us to work with files (read, write, and manipulate). Python provides built-in functions and methods to perform file operations.

CODE:

```
# WAP that copies first 10 bytes of a binary file into another
with open("file_handling_test/file1.txt", "rb") as f:
    a = f.read(10)
    print("First 10 bytes of file1: ", a)

with open("file2.txt", "wb+") as f2:
    print("File2 contents:")
    print(f2.read())
    f2.seek(0)
    t = f2.write(a)
    f2.seek(0)
    print("File2 contents after copying:")
    print(f2.read())

# WAP that accepts a file name as an input from the user. Open the file and count the number of times a character appears in the file

f = input("Enter the file name: ")
ch = input("Enter the character to be searched: ")
count = 0
with open("file_handling_test/"+f, "r") as f1:
    for line in f1:
        for c in line:
            if c == ch:
                count+=1
print("Count of given character in file: ", count)
```

```
# WAP to create a new directroy in the current directory
os.mkdir("new_dir")

# WAP that changes curr dur to newly created dir new_dir
os.chdir("new_dir")

# WAP to delete new_dir
os.rmdir("new_dir")

# WAP to print the absolute path of a file using os.path.join

file_name = "file1.txt"  # File name to get the absolute path
try:
    current_dir = os.getcwd()  # Get the current working directory
    abs_path = os.path.join(current_dir, file_name)  # Join path
    print(f"Absolute path of '{file_name}': {abs_path}")
except Exception as e:
    print(f"Error occurred: {e}")
```

RESULTS:

1) WAP that copies first 10 bytes of a binary file into another

```
# WAP that copies first 10 bytes of a binary file into another
with open("C:/Users/Satyartha Shukla/Desktop/MTech AI/CPBP/director/file1.txt", "rb") as f:
    a = f.read(10)
    print("First 10 bytes of file1: ", a)

with open("file2.txt", "wb+") as f2:
    print("File2 contents:")
    print(f2.read())
    f2.seek(0)
    t = f2.write(a)
    f2.seek(0)
    print("File2 contents after copying:")
    print(f2.read())
```

```
First 10 bytes of file1:  b'hello worl'
File2 contents:
b''
File2 contents after copying:
b'hello worl'
```

2) WAP that accepts a file name as an input from the user. Open the file and count the number of times a character appears in the file

```
# WAP that accepts a file name as an input from the user. Open the file and count the number of times a character appears in the file

f = input("Enter the file name: ")
ch = input("Enter the character to be searched: ")
count = 0
with open(f, "r") as f1:
    for line in f1:
        for c in line:
            if c == ch:
                count+=1
print("Count of given character in file: ", count)
```

```
Enter the file name:  C:/Users/Satyartha Shukla/Desktop/MTech AI/CPBP/director/file1.txt
Enter the character to be searched:  t
Count of given character in file:  2
```

3) WAP to create a new directory in the current directory, WAP that changes current directory to newly created directory new_dir, WAP to delete new_dir

```python
# WAP to create a new directroy in the current directory
import os
os.mkdir("new_dir")
```

```python
# WAP that changes curr dur to newly created dir new_dir
os.chdir("new_dir")
```

```python
# WAP to delete new_dir
os.rmdir("new_dir")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[43], line 2
      1 # WAP to delete new_dir
----> 2 os.rmdir("new_dir")

FileNotFoundError: [WinError 2] The system cannot find the file specified: 'new_dir'
```

4) WAP to print the absolute path of a file using os.path.join

```python
file_name = "file1.txt"  # File name to get the absolute path
try:
    current_dir = os.getcwd()  # Get the current working directory
    abs_path = os.path.join(current_dir, file_name)  # Join path
    print(f"Absolute path of '{file_name}': {abs_path}")
except Exception as e:
    print(f"Error occurred: {e}")
```

```
Absolute path of 'file1.txt': C:\Users\Satyartha Shukla\Desktop\MTech AI\CPBP\Practical\new_dir\new_dir\file1.txt
```

EXPERIMENT 11

OBJECTIVE: Demonstrate Class, Objects and Inheritance and develop code for given problem statements:

1) WAP with class Employee that keeps a track of the number of employees in an organisation and also stores their name, designation, and salary details.
2) WAP that has a class Circle. Use a class variable to define the value of constant pi. Use this class variable to calculate area and curcumference of a circle with specified radius.
3) WAP that has a class Point. Define another class Location which has 2 objects - location and destination. Also define a function in location that prints the reflection of destination on the x-axis.
4) WAP that has classes such as Student, Course, Department. Enroll a student in a course of a particular department. Classes are -
    a. Student details - name, roll no
    b. Course - name, code, year and semester
    c. Department – Name

THEORY:

Class in Python:

A class is a user-defined blueprint or prototype used to define an object. It provides a means of bundling data and functionality together. Objects are instances of a class, and they inherit the properties and behaviors defined in the class.

Key Components of a Class

1. Attributes

Attributes are variables that hold data about the object. They can be:

Class Attributes: Shared across all instances of the class.

Instance Attributes: Specific to each object, defined within the constructor (__init__).

2. Methods

Methods are functions defined inside a class. They operate on the attributes of the class.

Instance Methods: Use the self keyword to access instance attributes and methods. Operate on specific objects.

Class Methods: Use @classmethod and the cls parameter. Operate on the class itself and access class attributes.

Static Methods: Use @staticmethod. Do not operate on class or instance attributes and are independent of the class.

Inheritance

Inheritance is a fundamental concept in Object-Oriented Programming (OOP) that allows one class (the child or derived class) to acquire properties and behaviors (methods and attributes)

of another class (the parent or base class). It promotes code reuse and helps in building a hierarchical relationship between classes.

Types of Inheritance in Python

1. Single Inheritance
2. Multiple Inheritance
3. Multilevel Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

CODE:

WAP with class Employee that keeps a track of the nummber of employees in an organisation and also stores their name, designation, and salary details.

```
class Employee:
    global count_of_emp
    count_of_emp = 0

    def __init__(self):
        self.emp = {}
    def enterEmployeeDetails(self):
        i = int(input("Enter Employee id: "))
        n = input("Enter Employee Name: ")
        d = input("Enter Employee Designation: ")
        s = input("Enter Employee Salary: ")
        self.emp.update({str(i): [n, d, s]})
    def displayCount(self):
        print("Total count of employees: ", count_of_emp)
    def displayDetails(self):
        print("List of Employees and their details: ", self.emp)

e1 = Employee()
e1.enterEmployeeDetails()
count_of_emp += 1

e2 = Employee()
e2.enterEmployeeDetails()
count_of_emp += 1

e1.displayDetails()
e2.displayDetails()

e2.displayCount()
```

WAP that has a class Circle. Use a class variable to define the value of constant pi. Use this class variable to calculate area and curcumference of a circle with specified radius.

```
class Circle:
    def __init__(self, radius):
        self.radius = radius
        self.area = 0
        self.circum = 0
        self.pi = 3.14
    def calcArea(self):
        self.area = self.pi * self.radius * self.radius
    def calcCircum(self):
        self.circum = 2 * self.pi * self.radius
    def printDetails(self):
        print()
        print("Given radius: ", self.radius)
        print("Area of circle: ", self.area)
        print("Circumference of circle: ", self.circum)

c1 = Circle(7)
c1.calcArea()
c1.calcCircum()

c2 = Circle(10)
c2.calcArea()
c2.calcCircum()

c1.printDetails()
c2.printDetails()
```

WAP that has a class Point. Define another class Location which has 2 objects - location and destination.
Also define a function in location that prints the refelction of destination on the x-axis.

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.x_new = 0
        self.y_new = 0
    def display_point(self):
        print(f'X = {self.x}, Y = {self.y}')

class Location(Point):
    def reflection(self):
        self.y_new = self.y * -1
        self.x_new = self.x
```

```
        print(f'X_relected = {self.x_new}, Y_reflected = {self.y_new}')

location = Location(1, 2)
destination = Location(10, 20)

location.display_point()
destination.display_point()
destination.reflection()
# WAP that has classes such as Student, Course, Department. Enroll a student in a course of a particular department. Classes are -
# Student details - name, roll no
# Course -  name, code, year and semester
# Department - Name

# Base class: Person
class Person:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f"Name: {self.name}"


# Student class inheriting from Person
class Student(Person):
    def __init__(self, name, roll_no):
        super().__init__(name)
        self.roll_no = roll_no

    def __str__(self):
        return f"Student Name: {self.name}, Roll No: {self.roll_no}"


# Base class: AcademicEntity
class AcademicEntity:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f"{self.__class__.__name__} Name: {self.name}"


# Course class inheriting from AcademicEntity
class Course(AcademicEntity):
    def __init__(self, name, code, year, semester):
        super().__init__(name)
        self.code = code
```

```
        self.year = year
        self.semester = semester

    def __str__(self):
        return f"Course Name: {self.name}, Code: {self.code}, Year: {self.year}, Semester: {self.semester}"


# Department class inheriting from AcademicEntity
class Department(AcademicEntity):
    def __init__(self, name):
        super().__init__(name)
        self.courses = []  # List of courses in the department

    def add_course(self, course):
        self.courses.append(course)

    def __str__(self):
        return f"Department Name: {self.name}"


# Enrollment system for handling enrollments
class Enrollment:
    def __init__(self):
        self.enrollments = {}  # Maps students to courses

    def enroll_student(self, student, course, department):
        if student not in self.enrollments:
            self.enrollments[student] = []
        self.enrollments[student].append((course, department))

    def display_enrollments(self):
        for student, courses in self.enrollments.items():
            print(f"\n{student}:")
            for course, department in courses:
                print(f"  Enrolled in {course} of {department}")


# Main menu-driven program
if __name__ == "__main__":
    # Initialize data structures
    students = []
    courses = []
    departments = []
    enrollment_system = Enrollment()

    while True:
```

```
print("\nMenu:")
print("1. Add Department")
print("2. Add Course")
print("3. Add Student")
print("4. Enroll Student in a Course")
print("5. Display Enrollments")
print("6. Exit")

choice = input("Enter your choice: ")

if choice == "1":
    # Add a department
    dept_name = input("Enter department name: ")
    department = Department(dept_name)
    departments.append(department)
    print(f"Department '{dept_name}' added.")

elif choice == "2":
    # Add a course
    if not departments:
        print("No departments available. Add a department first.")
        continue
    dept_name = input("Enter the department for the course: ")
    department = next((d for d in departments if d.name == dept_name), None)
    if not department:
        print("Department not found.")
        continue
    course_name = input("Enter course name: ")
    course_code = input("Enter course code: ")
    course_year = input("Enter course year: ")
    course_semester = input("Enter course semester: ")
    course = Course(course_name, course_code, course_year, course_semester)
    department.add_course(course)
    courses.append(course)
    print(f"Course '{course_name}' added to department '{dept_name}'.")

elif choice == "3":
    # Add a student
    student_name = input("Enter student name: ")
    student_roll_no = input("Enter student roll number: ")
    student = Student(student_name, student_roll_no)
    students.append(student)
    print(f"Student '{student_name}' added.")

elif choice == "4":
    # Enroll a student in a course
    if not students or not courses:
```

```
            print("No students or courses available. Add them first.")
            continue
        student_roll_no = input("Enter student roll number: ")
        student = next((s for s in students if s.roll_no == student_roll_no), None)
        if not student:
            print("Student not found.")
            continue
        course_code = input("Enter course code: ")
        course = next((c for c in courses if c.code == course_code), None)
        if not course:
            print("Course not found.")
            continue
        department = next((d for d in departments if course in d.courses), None)
        if not department:
            print("Department for the course not found.")
            continue
        enrollment_system.enroll_student(student, course, department)
        print(f"Student '{student.name}' enrolled in course '{course.name}'.")

    elif choice == "5":
        # Display all enrollments
        enrollment_system.display_enrollments()

    elif choice == "6":
        # Exit the program
        print("Exiting program. Goodbye!")
        break

    else:
        print("Invalid choice. Please try again.")
```

RESULTS:

1) WAP with class Employee that keeps a track of the number of employees in an organisation and also stores their name, designation, and salary details.

```
# WAP with class Employee that keeps a track of the nummber of employees in an organisation and also stores their name, designation, and sal

class Employee:
    global count_of_emp
    count_of_emp = 0

    def __init__(self):
        self.emp = {}
    def enterEmployeeDetails(self):
        i = int(input("Enter Employee id: "))
        n = input("Enter Employee Name: ")
        d = input("Enter Employee Designation: ")
        s = input("Enter Employee Salary: ")
        self.emp.update({str(i): [n, d, s]})
    def displayCount(self):
        print("Total count of employees: ", count_of_emp)
    def displayDetails(self):
        print("List of Employees and their details: ", self.emp)

e1 = Employee()
e1.enterEmployeeDetails()
count_of_emp += 1

e2 = Employee()
e2.enterEmployeeDetails()
count_of_emp += 1

e1.displayDetails()
e2.displayDetails()

e2.displayCount()
```

```
Enter Employee id:  100
Enter Employee Name:  Satya
Enter Employee Designation:  Secretary
Enter Employee Salary:  50000
Enter Employee id:  01
Enter Employee Name:  CEO
Enter Employee Designation:  Shubham
Enter Employee Salary:  1500000
List of Employees and their details:  {'100': ['Satya', 'Secretary', '50000']}
List of Employees and their details:  {'1': ['CEO', 'Shubham', '1500000']}
Total count of employees:  2
```

2) WAP that has a class Circle. Use a class variable to define the value of constant pi. Use this class variable to calculate area and curcumference of a circle with specified radius.

```
# WAP that has a class Circle. Use a class variable to define the value of constant pi. Use this class variable to calculate area and curcumf

class Circle:
    def __init__(self, radius):
        self.radius = radius
        self.area = 0
        self.circum = 0
        self.pi = 3.14
    def calcArea(self):
        self.area = self.pi * self.radius * self.radius
    def calcCircum(self):
        self.circum = 2 * self.pi * self.radius
    def printDetails(self):
        print()
        print("Given radius: ", self.radius)
        print("Area of circle: ", self.area)
        print("Circumference of circle: ", self.circum)

c1 = Circle(7)
c1.calcArea()
c1.calcCircum()

c2 = Circle(10)
c2.calcArea()
c2.calcCircum()

c1.printDetails()
c2.printDetails()
```

```
Given radius:  7
Area of circle:  153.86
Circumference of circle:  43.96

Given radius:  10
Area of circle:  314.0
Circumference of circle:  62.800000000000004
```

3) WAP that has a class Point. Define another class Location which has 2 objects - location and destination. Also define a function in location that prints the reflection of destination on the x-axis.

```python
# WAP that has a class Point. Define another class Location which has 2 objects - location and destination.
# Also define a function in location that prints the refelction of destination on the x-axis.

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.x_new = 0
        self.y_new = 0
    def display_point(self):
        print(f'X = {self.x}, Y = {self.y}')

class Location(Point):
    def reflection(self):
        self.y_new = self.y * -1
        self.x_new = self.x
        print(f'X_relected = {self.x_new}, Y_reflected = {self.y_new}')

location = Location(1, 2)
destination = Location(10, 20)

location.display_point()
destination.display_point()
destination.reflection()
```

```
X = 1, Y = 2
X = 10, Y = 20
X_relected = 10, Y_reflected = -20
```

4) WAP that has classes such as Student, Course, Department. Enroll a student in a course of a particular department. Classes are -
    a. Student details - name, roll no
    b. Course - name, code, year and semester
    c. Department – Name

```python
# WAP that has classes such as Student, Course, Department. Enroll a student in a course of a particular department. Classes are - Student
class Person:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f"Name: {self.name}"

# Student class inheriting from Person
class Student(Person):
    def __init__(self, name, roll_no):
        super().__init__(name)
        self.roll_no = roll_no

    def __str__(self):
        return f"Student Name: {self.name}, Roll No: {self.roll_no}"

# Base class: AcademicEntity
class AcademicEntity:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f"{self.__class__.__name__} Name: {self.name}"

# Course class inheriting from AcademicEntity
class Course(AcademicEntity):
    def __init__(self, name, code, year, semester):
        super().__init__(name)
        self.code = code
        self.year = year
        self.semester = semester

    def __str__(self):
        return f"Course Name: {self.name}, Code: {self.code}, Year: {self.year}, Semester: {self.semester}"
```

```python
# Department class inheriting from AcademicEntity
class Department(AcademicEntity):
    def __init__(self, name):
        super().__init__(name)
        self.courses = []  # List of courses in the department

    def add_course(self, course):
        self.courses.append(course)

    def __str__(self):
        return f"Department Name: {self.name}"

# Enrollment system for handling enrollments
class Enrollment:
    def __init__(self):
        self.enrollments = {}  # Maps students to courses

    def enroll_student(self, student, course, department):
        if student not in self.enrollments:
            self.enrollments[student] = []
        self.enrollments[student].append((course, department))

    def display_enrollments(self):
        for student, courses in self.enrollments.items():
            print(f"\n{student}:")
            for course, department in courses:
                print(f"  Enrolled in {course} of {department}")
```

```python
# Main menu-driven program
if __name__ == "__main__":
    # Initialize data structures
    students = []
    courses = []
    departments = []
    enrollment_system = Enrollment()

    while True:
        print("\nMenu:")
        print("1. Add Department")
        print("2. Add Course")
        print("3. Add Student")
        print("4. Enroll Student in a Course")
        print("5. Display Enrollments")
        print("6. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            # Add a department
            dept_name = input("Enter department name: ")
            department = Department(dept_name)
            departments.append(department)
            print(f"Department '{dept_name}' added.")

        elif choice == "2":
            # Add a course
            if not departments:
                print("No departments available. Add a department first.")
                continue
            dept_name = input("Enter the department for the course: ")
            department = next((d for d in departments if d.name == dept_name), None)
            if not department:
                print("Department not found.")
                continue
            course_name = input("Enter course name: ")
```

```
        course_code = input("Enter course code: ")
        course_year = input("Enter course year: ")
        course_semester = input("Enter course semester: ")
        course = Course(course_name, course_code, course_year, course_semester)
        department.add_course(course)
        courses.append(course)
        print(f"Course '{course_name}' added to department '{dept_name}'.")

    elif choice == "3":
        # Add a student
        student_name = input("Enter student name: ")
        student_roll_no = input("Enter student roll number: ")
        student = Student(student_name, student_roll_no)
        students.append(student)
        print(f"Student '{student_name}' added.")

    elif choice == "4":
        # Enroll a student in a course
        if not students or not courses:
            print("No students or courses available. Add them first.")
            continue
        student_roll_no = input("Enter student roll number: ")
        student = next((s for s in students if s.roll_no == student_roll_no), None)
        if not student:
            print("Student not found.")
            continue
        course_code = input("Enter course code: ")
        course = next((c for c in courses if c.code == course_code), None)
        if not course:
            print("Course not found.")
            continue
        department = next((d for d in departments if course in d.courses), None)
        if not department:
            print("Department for the course not found.")
            continue
        enrollment_system.enroll_student(student, course, department)
        print(f"Student '{student.name}' enrolled in course '{course.name}'.")
```

```
        elif choice == "5":
            # Display all enrollments
            enrollment_system.display_enrollments()

        elif choice == "6":
            # Exit the program
            print("Exiting program. Goodbye!")
            break

        else:
            print("Invalid choice. Please try again.")
```

```
Menu:
1. Add Department
2. Add Course
3. Add Student
4. Enroll Student in a Course
5. Display Enrollments
6. Exit
Enter your choice:  1
Enter department name:  Physics
Department 'Physics' added.

Menu:
1. Add Department
2. Add Course
3. Add Student
4. Enroll Student in a Course
5. Display Enrollments
6. Exit
Enter your choice:  2
Enter the department for the course:  Physical Science
Department not found.

Menu:
1. Add Department
2. Add Course
```

EXPERIMENT 12

OBJECTIVE: Demonstrate Polymorphism, Error and Exception handling and code for given problem statements:

1) Demonstrate operator overloading

2) Demonstrate Method Overriding

3) WAP to handle the divide by zero exception

4) Demonstrate Raise Exceptions, Instantiating Exceptions, assertion

5) WAP that prompts the use to enter a number and prints the square of that number.

If no number is entered, then a Key Board Interrupt is generated

6) WAP which infinitely prints natural numbers. Raise the stopIterationException after displaying first 20 numbers tp exit from the program

7) WAP that randomly generates a number. Raise a UserDefined exception if the

number is below 0.1

THEORY:

Polymorphism

Polymorphism is the ability of an object to take many forms. In Python, polymorphism is commonly implemented through method overriding and operator overloading.

1. **Operator Overloading**: Allows us to redefine the behavior of operators (+, -, etc.) for user-defined classes. For example, overloading the + operator in the ComplexNumber class enables us to add two complex numbers.
2. **Method Overriding**: When a child class provides a specific implementation of a method that is already defined in its parent class. The child class method overrides the parent class method, allowing for dynamic behavior.

Error and Exception Handling

Python uses exceptions to handle runtime errors gracefully without abruptly terminating the program. This is done using try, except, finally, and else blocks.

1. **Divide by Zero Exception**: Raised when a number is divided by zero. This is handled using a try-except block where the exception is caught and managed.
2. **Custom (User-Defined) Exceptions**: Developers can define their own exceptions to signal specific errors. These exceptions are typically derived from the built-in Exception class.

3. **Assertion**: Assertions are used to verify assumptions in a program. If the condition of an assertion is false, the program raises an AssertionError. For example, assert num > 0 ensures the user enters a positive number.

Raise Exceptions

- The raise statement is used to trigger an exception manually. For example, raise ValueError("Invalid value") creates and raises a ValueError.
- Exceptions can also be instantiated using custom messages, making error handling more meaningful for specific use cases.

Key Concepts

1. **KeyboardInterrupt Exception**: Raised when the user interrupts program execution, such as pressing Ctrl+C. This can be simulated in the program to handle scenarios like invalid input.
2. **StopIteration Exception**: Used to signal the end of an iteration. This exception is automatically raised when a loop runs out of elements, but it can also be explicitly raised, as demonstrated in generating natural numbers up to 20.
3. **Random Number Generation and User-Defined Exceptions**: Demonstrates the use of Python's random module to generate a number and raise a custom exception if certain criteria are not met.

Applications of Concepts in Code

1. **Operator Overloading and Polymorphism**:
    - Example: Overloading + for adding complex numbers.
    - Benefits: Enhances code readability and makes user-defined objects behave like built-in types.
2. **Method Overriding**:
    - Example: Child class overriding the behavior of the greet() method.
    - Benefits: Allows child classes to provide specific implementations while maintaining the interface provided by the parent class.
3. **Error Handling**:
    - Example: Handling division by zero or invalid inputs.
    - Benefits: Prevents program crashes and ensures robust software.

CODE:

```
#Demonstrate operator overloading

class ComplexNumber:

    def __init__(self, real, imag):

        self.real = real

        self.imag = imag


    def __add__(self, other):
```

```
        return ComplexNumber(self.real + other.real, self.imag + other.imag)

    def __str__(self):
        return f"{self.real} + {self.imag}i"

c1 = ComplexNumber(2, 3)
c2 = ComplexNumber(4, 5)
result = c1 + c2  # Uses overloaded __add__ method
print("Sum of complex numbers:", result)

#Demonstrate Method Overriding
class Animal:
    def sound(self):
        print("Some generic animal sound")

class Dog(Animal):
    def sound(self):
        print("Bark")

dog = Dog()
dog.sound()  # Method overridden in the Dog class

#WAP to handle the divide by zero exception
try:
    numerator = float(input("Enter numerator: "))
    denominator = float(input("Enter denominator: "))
    result = numerator / denominator
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Division by zero is not allowed!")

#Demonstrate Raise Exceptions, Instantiating Exceptions, assertion
```

```
# Raising an exception
try:
    age = int(input("Enter your age: "))
    if age < 0:
        raise ValueError("Age cannot be negative!")
except ValueError as e:
    print(f"Exception: {e}")

# Instantiating custom exceptions
class CustomError(Exception):
    pass

try:
    x = -5
    if x < 0:
        raise CustomError("Negative number error")
except CustomError as e:
    print(f"Custom Exception: {e}")

# Assertion example
num = int(input("Enter a positive number: "))
assert num > 0, "Number must be positive!"
print(f"You entered: {num}")
```

#WAP that prompts the use to enter a number and prints the square of that number.If no number is entered, then a KeyBoardInterrupt is generated

```
try:
    num = input("Enter a number: ")
    if not num:
        raise KeyboardInterrupt("No input provided!")
    num = int(num)
    print("Square of the number:", num ** 2)
```

```
except KeyboardInterrupt as e:
    print(f"Error: {e}")
except ValueError:
    print("Error: Please enter a valid number.")
```

#WAP which infinitely prints natural numbers. Raise the stopIterationException after displaying first 20 numbers tp exit from the program.

```
class NaturalNumbers:
    def __init__(self):
        self.num = 1

    def __iter__(self):
        return self

    def __next__(self):
        if self.num > 20:  # Stop after printing 20 numbers
            raise StopIteration
        current_num = self.num
        self.num += 1
        return current_num

numbers = NaturalNumbers()
for number in numbers:
    print(number)
```

#WAP that randomly generates a number. Raise a UserDefined excepϴon if the number is below 0.1

```
import random

class SmallNumberError(Exception):
    """Custom exception for numbers below 0.1"""
    pass
```

```
try:
    num = random.random()  # Generate a random number between 0 and 1
    print("Generated number:", num)
    if num < 0.1:
        raise SmallNumberError("Generated number is too small!")
except SmallNumberError as e:
    print(f"Error: {e}")
```

Result:

1. **Demonstrate operator overloading**

```
]: #Demonstrate operator overloading
class ComplexNumber:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __add__(self, other):
        return ComplexNumber(self.real + other.real, self.imag + other.imag)

    def __str__(self):
        return f"{self.real} + {self.imag}i"

c1 = ComplexNumber(2, 3)
c2 = ComplexNumber(4, 5)
result = c1 + c2  # Uses overloaded __add__ method
print("Sum of complex numbers:", result)
```

```
Sum of complex numbers: 6 + 8i
```

2. Demonstrate Method Overriding

```
: #Demonstrate Method Overriding
class Animal:
    def sound(self):
        print("Some generic animal sound")

class Dog(Animal):
    def sound(self):
        print("Bark")

dog = Dog()
dog.sound()  # Method overridden in the Dog class
```

```
Bark
```

3. WAP to handle the divide by zero exception

```python
#WAP to handle the divide by zero exception
try:
    numerator = float(input("Enter numerator: "))
    denominator = float(input("Enter denominator: "))
    result = numerator / denominator
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Division by zero is not allowed!")
```

```
Enter numerator:  45
Enter denominator:  0
Error: Division by zero is not allowed!
```

4. Demonstrate Raise Exceptions, Instantiating Exceptions, assertion

```python
#Demonstrate Raise Exceptions, Instantiating Exceptions, assertion
# Raising an exception
try:
    age = int(input("Enter your age: "))
    if age < 0:
        raise ValueError("Age cannot be negative!")
except ValueError as e:
    print(f"Exception: {e}")

# Instantiating custom exceptions
class CustomError(Exception):
    pass

try:
    x = -5
    if x < 0:
        raise CustomError("Negative number error")
except CustomError as e:
    print(f"Custom Exception: {e}")

# Assertion example
# Assertion example
num = int(input("Enter a positive number: "))
assert num > 0, "Number must be positive!"
print(f"You entered: {num}")
```

```
Enter your age:  45
Custom Exception: Negative number error
Enter a positive number:  4
You entered: 4
```

5. WAP that prompts the use to enter a number and prints the square of that number. If no number is entered, then a KeyBoardInterrupt is generated

```
#WAP that prompts the use to enter a number and prints the square of that number.If no number is
try:
    num = input("Enter a number: ")
    if not num:
        raise KeyboardInterrupt("No input provided!")
    num = int(num)
    print("Square of the number:", num ** 2)
except KeyboardInterrupt as e:
    print(f"Error: {e}")
except ValueError:
    print("Error: Please enter a valid number.")
```

```
Enter a number:
Error: No input provided!
```

6. WAP which infinitely prints natural numbers. Raise the stopIterationException after displaying first 20 numbers tp exit from the program

```
#WAP which infinitely prints natural numbers. Raise the stopIterationException after displaying first 20 numbers tp exit
class NaturalNumbers:
    def __init__(self):
        self.num = 1

    def __iter__(self):
        return self

    def __next__(self):
        if self.num > 20:  # Stop after printing 20 numbers
            raise StopIteration
        current_num = self.num
        self.num += 1
        return current_num

numbers = NaturalNumbers()
for number in numbers:
    print(number)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

7. WAP that randomly generates a number. Raise a UserDefined exception if the number is below 0.1

```python
#WAP that randomly generates a number. Raise a UserDefined excepθon if the number is below 0.1
import random

class SmallNumberError(Exception):
    """Custom exception for numbers below 0.1"""
    pass

try:
    num = random.random()  # Generate a random number between 0 and 1
    print("Generated number:", num)
    if num < 0.1:
        raise SmallNumberError("Generated number is too small!")
except SmallNumberError as e:
    print(f"Error: {e}")
```

```
Generated number: 0.6551580294819157
```