CONTENTS

S. NO.	EXPERIMENT	DATE OF SUBMISSION	PAGE NO.	REMARKS
1.	Install Python and write basic programs to explore its syntax and functionality.	12 Aug	1	
2.	Demonstrate python operators and develop code for given problem statements.	19-Aug	2	
3.	Demonstrate conditional and loop statements and develop code for given problem statements.	26-Aug	5	
4.	Demonstrate list operations and develop code for given problem statements.	02-Sep	10	
5.	Demonstrate arrays and tuples and develop code for given problem statements.	09-Sep	12	
6.	Demonstrate functions and modules and develop code for given problem statements.	23-Sep	17	
7.	Demonstrate Set operations and develop code for given problem statements.	31-oct	20	
8.	Demonstrate dictionary operations and develop code for given problem statements.	21-Oct	23	
9.	Demonstrate strings and its related operations and develop code for given problem statements.	04-Nov	26	

10.	Demonstrate file handling and develop code for given problem statements.	11-Nov	32	
11.	Demonstrate Class, object and inheritance problem statements.	18-Nov	34	
12.	Demonstrate polymorphism, Error and Exception handling code for given problem statements.	25-Nov	38	

EXPERIMENT 1

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

THEORY:

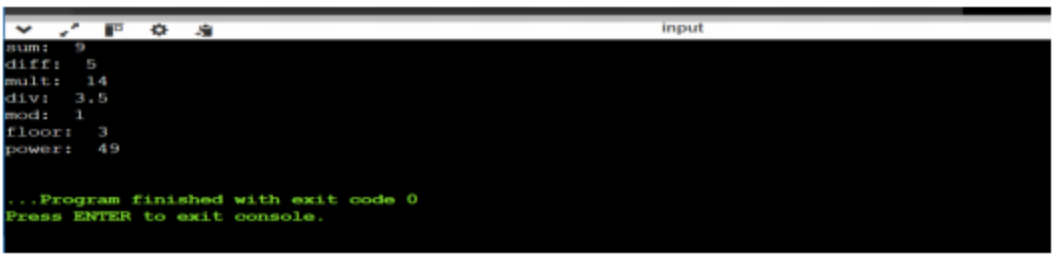
Python is a high-level, interpreted programming language created by Guido van Rossum in 1991. Known for its simplicity and readability, Python uses indentation for defining code blocks, making it beginner-friendly. It supports multiple programming paradigms, including procedural, object-oriented, and 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:

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:



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

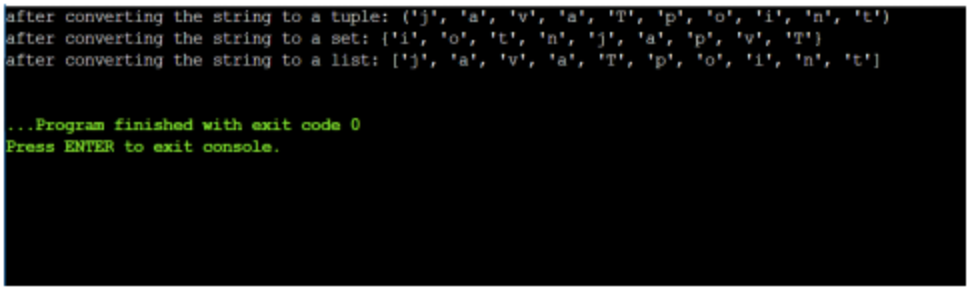
```
# Comparison Operators
a=5
b=2
print(a==b)
print(a!=b)
print(a>b)
print(a<b)
print(a<=b)
print(a>=b)
# Logical Operators
a=5
b=6
print((a>2) and (b>=6))
print((a>2) or (b>=6))
# Identity operators
x1=5
y1=5
x2='Hello'
y2='Hello'
x3=[1,2,3]
y3=[1,2,3]
print(x1 is not y1)
```

```
print(x2 is y2)
print(x3 is y3)
```

RESULTS

:

```
52
<class 'int'>
0x38
<class 'str'>
0o70
<class 'str'>


...Program finished with exit code 0
Press ENTER to exit console.
```



```
after converting the string to a tuple: ('j', 'a', 'v', 'a', 'T', 'p', 'o', 'i', 'n', 't')
after converting the string to a set: {'i', 'o', 't', 'n', 'j', 'a', 'p', 'v', 'T'}
after converting the string to a list: ['j', 'a', 'v', 'a', 'T', 'p', 'o', 'i', 'n', 't']


...Program finished with exit code 0
Press ENTER to exit console.
```

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


...Program finished with exit code 0
Press ENTER to exit console.
```

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


...Program finished with exit code 0
Press ENTER to exit console.
```

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 in programming allow decision-making based on conditions, using if and else to control program flow. For example, we can check if user input is a number or a character using methods like isnumeric() and isupper()/islower() for case checks. Loops (for and while) help repeat instructions, with for used for fixed ranges and while for conditions that continue until false. Nested loops can perform more complex tasks, like printing patterns. Factorials can be calculated with for loops, and leap years identified by divisibility rules. Summing series like 1 + 1/2 + 1/3 + ... is also done using loops. These constructs enable dynamic and efficient programming.

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
n=int(input("Enter the number: "))
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:

```
Enter the input: 1

*** USING IN-BUILT LIBRARIES ***
It's a number


...Program finished with exit code 0
Press ENTER to exit console.
```

```
*** 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: 5
Friday

...Program finished with exit code 0
Press ENTER to exit console.
```

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

...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter the number: 5
Factorial is:  120


...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter the year (1900-2101) to check whether leap year: 2001
Not leap year


...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter the number: 4
Sum of series is:  2.083333333333333


...Program finished with exit code 0
Press ENTER to exit console.
```

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:

In Python, lists are versatile data structures that allow for storing multiple elements. List slicing allows extracting specific sections of a list using a range of indices, while cloning creates a copy of the list. Various list methods can manipulate lists, such as insert() to add an element at a specific position, append() to add an element at the end, extend() to combine lists, and reverse() or reversed() to change the list order. The remove() method removes the first matching element, and pop() removes and returns an element at a specified index. List comprehension offers a concise way to create lists by applying an expression to each element in an iterable. Looping through lists allows processing each item, and calculating the sum and average of list elements can be done using built-in functions.

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)
# 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:

```
[2, 3, 4, 5]
['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']
[1, 9, 25, 49, 81]
1
a
abc
[2, 3, 4, 5]
8.9
index:  0
index:  1
index:  2
index:  3
index:  4
Sum =  55
Avg =  5.5


...Program finished with exit code 0
Press ENTER to exit console.
```

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:

In Python, arrays and tuples are used to store collections of data, but they differ in functionality. Arrays are mutable (allowing modification) and typically used for numerical operations, especially when using the array module. Common operations in arrays include creating, inserting, appending elements, slicing, and updating elements. Tuples, on the other hand, are immutable, meaning once created, their elements cannot be changed. They can hold elements of different data types and are often used for grouping data that should not change. Nested tuples and nested lists allow storing complex data structures like a list of tuples or tuples of lists. Tuple assignment is a convenient way to swap values without needing a temporary variable. Functions in Python can also return multiple values, such as area and circumference, using tuples. Additionally, tuple creation can be done with loops, and tuples can easily be formed by splitting data, such as an email address into a username and domain.

CODE:

```
# 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:

```
array('i', [1, 2, 3])
1 2 3 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])
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 ()
('Hello', 'Sam')
('Hello', 'Sam')
('S', 'a', 'm')
(5, 'aiojdio', 7, 'JFidsof')
((1, 2, 3), ('a', 'b', 'c'))
('arav', 97, 'B.Sc.')
('raghav', 87, 'BCA')


...Program finished with exit code 0
Press ENTER to exit console.
```

```
The original list of toppers is:  [['Arav', 'MSC', 92], ['Student2', 'MBA', 99], ['Student3', 'MTech', 94], ['Studen
t4', '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: 1
Press
1. To edit the name
2. To edit the branch
3. To edit the marks
Enter your choice (1/2/3): 2
Wrong choice entered!
New list is:  [['Arav', 'MSC', 92], ['Student2', 'MBA', 99], ['Student3', 'MTech', 94], ['Student4', 'BSC', 95]]
('Sam',)
(('Sam',),)
((('Sam',),),)
(((('Sam',),),),)
((((('Sam',),),),),)
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
Enter radius: 3
Area is:  28.274333882308138
Circumference is:  18.84955592153876
```

```
Enter the email address: shashil@gmail.com
('shashil', 'gmail.com')


...Program finished with exit code 0
Press ENTER to exit console.
```

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:

In Python, functions allow us to encapsulate code into reusable blocks, promoting modularity and reusability. Functions can accept inputs (parameters) and return values. Pass by value means that a copy of the argument is passed to the function, while pass by reference allows the function to modify the actual argument. Functions can also be used with return statements to provide outputs. Modules in Python are collections of functions and variables that can be imported into a program to extend its functionality. Recursion involves a function calling itself to solve a problem, often breaking it down into simpler subproblems. By using functions and modules, we can simplify complex tasks like calculating square numbers, checking even/odd status, calculating interest, and solving mathematical problems like finding powers of numbers.

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:

```
36
[1, 4, 9, 16, 25]
[1, 4, 9, 16, 25]
Enter num1: 2
Enter num1: 6
num1 - num2 =  -4
Enter num1: 4
Number is Even
Enter age of person: 24
Enter principal amount: 6000
Enter time in years: 2
Simple Interest is:  1200.0


...Program finished with exit code 0
Press ENTER to exit console.
```

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:

In Python, sets are unordered collections of unique elements that allow performing mathematical set operations like union, intersection, and difference. Set operations include adding elements using add() or update(), removing elements with remove() (which raises an error if the element is not found) or discard() (which doesn't raise an error if the element is absent), and clearing the entire set using clear(). Sets also support operations like checking if a set is a superset or subset of another set, and functions such as len() to find the size of the set and sum() to get the sum of its elements. Sets are useful for tasks requiring uniqueness and fast membership testing.

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:

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


...Program finished with exit code 0
Press ENTER to exit console.
```

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:

In Python, dictionaries are unordered collections of key-value pairs where each key is unique. They offer fast lookups, making them ideal for storing and accessing data efficiently. Common dictionary operations include accessing values using keys, updating or adding new key-value pairs, and deleting specific entries or the entire dictionary. Methods like len() return the size, copy() creates a shallow copy, and get() retrieves values safely. Other methods include items() for key-value pairs, values() for just the values, and update() to merge dictionaries. Iterating through a dictionary can be done through loops to access keys, values, or both. Python also provides methods like fromkeys() to create dictionaries with specified keys, and setdefault() to get values with a default if the key doesn't exist. Sorting a dictionary by values requires transforming the dictionary to a list of tuples and sorting them.

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:

```
Zara
7
{'Name': 'Zara', 'Age': 8, 'Class': 'First'}
{'Name': 'Zara', 'Age': 8, 'Class': 'First', 'School': 'DPS'}
{'Age': 8, 'Class': 'First', 'School': 'DPS'}
{}
Traceback (most recent call last):
  File "/home/main.py", line 24, in <module>
    print(dict1)
NameError: name 'dict1' is not defined. Did you mean: 'dict'?


...Program finished with exit code 1
Press ENTER to exit console.
```

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:

In Python, strings are sequences of characters, and they offer a variety of operations and methods to manipulate and process text. Slicing allows extracting substrings, specifying start and end indices, with the end index being exclusive. String methods, such as capitalize(), count(), find(), isalpha(), and len(), provide tools for modifying or checking properties of strings, such as capitalization, counting occurrences, or verifying if characters are alphabetic. Python also supports loops for iterating over strings, and encryption techniques like adding key values to each character can be implemented using ASCII values. Additionally, regular expressions (regex) offer powerful pattern matching capabilities for complex string searches, such as matching digits or specific sequences. Python's re module allows handling patterns efficiently with methods like match(), search(), and findall(), enabling flexible text processing.

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:

```
orl
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
Hello
**hello***
2
True
True
7
-1
```

```
substring not found
5
5
substring not found
True
False
True
True
False
True
False
5
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']
```

```
A
AB
ABC
ABCD
ABCDE
ABCDEF

Welcome to Python

Enter the string: hello
Enter the encryption key: 2
jgnnq


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


...Program finished with exit code 0
Press ENTER to exit console.
```

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 allows you to perform operations like reading from and writing to files, as well as manipulating file directories. Python provides various modes for opening files, such as 'r' for reading, 'w' for writing, 'a' for appending, and 'b' for binary files. You can use functions like open() to access files, and methods like read(), write(), and close() to manipulate their contents. File handling also extends to directories, where you can create, change, and delete directories using the os module. Additionally, you can interact with files using their absolute or relative paths, and Python's os.path module helps in managing paths and file system operations. These file and directory operations are crucial for tasks such as data storage, manipulation, and system management.

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")
```

RESULTS:

First 10 bytes of file1:

b'HelloWorld'

File2 contents:

b''

File2 contents after copying:

b'HelloWorld'

Enter the file name: sample.txt

Enter the character to be searched: a

Count of given character in file: 7

EXPERIMENT 11

OBJECTIVE: Demonstrate file handling 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) Write a menu driven program that keeps record of books and journals available in a library. Use logic that if you have list of books in a list, then you have a choice for read() and display() for that particular book.

THEORY:

In Python, file handling and classes are essential concepts that help in structuring and managing data efficiently. Classes allow for the creation of custom data types that can store and manipulate attributes and behaviors. A class can have class variables (shared among all instances) and instance variables (specific to each instance). File handling enables reading from and writing to files, allowing programs to store and retrieve data. For instance, creating a class to store employee details or manage library books requires understanding both class definitions and file operations to keep track of records. With class methods, such as constructors and other instance methods, you can easily manipulate attributes like salary, designation, or book titles. In practice, menu-driven programs help in creating interactive systems to manage and process data dynamically.

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()
```

Write a menu driven program that keeps record of books and journals available in a library.
class book

```
# attributes of constructor - title, author name, price of book
# two functions - read and display()
# use logic that if you have list of books in a list, then you have a choice for read() and display() for that particular book.


class Book():
    def __init__(self,):
        self.title = ""
        self.author = ""
        self.price = ""
    def read(self):
        print()
        self.title = input("Enter title of book: ")
        self.author = input("Enter name of author of book: ")
        self.price = input("Enter price of book: ")
    def display(self):
        print("-----------")
        print("Title of book: ", self.title)
        print("Author of book: ", self.author)
        print("Price of book: ", self.price)
        print("-----------")

list_of_books = []
while True:
    print()
    print("************************************************************")
    ch = input("Enter your choice:\n1. Press 1 to enter a new book into the system.\n2. Press 2 to display all records.\n3. Press 3 to search book by title and print its details\n4. Press 4 to search book by author and print its details\n5. Press anything else to exit\nEnter your choice: ")
    if int(ch) == 1:
        obj = Book()
        obj.read()
        list_of_books.append(obj)
    elif int(ch) == 2:
        for i in list_of_books:
            i.display()
            print()
    elif int(ch) == 3:
        pat = input("Enter the search keyword in title: ")
        for i in list_of_books:
            if pat.lower() in i.title.lower():
                i.display()
    elif int(ch) == 4:
        pat = input("Enter the search keyword in author name: ")
        for i in list_of_books:
```

```
            if pat.lower() in i.author.lower():
                i.display()
    else:
        print("Invalid choice! Exiting the program")
        break
```

RESULTS:

```
Enter Employee id: 1
Enter Employee Name: Ravi
Enter Employee Designation: HR
Enter Employee Salary: 60000
Enter Employee id: 2
Enter Employee Name: Kabir
Enter Employee Designation: Manager
Enter Employee Salary: 80000
List of Employees and their details:  {'1': ['Ravi', 'HR', '60000']}
List of Employees and their details:  {'2': ['Kabir', 'Manager', '80000']}
Total count of employees:  2

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

************************************************************
Enter your choice:
1. Press 1 to enter a new book into the system.
2. Press 2 to display all records.
3. Press 3 to search book by title and print its details
4. Press 4 to search book by author and print its details
5. Press anything else to exit
```

```
Enter your choice:
1. Press 1 to enter a new book into the system.
2. Press 2 to display all records.
3. Press 3 to search book by title and print its details
4. Press 4 to search book by author and print its details
5. Press anything else to exit
Enter your choice: 1

Enter title of book: Harry Potter
Enter name of author of book: JK Rowling
Enter price of book: 800

************************************************************
Enter your choice:
1. Press 1 to enter a new book into the system.
2. Press 2 to display all records.
3. Press 3 to search book by title and print its details
4. Press 4 to search book by author and print its details
5. Press anything else to exit
Enter your choice: 5
Invalid choice! Exiting the program


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 12

OBJECTIVE: Demonstrate inheritance and develop code for given problem statements:

1. 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.
2. 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:

Inheritance in object-oriented programming allows one class to acquire the properties and behaviors of another. This helps in code reuse and establishes a relationship between classes. A base class (or parent class) provides common attributes and methods, while a derived class (or child class) can inherit these properties and methods, and also add its own unique features or override the inherited methods. For example, in a class hierarchy for students and courses, you can have a base class Course, and derived classes like Student and Department that inherit common attributes, while also having their own specific attributes. Inheritance supports the idea of modular and scalable code by extending functionality without modifying existing code.

CODE:

```
# 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: