Lab Assignment File

For

Computer Programming Basic with Python (CPBP)

by

Name: Ashadullah Danish

Roll No: 24901307

MTech First Year 2024-26

Centre for Artificial Intelligence

Submitted to

Dr. Diksha Ma'am

Centre for Artificial Intelligence

Dr. B R Ambedkar National Institute of Technology

Jalandhar – 144008 (Punjab)

CONTENTS

11.	Demonstrate Classes, Objects and Inheritance and develop code for given problem statements	19-Nov		
12.	Demonstrate Polymorphism, Error and Exception Handling and develop code for given problem statements	26-Nov		

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:

```
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))

Output:
52
<class 'int'>

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

```
print(type(b))
```

Output:

```
0x38
<class 'str'>
# Convert int to octal
b = oct(56)
print(b)
print(type(b))
```

Output:

```
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)

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)
```

Output:

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

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)
```

Output:
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)
```

Output:
False
True
True
False
False
True

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

Output:
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)
```

Output:

```
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

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")
```

Output:

```
Enter the input: HI my name is Danish

*** USING IN-BUILT LIBRARIES ***
Invalid Input

Enter the input: 5
*** USING IN-BUILT LIBRARIES ***
It's a number
```

```
''' 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")
```

Output:

*** 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

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

Output:

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

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

Output:

```
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()
```

output:

```
      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()
```

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

Output:

```
N = 3
Factorial is:  6
```

```
# 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")
```

Output:

```
Enter the year (1900-2101) to check whether leap year: 2000
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)
```

Output:

Enter the number: 8

Sum of series is: 2.7178571428571425

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

CODE:

```
# List slicing

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

list2 = [1,2,3,4,5,6,7,8]

print(list2[1:5])
```

Output:

```
[2, 3, 4, 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)
```

Output:

```
['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']

```
# 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)
```

Output:

[1, 9, 25, 49, 81]

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

Output:

1

a

abc

[2, 3, 4, 5]

8.9

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

Output:

index: 0

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

```
# 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))
```

Output:

```
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

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)
```

Output:
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])

```
# 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)
```

Output:
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])

```
# 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=" ")
```

Output:
1 2 3 1 2 5
After updation
1 2 6 1 2 5 ()

```
# 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)
```

Output:
()

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

```
# 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)
```

Output:
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: 3
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: Danish
New list is: [['Arav', 'MSC', 92], ['Student2', 'MBA', 99], ['Student3', 'MTech', 94], [Danish, 'BSC', 95]]

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

Output:
('Sam',)
(('Sam',),)
((('Sam',),),)
(((('Sam',),),),)
((((('Sam',),),),),)

```
# 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}')
```

Output:
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

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)
```

Output:
Enter radius: 1
Area is: 3.141592653589793
Circumference is: 6.283185307179586

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)
```

Output:
Enter the email address:cloud.data.danish@gmail.com
('cloud.data.danish', 'gmail.com')

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

CODE:

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

obj = square(6)
print(obj)
output:
36
```

```
# 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)
```

Output:
[1, 4, 9, 16, 25]
[1, 4, 9, 16, 25]

```
# 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))
```

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

```
# 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))
```

Output:
Enter num: 5
Number is Odd

```
# 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)
```

Output:

```
Enter age of person: 75
Enter principal amount: 10000
Enter time in years: 2
Simple Interest is:  2400.0
```

```
# 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)
```

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.

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)
```

Output:

```
<class 'set'>
True
```

{'banana', 'apple', 'orange', 'cherry'}
{'banana', 'cherry', 'orange', 'papaya', 'apple', 'mango'}
{'banana', 'cherry', 'orange', 'papaya', 'apple', 'papaya2', 'mango', 'mango2'}
{'banana', 'cherry', 'orange', 'papaya', 'apple', 'papaya2', 'mango'}
{'cherry', 'orange', 'papaya', 'apple', 'papaya2', 'mango'}

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, 'mango', 16, 49, 81, 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()
# 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))
```

Output:
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

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)
```

Output:

```
Zara
7
{'Name': 'Zara', 'Age': 8, 'Class': 'First'}
{'Name': 'Zara', 'Age': 8, 'Class': 'First', 'School': 'DPS'}
```

```
{'Age': 8, 'Class': 'First', 'School': 'DPS'}
# NameError: name 'dict1' is not defined. Did you mean: 'dict'?
# 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)
```

Output:

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

Output:

```
a   time
b   money
c   value

a
b
c
time
money
value
```

```
# 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)
```

Output:

```
{'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

CODE:

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

Output:
orl

```
# 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
```

Output:

```
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


# 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")
```

Output:

```
Hello
**hello***
2
True
True
7
-1
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(','))
```

Output:

```
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']
```

```
# 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)
```

Output:
A
AB
ABC
ABCD
ABCDE
ABCDEF

```
# 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
```

Output:
Welcome to Python

```
# 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)
```

Output:
Enter the string: Danish
Enter the encryption key: 3
QlnkloRq#Mlqgo

```
# 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'))

Output:
['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.']

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+"
```

Output:
Enter the string: Danish
Ninsh

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())
```

Output:
0 8 (0, 8)
9 17 (9, 17)
18 27 (18, 27)

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")
```

Output:
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

CODE:

```
# WAP that copies first 10 bytes of a binary file into another
with open(r"H:\hello world.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())
```

Output:
First 10 bytes of file1: b'Welcome to'
File2 contents:
b''
File2 contents after copying:
b'Welcome to

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(r"H:\\"+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")
```

Output:
Enter the file name: hello world.txt
Enter the character to be searched: e
Count of given character in file: 3

EXPERIMENT 11

OBJECTIVE: Demonstrate Classes, 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 organization 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 circumference 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 - Student details - name, roll no

Course - name, code, year and semester

Department – Name

THEORY:

In Python, classes are templates for creating objects. A class defines the properties (attributes) and behaviours (methods) that the objects created from it will have. Objects are instances of a class, representing specific data and functionality.

Key concepts:

1. Class Definition: A class is defined using the class keyword, followed by the class name and a colon. Inside the class, methods are defined using the def keyword.

2. Object Creation: An object is created by calling the class name as a function. This initializes an instance of the class, allowing access to its attributes and methods.

3. Attributes: Attributes are variables that belong to a class or an object. They can hold data specific to the object or shared by all instances of the class (class variables vs instance variables).

4. Methods: Methods are functions defined inside a class. They define the behaviors of the objects and can access and modify the object's attributes. The first parameter of a method is usually self, which refers to the current instance of the class.

5. Constructor (__init__ method): The __init__ method is a special method called a constructor, which is automatically invoked when an object is created. It initializes the object's attributes.

6. Inheritance: Inheritance allows a class to inherit attributes and methods from another class, facilitating code reuse and extension of functionality.

7. Encapsulation: Encapsulation involves bundling the data (attributes) and methods that operate on the data within a single unit (class). It also restricts access to certain details of an object, usually through private attributes and methods.

8. Polymorphism: Polymorphism allows methods to behave differently based on the object that is calling them. This can be achieved through method overriding in subclasses.

Classes and objects in Python enable object-oriented programming, allowing for better organization, reusability, and maintainability of code.

Inheritance is a fundamental concept in object-oriented programming that allows one class to inherit the attributes and methods of another class. In Python, inheritance enables the creation of a new class that is a modified version of an existing class. The new class, called the child class or subclass, inherits features from the existing class, called the parent class or superclass.

Key concepts of inheritance include:

1. Base and Derived Classes: The base class (or parent class) is the class from which attributes and methods are inherited, while the derived class (or child class) is the class that inherits these properties and can extend or modify them.

2. Method Overriding: A child class can override or replace methods from the parent class. This allows the child class to provide its own implementation of a method that exists in the parent class.

3. super() Function: The super() function is used in the child class to call methods from the parent class. It is commonly used in the constructor (__init__) to initialize attributes from the parent class.

Types of inheritance Single Inheritance**:** In single inheritance, a class (child class) inherits from only one class (parent class). This is the simplest form of inheritance, where the child class can access the attributes and methods of a single parent class.

Multiple Inheritance**:** In multiple inheritance, a class (child class) inherits from more than one class (parent classes). The child class combines the features and behaviours of multiple parent classes. While powerful, it can also lead to complexity, especially if two parent classes have methods or attributes with the same name.

Multilevel Inheritance**:** In multilevel inheritance, a class (child class) inherits from a parent class, and then another class (grandchild class) inherits from the child class. This forms a chain of inheritance, where each class inherits from the class above it in the hierarchy.

Hierarchical Inheritance**:** In hierarchical inheritance, multiple classes inherit from a single parent class. This allows the parent class to define common attributes and methods that are shared by all the child classes.

Hybrid Inheritance**:** Hybrid inheritance is a combination of two or more types of inheritance, such as multiple and multilevel inheritance. It combines different inheritance structures,

which may result in a more complex hierarchy. Hybrid inheritance can sometimes cause issues like the "diamond problem," which Python addresses using the method resolution order (MRO).

Access to Parent Class Attributes and Methods: The child class inherits all the attributes and methods of the parent class, but it can also add new attributes and methods or modify existing ones.

Inheritance promotes code reusability, allows for extending functionality, and makes it easier to maintain and modify code by organizing it into a hierarchical structure.

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

```
class Employee:
    # Class variable to keep track of the number of employees
    employee_count = 0

    def __init__(self, name, designation, salary):
        # Instance variables for employee details
        self.name = name
        self.designation = designation
        self.salary = salary

        # Increment the employee count whenever a new employee is created
        Employee.employee_count += 1

    @classmethod
    def get_employee_count(cls):
        """Returns the current number of employees."""
        return cls.employee_count

    def display_info(self):
        """Displays the information of the employee."""
        print(f"Name: {self.name}, Designation: {self.designation}, Salary: ${self.salary:,.2f}")
```

```
# Example usage
if __name__ == "__main__":
    # Creating instances of employees
    emp1 = Employee("Danish", "Software Engineer", 85000)
    emp2 = Employee("Nilay", "Project Manager", 95000)
    emp3 = Employee("Mohit", "Data Scientist", 90000)

    # Displaying employee information
    emp1.display_info()
    emp2.display_info()
    emp3.display_info()

    # Displaying the total number of employees
    print(f"Total number of employees: {Employee.get_employee_count()}")
```

Output:

Name: Danish, Designation: Software Engineer, Salary: $85,000.00

Name: Nilay, Designation: Project Manager, Salary: $95,000.00

Name: Mohit, Designation: Data Scientist, Salary: $90,000.00

Total number of employees: 3

#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 circumference of a circle with specified radius.

```
class Circle:
    # Class variable for pi
    pi = 3.14159

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

```
    def area(self):
        """Calculate the area of the circle."""
        return Circle.pi * (self.radius ** 2)

    def circumference(self):
        """Calculate the circumference of the circle."""
        return 2 * Circle.pi * self.radius

# Example usage:
def main():
    # Create an instance of Circle with a specified radius
    radius = float(input("Enter the radius of the circle: "))
    circle = Circle(radius)

    # Calculate area and circumference
    area = circle.area()
    circumference = circle.circumference()

    # Display results
    print(f"Area of the circle: {area:.2f}")
    print(f"Circumference of the circle: {circumference:.2f}")

if __name__ == "__main__":
    main()
```

Output:

```
Enter the radius of the circle:  4
Area of the circle: 50.27
Circumference of the circle: 25.13
```

#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.

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __str__(self):
        return f"Point({self.x}, {self.y})"

class Location:
    def __init__(self, location_x, location_y, destination_x, destination_y):
        self.location = Point(location_x, location_y)
        self.destination = Point(destination_x, destination_y)

    def print_reflection_on_x_axis(self):
        # Reflection of the destination on the x-axis means inverting the y-coordinate
        reflected_destination = Point(self.destination.x, -self.destination.y)
        print(f"Reflection of destination {self.destination} on the x-axis: {reflected_destination}")

# Example usage:
if __name__ == "__main__":
    location = Location(1, 2, 3, 4)  # Creating location at (1, 2) and destination at (3, 4)
    location.print_reflection_on_x_axis()  # Print the reflection of the destination on the x-axis
```

Output:

```
Reflection of destination Point(3, 4) on the x-axis: Point(3, -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

```
class Student:
    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no

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



class Course:
    def __init__(self, name, code, year, semester):
        self.name = 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})"



class Department:
    def __init__(self, name):
        self.name = name
        self.courses = []
```

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

    def __str__(self):
        course_list = ', '.join(course.name for course in self.courses)
        return f"Department(Name: {self.name}, Courses: [{course_list}])"


class Enrollment:
    def __init__(self):
        self.enrollments = {}

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

    def show_enrollments(self):
        for course, students in self.enrollments.items():
            print(f"{course}:")
            for student in students:
                print(f"\t{student}")


# Example usage:

# Create departments
cs_department = Department("Computer Science")
me_department = Department("Mechanical Engineering")
```

```
# Create courses
cs_course_1 = Course("Data Structures", "CS101", 1, 1)
cs_course_2 = Course("Algorithms", "CS102", 1, 2)
me_course_1 = Course("Thermodynamics", "ME101", 1, 1)

# Add courses to departments
cs_department.add_course(cs_course_1)
cs_department.add_course(cs_course_2)
me_department.add_course(me_course_1)

# Create students
student_1 = Student("Alice", "001")
student_2 = Student("Bob", "002")
student_3 = Student("Charlie", "003")

# Enroll students
enrollment = Enrollment()
enrollment.enroll_student(student_1, cs_course_1)
enrollment.enroll_student(student_2, cs_course_1)
enrollment.enroll_student(student_3, me_course_1)

# Show enrollments
enrollment.show_enrollments()

# Print department details
print(cs_department)
print(me_department)
```

Output:

Course(Name: Data Structures, Code: CS101, Year: 1, Semester: 1):

Student(Name: Alice, Roll No: 001)

Student(Name: Bob, Roll No: 002)

Course(Name: Thermodynamics, Code: ME101, Year: 1, Semester: 1):

Student(Name: Charlie, Roll No: 003)

Department(Name: Computer Science, Courses: [Data Structures, Algorithms])

Department(Name: Mechanical Engineering, Courses: [Thermodynamics])

EXPERIMENT 12

OBJECTIVE: Demonstrate polymorphism, error and exception handling and develop 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 KeyBoardInterrupt 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 is a core concept of object-oriented programming that allows methods or functions to operate on objects of different classes through a uniform interface. It enables a single method to behave differently based on the object it is acting upon, thereby supporting flexibility and extensibility in software design. Polymorphism can be classified into two main types:

1. Compile-time Polymorphism: Also known as static polymorphism, this type occurs when the method to be invoked is determined at compile-time. Examples include method overloading and operator overloading.

2. Runtime Polymorphism: Also known as dynamic polymorphism, this type occurs when the method call is resolved at runtime. It is typically achieved through method overriding in inheritance, where a subclass provides its specific implementation of a method defined in its superclass.

Polymorphism promotes code reusability, maintainability, and the ability to design systems that are scalable and adaptable to change.

Error handling refers to the process of anticipating, detecting, and resolving errors in a program to ensure its smooth execution. Errors can occur due to various reasons such as invalid user input, hardware failures, or logic issues in the code. Errors are broadly categorized into:

1. Compile-time Errors: These are syntax or semantic errors detected by the compiler, preventing the program from compiling successfully.

2. Runtime Errors: These errors occur during program execution, such as division by zero or accessing invalid memory locations.

3. Logical Errors: These occur due to incorrect implementation of algorithms or logic, leading to unintended results.

Effective error handling involves identifying potential error-prone sections of code and incorporating mechanisms to handle errors gracefully, ensuring minimal disruption to program functionality.

Exception handling is a specialized mechanism in programming used to manage runtime errors, known as exceptions, in a structured manner. It allows developers to detect errors, handle them without crashing the program, and ensure normal program flow is restored. Most modern programming languages provide constructs for exception handling, typically through:

1. Try Block: Code that may throw an exception is placed inside the try block.

2. Catch Block: Handles specific types of exceptions. Multiple catch blocks can be used for different exception types.

3. Finally Block: Optional block executed after try and catch blocks, regardless of whether an exception was thrown or caught. It is typically used for cleanup operations.

4. Throw Statement: Used to explicitly throw an exception when a specific error condition occurs.

Exception handling improves program robustness, facilitates debugging, and ensures resource management by preventing resource leaks during error scenarios.

```
#Demonstrate operator overloading
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        """Overload the + operator to add two vectors."""
        if isinstance(other, Vector):
            return Vector(self.x + other.x, self.y + other.y)
        return NotImplemented

    def __sub__(self, other):
        """Overload the - operator to subtract two vectors."""
        if isinstance(other, Vector):
            return Vector(self.x - other.x, self.y - other.y)
        return NotImplemented

    def __str__(self):
        """Return a string representation of the vector."""
        return f"Vector({self.x}, {self.y})"

    def __repr__(self):
```

```
        """Return an unambiguous string representation of the vector."""
        return f"Vector({self.x}, {self.y})"

# Example usage:
if __name__ == "__main__":
    v1 = Vector(2, 3)
    v2 = Vector(4, 5)

    print(f"Vector 1: {v1}")
    print(f"Vector 2: {v2}")

    v3 = v1 + v2
    print(f"v1 + v2: {v3}")

    v4 = v1 - v2
    print(f"v1 - v2: {v4}")
```

Output:

```
Vector 1: Vector(2, 3)
Vector 2: Vector(4, 5)
v1 + v2: Vector(6, 8)
v1 - v2: Vector(-2, -2)
```

```
#Demonstrate Method Overriding
# Base class
class Animal:
    def speak(self):
        return "Some generic sound"

# Derived class
class Dog(Animal):
```

```
    def speak(self):
        return "Woof!"

# Another derived class
class Cat(Animal):
    def speak(self):
        return "Meow!"

# Function to demonstrate method overriding
def demonstrate_method_overriding():
    # Create instances of Dog and Cat
    generic_animal = Animal()
    dog = Dog()
    cat = Cat()

    # Call the speak method from each instance
    print(f'Animal: {generic_animal.speak()}')  # Outputs: Some generic sound
    print(f'Dog: {dog.speak()}')               # Outputs: Woof!
    print(f'Cat: {cat.speak()}')               # Outputs: Meow!

# Run the demonstration
if __name__ == "__main__":
    demonstrate_method_overriding()
```

Output:

```
Animal: Some generic sound
Dog: Woof!
Cat: Meow!
```

#WAP to handle the divide by zero exception

```
def divide_numbers():
    try:
        # Input from user
        numerator = float(input("Enter the numerator: "))
        denominator = float(input("Enter the denominator: "))

        # Attempting to perform the division
        result = numerator / denominator

        print(f"The result of {numerator} divided by {denominator} is {result}.")

    except ZeroDivisionError:
        print("Error: You cannot divide by zero. Please provide a non-zero denominator.")
    except ValueError:
        print("Error: Please enter valid numbers.")

# Call the function to execute
divide_numbers()
```

Output:

```
Enter the numerator:  5
Enter the denominator:  0
Error: You cannot divide by zero. Please provide a non-zero denominator.
```

```
#Demonstrate Raise Exceptions, Instantiating Exceptions, assertion
class CustomException(Exception):
    """Custom exception class for demonstration."""
    pass
```

```
def divide_numbers(numerator, denominator):
    """Divides two numbers and raises exceptions for error cases."""
    if denominator == 0:
        raise CustomException("Denominator cannot be zero.")
    return numerator / denominator

def check_positive_number(num):
    """Checks if a number is positive and raises an exception if not."""
    if num < 0:
        raise ValueError("The number must be positive.")
    return True

def assert_positive_number(num):
    """Uses assert to ensure the number is positive."""
    assert num >= 0, "Assertion Error: The number must be positive."

def main():
    # Demonstration of dividing numbers
    try:
        result = divide_numbers(10, 0)
        print(f"Result: {result}")
    except CustomException as e:
        print(f"Caught a custom exception: {e}")

    # Demonstration of checking a positive number
    try:
        check_positive_number(-5)
    except ValueError as e:
        print(f"Caught a value error: {e}")
```

```
    # Demonstration of assert
    try:
        assert_positive_number(-3)
    except AssertionError as e:
        print(f"Caught an assertion error: {e}")

    print("Code executed properly")

if __name__ == "__main__":
    main()
```

Output:

Caught a custom exception: Denominator cannot be zero.

Caught a value error: The number must be positive.

Caught an assertion error: Assertion Error: The number must be positive.

Code executed properly

#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

```
def main():
    try:
        # Prompt the user to enter a number
        user_input = input("Please enter a number: ")

        # Check if the input is empty
        if user_input.strip() == "":
            raise KeyboardInterrupt("No input was provided, raising KeyboardInterrupt.")

        # Convert input to a float (or int) to handle numerical input
        number = float(user_input)
```

```
        # Calculate the square of the number
        square = number ** 2
        # Print the square of the number
        print(f"The square of {number} is {square}")
    except KeyboardInterrupt as e:
        print(e)
    except ValueError:
        print("Invalid input! Please enter a valid number.")
if __name__ == "__main__":
    main()
```

Output:

```
Please enter a number:  u
Invalid input! 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 NaturalNumberGenerator:
    def __init__(self):
        self.current = 0  # Start from 0

    def __iter__(self):
        return self
    def __next__(self):
        if self.current < 20:  # Limit the output to the first 20 natural numbers
            self.current += 1
            return self.current
        else:
            raise StopIteration  # Raise StopIteration after 20 numbers
if __name__ == "__main__":
    generator = NaturalNumberGenerator()
```

```
    try:
        for number in generator:
            print(number)
    except StopIteration:
        print("Stopped iteration after displaying the first 20 natural numbers.")
```

Output:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

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

```
import random

# Define a custom exception class

class BelowThresholdError(Exception):
```

```
    """Exception raised when generated number is below the threshold of 0.1."""
    def __init__(self, value):
        self.value = value
        super().__init__(f"Generated number {value} is below the allowed threshold of 0.1.")

# Function to generate a random number and check it
def generate_random_number():
    # Generate a random float between 0 and 1
    number = random.random()  # This generates a float in the range [0.0, 1.0)
    print(f"Generated number: {number}")
    # Raise the custom exception if the number is below 0.1
    if number < 0.1:
        raise BelowThresholdError(number)
    return number
# Main execution block
if __name__ == "__main__":
    try:
        generate_random_number()
    except BelowThresholdError as e:
        print(e)
```

Output:

```
Generated number: 0.0813264541385399
Generated number 0.0813264541385399 is below the allowed threshold of 0.1.
```