**Lab Experiments**

of

**CPBP LAB**

For the Degree of

**MASTER OF TECHNOLOGY**

**ARTIFICIAL INTELLIGENCE**



Submitted By: Submitted To:

**Shashank Dr. Diksha Kumari**

**(24901323)**

**CENTER FOR ARTIFICIAL INTELLIGENCE**

**DR. B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY**

**JALANDHAR , PUNJAB (144011), 2024-25**

# **CONTENTS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S. NO. | EXPERIMENT | DATE OF SUBMISSION | PAGE NO. | REMARKS |
| 1. | Install Python and write basic programs to explore its syntax and functionality | 13-08-2024 | 4 |  |
| 2. | Demonstrate python operators and develop code for given problem statements | 20-08-2024 | 6 |  |
| 3. | Demonstrate conditional and loop statements and develop code for given problem statements | 27-08-2024 | 10 |  |
| 4. | Demonstrate list operations and develop code for given problem statements | 03-09-2024 | 16 |  |
| 5. | Demonstrate arrays and tuples and develop code for given problem statements | 10-09-2024 | 20 |  |
| 6. | Demonstrate functions and modules and develop code for given problem statements | 24-09-2024 | 27 |  |
| 7. | Demonstrate Set operations and develop code for given problem statements | 01-10-2024 | 30 |  |
| 8. | Demonstrate dictionary operations and develop code for given problem statements | 22-10-2024 | 34 |  |
| 9. | Demonstrate strings and its related operations and develop code for given problem statements | 05-11-2024 | 37 |  |
| 10. | Demonstrate file handling and develop code for given problem statements | 12-11-2024 | 45 |  |
| 11. | Demonstrate file handling and develop code for given problem statements | 19-11-2024 | 47 |  |
| 12. | Demonstrate inheritance and develop code for given problem statements | 26-11-2024 | 58 |  |

# **EXPERIMENT 1**

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

**THEORY**:

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

**INSTALLATION STEPS:**

Download Python:

Go to python.org.

Download the latest Python 3 version for your OS.

Install Python:

Run the installer.

Select "Add Python to PATH" and click Install Now.

Verify Installation:

Open a terminal and type: python --version (Windows/macOS/Linux).

Confirm the version is displayed.

Optional: Install an IDE like PyCharm, VS Code, or use IDLE (bundled with Python).

**CODE**:

# Simple Program to perform arithmetic operations on two numbers

a = 7

b = 2

print("sum: ", a+b)

print("diff: ", a-b)

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

print("div: ", a/b)

print("mod: ", a%b)

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

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

**RESULTS**:



# **EXPERIMENT 2**

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

1. Datatype Conversion:
   1. convert char to int, and find octal, hex value of given value
   2. convert string to tuple, set and list
2. Types of operators:
   1. perform arithmetic operations on 2 numbers
   2. 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))

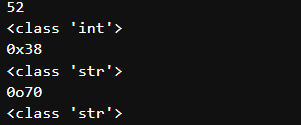
# Convert int to octal

b = oct(56)

print(b)

print(type(b))

Output:-



1. 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="")

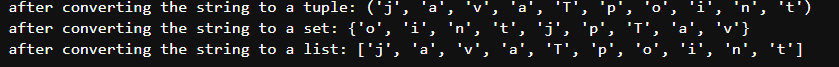
print(y)

y = list(x)

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

print(y)

Output:-



1. 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)

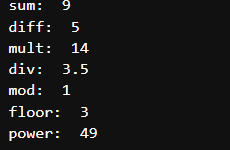
print("div: ", a/b)

print("mod: ", a%b)

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

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

Output:-



1. 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]

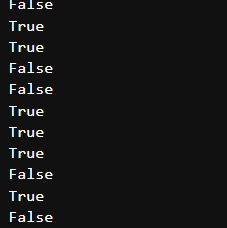
y3=[1,2,3]

print(x1 is not y1)

print(x2 is y2)

print(x3 is y3)

Output:-



# **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
      1. Nested loop to print given pattern

\*

\* \*

\* \* \*

\* \* \* \*

* 1. Demonstrate while loop inside for loop
  2. WAP to print the pattern

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

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

**THEORY**:

Conditional Statements

Conditional statements are used to execute a block of code based on whether a condition evaluates to True or False. These are also known as decision-making statements. The main conditional statements in Python are:

1. **if statement**: Executes a block of code if the condition is true.
2. **if-else statement**: Executes one block if the condition is true, and another block if the condition is false.
3. **if-elif-else statement**: Used to test multiple conditions sequentially.

Looping Statements

Loops are used to repeat a block of code multiple times. Python provides the following looping constructs:

1. **for loop**: Iterates over a sequence (e.g., list, tuple, string).
2. **while loop**: Repeats a block of code as long as a condition evaluates to True.
3. **Nested loops**: A loop inside another loop, used for multi-dimensional tasks like generating patterns.

**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")

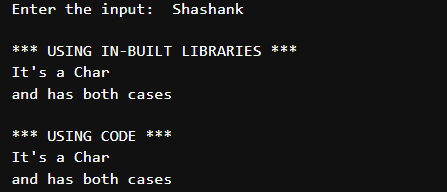
else:

print("Invalid Input")

else:

print("Invalid Input")

Output:-



# 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")

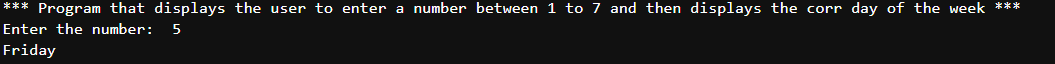
if num == 7:

print ("Sunday")

else:

print("Incorrect number")

Output:-



# 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 = ["Shashank", "Mohit", "Nilanshu"]

for name in names:

count = 0

while(count<5):

print(name, end=' ')

count+=1

print()

Output:-



# WAP to print the pattern

for i in range(1, 6):

for k in range(1, 6-i):

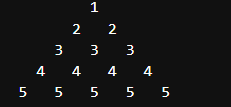
print(" ", end=" ")

for j in range(1,i+1):

print(i, " ", end=" ")

print()

Output:-



# 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

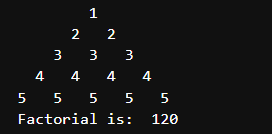
fact = 1

for i in range(2,n+1):

fact \*= i

print("Factorial is: ", fact)

Output:-



# 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:-



# 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:-



# **EXPERIMENT 4**

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

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

**THEORY**:

**Lists in Python**

A **list** is a mutable, ordered collection of elements. Python provides numerous operations for handling lists efficiently.

1. **List Slicing and Cloning**:
   * Slicing (start:stop:step) extracts portions of a list.
   * Cloning creates a shallow copy of a list using slicing ([:]) or the list() method.
2. **List Methods**:
   * **insert()**: Inserts an element at a specific index.
   * **append()**: Adds an element to the end.
   * **extend()**: Adds elements from another iterable.
   * **reverse()**: Reverses the list in place.
   * **reversed()**: Returns a reversed iterator of the list.
   * **remove()**: Removes the first occurrence of a value.
   * **pop()**: Removes and returns an element by index (default: last).
3. **List Comprehension**:
   * A concise method to create lists using loops and conditions in a single line.
4. **Looping in Lists**:
   * Use for or while loops to iterate through lists.
5. **Calculations**:
   * Use sum() to compute the total and divide by len() for the average.

**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 = ['S', 'H', 'A', 'S', 'H', 'A', 'N', 'K',]

print(List)

Sliced\_list = List[:-6]

print("Sliced: ", Sliced\_list)

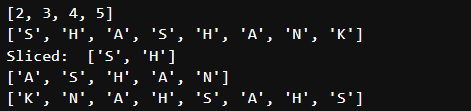
l2 = List[-6:-1]

print(l2)

l3 = List[::-1]

print(l3)

Output:-



# 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:-



# Looping in lists

ls = [1,'a',"shashank",[2,3,4,5],8.9]

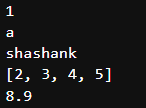
i = 0

while i < (len(ls)):

print(ls[i])

i+=1

Output:-



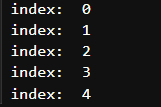
# Program to print index of values in a list

l1 = [1,2,3,4,5]

for i in range(len(l1)):

print("index: ", i)

Output:-



# 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:-



# **EXPERIMENT 5**

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

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

**THEORY**:

**Arrays and Tuples in Python**

**1. Arrays**  
Arrays are data structures that store elements of the same data type in contiguous memory locations. Python provides arrays through the array module, supporting basic operations like insertion, updating, slicing, and appending.

* **array()**: Used to create an array.
* **insert(index, value)**: Adds an element at a specific index.
* **append(value)**: Adds an element to the end of the array.
* **Slicing**: Access a subset of elements using start:stop:step.
* **Updating Elements**: Modify values by accessing their index.

**2. Tuples**  
Tuples are immutable, ordered collections of elements. Unlike lists, tuples cannot be modified after creation, which makes them ideal for storing fixed data.

* **Creating Tuples**: Tuples can be created using parentheses () or directly from other data types like strings and lists.
* **Mixed Data Types**: Tuples can hold elements of different types, including nested structures like lists and other tuples.

**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=" ")

Output:-



# 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)

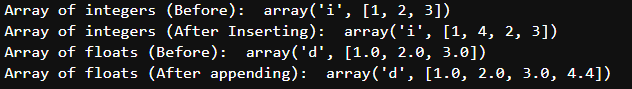
b = arr.array('d', [1,2,3])

print("Array of floats (Before): ", b)

b.append(4.4)

print("Array of floats (After appending): ", b)

Output:-



# 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: ")

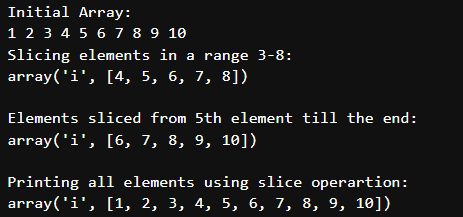
print(sliced\_array)

sliced\_array=a[:]

print("\nPrinting all elements using slice operartion: ")

print(sliced\_array)

Output:-



# Array Updation

import array

arr = array.array('i', [1,2,3,1,2,5])

for i in range(0,6):

print(arr[i], end = " ")

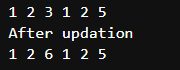
print("\nAfter updation")

arr[2]=6

for i in range(0,6):

print(arr[i], end=" ")

Output:-



# Create empty tuple:

tuple1 = ()

print(tuple1)

Output:-



# Create tuple using string:

tuple1 = ('Hello', 'Shashank')

print(tuple1)

Output:-



# Create tuple using list:

list1 = ['Hello', 'Shashank']

print(tuple(list1))

Output:-



# Create a tuple using built-in function:

tuple1 = tuple('Shashank')

print(tuple1)

Output:-



# Creating a tuple with mixed datatypes

tuple1 = (5, 'shashank', 7, 'JFidsof')

print(tuple1)

Output:-



# Nested tuples

t1 = (1,2,3)

t2 = ('a', 'b', 'c')

t3 = (t1, t2)

print(t3)

Output:-



# Program to demonstrate use of nested tuples

Toppers = (("shashank", 97, "Mtech"), ("raghav", 87, "BCA"))

for i in Toppers:

print(i)

Output:-



# WAP that has a nested list to store toppers details. Edit the details and reprint the details.

# Eg - l1 = ["Shashank", "Mtech", 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: ")

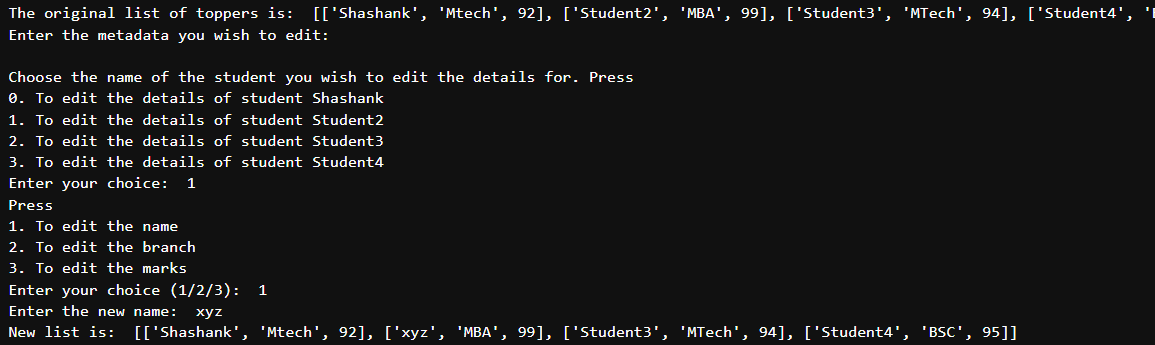
l1[ch1][2] = new\_name

else:

print("Wrong choice entered!")

print("New list is: ", l1)

Output:-



# Creating a tuple using Loop

t1 = ('Shashank')

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

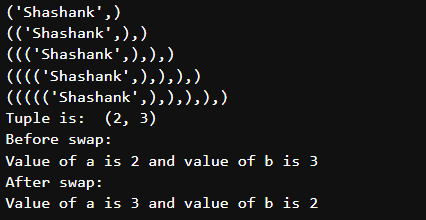
print(f'Value of a is {a} and value of b is {b}')

print("After swap: ")

(a, b) = (b, a)

print(f'Value of a is {a} and value of b is {b}')

Output:-



# 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

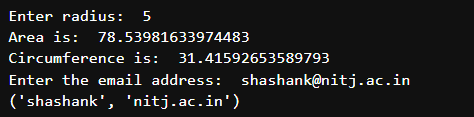
email = input("Enter the email address: ")

email = email.split("@")

email\_tuple = tuple(email)

print(email\_tuple)

Output:-



# **EXPERIMENT 6**

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

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

**THEORY**:

Functions in Python

Functions are reusable blocks of code designed to perform a specific task. Functions improve modularity and reduce code redundancy.

**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)

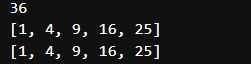
print(obj)

# Pass by value

obj = square([1,2,3,4,5])

print(obj)

Output:-



# 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:-



# 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:-



# 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

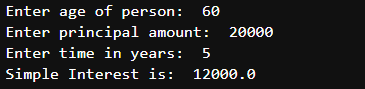
else:

r=10

si = principal\*r\*time/100

print("Simple Interest is: ", si)

Output:-



# 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)

Output:-



# **EXPERIMENT 7**

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

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

**THEORY**:

**Sets in Python**

A **set** is an unordered, mutable collection of unique elements. Python sets are used for membership tests, removing duplicates, and performing mathematical set operations like union, intersection, and difference.

**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")

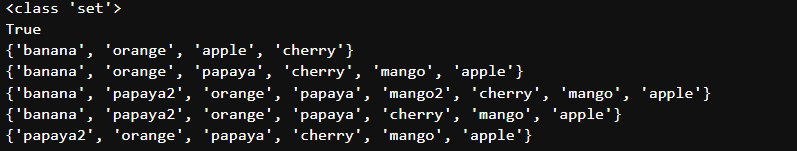
print(thisset)

# Remove item using discard()

thisset.discard("banana")

print(thisset)

Output:-



# 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")

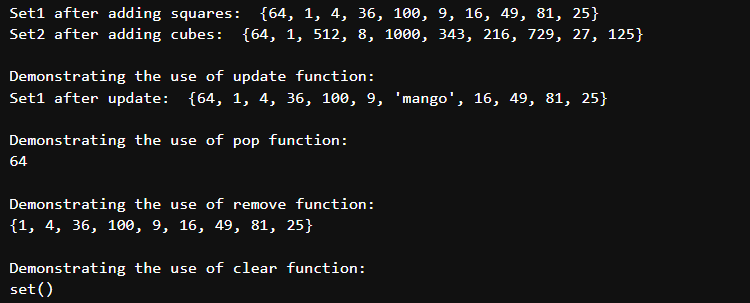
print(set1)

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

set1.clear()

print(set1)

Output:-



# 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))

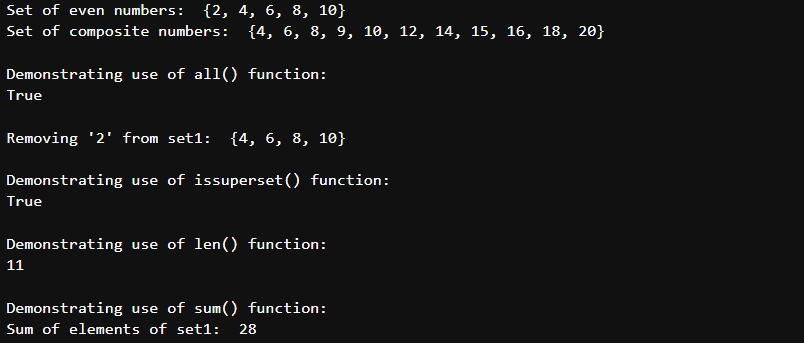
print("\nDemonstrating use of len() function: ")

print(len(set2))

print("\nDemonstrating use of sum() function: ")

print("Sum of elements of set1: ", sum(set1))

Output:-



# **EXPERIMENT 8**

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

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

**THEORY**:

A **dictionary** in Python is a collection of key-value pairs, where each key must be unique. Dictionaries are mutable, unordered, and allow for fast retrieval and modification of data. They are often used to store and manage related data in a structured way.

**Dictionary Operations**

1. **Accessing Values in a Dictionary**:  
   You can access a value in a dictionary by referring to its key, e.g., dict[key]. You can also use the get() method to avoid errors when a key is missing.
2. **Updating a Dictionary**:  
   You can add or modify an entry by assigning a value to a key, e.g., dict[key] = value.
3. **Adding New Values**:  
   You can add new key-value pairs using the same syntax used for updating, or use the update() method to merge two dictionaries.
4. **Deleting Particular Entries**:  
   To remove a specific entry, use del dict[key]. Alternatively, you can use pop() to remove a key-value pair and return the value.
5. **Clearing the Whole Dictionary**:  
   To remove all entries from a dictionary, use dict.clear().
6. **Deleting the Whole Dictionary**:  
   To delete the dictionary entirely, use del dict.

**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()

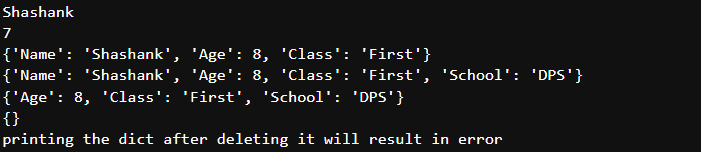
print(dict1)

# Delete whole dict

del dict1

print(dict1)

Output:-



# 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:

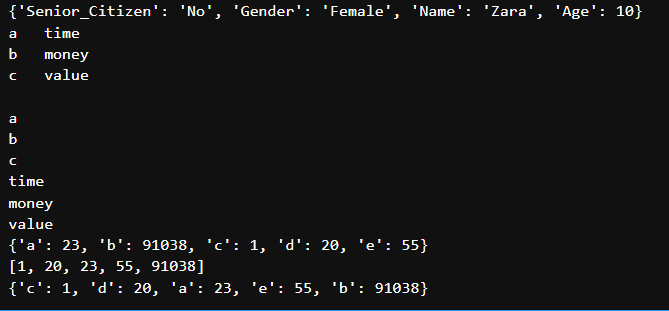
for j in dict1.keys():

if dict1.get(j) == i:

dict2[j] = i

print(dict2)

Output:-



# **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 –
     1. capitalize(), center(), count(), endswith(), startswith(), find(), index(), rfind(), rindex(), isalnum(), isalpha(), isdigit(), islower(), isupper(), len(), etc.
     2. WAP to print following pattern

A

AB

ABC

ABCD

ABCDE

ABCDEF

* + 1. WAP using while loop to iterate a given string
    2. WAP that encrypts a message by adding a key value to every character
    3. WAP that uses split function to split a multi-line string
    4. WAP that accepts a string from user and re-displays the same string after removing vowels
  1. Regular Expressions
     1. WAP to find patterns that begin with one or more characters followed by space and followed by one or more digits
     2. 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**:

**1) Slicing:**

* Slicing in Python extracts a portion of a string using string[start:end]. The start index is inclusive, and the end index is exclusive. For example, "World"[1:4] gives "orl".

**2) Power Calculation Without Formatting:**

* To display powers of a number, use a loop to raise the number to different powers. This avoids using formatting characters like % or f-string.

**3) String Methods:**

* Python strings come with built-in methods for manipulation:
  + capitalize(), center(), count(), endswith(), startswith(), find(), index(), isalnum(), isalpha(), isdigit(), islower(), isupper(), and len() are commonly used methods.

**4) While Loop to Iterate Through a String:**

* A while loop can iterate over a string, accessing each character using its index.

**5) Encrypting a Message:**

* Message encryption can be done by shifting the ASCII value of each character by a specified key, effectively "encrypting" the message.

**CODE**:

a = "HelloWorld"

# Get the characters from o in World to but not included d in "World"

print(a[-4:-1])

Output:-



# 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()

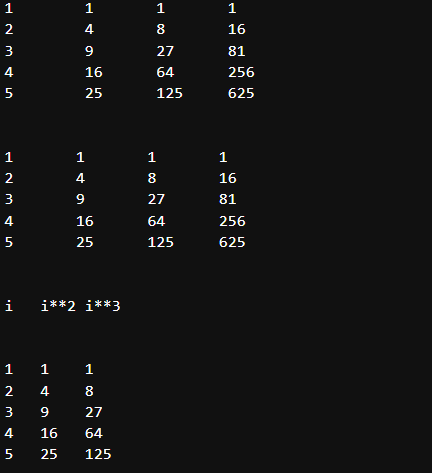
i = 1

while i<=5:

print("%-4d%-5d%-6d"%(i, i\*\*2, i\*\*3))

i+=1

Output:-



# 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())

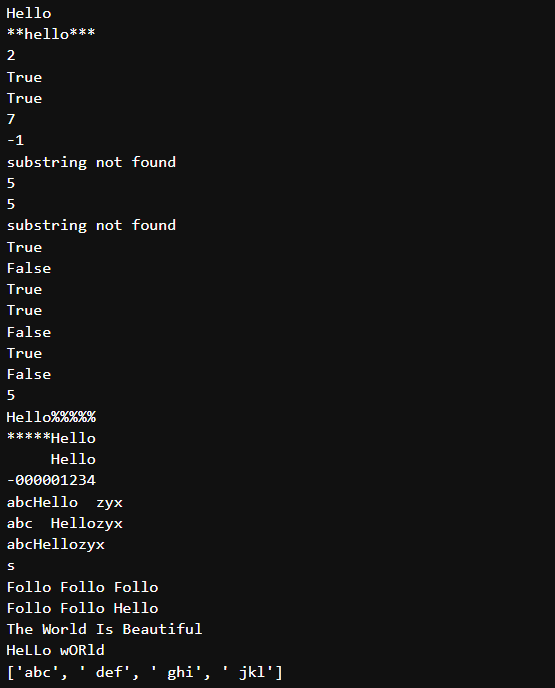
s = "hEllO WorLD"

print(s.swapcase())

s = "abc, def, ghi, jkl"

print(s.split(','))

Output:-



# WAP to print the pattern

for i in range(1, 7):

ch = 'A'

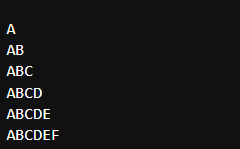
print()

for j in range(1, i+1):

print(ch, end="")

ch = chr(ord(ch)+1)

Output:-



# 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:-



# 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: "))

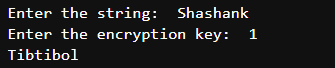
new\_s = ""

for i in s:

new\_s += chr(ord(i)+key)

print(new\_s)

Output:-



# 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:-



# 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:-



# Patterns that begin with one or more characters followed by space and followed by one or more digits

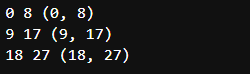
import re

matches = re.finditer(pattern, "LXI 2013,VXI 2015,VDI 20104,Maruti Suzuki Cars available with us")

for match in matches:

print(match.start(), match.end(), match.span())

Output:-



# 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:-



# **EXPERIMENT 10**

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

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

**THEORY**:

**File Handling in Python**: File handling in Python refers to the process of reading from, writing to, and performing other operations on files. Python provides built-in functions to handle files, such as open(), read(), write(), and others. Files can be opened in various modes like 'r' (read), 'w' (write), 'a' (append), 'b' (binary), etc. You can also manipulate file paths using the os and os.path modules.

**CODE**:

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

with open("D:/Subjects/Python/.ipynb\_checkpoints/shashankfile1.txt", "rb") as f:

a = f.read(10)

print("First 10 bytes of file1: ", a)

with open("D:/Subjects/Python/.ipynb\_checkpoints/shashankfile2.txt", "wb+") as f2:

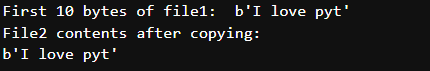
f2.write(a)

f2.seek(0) # Move the cursor to the beginning of the file to read it

print("File2 contents after copying:")

print(f2.read())

Output:-



# 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

import os

os.mkdir("new\_dir")

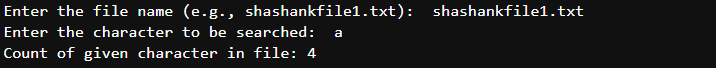
# WAP that changes curr dur to newly created dir new\_dir

os.chdir("new\_dir")

# WAP to delete new\_dir

os.rmdir("new\_dir")

Output:-



# **EXPERIMENT 11**

**OBJECTIVE:** Classes, objects and inheritance

11. Classes, objects and inheritance

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.

1. Inheritance
   * 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 parƟcular department. Classes are -
        1. Student details - name, roll no
        2. Course - name, code, year and semester
        3. Department – Name

**THEORY**:

**Short Theory on File Handling and Object-Oriented Programming**

**File Handling in Python**: File handling allows you to interact with files—whether reading, writing, or performing other file operations. The open() function is commonly used to open a file, and it supports various modes like reading ('r'), writing ('w'), and binary ('b'). File handling operations are essential for storing and manipulating persistent data.

**Object-Oriented Programming (OOP) in Python**: OOP enables you to model real-world entities using classes and objects. Key concepts include:

* **Class**: A blueprint for creating objects (instances).
* **Object**: An instance of a class containing data and methods.
* **Class Variables**: Shared variables that are the same for all instances of a class.
* **Instance Variables**: Variables that are specific to each instance.
* **Methods**: Functions defined within a class that operate on data.

**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()

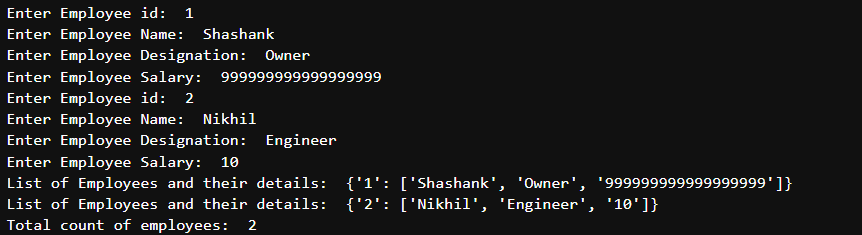
count\_of\_emp += 1

e1.displayDetails()

e2.displayDetails()

e2.displayCount()

Output:-



# 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)

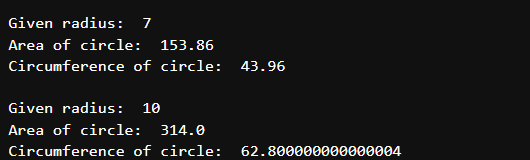
c2.calcArea()

c2.calcCircum()

c1.printDetails()

c2.printDetails()

Output:-



# 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():

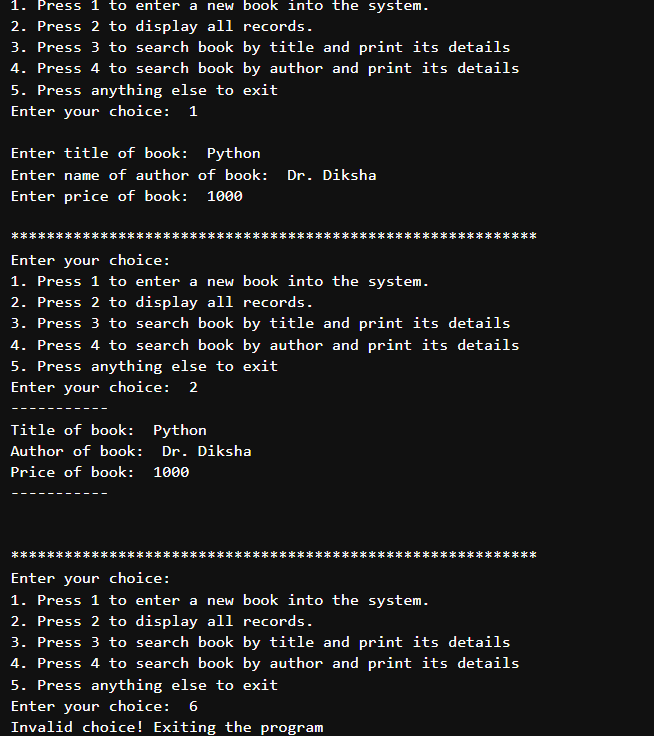
i.display()

else:

print("Invalid choice! Exiting the program")

break

Output:-



# 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)

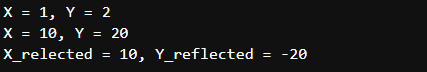
destination = Location(10, 20)

location.display\_point()

destination.display\_point()

destination.reflection()

Output:-



# 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!")

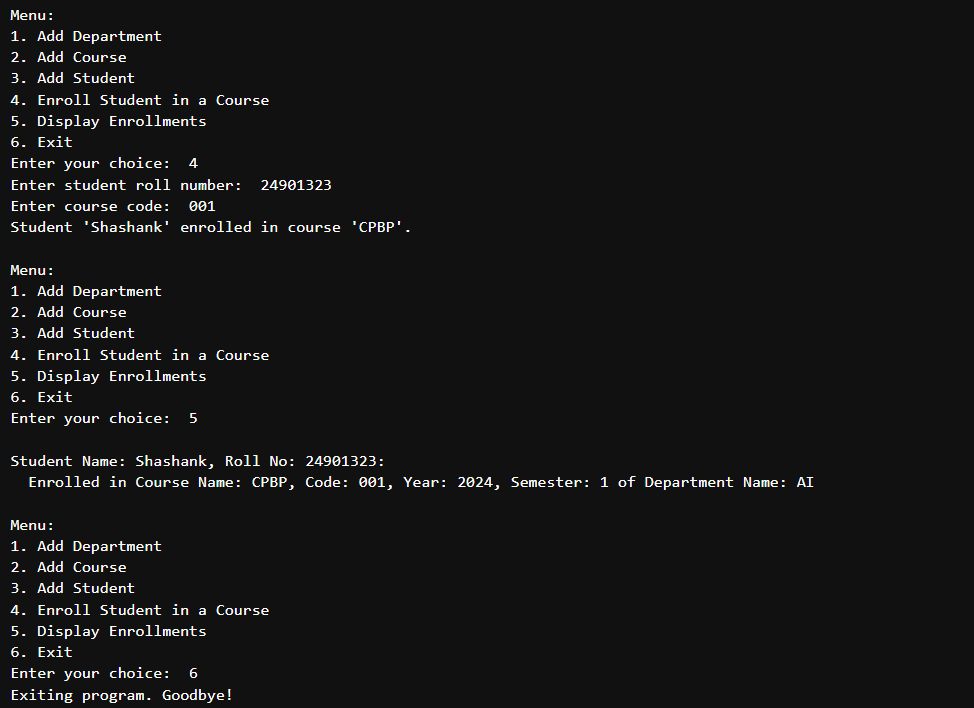
break

else:

print("Invalid choice. Please try again.")

Output:-





**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.

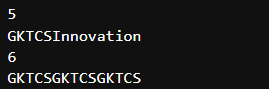
# Operator overloading

print(3+2)

print("GKTCS" + "Innovation")

print(3\*2)

print("GKTCS" \* 3)



# Method Overriding

class Employee:

def message(self):

print("In Employee Class")

class Company(Employee):

pass

class Company2(Employee):

def message(self):

print("In Company Class")

emp = Employee()

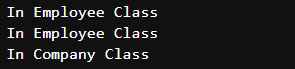
emp.message()

comp = Company()

comp.message()

comp2 = Company2()

comp2.message()



# WAP to handle the divide by zero exception

try:

num = int(input("Enter the numerator: "))

deno = int(input("Enter the denominator: "))

q = num / deno

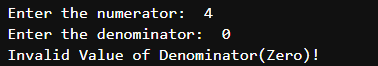
print('Quotient is: ', q)

except ZeroDivisionError:

print("Invalid Value of Denominator(Zero)!")

except ValueError:

print("Values must be integers!!")



# Raise Exceptions

try:

num = 10

print(num)

raise ValueError

except:

print("Exception occurred")



# Instantiating Exceptions

try:

num = 10

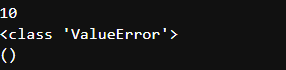
print(num)

raise ValueError

except Exception as errorobj:

print(type(errorobj))

print(errorobj.args)



# Program to demonstrate the use of assert keyword

def calculate\_average(marks):

# Ensure the input list is not empty

assert len(marks) > 0, "The list of marks cannot be empty!"

return sum(marks) / len(marks)

try:

marks\_list = [85, 90, 78, 92]

print("Average Marks:", calculate\_average(marks\_list))

empty\_list = []

print("Average Marks:", calculate\_average(empty\_list)) # This will trigger the assertion

except AssertionError as e:

print("AssertionError:", e)



# WAP that prompts the user to enter a number and prints the square of that number.

# If no number is entered, then a KeyBoardInterrupt is generated

try:

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

print(num \* num)

except KeyboardInterrupt:

print("Exception occurred")



# WAP which infinitely prints natural numbers.

# Raise the StopIterationException after displaying first 20 numbers to exit from the program

i = 1

try:

while True:

print(i)

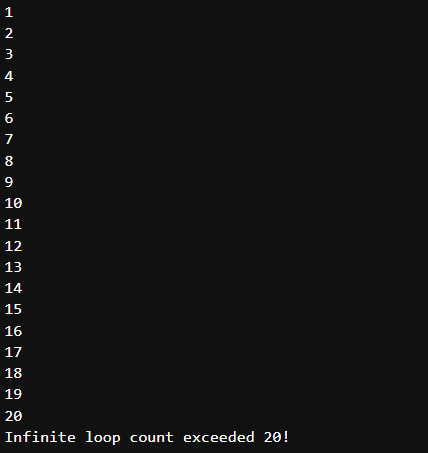
if i >= 20:

raise StopIteration

i += 1

except StopIteration:

print("Infinite loop count exceeded 20!")



# WAP that randomly generates a number and raises a UserDefined exception if the number is below 0.1

import random

class UserDefinedException(Exception):

pass

a = random.random()

try:

print(a)

if a < 0.1:

raise UserDefinedException

except UserDefinedException:

print("UserDefinedException raised as the value is less than 0.1")

