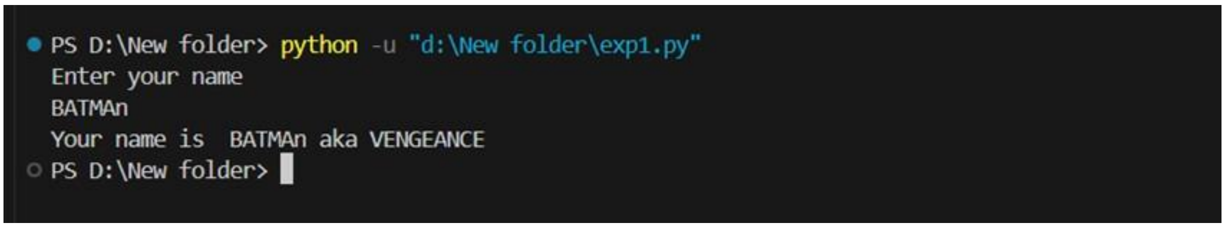
EXPERIMENT NO 1

Aim: Basic data types and operators: Create a program that prompts the user for their name and age and prints a personalized message.

Source Code:

```
name = input("Enter your name\n")

print("Your name is ",name,"aka VENGEANCE")
```

Output:



```
PS D:\New folder> python -u "d:\New folder\exp1.py"
Enter your name
BATMAn
Your name is  BATMAn aka VENGEANCE
PS D:\New folder>
```

Viva Questions:

Question: What is the purpose of the print() function in Python?

Answer: The print() function is used to display output on the console. It takes one or more arguments and prints them to the standard output (usually the console window)

Question: How can you print multiple values on the same line using the print() function?

Answer: You can print multiple values on the same line by separating them with commas within the print() function. For example, print("Hello", "world!") will output "Hello world!".

Question: What is the significance of the end parameter in the print() function?

Answer: The end parameter allows you to specify the character(s) that should be printed at the end of the output. By default, end is set to "\n" (newline), but you can change it to any other character(s) as needed.

Question: How can you prevent the print() function from automatically adding a newline character at the end of the output?

Answer: You can prevent print() from adding a newline character by setting the end parameter to an empty string "". For example, print("Hello", end="") will print "Hello" without a newline.

Question: How do you format output using the print() function in Python?

Answer: You can format output using f-strings or the format() method. For example, name = "Alice"; age = 30; print(f"My name is {name} and I am {age} years old.") will output "My name is Alice and I am 30 years old.

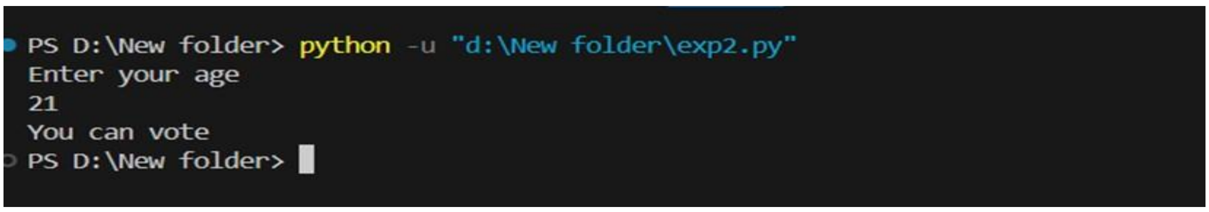
EXPERIMENT NO 2

Aim: Conditional statements: Create a program that prompts the user for their age and tells them if they can vote in the next election.

Source Code:

```
age = int(input("Enter your age\n"))
if age >=18 :
 print("You can vote")
else :
 print("You cannot vote")
```

Output:



```
PS D:\New folder> python -u "d:\New folder\exp2.py"
Enter your age
21
You can vote
PS D:\New folder> 
```

Viva Questions:

Question: What is the purpose of the if-else statement in Python?

Answer: The if-else statement allows for conditional execution of code. It evaluates a condition and executes one set of statements if the condition is true, and another set if the condition is false.

Question: How do you write a basic if-else statement in Python?

Answer: A basic if-else statement in Python is written as follows:

"If a certain condition is true, perform one action; otherwise, perform another

action." **Question: Can you have multiple conditions in an if-else statement?**

Answer: Yes, you can have multiple conditions using the elif (else if) statement. It allows you to specify additional conditions to be checked if the initial if condition is false.

Question: How can you nest if-else statements in Python?

Answer: You can nest if-else statements by placing one if-else statement inside another. This allows for more complex conditional logic.

Question: What is the ternary conditional operator in Python?

Answer: The ternary conditional operator is a concise way to write an if-else statement in a single line. It has the syntax value_if_true if condition else value_if_false.

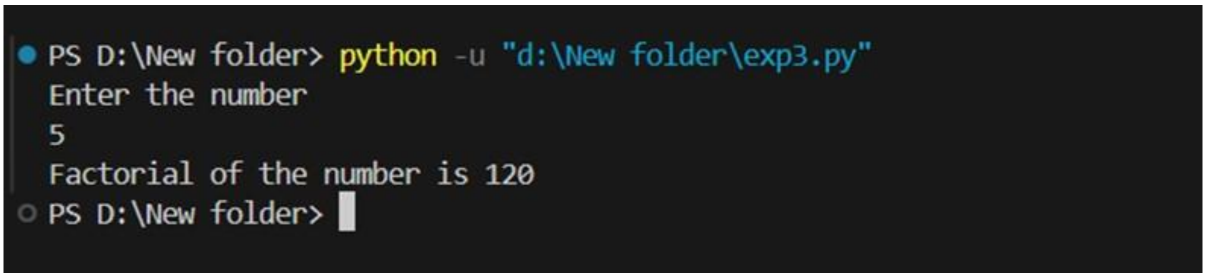
EXPERIMENT NO 3

Aim: Loops: Create a program that calculates the factorial of a number entered by the user using a loop

Source Code:

```
n = int(input("Enter the number\n"))
ans=1
while n:
 ans*=n
 n=n-1
print("Factorial of the number is",ans)
```

Output:



```
PS D:\New folder> python -u "d:\New folder\exp3.py"
Enter the number
5
Factorial of the number is 120
PS D:\New folder>
```

Viva Questions:

Question: What are loops used for in Python?

Answer: Loops are used to execute a block of code repeatedly as long as a specified condition is true, or for iterating over a sequence (such as lists, tuples, dictionaries, etc.).

Question: What is the difference between a for loop and a while loop?

Answer: A for loop is used for iterating over a sequence or iterable object, such as lists or strings, while a while loop is used to repeatedly execute a block of code as long as a specified condition is true.

Question: How do you exit a loop prematurely in Python?

Answer: You can exit a loop prematurely using the break statement. When the break statement is encountered inside a loop, the loop is terminated immediately, and the program control moves to the next statement after the loop.

Question: What is an infinite loop in Python?

Answer: An infinite loop is a loop that continues to execute indefinitely because its condition never evaluates to false. This can happen accidentally if the loop's condition is not properly defined or if there is no condition at all.

Question: How do you skip the current iteration of a loop in Python?

Answer: You can skip the current iteration of a loop and move to the next iteration using the continue statement. When the continue statement is encountered inside a loop, the current iteration is terminated, and the loop proceeds to the next iteration.

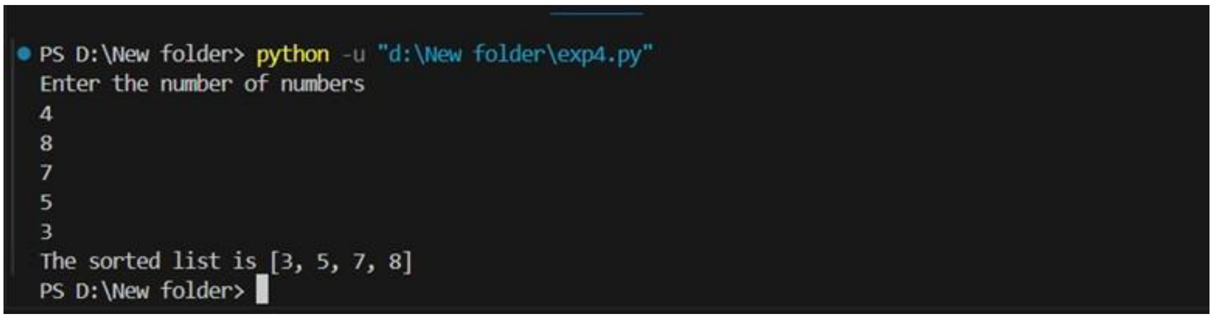
EXPERIMENT NO 4

Aim: Lists and arrays: Create a program that prompts the user for a list of numbers and then sorts them in ascending order.

Source Code:

```
n=int(input("Enter the number of numbers\n"))
array = []
while n :
 ele=int(input())
 array.append(ele)
 n=n-1
array.sort()
print("The sorted list is",array)
```

Output:



```
● PS D:\New folder> python -u "d:\New folder\exp4.py"
  Enter the number of numbers
  4
  8
  7
  5
  3
  The sorted list is [3, 5, 7, 8]
  PS D:\New folder> █
```

Viva Questions:

Question: What is a list in Python?

Answer: A list in Python is a collection of items or elements, ordered and mutable, which means you can change the content of a list after it has been created. Lists are represented by square brackets [ ].

Question: How do you access elements in a list?

Answer: You can access elements in a list using indexing. Indexing starts from 0 for the first element, 1 for the second element, and so on. Negative indexing is also supported, where -1 refers to the last element, -2 refers to the second last, and so forth.

Question: What is the difference between lists and arrays in Python?

Answer: In Python, lists and arrays are similar in that they both store collections of data. However, arrays are a part of the numpy module and are used for numerical computing, offering more functionality and efficiency compared to lists for mathematical operations.

Question: How do you append elements to a list in Python?

Answer: You can append elements to a list using the append() method. This method adds an item to the end of the list. Alternatively, you can use the insert() method to insert an element at a specified position.

Question: What are some common operations performed on lists in Python?

Answer: Some common operations on lists include adding elements, removing elements, accessing elements using indexing, slicing, iterating over elements using loops, checking if an element exists in a list, and sorting the elements. Lists offer a wide range of functionalities for manipulating data.

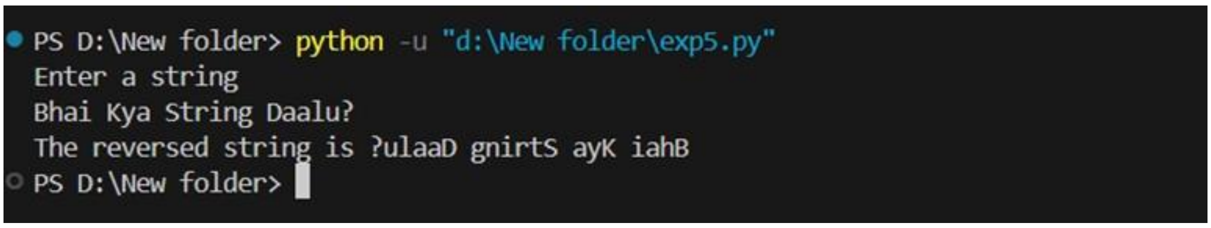
EXPERIMENT NO 5

Aim: Strings and string manipulation: Create a program that prompts the user for a string and then prints out the string reversed

Source Code:

```
str = input("Enter a string\n")

rev = str[::-1]

print("The reversed string is",rev)
```

Output:



```
PS D:\New folder> python -u "d:\New folder\exp5.py"
Enter a string
Bhai Kya String Daalu?
The reversed string is ?ulaaD gnirtS ayK iahB
PS D:\New folder>
```

Viva Questions:

Question: What is string manipulation in Python?

Answer: String manipulation refers to the process of modifying or manipulating strings in various ways, such as concatenating strings, splitting strings, replacing characters or substrings, converting cases, and more.

Question: How do you concatenate strings in Python?

Answer: You can concatenate strings in Python using the + operator. For example, "Hello" + " " + "World" will result in the string "Hello World".

Question: How do you split a string into substrings in Python?

Answer: You can split a string into substrings using the split() method. This method splits a string based on a specified separator and returns a list of substrings.

Question: How do you replace characters or substrings in a string in Python?

Answer: You can replace characters or substrings in a string using the replace() method. This method replaces all occurrences of a specified substring with another substring.

Question: How do you convert the case of a string in Python?

Answer: You can convert the case of a string in Python using methods such as upper() to convert to uppercase, lower() to convert to lowercase, capitalize() to capitalize the first character, and title() to capitalize the first character of each word.

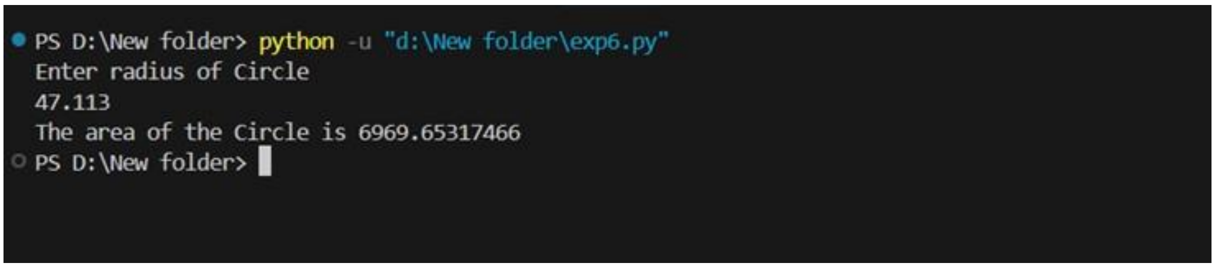
EXPERIMENT NO 6

Aim: Functions: Create a program that defines a function to calculate the area of a circle based on the radius entered by the user

Source Code:

```
def area(r):
 return 3.14*r*r


r = int(input("Enter radius of Circle\n"))
ans = area(r)
print("The area of the Circle is",ans)
```

Output:



```
PS D:\New folder> python -u "d:\New folder\exp6.py"
Enter radius of Circle
47.113
The area of the Circle is 6969.65317466
PS D:\New folder>
```

Viva Questions:

Question: What is a function in Python?

Answer: A function in Python is a block of reusable code that performs a specific task. It takes inputs, called parameters or arguments, and returns outputs. Functions help organize code, improve reusability, and facilitate modular programming.

Question: How do you define a function in Python?

Answer: You define a function in Python using the def keyword followed by the function name and parameters enclosed in parentheses. The function body is indented below the def statement.

Question: What is the difference between parameters and arguments in Python functions?

Answer: Parameters are variables listed in the function definition, while arguments are the values passed to the function when it is called. Parameters receive the values of arguments, allowing functions to work with different inputs.

Question: How do you call a function in Python?

Answer: You call a function in Python by using its name followed by parentheses () containing the arguments, if any, to be passed to the function. For example, my_function(arg1, arg2).

Question: Can a Python function return multiple values?

Answer: Yes, a Python function can return multiple values by returning them as a tuple. You can then unpack the tuple when calling the function to access individual values. For example, return value1, value2.

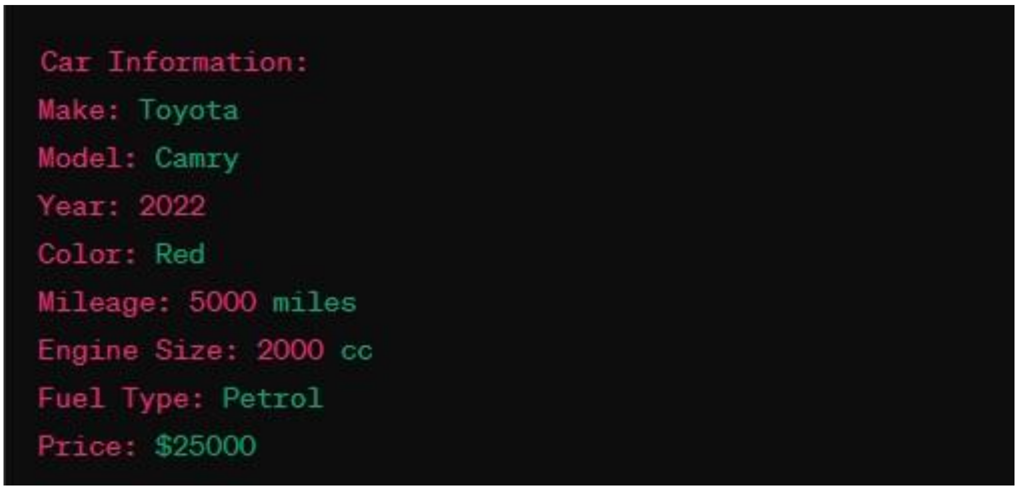
EXPERIMENT NO 7

Aim: Classes and objects: Create a program that defines a class to represent a car and then creates an object of that class with specific attributes

Source Code:

```
class Car:
def __init__(self, make, model, year, color, mileage, engine_size, fuel_type, price):
    self.make = make
    self.model = model
    self.year = year
    self.color = color
    self.mileage = mileage
    self.engine_size = engine_size
    self.fuel_type = fuel_type
    self.price = price
def display_info(self):
    print("Car Information:")
    print(f"Make: {self.make}")
    print(f"Model: {self.model}")
    print(f"Year: {self.year}")
    print(f"Color: {self.color}")
    print(f"Mileage: {self.mileage} miles")
    print(f"Engine Size: {self.engine_size} cc")
    print(f"Fuel Type: {self.fuel_type}")
    print(f"Price: ${self.price}")
# Creating an object of the Car class with specific attributes
my_car = Car("Toyota", "Camry", 2022, "Red", 5000, 2000, "Petrol", 25000)
# Displaying information about the car
my_car.display_info()
```

Output:



```
Car Information:
Make: Toyota
Model: Camry
Year: 2022
Color: Red
Mileage: 5000 miles
Engine Size: 2000 cc
Fuel Type: Petrol
Price: $25000
```

Viva Questions:

Question: What is the difference between a class and an object in Python?

Answer: A class is a blueprint or template for creating objects, while an object is an instance of a class. Classes define the properties and behaviors of objects, whereas objects are specific instances of those classes with unique data.

Question: How do you achieve encapsulation in Python classes?

Answer: Encapsulation is achieved in Python classes by using private attributes and methods, denoted by a single underscore _ or double underscore __prefix. These attributes and methods are only accessible within the class itself, providing data protection and hiding implementation details.

Question: What is inheritance and how does it work in Python classes?

Answer: Inheritance is a mechanism in which a new class (subclass) inherits properties and behaviors from an existing class (superclass). The subclass can extend or override the functionality of the superclass while inheriting its attributes and methods, promoting code reuse and modularity.

Question: Can you explain the concept of polymorphism in Python?

Answer: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables different classes to implement the same interface or method in different ways. Polymorphism in Python is achieved through method overriding and method overloading.

Question: How do you implement composition in Python classes?

Answer: Composition is a design principle where objects contain other objects as part of their state. In Python, composition is implemented by creating objects of other classes within a class and using them to provide functionality. This promotes code reuse and modularity without relying on inheritance.

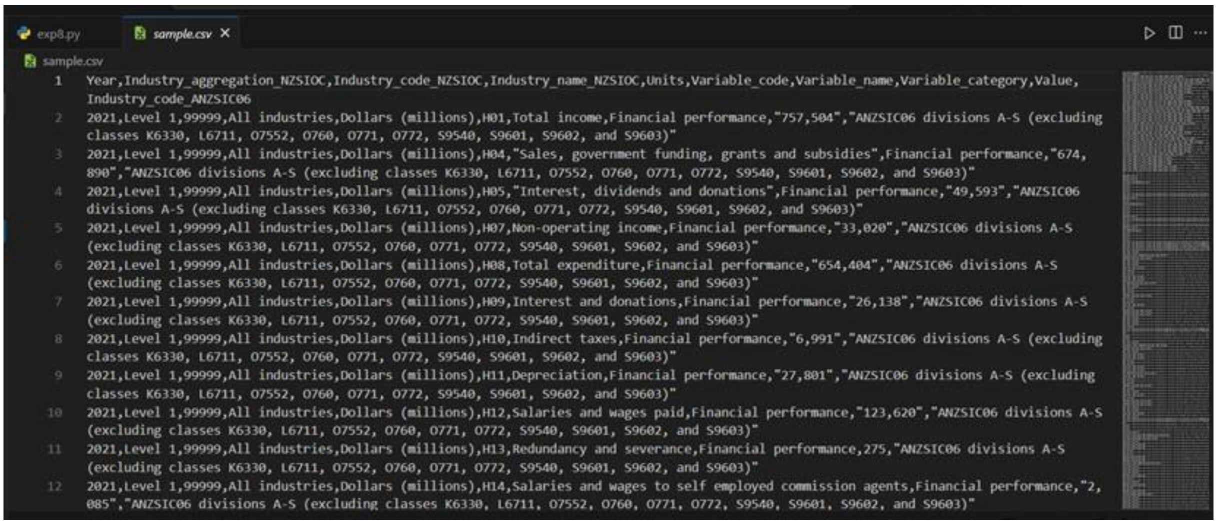
EXPERIMENT NO 8

Aim: File input/output: Create a program that reads data from a file and writes it to another file in a different format

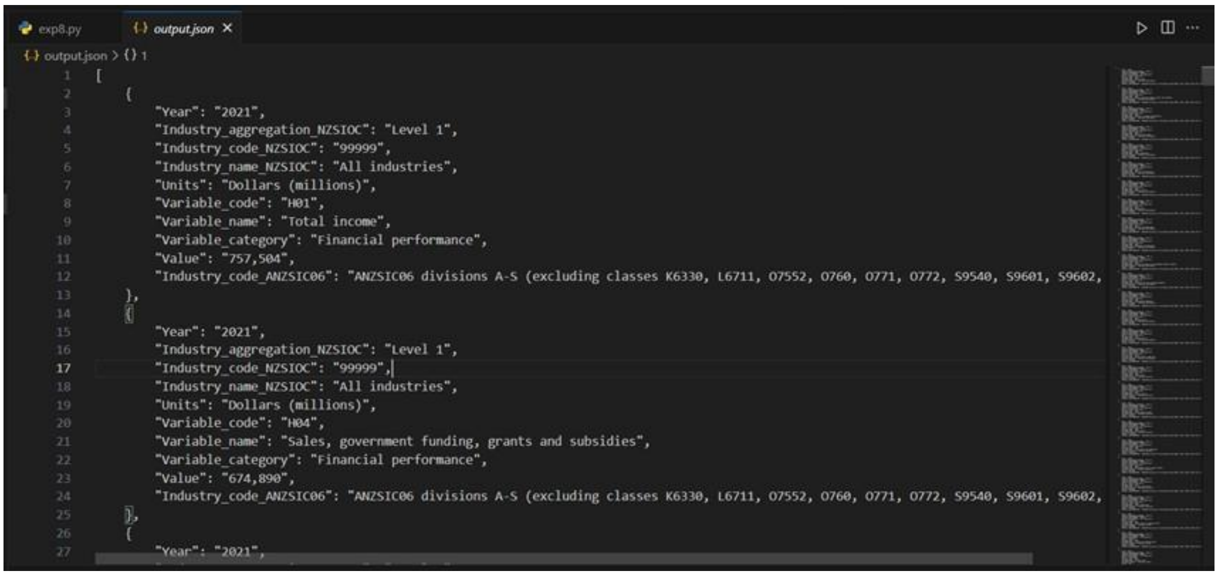
Source Code:

```
import csv

import json

def convert_csv_to_json(input_file, output_file):

        with open(input_file, 'r') as csv_file:

                 csv_reader = csv.DictReader(csv_file)

                data = []

                for row in csv_reader:

                        data.append(row)

        with open(output_file, 'w') as json_file:

                json.dump(data, json_file, indent=4)

input_file_path ='sample.csv'

output_file_path = 'output.json'

convert_csv_to_json(input_file_path, output_file_path)

print("Conversion complete. Data has been written to", output_file_path)
```

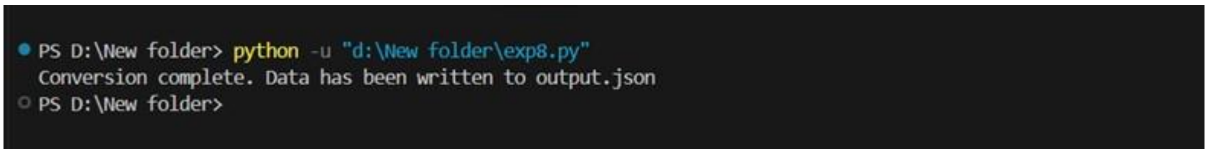
Output:



```
Year,Industry_aggregation_NZSIOC,Industry_code_NZSIOC,Industry_name_NZSIOC,Units,Variable_code,Variable_name,Variable_category,Value,Industry_code_ANZSIC06
2021,Level 1,99999,All industries,Dollars (millions),H01,Total income,Financial performance,"757,504","ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602, and S9603)"
2021,Level 1,99999,All industries,Dollars (millions),H04,"Sales, government funding, grants and subsidies",Financial performance,"674,890","ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602, and S9603)"
2021,Level 1,99999,All industries,Dollars (millions),H05,"Interest, dividends and donations",Financial performance,"49,593","ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602, and S9603)"
2021,Level 1,99999,All industries,Dollars (millions),H07,Non-operating income,Financial performance,"33,020","ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602, and S9603)"
2021,Level 1,99999,All industries,Dollars (millions),H08,Total expenditure,Financial performance,"654,404","ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602, and S9603)"
2021,Level 1,99999,All industries,Dollars (millions),H09,Interest and donations,Financial performance,"26,138","ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602, and S9603)"
2021,Level 1,99999,All industries,Dollars (millions),H10,Indirect taxes,Financial performance,"6,991","ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602, and S9603)"
2021,Level 1,99999,All industries,Dollars (millions),H11,Depreciation,Financial performance,"27,801","ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602, and S9603)"
2021,Level 1,99999,All industries,Dollars (millions),H12,Salaries and wages paid,Financial performance,"123,620","ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602, and S9603)"
2021,Level 1,99999,All industries,Dollars (millions),H13,Redundancy and severance,Financial performance,275,"ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602, and S9603)"
2021,Level 1,99999,All industries,Dollars (millions),H14,Salaries and wages to self employed commission agents,Financial performance,"2,085","ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602, and S9603)"
```



```
[
    {
        "Year": "2021",
        "Industry_aggregation_NZSIOC": "Level 1",
        "Industry_code_NZSIOC": "99999",
        "Industry_name_NZSIOC": "All industries",
        "Units": "Dollars (millions)",
        "Variable_code": "H01",
        "Variable_name": "Total income",
        "Variable_category": "Financial performance",
        "Value": "757,504",
        "Industry_code_ANZSIC06": "ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602,
    },
    {
        "Year": "2021",
        "Industry_aggregation_NZSIOC": "Level 1",
        "Industry_code_NZSIOC": "99999",
        "Industry_name_NZSIOC": "All industries",
        "Units": "Dollars (millions)",
        "Variable_code": "H04",
        "Variable_name": "Sales, government funding, grants and subsidies",
        "Variable_category": "Financial performance",
        "Value": "674,890",
        "Industry_code_ANZSIC06": "ANZSIC06 divisions A-S (excluding classes K6330, L6711, O7552, O760, O771, O772, S9540, S9601, S9602,
    },
    {
        "Year": "2021",
```



```
PS D:\New folder> python -u "d:\New folder\exp8.py"
Conversion complete. Data has been written to output.json
PS D:\New folder>
```

Viva Questions:

Question: What is the difference between reading a file in text mode and binary mode in Python?

Answer: Text mode is used for reading and writing text files, where data is encoded as text using a specified encoding (e.g., UTF-8). Binary mode, on the other hand, is used for reading and writing binary files, such as images or executables, where data is represented as raw bytes.

Question: How do you handle large files efficiently in Python?

Answer: You can handle large files efficiently in Python by using techniques like buffered reading and writing, reading and writing files in chunks, and using memory-mapped files. These techniques help reduce memory usage and improve performance when working with large files.

Question: What are context managers in Python file I/O, and how do you use them?

Answer: Context managers are objects that define __enter__() and __exit__() methods, allowing them to be used with the with statement. They provide resource management and ensure that resources are properly acquired and released, such as file handles in file I/O operations.

Question: How do you handle errors and exceptions in file I/O operations in Python?
Answer: You can handle errors and exceptions in file I/O operations by using try and except blocks to catch and handle specific exceptions, such as FileNotFoundError, PermissionError, or IOError. Additionally, you can use the finally block to perform cleanup operations.

Question: What are some best practices for working with files in Python?

Answer: Some best practices for working with files in Python include using context managers (with statement) to ensure proper resource management, handling exceptions gracefully, closing files explicitly using the close() method or by using the with statement, and using file modes (r, w, a, b, +) appropriately based on the intended operation.

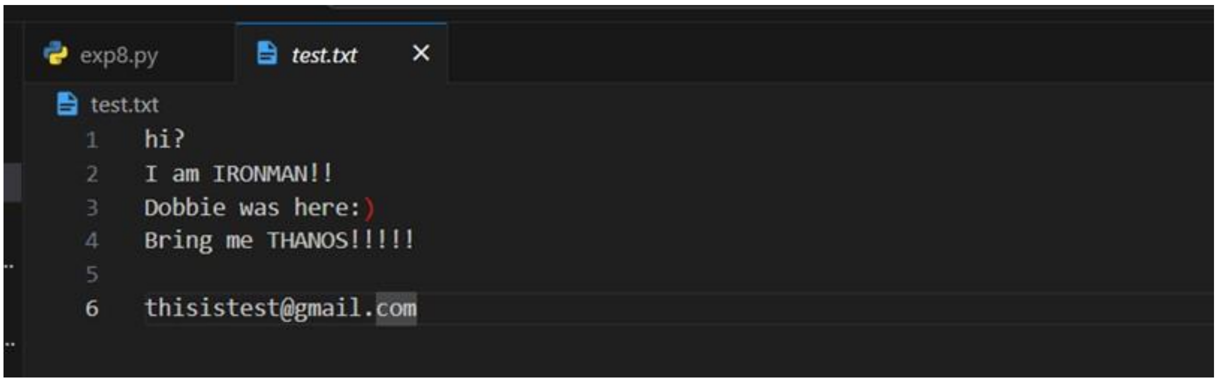
EXPERIMENT NO 9

Aim: Regular expressions: Create a program that uses regular expressions to find all instances of a specific pattern in a text file

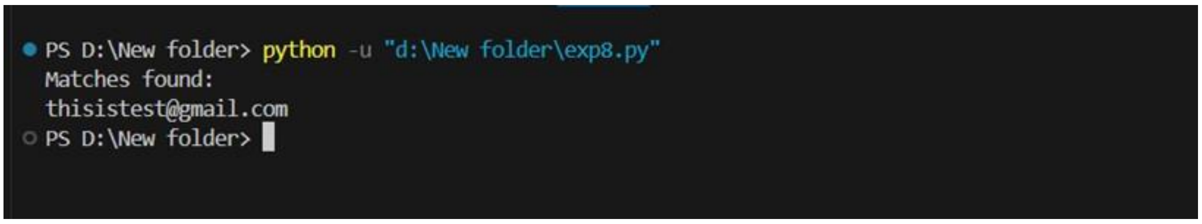
Source Code:

```
import re

def find_pattern_in_file(pattern, file_path):
        with open(file_path, 'r') as file:
                content = file.read()
        matches = re.findall(pattern, content)
        return matches


pattern_to_find = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
file_path ='test.txt'
matches = find_pattern_in_file(pattern_to_find, file_path)
print("Matches found:")
for match in matches:
print(match)
```

Output:



```
hi?
I am IRONMAN!!
Dobbie was here:)
Bring me THANOS!!!!!

thisistest@gmail.com
```



```
PS D:\New folder> python -u "d:\New folder\exp8.py"
Matches found:
thisistest@gmail.com
PS D:\New folder>
```

Viva Questions:

Question: What is a regular expression and how is it used in Python?

Answer: A regular expression, or regex, is a sequence of characters that defines a search pattern. In Python, the re module provides support for working with regular expressions. Regular expressions are used for searching, matching, and manipulating text based on patterns.

Question: What are some common metacharacters used in regular expressions and what do they represent?

Answer: Some common metacharacters used in regular expressions include . (dot), which matches any single character except newline, * (asterisk), which matches zero or more occurrences of the preceding character, + (plus), which matches one or more occurrences of the preceding character, ^ (caret), which matches the start of the string, and $ (dollar), which matches the end of the string.

Question: What is the difference between greedy and non-greedy matching in regular expressions?

Answer: Greedy matching in regular expressions attempts to match as much of the string as possible, while non-greedy or lazy matching attempts to match as little as possible. Greedy matching is denoted by * and + quantifiers, while non-greedy matching is denoted by adding a ? after the quantifier.

Question: How do you capture groups in regular expressions and what are they used for?

Answer: Capture groups in regular expressions are used to extract and isolate specific parts of a matched string. They are denoted by enclosing a part of the pattern in parentheses ( ). Captured groups can be referenced later in the regex or used to extract substrings from the matched text.

Question: What are lookahead and lookbehind assertions in regular expressions?

Answer: Lookahead and lookbehind assertions in regular expressions are used to assert whether a certain pattern is present or absent ahead of or behind the current position in the string, without consuming characters in the match. Lookahead assertions are denoted by (?=pattern) or (?!pattern), while lookbehind assertions are denoted by (?<=pattern) or (?<!pattern).

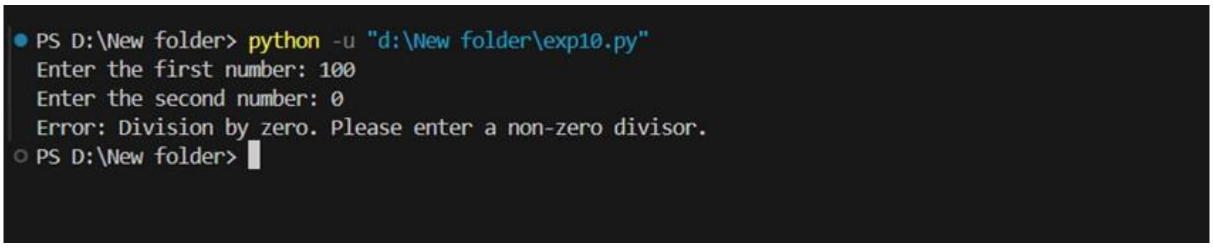
EXPERIMENT NO 10

Aim: Exception handling: Create a program that prompts the user for two numbers and then divides them, handling any exceptions that may arise

Source Code:

```
def divide_numbers():
 while True:
        try:
               num1 = float(input("Enter the first number: "))
               break
        except ValueError:
               print("Invalid input. Please enter a valid number.")
 while True:
        try:
               num2 = float(input("Enter the second number: "))
               break
        except ValueError:
               print("Invalid input. Please enter a valid number.")
 try:
        result = num1 / num2
        print("The result of dividing", num1, "by", num2, "is:", result)
 except ZeroDivisionError:
        print("Error: Division by zero. Please enter a non-zero divisor.")


divide_numbers()
```

Output:



```
PS D:\New folder> python -u "d:\New folder\exp10.py"
Enter the first number: 100
Enter the second number: 0
Error: Division by zero. Please enter a non-zero divisor.
PS D:\New folder>
```

Viva Questions:

Question: What is the purpose of exception handling in Python?

Answer: Exception handling in Python is used to gracefully manage errors and exceptions that occur during program execution. It allows you to detect and handle exceptional conditions, preventing your program from crashing and providing a way to recover from errors.

Question: What are some common built-in exceptions in Python and when do they occur?

Answer: Some common built-in exceptions in Python include SyntaxError, which occurs when there is a syntax error in the code, TypeError, which occurs when an operation or function is applied to an object of inappropriate type, and ZeroDivisionError, which occurs when division or modulo by zero is attempted.

Question: How do you handle multiple exceptions in Python?

Answer: You can handle multiple exceptions in Python by using multiple except blocks, each handling a specific type of exception. Alternatively, you can use a single except block followed by a tuple of exception types to handle multiple exceptions with the same block of code.

Question: What is the purpose of the finally block in Python exception handling?

Answer: The finally block in Python exception handling is used to define cleanup code that should be executed regardless of whether an exception occurs or not. It ensures that certain operations, such as closing files or releasing resources, are always performed, even if an exception is raised.

Question: How do you raise custom exceptions in Python?

Answer: You can raise custom exceptions in Python by using the raise statement followed by an instance of the desired exception class. Custom exceptions are typically defined by subclassing the built-in Exception class or one of its subclasses, allowing you to create specialized exceptions for your application.

PROGRAM 11

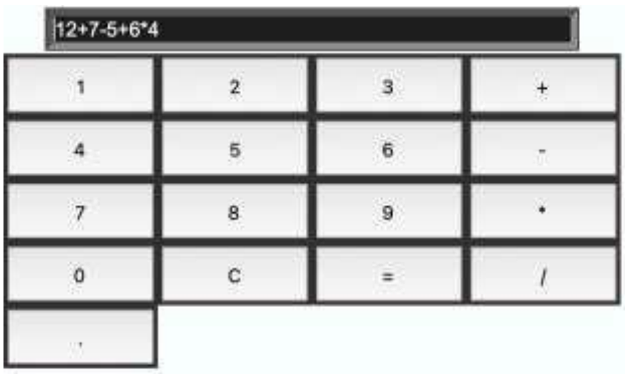
Aim: GUI programming: Create a program that uses a graphical user interface (GUI) to allow the user to perform simple calculations.

Source Code:

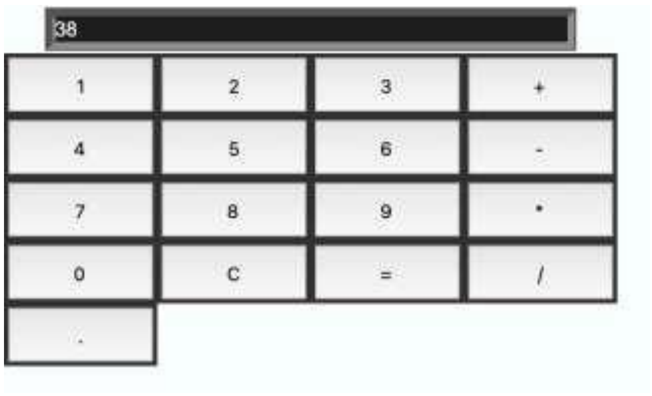
```
from tkinter import *
a = "" def press(n):
global a     a +=
str(n)
equation.set(a) def
equalpress():     try:
global a         t =
str(eval(a))
equation.set(t)
        a = ""     except:
equation.set("Error")
        a = "" def clear():
global a     a = ""
equation.set("") if __name___
== "_main_":       gui =
Tk()
    gui.configure(background="#f8fdfe")
    gui.title("Calculator")
gui.geometry("450x300")
equation = StringVar()     entry_font
= ('Arial', 14)
    a_field = Entry(gui, textvariable=equation, font=entry_font, bd=5)
a_field.grid(columnspan=5, ipadx=100)     equation.set
    button1 = Button(gui, text="1", fg="black", bg="white", command=lambda: press(1),height=2,
width=8)
    button1.grid(row=2, column=0)
    button2 = Button(gui, text="2", fg="black", bg="white", command=lambda: press(2),height=2,
width=8)
    button2.grid(row=2, column=1)
    button3 = Button(gui, text="3", fg="black", bg="white", command=lambda: press(3),height=2,
width=8)
    button3.grid(row=2, column=2)
    button4 = Button(gui, text="4", fg="black", bg="white", command=lambda: press(4),height=2,
width=8)
    button4.grid(row=3, column=0)
    button5 = Button(gui, text="5", fg="black", bg="white", command=lambda: press(5),height=2,
width=8)
    button5.grid(row=3, column=1)
    button6 = Button(gui, text="6", fg="black", bg="white", command=lambda: press(6),height=2,
width=8)
    button6.grid(row=3, column=2)
    button7 = Button(gui, text="7", fg="black", bg="white", command=lambda: press(7),height=2,
width=8)
```

```
    button7.grid(row=4, column=0)
    button8 = Button(gui, text="8", fg="black", bg="white", command=lambda: press(8),height=2,
width=8)
    button8.grid(row=4, column=1)
    button9 = Button(gui, text="9", fg="black", bg="white", command=lambda: press(9),height=2,
width=8)
    button9.grid(row=4, column=2)
    button0 = Button(gui, text="0", fg="black", bg="white", command=lambda: press(0),height=2,
width=8)
    button0.grid(row=5, column=0)
    plus = Button(gui, text="+", fg="black", bg="white", command=lambda: press("+"),height=2,
width=8)
    plus.grid(row=2, column=3)
    minus = Button(gui, text="-", fg="black", bg="white", command=lambda: press("-"),height=2,
width=8)
    minus.grid(row=3, column=3)
    multiply = Button(gui, text="*", fg="black", bg="white", command=lambda: press("*"),height=2,
width=8)
    multiply.grid(row=4, column=3)
    divide = Button(gui, text="/", fg="black", bg="white", command=lambda: press("/"),height=2,
width=8)
    divide.grid(row=5, column=3)
    equal = Button(gui, text="=", fg="black", bg="white", command=equalpress, height=2,width=8)
equal.grid(row=5, column=2)
    clear = Button(gui, text="C", fg="black", bg="white", command=clear, height=2, width=8)
clear.grid(row=5, column=1)
    decimal = Button(gui, text=".", fg="black", bg="white", command=lambda: press("."),height=2,
width=8)
    decimal.grid(row=6, column=0)
gui.mainloop()
```

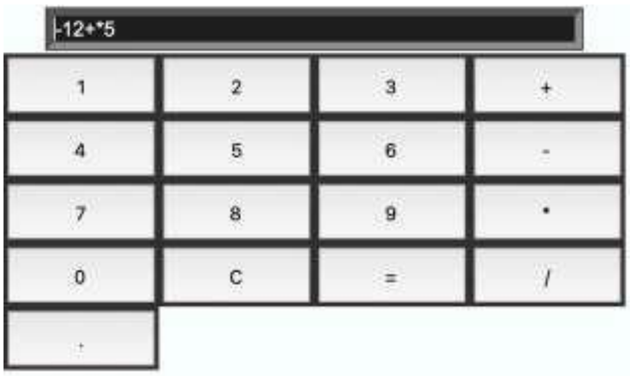
Output:



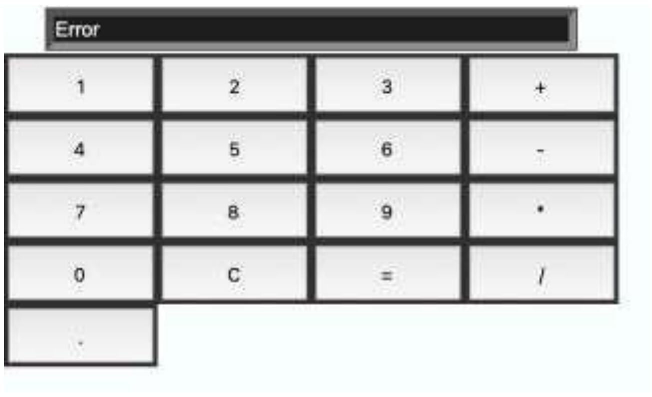
```
12+7-5+6*4
```



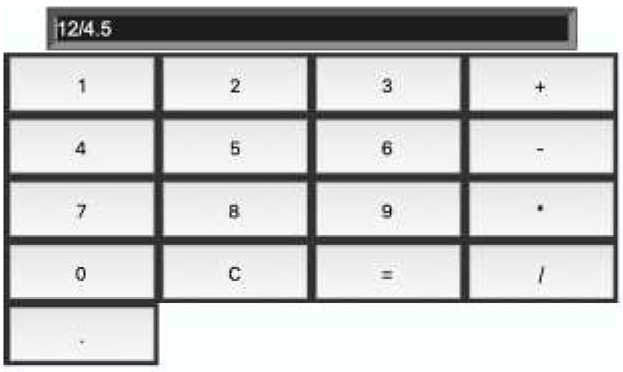
```
38
```



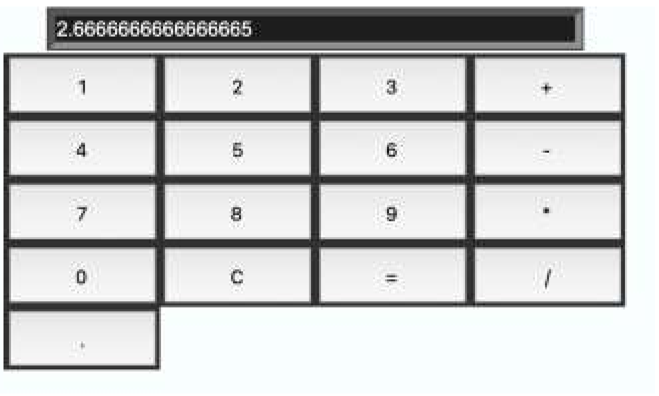
```
-12+*5
```



```
Error
```



```
12/4.5
```



```
2.6666666666666665
```

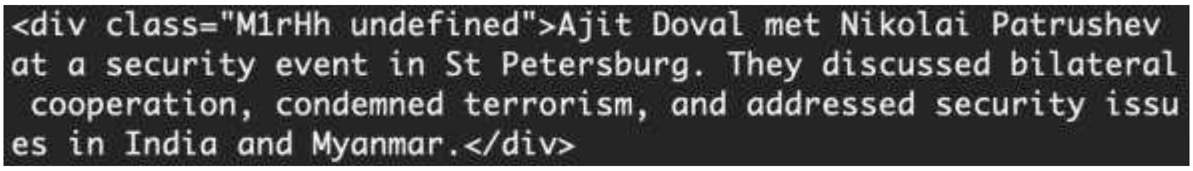
Program-12

Aim: Web scraping: Create a program that uses a web scraping library to extract data from a website and then stores it in a database.

Source Code:

```
import requests
from bs4 import BeautifulSoup
response = requests.get("https://timesofindia.indiatimes.com/india/nsa-doval-meets-
russiancounterpart-reviews-progress-in-bilateral-cooperation/articleshow/109569291.cms") soup =
BeautifulSoup(response.content, "html.parser") s = soup.find('div', class_='M1rHh undefined')
print(s)
```

Output



```
<div class="M1rHh undefined">Ajit Doval met Nikolai Patrushev
at a security event in St Petersburg. They discussed bilateral
 cooperation, condemned terrorism, and addressed security issu
es in India and Myanmar.</div>
```

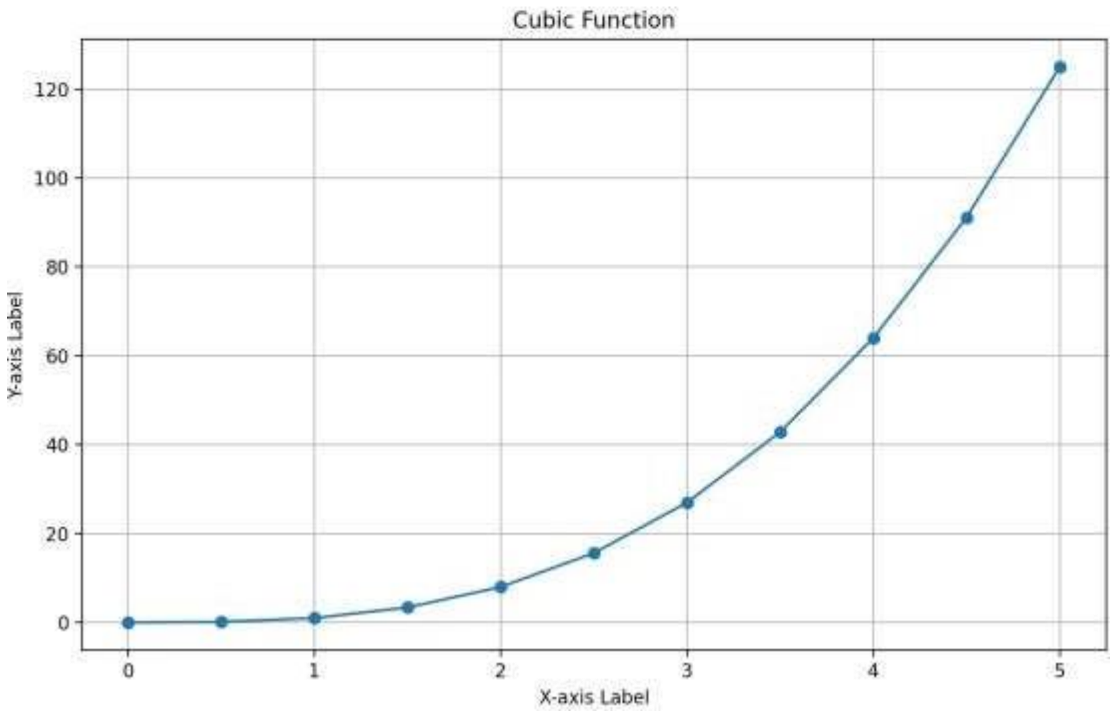
PROGRAM 13

Aim: Data visualization: Create a program that reads data from a file and then creates a visualization of that data using a data visualization library.

Source Code:

```
import matplotlib.pyplot as plt

def read_data_from_file(file_path):
   x_data = []     y_data = []     with
open(file_path, 'r') as file:         for line in
file:             x, y = map(float,
line.strip().split(','))
         x_data.append(x)
y_data.append(y)     return
x_data, y_data

def create_visualization(x_data, y_data):
   plt.figure(figsize=(10, 6))
   plt.plot(x_data, y_data, marker='o', linestyle='-')
plt.title('Cubic Function')      plt.xlabel('X-axis
Label')       plt.ylabel('Y-axis Label')
plt.grid(True)       plt.show()

if _name___== "_main_":
   file_path = 'data.txt'
   x_data, y_data = read_data_from_file(file_path)
create_visualization(x_data, y_data)
```

Output:



Cubic Function
Y-axis Label
X-axis Label
0
20
40
60
80
100
120
0
1
2
3
4
5

Program-14

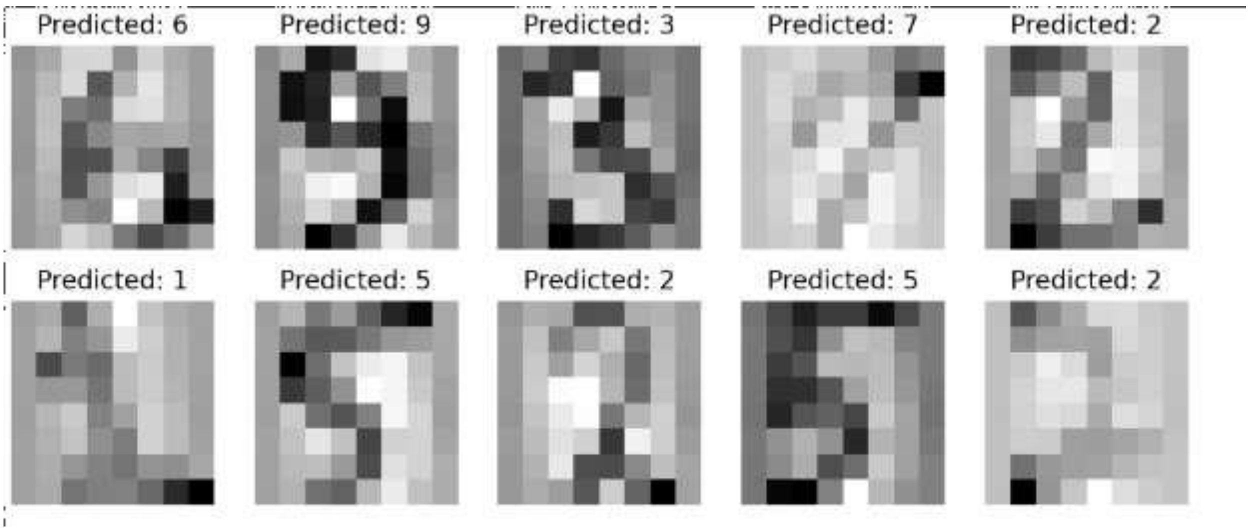
Aim: Machine learning: Create a program that uses a machine learning library to classify images based on their content.

Source Code:

```
import numpy as np import matplotlib.pyplot as plt
from sklearn import datasets from
sklearn.model_selection import train_test_split from
sklearn.preprocessing import StandardScaler from
sklearn.svm import SVC
from sklearn.metrics import accuracy_score digits

= datasets.load_digits()

X = digits.data y
= digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train) X_test
= scaler.transform(X_test)

svm_classifier = SVC(kernel='linear', random_state=42)

svm_classifier.fit(X_train, y_train) y_pred =

svm_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred) print("Accuracy:",
accuracy)

fig, axes = plt.subplots(2, 5, figsize=(10, 4)) for i, ax
in enumerate(axes.flat):
ax.imshow(X_test[i].reshape(8, 8), cmap='binary')
   ax.set_title(f"Predicted: {y_pred[i]}")
ax.axis('off') plt.show()
```

Output:

```
Accuracy: 0.975
```



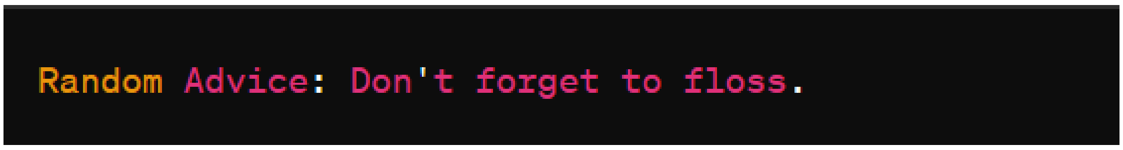
Program-15

Aim: Networking: Create a program that uses a networking library to communicate with a server and retrieve data from it.

Source Code:

```
import requests url =
"https://api.adviceslip.com/advice
" try:
   response = requests.get(url)
   response.raise_for_status() # Raises an HTTPError for bad responses (4xx
and 5xx) data = response.json()                              advice =
data['slip']['advice']       print(f"Random Advice:
{advice}") except requests.exceptions.RequestException as e:     print(f"Error: {e}")
```

Output:



```
Random Advice: Don't forget to floss.
```