**Assignment**

Q1. Write a program to find whether a number is Even or Odd and Prime.

Answer:

def is\_even\_odd(number):

    if number % 2 == 0:

        return "Even"

    else:

        return "Odd"

def is\_prime(number):

    if number < 2:

        return False

    for i in range(2, int(number \*\* 0.5) + 1):

        if number % i == 0:

            return False

    return True

# Get user input

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

# Check if the number is even or odd

even\_odd = is\_even\_odd(num)

print(f"The number {num} is {even\_odd}.")

# Check if the number is prime or not

prime = is\_prime(num)

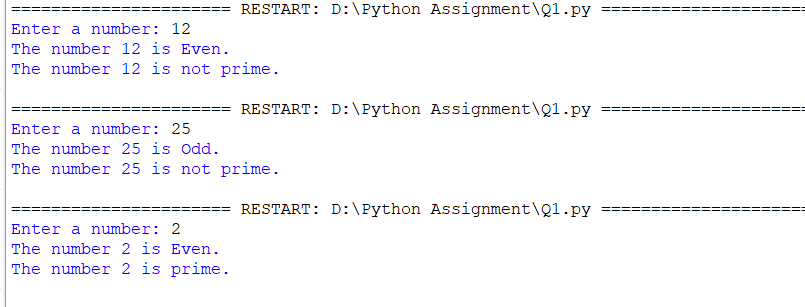
if prime:

    print(f"The number {num} is prime.")

else:

    print(f"The number {num} is not prime.")

OutPut:



Q2. Write a program to find whether a number is Armstrong or not.

Answer:

def is\_armstrong(number):

    # Convert the number to a string to count the number of digits

    num\_str = str(number)

    num\_digits = len(num\_str)

    # Calculate the sum of the cubes of each digit

    armstrong\_sum = 0

    for digit in num\_str:

        armstrong\_sum += int(digit) \*\* num\_digits

    if armstrong\_sum == number:

        return True

    else:

        return False

# Get user input

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

# Check if the number is Armstrong or not

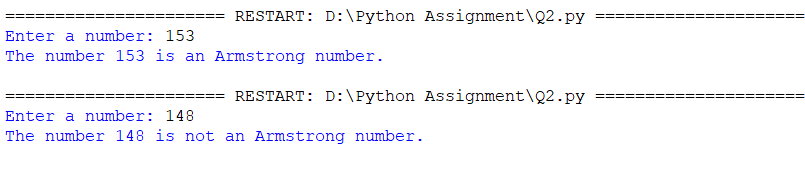
if is\_armstrong(num):

    print(f"The number {num} is an Armstrong number.")

else:

    print(f"The number {num} is not an Armstrong number.")

Output:



Q3. Write a program to find out whether a year is Leap Year or not.

Answer:

def is\_leap\_year(year):

    if year % 4 == 0:

        if year % 100 == 0:

            if year % 400 == 0:

                return True

            else:

                return False

        else:

            return True

    else:

        return False

# Get user input

year = int(input("Enter a year: "))

# Check if the year is a leap year or not

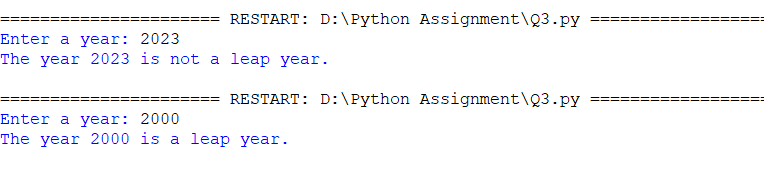
if is\_leap\_year(year):

    print(f"The year {year} is a leap year.")

else:

    print(f"The year {year} is not a leap year.")

Output:



Q4. Write a program to accept a number from the user and find the factorial of the number.

Answer:

def factorial(number):

    if number == 0:

        return 1

    else:

        fact = 1

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

            fact \*= i

        return fact

# Get user input

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

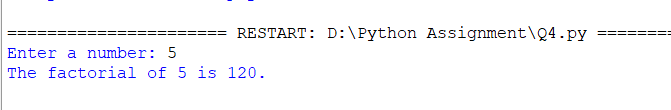
# Calculate the factorial of the number

fact\_result = factorial(num)

# Print the factorial result

print(f"The factorial of {num} is {fact\_result}.")

Output:



Q5. Write a program to find multiples of the given number

def find\_multiples(number, count):

    multiples = []

    for i in range(1, count + 1):

        multiples.append(number \* i)

    return multiples

# Get user input

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

count = int(input("Enter the count of multiples: "))

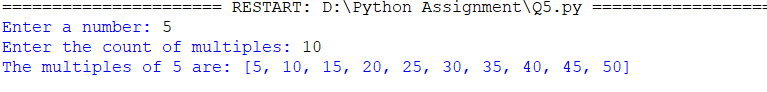
# Find the multiples of the given number

multiples\_list = find\_multiples(num, count)

# Print the multiples

print(f"The multiples of {num} are: {multiples\_list}")

OutPut:



Q6. Write a program to demonstrate the use of default parameters in methods.

Answer:

def greet(name, message="Hello"):

    print(f"{message}, {name}!")

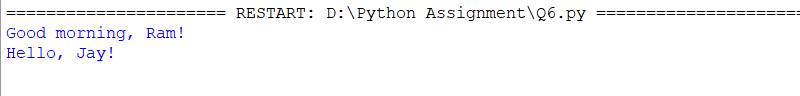
# Calling the function with both parameters

greet("John", "Good morning")

# Calling the function with only the 'name' parameter

greet("Alice")

Output:



Q7. Write a python program to calculate the total number of upper case and lower case characters.

Answer:

def count\_upper\_lower(string):

    upper\_count = 0

    lower\_count = 0

    for char in string:

        if char.isupper():

            upper\_count += 1

        elif char.islower():

            lower\_count += 1

    return upper\_count, lower\_count

# Get user input

input\_string = input("Enter a string: ")

# Calculate the count of uppercase and lowercase characters

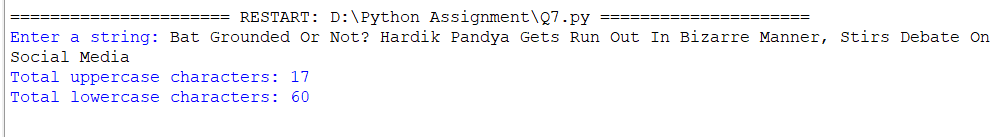
upper, lower = count\_upper\_lower(input\_string)

# Print the results

print(f"Total uppercase characters: {upper}")

print(f"Total lowercase characters: {lower}")

Output:



Q8. Write a python program to calculate the average of a given number using the lambda function.

Answer:

# Get user input for a list of numbers

numbers = input("Enter a list of numbers, separated by spaces: ").split()

numbers = [int(num) for num in numbers]

# Calculate the average using lambda function

average = lambda num\_list: sum(num\_list) / len(num\_list)

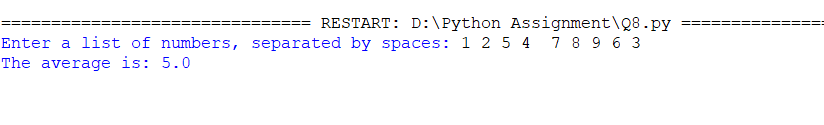
# Call the lambda function to calculate the average

result = average(numbers)

# Print the average

print(f"The average is: {result}")

Output:



Q9. Write a program to define multiple decorator functions to find out square root in the second

decorator and then factorial of square root in the first decorator function of a given number.

Answer:

import math

def square\_root\_decorator(func):

    def wrapper(number):

        result = func(number)

        sqrt\_result = math.sqrt(result)

        print(f"The square root of {result} is: {sqrt\_result}")

        return sqrt\_result

    return wrapper

def factorial\_decorator(func):

    def wrapper(number):

        result = func(number)

        factorial\_result = 1

        for i in range(2, int(result) + 1):

            factorial\_result \*= i

        print(f"The factorial of the square root ({result}) is: {factorial\_result}")

        return factorial\_result

    return wrapper

@factorial\_decorator

@square\_root\_decorator

def calculate(number):

    return number

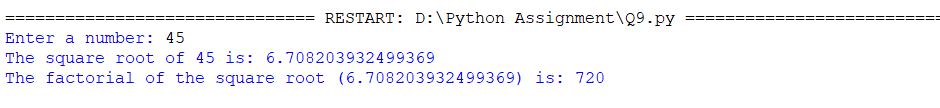
# Get user input

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

# Call the decorated function

calculate(num)

Output:



Q10. Write a program to define a simple class.

Answer:

class Person:

    def \_\_init\_\_(self, name, age):

        self.name = name

        self.age = age

    def introduce(self):

        print(f"My name is {self.name} and I am {self.age} years old.")

# Create an instance of the Person class

name=input("Enter the name")

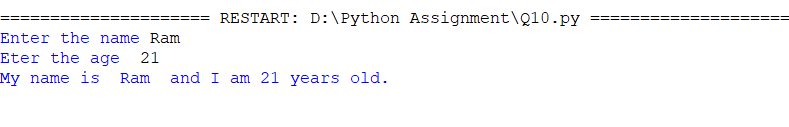
age=int(input("Eter the age")

person1 = Person(name,age)

# Call the introduce method

person1.introduce()

Output:



Q11. Write a program to illustrate the creation of multiple objects for a class.

Answer:

class Person:

    def \_\_init\_\_(self, name, age):

        self.name = name

        self.age = age

    def introduce(self):

        print(f"My name is {self.name} and I am {self.age} years old.")

# Create multiple objects of the Person class

person1 = Person("Jay ", 25)

person2 = Person("Aman", 30)

person3 = Person("Raj", 35)

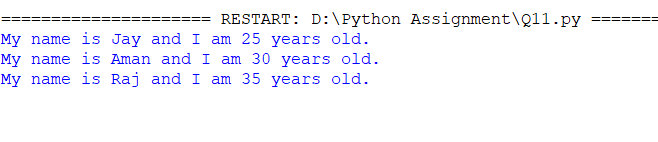
# Call the introduce method for each object

person1.introduce()

person2.introduce()

person3.introduce()

Output:



Q12. Write a program to demonstrate the use of constructor in a class.

Answer:

class Person:

    def \_\_init\_\_(self, name, age):

        """

        Constructor method for the Person class.

        Initializes the name and age attributes.

        """

        self.name = name

        self.age = age

    def introduce(self):

        """

        Method to introduce the person.

        Prints the name and age of the person.

        """

        print(f"My name is {self.name} and I am {self.age} years old.")

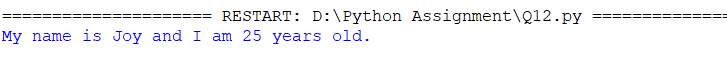
# Create an instance of the Person class

person = Person("Joy", 25)

# Call the introduce method

person.introduce()

Output:



Q13. Write a program to demonstrate the use of destructors in a class.

Answer:

class Person:

    def \_\_init\_\_(self, name):

        self.name = name

    def \_\_del\_\_(self):

        print(f"Deleting the Person object for {self.name}.")

    def introduce(self):

        print(f"My name is {self.name}.")

name = input("Enter the person's name: ")

person = Person("name")

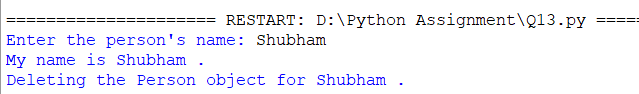
# Call the introduce method

person.introduce()

# Explicitly delete the object

del person

Output:



Q14. Write a program to demonstrate dynamic attribute of a class.

class Car:

    def \_\_init\_\_(self, brand):

        self.brand = brand

# Create an instance of the Car class

brand=input("Enter Brand Name")

car = Car(brand)

# Add dynamic attributes to the car object

car.color = "Red"

car.price = 50000

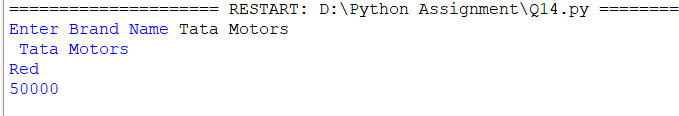
# Access the dynamic attributes

print(car.brand)

print(car.color)

print(car.price)

Output:



Q15. Write a program to call class methods using objects.

Aanswer:

class MathUtils:

    @staticmethod

    def add(x, y):

        return x + y

    @staticmethod

    def multiply(x, y):

        return x \* y

# Create an instance of the MathUtils class

math\_obj = MathUtils()

# Call the class methods using the object

a=int(input("Enter the a number "))

b=int(input("Enter the a number "))

sum\_result = math\_obj.add(a, b)

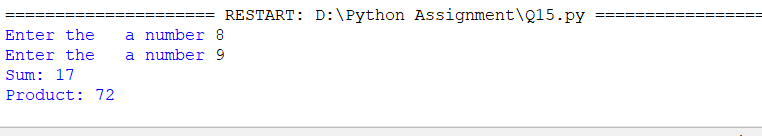
product\_result = math\_obj.multiply(a,b)

# Print the results

print(f"Sum: {sum\_result}")

print(f"Product: {product\_result}")

Output:



Q16. Write a program to demonstrate the use of super function.

Answer:

class Vehicle:

    def \_\_init\_\_(self, name):

        self.name = name

    def start(self):

        print(f"{self.name} is starting.")

class Car(Vehicle):

    def \_\_init\_\_(self, name, color):

        super().\_\_init\_\_(name)

        self.color = color

    def start(self):

        super().start()

        print(f"The {self.color} car is ready to go.")

# Create an instance of the Car class

name=input("Enter the  Vehicle name:")

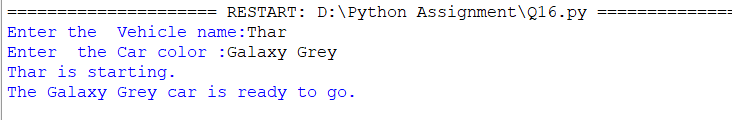
color=input("Enter  the Car color :")

car = Car(name, color)

# Call the start method

car.start()

Output:



Q17. Write a program to calculate the arc length of an angle by assigning values to the

radius and angle data attributes of the class ArcLength.

Answer:

import math

class ArcLength:

    def \_\_init\_\_(self, radius, angle):

        self.radius = radius

        self.angle = angle

    def calculate\_arc\_length(self):

        arc\_length = (2 \* math.pi \* self.radius \* self.angle) / 360

        return arc\_length

# Create an instance of the ArcLength class

radius = float(input("Enter the radius: "))

angle = float(input("Enter the angle in degrees: "))

arc = ArcLength(radius, angle)

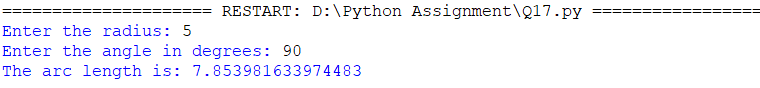
# Calculate the arc length

result = arc.calculate\_arc\_length()

# Print the result

print(f"The arc length is: {result}")

Output:



Q18. Write a python program to calculate gross pay(hourly paid employee) using class and object The input as number of hour work and extra hours,payments mode to the employee are based On the number of hours she has worked. The working hour is 8 hours(per day) and the amount 1500\- is paid. If work is less than 8 hours, payment is75\- per hour. If work is more than 8 hours pay ₹75 per Hour

Answer:

class HourlyPaidEmployee:

    def \_\_init\_\_(self, hourly\_rate, ideal\_hours, ideal\_payment):

        self.hourly\_rate = hourly\_rate

        self.ideal\_hours = ideal\_hours

        self.ideal\_payment = ideal\_payment

    def calculate\_gross\_pay(self, hours\_worked):

        if hours\_worked <= self.ideal\_hours:

            gross\_pay = self.ideal\_payment - (self.ideal\_hours - hours\_worked) \* self.hourly\_rate

        else:

            extra\_hours = hours\_worked - self.ideal\_hours

            overtime\_pay = 75 \* extra\_hours

            gross\_pay = self.ideal\_payment + overtime\_pay

        return gross\_pay

hourly\_rate = 75  # Payment rate per hour

ideal\_hours = 8

ideal\_payment = 1500

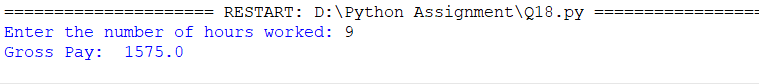
employee = HourlyPaidEmployee(hourly\_rate, ideal\_hours, ideal\_payment)

hours\_worked = float(input("Enter the number of hours worked: "))

gross\_pay = employee.calculate\_gross\_pay(hours\_worked)

print("Gross Pay: ", gross\_pay)

Output:



Q19. Write a program to calc discount based on SP using class and object, take product name,

rate and quantity as input.

the discount will be calculated as

if (order<10000):

discount=20%

if (order<50000):

discount=10%

if (order>50000):

discount=5%

Answer:

class DiscountCalculator:

    def \_\_init\_\_(self, product\_name, rate, quantity):

        self.product\_name = product\_name

        self.rate = rate

        self.quantity = quantity

    def calculate\_discount(self):

        order = self.rate \* self.quantity

        if order < 10000:

            discount = 0.2 \* order

        elif order < 50000:

            discount = 0.1 \* order

        else:

            discount = 0.05 \* order

        return discount

# Get user input for product details

product\_name = input("Enter product name: ")

rate = float(input("Enter product rate: "))

quantity = int(input("Enter product quantity: "))

# Create an instance of the DiscountCalculator class

calculator = DiscountCalculator(product\_name, rate, quantity)

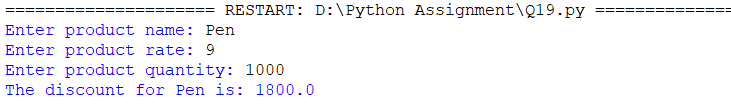
# Calculate the discount

discount = calculator.calculate\_discount()

# Print the result

print(f"The discount for {calculator.product\_name} is: {discount}")

Output:



Q20. Write Python Program to Simulate a Bank Account for deposit, withdrawal and balance

Operations.

Answer:

class BankAccount:

    def \_\_init\_\_(self, account\_number, initial\_balance):

        self.account\_number = account\_number

        self.balance = initial\_balance

    def deposit(self, amount):

        self.balance += amount

        print(f"Amount {amount} deposited successfully. New balance: {self.balance}")

    def withdraw(self, amount):

        if self.balance >= amount:

            self.balance -= amount

            print(f"Amount {amount} withdrawn successfully. New balance: {self.balance}")

        else:

            print("Insufficient balance. Withdrawal failed.")

    def check\_balance(self):

        print(f"Account balance: {self.balance}")

# Create a BankAccount object

account\_number = input("Enter account number: ")

initial\_balance = float(input("Enter initial balance: "))

account = BankAccount(account\_number, initial\_balance)

# Perform bank operations

while True:

    print("\nBank Account Operations:")

    print("1. Deposit")

    print("2. Withdraw")

    print("3. Check Balance")

    print("4. Quit")

    choice = input("Enter your choice (1-4): ")

    if choice == '1':

        amount = float(input("Enter the amount to deposit: "))

        account.deposit(amount)

    elif choice == '2':

        amount = float(input("Enter the amount to withdraw: "))

        account.withdraw(amount)

    elif choice == '3':

        account.check\_balance()

    elif choice == '4':

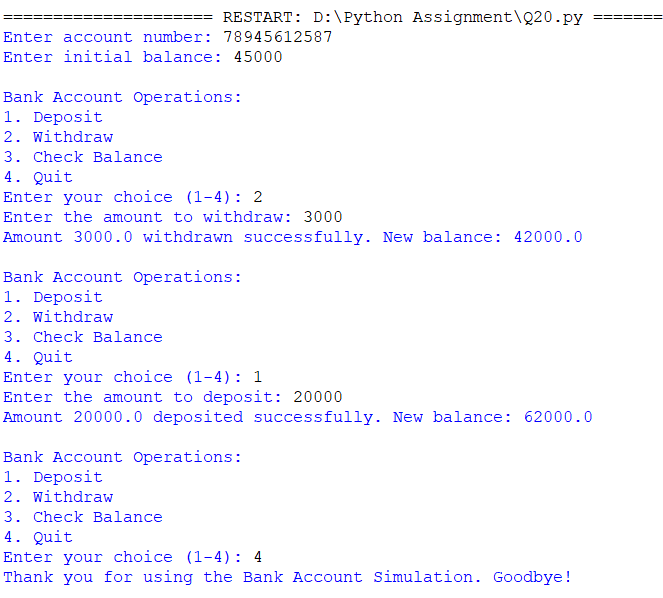
        print("Thank you for using the Bank Account Simulation. Goodbye!")

        break

    else:

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

Output:



Q21. Write a program to demonstrate use of delegates

Answer:

def add(x, y):

    return x + y

def subtract(x, y):

    return x - y

def multiply(x, y):

    return x \* y

def calculate(operation, x, y):

    return operation(x, y)

# Define a lambda function as a delegate

delegate = lambda x, y: x / y

# Use the delegates to perform calculations

a=int(input("Enter The a Number :"))

b=int(input("Enter The b  Number  :"))

result1 = calculate(add, a, b)

result2 = calculate(subtract, a, b)

result3 = calculate(multiply, a, b)

result4 = calculate(delegate, a, b)

# Print the results

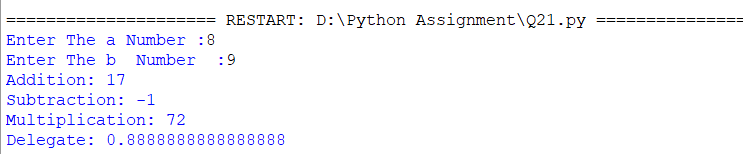
print(f"Addition: {result1}")

print(f"Subtraction: {result2}")

print(f"Multiplication: {result3}")

print(f"Delegate: {result4}")

Output:



Q22. Write a program using regular expressions to find all the digits in a string.

Answer:

import re

def find\_digits(string):

    pattern = r'\d+'

    digits = re.findall(pattern, string)

    return digits

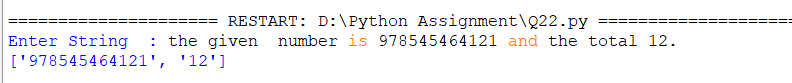
# Example usage

input\_string = input("Enter String  :")

result = find\_digits(input\_string)

print(result)

Output:



Q23. Write a program using regular expression to remove all whitespaces from a string.

Answer:

import re

def remove\_whitespaces(string):

    pattern = r'\s+'  # Regular expression pattern to match one or more whitespaces

    result = re.sub(pattern, '', string)

    return result

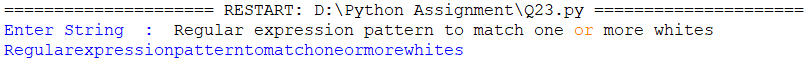
# Example usage

input\_string =input("Enter String  :")

result = remove\_whitespaces(input\_string)

print(result)

Output:



Q24. Write a program to validate phone number using regular expressions.

Answer:

import re

def validate\_phone\_number(phone\_number):

pattern = r'^(\+\d{1,3}\s?)?(\(\d{1,4}\)\s?)?[\d\- ]{7,}$'

match = re.match(pattern, phone\_number)

if match:

return True

else:

return False

# Example usage

phone\_number = input("Enter phone Number: ")

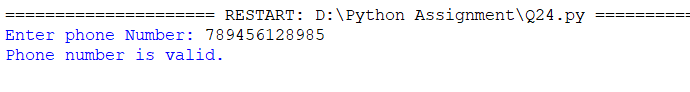
if validate\_phone\_number(phone\_number):

print("Phone number is valid.")

else:

print("Phone number is not valid."):

Output:



Q25. Write a program to Validate email using regular expressions.

Answer:

import re

def validate\_email(email):

    pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'

    match = re.match(pattern, email)

    if match:

        return True

    else:

        return False

# Example usage

email = input("Enter the Email :")

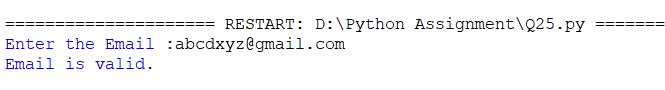
if validate\_email(email):

    print("Email is valid.")

else:

    print("Email is not valid.")

Output:



Q26. Write a program to Validate url using regular expression.

Answer:

import re

def validate\_url(url):

    pattern = r'^(http|https)://([\w.-]+)\.([\w.-]+)/?'

    match = re.match(pattern, url)

    if match:

        return True

    else:

        return False

# Example usage

#url = "https://www.example.com" then in true statement

url=input("Enter  The Url :")

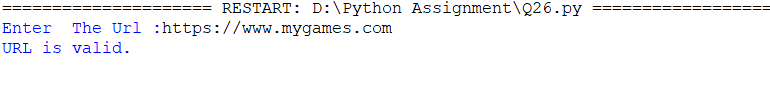
if validate\_url(url):

    print("URL is valid.")

else:

    print("URL is not valid.")

Output:



Q27. Write a program to validate integer input using exception handling.

Answer:

def validate\_integer\_input():

    while True:

        try:

            number = int(input("Enter an integer: "))

            return number

        except ValueError:

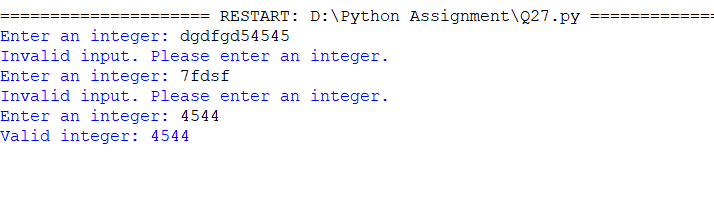
            print("Invalid input. Please enter an integer.")

# Example usage

integer = validate\_integer\_input()

print("Valid integer:", integer)

Output:



Q28. Write a program to demonstrate use of IndexError

Answer:

def demonstrate\_index\_error():

    my\_list = [1, 2, 3, 4, 5]

    try:

        # Trying to access an element at an index that is out of range

        print(my\_list[10])

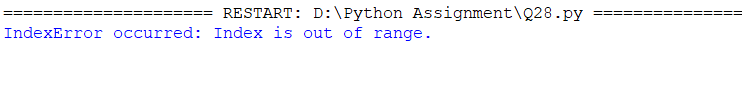
    except IndexError:

        print("IndexError occurred: Index is out of range.")

# Example usage

demonstrate\_index\_error()

Output:



Q29. Write a program to demonstrate use of ZeroDivisionError.

def demonstrate\_zero\_division\_error():

    try:

        dividend = int(input("Enter the dividend: "))

        divisor = int(input("Enter the divisor: "))

        result = dividend / divisor

        print("Result:", result)

    except ZeroDivisionError:

        print("ZeroDivisionError occurred: Cannot divide by zero.")

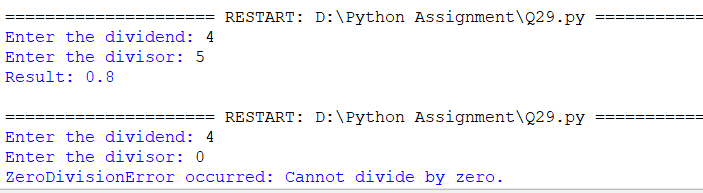
    except ValueError:

        print("ValueError occurred: Invalid input.")

# Example usage

demonstrate\_zero\_division\_error()

Output:



Q30. Write a program to calculate the square and cube of a number using Multithreading.

import threading

def calculate\_square(number):

square = number \*\* 2

print("Square of", number, "is", square)

def calculate\_cube(number):

cube = number \*\* 3

print("Cube of", number, "is", cube)

def calculate\_square\_and\_cube(number):

square\_thread = threading.Thread(target=calculate\_square, args=(number,))

cube\_thread = threading.Thread(target=calculate\_cube, args=(number,))

square\_thread.start()

square\_thread.join()

cube\_thread.start()

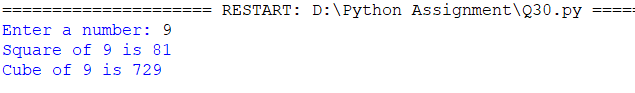
cube\_thread.join()

# Example usage

number = int(input("Enter a number: "))

calculate\_square\_and\_cube(number)

Output:



Q31 .Write a program to print a table of two numbers using thread synchronization.

Answer:

import threading

def print\_table(number, limit, lock):

    for i in range(1, limit + 1):

        # Acquire the lock to ensure exclusive access to the console

        lock.acquire()

        print(number, "x", i, "=", number \* i)

        # Release the lock to allow other threads to acquire it

        lock.release()

def print\_table\_with\_synchronization(number1, number2, limit):

    lock = threading.Lock()

    thread1 = threading.Thread(target=print\_table, args=(number1, limit, lock))

    thread2 = threading.Thread(target=print\_table, args=(number2, limit, lock))

    thread1.start()

    thread2.start()

    thread1.join()

    thread2.join()

# Example usage

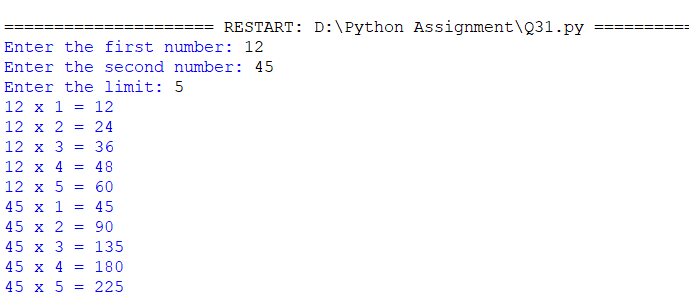
number1 = int(input("Enter the first number: "))

number2 = int(input("Enter the second number: "))

limit = int(input("Enter the limit: "))

print\_table\_with\_synchronization(number1, number2, limit)

Output:



Q32. Write a python program to create a new file in another directory.

Answer:

import os

arr=os.path.join("F:","Student.txt")

f=open(arr,'w+')

f.write("New File")

print("File is created")

f.close()

Output:



Q33. Write a program to copy contents from one file to another file in python. Open the source file in read mode.

Answer:

f1 = open("newFile.txt", "r")

# Open the destination file in write mode

f2 = open("newFile2.txt", "w")

# Read from the source file and write to the destination file

f2.write(f1.read())

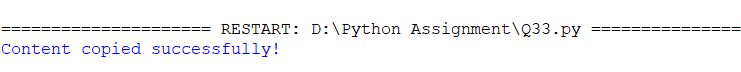
print("Content copied successfully!")

# Close both files

f1.close()

f2.close()

Output:



Q34. Write a python program to count no. of lines in files.

Answer:

def count\_lines\_in\_file(filename):

    line\_count = 0

    with open(filename, 'r') as file:

        for line in file:

            line\_count += 1

    return line\_count

# Example usage

filename = "newFile.txt"  # Replace with the desired filename or provide the file path

lines = count\_lines\_in\_file(filename)

print("Number of lines in the file:", lines)

Output:



Q35. Write a python code to create a database "dbinfo”use this sdb object to create collection "customer"

write code for:

1) insert 10 records (use dictionary with fields{name,cus\_id,doj,address,email,mobile

number,experience})

2) select \* from "customer"

3) search records by specific name

4) delete records by specific customer id.

Answer:

from pymongo import MongoClient

# Establish a connection to the MongoDB server

client = MongoClient('mongodb://localhost:27017/')

# Create or access the "dbinfo" database

db = client['dbinfo']

# Create or access the "customer" collection

customer\_collection = db['customer']

# 1) Insert 10 records

records = [

    {

        'name': 'John Doe',

        'cus\_id': 2,

        'doj': '2022-03-01',

        'address': '123 Main Street',

        'email': 'john.doe@example.com',

        'mobile\_number': '1234567890',

        'experience': 5

    },

     {

        'name': 'Raje ',

        'cus\_id': 3,

        'doj': '2022-01-01',

        'address': '123 Main Street',

        'email': 'john.doe@example.com',

        'mobile\_number': '1234567890',

        'experience': 2

    },

     {

        'name': 'Ram',

        'cus\_id': 4,

        'doj': '2022-01-01',

        'address': '123 Main Street',

        'email': 'john.doe@example.com',

        'mobile\_number': '1234567890',

        'experience': 9

    },

     {

        'name': 'Jay',

        'cus\_id': 5,

        'doj': '2022-01-01',

        'address': '123 Main Street',

        'email': 'john.doe@example.com',

        'mobile\_number': '1234567890',

        'experience': 3

    },

     {

        'name': 'John Doe',

        'cus\_id': 6,

        'doj': '2022-01-01',

        'address': '123 Main Street',

        'email': 'john.doe@example.com',

        'mobile\_number': '1234567890',

        'experience': 8

    },

    # Add more records here...

]

# Insert the records into the "customer" collection

customer\_collection.insert\_many(records)

# 2) Select all records from "customer"

all\_records = customer\_collection.find()

print("All records in 'customer' collection:")

for record in all\_records:

    print(record)

# 3) Search records by specific name

name = 'John Doe'

search\_query = {'name': name}

search\_results = customer\_collection.find(search\_query)

print("Records with name '{}':".format(name))

for result in search\_results:

    print(result)

# 4) Delete records by specific customer id

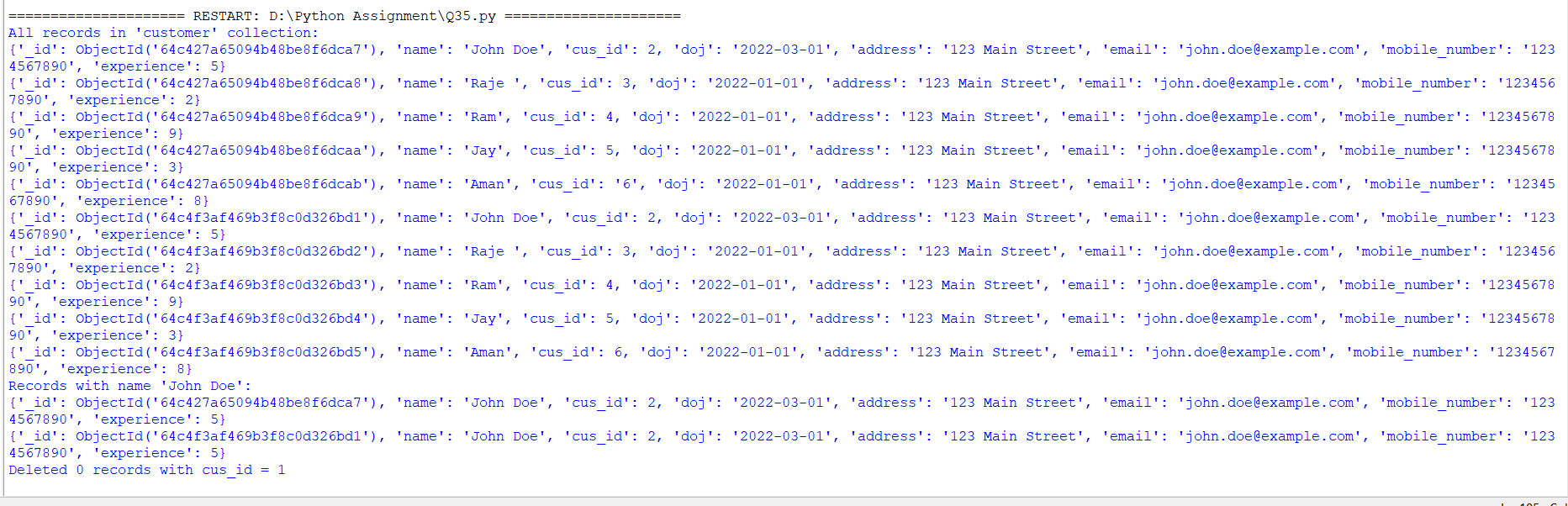
cus\_id = 1

delete\_query = {'cus\_id': cus\_id}

delete\_result = customer\_collection.delete\_many(delete\_query)

print("Deleted {} records with cus\_id = {}".format(delete\_result.deleted\_count, cus\_id))

Output:



Q36. write a python program to create "doctor" collections under "dbinfo" with the table having fields as ({doc\_id,name,hosp\_id,doj,speciality,salary,exp})

write a code to:

1) insert 5 records in collections

2) update exp of doctor

3) search record after updation

4) sort documents in ascending order

Answer:

from pymongo import MongoClient

from pprint import pprint

# Establish a connection to the MongoDB server

client = MongoClient('mongodb://localhost:27017/')

# Create or access the "dbinfo" database

db = client['dbinfo']

# Create or access the "doctor" collection

doctor\_collection = db['doctor']

# 1) Insert 5 records

records = [

    {

        'doc\_id': 1,

        'name': 'Dr. Joy Smith',

        'hosp\_id': 101,

        'doj': '2022-01-01',

        'speciality': 'Cardiology',

        'salary': 10000,

        'exp': 5

    },

     {

        'doc\_id': 2,

        'name': 'Dr. Ram Bhange',

        'hosp\_id': 105,

        'doj': '2022-01-01',

        'speciality': 'Cardiology',

        'salary': 10000,

        'exp': 9

    },

     {

        'doc\_id': 12,

        'name': 'Dr. kunal Bhavsar',

        'hosp\_id': 101,

        'doj': '2022-01-01',

        'speciality': 'Cardiology',

        'salary': 10000,

        'exp': 2

    },

     {

        'doc\_id': 16,

        'name': 'Dr. Shubham Bhor',

        'hosp\_id': 101,

        'doj': '2022-01-01',

        'speciality': 'Cardiology',

        'salary': 10000,

        'exp': 7

    },

     {

        'doc\_id': 9,

        'name': 'Dr. John Smith',

        'hosp\_id': 145,

        'doj': '2022-01-01',

        'speciality': 'Cardiology',

        'salary': 10000,

        'exp': 5

    },

     {

        'doc\_id': 8,

        'name': 'Dr. John Smith',

        'hosp\_id': 180,

        'doj': '2022-01-01',

        'speciality': 'Cardiology',

        'salary': 10000,

        'exp': 7

    },

    # Add more records here...

]

# Insert the records into the "doctor" collection

doctor\_collection.insert\_many(records)

# 2) Update experience of doctor

doc\_id = 1

new\_exp = 7

update\_query = {'doc\_id': doc\_id}

update\_data = {'$set': {'exp': new\_exp}}

doctor\_collection.update\_many(update\_query, update\_data)

# 3) Search record after updation

search\_query = {'doc\_id': doc\_id}

search\_results = doctor\_collection.find(search\_query)

print("Records with doc\_id = {} after updating experience:".format(doc\_id))

for result in search\_results:

    pprint(result)

# 4) Sort documents in ascending order

sort\_field = 'doc\_id'

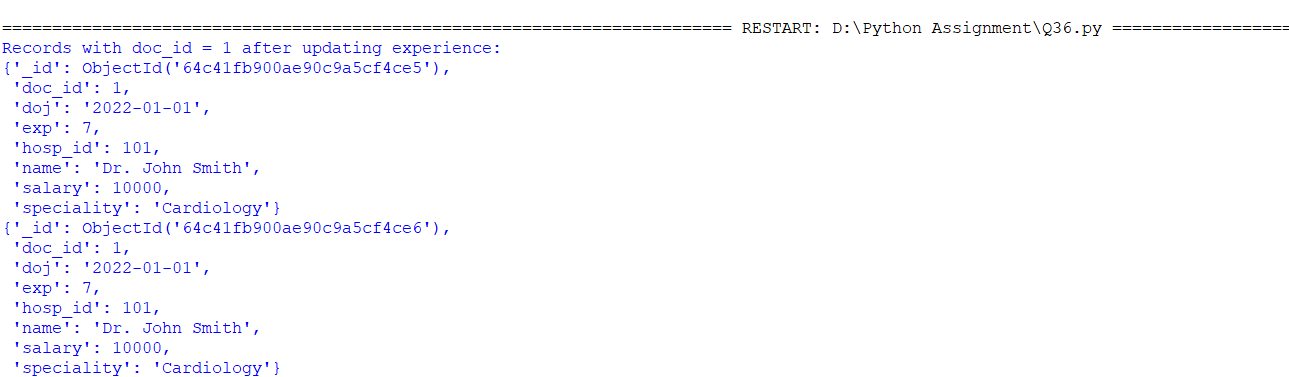
sorted\_results = doctor\_collection.find().sort(sort\_field, 1)

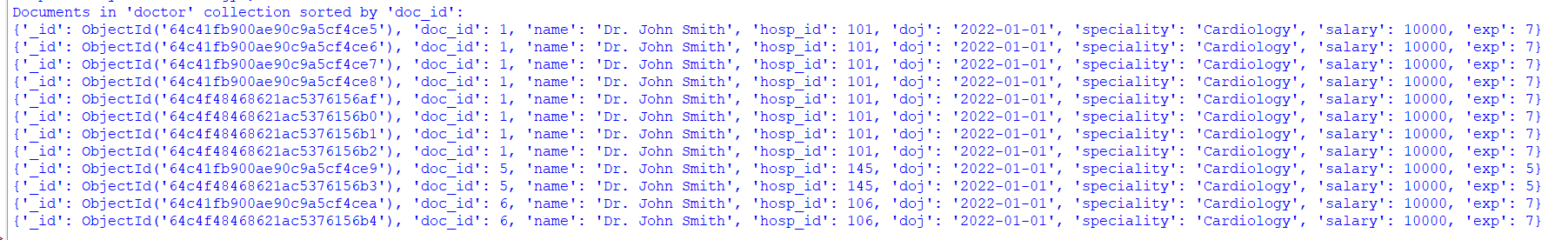
print("Documents in 'doctor' collection sorted by '{}':".format(sort\_field))

for result in sorted\_results:

    pprint(result)

Output:





Q37. Write a program using numpy to display array in the format

type of array,

dimension of array,

shape of array,size of array,

type of element of array

arrange the array

Answer:

import numpy as np

def display\_array\_info(arr):

    # Type of array

    print("Type of array:", type(arr))

    # Dimension of array

    print("Dimension of array:", arr.ndim)

    # Shape of array

    print("Shape of array:", arr.shape)

    # Size of array

    print("Size of array:", arr.size)

    # Type of element in array

    print("Type of element in array:", arr.dtype)

    # Arrange the array

    print("Arranged array:")

    print(np.sort(arr))

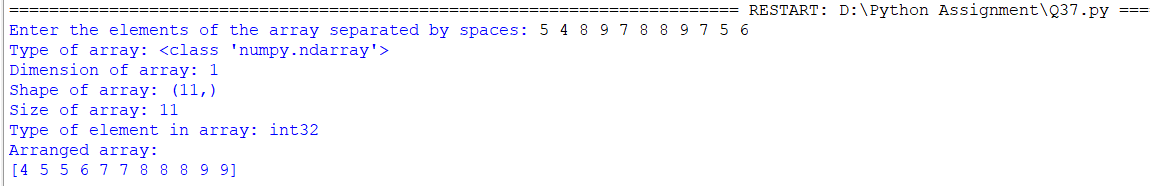
# Example usage

user\_array = input("Enter the elements of the array separated by spaces: ")

my\_array = np.array(user\_array.split(), dtype=int)

display\_array\_info(my\_array)

Output:



Q38.Write a program to create numpy array and use the functions as 0's,1's,lines space , random function

and sum of array

Answer:

import numpy as np

# Create an array of zeros

def create\_zeros\_array(rows, cols):

    return np.zeros((rows, cols))

# Create an array of ones

def create\_ones\_array(rows, cols):

    return np.ones((rows, cols))

# Create an array with linearly spaced values

def create\_linspace\_array(start, end, num):

    return np.linspace(start, end, num=num)

# Create an array with random values

def create\_random\_array(rows, cols):

    return np.random.random((rows, cols))

# Sum of array

def get\_array\_sum(arr):

    return np.sum(arr)

# Example usage

rows = int(input("Enter the number of rows: "))

cols = int(input("Enter the number of columns: "))

zeros\_array = create\_zeros\_array(rows, cols)

print("Zeros Array:")

print(zeros\_array)

ones\_array = create\_ones\_array(rows, cols)

print("\nOnes Array:")

print(ones\_array)

start = float(input("\nEnter the start value for linspace: "))

end = float(input("Enter the end value for linspace: "))

num = int(input("Enter the number of elements for linspace: "))

linspace\_array = create\_linspace\_array(start, end, num)

print("\nLinspace Array:")

print(linspace\_array)

random\_array = create\_random\_array(rows, cols)

print("\nRandom Array:")

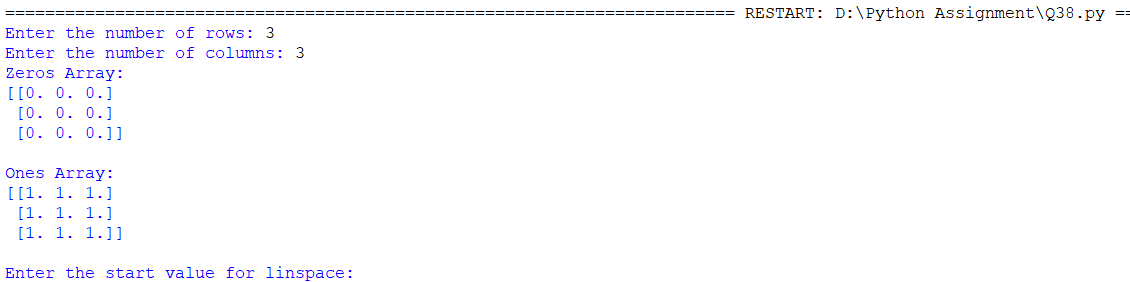
print(random\_array)

# Get sum of array

array\_sum = get\_array\_sum(random\_array)

print("\nSum of Array:", array\_sum)

Output:



Q39. Write a program to create numpy arrays and use the functions as reshape , flatten and transpose?

Answer:

import numpy as np

# Create a 1D array

arr1d = np.array([1, 2, 3, 4, 5,6])

print("1D array:")

print(arr1d)

print()

# Create a 2D array

arr2d = np.array([[1, 2, 3], [4, 5, 6]])

print("2D array:")

print(arr2d)

print()

# Reshape the 1D array into a 2D array

reshaped\_arr = arr1d.reshape(2, 3)

print("Reshaped array:")

print(reshaped\_arr)

print()

# Flatten the 2D array

flattened\_arr = arr2d.flatten()

print("Flattened array:")

print(flattened\_arr)

print()

# Transpose the 2D array

transposed\_arr = arr2d.transpose()

print("Transposed array:")

print(transposed\_arr)

Output:



Q40. Write a program to save and load the array in binary and text files.

Answer:

import numpy as np

# Create a NumPy array

arr = np.array([[1, 2, 3], [4, 5, 6]])

# Save the array to a binary file

np.save("array\_binary.npy", arr)

# Save the array to a text file

np.savetxt("array\_text.txt", arr)

# Load the array from the binary file

loaded\_arr\_binary = np.load("array\_binary.npy")

print("Loaded array from binary file:")

print(loaded\_arr\_binary)

print()

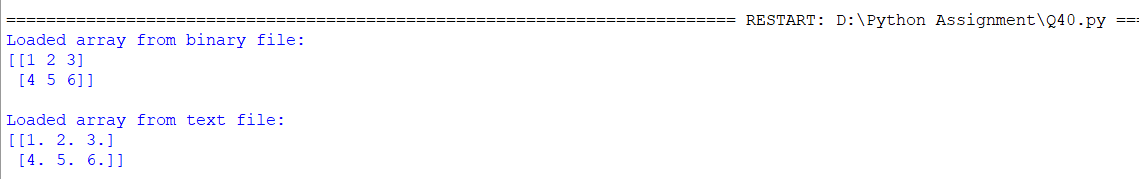
# Load the array from the text file

loaded\_arr\_text = np.loadtxt("array\_text.txt")

print("Loaded array from text file:")

print(loaded\_arr\_text)

Output:



Q41. Write a python program to create the pandas series from an array.

Answer:

import pandas as pd

import numpy as np

# Create an array

arr = np.array([10, 20, 30, 40, 50])

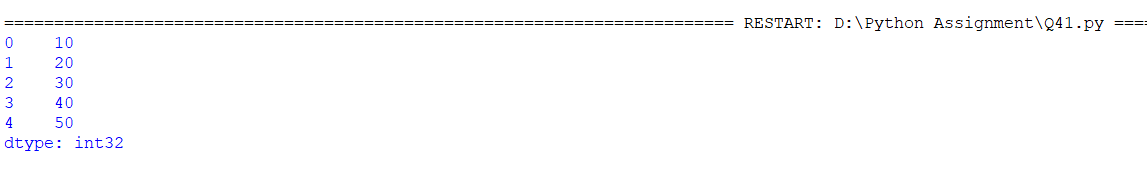
# Create a Pandas Series from the array

series = pd.Series(arr)

# Print the Pandas Series

print(series)

Output:



Q42. Write a python program to create the pandas series using a dictionary and specify the index.

Answer:

import pandas as pd

# Create a dictionary

data = {'A': 10, 'B': 20, 'C': 30, 'D': 40, 'E': 50}

# Specify the index

index = ['B', 'C', 'A', 'D', 'E']

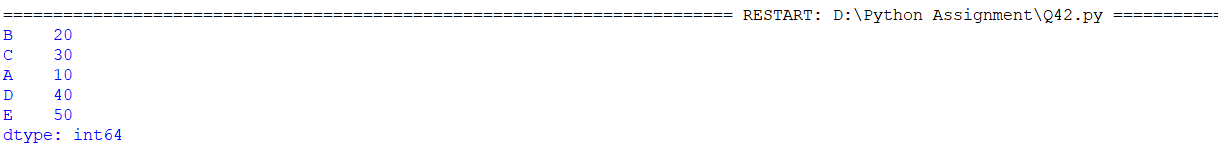
# Create a Pandas Series from the dictionary with specified index

series = pd.Series(data, index=index)

# Print the Pandas Series

print(series)

Output:



Q43.Write a program to access the elements of series in pandas from csv file, also access the

element from

different positions

import pandas as pd

# Step 1: Read the CSV file into a DataFrame

data = pd.read\_csv('data.csv')

# Step 2: Access the elements of the Series

# Access the entire "name" column as a Series

name\_series = data['name']

# Access the element at index 1

element\_at\_index\_1 = name\_series[1]

# Access the element at position 2

element\_at\_position\_2 = name\_series.iloc[2]

# Access the element at the last position

element\_at\_last\_position = name\_series.iloc[-1]

# Print the Series and accessed elements

print("Series:")

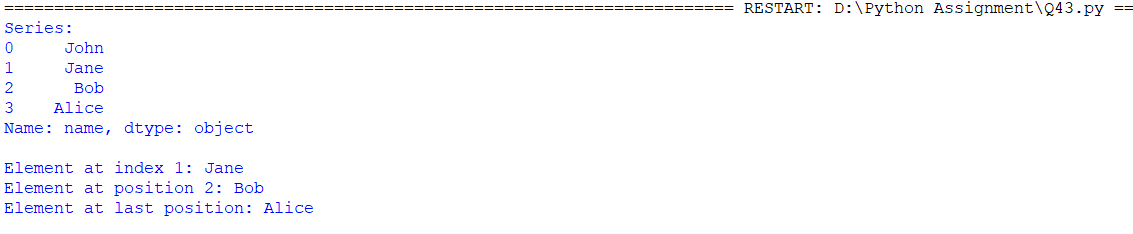
print(name\_series)

print("\nElement at index 1:", element\_at\_index\_1)

print("Element at position 2:", element\_at\_position\_2)

print("Element at last position:", element\_at\_last\_position)

Output:



Q44. Write a program using pandas to the first three rows of a given data frame.

Answer:

import pandas as pd

# Sample data for the DataFrame

data = {

    'Name': ['John', 'Jane', 'Bob', 'Alice', 'Michael'],

    'Age': [30, 25, 40, 35, 28],

    'City': ['New York', 'Los Angeles', 'Chicago', 'San Francisco', 'Seattle']

}

# Create the DataFrame

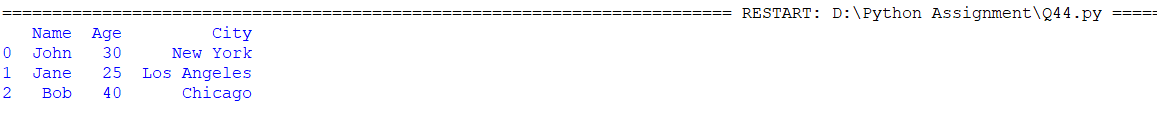
df = pd.DataFrame(data)

# Print the first three rows of the DataFrame using the head() method

first\_three\_rows = df.head(3)

print(first\_three\_rows)

Output:



Q45. Write a program using pandas to create and display a dataframe from a specified dictionary data which

has the index labels sample dataframe.

exam\_data = {"name":[choose any 10 names], "score":[10 scores],"attempts":[10

entries],"qualify":[10 boolean values]}

labels=['a','b',...10 label]

Anwser:

import pandas as pd

# Sample data for the DataFrame

exam\_data = {

    "name": ['John', 'Jane', 'Bob', 'Alice', 'Michael', 'Emma', 'David', 'Olivia', 'Ethan', 'Sophia'],

    "score": [85, 92, 78, 90, 88, 95, 82, 89, 75, 93],

    "attempts": [1, 2, 1, 3, 2, 1, 2, 3, 1, 2],

    "qualify": [True, True, False, True, True, True, False, True, False, True]

}

# Create the DataFrame with specified index labels

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df = pd.DataFrame(exam\_data, index=labels)

# Display the DataFrame

print(df)

Output:

