# Practical Assignment No: 03 Roll No: MC2426055

# Student Name: KARANDE SAMIR BABASAHEB

**Date: 12/10/2024**

**PROGRAM-1:** Write a simple Python class named Student and display its type. Also display the dict attribute keys and values of the module attribute of the Student class.

**CODE :-**

class Student:

def **init**(self, name, age):

self.name = name self.age = age

student1 = Student("Sameer", 23)

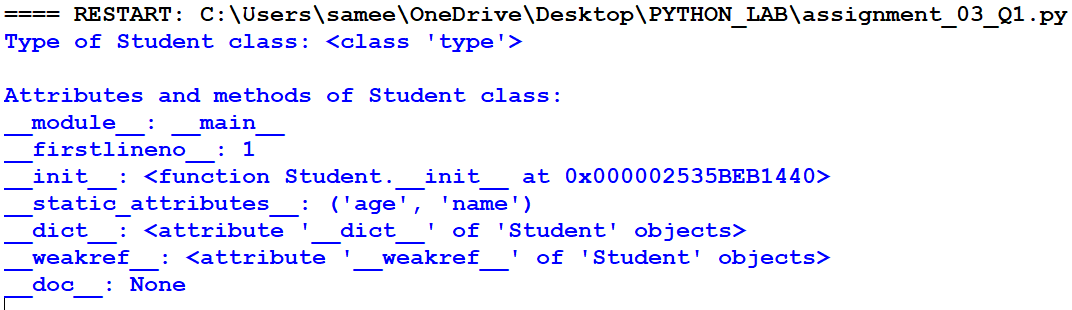
print("Type of Student class:", type(Student))

print("\nAttributes and methods of Student class:")

for key, value in Student.**dict**.items():

print(f"{key}: {value}")

# Output:



**PROGRAM-2:** Write a Python class Student with two attributes, student narne and marks. Modify any attribute value of the class and show the original and modified values of attributes.

**CODE :**-

class Student:

def **init**(self, student\_name, marks):

self.student\_name=student\_nameself.marks=marks student1=Student( "samir", 85)

print("Original student name:", student1.student\_name)

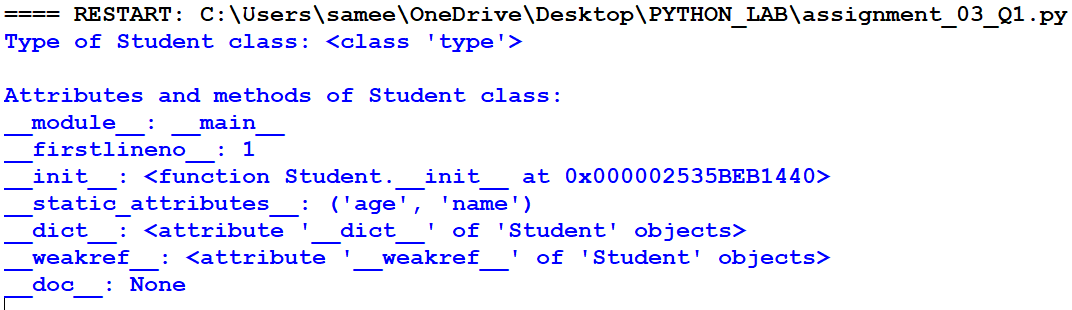
print("Original marks:", student1.marks)

student1.marks = 90

print("\nModified student name:", student1.student\_name)

print("Modified marks:", student1.marks)

# Output:



**PROGRAM-3**: Write a Python class to convert an integer to a Roman numeral.

**CODE** :-

class IntegerToRoman:

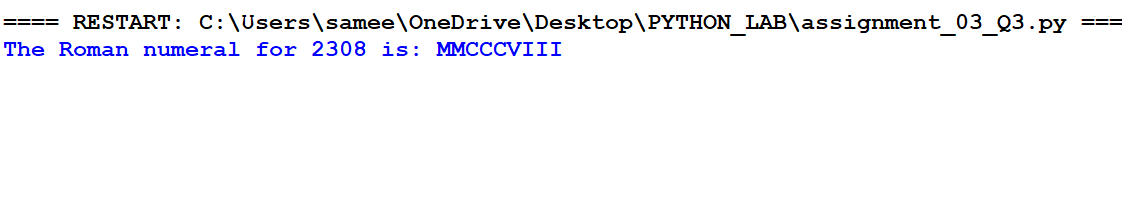
def **init**(self, number):

self.number = number

def to\_roman(self):  
 val = [1000,900,500,400,100,90,50,40,10,9,5,4,1]  
 syb = ["M", "CM", "D", "CD","C", "XC", "L","X L","X","IX", "V", "IV","I"]  
   
 roman\_numeral = ""  
 num = self.number   
 for i in range(len(val)):  
 while num >= val[i]:  
 roman\_numeral += syb[i]  
 num -= val[i]  
   
 return roman\_numeral  
 integer = 2308 converter = IntegerToRoman(integer) roman = converter.to\_roman()

print(f"The Roman numeral for {integer} is: {roman}")

# Output:



# PROGRAM-4: Write a Python class to convert a Roman numeral to an integer.

# CODE :

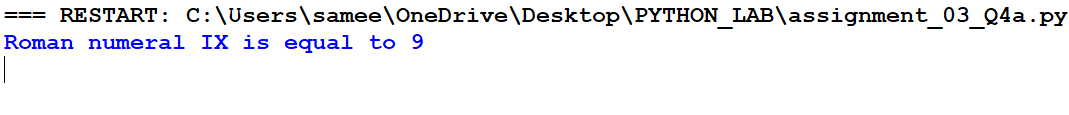
# class RomanToInteger:

def \_\_init\_\_(self, roman):  
 self.roman = roman  
 self.roman\_to\_int = {'I': 1,'V': 5,'X': 10,'L': 50,'C': 100,'D': 500,'M': 1000}  
  
def convert(self):  
 total = 0  
 prev\_value = 0  
 for char in reversed(self.roman):  
 current\_value = self.roman\_to\_int[char]   
 if current\_value < prev\_value:  
 total -= current\_value  
 else:  
 total += current\_value  
 prev\_value = current\_value  
 return total

roman\_numeral = "IX" converter = RomanToInteger(roman\_numeral) integer\_value = converter.convert()

print(f"Roman numeral {roman\_numeral} is equal to {integer\_value}")

# Output:



**PROGRAM-5**: Write a program to create a class Shape with methods called get\_perimeter and get\_area. Create a subclass called Circle that overrides the get perimeter and get area methods to calculate the area and perimeter of a circle

**CODE** :-

import math class Shape:

def get\_perimeter(self):

pass

def get\_area(self):

pass

class Circle(Shape):

def **init**(self, radius):

self.radius = radius

def get\_perimeter(self):  
 return 2 \* math.pi \* self.radius  
  
def get\_area(self):  
 return math.pi \* (self.radius \*\* 2)

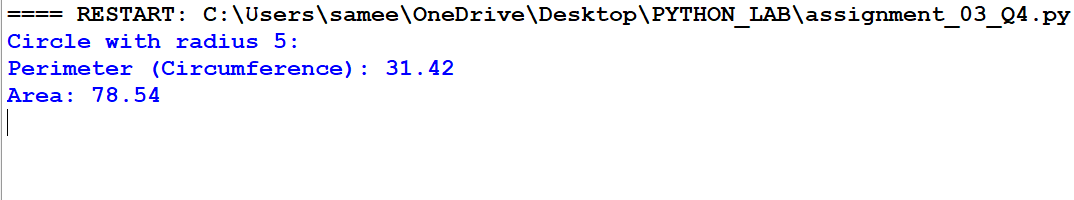
radius = 5 circle = Circle(radius)

print(f"Circle with radius {radius}:")

print(f"Perimeter (Circumference): {circle.get\_perimeter():.2f}")

print(f"Area: {circle.get\_area():.2f}")

**Output:**



**PROGRAM-6**: Write a program to create a Vehicle class hierarchy. The

bus class should be Vehicle, with subclasses Truck, Car, and Motorcycle. Each subclass should have properties such as make, model, year, and fuel type. Implement methods for calculating fuel efficiency, distance traveled, and maximum speed.

**CODE :-**

class Vehicle:

def **init**(self, make, model, year, fuel\_type):

self.make = make self.model = model self.year = year self.fuel\_type = fuel\_type

def fuel\_efficiency(self, distance, fuel\_consumed):  
 ""Calculate the fuel efficiency (km per liter).""  
 return distance / fuel\_consumed  
  
 def distance\_travelled(self, time, speed):  
 ""Calculate the distance travelled (km).""  
 return time \* speed  
  
 def maximum\_speed(self):  
 ""Return the maximum speed of the vehicle. Should be overridden by subclasses.""  
 pass

class Truck(Vehicle):

def **init**(self, make, model, year, fuel\_type, payload\_capacity): super().**init**(make, model, year, fuel\_type)

self.payload\_capacity = payload\_capacity

def maximum\_speed(self):  
 return 120  
  
 def fuel\_efficiency(self, distance, fuel\_consumed):  
 ""Override fuel\_efficiency to account for the truck's size and w eight.""  
 efficiency = super().fuel\_efficiency(distance, fuel\_consumed)  
 return efficiency - 1

class Car(Vehicle):

def **init**(self, make, model, year, fuel\_type, num\_doors):

super().**init**(make, model, year, fuel\_type) self.num\_doors = num\_doors

def maximum\_speed(self):  
 return 200

class Motorcycle(Vehicle):

def **init**(self, make, model, year, fuel\_type, engine\_capacity):

super().**init**(make, model, year, fuel\_type) self.engine\_capacity = engine\_capacity

def maximum\_speed(self):  
 return 180

truck = Truck("Ford", "F-150", 2020, "Diesel", 1000) car = Car("Toyota", "Camry", 2022, "Petrol", 4) motorcycle = Motorcycle("Harley-Davidson", "Sportster", 2021, "Petrol", 1200)

print(f"Truck - {truck.make} {truck.model} ({truck.year})")

print(f"Fuel Efficiency: {truck.fuel\_efficiency(500, 50):.2f} km/l") print(f"Distance Travelled (in 5 hours at 80 km/h): {truck.distance\_travelled(5, 80)} km")

print(f"Maximum Speed: {truck.maximum\_speed()} km/h") print()

print(f"Car - {car.make} {car.model} ({car.year})")

print(f"Fuel Efficiency: {car.fuel\_efficiency(500, 40):.2f} km/l")

print(f"Distance Travelled (in 5 hours at 100 km/h): {car.distance\_travelled(5, 100)} km")

print(f"Maximum Speed: {car.maximum\_speed()} km/h") print()

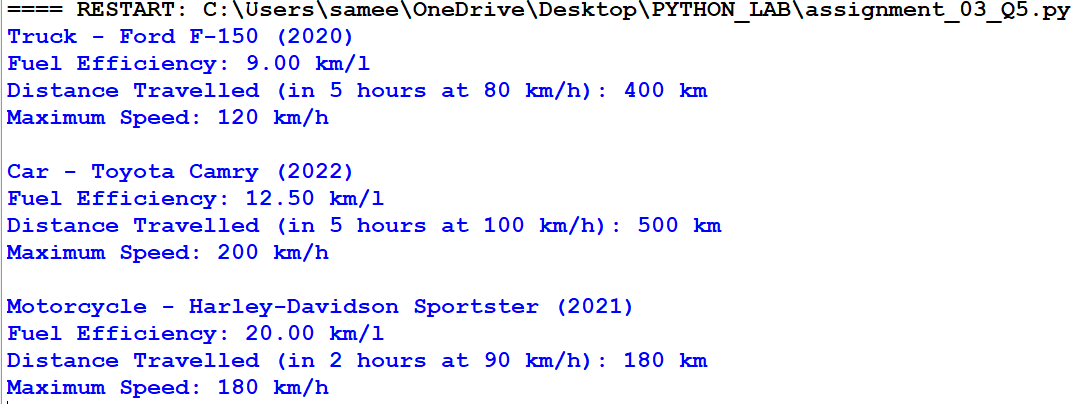
print(f"Motorcycle - {motorcycle.make} {motorcycle.model} ({motorcycle.year})")

print(f"Fuel Efficiency: {motorcycle.fuel\_efficiency(200, 10):.2f} km/l") print(f"Distance Travelled (in 2 hours at 90 km/h): {motorcycle.distance\_travelled(2, 90)} km")

print(f"Maximum Speed: {motorcycle.maximum\_speed()} km/h")

OUTPUT :-

**OUTPUT : -**



**PROGRAM-7**: Implement a class BankAccount with methods for deposit, withdrawal, and displaying the account balance.

**CODE** : -

class BankAccount:

def **init**(self, account\_holder, initial\_balance=0):

"""Initialize the BankAccount with account holder and an optional initial balance."""

self.account\_holder = account\_holder self.balance = initial\_balance

def deposit(self, amount):  
 """Deposit money into the account."""  
 if amount > 0:  
 self.balance += amount  
 print(f"Deposited {amount}. Current balance: {self.balance}")  
 else:  
 print("Deposit amount must be greater than 0.")  
  
def withdraw(self, amount):  
 """Withdraw money from the account."""  
 if amount > 0:  
 if self.balance >= amount:  
 self.balance -= amount  
 print(f"Withdrew {amount}. Current balance: {self.balance}")  
 else:  
 print("Insufficient funds.")  
 else:  
 print("Withdrawal amount must be greater than 0.")  
  
def display\_balance(self):  
 """Display the current account balance."""  
 print(f"Account balance: {self.balance}")

account = BankAccount("Tom cruise", 5000)

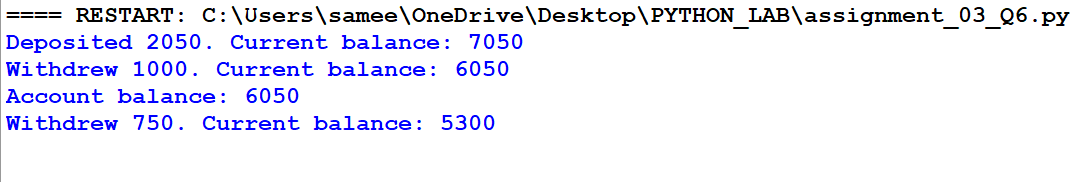
account.deposit(2050)

account.withdraw(1000)

account.display\_balance()

account.withdraw(750)

**OUTPUT :-**



**PROGRAM-8:** Develop a hierarchy of shapes (e.g., Shape, Circle, Rectangle) with area and perimeter.

**CODE :**-

import math class Shape:

def **init**(self):

pass

def area(self):

"""Calculate area of the shape. Should be overridden by subclasses."""

pass

def perimeter(self):

"""Calculate perimeter of the shape. Should be overridden by subclasses.""" pass

class Circle(Shape):

def **init**(self, radius):

super().**init**() self.radius = radius def area(self):

"""Area of a circle: π \* radius^2"""

return math.pi \* self.radius \*\* 2

def perimeter(self): """Perimeter (circumference) of a circle: 2 \* π \* radius""" return 2 \* math.pi \* self.radius

class Rectangle(Shape):

def **init**(self, length, width):

super().**init**() self.length = length self.width = width

def area(self):  
 """Area of a rectangle: length \* width"""  
 return self.length \* self.width  
  
 def perimeter(self):  
 """Perimeter of a rectangle: 2 \* (length + width)"""  
 return 2 \* (self.length + self.width)

class Square(Rectangle):

def **init**(self, side\_length):

super().**init**(side\_length, side\_length)

def area(self):

return self.length \*\* 2

def perimeter(self):  
 return 4 \* self.length

circle = Circle(5)

rectangle = Rectangle(4, 6)

square = Square(4)

print("Circle with radius 5:")

print(f"Area: {circle.area():.2f}")

print(f"Perimeter: {circle.perimeter():.2f}")

print()

print("Rectangle with length 4 and width 6:")

print(f"Area: {rectangle.area():.2f}")

print(f"Perimeter: {rectangle.perimeter():.2f}")

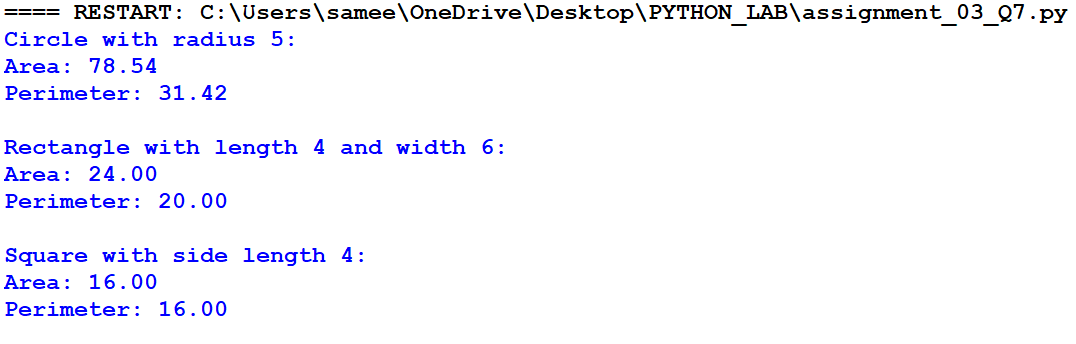
print()

print("Square with side length 4:")

print(f"Area: {square.area():.2f}")

print(f"Perimeter: {square.perimeter():.2f}")

**OUTPUT :-**



**PROGRAM-9:** Write a Python program to Overload the + operator for concatenating two custom string objects

**CODE :**-

class CustomString:

def **init**(self, value):

self.value = value

def **add**(self, other):

if isinstance(other, CustomString):

return CustomString(self.value + other.value)

else: raise TypeError("Operands must be of type CustomString")

def **str**(self):

return self.value str1 = CustomString("ZEAL") str2=CustomString("MCA") result = str1 + str2 print(result)

# Output:-

**PROGRAM-10:** Create a static method to validate user credentials (eg., email format, password length)

**CODE :**-

import re

class UserCredentials:

@staticmethod

def validate\_email(email):

email\_pattern = r'^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-.]+$'

if re.match(email\_pattern, email):

return True

else: return False

@staticmethod  
def validate\_password(password):  
 if len(password) >= 8:  
 return True  
 else:  
 return False

email = "[user@example.com](mailto:user@example.com)" password = "password123"

if UserCredentials.validate\_email(email):

print(f"Email '{email}' is valid.")

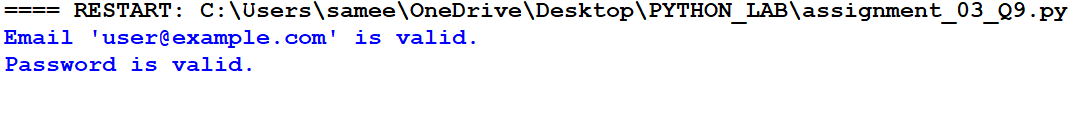
else: print(f"Email '{email}' is invalid.")

if UserCredentials.validate\_password(password):

print(f"Password is valid.")

else: print(f"Password is invalid.")

# Output:-



**PROGRAM-11:** Implement a class Library containing a list of Book objects.

**CODE :**-

class Book:

def **init**(self, title, author, year\_published):

self.title = title self.author = author

self.year\_published = year\_published

def \_\_str\_\_(self):  
 return f"'{self.title}' by {self.author}, published in {self.year\_published}"

class Library:

def **init**(self):

self.books = []

def add\_book(self, book):  
 if isinstance(book, Book):  
 self.books.append(book)  
 print(f"Book '{book.title}' added to th elibrary.")  
 else:  
 print("Only book objects can be added to the library.")  
  
 def remove\_book(self, title):  
 for book in self.books:  
 if book.title == title:  
 self.books.remove(book)  
 print(f"Book '{title}' removed from the library.")  
 return  
 print(f"Book '{title}' not found in the library.")  
  
   
 def display\_books(self):  
 if not self.books:  
 print("No books available in the library.")  
 else:  
 print("Books in the library:")  
 for book in self.books:  
 print(book)

book1 = Book("1984", "George Orwell", 1949)

book2 = Book("To Kill a Mockingbird", "Harper Lee", 1960)

book3 = Book("The Great Gatsby", "F. Scott Fitzgerald", 1925)

library = Library()

library.add\_book(book1) library.add\_book(book2) library.add\_book(book3)

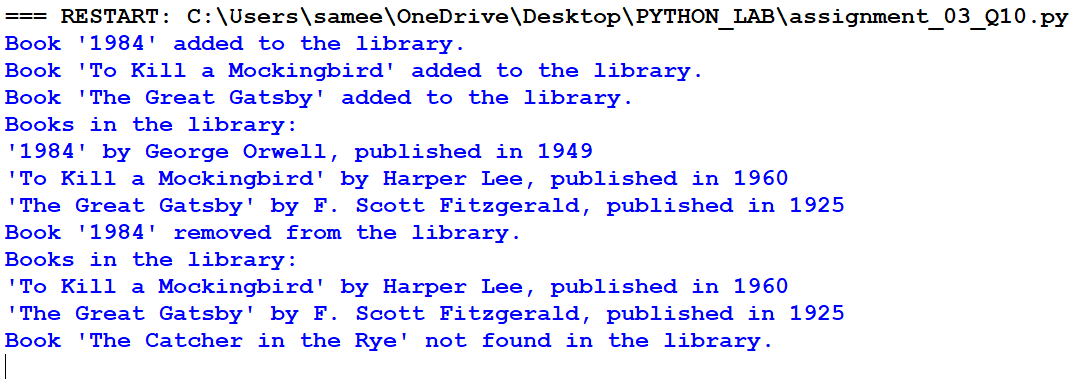
library.display\_books()

library.remove\_book("1984")

library.display\_books()

library.remove\_book("The Catcher in the Rye")

**Output:**



**PROGRAM 12 :-** Write a python program to Demonstrate method overriding in a superclass-subclass relationship for a payment system.

**CODE :-**

class Payment:

def **init**(self, amount):

self.amount = amount

def process\_payment(self):

raise NotImplementedError("Subclasses must implement this method.")

class CreditCardPayment(Payment):

def **init**(self, amount, card\_number):

super().**init**(amount) self.card\_number = card\_number

def process\_payment(self):  
 print(f"Processing credit card payment of ${self.amount} using card number {self.card\_number}.")

class PayPalPayment(Payment):

def **init**(self, amount, email):

super().**init**(amount) self.email = email

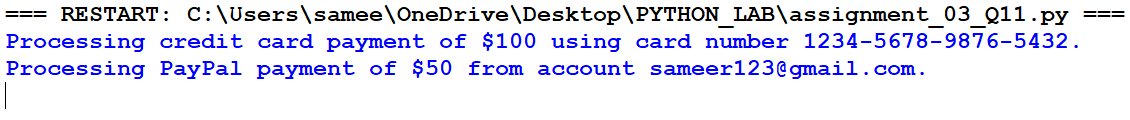
def process\_payment(self):  
 print(f"Processing PayPal payment of ${self.amount} from account {self.email}.")

credit\_card\_payment = CreditCardPayment(100, "1234-5678-9876-5432")

paypal\_payment = PayPalPayment(50, "sameer123@gmail.com")

credit\_card\_payment.process\_payment() paypal\_payment.process\_payment()

**OUTPUT :-**



**PROGRAM 13 :-** Write a program to validate a password that meets certain criteria (length, uppercase, digit, special character).

**CODE :-**

import re

class PasswordValidator:

@staticmethod

def validate\_password(password):

if len(password) < 8:

return "Password must be at least 8 characters long."

if not re.search(r'[A-Z]', password):

return "Password must contain at least one uppercase letter."

if not re.search(r'\d', password):

return "Password must contain at least one digit."

if not re.search(r'[!@#$%^&\*(),.?":{}|<>]', password):

return "Password must contain at least one special character."

return "Password is valid."

passwords = [

"password",

"Password1",

"Password@123",

"pass@123",

"P@ssw0rd"

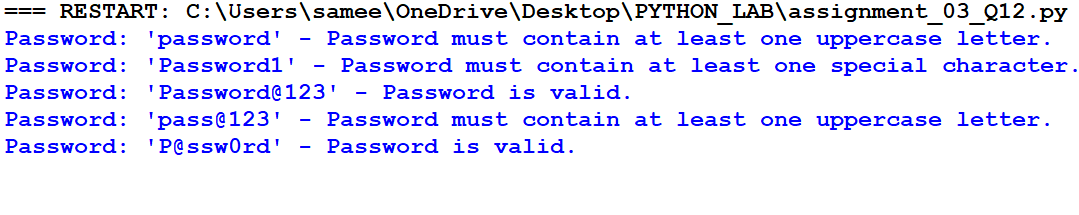
]

for password in passwords:

result = PasswordValidator.validate\_password(password)

print(f"Password: '{password}' - {result}")

**OUTPUT :-**



**PROGRAM 14 :-** Write a program Parse email addresses from a given text and classify them by domain.

**CODE :-**

import re from collections

import defaultdict

class EmailParser:

@staticmethod  
def parse\_emails(text):  
 """Extract all email addresses from the given text."""  
 email\_pattern = r'[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'  
 emails = re.findall(email\_pattern, text)  
 return emails  
  
@staticmethod  
def classify\_by\_domain(emails):  
 domain\_dict = defaultdict(list)  
   
 for email in emails:  
 domain = email.split('@')[1]  
 domain\_dict[domain].append(email)  
   
 return domain\_dict

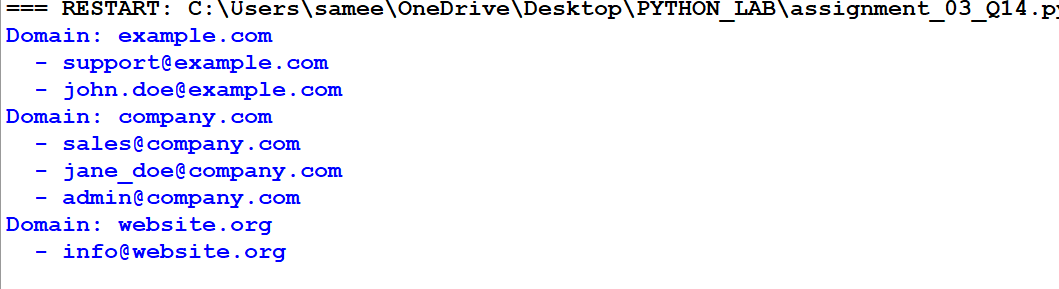
text = """ Contact us at [support@example.com](mailto:support@example.com) or [sales@company.com](mailto:sales@company.com). For inquiries, reach out to [john.doe@example.com](mailto:john.doe@example.com) or [jane\_doe@company.com](mailto:jane_doe@company.com). You can also email [admin@company.com](mailto:admin@company.com) or [info@website.org](mailto:info@website.org). """

emails = EmailParser.parse\_emails(text)

classified\_emails = EmailParser.classify\_by\_domain(emails)

for domain, email\_list in classified\_emails.items(): print(f"Domain: {domain}") for email in email\_list: print(f" - {email}")

**OUTPUT :-**



**PROGRAM 15 :-** Write a program to Create threads to perform independent tasks such as downloading multiple files Simultaneously.

**CODE :-**

import threading

import time

import requests

def download\_file(url, file\_name):

print(f"Started downloading: {file\_name} from {url}")

time.sleep(3)  
  
print(f"Completed downloading: {file\_name} from {url}")

file\_info = [ ("[http://example.com/file1"](http://example.com/file1%22), "file1.txt"),

("[http://example.com/file2"](http://example.com/file2%22), "file2.txt"), ("[http://example.com/file3"](http://example.com/file3%22), "file3.txt") ]

def download\_files\_concurrently(): threads = []

for url, file\_name in file\_info:  
 thread = threading.Thread(target=download\_file, args=(url, file\_name))  
 threads.append(thread)  
 thread.start()  
for thread in threads:  
 thread.join()  
print("All downloads are complete!")

download\_files\_concurrently()

**PROGRAM 16 :-** Write a Python program to create two threads. The first thread will print even numbers, and the second thread will print odd numbers. Demonstrate the concept of multithreading.

**CODE :-**

import threading

import time

def print\_even\_numbers():

for i in range(0, 20, 2):

print(f"Even: {i}")

time.sleep(0.5)

def print\_odd\_numbers():

for i in range(1, 20, 2):

print(f"Odd: {i}")

time.sleep(0.5)

even\_thread = threading.Thread(target=print\_even\_numbers)

odd\_thread = threading.Thread(target=print\_odd\_numbers)

even\_thread.start()

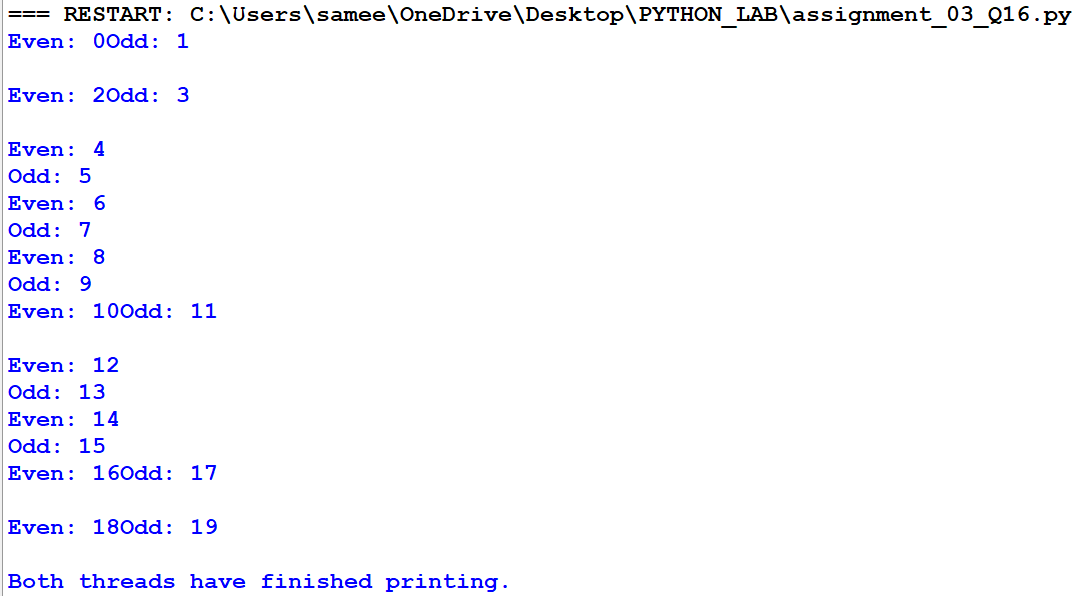
odd\_thread.start()

even\_thread.join()

odd\_thread.join()

print("Both threads have finished printing.")

**OUTPUT :-**



**PROGRAM 17 :-**

**CODE :-**

import threading

import math

def compute\_factorial(number):

try: result = math.factorial(number)

print(f"Factorial of {number} is {result}")

except Exception as e: print(f"Error computing factorial of {number}: {e}")

def main():

numbers = [5, 10, 15, 20, 25]

threads = []  
for num in numbers:  
 thread = threading.Thread(target=compute\_factorial, args=(num,))  
 threads.append(thread)  
 thread.start()  
  
for thread in threads:  
 thread.join()  
  
print("All computations are complete.")

if **name** == "**main**": main()

**OUTPUT :-**

