MILESTONE ASSIGNMENT

In [1]:
```
# Q.1 What is the  difference between static and dynamic variable in python?
#  Python doesn't have static variables in the same way as languages like C++ or Ja

# Understanding the Misconception
# The concept of static variables is often associated with languages that support c

# Dynamic Variables in Python
# In Python, all variables are dynamic. This means:

# Memory allocation: Variables are created at runtime when they are assigned a valu
# Data type: The type of a variable can change during program execution.
# Scope: Variables have a scope determined by the code block where they are defined
# Lifetime: Variables exist until they are no longer referenced and are automatical
# Example
# Python
def my_function():
    x = 10  # Local variable, created at runtime
    print(x)

x = 20  # Global variable, created at runtime

my_function()
print(x)
```

```
10
20
```

In [2]:
```
# Q.2 Explain the purpose of "pop","popitem","clear()" in a dictionary with suitabl
# Understanding pop, popitem, and clear() in Python Dictionaries

# POP()
# The pop() method removes and returns an element from a dictionary based on the sp
my_dict = {'apple': 3, 'banana': 2, 'orange': 1}

# Remove and return the value for 'apple'
removed_value = my_dict.pop('apple')
print(removed_value)  # Output: 3
print(my_dict)  # Output: {'banana': 2, 'orange': 1}

# Remove 'grape' and return a default value if not found
value = my_dict.pop('grape', 'Key not found')
print(value)

# POPITEMS()
# The popitem() method removes and returns an arbitrary (key, value) pair from the


my_dict = {'apple': 3, 'banana': 2, 'orange': 1}

# Remove and return the value for 'apple'
removed_value = my_dict.pop('apple')
print(removed_value)  # Output: 3
```

```
print(my_dict)  # Output: {'banana': 2, 'orange': 1}

# Remove 'grape' and return a default value if not found
value = my_dict.pop('grape', 'Key not found')
print(value)  # Output: Key not found

# CLEAR()
# The clear() method removes all elements from a dictionary, leaving it empty.

#Python
my_dict = {'apple': 3, 'banana': 2, 'orange': 1}

my_dict.clear()
print(my_dict)  # Output: {}
```

```
3
{'banana': 2, 'orange': 1}
Key not found
3
{'banana': 2, 'orange': 1}
Key not found
{}
```

In [3]:

```
# Q.3 What do you mean by frozenSet? Explain it with suitable examples.
# Frozenset in Python
# A frozenset is an immutable version of a Python set. This means that once a froze

# Key Characteristics:
# Immutable: Unchangeable after creation.
# Unordered: Elements have no specific order.
# Unique: No duplicate elements.

my_list = [1, 2, 3, 2, 4]
my_frozenset = frozenset(my_list)
print(my_frozenset)
```

```
frozenset({1, 2, 3, 4})
```

In [4]:

```
# Q.4 Difference between mutable and immutable data types in python and give exampl
# Mutable vs Immutable Data Types in Python
# Mutable Data Types
# Mutable data types are those whose values can be changed after they are created.
# Examples of mutable data types:

# Lists:
my_list = [1, 2, 3]
my_list.append(4)  # Modifying the list
print(my_list)


# Immutable Data Types
# Immutable data types are those whose values cannot be changed after they are crea

# Examples of immutable data types:

# Numbers (int, float, complex):
```

```
x = 10
x = x + 1  # Creates a new integer object
print(x)
```

```
[1, 2, 3, 4]
11
```

In [5]:
```
# Q.5 What is __init__? Explain with an examples.
# init in Python
# init is a special method in Python, often referred to as a constructor. It's auto

class Dog:
    def __init__(self, name, breed, age):
        self.name = name
        self.breed = breed
        self.age = age

    def bark(self):
        print(f"{self.name} barks!")

# Create objects
dog1 = Dog("Buddy", "Golden Retriever", 3)
dog2 = Dog("Max", "Labrador", 5)

# Access attributes and call methods
print(dog1.name)
dog2.bark()
```

```
Buddy
Max barks!
```

In [6]:
```
# Q.6 What is docstring in python ? Explain with an examples?
# Docstrings in Python
# Docstrings are strings that document a Python module, class, function, or method.

def add(x, y):
  """Adds two numbers and returns the sum."""
  return x + y

print(add.__doc__)
```

```
Adds two numbers and returns the sum.
```

In [7]:
```
# Q.8 What is break,continue and pass in python?

# The break :-
# statement is used to terminate the loop prematurely.
# When the break statement is encountered, the loop is immediately exited,
# and the control flow moves to the next statement after the loop.Break

for i in range(10):
    if i == 5:
        break
    print(i)

# Continue:-
# The continue statement is used to skip the rest of the current iteration of a loo
```

```
#  and move to the next iteration. The loop does not terminate,
#  but the code after the continue statement is not executed for the current iterat

for i in range(5):
    if i == 2:
        continue
    print(i)

# Pass :-
# The pass statement is a null operation.
# It does nothing. It is often used as a placeholder when a statement
# is syntactically required but no code needs to be executed.

def function():
    pass
```

```
0
1
2
3
4
0
1
3
4
```

In [8]:

```
# Q.15 What are decorators in python? Explain it with an example.Write down its use
# Decorators in Python
# A decorator is a function that takes another function as an argument, adds some f

def decorator_function(original_function):
    def wrapper_function(*args, **kwargs):
        print("This is the decorator function before the original function")
        result = original_function(*args, **kwargs)
        print("This is the decorator function after the original function")
        return result
    return wrapper_function

@decorator_function
def my_function(x, y):
    print(f"x is {x} and y is {y}")
    return x + y

result = my_function(3, 4)
print(result)
```

```
This is the decorator function before the original function
x is 3 and y is 4
This is the decorator function after the original function
7
```

In [9]:

```
# Q.17 What is lambda in python? why is it used?
# Lambda Functions in Python
# Lambda functions are anonymous functions, meaning they don't have a specific name

add = lambda x, y: x + y
```

```
result = add(3, 4)
print(result)
```

```
7
```

In [10]:
```
# Q.18 Explain split() and join() function in python?
# split() and join() in Python
# split():-
# The split() method in Python is used to break a string into a list of substrings

text = "This is a sample string"
words = text.split()  # Split by whitespace
print(words)  # Output: ['This', 'is', 'a', 'sample', 'string']

numbers = "1,2,3,4,5"
number_list = numbers.split(",")  # Split by comma
print(number_list)

# join():-
# The join() method is used to join the elements of an iterable (like a list or tup

words = ["hello", "world"]
joined_string = " ".join(words)  # Join with space
print(joined_string)  # Output: hello world

number_list = ["1", "2", "3"]
joined_numbers = "-".join(number_list)  # Join with hyphen
print(joined_numbers)
```

```
['This', 'is', 'a', 'sample', 'string']
['1', '2', '3', '4', '5']
hello world
1-2-3
```

In [11]:
```
# Q.19 What are iterators,iterable & generators in python?

#(1)  Iterables
# An iterable is any object that can be iterated over. This means you can use a for
my_list = [1, 2, 3]  # A list is iterable
for num in my_list:
    print(num)

#(2)Iterators
# An iterator is an object that implements the iterator protocol. It has two method

my_list = [1, 2, 3]
my_iterator = iter(my_list)

print(next(my_iterator))
print(next(my_iterator))
print(next(my_iterator))

# Generators
# Generators are a special type of iterator created using the yield keyword. They p

def my_generator():
    yield 1
```

```
    yield 2
    yield 3

for num in my_generator():
    print(num)
```

```
1
2
3
1
2
3
1
2
3
```

In [12]:

```
# Q.22 How will you check if a class is child  of another class?
# Checking if a Class is a Child of Another Class in Python
#Python provides the issubclass() function to determine if a class is a subclass of
class Animal:
    pass

class Dog(Animal):
    pass

class Cat(Animal):
    pass

class GoldenRetriever(Dog):
    pass

print(issubclass(Dog, Animal))  # Output: True
print(issubclass(Cat, Dog))  # Output: False
print(issubclass(GoldenRetriever, Animal))
```

```
True
False
True
```

In [13]:

```
# Q.25 What is polymorphism? Explain its with an example.

# Understanding Polymorphism
# Imagine you have a function that expects an animal object. You want to call a mak

class Animal:
    def make_sound(self):
        print("Generic animal sound")

class Dog(Animal):
    def make_sound(self):
        print("Woof!")

class Cat(Animal):
    def make_sound(self):
        print("Meow!")
```

```
def animal_sound(animal):
    animal.make_sound()


# Create objects
dog = Dog()
cat = Cat()

# Call the function with different animal objects
animal_sound(dog)  # Output: Woof!
animal_sound(cat)  # Output: Meow!
```

```
Woof!
Meow!
```

In [14]:
```
# Q.27 name=["mohan","dash","karam","chandra","gandhi","bapu"] do the following ope
# a) add an  element "freedom_fighter" in the list at the 0 index

name = ["mohan", "dash", "karam", "chandra", "gandhi", "bapu"]

# Add "freedom_fighter" at index 0
name.insert(0, "freedom_fighter")

print(name)
```

```
['freedom_fighter', 'mohan', 'dash', 'karam', 'chandra', 'gandhi', 'bapu']
```

In [15]:
```
# Q.28 name=["mohan","dash","karam","chandra","gandhi","bapu"] do the following ope

# b) find the output of the following , and explain how?

name=["freedomfighter","Bapuji","mohan","dash","karam","chandra","gandhi"]
length1=len((name[-len(name)+1:-1:2]))
length2=len((name[-len(name)+1:-1]))
print(length1+length2)
```

```
8
```

In [16]:
```
# Q.28 name=["mohan","dash","karam","chandra","gandhi","bapu"] do the following ope
# c) add two more elements in the name["Netaji","bose"] at the end of the list.

name = ["mohan", "dash", "karam", "chandra", "gandhi", "bapu"]

# Add two more elements to the end of the list
name.extend(["Netaji", "bose"])

print(name)
```

```
['mohan', 'dash', 'karam', 'chandra', 'gandhi', 'bapu', 'Netaji', 'bose']
```

In [17]:
```
# Q.28 name = ["mohan", "dash", "karam", "chandra", "gandhi", "bapu"]

# d) What will be the value of temp:

name=["bapuji","dash","karam","chandra","gandhi","mohan"]
temp=name[-1]
name[-1]=name[0]
```

```
name[0]=temp
print(name)
```

```
['mohan', 'dash', 'karam', 'chandra', 'gandhi', 'bapuji']
```

In [18]:
```
# Q.29 Find the output of the following
animal = ['human','cat','mat','cat','rat','human','lion']
print(animal.count("human"))
print(animal.index("rat"))
print(len(animal))
```

```
2
4
7
```

In [19]:
```
# Q.30 Tuple=(10,20,"Apple",3.4,'a',["master","ji"],("sita","geeta",22)[{"roll_no":
# a) print(len(Tuple))

Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"rol

print(len(Tuple))
```

```
8
```

In [20]:
```
# Q.30 Tuple=(10,20,"Apple",3.4,'a',["master","ji"],("sita","geeta",22)[{"roll_no":

# b) print(Tuple[-1][-1]["name"])

Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"rol

print(Tuple[-1][-1]["name"])
```

```
navneet
```

In [21]:
```
# Q.30 Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22),

# C) fetch the values of roll_no from this tuple.

Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"rol

# Access the last element of the tuple, which is a list of dictionaries
list_of_dicts = Tuple[-1]

# Extract the roll_no value from the first dictionary
roll_no = list_of_dicts[0]["roll_no"]

print(roll_no)
```

```
1
```

In [22]:
```
# Q.30 Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22),

# d) print(Tuple[-3][1])
Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"rol

print(Tuple[-3][1])
```

```
ji
```

In [23]:
```
# Q.30 Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22),

#  e) fetch the element '22' from this tuple.

Tuple = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"rol

result = Tuple[6][2]
print(result)  # This will output: 22
```

```
22
```

In [24]:
```
# Q.31 Write a program to display the appropriate message as per the colour of sign

def display_signal_message(signal_color):
    # Convert the input to lowercase to handle different cases
    signal_color = signal_color.lower()

    # Determine the message based on the signal color
    if signal_color == 'red':
        return "Stop"
    elif signal_color == 'yellow':
        return "Stay"
    elif signal_color == 'green':
        return "Go"
    else:
        return "Invalid color! Please enter 'red', 'yellow', or 'green'."

def main():
    # Prompt the user to enter the color of the signal
    signal_color = input("Enter the color of the traffic signal (red, yellow, green

    # Display the appropriate message
    message = display_signal_message(signal_color)
    print(message)

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

```
Stop
```

In [25]:
```
# Q.32 Write a program to create a simple calculator performing only four basic ope
def calculator():
  """Performs basic arithmetic operations."""

  while True:
    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Divide")
    print("4. Multiply")
    print("5. Exit")

    choice = input("Enter choice (1/2/3/4/5): ")

    if choice in ('1', '2', '3', '4'):
```

```
      num1 = float(input("Enter first number: "))
      num2 = float(input("Enter second number: "))

      if choice == '1':
        print(num1, "+", num2,"=", num1 + num2)
      elif choice == '2':
        print(num1, "-", num2, "=", num1 - num2)
      elif choice == '3':
        if num2 == 0:
          print("Error: Division by zero")
      else:
          print(num1, "/", num2, "=", num1 / num2)
    elif choice == '4':
      print(num1, "*", num2, "=", num1 * num2)

    elif choice == '5':
      break

    else:
      print("Invalid input")

calculator()
```

```
Select operation:
1. Add
2. Subtract
3. Divide
4. Multiply
5. Exit

5.0 + 8.0 = 13.0
Select operation:
1. Add
2. Subtract
3. Divide
4. Multiply
5. Exit

8.0 - 3.0 = 5.0
Select operation:
1. Add
2. Subtract
3. Divide
4. Multiply
5. Exit

Select operation:
1. Add
2. Subtract
3. Divide
4. Multiply
5. Exit
```

In [26]:

```
# Q.33 Write a program to find the larger of the three pre-specified numbers using

num1 = 10
num2 = 25
```

```
num3 = 15

largest = num1 if (num1 >= num2 and num1 >= num3) else (num2 if num2 >= num3 else n

print("The largest number is:", largest)
```

```
The largest number is: 25
```

In [27]:

```
# Q.34 Write a program to find the factor of whole number using a while loop.
def find_factors(number):
  """Finds the factors of a given number using a while loop.

  Args:
    number: The number to find factors for.

  Returns:
    A list of factors of the number.
  """

  factors = []
  divisor = 1

  while divisor <= number:
    if number % divisor == 0:
      factors.append(divisor)
    divisor += 1

  return factors

# Get input from the user
num = int(input("Enter a whole number: "))

# Find and print the factors
factors = find_factors(num)
print("Factors of", num, "are:", factors)
```

```
Factors of 5 are: [1, 5]
```

In [28]:

```
# Q.35 Write a program to find the sum of all the positive numbers entered by the u

def sum_positive_numbers():
  """Calculates the sum of positive numbers entered by the user."""

  total = 0
  number = 0

  while number >= 0:
    number = int(input("Enter a positive number (or a negative number to stop): "))
    if number >= 0:
      total += number

  print("The sum of the positive numbers is:", total)

sum_positive_numbers()
```

```
The sum of the positive numbers is: 10
```

In [29]:
```python
# Q.36 Write a program to find prime numbers between 2 to 100 using nested for loop

def sum_positive_numbers():
  """Calculates the sum of positive numbers entered by the user."""

  total = 0
  number = 0

  while number >= 0:
    number = int(input("Enter a positive number (or a negative number to stop): "))
    if number >= 0:
      total += number

  print("The sum of the positive numbers is:", total)

sum_positive_numbers()
```

```
The sum of the positive numbers is: 25
```

In [30]:
```python
# Q.36 Write a program for the following.
# a) accept the marks of the student in five major subject and display the same.
#  Criteria                                        Grade
  # percentage > 85                                    A
  # percentage <85 & percentage >=75                   B
  # percentage <75 & percentage >=50                   C
  # percentage >30 & percentage <=50                   D
  # percentage <30                                     E

def calculate_grade(marks):
  """Calculates and displays the grade based on the given marks.

  Args:
    marks: A list of marks in five subjects.
  """

  total_marks = sum(marks)
  percentage = (total_marks / 500) * 100

  print("Total Marks:", total_marks)
  print("Percentage:", percentage)

  if percentage > 85:
    grade = "A"
  elif percentage >= 75:
    grade = "B"
  elif percentage >= 50:
    grade = "C"
  elif percentage >= 30:
    grade = "D"
  else:
    grade = "E"

  print("Grade:", grade)
```

```
# Get marks from the user
subjects = ["English", "Maths", "Science", "Social Science", "Hindi"]
marks = []
for subject in subjects:
  mark = int(input(f"Enter marks for {subject}: "))
  marks.append(mark)

# Calculate and display grade
calculate_grade(marks)
```

```
Total Marks: 419
Percentage: 83.8
Grade: B
```

In [31]:

```
# Q.36 Write a program for the following.
 # b) Calculate the sum of the marks of all subject .Divide the total marks of the
# Criteria                                        Grade
  # percentage > 85                                    A
  # percentage <85 & percentage >=75                   B
  # percentage <75 & percentage >=50                   C
  # percentage >30 & percentage <=50                   D
  # percentage <30                                     E

def calculate_grade():
  """Calculates and displays the percentage and grade based on given marks."""

  subjects = ["English", "Maths", "Science", "Social Science", "Hindi"]
  marks = []

  for subject in subjects:
    mark = int(input(f"Enter marks for {subject}: "))
    marks.append(mark)

  total_marks = sum(marks)
  percentage = (total_marks / 500) * 100

  print("Total Marks:", total_marks)
  print("Percentage:", percentage)

  if percentage > 85:
    grade = "A"
  elif percentage >= 75:
    grade = "B"
  elif percentage >= 50:
    grade = "C"
  elif percentage >= 30:
    grade = "D"
  else:
    grade = "E"

  print("Grade:", grade)

calculate_grade()
```

```
Total Marks: 491
Percentage: 98.2
Grade: A
```

In [32]:

```python
# Q.36 Write a program for the following.
 # c) find the grade of the student as per the following criteria. Hint use match &
# Criteria                                        Grade
  # percentage > 85                                    A
  # percentage <85 & percentage >=75                   B
  # percentage <75 & percentage >=50                   C
  # percentage >30 & percentage <=50                   D
  # percentage <30                                     E

def calculate_grade(percentage):
  """Calculates the grade based on the given percentage using match-case.

  Args:
    percentage: The student's percentage.

  Returns:
    The grade corresponding to the percentage.
  """

  match percentage:
    case percentage if percentage > 85:
      return "A"
    case percentage if percentage >= 75:
      return "B"
    case percentage if percentage >= 50:
      return "C"
    case percentage if percentage >= 30:
      return "D"
    case _:
      return "E"

def main():
  subjects = ["English", "Maths", "Science", "Social Science", "Hindi"]
  marks = []

  for subject in subjects:
    mark = int(input(f"Enter marks for {subject}: "))
    marks.append(mark)

  total_marks = sum(marks)
  percentage = (total_marks / 500) * 100

  print("Total Marks:", total_marks)
  print("Percentage:", percentage)

  grade = calculate_grade(percentage)
  print("Grade:", grade)

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

```
Total Marks: 344
Percentage: 68.8
Grade: C
```

In [33]:
```
# Q.37 Write a program for VIBGYOR Spectrum based on their Wavelength using wavelen
#  COLOUR                              Wavelength(nm)
#  violet                              400 to 440
#  indigo                              440 to 460
#  blue                                460 to 500
#  green                               500 to 570
#  yellow                              570 to 590
#  orange                              590 to 620
#  red                                 620 to 720

def get_color_from_wavelength(wavelength):
  """Determines the color based on the given wavelength.

  Args:
    wavelength: The wavelength of the light in nanometers.

  Returns:
    The color corresponding to the wavelength, or 'Unknown' if not in range.
  """

  if 400 <= wavelength < 440:
    return 'violet'
  elif 440 <= wavelength < 460:
    return 'indigo'
  elif 460 <= wavelength < 500:
    return 'blue'
  elif 500 <= wavelength < 570:
    return 'green'
  elif 570 <= wavelength < 590:
    return 'yellow'
  elif 590 <= wavelength < 620:
    return 'orange'
  elif 620 <= wavelength <= 720:
    return 'red'
  else:
    return 'Unknown'

# Example usage:
wavelength = float(input("Enter the wavelength in nanometers: "))
color = get_color_from_wavelength(wavelength)
print("The color is:", color)
```

```
The color is: red
```

In [34]:
```
# Q.38 Consider the gravitational interactions between the earth,moon,and sun in ou
# mass_earth=5.972e24 # mass of earth in kilograms
# mass_moon=7.34767309e22 # mass of moon in kilograms
# mass_sun=1.989e30 # mass of sun in kilograms

# distance_earth_sun=1.496e11 # distance between earth and sun in meters
# distance_moon_earth=3.844e8 # distance between moon and earth in meters
```

```
#  a) Calculate the gravitational force between the earth and the sun.

import math

# Given values
G = 6.67430e-11  # Gravitational constant
mass_earth = 5.972e24
mass_sun = 1.989e30
distance_earth_sun = 1.496e11

# Calculate the gravitational force
force_earth_sun = G * (mass_earth * mass_sun) / (distance_earth_sun**2)

print("Gravitational force between Earth and Sun:", force_earth_sun, "N")
```

```
Gravitational force between Earth and Sun: 3.5423960813684973e+22 N
```

In [1]:

```
# Q.38 Consider the gravitational interactions between the earth,moon,and sun in ou
# mass_earth=5.972e24 # mass of earth in kilograms
# mass_moon=7.34767309e22 # mass of moon in kilograms
# mass_sun=1.989e30 # mass of sun in kilograms

# distance_earth_sun=1.496e11 # distance between earth and sun in meters
# distance_moon_earth=3.844e8 # distance between moon and earth in meters

# b) Calculate the gravitational force between the moon and the earth.

import math

# Given values
G = 6.67430e-11  # Gravitational constant
mass_earth = 5.972e24
mass_moon = 7.34767309e22
distance_moon_earth = 3.844e8

# Calculate the gravitational force
force_moon_earth = G * (mass_earth * mass_moon) / (distance_moon_earth**2)

print("Gravitational force between Moon and Earth:", force_moon_earth, "N")
```

```
Gravitational force between Moon and Earth: 1.9820225456526813e+20 N
```

In [2]:

```
# Q.39 Design and implement a python program for managing student information using
# a) Define the `student` class with encapsulated attributes.

class Student:
    def __init__(self, name, age, roll_number):
        self.__name = name
        self.__age = age
        self.__roll_number = roll_number

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name
```

```
    def get_age(self):
        return self.__age

    def set_age(self, age):
        if age >= 0:
            self.__age = age
        else:
            raise ValueError("Age cannot be negative")

    def get_roll_number(self):
        return self.__roll_number

    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number
```

In [3]:
```
# Q.39 Design and implement a python program for managing student information using
# b) Implement getter and setter methods for the attributes.

class Student:
    def __init__(self, name, age, roll_number):
        self.__name = name
        self.__age = age
        self.__roll_number = roll_number

    # Getter methods
    def get_name(self):
        return self.__name

    def get_age(self):
        return self.__age

    def get_roll_number(self):
        return self.__roll_number

    # Setter methods
    def set_name(self, name):
        self.__name = name

    def set_age(self, age):
        if age >= 0:
            self.__age = age
        else:
            raise ValueError("Age cannot be negative")

    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number
```

In [4]:
```
# Q.39 Design and implement a python program for managing student information using
# c) Write methods to display student information and update details.

class Student:
    def __init__(self, name, age, roll_number):
        self.__name = name
        self.__age = age
        self.__roll_number = roll_number
```

```
    # Getter methods
    def get_name(self):
        return self.__name

    def get_age(self):
        return self.__age

    def get_roll_number(self):
        return self.__roll_number

    # Setter methods
    def set_name(self, name):
        self.__name = name

    def set_age(self, age):
        if age >= 0:
            self.__age = age
        else:
            raise ValueError("Age cannot be negative")

    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number

    # Display student information
    def display_student_info(self):
        print("Name:", self.__name)
        print("Age:", self.__age)
        print("Roll Number:", self.__roll_number)

    # Update student details
    def update_student_details(self, new_name, new_age, new_roll_number):
        self.set_name(new_name)
        self.set_age(new_age)
        self.set_roll_number(new_roll_number)
```

In [5]:

```
#  Q.39 Design and implement a python program for managing student information usin
#  d) Create instance of the `student` class and test the implemented functionality

class Student:
    def __init__(self, name, age, roll_number):
        self.__name = name
        self.__age = age
        self.__roll_number = roll_number

    # Getter methods
    def get_name(self):
        return self.__name

    def get_age(self):
        return self.__age

    def get_roll_number(self):
        return self.__roll_number

    # Setter methods
```

```python
    def set_name(self, name):
        self.__name = name

    def set_age(self, age):
        if age >= 0:
            self.__age = age
        else:
            raise ValueError("Age cannot be negative")

    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number

    # Display student information
    def display_student_info(self):
        print("Name:", self.__name)
        print("Age:", self.__age)
        print("Roll Number:", self.__roll_number)

    # Update student details
    def update_student_details(self, new_name, new_age, new_roll_number):
        self.set_name(new_name)
        self.set_age(new_age)
        self.set_roll_number(new_roll_number)

# Create instances of the Student class
student1 = Student("Alice", 20, 12345)
student2 = Student("Bob", 22, 54321)

# Test the functionality
student1.display_student_info()

print("\nUpdating student1 details...")
student1.update_student_details("Alice Updated", 21, 12346)
student1.display_student_info()

print("\nAccessing student2 details using getters:")
print("Name:", student2.get_name())
print("Age:", student2.get_age())
print("Roll Number:", student2.get_roll_number())
```

```
Name: Alice
Age: 20
Roll Number: 12345

Updating student1 details...
Name: Alice Updated
Age: 21
Roll Number: 12346

Accessing student2 details using getters:
Name: Bob
Age: 22
Roll Number: 54321
```

In [6]:

```python
# Q.40 Develop a python program for managing library resources efficiently. design
# a) Create the `LibraryBook` class with encaptulated attributes.
```

```
class LibraryBook:
    def __init__(self, book_name, author):
        self.__book_name = book_name
        self.__author = author
        self.__is_available = True

    def get_book_name(self):
        return self.__book_name

    def get_author(self):
        return self.__author

    def is_available(self):
        return self.__is_available
```

In [7]:
```
# Q.40 Develop a python program for managing library resources efficiently. design
# b) Implement methods for borrowing and returning books

class LibraryBook:
    def __init__(self, book_name, author):
        self.__book_name = book_name
        self.__author = author
        self.__is_available = True

    def get_book_name(self):
        return self.__book_name

    def get_author(self):
        return self.__author

    def is_available(self):
        return self.__is_available

    def borrow_book(self):
        if self.__is_available:
            self.__is_available = False
            print(f"{self.__book_name} by {self.__author} has been borrowed.")
        else:
            print(f"{self.__book_name} is currently unavailable.")

    def return_book(self):
        if not self.__is_available:
            self.__is_available = True
            print(f"{self.__book_name} by {self.__author} has been returned.")
        else:
            print(f"{self.__book_name} is already available.")
```

In [8]:
```
# Q.40 Develop a python program for managing library resources efficiently. design
# d) Test the borrowing and returning functionality with sample data.
class LibraryBook:
    def __init__(self, book_name, author):
        self.__book_name = book_name
        self.__author = author
        self.__availability_status = True  # True indicates the book is available
```

```
    def borrow_book(self):
        if self.__availability_status:
            self.__availability_status = False
            print(f"You have borrowed '{self.__book_name}' by {self.__author}.")
        else:
            print(f"Sorry, '{self.__book_name}' is currently unavailable.")

    def return_book(self):
        if not self.__availability_status:
            self.__availability_status = True
            print(f"Thank you for returning '{self.__book_name}'.")
        else:
            print(f"'{self.__book_name}' was not borrowed, so it cannot be returned

    def get_book_info(self):
        status = "Available" if self.__availability_status else "Unavailable"
        return f"Book: {self.__book_name}, Author: {self.__author}, Status: {status

# Testing the borrowing and returning functionality with sample data

book1 = LibraryBook("1984", "George Orwell")
book2 = LibraryBook("To Kill a Mockingbird", "Harper Lee")

# Display initial status
print(book1.get_book_info())
print(book2.get_book_info())

# Borrow books
book1.borrow_book()
book2.borrow_book()

# Try borrowing the same book again
book1.borrow_book()

# Display status after borrowing
print(book1.get_book_info())
print(book2.get_book_info())

# Return books
book1.return_book()
book2.return_book()

# Try returning the same book again
book1.return_book()

# Display status after returning
print(book1.get_book_info())
print(book2.get_book_info())
```

```
Book: 1984, Author: George Orwell, Status: Available
Book: To Kill a Mockingbird, Author: Harper Lee, Status: Available
You have borrowed '1984' by George Orwell.
You have borrowed 'To Kill a Mockingbird' by Harper Lee.
Sorry, '1984' is currently unavailable.
Book: 1984, Author: George Orwell, Status: Unavailable
Book: To Kill a Mockingbird, Author: Harper Lee, Status: Unavailable
Thank you for returning '1984'.
Thank you for returning 'To Kill a Mockingbird'.
'1984' was not borrowed, so it cannot be returned.
Book: 1984, Author: George Orwell, Status: Available
Book: To Kill a Mockingbird, Author: Harper Lee, Status: Available
```

In [14]:

```python
# Q.41 Create a simple banking system using object-oriented concepts in python.Desi
# a) Define base class (es) for bank accounts with common attributes and methods.

class BankAccount:
    def __init__(self, account_number, account_holder, balance=0.0):
        self.__account_number = account_number
        self.__account_holder = account_holder
        self.__balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposited ${amount:.2f}. New balance is ${self.__balance:.2f}."
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if self.__balance >= amount:
                self.__balance -= amount
                print(f"Withdrew ${amount:.2f}. New balance is ${self.__balance:.2f
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def get_balance(self):
        return f"Account balance: ${self.__balance:.2f}"

    def get_account_info(self):
        return f"Account Number: {self.__account_number}, Account Holder: {self.__a

class SavingsAccount(BankAccount):
    def __init__(self, account_number, account_holder, balance=0.0, interest_rate=0
        super().__init__(account_number, account_holder, balance)
        self.__interest_rate = interest_rate

    def add_interest(self):
        interest = self._BankAccount__balance * self.__interest_rate
        self.deposit(interest)
        print(f"Interest added: ${interest:.2f}. New balance is ${self._BankAccount

class CheckingAccount(BankAccount):
```

```
    def __init__(self, account_number, account_holder, balance=0.0, overdraft_limit
        super().__init__(account_number, account_holder, balance)
        self.__overdraft_limit = overdraft_limit

    def withdraw(self, amount):
        if amount > 0:
            if self._BankAccount__balance + self.__overdraft_limit >= amount:
                self._BankAccount__balance -= amount
                print(f"Withdrew ${amount:.2f}. New balance is ${self._BankAccount_
            else:
                print("Insufficient funds and overdraft limit reached.")
        else:
            print("Withdrawal amount must be positive.")
# Creating accounts
savings = SavingsAccount("SA12345", "John Doe", 1000.0, 0.03)
checking = CheckingAccount("CA12345", "Jane Smith", 500.0, 300.0)

# Deposit money
savings.deposit(500)
checking.deposit(200)

# Withdraw money
savings.withdraw(200)
checking.withdraw(700)

# Add interest to savings account
savings.add_interest()

# Check balances
print(savings.get_balance())
print(checking.get_balance())

# Account information
print(savings.get_account_info())
print(checking.get_account_info())
```

```
Deposited $500.00. New balance is $1500.00.
Deposited $200.00. New balance is $700.00.
Withdrew $200.00. New balance is $1300.00.
Withdrew $700.00. New balance is $0.00.
Deposited $39.00. New balance is $1339.00.
Interest added: $39.00. New balance is $1339.00.
Account balance: $1339.00
Account balance: $0.00
Account Number: SA12345, Account Holder: John Doe, Balance: $1339.00
Account Number: CA12345, Account Holder: Jane Smith, Balance: $0.00
```

In [15]:

```
# Q.41 Create a simple banking system using object-oriented concepts in python.Desi
# b) Implement subclasses for specific account types(e.g.,savingAccount,CheckingAcc
class BankAccount:
    def __init__(self, account_number, account_holder, balance=0.0):
        self.__account_number = account_number
        self.__account_holder = account_holder
        self.__balance = balance

    def deposit(self, amount):
```

```
        if amount > 0:
            self.__balance += amount
            print(f"Deposited ${amount:.2f}. New balance is ${self.__balance:.2f}."
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if self.__balance >= amount:
                self.__balance -= amount
                print(f"Withdrew ${amount:.2f}. New balance is ${self.__balance:.2f
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def get_balance(self):
        return f"Account balance: ${self.__balance:.2f}"

    def get_account_info(self):
        return f"Account Number: {self.__account_number}, Account Holder: {self.__a
class SavingsAccount(BankAccount):
    def __init__(self, account_number, account_holder, balance=0.0, interest_rate=0
        super().__init__(account_number, account_holder, balance)
        self.__interest_rate = interest_rate

    def add_interest(self):
        interest = self._BankAccount__balance * self.__interest_rate
        self.deposit(interest)
        print(f"Interest added: ${interest:.2f}. New balance is ${self._BankAccount
class CheckingAccount(BankAccount):
    def __init__(self, account_number, account_holder, balance=0.0, overdraft_limit
        super().__init__(account_number, account_holder, balance)
        self.__overdraft_limit = overdraft_limit

    def withdraw(self, amount):
        if amount > 0:
            if self._BankAccount__balance + self.__overdraft_limit >= amount:
                self._BankAccount__balance -= amount
                print(f"Withdrew ${amount:.2f}. New balance is ${self._BankAccount_
            else:
                print("Insufficient funds and overdraft limit reached.")
        else:
            print("Withdrawal amount must be positive.")
# Creating accounts
savings = SavingsAccount("SA12345", "John Doe", 1000.0, 0.03)
checking = CheckingAccount("CA12345", "Jane Smith", 500.0, 300.0)

# Deposit money
savings.deposit(500)
checking.deposit(200)

# Withdraw money
savings.withdraw(200)
checking.withdraw(700)
```

```
# Add interest to savings account
savings.add_interest()

# Check balances
print(savings.get_balance())
print(checking.get_balance())

# Account information
print(savings.get_account_info())
print(checking.get_account_info())
```

```
Deposited $500.00. New balance is $1500.00.
Deposited $200.00. New balance is $700.00.
Withdrew $200.00. New balance is $1300.00.
Withdrew $700.00. New balance is $0.00.
Deposited $39.00. New balance is $1339.00.
Interest added: $39.00. New balance is $1339.00.
Account balance: $1339.00
Account balance: $0.00
Account Number: SA12345, Account Holder: John Doe, Balance: $1339.00
Account Number: CA12345, Account Holder: Jane Smith, Balance: $0.00
```

In [16]:

```
# Q.41 Create a simple banking system using object-oriented concepts in python.Desi
# d) Test the banking system by creating instance of different accounts types and p
class BankAccount:
    def __init__(self, account_number, account_holder, balance=0.0):
        self.__account_number = account_number
        self.__account_holder = account_holder
        self.__balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposited ${amount:.2f}. New balance is ${self.__balance:.2f}."
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if self.__balance >= amount:
                self.__balance -= amount
                print(f"Withdrew ${amount:.2f}. New balance is ${self.__balance:.2f
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def get_balance(self):
        return f"Account balance: ${self.__balance:.2f}"

    def get_account_info(self):
        return f"Account Number: {self.__account_number}, Account Holder: {self.__a


class SavingsAccount(BankAccount):
    def __init__(self, account_number, account_holder, balance=0.0, interest_rate=0
```

```
        super().__init__(account_number, account_holder, balance)
        self.__interest_rate = interest_rate

    def add_interest(self):
        interest = self._BankAccount__balance * self.__interest_rate
        self.deposit(interest)
        print(f"Interest added: ${interest:.2f}. New balance is ${self._BankAccount


class CheckingAccount(BankAccount):
    def __init__(self, account_number, account_holder, balance=0.0, overdraft_limit
        super().__init__(account_number, account_holder, balance)
        self.__overdraft_limit = overdraft_limit

    def withdraw(self, amount):
        if amount > 0:
            if self._BankAccount__balance + self.__overdraft_limit >= amount:
                self._BankAccount__balance -= amount
                print(f"Withdrew ${amount:.2f}. New balance is ${self._BankAccount_
            else:
                print("Insufficient funds and overdraft limit reached.")
        else:
            print("Withdrawal amount must be positive.")


# Testing the Banking System

# Create instances of SavingsAccount and CheckingAccount
savings = SavingsAccount("SA12345", "John Doe", 1000.0, 0.03)
checking = CheckingAccount("CA12345", "Jane Smith", 500.0, 300.0)

# Perform Transactions

# Display initial account info
print(savings.get_account_info())
print(checking.get_account_info())

# Deposit money into both accounts
savings.deposit(500)
checking.deposit(200)

# Withdraw money from both accounts
savings.withdraw(200)
checking.withdraw(700)

# Try to withdraw more than available in checking account (overdraft)
checking.withdraw(100)

# Add interest to the savings account
savings.add_interest()

# Display final balances
print(savings.get_balance())
print(checking.get_balance())

# Display final account info
```

```
print(savings.get_account_info())
print(checking.get_account_info())
```

```
Account Number: SA12345, Account Holder: John Doe, Balance: $1000.00
Account Number: CA12345, Account Holder: Jane Smith, Balance: $500.00
Deposited $500.00. New balance is $1500.00.
Deposited $200.00. New balance is $700.00.
Withdrew $200.00. New balance is $1300.00.
Withdrew $700.00. New balance is $0.00.
Withdrew $100.00. New balance is $-100.00.
Deposited $39.00. New balance is $1339.00.
Interest added: $39.00. New balance is $1339.00.
Account balance: $1339.00
Account balance: $-100.00
Account Number: SA12345, Account Holder: John Doe, Balance: $1339.00
Account Number: CA12345, Account Holder: Jane Smith, Balance: $-100.00
```

In [17]:
```
# Q.42 Write a python program that models different animals and their sounds. desig
# a) Define  the `animal` class with a method `make_sound()`.
class Animal:
    def make_sound(self):
        return "Some generic animal sound"
```

In [18]:
```
# Q.42 Write a python program that models different animals and their sounds. desig
# b) Create a subclasses `dog` and `cat` that override the `make_sound()`method.

class Animal:
    def make_sound(self):
        print("Generic animal sound")

class Dog(Animal):
    def make_sound(self):
        print("Woof!")

class Cat(Animal):
    def make_sound(self):
      print("Meow!")
```

In [20]:
```
 # Q.42 Write a python program that models different animals and their sounds. desi
# c) Implement the  sound  generation logic for each subclass.

class Animal:
    def make_sound(self):
        print("Generic animal sound")

class Dog(Animal):
    def make_sound(self):
        print("Woof!")

class Cat(Animal):
    def make_sound(self):
        print("Meow!")
```

In [21]:
```
# Q.42 Write a python program that models different animals and their sounds. desig
# d) Test the program  by creating instance of  `Dog` and `cat` and  calling the `m
```

```
class Animal:
    def make_sound(self):
        print("Generic animal sound")

class Dog(Animal):
    def make_sound(self):
        print("Woof!")

class Cat(Animal):
    def make_sound(self):
        print("Meow!")


# Create instances of Dog and Cat
dog = Dog()
cat = Cat()

# Call make_sound() method for each instance
dog.make_sound()  # Output: Woof!
cat.make_sound()  # Output: Meow!
```

```
Woof!
Meow!
```

In [23]:

```
# Q.43 Write a code for restaurent management system using Oops.
# a) Create a menu_item class that has attributes such as name,description,price an
class MenuItem:
    def __init__(self, name, description, price, category):
        self.name = name
        self.description = description
        self.price = price
        self.category = category

    def get_item_info(self):
        return f"{self.name} - {self.category}: ${self.price:.2f}\nDescription: {se
```

In [25]:

```
# Q.43 Write a code for restaurent management system using Oops.
# b) Implement methods to add a new menu_item,update menu item information, and rem
class Menu:
    def __init__(self):
        self.items = []

    def add_item(self, item):
        self.items.append(item)
        print(f"{item.name} has been added to the menu.")

    def update_item(self, name, new_name=None, new_description=None, new_price=None
        for item in self.items:
            if item.name == name:
                if new_name:
                    item.name = new_name
                if new_description:
                    item.description = new_description
                if new_price is not None:
                    item.price = new_price
                if new_category:
```

```
                    item.category = new_category
                print(f"{name} has been updated.")
                return
        print(f"Item named {name} not found in the menu.")

    def remove_item(self, name):
        for item in self.items:
            if item.name == name:
                self.items.remove(item)
                print(f"{name} has been removed from the menu.")
                return
        print(f"Item named {name} not found in the menu.")

    def display_menu(self):
        if not self.items:
            print("The menu is empty.")
        else:
            for item in self.items:
                print(item.get_item_info())
                print('-' * 30)

# Reusing the previously defined MenuItem class

class MenuItem:
    def __init__(self, name, description, price, category):
        self.name = name
        self.description = description
        self.price = price
        self.category = category

    def get_item_info(self):
        return f"{self.name} - {self.category}: ${self.price:.2f}\nDescription: {se
```

In [26]:

```
# Q.43 Write a code for restaurent management system using Oops.
# c) use encapsulation to hide the menu item's unique identification number.

class MenuItem:
    def __init__(self, name, description, price, category, item_id):
        self.name = name
        self.description = description
        self.price = price
        self.category = category
        self.__item_id = item_id  # Encapsulated attribute

    def get_item_id(self):
        return self.__item_id

    def __str__(self):
        return f"{self.name} ({self.category}): {self.description} - ${self.price:.

class Menu:
    def __init__(self):
        self.menu_items = []
        self.next_item_id = 1

    def add_item(self, name, description, price, category):
```

```
        item_id = self.next_item_id
        self.next_item_id += 1
        menu_item = MenuItem(name, description, price, category, item_id)
        self.menu_items.append(menu_item)
        return menu_item

    def update_item(self, item_id, new_description, new_price, new_category):
        for item in self.menu_items:
            if item.get_item_id() == item_id:
                item.description = new_description
                item.price = new_price
                item.category = new_category
                break

    def remove_item(self, item_id):
        for item in self.menu_items:
            if item.get_item_id() == item_id:
                self.menu_items.remove(item)
                break

    def get_menu_items(self):
        return self.menu_items
```

In [27]:

```
# Q.43 Write a code for restaurent management system using Oops.
# d) inherit  from the menu_item class to create a fooditem class and a Beaverageit

class MenuItem:
    def __init__(self, name, description, price, category, item_id):
        self.name = name
        self.description = description
        self.price = price
        self.category = category
        self.__item_id = item_id  # Encapsulated attribute

    def get_item_id(self):
        return self.__item_id

    def __str__(self):
        return f"{self.name} ({self.category}): {self.description} - ${self.price:.

class FoodItem(MenuItem):
    def __init__(self, name, description, price, category, item_id, is_vegetarian):
        super().__init__(name, description, price, category, item_id)
        self.is_vegetarian = is_vegetarian

class BeverageItem(MenuItem):
    def __init__(self, name, description, price, category, item_id, size):
        super().__init__(name, description, price, category, item_id)
        self.size = size
```

In [28]:

```
# Q.44 Write a code for hotel management system using Oops.
# a) Create a room class that has attributes such as room number,room type,rate, an

class Room:
    def __init__(self, room_number, room_type, rate):
```

```
        self.room_number = room_number
        self.room_type = room_type
        self.rate = rate
        self.__availability = True  # Private attribute

    def is_available(self):
        return self.__availability

    def set_availability(self, availability):
        self.__availability = availability
```

In [29]:

```
# Q.44 Write a code for hotel management system using Oops.
# b) Implement methods to book  a room,check in a guest,and check out a guest.

class Room:
    def __init__(self, room_number, room_type, rate):
        self.room_number = room_number
        self.room_type = room_type
        self.rate = rate
        self.__availability = True

    def is_available(self):
        return self.__availability

    def set_availability(self, availability):
        self.__availability = availability

class Hotel:
    def __init__(self):
        self.rooms = []

    def add_room(self, room):
        self.rooms.append(room)

    def book_room(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number and room.is_available():
                room.set_availability(False)
                print(f"Room {room_number} booked successfully.")
                return room
        else:
            print("Room not available.")
            return None

    def check_in(self, guest_name, room):
        if not room.is_available():
            print(f"Guest {guest_name} checked in to room {room.room_number}.")

    def check_out(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number and not room.is_availability():
                room.set_availability(True)
                print(f"Guest checked out from room {room_number}.")
                return
```

```
        else:
            print("Room not occupied.")
```

In [30]:
```
# Q.44 Write a code for hotel management system using Oops.
# c) use encapsulation to hide the room's unique identification number.

class Room:
    def __init__(self, room_number, room_type, rate):
        self.__room_number = room_number
        self.room_type = room_type
        self.rate = rate
        self.__availability = True

    def get_room_number(self):
        return self.__room_number

    def is_available(self):
        return self.__availability

    def set_availability(self, availability):
        self.__availability = availability

class Hotel:
    def __init__(self):
        self.rooms = []

    def add_room(self, room):
        self.rooms.append(room)

    def book_room(self, room_number):
        for room in self.rooms:
            if room.get_room_number() == room_number and room.is_available():
                room.set_availability(False)
                print(f"Room {room_number} booked successfully.")
                return room
        else:
            print("Room not available.")
            return None

    def check_in(self, guest_name, room):
        if not room.is_available():
            print(f"Guest {guest_name} checked in to room {room.get_room_number()}.

    def check_out(self, room_number):
        for room in self.rooms:
            if room.get_room_number() == room_number and not room.is_availability()
                room.set_availability(True)
                print(f"Guest checked out from room {room_number}.")
                return
        else:
            print("Room not occupied.")
```

In [31]:
```
# Q.44 Write a code for hotel management system using Oops.
# d) Inherit from the room class to create a suitroom class and a standardroom clas
class Room:
```

```
    def __init__(self, room_number, room_type, rate):
        self.__room_number = room_number
        self.room_type = room_type
        self.rate = rate
        self.__availability = True

    def get_room_number(self):
        return self.__room_number

    def is_available(self):
        return self.__availability

    def set_availability(self, availability):
        self.__availability = availability

class SuiteRoom(Room):
    def __init__(self, room_number, rate, capacity):
        super().__init__(room_number, "Suite", rate)
        self.capacity = capacity

class StandardRoom(Room):
    def __init__(self, room_number, rate, bed_type):
        super().__init__(room_number, "Standard", rate)
        self.bed_type = bed_type
```

In [32]:

```
# Q.45 Write a Code for event management system using Oops.
# a) Create an event class that has attributes such as name,date,time,location and

class Event:
    def __init__(self, name, date, time, location):
        self.name = name
        self.date = date
        self.time = time
        self.location = location
        self.__attendees = []

    def add_attendee(self, attendee):
        self.__attendees.append(attendee)

    def get_attendees(self):
        return self.__attendees
```

In [33]:

```
# Q.45 Write a Code for event management system using Oops.
# b) Implement methods to create a new event,add or remove attendees, and get the t

class Event:
    def __init__(self, name, date, time, location):
        self.name = name
        self.date = date
        self.time = time
        self.location = location
        self.__attendees = []

    def add_attendee(self, attendee):
        self.__attendees.append(attendee)
```

```
    def remove_attendee(self, attendee):
        if attendee in self.__attendees:
            self.__attendees.remove(attendee)

    def get_attendees(self):
        return self.__attendees

    def get_total_attendees(self):
        return len(self.__attendees)
```

In [34]:
```
# Q.45 Write a Code for event management system using Oops.
# c) use encapsulation to hide the events unique identification number.

class Event:
    def __init__(self, name, date, time, location, event_id):
        self.name = name
        self.date = date
        self.time = time
        self.location = location
        self.__attendees = []
        self.__event_id = event_id

    def get_event_id(self):
        return self.__event_id

    def add_attendee(self, attendee):
        self.__attendees.append(attendee)

    def remove_attendee(self, attendee):
        if attendee in self.__attendees:
            self.__attendees.remove(attendee)

    def get_attendees(self):
        return self.__attendees

    def get_total_attendees(self):
        return len(self.__attendees)
```

In [35]:
```
# Q.45 Write a Code for event management system using Oops.
# d) Inherit from the event class to create a privateevent class and a public event

class Event:
    def __init__(self, name, date, time, location, event_id):
        self.name = name
        self.date = date
        self.time = time
        self.location = location
        self.__attendees = []
        self.__event_id = event_id

    def get_event_id(self):
        return self.__event_id

    def add_attendee(self, attendee):
```

```
        self.__attendees.append(attendee)

    def remove_attendee(self, attendee):
        if attendee in self.__attendees:
            self.__attendees.remove(attendee)

    def get_attendees(self):
        return self.__attendees

    def get_total_attendees(self):
        return len(self.__attendees)

class PrivateEvent(Event):
    def __init__(self, name, date, time, location, event_id, invitees):
        super().__init__(name, date, time, location, event_id)
        self.invitees = invitees

class PublicEvent(Event):
    def __init__(self, name, date, time, location, event_id, registration_fee):
        super().__init__(name, date, time, location, event_id)
        self.registration_fee = registration_fee
```

In [36]:

```
# Q.46 Write a code for airline reservation system using Oops.
# a) Create a flight class that has attributes such as flight number,departure and

class Flight:
    def __init__(self, flight_number, departure_airport, arrival_airport, departure
        self.flight_number = flight_number
        self.departure_airport = departure_airport
        self.arrival_airport = arrival_airport
        self.departure_time = departure_time
        self.arrival_time = arrival_time

        self.__available_seats = 150  # Assuming 150 seats as default

    def get_available_seats(self):
        return self.__available_seats

    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            return True
        else:
            return False
```

In [37]:

```
# Q.46 Write a code for airline reservation system using Oops.
# b) Implement methods to book a seat,cancel a reservation , and get the remaining

class Flight:
    def __init__(self, flight_number, departure_airport, arrival_airport, departure
        self.flight_number = flight_number
        self.departure_airport = departure_airport
        self.arrival_airport = arrival_airport
        self.departure_time = departure_time
        self.arrival_time = arrival_time
```

```
        self.__available_seats = 150  # Assuming 150 seats as default

    def get_available_seats(self):
        return self.__available_seats

    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            return True
        else:
            return False

    def cancel_seat(self):
        if self.__available_seats < 150:
            self.__available_seats += 1
            return True
        else:
            return False
```

In [38]:

```
# Q.46 Write a code for airline reservation system using Oops.
# c) Use encapsulation to hide the flight's uique  identification number.
class Flight:
    def __init__(self, flight_number, departure_airport, arrival_airport, departure
        self.__flight_number = flight_number
        self.departure_airport = departure_airport
        self.arrival_airport = arrival_airport
        self.departure_time = departure_time
        self.arrival_time = arrival_time

        self.__available_seats = 150  # Assuming 150 seats as default

    def get_flight_number(self):
        return self.__flight_number

    def get_available_seats(self):
        return self.__available_seats

    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            return True
        else:
            return False

    def cancel_seat(self):
        if self.__available_seats < 150:
            self.__available_seats += 1
            return True
        else:
            return False
```

In [39]:

```
# Q.46 Write a code for airline reservation system using Oops.
# d) Inherit from the flight class to create a domesticflight class and an internat
```

```
class Flight:
    def __init__(self, flight_number, departure_airport, arrival_airport, departure
        self.__flight_number = flight_number
        self.departure_airport = departure_airport
        self.arrival_airport = arrival_airport
        self.departure_time = departure_time
        self.arrival_time = arrival_time

        self.__available_seats = 150  # Assuming 150 seats as default

    def get_flight_number(self):
        return self.__flight_number

    def get_available_seats(self):
        return self.__available_seats

    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            return True
        else:
            return False

    def cancel_seat(self):
        if self.__available_seats < 150:
            self.__available_seats += 1
            return True
        else:
            return False

class DomesticFlight(Flight):
    def __init__(self, flight_number, departure_airport, arrival_airport, departure
        super().__init__(flight_number, departure_airport, arrival_airport, departu
        self.meal_type = meal_type

class InternationalFlight(Flight):
    def __init__(self, flight_number, departure_airport, arrival_airport, departure
        super().__init__(flight_number, departure_airport, arrival_airport, departu
        self.visa_required = visa_required
```

In [41]:

```
# Q.47 Define a python module  named constants.pycontaining constants like pi and t
# constants.py

# Mathematical constant pi
PI = 3.141592653589793

# Speed of light in vacuum (in meters per second)
SPEED_OF_LIGHT = 299792458

# Gravitational constant (in m^3 kg^-1 s^-2)
GRAVITATIONAL_CONSTANT = 6.67430e-11

# Planck constant (in joule seconds)
PLANCK_CONSTANT = 6.62607015e-34

# Avogadro's number (in mol^-1)
```

```
AVOGADRO_NUMBER = 6.02214076e23
# example_usage.py

from constants import PI, SPEED_OF_LIGHT

print(f"The value of pi is: {PI}")
print(f"The speed of light is: {SPEED_OF_LIGHT} m/s")
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[41], line 20
     17 AVOGADRO_NUMBER = 6.02214076e23
     18 # example_usage.py
---> 20 from constants import PI, SPEED_OF_LIGHT
     22 print(f"The value of pi is: {PI}")
     23 print(f"The speed of light is: {SPEED_OF_LIGHT} m/s")

ModuleNotFoundError: No module named 'constants'
```

In [42]:

```
# Q.48 Write a python module named calculator.py containing functions for additions
import calculator

result = calculator.add(5, 3)
print(result)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[42], line 2
      1 # Q.48 Write a python module named calculator.py containing functions for
additions,subtractions,multiplication and division.
----> 2 import calculator
      4 result = calculator.add(5, 3)
      5 print(result)

ModuleNotFoundError: No module named 'calculator'
```

In [45]:

```
# Q.49 Implement a python package structure for a project named ecommerce,containin

#ecommerce/
# │
# ├── __init__.py
# ├── product_management/
# │   ├── __init__.py
# │   ├── product.py
# │   ├── inventory.py
# │   └── category.py
# │
# └── order_processing/
#     ├── __init__.py
#     ├── order.py
#     ├── payment.py
#     └── shipping.py
# product.py

class Product:
    def __init__(self, name, price, category):
```

```
        self.name = name
        self.price = price
        self.category = category

    def get_product_info(self):
        return f"{self.name} - {self.category}: ${self.price:.2f}"
# inventory.py

class Inventory:
    def __init__(self):
        self.items = {}

    def add_product(self, product, quantity):
        self.items[product] = self.items.get(product, 0) + quantity

    def get_stock(self, product):
        return self.items.get(product, 0)
# category.py

class Category:
    def __init__(self, name):
        self.name = name
# order.py

class Order:
    def __init__(self, order_id, customer_name):
        self.order_id = order_id
        self.customer_name = customer_name
        self.items = []

    def add_item(self, product, quantity):
        self.items.append((product, quantity))

    def get_order_summary(self):
        summary = f"Order ID: {self.order_id}\nCustomer: {self.customer_name}\nItem
        for product, quantity in self.items:
            summary += f"- {product.get_product_info()} x {quantity}\n"
        return summary
# payment.py

class Payment:
    def __init__(self, amount, method):
        self.amount = amount
        self.method = method

    def process_payment(self):
        return f"Processed payment of ${self.amount:.2f} using {self.method}."
# shipping.py

class Shipping:
    def __init__(self, address):
        self.address = address

    def ship_order(self, order):
        return f"Order {order.order_id} will be shipped to {self.address}."
```

In [46]:
```
# Q.50 Implement a python module named string_utils.py containing functions for str

def reverse_string(text):
  """Reverses the order of characters in a string.

  Args:
      text: The string to be reversed.

  Returns:
      A new string with the characters in reversed order.
  """
  return text[::-1]

def capitalize_first(text):
  """Capitalizes the first letter of a string and returns the new string.

  Args:
      text: The string to be capitalized.

  Returns:
      A new string with the first letter capitalized.
  """
  return text.capitalize()

def is_palindrome(text):
  """Checks if a string is a palindrome (reads the same backward as forward).

  Args:
      text: The string to be checked.

  Returns:
      True if the string is a palindrome, False otherwise.
  """
  text = text.lower().replace(" ", "")  # Case-insensitive and remove spaces
  return text == text[::-1]

def slugify(text):
  """Converts a string to a slug (lowercase, spaces replaced with hyphens).

  Args:
      text: The string to be converted to a slug.

  Returns:
      A new string in lowercase with spaces replaced by hyphens.
  """
  return text.lower().replace(" ", "-")
```

In [47]:
```
# Q.51 Write a python module  named file_operations.py with functions for reading,w

def read_file(filepath):
  """
  Reads the content of a file and returns it as a string.

  Args:
      filepath (str): The path to the file to be read.
```

```
    Returns:
        str: The content of the file, or None if an error occurs.
    """
    try:
      with open(filepath, 'r') as file:
        return file.read()
    except FileNotFoundError:
      print(f"Error: File not found: {filepath}")
      return None

def write_file(filepath, content):
    """
    Writes the provided content to a file.

    Args:
        filepath (str): The path to the file to be written to.
        content (str): The content to be written to the file.
    """
    try:
      with open(filepath, 'w') as file:
        file.write(content)
      print(f"Successfully wrote to file: {filepath}")
    except (IOError, OSError) as error:
      print(f"Error writing to file: {filepath} - {error}")

def append_to_file(filepath, content):
    """
    Appends the provided content to a file.

    Args:
        filepath (str): The path to the file to be appended to.
        content (str): The content to be appended to the file.
    """
    try:
      with open(filepath, 'a') as file:
        file.write(content)
      print(f"Successfully appended to file: {filepath}")
    except (IOError, OSError) as error:
      print(f"Error appending to file: {filepath} - {error}")
```

In [48]:

```
# Q.52 Write a python program to create a text file named 'employees.txt' and write

def handle_file_operation(filepath, content, mode='w'):
    """
    Reads, writes, or appends content to a file based on the mode.

    Args:
        filepath (str): The path to the file.
        content (str): The content to write or append.
        mode (str, optional): The file open mode ('r' for read, 'w' for write, 'a' fo

    Returns:
        str: The content of the file (for read mode) or None on error.
        Exception: The specific exception encountered during the operation (if any).
    """
```

```
  try:
    with open(filepath, mode) as file:
      if mode == 'r':
        return file.read()
      else:
        file.write(content)
        return None  # Or return success message
  except FileNotFoundError:
    return f"Error: File not found: {filepath}"
  except (PermissionError, UnicodeDecodeError) as error:
    return error  # Return specific exception

# Usage (Read)
file_content = handle_file_operation("myfile.txt")
if isinstance(file_content, str):
  print(file_content)
else:
  print(file_content)  # Handle specific error returned

# Usage (Write)
write_result = handle_file_operation("newfile.txt", "This is new content.")
if not isinstance(write_result, Exception):
  print("Successfully wrote to file.")
else:
  print(write_result)  # Handle specific error returned
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[48], line 29
     26     return error  # Return specific exception
     28 # Usage (Read)
---> 29 file_content = handle_file_operation("myfile.txt")
     30 if isinstance(file_content, str):
     31   print(file_content)

TypeError: handle_file_operation() missing 1 required positional argument: 'conten
t'
```

In [49]:

```
# Q.53 Develop a python  script that opens  an existing text file named 'inventory.
def read_inventory_file(file_path):
  """Reads the content of an inventory file line by line.

  Args:
      file_path (str): The path to the inventory file.
  """

  try:
      with open(file_path, 'r') as file:
          for line in file:
              print(line.strip())
  except FileNotFoundError:
      print(f"Error: Inventory file '{file_path}' not found.")

# Example usage:
file_path = 'inventory.txt'
read_inventory_file(file_path)
```

```
Error: Inventory file 'inventory.txt' not found.
```

In [52]:
```
# Q.54 Create a python script that reads a text file named 'expenses.txt' and calcu

def calculate_total_expenses(file_path):
  """Calculates the total expenses from a text file.

  Args:
      file_path (str): The path to the expenses file.

  Returns:
      float: The total amount spent.
  """

  total_expenses = 0.0
  try:
    with open(file_path, 'r') as file:
      for line in file:
        amount = float(line.strip())
        total_expenses += amount
  except FileNotFoundError:
    print(f"Error: Expenses file '{file_path}' not found.")
  except ValueError:
    print("Error: Invalid data format in expenses file.")
  return total_expenses

# Example usage:
file_path = 'expenses.txt'
total_amount = calculate_total_expenses(file_path)
print("Total expenses:", total_amount)
```

```
Error: Expenses file 'expenses.txt' not found.
Total expenses: 0.0
```

In [51]:
```
# Q.55 Create a python program that reads a text file named  'paragraph.txt' and  c

import collections

def count_word_occurrences(file_path):
  """Counts the occurrences of each word in a text file.

  Args:
      file_path (str): The path to the text file.
  """

  word_counts = collections.defaultdict(int)

  try:
    with open(file_path, 'r') as file:
      for line in file:
        words = line.split()
        for word in words:
          word_counts[word.lower()] += 1
  except FileNotFoundError:
    print(f"Error: File not found: {file_path}")
  else:
```

```
    for word, count in sorted(word_counts.items()):
      print(f"{word}: {count}")

# Example usage:
file_path = 'paragraph.txt'
count_word_occurrences(file_path)
```

```
Error: File not found: paragraph.txt
```

In [53]:

```
#. Q.56 What do you mean by measure of central tendency and measure of dispersion.h

# In statistics, measures of central tendency and measures of dispersion are two im

# Measure of Central Tendency:

# A measure of central tendency represents the "center" or "typical" value of a dat

# Mean: The average of all values in the dataset, calculated by adding all values a
# Median: The middle value when the data is arranged in ascending or descending ord
# Mode: The most frequent value in the dataset. A dataset can have multiple modes (
# Measure of Dispersion:

# A measure of dispersion describes how spread out the data points are in a dataset

# Range: The difference between the highest and lowest values in the dataset.
# Variance: The average of the squared deviations from the mean. It represents how
# Standard Deviation: The square root of the variance. It's expressed in the same u
# These measures are used together to get a complete picture of a dataset. For exam

# Calculating Measures:

# There are various ways to calculate these measures, depending on the size and typ

# Python
import statistics

data = [5, 8, 12, 7, 3, 10]

# Measures of Central Tendency
mean = statistics.mean(data)
median = statistics.median(data)
mode = statistics.mode(data)

# Measures of Dispersion
range_value = max(data) - min(data)
variance = statistics.variance(data)
standard_deviation = statistics.stdev(data)

print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Mode: {mode}")
print(f"Range: {range_value}")
print(f"Variance: {variance}")
print(f"Standard Deviation: {standard_deviation}")
```

```
Mean: 7.5
Median: 7.5
Mode: 5
Range: 9
Variance: 10.7
Standard Deviation: 3.271085446759225
```

In [54]:
```
# Q.63 In a partially destroyed laboratory record  of an analysis of correlation da
# a) what are the mean value of x and y.

# Understanding the Problem
# We have two regression equations:
# 40x-18y=214
# 8x-10y = -66
# We also know that the variance of x (σ²x) is 9.

# Our goal is to find the mean values of x (x̄) and y (ȳ).

# Solution
# Key point: The point of intersection of the two regression lines is the point (x̄,

# Steps:

# Convert the given equations into the standard form of regression lines:
y = bxy * x + a
x = byx * y + a
#Find the values of bxy and byx from the given equations.
# Use the formula for the intersection point (x̄, ȳ) of the two regression lines:
x̄ = (a1b2 - a2b1) / (b1b2 - 1)
ȳ = (b1a2 - b2a1) / (b1b2 - 1)
# Where:

# a1 and b1 are the constants in the regression equation for y on x.
# a2 and b2 are the constants in the regression equation for x on y.
# Calculations:

# Rewrite the equations in standard form:
y = (4/5)x + 6.6
x = (9/10)y + 5.35
# Identify the values:
bxy = 4/5
byx = 9/10
a1 = 6.6
a2 = 5.35
# Calculate the means:
x̄ = ((6.6 * 9/10) - (5.35 * 4/5)) / ((4/5 * 9/10) - 1)
ȳ = ((4/5 * 5.35) - (9/10 * 6.6)) / ((4/5 * 9/10) - 1)
```

```
  Cell In[54], line 31
    y = (4/5)x + 6.6
              ^
SyntaxError: invalid syntax
```

In [55]:
```
# Q.63 In a partially destroyed laboratory record  of an analysis of correlation da
# b) the coefficient of correlation between x and y.
```

```
# Understanding the Problem
# We have the regression equations:

8x - 10y = -66
40x - 18y = 214
# We also know the variance of x (σ²x) = 9.

# Our goal is to find the coefficient of correlation (r).

# Solution
# Key point: The coefficient of correlation (r) is the square root of the product o

# Steps:

# Convert the given equations into the standard form of regression lines:
y = bxy * x + a
x = byx * y + a
# Find the values of bxy and byx from the given equations.
# Calculate the coefficient of correlation (r) using the formula:
# r = sqrt(bxy * byx)
# Calculations:
# We already found bxy and byx in the previous part:

bxy = 4/5
byx = 9/10
# Now, calculate r:

r = sqrt((4/5) * (9/10))
r = sqrt(36/50)
r = 6/5
# Therefore, the coefficient of correlation (r) between x and y is 6/5.
```

```
  Cell In[55], line 7
    8x - 10y = -66
    ^
SyntaxError: invalid decimal literal
```

In [56]:

```
# Q.71 A random sample of size 25 from a population gives the sample standard deriv
# To test the hypothesis that the population standard deviation is 10.5 based on a

# Hypotheses
# Null Hypothesis (H₀): The population standard deviation is 10.5.
# Alternative Hypothesis (H₁): The population standard deviation is not 10.5.
# Chi-Square Test for Variance
# The test statistic for the chi-square test for variance is calculated using the f

# ⍰
# 2
# =
# (
# ⍰
# −
# 1
# )
# ·
# ⍰
```

```
# 2
# ⍰
# 2
# χ
# 2
#  =
# σ
# 2

# (n−1)⋅s
# 2

#



# where:

# ⍰
# n = sample size
# ⍰
# s = sample standard deviation
# ⍰
# σ = hypothesized population standard deviation
# Given Data
# Sample size,
# ⍰
# =
# 25
# n=25
# Sample standard deviation,
# ⍰
# =
# 9.0
# s=9.0
# Hypothesized population standard deviation,
# ⍰
# =
# 10.5
# σ=10.5
# Steps to Calculate the Chi-Square Statistic
# Calculate the sample variance:

# ⍰
# 2
# =
# 9.
# 0
# 2
# =
# 81
# s
# 2
#  =9.0
# 2
#  =81
```

```
# Calculate the chi-square test statistic:

# ?
# 2
# =
# (
# ?
# −
# 1
# )
# ·
# ?
# 2
# ?
# 2
# χ
# 2
#  =
# σ
# 2

# (n−1)·s
# 2

#

# ?
# 2
# =
# (
# 25
# −
# 1
# )
# ·
# 81
# 10.
# 5
# 2
# χ
# 2
#  =
# 10.5
# 2

# (25−1)·81
#

# ?
# 2
# =
# 24
# ·
# 81
# 110.25
# χ
```

```
# 2
#  =
# 110.25
# 24·81
#

# ⍰
# 2
# =
# 1944
# 110.25
# χ
# 2
#  =
# 110.25
# 1944
#

# ⍰
# 2
# ≈
# 17.63
# χ
# 2
#  ≈17.63

# Determine the degrees of freedom:

# The degrees of freedom for this test are
# ⍰
# −
# 1
# n−1:
# Degrees of freedom
# =
# 25
# −
# 1
# =
# 24
# Degrees of freedom=25−1=24

# Find the critical value and p-value:

# To make a decision, compare the calculated chi-square statistic to the critical v
# ⍰
# =
# 0.05
# α=0.05 (typically used for a 95% confidence level). You can also calculate the p-

# Critical value: For a two-tailed test at
# ⍰
# =
# 0.05
# α=0.05 and 24 degrees of freedom, use chi-square tables or statistical software t
# Lower critical value:
```

```
# ⍰
# 0.025
# ,
# 24
# 2
# χ
# 0.025,24
# 2
#

# Upper critical value:
# ⍰
# 0.975
# ,
# 24
# 2
# χ
# 0.975,24
# 2
#

# For 24 degrees of freedom:

# The lower critical value is approximately 12.40.
# The upper critical value is approximately 36.42.
# Decision Rule
# Reject H₀ if the chi-square statistic is less than the lower critical value or gr
# Fail to reject H₀ if the chi-square statistic falls between the critical values.
# Conclusion
# In this case, the calculated chi-square statistic (17.63) falls between the criti

# Fail to reject the null hypothesis.
```

In [57]:

```
# Q.73 To study the performance of three detergents and three different water tempr
# WATER TEMPLATE           DETERGENT A            DETERGENT B           DETERGENT C
#   COLD WATER                  57                     55                     67
#   WORM WATER                  49                     52                     68
#   HOT WATER                   54                     46                     58

# Understanding the Problem
# We have a dataset showing the whiteness readings for three detergents under three

# Appropriate Statistical Test
# A Two-Way ANOVA (Analysis of Variance) is suitable for this type of data. It allo

# Steps Involved
# Organize the Data:

# Create a table or data frame to clearly represent the data.
# Calculate the total and average whiteness for each detergent, water temperature,
# Calculate Sum of Squares:

# Calculate the total sum of squares (SST), sum of squares due to detergents (SSD),
# Calculate Mean Square Values:

# Divide the sum of squares by their respective degrees of freedom to obtain mean s
```

```
# Calculate F-Statistic:

# Calculate F-statistics for detergents and water temperature.
# Determine Critical Values:

# Find the critical F-values from the F-distribution table based on the degrees of
# Make Decisions:

# Compare the calculated F-statistics with the critical F-values.
# If the calculated F-value is greater than the critical F-value, reject the null h
```

In [62]:

```
# Q.74 How would you create a basic flask route that displays 'hello,world'! on the
 #pip install flask
from flask import Flask

# Create a new Flask application instance
app = Flask(__name__)

# Define a route for the homepage
@app.route('/')
def hello_world():
    return 'Hello, World!'

# Run the application
if __name__ == '__main__':
    app.run(debug=True)
#python app.py
Hello, World
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[62], line 3
      1 # Q.74 How would you create a basic flask route that displays 'hello,worl
d'! on the homepage.
      2  #pip install flask
----> 3 from flask import Flask
      5 # Create a new Flask application instance
      6 app = Flask(__name__)

ModuleNotFoundError: No module named 'flask'
```

In [63]:

```
# Q.75 Explain how to setup a flask application to handle from submissions using PO

from flask import Flask, render_template, request

app = Flask(__name__)

<!DOCTYPE html>
<html>
<head>
    <title>Form Example</title>
</head>
<body>
    <form method="POST" action="/">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name"><br><br>
```

```
        <input type="submit" value="Submit">
    </form>
</body>
</html>

from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        name = request.form['name']

        # Do something with the name, like saving it to a database
        return f"Hello, {name}!"
    else:
        return render_template('form.html')

if __name__ == '__main__':
    app.run(debug=True)
```

```
  Cell In[63], line 7
    <!DOCTYPE html>
    ^
SyntaxError: invalid syntax
```

In [64]:

```
# Q.76 Write a flask route that accept a parameter in the URL and display its on pa

from flask import Flask

app = Flask(__name__)

@app.route('/hello/<name>')
def hello(name):
    return f"Hello, {name}!"

if __name__ == '__main__':
    app.run(debug=True)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[64], line 3
      1 # Q.76 Write a flask route that accept a parameter in the URL and display
its on page.
----> 3 from flask import Flask
      5 app = Flask(__name__)
      7 @app.route('/hello/<name>')
      8 def hello(name):

ModuleNotFoundError: No module named 'flask'
```

In [86]:

```
# Q.77 How can you implement user authentication in a flask application?

# Understanding the Basics
```

```
# User authentication is a crucial aspect of web applications. It involves verifyin

# Key Components:

# User Model: Represents user data (username, email, password, etc.).
# Password Hashing: Securely stores user passwords.
# Session Management: Tracks user login status.
# Login and Logout Functionality: Provides routes for user authentication.
# Protected Routes: Restricts access to certain routes based on user authenticatio

# from flask import Flask, render_template, redirect, url_for, flash
# from flask_sqlalchemy import SQLAlchemy
# from flask_login import LoginManager, UserMixin, login_user,
#  login_required, logout_user, current_user


# App and database configuration
app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
db = SQLAlchemy(app)

# Login manager
login_manager= LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'

# User model
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True,
 nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(128), nullable=False)

    def __repr__(self):
        return '<User %r>' % self.username


# Load user from database
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

# Login and registration routes
# ...

# Protected route
@app.route('/dashboard')
@login_required
def dashboard():
    return 'Dashboard'
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[86], line 22
      1 # Q.77 How can you implement user authentication in a flask application?
      2
      3 # Understanding the Basics
   (...)
     20
     21 # App and database configuration
---> 22 app = Flask(__name__)
     23 app.config['SECRET_KEY'] = 'your_secret_key'
     24 app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'

NameError: name 'Flask' is not defined
```

In [66]:

```
# Q.78 Describe the process of connecting a flask app to SQlite database using SQLa

pip install Flask Flask-SQLAlchemy

from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///mydatabase.db'

db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)

    email = db.Column(db.String(120), unique=True, nullable=False)

    def __repr__(self):
        return '<User %r>' % self.username

        with app.app_context():
    db.create_all()

    @app.route('/')
def index():
    users = User.query.all()
    return render_template('index.html', users=users)
```

```
  Cell In[66], line 3
    pip install Flask Flask-SQLAlchemy
        ^
SyntaxError: invalid syntax
```

In [68]:

```
# Q.79 would you create a RESTful API endpoint in flask that returns JSON data?

pip install flask
from flask import Flask, jsonify

# Create a new Flask application instance
app = Flask(__name__)
```

```
# Define a route that returns JSON data
@app.route('/api/data', methods=['GET'])
def get_data():
    data = {
        'name': 'John Doe',
        'age': 30,
        'email': 'john.doe@example.com'
    }
    return jsonify(data)

# Run the application
if __name__ == '__main__':
    app.run(debug=True)
python app.py
{
    "name": "John Doe",
    "age": 30,
    "email": "john.doe@example.com"
}
```

```
  Cell In[68], line 3
    pip install flask
        ^
SyntaxError: invalid syntax
```

In [69]:

```
# Q.80 Explain how to use  flask-WTF to create and validate  forms in a flask appli

pip install Flask-WTF

from flask import Flask, render_template, request, redirect, url_for
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators
 import DataRequired

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
  # Replace with a strong secret key

# ... other configurations
class MyForm(FlaskForm):
    name = StringField('Name', validators=[DataRequired()])
    submit = SubmitField('Submit')

    class MyForm(FlaskForm):
    name = StringField('Name', validators=[DataRequired()])
    submit = SubmitField('Submit')
    <!DOCTYPE html>
<html>
<head>
    <title>My Form</title>
</head>
<body>
    <form method="POST">
        {{ form.hidden_tag() }}
```

```
        {{ form.name.label }} {{ form.name(class='form-control') }}
        {{ form.submit(class='btn btn-primary') }}
    </form>
</body>
</html>
```

```
  Cell In[69], line 3
    pip install Flask-WTF
        ^
SyntaxError: invalid syntax
```

In [70]:

```
# Q.82 Describe the steps to  create a flask blueprint and why you might use one.

from flask import Blueprint
auth_blueprint = Blueprint('auth', __name__, template_folder='templates', static_fo
@auth_blueprint.route('/login')
def login():
    # Login logic
    return 'Login page'

    from flask import Flask

app = Flask(__name__)
app.register_blueprint(auth_blueprint, url_prefix='/auth')
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[70], line 3
      1 # Q.82 Describe the steps to  create a flask blueprint and why you might u
se one.
----> 3 from flask import Blueprint
      4 auth_blueprint = Blueprint('auth', __name__, template_folder='templates',
static_folder='static')
      5 @auth_blueprint.route('/login')
      6 def login():
      7     # Login logic

ModuleNotFoundError: No module named 'flask'
```

In [84]:

```
# Q.84 Make a fully functional web application  using flask,mongodb,signup,signinpa

pip install Flask pymongo flask-bcrypt
import os

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'your_secret_key'
    MONGODB_URI = 'mongodb://localhost:27017/your_database_name'
from flask import Flask
from flask_pymongo import PyMongo
from werkzeug.security import generate_password_hash, check_password_hash

app = Flask(__name__)
app.config.from_object('config')
mongo = PyMongo(app)

class User(object):
```

```
    def __init__(self, username, email, password):
        self.username = username
        self.email = email
        self.password = generate_password_hash(password)


    def is_authenticated(self):
        return True

    def is_active(self):
        return True

    def is_anonymous(self):
        return False

    def get_id(self):
        return
self.username

    def check_password(self, password):
        return check_password_hash(self.password, password)
        from flask import Flask, render_template, request, redirect, url_for, flash
from flask_pymongo import PyMongo
from werkzeug.security import generate_password_hash, check_password_hash

from models import User, mongo

app = Flask(__name__)
app.config.from_object('config')
mongo = PyMongo(app)

# ... your routes here ...

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']

        # Check if user exists

        user = mongo.db.users.find_one({'username': username})
        if user:
            flash('Username already exists')
            return redirect(url_for('signup'))
        # Create new user
        new_user = User(username, email, password)
        mongo.db.users.insert_one(new_user.__dict__)
        flash('User created successfully')
        return redirect(url_for('login'))
    return render_template('signup.html')

password = request.form['password']
        user
= mongo.db.users.find_one({'username': username})
```

```
        if user and user['password'] == password:
            # Successful login
            session['user'] = username
            return redirect(url_for('home'))
        else:
            flash('Invalid username or password')
        return redirect(url_for('login'))
return render_template('login.html')

@app.route('/home')
def home():
    if 'user' in session:
        return 'Hello, Geeks!'
    else:
        return redirect(url_for('login'))
```

```
  Cell In[84], line 3
    pip install Flask pymongo flask-bcrypt
        ^
SyntaxError: invalid syntax
```

In [73]:

```
# Q.86 Create a database name travel_planner in my sql,and create a table name book

-- Create the database
CREATE DATABASE travel_planner;

-- Use the database
USE travel_planner;

-- Create the table 'bookings'
CREATE TABLE bookings (
    user_id INT,
    flight_id INT,
    hotel_id INT,
    activity_id INT,
    booking_date DATE
);

-- Insert some dummy data into the table
INSERT INTO bookings (user_id, flight_id, hotel_id, activity_id, booking_date) VALU
(1, 101, 201, 301, '2024-08-01'),
(2, 102, 202, 302, '2024-08-02'),
(3, 103, 203, 303, '2024-08-03'),
(4, 104, 204, 304, '2024-08-04'),
(5, 105, 205, 305, '2024-08-05');

import pandas as pd
import mysql.connector

# Connect to the MySQL database
conn = mysql.connector.connect(
    host="localhost",
    user="your_username",
    password="your_password",
    database="travel_planner"
```

```
)

# Query the table
query = "SELECT * FROM bookings;"

# Read the table into a Pandas DataFrame
df = pd.read_sql(query, conn)

# Close the connection
conn.close()

# Display the DataFrame
print(df)
```

```
   user_id  flight_id  hotel_id  activity_id booking_date
0        1        101       201          301   2024-08-01
1        2        102       202          302   2024-08-02
2        3        103       203          303   2024-08-03
3        4        104       204          304   2024-08-04
4        5        105       205          305   2024-08-05
```

```
  Cell In[73], line 50
    0        1        101       201          301   2024-08-01
                                                             ^
SyntaxError: leading zeros in decimal integer literals are not permitted; use an 0
o prefix for octal integers
```

MACHINE LEARNING

In [74]:
```
# Q.129 Do the EDA on the given dataset : lung cancer,and extract some useful infor
# Dataset description:- lung cancer is one of the most prevalent and deadly form of

# Disclaimer: Without the actual dataset, I can only provide a general outline of t

# Potential Dataset Structure
# Assuming a typical lung cancer dataset, it might contain the following features:

# Demographic information: Age, gender, race, ethnicity, occupation, smoking histor
# Medical history: Family history of cancer, other diseases, medication history, et
# Symptoms: Cough, shortness of breath, chest pain, weight loss, etc.
# Diagnostic tests: X-ray, CT scan, MRI, biopsy results, etc.
# Treatment details: Type of treatment, duration, response, side effects, etc.
# Outcome: Survival rate, stage of cancer, recurrence, etc.
# Exploratory Data Analysis (EDA) Steps

import pandas as pd

# Load the dataset
df = pd.read_csv('lung_cancer_dataset.csv')
print(df.shape)
print(df.columns)
print(df.head())
print(df.dtypes)
print(df.isnull().sum())
print(df.describe())
print(df.describe(include=['object']))
```

```
import matplotlib.pyplot as plt
df.hist(figsize=(12, 10))
plt.show()

import seaborn as sns
sns.boxplot(x=df['Age'])
plt.show()

sns.countplot(x='Gender', data=df)
plt.show()

correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()

sns.pairplot(df)
plt.show()

df_grouped = df.groupby('Stage').mean()
print(df_grouped)


from lifelines import KaplanMeierFitter

kmf = KaplanMeierFitter()
kmf.fit(durations=df['Survival_Time'], event_observed=df['Event_Observed'])
kmf.plot_survival_function()
plt.show()

from sklearn.ensemble import RandomForestClassifier

X = df.drop(['Outcome'], axis=1)  # Features
y = df['Outcome']  # Target

model = RandomForestClassifier()
model.fit(X, y)
importance = model.feature_importances_

# Visualize feature importance
feature_importances = pd.DataFrame(importance, index=X.columns, columns=['Importanc
feature_importances.sort_values(by='Importance', ascending=False).plot(kind='bar')
plt.show()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[74], line 20
     17 import pandas as pd
     19 # Load the dataset
---> 20 df = pd.read_csv('lung_cancer_dataset.csv')
     21 print(df.shape)
     22 print(df.columns)

File /opt/conda/lib/python3.10/site-packages/pandas/util/_decorators.py:211, in de
precate_kwarg.<locals>._deprecate_kwarg.<locals>.wrapper(*args, **kwargs)
    209     else:
    210         kwargs[new_arg_name] = new_arg_value
--> 211 return func(*args, **kwargs)

File /opt/conda/lib/python3.10/site-packages/pandas/util/_decorators.py:331, in de
precate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File /opt/conda/lib/python3.10/site-packages/pandas/io/parsers/readers.py:950, in
read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, sq
ueeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_val
ues, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na
_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_
col, date_parser, dayfirst, cache_dates, iterator, chunksize, compression, thousan
ds, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment,
encoding, encoding_errors, dialect, error_bad_lines, warn_bad_lines, on_bad_lines,
delim_whitespace, low_memory, memory_map, float_precision, storage_options)
    935 kwds_defaults = _refine_defaults_read(
    936     dialect,
    937     delimiter,
   (...)
    946     defaults={"delimiter": ","},
    947 )
    948 kwds.update(kwds_defaults)
--> 950 return _read(filepath_or_buffer, kwds)

File /opt/conda/lib/python3.10/site-packages/pandas/io/parsers/readers.py:605, in
_read(filepath_or_buffer, kwds)
    602 _validate_names(kwds.get("names", None))
    604 # Create the parser.
--> 605 parser = TextFileReader(filepath_or_buffer, **kwds)
    607 if chunksize or iterator:
    608     return parser

File /opt/conda/lib/python3.10/site-packages/pandas/io/parsers/readers.py:1442, in
TextFileReader.__init__(self, f, engine, **kwds)
   1439     self.options["has_index_names"] = kwds["has_index_names"]
   1441 self.handles: IOHandles | None = None
-> 1442 self._engine = self._make_engine(f, self.engine)
```

```
File /opt/conda/lib/python3.10/site-packages/pandas/io/parsers/readers.py:1735, in
TextFileReader._make_engine(self, f, engine)
   1733     if "b" not in mode:
   1734         mode += "b"
-> 1735 self.handles = get_handle(
   1736     f,
   1737     mode,
   1738     encoding=self.options.get("encoding", None),
   1739     compression=self.options.get("compression", None),
   1740     memory_map=self.options.get("memory_map", False),
   1741     is_text=is_text,
   1742     errors=self.options.get("encoding_errors", "strict"),
   1743     storage_options=self.options.get("storage_options", None),
   1744 )
   1745 assert self.handles is not None
   1746 f = self.handles.handle

File /opt/conda/lib/python3.10/site-packages/pandas/io/common.py:856, in get_handl
e(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_o
ptions)
    851 elif isinstance(handle, str):
    852     # Check whether the filename is to be opened in binary mode.
    853     # Binary mode does not support 'encoding' and 'newline'.
    854     if ioargs.encoding and "b" not in ioargs.mode:
    855         # Encoding
--> 856         handle = open(
    857             handle,
    858             ioargs.mode,
    859             encoding=ioargs.encoding,
    860             errors=errors,
    861             newline="",
    862         )
    863     else:
    864         # Binary mode
    865         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'lung_cancer_dataset.csv'
```

In [75]:

```
# Q.130 Do the EDA on this dataset: presidential election polls 2024 dataset and ex
# Dataset description:- this dataset comprises the result of a nationwide president
# link:-dataset nationwide russian election poll data from march 4,2024

pip install pandas numpy matplotlib seaborn
import pandas as pd

# Load the dataset
df = pd.read_csv('path_to_presidential_election_polls_2024.csv')

# Display the first few rows of the dataset
print(df.head())
print(df.shape)
print(df.columns)
print(df.head())
print(df.dtypes)
print(df.isnull().sum())
```

```
print(df.describe())
print(df.describe(include=['object']))
import matplotlib.pyplot as plt

df.hist(figsize=(12, 10))
plt.show()
import seaborn as sns

sns.boxplot(x=df['Some_Numerical_Column'])
plt.show()
sns.countplot(x='Candidate', data=df)
plt.show()
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()

sns.pairplot(df)
plt.show()
df_grouped = df.groupby('Candidate').mean()
print(df_grouped)

df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df['Candidate'].resample('W').mean().plot()
plt.show()
from sklearn.ensemble import RandomForestClassifier

X = df.drop(['Outcome'], axis=1)  # Features
y = df['Outcome']  # Target

model = RandomForestClassifier()
model.fit(X, y)
importance = model.feature_importances_

feature_importances = pd.DataFrame(importance, index=X.columns, columns=['Importanc
feature_importances.sort_values(by='Importance', ascending=False).plot(kind='bar')
plt.show()
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv('path_to_presidential_election_polls_2024.csv')

# Overview
print(df.shape)
print(df.columns)
print(df.head())

# Descriptive statistics
print(df.describe())
print(df.describe(include=['object']))

# Distribution of numerical features
df.hist(figsize=(12, 10))
plt.show()
```

```
# Categorical data analysis
sns.countplot(x='Candidate', data=df)
plt.show()

# Correlation matrix
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()

# Grouped analysis
df_grouped = df.groupby('Candidate').mean()
print(df_grouped)

# Time series analysis if applicable
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df['Candidate'].resample('W').mean().plot()
plt.show()
```

```
  Cell In[75], line 5
    pip install pandas numpy matplotlib seaborn
        ^
SyntaxError: invalid syntax
```

In [ ]: