```
# 1.2
'''
- a) `Serial_no: Invalid - The dot `.` is not allowed in identifier
names.
- b) `1st_Room`: Invalid - Identifiers cannot start with a digit.
- c) `Hundred$`: Invalid - The dollar sign `$` is not allowed in
identifier names.
- d) `Total_Marks`: Valid Underscores `_` are allowed, and the name
follows all identifier rules.
- e) `total-Marks`: Invalid - The hyphen `-` is not allowed in
identifier names.
- f) `Total Marks`: Invalid - Spaces are not allowed in identifier
names.
- g) `True`: Invalid - `True` is a reserved keyword in Python and
cannot be used as an identifier.
- h) `_Percentag`**: Valid - Underscores are allowed, and the name
follows all identifier rules.'''

'\n- **a) `Serial_no.`**: **Invalid** - The dot `.` is not allowed in
identifier names.\n- **b) `1st_Room`**: **Invalid** - Identifiers
cannot start with a digit.\n- **c) `Hundred$`**: **Invalid** - The
dollar sign `$` is not allowed in identifier names.\n- **d)
`Total_Marks`**: **Valid** - Underscores `_` are allowed, and the name
follows all identifier rules.\n- **e) `total-Marks`**: **Invalid** -
The hyphen `-` is not allowed in identifier names.\n- **f) `Total
Marks`**: **Invalid** - Spaces are not allowed in identifier names.\n-
**g) `True`**: **Invalid** - `True` is a reserved keyword in Python
and cannot be used as an identifier.\n- **h) `_Percentag`**: **Valid**
- Underscores are allowed, and the name follows all identifier rules.'

# 1.3

# a) add an element "freedom_fighter" in this list at the 0th index.

name = ["Mohan", "dash", "karam", "chandra", "gandhi", "Bapu"]
name.insert(0, "freedom_fighter")
print(name)


# c
name = ["Mohan", "dash", "karam", "chandra", "gandhi", "Bapu"]
name.extend(["NetaJi", "Bose"])
print(name)


# d
['Mohan', 'dash', 'karam', 'chandra', 'gandi', 'Bapuji']

['freedom_fighter', 'Mohan', 'dash', 'karam', 'chandra', 'gandhi',
'Bapu']
```

```
['Mohan', 'dash', 'karam', 'chandra', 'gandhi', 'Bapu', 'NetaJi',
'Bose']

['Mohan', 'dash', 'karam', 'chandra', 'gandi', 'Bapuji']

# 1.4

'''Output:

1. `animal.count('Human')` → `2`
   **Explanation**: There are 2 occurrences of ``Human'' in the list.

2. `animal.index('rat')` → `4`
   **Explanation**: ``rat'' is found at index 4.

3. `len(animal)` → `7`
   **Explanation**: The list `animal` has 7 elements.'''

"Output:\n\n1. `animal.count('Human')` → `2`  \n   **Explanation**:
There are 2 occurrences of ``Human'' in the list.\n\n2.
`animal.index('rat')` → `4`  \n   **Explanation**: ``rat'' is found at
index 4.\n\n3. `len(animal)` → `7`  \n   **Explanation**: The list
`animal` has 7 elements."

# 1.5

# a. 8
# b. Navneet
# c. tuple1[8][0]['roll_no']  # Output: 'N1'
# d. Output: 'navneet'
# e.element_22 = tuple1[1][2]

# 1.6


signal_color = input("Enter the color of the signal (RED, YELLOW,
GREEN): ").strip().upper()


if signal_color == "RED":
    print("Stop")
elif signal_color == "YELLOW":
    print("Stay")
elif signal_color == "GREEN":
    print("Go")
else:
    print("Invalid color! Please enter RED, YELLOW, or GREEN.")

Enter the color of the signal (RED, YELLOW, GREEN):  red

Stop
```

```python
# 1.7

# Function to perform calculations
def calculator():
    print("Simple Calculator")
    print("Select operation:")
    print("1. Addition (+)")
    print("2. Subtraction (-)")
    print("3. Multiplication (*)")
    print("4. Division (/)")


    operation = input("Enter the operation (1/2/3/4): ")


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


    if operation == '1':
        result = num1 + num2
        print(f"{num1} + {num2} = {result}")

    elif operation == '2':
        result = num1 - num2
        print(f"{num1} - {num2} = {result}")

    elif operation == '3':
        result = num1 * num2
        print(f"{num1} * {num2} = {result}")

    elif operation == '4':
        if num2 != 0:
            result = num1 / num2
            print(f"{num1} / {num2} = {result}")
        else:
            print("Error! Division by zero.")

    else:
        print("Invalid operation selected.")

calculator()
```

```
Simple Calculator
Select operation:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
```

```
Enter the operation (1/2/3/4):  2
Enter first number:  3
Enter second number:  4

3.0 - 4.0 = -1.0
```

```
# 1.8

num1 = 10
num2 = 20
num3 = 15


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


print(f"The largest number among {num1}, {num2}, and {num3} is:
{largest}")
```

```
The largest number among 10, 20, and 15 is: 20
```

```
# 1.9
def find_factors(number):
    i = 1
    factors = []

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

    return factors


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


factors = find_factors(number)
print(f"The factors of {number} are: {factors}")
```

```
Enter a whole number:  6

The factors of 6 are: [1, 2, 3, 6]
```

```
# 1.10


sum_of_numbers = 0

while True:
```

```
    number = float(input("Enter a positive number (enter a negative
number to stop): "))


    if number < 0:
        break


    sum_of_numbers += number


print(f"The sum of all positive numbers entered is: {sum_of_numbers}")
```

```
Enter a positive number (enter a negative number to stop):  4
Enter a positive number (enter a negative number to stop):  4
Enter a positive number (enter a negative number to stop):  4
Enter a positive number (enter a negative number to stop):  4
Enter a positive number (enter a negative number to stop):  4
Enter a positive number (enter a negative number to stop):  4
Enter a positive number (enter a negative number to stop):  4
Enter a positive number (enter a negative number to stop):  4
Enter a positive number (enter a negative number to stop):  4
Enter a positive number (enter a negative number to stop):  4
Enter a positive number (enter a negative number to stop):  4
Enter a positive number (enter a negative number to stop):  -1

The sum of all positive numbers entered is: 44.0
```

```
# 1.11


for num in range(2, 101):
    is_prime = True


    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            is_prime = False
            break

    if is_prime:
        print(num, end=" ")


2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

1.12

a

def calculate_grade(marks): total_marks = sum(marks) percentage = (total_marks / 500) * 100

```
if percentage > 85:
    grade = 'A'
elif percentage >= 75:
    grade = 'B'
elif percentage >= 50:
    grade = 'C'
elif percentage > 30:
    grade = 'D'
else:
    grade = 'Reappear'

return percentage, grade
```

marks = [] for i in range(1, 6): mark = float(input(f"Enter marks for subject {i}: ")) marks.append(mark)

percentage, grade = calculate_grade(marks)

print("\nMarks in five subjects:", marks) print(f"Total Percentage: {percentage:.2f}%") print(f"Grade: {grade}")

```python
# 1.12
# b
def find_grade(percentage):
    match percentage:
        case p if p > 85:
            return "A"
        case p if 75 <= p <= 85:
            return "B"
        case p if 50 <= p < 75:
            return "C"
        case p if 30 <= p < 50:
            return "D"
        case p if p < 30:
            return "Reappear"
        case _:
            return "Invalid percentage"


percentage = float(input("Enter the student's percentage: "))
```

```
grade = find_grade(percentage)
print(f"The student's grade is: {grade}")
```

```
Enter the student's percentage:  55

The student's grade is: C
```

```
# 1.13

def find_color(wavelength):
    match wavelength:
        case wl if 400.0 <= wl <= 440:
            return "Violet"
        case wl if 440 < wl <= 460:
            return "Indigo"
        case wl if 460 < wl <= 500:
            return "Blue"
        case wl if 500 < wl <= 570:
            return "Green"
        case wl if 570 < wl <= 590:
            return "Yellow"
        case wl if 590 < wl <= 620:
            return "Orange"
        case wl if 620 < wl <= 720:
            return "Red"
        case _:
            return "Wavelength not in the visible spectrum"


wavelength = float(input("Enter the wavelength in nanometers: "))


color = find_color(wavelength)
print(f"The color corresponding to the wavelength {wavelength} nm is:
{color}")
```

```
Enter the wavelength in nanometers:  344

The color corresponding to the wavelength 344.0 nm is: Wavelength not
in the visible spectrum
```

```
# 1.14
# a

G = 6.674e-11
mass_earth = 5.972e24
mass_sun = 0.989e30
distance_earth_sun = 1.496e11
```

```
force_earth_sun = (G * mass_earth * mass_sun) / (distance_earth_sun ** 2)


print(f"The gravitational force between the Earth and the Sun is: {force_earth_sun:.2e} N")
```

```
The gravitational force between the Earth and the Sun is: 1.76e+22 N
```

```
# 1.14
# b

G = 6.674 * 10**-11
mass_earth = 5.972e24
mass_moon = 7.34767309e22
distance_moon_earth = 3.844e8

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

print(f"The gravitational force between the Moon and the Earth is: {gravitational_force} N")
```

```
The gravitational force between the Moon and the Earth is: 1.9819334566450407e+20 N
```

```
# 1.14
# c

G = 6.674e-11

mass_earth = 5.972e24
mass_moon = 7.34767309e22
mass_sun = 1.989e30


distance_earth_sun = 1.496e11
distance_moon_earth = 3.844e8


force_earth_moon = G * (mass_earth * mass_moon) / distance_moon_earth**2


force_earth_sun = G * (mass_earth * mass_sun) / distance_earth_sun**2

force_earth_moon, force_earth_sun
```

```
(1.9819334566450407e+20, 3.5422368558580452e+22)
```

```python
# 1.14
# d

G = 6.674e-11

mass_earth = 5.972e24
mass_moon = 7.34767309e22
mass_sun = 1.989e30

distance_earth_sun = 1.496e11
distance_moon_earth = 3.844e8


force_earth_moon = G * (mass_earth * mass_moon) / (distance_moon_earth
** 2)


force_earth_sun = G * (mass_earth * mass_sun) / (distance_earth_sun **
2)

force_earth_moon, force_earth_sun
```

```
(1.9819334566450407e+20, 3.5422368558580452e+22)
```

```python
# 6
# a.
class MenuItem:
    def __init__(self, name, description, price, category):
        self.name = name
        self.description = description
        self.price = price
        self.category = category

    def __str__(self):
        return f"{self.name} ({self.category}): {self.description} - $
{self.price:.2f}"


class Menu:
    def __init__(self):
        self.items = []

    def add_item(self, item):
        self.items.append(item)
        print(f"Added: {item.name}")

    def remove_item(self, name):
        for item in self.items:
            if item.name == name:
                self.items.remove(item)
                print(f"Removed: {name}")
```

```
                return
        print(f"Item '{name}' not found.")

    def display_menu(self):
        if not self.items:
            print("The menu is currently empty.")
            return
        print("Menu:")
        for item in self.items:
            print(item)



if __name__ == "__main__":

    restaurant_menu = Menu()


    item1 = MenuItem("Cheeseburger", "Juicy beef patty with cheese",
8.99, "Main Course")
    item2 = MenuItem("Caesar Salad", "Fresh romaine lettuce with
Caesar dressing", 6.99, "Salad")
    item3 = MenuItem("Apple Pie", "Homemade apple pie with vanilla ice
cream", 4.99, "Dessert")


    restaurant_menu.add_item(item1)
    restaurant_menu.add_item(item2)
    restaurant_menu.add_item(item3)


    restaurant_menu.display_menu()


    restaurant_menu.remove_item("Caesar Salad")

    restaurant_menu.display_menu()
```

```
Added: Cheeseburger
Added: Caesar Salad
Added: Apple Pie
Menu:
Cheeseburger (Main Course): Juicy beef patty with cheese - $8.99
Caesar Salad (Salad): Fresh romaine lettuce with Caesar dressing -
$6.99
Apple Pie (Dessert): Homemade apple pie with vanilla ice cream - $4.99
Removed: Caesar Salad
Menu:
Cheeseburger (Main Course): Juicy beef patty with cheese - $8.99
Apple Pie (Dessert): Homemade apple pie with vanilla ice cream - $4.99
```

```python
# 6
# b
class MenuItem:
    def __init__(self, name, price, description):
        self.name = name
        self.price = price
        self.description = description

    def __str__(self):
        return f"{self.name} - ${self.price:.2f}: {self.description}"


class RestaurantMenu:
    def __init__(self):
        self.menu_items = {}

    def add_menu_item(self, item_name, price, description):
        if item_name in self.menu_items:
            print(f"Item '{item_name}' already exists. Use update to
modify it.")
        else:
            new_item = MenuItem(item_name, price, description)
            self.menu_items[item_name] = new_item
            print(f"Added menu item: {new_item}")

    def update_menu_item(self, item_name, new_price=None,
new_description=None):
        if item_name in self.menu_items:
            item = self.menu_items[item_name]
            if new_price is not None:
                item.price = new_price
            if new_description is not None:
                item.description = new_description
            print(f"Updated menu item: {item}")
        else:
            print(f"Item '{item_name}' does not exist in the menu.")

    def remove_menu_item(self, item_name):
        if item_name in self.menu_items:
            removed_item = self.menu_items.pop(item_name)
            print(f"Removed menu item: {removed_item}")
        else:
            print(f"Item '{item_name}' does not exist in the menu.")

    def display_menu(self):
        if not self.menu_items:
            print("The menu is currently empty.")
        else:
            print("Restaurant Menu:")
            for item in self.menu_items.values():
```

```
                print(item)


if __name__ == "__main__":
    restaurant_menu = RestaurantMenu()


    restaurant_menu.add_menu_item("Burger", 5.99, "Juicy beef burger
with lettuce and tomato.")
    restaurant_menu.add_menu_item("Pizza", 8.99, "Cheese pizza with a
variety of toppings.")
    restaurant_menu.add_menu_item("Pasta", 7.49, "Pasta in a creamy
Alfredo sauce.")


    restaurant_menu.display_menu()


    restaurant_menu.update_menu_item("Pizza", new_price=9.49,
new_description="Large cheese pizza with pepperoni.")


    restaurant_menu.remove_menu_item("Burger")


    restaurant_menu.display_menu()
```

```
Added menu item: Burger - $5.99: Juicy beef burger with lettuce and
tomato.
Added menu item: Pizza - $8.99: Cheese pizza with a variety of
toppings.
Added menu item: Pasta - $7.49: Pasta in a creamy Alfredo sauce.
Restaurant Menu:
Burger - $5.99: Juicy beef burger with lettuce and tomato.
Pizza - $8.99: Cheese pizza with a variety of toppings.
Pasta - $7.49: Pasta in a creamy Alfredo sauce.
Updated menu item: Pizza - $9.49: Large cheese pizza with pepperoni.
Removed menu item: Burger - $5.99: Juicy beef burger with lettuce and
tomato.
Restaurant Menu:
Pizza - $9.49: Large cheese pizza with pepperoni.
Pasta - $7.49: Pasta in a creamy Alfredo sauce.
```

```
# 6
# c
class MenuItem:
    def __init__(self, item_id, name, price):
        self.__item_id = item_id
```

```
        self.__name = name
        self.__price = price

    def get_item_id(self):
        return self.__item_id

    def get_name(self):
        return self.__name

    def get_price(self):
        return self.__price

    def __str__(self):
        return f"{self.__name} (ID: {self.__item_id}) - $
{self.__price:.2f}"


class Restaurant:
    def __init__(self, name):
        self.name = name
        self.menu_items = []

    def add_menu_item(self, item_id, name, price):
        new_item = MenuItem(item_id, name, price)
        self.menu_items.append(new_item)
        print(f"Added: {new_item}")

    def display_menu(self):
        print(f"\nMenu for {self.name}:")
        for item in self.menu_items:
            print(item)

    def get_total_price(self, item_ids):
        total = 0
        for item_id in item_ids:
            for item in self.menu_items:
                if item.get_item_id() == item_id:
                    total += item.get_price()
                    break
        return total



if __name__ == "__main__":
    restaurant = Restaurant("The Gourmet Kitchen")

    restaurant.add_menu_item(101, "Spaghetti Carbonara", 12.50)
    restaurant.add_menu_item(102, "Margherita Pizza", 10.00)
    restaurant.add_menu_item(103, "Caesar Salad", 8.50)
    restaurant.add_menu_item(104, "Tiramisu", 5.00)
```

```
    restaurant.display_menu()


    selected_items = [101, 103, 104]
    total_price = restaurant.get_total_price(selected_items)
    print(f"\nTotal price for selected items: ${total_price:.2f}")
```

```
Added: Spaghetti Carbonara (ID: 101) - $12.50
Added: Margherita Pizza (ID: 102) - $10.00
Added: Caesar Salad (ID: 103) - $8.50
Added: Tiramisu (ID: 104) - $5.00

Menu for The Gourmet Kitchen:
Spaghetti Carbonara (ID: 101) - $12.50
Margherita Pizza (ID: 102) - $10.00
Caesar Salad (ID: 103) - $8.50
Tiramisu (ID: 104) - $5.00

Total price for selected items: $26.00
```

```
# 6
# d


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

    def get_details(self):
        return f"{self.name}: ${self.price:.2f}"

class FoodItem(MenuItem):
    def __init__(self, name, price, calories):
        super().__init__(name, price)
        self.calories = calories

    def get_details(self):
        return f"{self.name} (Food): ${self.price:.2f}, Calories:
{self.calories} kcal"


class BeverageItem(MenuItem):
    def __init__(self, name, price, volume):
        super().__init__(name, price)
        self.volume = volume

    def get_details(self):
        return f"{self.name} (Beverage): ${self.price:.2f}, Volume:
{self.volume} ml"
```

```
class RestaurantMenu:
    def __init__(self):
        self.menu_items = []

    def add_item(self, item):
        self.menu_items.append(item)

    def display_menu(self):
        print("Restaurant Menu:")
        for item in self.menu_items:
            print(item.get_details())


if __name__ == "__main__":

    menu = RestaurantMenu()


    burger = FoodItem("Cheeseburger", 8.99, 500)
    pizza = FoodItem("Margherita Pizza", 10.49, 700)


    cola = BeverageItem("Cola", 1.99, 355)
    lemonade = BeverageItem("Lemonade", 2.49, 500)


    menu.add_item(burger)
    menu.add_item(pizza)
    menu.add_item(cola)
    menu.add_item(lemonade)


    menu.display_menu()
```

```
Restaurant Menu:
Cheeseburger (Food): $8.99, Calories: 500 kcal
Margherita Pizza (Food): $10.49, Calories: 700 kcal
Cola (Beverage): $1.99, Volume: 355 ml
Lemonade (Beverage): $2.49, Volume: 500 ml
```

```
# 7
# a

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

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


    def check_in(self):
        if self.__is_available:
            self.__is_available = False
            print(f"Room {self.room_number} has been checked in.")
        else:
            print(f"Room {self.room_number} is not available.")


    def check_out(self):
        if not self.__is_available:
            self.__is_available = True
            print(f"Room {self.room_number} has been checked out.")
        else:
            print(f"Room {self.room_number} is already available.")


    def get_details(self):
        availability_status = "Available" if self.__is_available else
"Not Available"
        return (f"Room {self.room_number}: Type = {self.room_type},
Rate = ${self.rate:.2f} per night, "
                f"Availability = {availability_status}")


if __name__ == "__main__":

    room1 = Room(101, "Single", 100.00)
    room2 = Room(102, "Double", 150.00)
    room3 = Room(103, "Suite", 250.00)


    print(room1.get_details())
    print(room2.get_details())
    print(room3.get_details())


    room1.check_in()
    print(room1.get_details())

    room1.check_in()


    room1.check_out()
    print(room1.get_details())
```

```
    room1.check_out()
```

```
Room 101: Type = Single, Rate = $100.00 per night, Availability =
Available
Room 102: Type = Double, Rate = $150.00 per night, Availability =
Available
Room 103: Type = Suite, Rate = $250.00 per night, Availability =
Available
Room 101 has been checked in.
Room 101: Type = Single, Rate = $100.00 per night, Availability = Not
Available
Room 101 is not available.
Room 101 has been checked out.
Room 101: Type = Single, Rate = $100.00 per night, Availability =
Available
Room 101 is already available.
```

```
# 7b

class Room:
    def __init__(self, room_number, room_type, price):
        self.room_number = room_number
        self.room_type = room_type
        self.price = price
        self.is_booked = False
        self.guest_name = None

    def book_room(self, guest_name):
        if not self.is_booked:
            self.is_booked = True
            self.guest_name = guest_name
            print(f"Room {self.room_number} has been booked by
{guest_name}.")
        else:
            print(f"Room {self.room_number} is already booked.")

    def check_in(self, guest_name):
        if self.is_booked and self.guest_name == guest_name:
            print(f"{guest_name} has checked into room
{self.room_number}.")
        else:
            print(f"Check-in failed for room {self.room_number}.
Either the room is not booked or the name doesn't match.")

    def check_out(self):
        if self.is_booked:
            print(f"{self.guest_name} has checked out from room
{self.room_number}.")
```

```
            self.is_booked = False
            self.guest_name = None
        else:
            print(f"Room {self.room_number} is not currently booked.")

    def get_room_details(self):
        booking_status = "Booked" if self.is_booked else "Available"
        return f"Room {self.room_number}: {self.room_type}, Price: $
{self.price:.2f}, Status: {booking_status}"

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

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

    def find_room(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number:
                return room
        return None

    def display_rooms(self):
        print(f"{self.name} Room List:")
        for room in self.rooms:
            print(room.get_room_details())

    def book_room(self, room_number, guest_name):
        room = self.find_room(room_number)
        if room:
            room.book_room(guest_name)
        else:
            print(f"Room {room_number} does not exist.")

    def check_in(self, room_number, guest_name):
        room = self.find_room(room_number)
        if room:
            room.check_in(guest_name)
        else:
            print(f"Room {room_number} does not exist.")

    def check_out(self, room_number):
        room = self.find_room(room_number)
        if room:
            room.check_out()
        else:
            print(f"Room {room_number} does not exist.")
```

```python
if __name__ == "__main__":

    hotel = Hotel("Grand Python Hotel")

    hotel.add_room(Room(101, "Single", 100.00))
    hotel.add_room(Room(102, "Double", 150.00))
    hotel.add_room(Room(103, "Suite", 300.00))


    hotel.display_rooms()


    hotel.book_room(101, "Alice")


    hotel.book_room(101, "Bob")


    hotel.check_in(101, "Alice")


    hotel.check_in(101, "Bob")


    hotel.check_out(101)


    hotel.check_out(101)


    hotel.display_rooms()
```

```
Grand Python Hotel Room List:
Room 101: Single, Price: $100.00, Status: Available
Room 102: Double, Price: $150.00, Status: Available
Room 103: Suite, Price: $300.00, Status: Available
Room 101 has been booked by Alice.
Room 101 is already booked.
Alice has checked into room 101.
Check-in failed for room 101. Either the room is not booked or the
name doesn't match.
Alice has checked out from room 101.
Room 101 is not currently booked.
Grand Python Hotel Room List:
Room 101: Single, Price: $100.00, Status: Available
Room 102: Double, Price: $150.00, Status: Available
Room 103: Suite, Price: $300.00, Status: Available
```

```python
# 7c
```

```
class HotelRoom:
    def __init__(self, room_number, room_type, price_per_night):
        self.__room_number = room_number  # Private attribute
(encapsulated)
        self.room_type = room_type
        self.price_per_night = price_per_night
        self.is_occupied = False


    def get_room_number(self):
        return self.__room_number


    def book_room(self):
        if not self.is_occupied:
            self.is_occupied = True
            print(f"Room {self.__room_number} booked successfully.")
        else:
            print(f"Room {self.__room_number} is already occupied.")


    def checkout(self):
        if self.is_occupied:
            self.is_occupied = False
            print(f"Room {self.__room_number} is now available.")
        else:
            print(f"Room {self.__room_number} is already available.")

    def get_room_details(self):
        occupancy_status = "Occupied" if self.is_occupied else
"Available"
        return (f"Room Number: {self.get_room_number()}, Type:
{self.room_type}, "
                f"Price per Night: ${self.price_per_night}, Status:
{occupancy_status}")

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


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


    def display_rooms(self):
        print("Hotel Room Details:")
        for room in self.rooms:
            print(room.get_room_details())
```

```
    def find_room_by_number(self, room_number):
        for room in self.rooms:
            if room.get_room_number() == room_number:
                return room
        return None


if __name__ == "__main__":

    hotel = HotelManagement()

    room1 = HotelRoom(101, "Single", 100)
    room2 = HotelRoom(102, "Double", 150)
    room3 = HotelRoom(103, "Suite", 250)

    hotel.add_room(room1)
    hotel.add_room(room2)
    hotel.add_room(room3)


    hotel.display_rooms()


    room_to_book = hotel.find_room_by_number(101)
    if room_to_book:
        room_to_book.book_room()

    room_to_checkout = hotel.find_room_by_number(101)
    if room_to_checkout:
        room_to_checkout.checkout()


    hotel.display_rooms()
```

```
Hotel Room Details:
Room Number: 101, Type: Single, Price per Night: $100, Status:
Available
Room Number: 102, Type: Double, Price per Night: $150, Status:
Available
Room Number: 103, Type: Suite, Price per Night: $250, Status:
Available
Room 101 booked successfully.
Room 101 is now available.
Hotel Room Details:
Room Number: 101, Type: Single, Price per Night: $100, Status:
Available
Room Number: 102, Type: Double, Price per Night: $150, Status:
Available
Room Number: 103, Type: Suite, Price per Night: $250, Status:
Available
```

```python
# 7d


class Room:
    def __init__(self, room_number, price_per_night,
is_occupied=False):
        self.room_number = room_number
        self.price_per_night = price_per_night
        self.is_occupied = is_occupied

    def check_in(self):
        if not self.is_occupied:
            self.is_occupied = True
            return f"Room {self.room_number} is now occupied."
        else:
            return f"Room {self.room_number} is already occupied."

    def check_out(self):
        if self.is_occupied:
            self.is_occupied = False
            return f"Room {self.room_number} is now available."
        else:
            return f"Room {self.room_number} is already available."

    def get_details(self):
        status = "Occupied" if self.is_occupied else "Available"
        return f"Room {self.room_number}: $
{self.price_per_night:.2f}/night, Status: {status}"


class SuiteRoom(Room):
    def __init__(self, room_number, price_per_night,
is_occupied=False, has_jacuzzi=True):
        super().__init__(room_number, price_per_night, is_occupied)
        self.has_jacuzzi = has_jacuzzi

    def get_details(self):
        status = "Occupied" if self.is_occupied else "Available"
        jacuzzi_info = "with Jacuzzi" if self.has_jacuzzi else
"without Jacuzzi"
        return f"Suite Room {self.room_number}: $
{self.price_per_night:.2f}/night, {jacuzzi_info}, Status: {status}"


class StandardRoom(Room):
    def __init__(self, room_number, price_per_night,
is_occupied=False, has_sea_view=False):
        super().__init__(room_number, price_per_night, is_occupied)
        self.has_sea_view = has_sea_view
```

```
    def get_details(self):
        status = "Occupied" if self.is_occupied else "Available"
        view_info = "with Sea View" if self.has_sea_view else "without
Sea View"
        return f"Standard Room {self.room_number}: $
{self.price_per_night:.2f}/night, {view_info}, Status: {status}"


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

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

    def display_rooms(self):
        print("Hotel Rooms:")
        for room in self.rooms:
            print(room.get_details())

    def find_room_by_number(self, room_number):
        for room in self.rooms:
            if room.room_number == room_number:
                return room
        return None


if __name__ == "__main__":

    hotel = HotelManagementSystem()


    suite1 = SuiteRoom(room_number=101, price_per_night=250.0)
    standard1 = StandardRoom(room_number=102, price_per_night=150.0,
has_sea_view=True)
    suite2 = SuiteRoom(room_number=103, price_per_night=300.0,
has_jacuzzi=False)
    standard2 = StandardRoom(room_number=104, price_per_night=120.0)


    hotel.add_room(suite1)
    hotel.add_room(standard1)
    hotel.add_room(suite2)
    hotel.add_room(standard2)


    hotel.display_rooms()


    print(suite1.check_in())
```

```
    hotel.display_rooms()


    print(suite1.check_out())


    hotel.display_rooms()
```

```
Hotel Rooms:
Suite Room 101: $250.00/night, with Jacuzzi, Status: Available
Standard Room 102: $150.00/night, with Sea View, Status: Available
Suite Room 103: $300.00/night, without Jacuzzi, Status: Available
Standard Room 104: $120.00/night, without Sea View, Status: Available
Room 101 is now occupied.
Hotel Rooms:
Suite Room 101: $250.00/night, with Jacuzzi, Status: Occupied
Standard Room 102: $150.00/night, with Sea View, Status: Available
Suite Room 103: $300.00/night, without Jacuzzi, Status: Available
Standard Room 104: $120.00/night, without Sea View, Status: Available
Room 101 is now available.
Hotel Rooms:
Suite Room 101: $250.00/night, with Jacuzzi, Status: Available
Standard Room 102: $150.00/night, with Sea View, Status: Available
Suite Room 103: $300.00/night, without Jacuzzi, Status: Available
Standard Room 104: $120.00/night, without Sea View, Status: Available
```

```
# 8a


class Person:
    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender

    def get_details(self):
        return f"Name: {self.name}, Age: {self.age}, Gender:
{self.gender}"


class Member(Person):
    def __init__(self, name, age, gender, membership_type):
        super().__init__(name, age, gender)
        self.membership_type = membership_type
        self.workout_plan = None

    def assign_workout_plan(self, workout_plan):
        self.workout_plan = workout_plan

    def get_details(self):
```

```
        details = super().get_details()
        details += f", Membership Type: {self.membership_type}"
        if self.workout_plan:
            details += f", Workout Plan: {self.workout_plan.name}"
        return details


class Trainer(Person):
    def __init__(self, name, age, gender, specialty):
        super().__init__(name, age, gender)
        self.specialty = specialty
        self.assigned_members = []

    def assign_member(self, member):
        self.assigned_members.append(member)

    def get_details(self):
        details = super().get_details()
        details += f", Specialty: {self.specialty}, Assigned Members:
{len(self.assigned_members)}"
        return details


class WorkoutPlan:
    def __init__(self, name, duration, exercises):
        self.name = name
        self.duration = duration  # in weeks
        self.exercises = exercises

    def get_details(self):
        return f"Workout Plan: {self.name}, Duration: {self.duration}
weeks, Exercises: {', '.join(self.exercises)}"


class FitnessClub:
    def __init__(self):
        self.members = []
        self.trainers = []
        self.workout_plans = []

    def add_member(self, member):
        self.members.append(member)

    def add_trainer(self, trainer):
        self.trainers.append(trainer)

    def add_workout_plan(self, workout_plan):
        self.workout_plans.append(workout_plan)

    def assign_trainer_to_member(self, trainer_name, member_name):
```

```
        trainer = next((t for t in self.trainers if t.name ==
trainer_name), None)
        member = next((m for m in self.members if m.name ==
member_name), None)
        if trainer and member:
            trainer.assign_member(member)
            return f"Trainer {trainer_name} assigned to member
{member_name}"
        return "Trainer or member not found"

    def display_members(self):
        print("Members:")
        for member in self.members:
            print(member.get_details())

    def display_trainers(self):
        print("Trainers:")
        for trainer in self.trainers:
            print(trainer.get_details())

    def display_workout_plans(self):
        print("Workout Plans:")
        for plan in self.workout_plans:
            print(plan.get_details())

if __name__ == "__main__":

    club = FitnessClub()


    beginner_plan = WorkoutPlan("Beginner", 4, ["Pushups", "Squats",
"Plank"])
    advanced_plan = WorkoutPlan("Advanced", 8, ["Deadlift", "Bench
Press", "Pull-ups"])
    club.add_workout_plan(beginner_plan)
    club.add_workout_plan(advanced_plan)


    member1 = Member("Alice", 30, "Female", "Gold")
    member2 = Member("Bob", 25, "Male", "Silver")
    member1.assign_workout_plan(beginner_plan)
    member2.assign_workout_plan(advanced_plan)
    club.add_member(member1)
    club.add_member(member2)


    trainer1 = Trainer("John", 40, "Male", "Strength Training")
    trainer2 = Trainer("Emma", 35, "Female", "Cardio")
    club.add_trainer(trainer1)
    club.add_trainer(trainer2)
```

```
    club.assign_trainer_to_member("John", "Alice")
    club.assign_trainer_to_member("Emma", "Bob")


    club.display_members()
    club.display_trainers()
    club.display_workout_plans()
```

```
Members:
Name: Alice, Age: 30, Gender: Female, Membership Type: Gold, Workout
Plan: Beginner
Name: Bob, Age: 25, Gender: Male, Membership Type: Silver, Workout
Plan: Advanced
Trainers:
Name: John, Age: 40, Gender: Male, Specialty: Strength Training,
Assigned Members: 1
Name: Emma, Age: 35, Gender: Female, Specialty: Cardio, Assigned
Members: 1
Workout Plans:
Workout Plan: Beginner, Duration: 4 weeks, Exercises: Pushups, Squats,
Plank
Workout Plan: Advanced, Duration: 8 weeks, Exercises: Deadlift, Bench
Press, Pull-ups
```

```
# 8b

from datetime import datetime, timedelta


class Member:
    def __init__(self, member_id, name, membership_type,
membership_start_date):
        self.member_id = member_id
        self.name = name
        self.membership_type = membership_type
        self.membership_start_date = membership_start_date
        self.membership_end_date =
self.calculate_membership_end_date()

    def calculate_membership_end_date(self):
        if self.membership_type == "Monthly":
            return self.membership_start_date + timedelta(days=30)
        elif self.membership_type == "Annual":
            return self.membership_start_date + timedelta(days=365)

    def renew_membership(self, membership_type):
        self.membership_type = membership_type
        self.membership_start_date = datetime.now()
```

```
        self.membership_end_date =
self.calculate_membership_end_date()
        print(f"Membership for {self.name} has been renewed. New
expiry date: {self.membership_end_date.strftime('%Y-%m-%d')}")

    def cancel_membership(self):
        self.membership_end_date = datetime.now()
        print(f"Membership for {self.name} has been canceled.")

    def display_member_info(self):
        print(f"Member ID: {self.member_id}")
        print(f"Name: {self.name}")
        print(f"Membership Type: {self.membership_type}")
        print(f"Membership Start Date:
{self.membership_start_date.strftime('%Y-%m-%d')}")
        print(f"Membership End Date:
{self.membership_end_date.strftime('%Y-%m-%d')}")


class FitnessClub:
    def __init__(self):
        self.members = {}
        self.next_member_id = 1

    def register_member(self, name, membership_type):
        member_id = self.next_member_id
        self.next_member_id += 1
        membership_start_date = datetime.now()
        new_member = Member(member_id, name, membership_type,
membership_start_date)
        self.members[member_id] = new_member
        print(f"New member registered: {name}")
        return member_id

    def renew_membership(self, member_id, membership_type):
        if member_id in self.members:
            member = self.members[member_id]
            member.renew_membership(membership_type)
        else:
            print(f"No member found with ID {member_id}")

    def cancel_membership(self, member_id):
        if member_id in self.members:
            member = self.members[member_id]
            member.cancel_membership()
            del self.members[member_id]
            print(f"Member ID {member_id} has been removed from the
system.")
        else:
            print(f"No member found with ID {member_id}")
```

```
    def display_all_members(self):
        if self.members:
            print("All members of the fitness club:")
            for member in self.members.values():
                member.display_member_info()
                print("-" * 30)
        else:
            print("No members registered in the fitness club.")


if __name__ == "__main__":

    club = FitnessClub()


    member_id_1 = club.register_member("Alice", "Monthly")
    member_id_2 = club.register_member("Bob", "Annual")


    club.display_all_members()


    club.renew_membership(member_id_1, "Annual")


    club.cancel_membership(member_id_2)


    club.display_all_members()
```

```
New member registered: Alice
New member registered: Bob
All members of the fitness club:
Member ID: 1
Name: Alice
Membership Type: Monthly
Membership Start Date: 2024-08-22
Membership End Date: 2024-09-21
------------------------------
Member ID: 2
Name: Bob
Membership Type: Annual
Membership Start Date: 2024-08-22
Membership End Date: 2025-08-22
------------------------------
Membership for Alice has been renewed. New expiry date: 2025-08-22
Membership for Bob has been canceled.
Member ID 2 has been removed from the system.
All members of the fitness club:
```

```
Member ID: 1
Name: Alice
Membership Type: Annual
Membership Start Date: 2024-08-22
Membership End Date: 2025-08-22
------------------------------

# 8c
class FitnessEvent:
    def __init__(self, name, date, time, instructor, event_id):
        self.name = name
        self.date = date
        self.time = time
        self.instructor = instructor
        self.__event_id = event_id


    def get_event_id(self):
        return self.__event_id

    def get_event_details(self):
        return f"Event: {self.name}, Date: {self.date}, Time:
{self.time}, Instructor: {self.instructor}"

class FitnessClub:
    def __init__(self, club_name):
        self.club_name = club_name
        self.events = []

    def add_event(self, event):
        self.events.append(event)

    def remove_event(self, event_id):
        self.events = [event for event in self.events if
event.get_event_id() != event_id]

    def display_events(self):
        print(f"{self.club_name} - Scheduled Events:")
        for event in self.events:
            print(event.get_event_details())


if __name__ == "__main__":

    my_club = FitnessClub("Healthy Living Fitness Club")


    yoga = FitnessEvent("Yoga Session", "2024-09-01", "08:00 AM",
"Alice Smith", event_id="EVT001")
    spinning = FitnessEvent("Spinning Class", "2024-09-02", "06:00
```

```
PM", "John Doe", event_id="EVT002")
    pilates = FitnessEvent("Pilates Workshop", "2024-09-03", "10:00
AM", "Emily Johnson", event_id="EVT003")


    my_club.add_event(yoga)
    my_club.add_event(spinning)
    my_club.add_event(pilates)


    my_club.display_events()


    my_club.remove_event("EVT002")


    print("\nAfter removing Spinning Class:")
    my_club.display_events()
```

```
Healthy Living Fitness Club - Scheduled Events:
Event: Yoga Session, Date: 2024-09-01, Time: 08:00 AM, Instructor:
Alice Smith
Event: Spinning Class, Date: 2024-09-02, Time: 06:00 PM, Instructor:
John Doe
Event: Pilates Workshop, Date: 2024-09-03, Time: 10:00 AM, Instructor:
Emily Johnson

After removing Spinning Class:
Healthy Living Fitness Club - Scheduled Events:
Event: Yoga Session, Date: 2024-09-01, Time: 08:00 AM, Instructor:
Alice Smith
Event: Pilates Workshop, Date: 2024-09-03, Time: 10:00 AM, Instructor:
Emily Johnson
```

```
# 8d


class Member:
    def __init__(self, member_id, name, membership_type):
        self.member_id = member_id
        self.name = name
        self.membership_type = membership_type

    def get_details(self):
        return f"Member ID: {self.member_id}, Name: {self.name},
Membership Type: {self.membership_type}"

    def renew_membership(self):
        print(f"Membership for {self.name} has been renewed.")
```

```
class FamilyMember(Member):
    def __init__(self, member_id, name, family_members,
membership_type="Family"):
        super().__init__(member_id, name, membership_type)
        self.family_members = family_members

    def add_family_member(self, family_member_name):
        self.family_members.append(family_member_name)
        print(f"Family member {family_member_name} added to
{self.name}'s membership.")

    def get_details(self):
        family_list = ', '.join(self.family_members)
        return f"{super().get_details()}, Family Members:
{family_list}"


class IndividualMember(Member):
    def __init__(self, member_id, name, membership_type="Individual"):
        super().__init__(member_id, name, membership_type)

    def upgrade_to_family(self, family_members):
        print(f"Upgrading {self.name} to Family Membership.")
        return FamilyMember(self.member_id, self.name, family_members)


class FitnessClub:
    def __init__(self):
        self.members = []

    def add_member(self, member):
        self.members.append(member)
        print(f"Member {member.name} added to the club.")

    def display_members(self):
        print("Fitness Club Members:")
        for member in self.members:
            print(member.get_details())


if __name__ == "__main__":

    club = FitnessClub()


    john = IndividualMember(1, "John Doe")


    jane = FamilyMember(2, "Jane Smith", ["Tom Smith", "Lucy Smith"])
```

```
    club.add_member(john)
    club.add_member(jane)


    club.display_members()


    john_family = john.upgrade_to_family(["Anna Doe"])
    club.add_member(john_family)


    jane.add_family_member("Mark Smith")


    club.display_members()
```

```
Member John Doe added to the club.
Member Jane Smith added to the club.
Fitness Club Members:
Member ID: 1, Name: John Doe, Membership Type: Individual
Member ID: 2, Name: Jane Smith, Membership Type: Family, Family
Members: Tom Smith, Lucy Smith
Upgrading John Doe to Family Membership.
Member John Doe added to the club.
Family member Mark Smith added to Jane Smith's membership.
Fitness Club Members:
Member ID: 1, Name: John Doe, Membership Type: Individual
Member ID: 2, Name: Jane Smith, Membership Type: Family, Family
Members: Tom Smith, Lucy Smith, Mark Smith
Member ID: 1, Name: John Doe, Membership Type: Family, Family Members:
Anna Doe
```

```
# 9a

from datetime import datetime

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_name):
        self.__attendees.append(attendee_name)

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

```
        else:
            print(f"{attendee_name} is not in the attendees list.")

    def get_attendees(self):
        return self.__attendees

    def get_details(self):
        return (f"Event: {self.name}\n"
                f"Date: {self.date}\n"
                f"Time: {self.time}\n"
                f"Location: {self.location}\n"
                f"Number of Attendees: {len(self.__attendees)}")


if __name__ == "__main__":

    event = Event("Python Workshop", "2024-09-15", "10:00 AM", "Tech
Conference Hall")


    event.add_attendee("Alice")
    event.add_attendee("Bob")
    event.add_attendee("Charlie")


    print(event.get_details())


    print("\nAttendees:")
    for attendee in event.get_attendees():
        print(attendee)


    event.remove_attendee("Bob")


    print("\nAfter removing an attendee:")
    print(event.get_details())
```

```
Event: Python Workshop
Date: 2024-09-15
Time: 10:00 AM
Location: Tech Conference Hall
Number of Attendees: 3

Attendees:
Alice
Bob
Charlie

After removing an attendee:
```

```
Event: Python Workshop
Date: 2024-09-15
Time: 10:00 AM
Location: Tech Conference Hall
Number of Attendees: 2

# 9b


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_name):
        self._attendees.append(attendee_name)
        print(f"Added {attendee_name} to the event {self.name}.")

    def remove_attendee(self, attendee_name):
        if attendee_name in self._attendees:
            self._attendees.remove(attendee_name)
            print(f"Removed {attendee_name} from the event
{self.name}.")
        else:
            print(f"{attendee_name} is not in the attendee list for
{self.name}.")

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

    def get_event_details(self):
        return f"Event: {self.name}\nDate: {self.date}\nTime:
{self.time}\nLocation: {self.location}\nTotal Attendees:
{self.get_total_attendees()}"


if __name__ == "__main__":

    event = Event("Python Workshop", "2024-09-01", "10:00 AM",
"Conference Hall A")


    event.add_attendee("Alice")
    event.add_attendee("Bob")
    event.add_attendee("Charlie")
```

```
    event.remove_attendee("Bob")


    total_attendees = event.get_total_attendees()
    print(f"Total number of attendees: {total_attendees}")


    print("\nEvent Details:")
    print(event.get_event_details())
```

```
Added Alice to the event Python Workshop.
Added Bob to the event Python Workshop.
Added Charlie to the event Python Workshop.
Removed Bob from the event Python Workshop.
Total number of attendees: 2

Event Details:
Event: Python Workshop
Date: 2024-09-01
Time: 10:00 AM
Location: Conference Hall A
Total Attendees: 2
```

```
# 9c

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

    def __generate_event_id(self):
        import random
        return random.randint(1000, 9999)

    def add_attendee(self, attendee_name):
        self.attendees.append(attendee_name)
        print(f"{attendee_name} has been added to the event:
{self.name}")

    def remove_attendee(self, attendee_name):
        if attendee_name in self.attendees:
            self.attendees.remove(attendee_name)
            print(f"{attendee_name} has been removed from the event:
{self.name}")
        else:
            print(f"{attendee_name} is not in the attendee list.")
```

```
    def get_event_details(self):
        return (f"Event ID: {self.__event_id}\n"
                f"Name: {self.name}\n"
                f"Date: {self.date}\n"
                f"Time: {self.time}\n"
                f"Location: {self.location}\n"
                f"Attendees: {', '.join(self.attendees) if
self.attendees else 'No attendees yet'}")

class EventManagementSystem:
    def __init__(self):
        self.events = []

    def create_event(self, name, date, time, location):
        new_event = Event(name, date, time, location)
        self.events.append(new_event)
        print(f"Event '{name}' created successfully.")

    def display_events(self):
        if not self.events:
            print("No events available.")
            return
        print("List of Events:")
        for event in self.events:
            print(event.get_event_details())
            print("-" * 40)


if __name__ == "__main__":
    ems = EventManagementSystem()


    ems.create_event("Music Concert", "2024-09-01", "18:00", "Central
Park")
    ems.create_event("Art Exhibition", "2024-09-10", "10:00", "City
Gallery")


    ems.display_events()


    ems.events[0].add_attendee("Alice")
    ems.events[0].add_attendee("Bob")


    print("\nUpdated Event Details:")
    print(ems.events[0].get_event_details())
```

```
    ems.events[0].remove_attendee("Alice")


    print("\nUpdated Event Details after removal:")
    print(ems.events[0].get_event_details())
```

```
Event 'Music Concert' created successfully.
Event 'Art Exhibition' created successfully.
List of Events:
Event ID: 9627
Name: Music Concert
Date: 2024-09-01
Time: 18:00
Location: Central Park
Attendees: No attendees yet
----------------------------------------
Event ID: 9515
Name: Art Exhibition
Date: 2024-09-10
Time: 10:00
Location: City Gallery
Attendees: No attendees yet
----------------------------------------
Alice has been added to the event: Music Concert
Bob has been added to the event: Music Concert

Updated Event Details:
Event ID: 9627
Name: Music Concert
Date: 2024-09-01
Time: 18:00
Location: Central Park
Attendees: Alice, Bob
Alice has been removed from the event: Music Concert

Updated Event Details after removal:
Event ID: 9627
Name: Music Concert
Date: 2024-09-01
Time: 18:00
Location: Central Park
Attendees: Bob
```

```
# 9d


class Event:
    def __init__(self, name, date, location):
        self.name = name
        self.date = date
```

```
        self.location = location

    def get_details(self):
        return f"Event: {self.name}\nDate: {self.date}\nLocation:
{self.location}"


class PrivateEvent(Event):
    def __init__(self, name, date, location, guest_list):
        super().__init__(name, date, location)
        self.guest_list = guest_list  # List of guests invited

    def get_details(self):
        return (f"Private Event: {self.name}\nDate: {self.date}\
nLocation: {self.location}\n"
                f"Guests: {', '.join(self.guest_list)}")

    def add_guest(self, guest_name):
        self.guest_list.append(guest_name)

    def remove_guest(self, guest_name):
        self.guest_list.remove(guest_name)


class PublicEvent(Event):
    def __init__(self, name, date, location, ticket_price,
max_attendees):
        super().__init__(name, date, location)
        self.ticket_price = ticket_price
        self.max_attendees = max_attendees
        self.attendees = 0

    def get_details(self):
        return (f"Public Event: {self.name}\nDate: {self.date}\
nLocation: {self.location}\n"
                f"Ticket Price: ${self.ticket_price:.2f}\nMax
Attendees: {self.max_attendees}\n"
                f"Current Attendees: {self.attendees}")

    def register_attendee(self):
        if self.attendees < self.max_attendees:
            self.attendees += 1
            print(f"Attendee registered for {self.name}. Total
attendees: {self.attendees}")
        else:
            print(f"Registration closed for {self.name}. Event is
fully booked.")


class EventManagementSystem:
```

```
    def __init__(self):
        self.events = []

    def add_event(self, event):
        self.events.append(event)

    def display_events(self):
        print("Event List:")
        for event in self.events:
            print(event.get_details())
            print("-" * 40)

if __name__ == "__main__":

    system = EventManagementSystem()


    birthday_party = PrivateEvent("John's Birthday Party", "2024-09-
15", "John's House", ["Alice", "Bob", "Charlie"])
    wedding = PrivateEvent("Alice & Bob's Wedding", "2024-10-20",
"Beach Resort", ["John", "Charlie", "Dave"])


    concert = PublicEvent("Summer Concert", "2024-08-30", "City
Arena", 50.00, 500)
    exhibition = PublicEvent("Art Exhibition", "2024-09-05", "Art
Gallery", 20.00, 200)


    system.add_event(birthday_party)
    system.add_event(wedding)
    system.add_event(concert)
    system.add_event(exhibition)


    system.display_events()


    concert.register_attendee()


    birthday_party.add_guest("Dave")


    system.display_events()
```

```
Event List:
Private Event: John's Birthday Party
Date: 2024-09-15
Location: John's House
```

```
Guests: Alice, Bob, Charlie
----------------------------------------
Private Event: Alice & Bob's Wedding
Date: 2024-10-20
Location: Beach Resort
Guests: John, Charlie, Dave
----------------------------------------
Public Event: Summer Concert
Date: 2024-08-30
Location: City Arena
Ticket Price: $50.00
Max Attendees: 500
Current Attendees: 0
----------------------------------------
Public Event: Art Exhibition
Date: 2024-09-05
Location: Art Gallery
Ticket Price: $20.00
Max Attendees: 200
Current Attendees: 0
----------------------------------------
Attendee registered for Summer Concert. Total attendees: 1
Event List:
Private Event: John's Birthday Party
Date: 2024-09-15
Location: John's House
Guests: Alice, Bob, Charlie, Dave
----------------------------------------
Private Event: Alice & Bob's Wedding
Date: 2024-10-20
Location: Beach Resort
Guests: John, Charlie, Dave
----------------------------------------
Public Event: Summer Concert
Date: 2024-08-30
Location: City Arena
Ticket Price: $50.00
Max Attendees: 500
Current Attendees: 1
----------------------------------------
Public Event: Art Exhibition
Date: 2024-09-05
Location: Art Gallery
Ticket Price: $20.00
Max Attendees: 200
Current Attendees: 0
----------------------------------------

# 10a
```

```python
class Flight:
    def __init__(self, flight_number, departure_airport,
arrival_airport, departure_time, arrival_time, available_seats):
        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 = available_seats  # Private attribute
for available seats

    def get_flight_details(self):
        return (f"Flight Number: {self.flight_number}\n"
                f"Departure Airport: {self.departure_airport}\n"
                f"Arrival Airport: {self.arrival_airport}\n"
                f"Departure Time: {self.departure_time}\n"
                f"Arrival Time: {self.arrival_time}\n"
                f"Available Seats: {self.__available_seats}")

    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            print(f"Seat booked successfully. Seats left:
{self.__available_seats}")
        else:
            print("No seats available.")

    def cancel_seat(self):
        self.__available_seats += 1
        print(f"Seat cancellation successful. Seats left:
{self.__available_seats}")

    def get_available_seats(self):
        return self.__available_seats



if __name__ == "__main__":

    flight1 = Flight("AI101", "JFK", "LAX", "2024-09-01 08:00", "2024-
09-01 11:00", 10)


    print(flight1.get_flight_details())


    flight1.book_seat()


    flight1.cancel_seat()
```

```
    print(f"Available seats: {flight1.get_available_seats()}")
```

```
Flight Number: AI101
Departure Airport: JFK
Arrival Airport: LAX
Departure Time: 2024-09-01 08:00
Arrival Time: 2024-09-01 11:00
Available Seats: 10
Seat booked successfully. Seats left: 9
Seat cancellation successful. Seats left: 10
Available seats: 10
```

```
# 10b


class Seat:
    def __init__(self, seat_number):
        self.seat_number = seat_number
        self.is_booked = False

    def book(self):
        if not self.is_booked:
            self.is_booked = True
            print(f"Seat {self.seat_number} successfully booked.")
        else:
            print(f"Seat {self.seat_number} is already booked.")

    def cancel(self):
        if self.is_booked:
            self.is_booked = False
            print(f"Reservation for seat {self.seat_number}
successfully canceled.")
        else:
            print(f"Seat {self.seat_number} is not booked.")

    def __str__(self):
        return f"Seat {self.seat_number}: {'Booked' if self.is_booked
else 'Available'}"



class Airplane:
    def __init__(self, total_seats):
        self.seats = [Seat(i+1) for i in range(total_seats)]

    def book_seat(self, seat_number):
        if 1 <= seat_number <= len(self.seats):
            self.seats[seat_number - 1].book()
        else:
```

```
            print(f"Seat {seat_number} does not exist.")

    def cancel_reservation(self, seat_number):
        if 1 <= seat_number <= len(self.seats):
            self.seats[seat_number - 1].cancel()
        else:
            print(f"Seat {seat_number} does not exist.")

    def get_available_seats(self):
        available_seats = [seat for seat in self.seats if not
seat.is_booked]
        print(f"Available seats: {[seat.seat_number for seat in
available_seats]}")
        return available_seats



if __name__ == "__main__":

    airplane = Airplane(10)


    airplane.book_seat(3)
    airplane.book_seat(5)
    airplane.book_seat(7)


    airplane.book_seat(3)


    airplane.get_available_seats()


    airplane.cancel_reservation(5)


    airplane.cancel_reservation(8)


    airplane.get_available_seats()
```

```
Seat 3 successfully booked.
Seat 5 successfully booked.
Seat 7 successfully booked.
Seat 3 is already booked.
Available seats: [1, 2, 4, 6, 8, 9, 10]
Reservation for seat 5 successfully canceled.
Seat 8 is not booked.
Available seats: [1, 2, 4, 5, 6, 8, 9, 10]
```

```python
# 10c

class Flight:
    def __init__(self, flight_number, origin, destination, duration):
        self.__flight_number = flight_number
        self.origin = origin
        self.destination = destination
        self.duration = duration

    def get_flight_details(self):
        return f"Flight from {self.origin} to {self.destination} takes
{self.duration} hours."

    def get_flight_number(self):
        return self.__flight_number

class Passenger:
    def __init__(self, name, passport_number):
        self.name = name
        self.passport_number = passport_number

    def get_passenger_details(self):
        return f"Passenger: {self.name}, Passport Number:
{self.passport_number}"

class Reservation:
    def __init__(self, flight, passenger, seat_number):
        self.flight = flight
        self.passenger = passenger
        self.seat_number = seat_number

    def get_reservation_details(self):
        flight_info = f"Flight Number:
{self.flight.get_flight_number()} -
{self.flight.get_flight_details()}"
        passenger_info = self.passenger.get_passenger_details()
        return f"{passenger_info}, Seat Number: {self.seat_number},
{flight_info}"

class AirlineReservationSystem:
    def __init__(self):
        self.reservations = []

    def make_reservation(self, flight, passenger, seat_number):
        reservation = Reservation(flight, passenger, seat_number)
        self.reservations.append(reservation)
        print(f"Reservation made for {passenger.name} on flight
{flight.get_flight_number()}.")

    def display_reservations(self):
```

```
        print("Reservations List:")
        for reservation in self.reservations:
            print(reservation.get_reservation_details())

if __name__ == "__main__":

    flight1 = Flight("AA123", "New York", "Los Angeles", 6)
    flight2 = Flight("BA456", "London", "Paris", 2)


    passenger1 = Passenger("John Doe", "A12345678")
    passenger2 = Passenger("Jane Smith", "B98765432")


    reservation_system = AirlineReservationSystem()


    reservation_system.make_reservation(flight1, passenger1, "12A")
    reservation_system.make_reservation(flight2, passenger2, "7B")


    reservation_system.display_reservations()
```

```
Reservation made for John Doe on flight AA123.
Reservation made for Jane Smith on flight BA456.
Reservations List:
Passenger: John Doe, Passport Number: A12345678, Seat Number: 12A,
Flight Number: AA123 - Flight from New York to Los Angeles takes 6
hours.
Passenger: Jane Smith, Passport Number: B98765432, Seat Number: 7B,
Flight Number: BA456 - Flight from London to Paris takes 2 hours.
```

```
# 10d


class Flight:
    def __init__(self, flight_number, origin, destination, duration):
        self.flight_number = flight_number
        self.origin = origin
        self.destination = destination
        self.duration = duration
        self.passenger_list = []

    def add_passenger(self, passenger_name):
        self.passenger_list.append(passenger_name)

    def get_passenger_list(self):
        return self.passenger_list

    def get_flight_details(self):
        return (f"Flight Number: {self.flight_number}\n"
```

```
                f"Origin: {self.origin}\n"
                f"Destination: {self.destination}\n"
                f"Duration: {self.duration} hours\n")

class DomesticFlight(Flight):
    def __init__(self, flight_number, origin, destination, duration,
in_flight_meal=False):
        super().__init__(flight_number, origin, destination, duration)
        self.in_flight_meal = in_flight_meal

    def get_flight_details(self):
        meal_info = "Yes" if self.in_flight_meal else "No"
        return (super().get_flight_details() +
                f"In-Flight Meal: {meal_info}\n"
                f"Type: Domestic\n")


class InternationalFlight(Flight):
    def __init__(self, flight_number, origin, destination, duration,
passport_required=True):
        super().__init__(flight_number, origin, destination, duration)
        self.passport_required = passport_required

    def get_flight_details(self):
        passport_info = "Required" if self.passport_required else "Not
Required"
        return (super().get_flight_details() +
                f"Passport: {passport_info}\n"
                f"Type: International\n")

if __name__ == "__main__":

    domestic_flight = DomesticFlight("AI101", "New York", "Los
Angeles", 5.5, in_flight_meal=True)
    domestic_flight.add_passenger("John Doe")
    domestic_flight.add_passenger("Jane Smith")


    international_flight = InternationalFlight("AI201", "New York",
"London", 7.0)
    international_flight.add_passenger("Alice Johnson")
    international_flight.add_passenger("Bob Lee")


    print(domestic_flight.get_flight_details())
    print("Passengers:", domestic_flight.get_passenger_list())
    print()
    print(international_flight.get_flight_details())
    print("Passengers:", international_flight.get_passenger_list())
```

```
Flight Number: AI101
Origin: New York
Destination: Los Angeles
Duration: 5.5 hours
In-Flight Meal: Yes
Type: Domestic

Passengers: ['John Doe', 'Jane Smith']

Flight Number: AI201
Origin: New York
Destination: London
Duration: 7.0 hours
Passport: Required
Type: International

Passengers: ['Alice Johnson', 'Bob Lee']
```

11.

```python
# constants.py

# Mathematical constants
PI = 3.141592653589793  # Value of pi

# Physical constants
SPEED_OF_LIGHT = 299792458
GRAVITATIONAL_CONSTANT = 6.67430e-11
PLANCK_CONSTANT = 6.62607015e-34
AVOGADRO_CONSTANT = 6.02214076e23
BOLTZMANN_CONSTANT = 1.380649e-23
```

```python
# 12

# calculator.py

def add(a, b):
    """Return the sum of two numbers."""
    return a + b

def subtract(a, b):
    """Return the difference of two numbers."""
    return a - b

def multiply(a, b):
    """Return the product of two numbers."""
    return a * b
```

```
def divide(a, b):
    """Return the quotient of two numbers. Raises an error if dividing
by zero."""
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a / b




# ecommerce/product_management.py

class Product:
    def __init__(self, name, price, stock):
        self.name = name
        self.price = price
        self.stock = stock

    def update_stock(self, quantity):
        """Update the stock of the product."""
        self.stock += quantity

    def is_available(self):
        """Check if the product is in stock."""
        return self.stock > 0

    def __str__(self):
        return f"Product: {self.name}, Price: {self.price}, Stock:
{self.stock}"



# ecommerce/order_processing.py

class Order:
    def __init__(self, product, quantity):
        self.product = product
        self.quantity = quantity
        self.status = "Pending"

    def process_order(self):
        """Process the order if the product is available."""
        if self.product.is_available() and self.product.stock >=
self.quantity:
            self.product.update_stock(-self.quantity)
            self.status = "Completed"
        else:
            self.status = "Failed: Insufficient stock"

    def __str__(self):
```

```
        return f"Order: {self.product.name}, Quantity:
{self.quantity}, Status: {self.status}"



# 16.


def write_employee_details(filename):
    with open(filename, 'w') as file:
        while True:

            name = input("Enter employee name (or type 'exit' to
finish): ")
            if name.lower() == 'exit':
                break
            age = input("Enter employee age: ")
            salary = input("Enter employee salary: ")


            file.write(f"Name: {name}, Age: {age}, Salary: {salary}\
n")


filename = 'employees.txt'


write_employee_details(filename)

print(f"Employee details have been written to {filename}.")
```

```
Enter employee name (or type 'exit' to finish):  jatin
Enter employee age:  22
Enter employee salary:  2000000
Enter employee name (or type 'exit' to finish):  exit

Employee details have been written to employees.txt.
```

```
# 17

# Function to read and display the contents of a text file
def read_inventory(filename):
    try:
        with open(filename, 'r') as file:  # Open the file in read
mode
            for line in file:  # Iterate through each line in the file
                print(line.strip())  # Print each line, removing
leading/trailing whitespace
    except FileNotFoundError:
        print(f"The file '{filename}' does not exist.")
```

```
    except Exception as e:
        print(f"An error occurred: {e}")

# Specify the filename
filename = 'inventory.txt'

# Call the function to read the inventory file
read_inventory(filename)
```

```
The file 'inventory.txt' does not exist.
```

```
# 18

def calculate_total_expenses(filename):
    total_amount = 0.0

    try:
        with open(filename, 'r') as file:
            for line in file:

                if ':' in line:
                    item, amount = line.split(':', 1)
                    amount = amount.strip()

                    try:
                        total_amount += float(amount)
                    except ValueError:
                        print(f"Invalid amount found for
{item.strip()}: {amount}. Skipping this line.")

    except FileNotFoundError:
        print(f"The file '{filename}' does not exist.")
    except Exception as e:
        print(f"An error occurred: {e}")

    return total_amount


filename = 'expense.txt'


total_expenses = calculate_total_expenses(filename)
print(f"Total amount spent on expenses: ${total_expenses:.2f}")
```

```
The file 'expense.txt' does not exist.
Total amount spent on expenses: $0.00
```

```
# 32.
```

```
'''
z=(x-mean)/ std deviation

a) Proportion of students scoring more than 55 marks:

z for 55:

 z=(55-49)/6=1

 using z-table , the area to the left of z =1 approx 0.8413

 propotion scoring more than 55:

 1-0.8413 = 0.1587

 b) Proportion of students scoring more than 70 marks:

 z for 70 :

 z=(70-49)/6 = 3.5

 =0.9998

 1-0.9998 = 0.0002'''
```

```
'\nz=(x-mean)/ std deviation\n\na) Proportion of students scoring more than 55 marks:\n\nz for 55:\n\n z=(55-49)/6=1\n\n using z-table , the area to the left of z =1 approx 0.8413\n\n propotion scoring more than 55:\n\n 1-0.8413 = 0.1587 \n\n b) Proportion of students scoring more than 70 marks:\n\n z for 70 :\n\n z=(70-49)/6 = 3.5 \n\n =0.9998\n\n 1-0.9998 = 0.0002'
```

```
# 33

'''
mean=65
std deviation = 5
total students =500

a) Students with height greater than 70 inches:

Z for 70:

z=(70-65)/5 =1

 Z = 1 is approximately 0.8413.

 Proportion greater than 70:
1−0.8413=0.1587

Number of students:
```

```
0.1587×500≈79.35⟹Approximately 79 students.



(b) Students with height between 60 and 70 inches:
Z for 60:

z=(60-65)/5=-1

Z = -1 is approximately 0.1587.

Area between 60 and 70:
0.8413−0.1587=0.6826

Number of students:
0.6826×500≈341.3⟹Approximately 341 students.'''

'\nmean=65\nstd deviation = 5\ntotal students =500\n\na) Students with height greater than 70 inches:\n\nZ for 70:\n\nz=(70-65)/5 =1\n\n Z = 1 is approximately 0.8413.\n\n Proportion greater than 70:\n1−0.8413=0.1587 \n\nNumber of students:\n0.1587×500≈79.35⟹Approximately\xa079\xa0students.\n\n\n\n(b) Students with height between 60 and 70 inches:\nZ for 60:\n\nz=(60-65)/5=-1\n\nZ = -1 is approximately 0.1587.\n\nArea between 60 and 70:\n0.8413−0.1587=0.6826\n\nNumber of students:\n0.6826×500≈341.3⟹Approximately\xa0341\xa0students.'

# 34
'''Statistical Hypothesis:
A statistical hypothesis is a specific claim or assumption about a population parameter, such as the mean or proportion. It's tested using sample data to determine whether to reject or fail to reject the hypothesis.

Errors in Hypothesis Testing:
1. Type I Error:Rejecting the null hypothesis when it is actually true (false positive).
2. Type II Error Failing to reject the null hypothesis when it is actually false (false negative).

Sample:
A sample is a subset of individuals or observations selected from a larger population for the purpose of analysis.

- Large Samples:** Typically, a sample size greater than 30 is considered large. Large samples provide more reliable estimates and reduce the margin of error.
- Small Samples:** Samples with a size of 30 or less. They are more sensitive to variability and require different statistical methods, such as the t-test instead of the z-test, for analysis.'''
```

```
"**Statistical Hypothesis:**\nA statistical hypothesis is a specific claim or assumption about a population parameter, such as the mean or proportion. It's tested using sample data to determine whether to reject or fail to reject the hypothesis.\n\n**Errors in Hypothesis Testing:**\n1. **Type I Error:** Rejecting the null hypothesis when it is actually true (false positive).\n2. **Type II Error:** Failing to reject the null hypothesis when it is actually false (false negative).\n\n**Sample:**\nA sample is a subset of individuals or observations selected from a larger population for the purpose of analysis.\n\n- **Large Samples:** Typically, a sample size greater than 30 is considered large. Large samples provide more reliable estimates and reduce the margin of error.\n  \n- **Small Samples:** Samples with a size of 30 or less. They are more sensitive to variability and require different statistical methods, such as the t-test instead of the z-test, for analysis."

# 37.
'''Statistical Hypothesis:
A claim about a population parameter, tested using sample data.

Errs in Hypothesis Testing:
- Type I Error:False positive.
- Type II Error False negative.

Sample:
A subset of a population used for analysis.

- Large Sample: Size > 30.
- Small Sample: Size ≤ 30.'''

'Statistical Hypothesis:\nA claim about a population parameter, tested using sample data.\n\n**Errors in Hypothesis Testing:**  \n- **Type I Error:** False positive.\n- **Type II Error:** False negative.\n\n**Sample:**  \nA subset of a population used for analysis.\n\n- **Large Sample:** Size > 30.\n- **Small Sample:** Size ≤ 30.'

# 39
```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, World!"

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

```
  Cell In[1], line 2
    ```python
    ^
SyntaxError: invalid syntax


# 40

import Flask
import request
   - Define a route with `methods=['GET', 'POST']`.

2. Create HTML Form:
   - Use an HTML form with `method="POST"`.

3. Handle Form Submission:
   - In the route, check `if request.method == 'POST'` and access form
data using `request.form`.

Example:
```python
from flask import Flask, request

app = Flask(__name__)

@app.route('/submit', methods=['GET', 'POST'])
def submit():
    if request.method == 'POST':
        data = request.form['field_name']
        return f"Received: {data}"
    return '''
        <form method="POST">
            <input name="field_name">
            <input type="submit">
        </form>
    '''

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

  Cell In[7], line 5
    - Define a route with `methods=['GET', 'POST']`.
    ^
IndentationError: unexpected indent

41
```
```

```
'''from flask import Flask

app = Flask(__name__)

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

if __name__ == '__main__':
    app.run()
'''

'from flask import Flask\n\napp =
Flask(__name__)\n\n@app.route(\'/greet/<name>\')\ndef greet(name):\n
return f"Hello, {name}!"\n\nif __name__ == \'__main__\':\n
app.run()\n'

# 43

'''
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

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

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

if __name__ == '__main__':
    db.create_all()
    app.run()'''

"\nfrom flask import Flask\nfrom flask_sqlalchemy import SQLAlchemy\n\
napp = Flask(__name__)\napp.config['SQLALCHEMY_DATABASE_URI'] =
'sqlite:///your_database.db'\ndb = SQLAlchemy(app)\n\nclass
User(db.Model):\n    id = db.Column(db.Integer, primary_key=True)\n
name = db.Column(db.String(80), nullable=False)\n\nif __name__ ==
'__main__':\n    db.create_all()\n    app.run()"

# 44.How would you create a RESTful API endpoint in Flask that returns
JSON data?

# ans
'''

from flask import Flask, jsonify

app = Flask(__name__)
```

```
@app.route('/api/data', methods=['GET'])
def get_data():
    data = {"key": "value", "number": 123}
    return jsonify(data)

if __name__ == '__main__':
    app.run()
'''

'\n\nfrom flask import Flask, jsonify\n\napp = Flask(__name__)\n\
n@app.route(\'/api/data\', methods=[\'GET\'])\ndef get_data():\n
data = {"key": "value", "number": 123}\n    return jsonify(data)\n\nif
__name__ == \'__main__\':\n    app.run()\n'

# 45
'''
from flask import Flask, render_template, request
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'

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

@app.route('/form', methods=['GET', 'POST'])
def form():
    form = MyForm()
    if form.validate_on_submit():
        return f"Hello, {form.name.data}!"
    return render_template('form.html', form=form)

if __name__ == '__main__':
    app.run()
'''

'\nfrom flask import Flask, render_template, request\nfrom flask_wtf
import FlaskForm\nfrom wtforms import StringField, SubmitField\nfrom
wtforms.validators import DataRequired\n\napp = Flask(__name__)\
napp.config[\'SECRET_KEY\'] = \'your_secret_key\'\n\nclass
MyForm(FlaskForm):\n    name = StringField(\'Name\',
validators=[DataRequired()])\n    submit = SubmitField(\'Submit\')\n\
n@app.route(\'/form\', methods=[\'GET\', \'POST\'])\ndef form():\n
form = MyForm()\n    if form.validate_on_submit():\n        return
f"Hello, {form.name.data}!"\n    return render_template(\'form.html\',
form=form)\n\nif __name__ == \'__main__\':\n    app.run()\n'
```

```
# 46

'''
from flask import Flask, request, redirect, url_for

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads/'

@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        file = request.files['file']
        if file:
            file.save(f"{app.config['UPLOAD_FOLDER']}{file.filename}")
            return 'File uploaded successfully!'
    return '''
        <form method="POST" enctype="multipart/form-data">
            <input type="file" name="file">
            <input type="submit">
        </form>
    '''

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

```
  File <string>:21
    '''
       ^
IndentationError: unindent does not match any outer indentation level
```

```
# 47
'''


1.
Import Blueprint:**

Import `Blueprint` from `flask`.
2.
Create a Blueprint:

Define a blueprint: `bp = Blueprint('name', __name__)`.
3.
Define Routes:
   - Use the blueprint to define routes: `@bp.route('/path')`.

4. Register the Blueprint:
   - Register the blueprint in the main app file:
```

```
`app.register_blueprint(bp)`.
'''

"\n**Steps to Create a Flask Blueprint:**\n\n1. \nImport Blueprint:**\n\nImport `Blueprint` from `flask`.\n2. \nCreate a Blueprint:\n\nDefine a blueprint: `bp = Blueprint('name', __name__)`.\n3. \nDefine Routes:\n   - Use the blueprint to define routes: `@bp.route('/path')`.\n\n4. Register the Blueprint:\n   - Register the blueprint in the main app file: `app.register_blueprint(bp)`.\n"

# 48
'''
server {
    listen 80;
    server_name your_domain.com;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
'''

'\nserver {\n    listen 80;\n    server_name your_domain.com;\n\n location / {\n        proxy_pass http://127.0.0.1:8000;\n proxy_set_header Host $host;\n        proxy_set_header X-Real-IP $remote_addr;\n        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;\n        proxy_set_header X-Forwarded-Proto $scheme;\n    }\n}\n'

# 49

'''
from flask import Flask, render_template, request, redirect, url_for, session, flash
from flask_pymongo import PyMongo
from werkzeug.security import generate_password_hash, check_password_hash

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['MONGO_URI'] = 'mongodb://localhost:27017/user_db'

mongo = PyMongo(app)

@app.route('/')
def index():
    if 'username' in session:
```

```
        return f"Hello Geeks! Welcome, {session['username']}!"
    return redirect(url_for('signin'))

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

        # Check if user already exists
        if mongo.db.users.find_one({'username': username}):
            flash('Username already exists!')
            return redirect(url_for('signup'))

        mongo.db.users.insert_one({'username': username, 'password':
hashed_password})
        flash('Signup successful! Please log in.')
        return redirect(url_for('signin'))

    return render_template('signup.html')

@app.route('/signin', methods=['GET', 'POST'])
def signin():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = mongo.db.users.find_one({'username': username})

        if user and check_password_hash(user['password'], password):
            session['username'] = username
            return redirect(url_for('index'))
        else:
            flash('Invalid username or password!')

    return render_template('signin.html')

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('signin'))

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



<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Signup</title>
</head>
<body>
    <h2>Signup</h2>
    <form method="POST">
        <input type="text" name="username" placeholder="Username"
required>
        <input type="password" name="password" placeholder="Password"
required>
        <button type="submit">Signup</button>
    </form>
    <a href="{{ url_for('signin') }}">Already have an account? Sign
in</a>
</body>
</html>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Signin</title>
</head>
<body>
    <h2>Signin</h2>
    <form method="POST">
        <input type="text" name="username" placeholder="Username"
required>
        <input type="password" name="password" placeholder="Password"
required>
        <button type="submit">Signin</button>
    </form>
    <a href={{ url_for('signup') }}>"Don't have an account"?
Signup</a>
</body>
</html>


python app.py

'''


'\nfrom flask import Flask, render_template, request, redirect,
url_for, session, flash\nfrom flask_pymongo import PyMongo\nfrom
werkzeug.security import generate_password_hash, check_password_hash\
```

```
n\napp = Flask(__name__)\napp.config[\'SECRET_KEY\']
= \'your_secret_key\'  \napp.config[\'MONGO_URI\'] =
\'mongodb://localhost:27017/user_db\'  \n\nmongo = PyMongo(app)\n\
n@app.route(\'/\')\ndef index():\n    if \'username\' in session:\n
return f"Hello Geeks! Welcome, {session[\'username\']}!"\n    return
redirect(url_for(\'signin\'))\n\n@app.route(\'/signup\',
methods=[\'GET\', \'POST\'])\ndef signup():\n    if request.method
== \'POST\':\n        username = request.form[\'username\']\n
password = request.form[\'password\']\n        hashed_password =
generate_password_hash(password)\n\n        # Check if user already
exists\n        if mongo.db.users.find_one({\'username\': username}):\
n            flash(\'Username already exists!\')\n            return
redirect(url_for(\'signup\'))\n\n
mongo.db.users.insert_one({\'username\': username, \'password\':
hashed_password})\n        flash(\'Signup successful! Please log
in.\')\n        return redirect(url_for(\'signin\'))\n    \n    return
render_template(\'signup.html\')\n\n@app.route(\'/signin\',
methods=[\'GET\', \'POST\'])\ndef signin():\n    if request.method
== \'POST\':\n        username = request.form[\'username\']\n
password = request.form[\'password\']\n        user =
mongo.db.users.find_one({\'username\': username})\n\n        if user
and check_password_hash(user[\'password\'], password):\n
session[\'username\'] = username\n            return
redirect(url_for(\'index\'))\n        else:\n
flash(\'Invalid username or password!\')\n\n    return
render_template(\'signin.html\')\n\n@app.route(\'/logout\')\ndef
logout():\n    session.pop(\'username\', None)\n    return
redirect(url_for(\'signin\'))\n\nif __name__ == \'__main__\':\n
app.run(debug=True)\n\n\n\n<!DOCTYPE html>\n<html lang="en">\n<head>\n
<meta charset="UTF-8">\n    <meta name="viewport"
content="width=device-width, initial-scale=1.0">\n
<title>Signup</title>\n</head>\n<body>\n    <h2>Signup</h2>\n    <form
method="POST">\n        <input type="text" name="username"
placeholder="Username" required>\n        <input type="password"
name="password" placeholder="Password" required>\n        <button
type="submit">Signup</button>\n    </form>\n    <a
href="{{ url_for(\'signin\') }}">Already have an account? Sign in</a>\
n</body>\n</html>\n\n<!DOCTYPE html>\n<html lang="en">\n<head>\n
<meta charset="UTF-8">\n    <meta name="viewport"
content="width=device-width, initial-scale=1.0">\n
<title>Signin</title>\n</head>\n<body>\n    <h2>Signin</h2>\n    <form
method="POST">\n        <input type="text" name="username"
placeholder="Username" required>\n        <input type="password"
name="password" placeholder="Password" required>\n        <button
type="submit">Signin</button>\n    </form>\n    <a
href={{ url_for(\'signup\') }}>"Don\'t have an account"? Signup</a>\
n</body>\n</html>\n\n\npython app.py\n\n'
```

```
# 2I Do the EDA on the given dataset: Lung cancer, and extract some
useful information from this.

'''the dataset contains 3,000 entries and 16 columns, which include
various features such as GENDER, AGE, SMOKING, and the target variable
LUNG_CANCER. Here's a brief overview of the columns:

GENDER: Gender of the individual (M/F).
AGE: Age of the individual.
SMOKING, YELLOW_FINGERS, ANXIETY, etc.: Various features indicating
health behaviors and symptoms.
LUNG_CANCER: Target variable indicating whether the individual has
lung cancer (YES/NO).
Next, let's perform some exploratory data analysis (EDA) to extract
useful insights from this dataset:

Basic Descriptive Statistics: Summary statistics of the numeric
features.
Correlation Analysis: Identify relationships between the features and
the target variable.
Distribution of the Target Variable: How many individuals have lung
cancer.
Distribution by Gender: Analysis of lung cancer cases by gender.
Age Distribution: Age distribution of individuals with and without
lung cancer.
I'll start with these analyses.

Summary of Exploratory Data Analysis (EDA)
Basic Descriptive Statistics:

The dataset contains individuals aged between 30 and 80, with a mean
age of approximately 55 years.
The features SMOKING, YELLOW_FINGERS, ANXIETY, etc., are binary or
categorical, with values predominantly ranging between 1 and 2.
Distribution of the Target Variable (LUNG_CANCER):

There are 1,537 individuals without lung cancer and 1,463 individuals
with lung cancer in the dataset, indicating a relatively balanced
dataset.
Distribution by Gender:

The gender distribution for lung cancer cases is relatively balanced:
Males: 758 with lung cancer, 729 without.
Females: 705 with lung cancer, 808 without.
Age Distribution by Lung Cancer Status:

Individuals with lung cancer tend to be slightly older on average
(mean age ~58) compared to those without lung cancer (mean age ~52).
Correlation Analysis:

The correlation between features is generally low, with the highest
```

```
correlations being around 0.3.
Some weak positive correlations exist between certain features, such
as SMOKING and YELLOW_FINGERS, which might be expected.'''



# 3.

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


df = pd.read_csv('election2024.csv')

# Basic information about the dataset
print("Basic Information:")
print(df.info())
print("\nFirst 5 rows of the dataset:")
print(df.head())

# Statistical summary
print("\nStatistical Summary:")
print(df.describe())

# Checking for missing values
print("\nMissing Values:")
print(df.isnull().sum())

# Visualizing the distribution of poll results by candidate (assuming
a column like 'Candidate' exists)
if 'Candidate' in df.columns and 'Poll_Percentage' in df.columns:
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=df, x='Candidate', y='Poll_Percentage')
    plt.title('Distribution of Poll Percentages by Candidate')
    plt.xticks(rotation=90)
    plt.show()

# Visualizing the trend of polling over time (assuming a column like
'Date' exists)
if 'Date' in df.columns and 'Poll_Percentage' in df.columns:
    df['Date'] = pd.to_datetime(df['Date'])
    plt.figure(figsize=(12, 6))
    sns.lineplot(data=df, x='Date', y='Poll_Percentage',
hue='Candidate', marker='o')
    plt.title('Polling Trend Over Time')
    plt.xticks(rotation=45)
    plt.show()

# Checking data types
print("Data Types in the Dataset:")
```

```
print(df.dtypes)

# Converting all columns to numeric where applicable
df_numeric = df.apply(pd.to_numeric, errors='coerce')

# Dropping columns that are still non-numeric or where all values are
NaN
df_numeric = df_numeric.dropna(axis=1, how='all')

# Now calculate the correlation matrix
if df_numeric.empty:
    print("No numerical data available after conversion and
cleaning.")
else:
    correlation_matrix = df_numeric.corr()

    if correlation_matrix.empty:
        print("Correlation matrix is empty. No numerical features to
correlate.")
    else:
        # Visualizing correlations
        plt.figure(figsize=(12, 8))
        sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
        plt.title('Correlation Matrix')
        plt.show()

# Handling missing data (if any)
df_cleaned = df.dropna()  # or use df.fillna() for imputation
print("\nAfter Handling Missing Data:")
print(df_cleaned.info())

# Further analysis can include analyzing specific features, performing
feature engineering, etc.
```

```
Basic Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600 entries, 0 to 1599
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   id                   1600 non-null   object
 1   sex                  1600 non-null   object
 2   age                  1600 non-null   float64
 3   federal_district     1600 non-null   object
 4   type_of_city         1600 non-null   object
 5   knows_election_date  1600 non-null   object
 6   will_vote            1600 non-null   object
```

```
 7   candidate             1600 non-null   object
 8   television_usage      1600 non-null   object
 9   internet_usage        1600 non-null   object
 10  education             1600 non-null   object
 11  income                1600 non-null   object
 12  employment            1600 non-null   object
 13  job_type              692 non-null    object
 14  company_type          879 non-null    object
 15  weight1               1600 non-null   float64
dtypes: float64(2), object(14)
memory usage: 200.1+ KB
None

First 5 rows of the dataset:
                 id   sex   age federal_district  \
0  07169ed8148ce047  male  18.0  north caucasian
1  0716a4f3354cecdd  male  23.0  north caucasian
2  0716889b304ce79c  male  20.0            volga
3  07168e28b5cce563  male  22.0     northwestern
4  0716a563914ce549  male  21.0         southern

                                    type_of_city    knows_election_date
\
0                                        village     named correct date

1                                        village     named correct date

2         city with population of less than 50k     named correct date

3  city with population of 1 million and higher  not sure or no answer

4  city with population of 1 million and higher     named correct date


        will_vote candidate      television_usage      internet_usage
\
0        not sure     Putin  several times a week  over 4 hours a day

1        not sure     Putin      once half a year  over 4 hours a day

2  definitely yes     Putin  several times a week  over 4 hours a day

3        not sure  Davankov  several times a week  over 4 hours a day

4  definitely yes     Putin        does not watch  over 4 hours a day


                     education     income        employment  \
0  incomplete school education  very high      entrepreneur
1                    college  very high     work for hire
2                    college  very high     work for hire
```

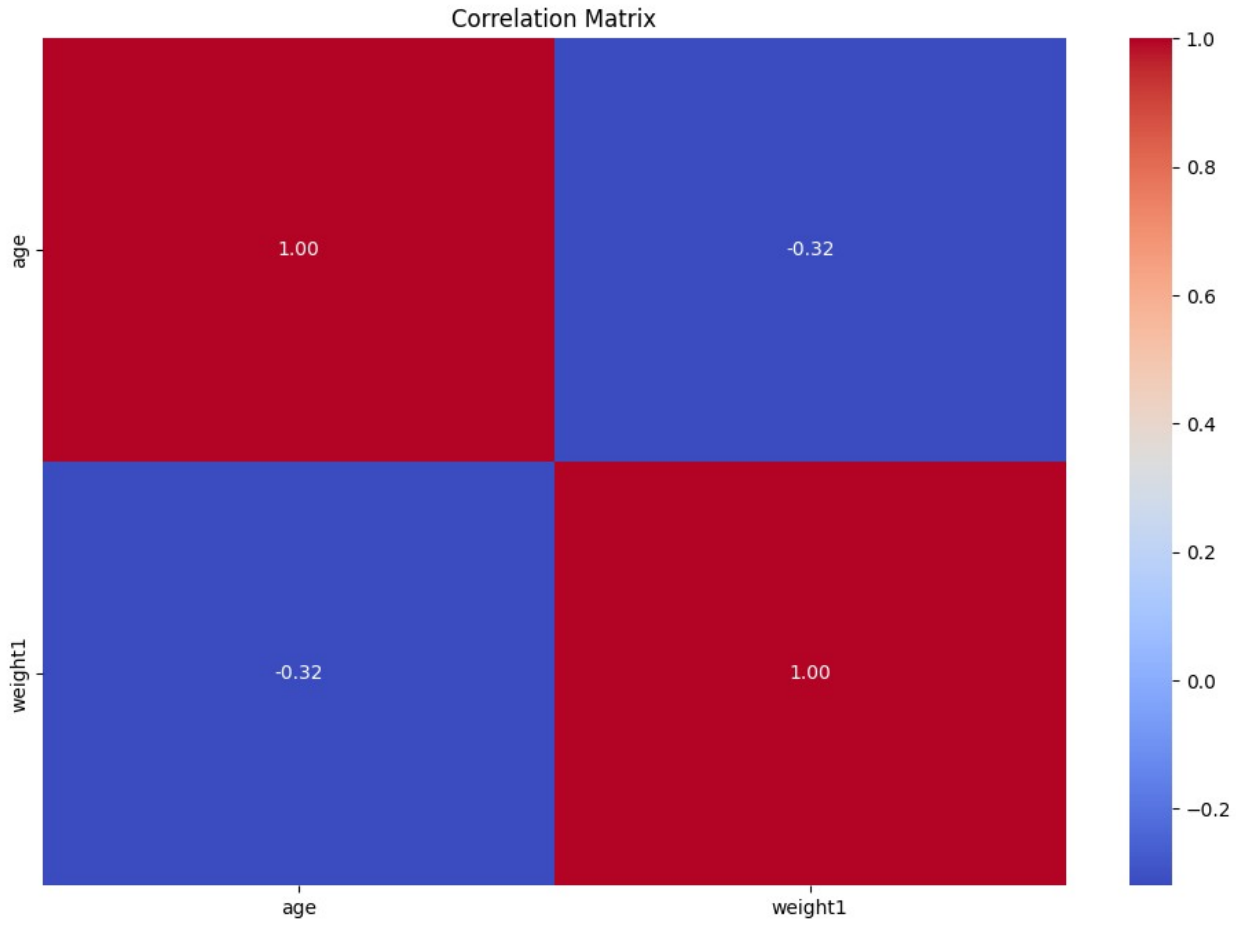
```
3                       college  very high        unemployed
4               bachelor degree  very high  employed student

                    job_type                         company_type
weight1
0                       NaN                              farming
1.445172
1  commercial organization                                trade
1.445172
2   law enforcement agency              law enforcement agency
1.301691
3                       NaN                                NaN
1.538628
4  commercial organization  tech, programming, communications
1.967015

Statistical Summary:
                age      weight1
count  1600.000000  1600.000000
mean     49.936250     1.000000
std      16.901797     0.327084
min      18.000000     0.468226
25%      37.000000     0.772224
50%      49.000000     0.921724
75%      64.000000     1.158913
max      90.000000     2.515072

Missing Values:
id                        0
sex                       0
age                       0
federal_district          0
type_of_city              0
knows_election_date       0
will_vote                 0
candidate                 0
television_usage          0
internet_usage            0
education                 0
income                    0
employment                0
job_type                908
company_type            721
weight1                   0
dtype: int64
Data Types in the Dataset:
id                       object
sex                      object
age                     float64
federal_district         object
```

```
type_of_city             object
knows_election_date      object
will_vote                object
candidate                object
television_usage         object
internet_usage           object
education                object
income                   object
employment               object
job_type                 object
company_type             object
weight1                 float64
dtype: object
```




```
After Handling Missing Data:
<class 'pandas.core.frame.DataFrame'>
Index: 670 entries, 1 to 1598
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
```

```
 0   id                    670 non-null    object
 1   sex                   670 non-null    object
 2   age                   670 non-null    float64
 3   federal_district      670 non-null    object
 4   type_of_city          670 non-null    object
 5   knows_election_date   670 non-null    object
 6   will_vote             670 non-null    object
 7   candidate             670 non-null    object
 8   television_usage      670 non-null    object
 9   internet_usage        670 non-null    object
 10  education             670 non-null    object
 11  income                670 non-null    object
 12  employment            670 non-null    object
 13  job_type              670 non-null    object
 14  company_type          670 non-null    object
 15  weight1               670 non-null    float64
dtypes: float64(2), object(14)
memory usage: 89.0+ KB
None

# 3rd question continue ...


'''
Here are some key insights from the dataset:

Age Distribution:

The respondents' ages range from 18 to 90 years old, with a mean age
of about 50 years.
25% of the respondents are younger than 37, and 25% are older than 64.
Gender Distribution:

The sample is slightly skewed towards females, who make up
approximately 52.7% of the respondents, while males make up 47.3%.
Voting Intention:

A majority of respondents (66.4%) indicated that they will
"definitely" vote.
13.6% are "likely" to vote, while 10.4% are "not sure."
Candidate Preference:

The majority of respondents (70.5%) indicated that they would vote for
Putin.
10.3% are undecided ("struggle to answer"), while 6% said they would
not participate in the election.
Education Level:

The largest group of respondents (43.1%) has a college education.
34.4% have a bachelor's degree, and 11.3% have completed school
```

```
education
'''



                                        '''THEORTICAL QUESTIONS '''
```

'THEORTICAL QUESTIONS '

```
'''
1.1
a)
In Python, static variables (often called class variables) are shared
across all instances of a
class, while dynamic variables (typically instance variables) are
unique to each instance and
can change based on the object's state.
b)
- `pop(key)`: Removes and returns the value associated with the
specified key.
Example:* `my_dict.pop('a')` removes the key ``a'' and returns its
value.
- `popitem()`: Removes and returns the last inserted key-value pair as
a tuple.
Example: `my_dict.popitem()` removes and returns the last item in the
dictionary.
- `clear()`: Removes all items from the dictionary, leaving it empty.
*Example:* `my_dict.clear()` empties the dictionary.
c.
A `frozenset` is an immutable version of a Python set, meaning its
elements cannot be
changed, added, or removed after creation. It's useful for creating
sets that need to remain
constant.
d.
Mutable data types can be changed after creation, while **immutable
data types** cannot be
altered.
- Mutable examples: `list`, `dict`, `set`
- Immutable examples: `int`, `float`, `tuple`, `str`, `frozenset`
e.
__init__ is a special method in Python used to initialize objects when
they are created. It sets
the initial state of the object
'''
```

```
'''
f.
A docstring in Python is a string literal used to document a module,
class, method, or
function. It provides a description of what the code does.
G.
Unit tests in Python are tests that validate the functionality of
small, isolated pieces of code
(usually functions or methods) to ensure they work as expected. They
are typically written
using the `unittest` module.
h.
- `break`: Exits the nearest loop prematurely.
- `continue`: Skips the current iteration and proceeds to the next
loop iteration.
- `pass` Acts as a placeholder; does nothing when executed.
i.
In Python, `self` refers to the instance of the class and is used to
access instance variables
and methods from within the class.
j.
- Global attributes: Variables accessible throughout the entire
module.
- Protected attributes: Variables prefixed with a single underscore
(`_`), indicating they
should not be accessed outside the class or its subclasses.
- Private attributes: Variables prefixed with double underscores
(`__`), intended to be
inaccessible outside the class they are defined in.
k.
Modules are individual files containing Python code (functions,
classes, variables) that can
be imported and used in other Python scripts.
Packages are collections of related modules organized in directories,
allowing for a
hierarchical structuring of modules. Each package contains an
`__init__.py` file'''


'''
l
Lists are mutable collections that can be modified (elements can be
added, removed, or
changed). Tuples are immutable collections that cannot be modified
after creation.
Key difference: Lists use square brackets `[]`, while tuples use
parentheses `()`.
m.
An interpreted language is executed line-by-line by an interpreter at
```

```
runtime, while a
**dynamically typed language** determines variable types at runtime
rather than at compile
time.
Differences:
1. Execution: Interpreted languages are executed by an interpreter;
dynamically typed
languages check types during execution.
2. Compilation**: Interpreted languages do not require compilation;
dynamically typed
languages can be compiled but check types at runtime.
3. Error Detection**: Interpreted languages can show errors during
execution; dynamically
typed languages can only show type-related errors when the code is
run.
4. Performance: Interpreted languages may be slower due to line-by-
line execution;
dynamically typed languages can be optimized at compile time.
5. Flexibility Dynamically typed languages allow variables to change
types; interpreted
languages focus on immediate execution without compilation.
n.
List comprehensions** are concise ways to create lists using a single
line of code, typically
involving a for loop and an optional condition.
Dict comprehensions** are similar but create dictionaries, using key-
value pairs.
Examples:
- List comprehension**: `[x**2 for x in range(5)]` creates a list of
squares.
- **Dict comprehension**: `{x: x**2 for x in range(5)}` creates a
dictionary of numbers and
their squares.
'''


'''
Decorators in Python are functions that modify or enhance the behavior
of other functions
or methods without changing their code. They are typically used to add
functionality, such as
logging, access control, or timing.
p.
Memory in Python is managed through automatic garbage collection,
which tracks and frees
up memory that is no longer in use. Python uses reference counting to
keep track of the
number of references to objects, and when an object's reference count
reaches zero, it is
```

```
deallocated. Additionally, a cyclic garbage collector handles circular
references.
q.
A lambda in Python is an anonymous function defined using the `lambda`
keyword. It can
take any number of arguments but can only have one expression.
Usage: Lambdas are used for creating small, throwaway functions, often
for short-term use
in higher-order functions like `map()`, `filter()`, or `sorted()`.
r.
- `split(sep)`: This method splits a string into a list of substrings
based on a specified
separator (`sep`). If no separator is provided, it splits by
whitespace.
Example: `"Hello World".split() # Output: ['Hello', 'World']`
- `join(iterable)`: This method concatenates the elements of an
iterable (like a list) into a
single string, using the string on which it is called as a separator.
Example: `" ".join(['Hello', 'World']) # Output: 'Hello World'''


'''
s.
- Iterable: An object that can be looped over, implementing the
`__iter__()` method (e.g.,
lists, strings).

- Iterator: An object that retrieves elements from an iterable one at
a time, implementing the
`__next__()` method.
- Generator: A type of iterator created using a function with the
`yield` keyword, producing
values on-the-fly and allowing for lazy evaluation.
t.
In Python 2, `range()` returns a list of numbers, while **`xrange()`**
returns an iterator that
generates numbers on-the-fly, using less memory for large ranges. In
Python 3,
**`xrange()`** is no longer available, and **`range()`** behaves like
`xrange()` from Python 2,
returning an immutable sequence type.
u.
The pillars of Object-Oriented Programming (OOP) are:
1. Encapsulation: Bundling data and methods that operate on the data
within a single unit
(class) and restricting access to some components.
2. Abstraction: Hiding complex implementation details and exposing
only the necessary
features of an object.
```

```
3. Inheritanc: Allowing a new class to inherit attributes and methods
from an existing class,
promoting code reuse.
4. Polymorphism: Enabling a single interface to represent different
underlying data types,
allowing methods to be used interchangeably across different classes.
v.
class Parent:
pass
class Child(Parent):
pass
print(issubclass(Child, Parent)) # Output: True
'''



'''
w
Inheritance in Python allows a class (child or subclass) to inherit
attributes and methods from
another class (parent or superclass). This promotes code reuse and
establishes a
relationship between classes.
Types of inheritance – single
Multiple
Multilevel
Hierarchial
Hybrid
x.
Encapsulation is a fundamental concept in object-oriented programming
that restricts direct
access to an object's data and methods, bundling them within a class.
This helps protect the
integrity of the object's state and promotes modularity.
y.
Polymorphism in Python allows different classes to be treated as
instances of the same
class through a common interface. It enables methods to be used
interchangeably across
different classes, supporting method overriding and operator
overloading..
20.
Measures of Central Tendency refer to statistical metrics that
describe the center point or
typical value of a dataset. The most common measures are:
1. Mean: The average of all data points.
2. Median: The middle value when data is sorted.
3. Mode: The most frequently occurring value.
Measures of Dispersion indicate the spread or variability within a
dataset. Common
```

```
measures include:
1. Range: The difference between the maximum and minimum values.
2.Variance: The average of the squared differences from the mean
3. Standard Deviation: The square root of the variance, representing
the average distance
from the mean.
Calculations:
- Meaan: \( \text{Mean} = \frac{\sum x}{n} \)
- Median: Sort data and find the middle value (or average the two
middle values if even).
- Mode: Identify the value(s) that appear most frequently.
- Range: \( \text{Range} = \text{Max} - \text{Min} \)
- Variance: \( \text{Variance} = \frac{\sum (x - \text{Mean})^2}{n} \)
- Standard Deviation**: \( \text{SD} = \sqrt{\text{Variance}} \)
These measures help summarize and understand the characteristics of
the data.
'''



'''
21.
Skewness is a statistical measure that describes the asymmetry of the
distribution of values
in a dataset. It indicates whether the data is skewed to the left
(negative skew) or to the right
(positive skew) compared to a normal distribution.
Types of Skewness:
1. Positive Skewness (Right Skew):
- The tail on the right side of the distribution is longer or fatter.
- The majority of the data points are concentrated on the left.
- Example: Income distribution in many populations.
2. Negative Skewness (Left Skew):
- The tail on the left side of the distribution is longer or fatter.
- The majority of the data points are concentrated on the right.
- Example: Age at retirement.
3. Zero Skewness:
- The distribution is symmetric, resembling a normal distribution.
- Example: Heights of individuals in a homogeneous population.
'''



'''
2.
Probability Mass Function (PMF): The PMF is used for discrete random
variables and gives
the probability that a discrete random variable is exactly equal to a
specific value. It is
represented as \( P(X = x) \) and the sum of all probabilities in the
```

```
PMF equals 1.
Probability Density Function (PDF): The PDF is used for continuous
random variables and
describes the likelihood of a random variable falling within a
particular range of values. The
PDF itself does not give probabilities; instead, the probability of a
variable falling within a
certain interval is found by integrating the PDF over that interval.
The total area under the
PDF curve equals 1
'''


'''
23.
Correlation is a statistical measure that describes the strength and
direction of a relationship
between two variables. It quantifies how changes in one variable are
associated with
changes in another.
Types of Correlation:
1. Positive Correlation:
- As one variable increases, the other variable also increases.
- Example: Height and weight.
2. Negative Correlation:
- As one variable increases, the other variable decreases.
- Example: Temperature and heating costs.
3. No Correlation:
- There is no discernible relationship between the two variables.

- Example: Shoe size and intelligence.
Methods of Determining Correlation:
1. Pearson Correlation Coefficient:
- Measures linear correlation between two continuous variables.
- Ranges from -1 to 1; 1 indicates perfect positive correlation, -1
indicates perfect negative
correlation, and 0 indicates no correlation.
2. Spearman's Rank Correlation:
- A non-parametric measure that assesses how well the relationship
between two variables
can be described by a monotonic function.
- Suitable for ordinal data or non-linear relationships.
3. Kendall's Tau:
- Another non-parametric measure of correlation that calculates the
correlation between
two ranked variables.
- Useful for small datasets or when there are many tied ranks.
4. Scatter Plot:
- A graphical method that visually shows the relationship between two
```

```
variables, helping to
identify correlation types.
These methods help in understanding the strength and direction of
relationships between
variables in data analysis.
'''



'''
25.
Here are four key differences between correlation and regression:
1. Purpose
- Correlation measures the strength and direction of a relationship
between two variables.
- Regression estimates the relationship between a dependent variable
and one or more
independent variables to make predictions.
2. Type of Variables
- Correlation is typically used for two variables without
distinguishing between dependent
and independent variables.
- Regression requires a clear distinction, with one dependent variable
and one or more
independent variables.
3. Output:
- Correlation results in a correlation coefficient (e.g., Pearson's r)
that ranges from -1 to 1.
- Regression provides a regression equation (e.g., \(Y = a + bX\))
that predicts the
dependent variable based on the independent variables.
4. Assumptions:
- Correlation assumes a linear relationship but does not require the
variables to be
normally distributed.
- Regression often assumes that the residuals are normally distributed
and homoscedastic
(constant variance).
These differences highlight the distinct roles that correlation and
regression play in statistical
analysis
'''



'''
28.
Normal Distribution: A normal distribution is a continuous probability
distribution that is
symmetric about the mean, depicting the distribution of values in a
```

```
bell-shaped curve. It is
characterized by its mean (μ) and standard deviation (σ).
Four Assumptions of Normal Distribution:
1. Symmetry: The distribution is symmetric around the mean, meaning
the left and right
sides are mirror images.
2. Mean, Median, Mode: In a normal distribution, the mean, median, and
mode are all equal.
3. Asymptotic The tails of the distribution approach, but never touch,
the horizontal axis,
extending indefinitely in both directions.
4. Defined by Mean and Standard Deviation: The shape of the normal
distribution is fully
determined by its mean and standard deviation, with approximately 68%
of data falling within
one standard deviation from the mean
'''



'''
29
Here are the key characteristics or properties of the normal
distribution curve:
1. Bell-Shaped: The curve is symmetric and bell-shaped, centered
around the mean.
2. Mean, Median, Mode: All three measures of central tendency are
equal and located at the
center of the curve.
3. Symmetry: The left and right halves of the curve are mirror images,
meaning the
distribution is symmetrical.
4. Asymptotic: The tails approach the horizontal axis but never touch
it, extending indefinitely
in both directions.
5. Empirical Rule: Approximately 68% of the data falls within one
standard deviation (σ),
95% within two standard deviations, and 99.7% within three standard
deviations from the
mean (μ).
6. Area Under the Curve: The total area under the curve equals 1,
representing the total
probability.
7. Defined by Mean and Standard Deviation The shape and spread of the
curve are
determined solely by its mean (μ) and standard deviation (σ).
8. Continuous The normal distribution is a continuous probability
distribution, meaning it can
take any value within a range.
'''
```

```
'''
30.
Here are the evaluations of each option regarding the Normal
Distribution Curve:
(a) Correct: Within a range of ±0.6745σ, the middle 50% of
observations occur, covering
25% on each side.
(b) Correct: Mean ±1σ covers approximately 68.268% of the area, with
34.134% on either
side of the mean.
(c) Correct: Mean ±2σ covers approximately 95.45% of the area, with
47.725% on either
side of the mean.
(d) Correct Mean ±3σ covers approximately 99.73% of the area, with
49.865% on either side
of the mean.
(e) Correct: Only about 0.27% of the area is outside the range of μ
±3σ.
All options (a) to (e) are correct regarding the properties of the
Normal Distribution Curve
'''


                          ''' 50 . MACHINE LEARNING '''

' 50 . MACHINE LEARNING '


'''
50.1
- Series A one-dimensional labeled array capable of holding any data
type (integers, strings,
floating-point numbers, etc.). It can be thought of as a single column
of data.
- DataFrame: A two-dimensional labeled data structure with columns of
potentially different
types. It is similar to a spreadsheet or SQL table, where each column
can be a different data
type.
'''


'''
50.3
- `loc`: Used for label-based indexing, allowing you to access rows
and columns by their
```

```
labels (names). It includes the endpoints.
-`iloc`: Used for positional indexing, allowing you to access rows and
columns by their
integer positions (indices). It excludes the endpoint
'''


'''
50.4
- Supervised Learning: Involves training a model on labeled data,
where the output (target) is
known. The model learns to predict the output from the input data.
- Unsupervised Learning: Involves training a model on unlabeled data,
where the output is
unknown. The model identifies patterns, structures, or groupings in
the data without
predefined labels.


50.5
The bias-variance tradeoff is the balance between two types of errors
in a model:
- Bias: Error due to overly simplistic assumptions in the learning
algorithm, leading to
underfitting and poor performance on training data.
- Variance: Error due to excessive complexity in the model, leading to
overfitting and poor
generalization to unseen data.
The tradeoff involves finding a model that minimizes both bias and
variance for optimal
predictive performance.
'''


'''
50.6
- Precision: The ratio of true positive predictions to the total
predicted positives. It measures
the accuracy of positive predictions.
- Recall (Sensitivity): The ratio of true positive predictions to the
total actual positives. It
measures the ability to identify all relevant instances.
- Accuracy: The ratio of correct predictions (both true positives and
true negatives) to the
total predictions made. It measures overall correctness but can be
misleading in imbalanced
datasets.
Difference: Precision and recall focus specifically on positive
predictions, while accuracy
```

```
considers both positive and negative predictions.


50.7
Overfitting occurs when a model learns noise and details from the
training data to the extent
that it negatively impacts its performance on new data. This often
results in high accuracy on
training data but poor generalization to unseen data.
Prevention Methods:
1. Cross-Validation: Use techniques like k-fold cross-validation to
ensure the model
generalizes well.
2. Regularization: Apply techniques like L1 (Lasso) or L2 (Ridge)
regularization to penalize
overly complex models.
3. Pruning: In decision trees, reduce complexity by removing branches
that provide little
predictive power.
4. Early Stopping: Stop training when performance on a validation set
begins to decline.
5. Reduce Model Complexity: Use a simpler model with fewer parameters.
6. Increase Training Data: Provide more diverse training examples to
help the model learn
better generalization.
'''


'''
50.8
Cross-validation is a technique used to assess the performance and
generalization ability of
a machine learning model. It involves splitting the dataset into
multiple subsets (folds). The
model is trained on a subset of the data and tested on a different
subset, iterating this
process several times. The results are averaged to provide a more
reliable estimate of the
model's performance on unseen data, helping to prevent overfitting.
'''


'''
50.9
- Classification Problem: Involves predicting discrete labels or
categories (e.g., spam vs. not
spam). The output is a class label.
- Regression Problem: Involves predicting continuous values (e.g.,
house prices or
```

```
temperature). The output is a numeric value.


50.10
Ensemble Learning is a machine learning technique that combines
multiple models
(classifiers or regressors) to improve overall performance. The idea
is that by aggregating
the predictions of several models, the ensemble can achieve better
accuracy, robustness,
and generalization than any individual model. Common methods include
bagging (e.g.,
Random Forest) and boosting (e.g., AdaBoost, Gradient Boosting).


50.11
Gradient Descent is an optimization algorithm used to minimize a
function by iteratively
adjusting parameters in the opposite direction of the gradient (or
slope) of the function at the
current point.
How It Works:
1. Initialize Parameters: Start with random parameter values.
2. Compute Gradient: Calculate the gradient of the loss function with
respect to the
parameters.
3. Update Parameters Adjust the parameters using the formula
4. Repeat: Continue this process until convergence (when changes are
minimal) or a
maximum number of iterations is reached. ..This process helps find the
optimal parameters
that minimize the loss function.
'''


'''
50.12
Curse of Dimensionality refers to the challenges and issues that arise
when analyzing and
organizing data in high-dimensional spaces. In machine learning, it
can lead to:
1. Overfitting: Models may become too complex and fit noise in the
data rather than the
underlying distribution..
2. Increased Computational Cost: More dimensions require more data to
maintain statistical
significance, leading to higher resource consumption.
3. Sparse Data: Data points become sparse in high dimensions, making
it difficult for models
```

```
to learn patterns effectively.
Reducing dimensionality (e.g., through techniques like PCA) can help
mitigate these issues.



50.13
Difference: L1 can result in feature selection, while L2 shrinks
coefficients without
eliminating them.



50.14
A confusion matrix is a table used to evaluate the performance of a
classification model by
comparing predicted labels with actual labels.
Usage: It helps calculate key metrics such as accuracy, precision,
recall, and F1-score,
providing insights into the model's performance on different classes
'''



'''
50.15
The AUC-ROC curve (Area Under the Receiver Operating Characteristic
curve) is a
graphical representation of a classifier's performance across
different threshold values.
- ROC Curve: Plots the true positive rate (sensitivity) against the
false positive rate (1 -
specificity) at various thresholds.
- AUC: Measures the area under the ROC curve, ranging from 0 to 1. An
AUC of 1
indicates perfect classification, while an AUC of 0.5 indicates no
discrimination (random
guessing).


50.16
The k-nearest neighbors (KNN) algorithm is a simple, instance-based
machine learning
method used for classification and regression.
KNN is non-parametric and can adapt to complex decision boundaries but
can be
computationally expensive with large datasets.



50.17
```

```
Support Vector Machine (SVM) is a supervised machine learning
algorithm used for
classification and regression tasks.
SVM works by finding the optimal hyperplane that separates data points
of different classes
in a high-dimensional space. The goal is to maximize the margin
between the closest points
(support vectors) of each class and the hyperplane. This helps ensure
better generalization
to unseen data. SVM can also handle non-linear data by using kernel
functions to transform
the input space into a higher-dimensional space.
'''


'''
50.18
The kernel trick in Support Vector Machines (SVM) allows the algorithm
to operate in a
high-dimensional space without explicitly computing the coordinates of
the data points in that
space.
How It Works:
- Instead of transforming the input data into higher dimensions, the
kernel trick uses a kernel
function (e.g., polynomial, radial basis function) to compute the
inner products between data
points directly in the original input space.
- This enables SVM to create complex decision boundaries for non-
linearly separable data
while maintaining computational efficiency, as it avoids the explicit
transformation of all data
points.
'''


'''
50.19
Different types of kernels used in SVM include:
1. Linear Kernel:
- Usage: When the data is linearly separable. It is the simplest and
most efficient kernel.
2. Polynomial Kernel:
- Usage: When the relationship between features is polynomial. Useful
for data that
requires polynomial decision boundaries.
3. Radial Basis Function (RBF) Kernel (Gaussian Kernel):
- Usage: For non-linear data where the decision boundary is not clear.
```

```
It maps data into
an infinite-dimensional space, allowing for complex boundaries.
4. Sigmoid Kernel:
- Usage: Less commonly used; it behaves like a neural network.
Suitable for certain types
of non-linear relationships but can be less stable.
'''


'''
50.20
A hyperplane in SVM is a decision boundary that separates different
classes in the feature
space.
Determination:
It is determined by finding the optimal hyperplane that maximizes the
margin between the
closest data points (support vectors) of each class. This is achieved
through optimization
techniques that minimize classification errors while maximizing the
distance to the support
vectors, ensuring better generalization to unseen data.



50.21
Pros of SVM:
1. Effective in High Dimensions: Performs well with high-dimensional
data.
2. Robust to Overfitting: Particularly effective in cases where the
number of dimensions
exceeds the number of samples.
3. Versatile: Can be used for linear and non-linear classification
with different kernel
functions.
Cons of SVM:
1. Memory Intensive: Requires significant memory and computation time,
especially with
large datasets.
2. Difficult to Interpret: The model can be less interpretable
compared to simpler models.
3. Choice of Parameters: Performance is sensitive to the choice of
kernel and
hyperparameters (e.g., C and gamma), requiring careful tuning.



50.22
- Hard Margin SVM: Requires that all data points are correctly
```

```
classified with a clear margin.
It is used when the data is linearly separable without any noise or
outliers. It can lead to
overfitting if the data is not perfectly separable.
- Soft Margin SVM: Allows some data points to be misclassified (or to
fall within the margin)
by introducing slack variables. It provides a balance between
maximizing the margin and
minimizing classification errors, making it more robust to noise and
outliers in the dataset.
'''


'''
50.23
The process of constructing a decision tree involves the following
steps:
1. Select the Best Feature: Choose the feature that best splits the
data based on a criterion
(e.g., Gini impurity, entropy, or mean squared error) to maximize
information gain.
2. Create Nodes: Form a decision node for the selected feature and
branches for each
possible value or range of the feature.
3. Split the Dataset: Divide the dataset into subsets based on the
selected feature's values.
4. Repeat: Recursively repeat the process for each subset, selecting
the best feature at
each step, until a stopping criterion is met (e.g., maximum depth,
minimum samples per leaf,
or pure nodes).
5. Prune (if necessary): After the tree is fully grown, prune it to
remove branches that have
little predictive power to improve generalization and prevent
overfitting.



50.24
A decision tree is a supervised machine learning algorithm used for
classification and
regression tasks.
Working Principle:
1. Splitting: The tree is built by recursively splitting the dataset
into subsets based on feature
values. Each split is made to maximize the separation of classes (for
classification) or
minimize variance (for regression).
2. Node Creation: Each internal node represents a feature, each branch
```

```
represents a
decision rule, and each leaf node represents the predicted outcome
(class label or value).
3. Decision Making: To make predictions, the model follows the
decision rules from the root
to a leaf node based on the input features.
Decision trees are easy to interpret and visualize, but they can be
prone to overfitting if not
properly controlled.



'''
50.25
Information Gain is a metric used to measure the effectiveness of an
attribute in classifying
the data. It quantifies the reduction in entropy (uncertainty)
achieved by splitting the dataset
based on a specific attribute.
Usage in Decision Trees:
- During the tree-building process, information gain is calculated for
each attribute at a node.
The attribute with the highest information gain is selected for the
split, as it provides the most
useful information for classifying the data.
- This process continues recursively until a stopping criterion is
met, such as reaching a
maximum depth or having all data points in a leaf node belong to the
same class.



50.26
Gini impurity is a metric used to measure the impurity or purity of a
dataset in decision trees.
It quantifies how often a randomly chosen element from the set would
be incorrectly labeled
if it was randomly labeled according to the distribution of labels in
the subset.
Role in Decision Trees
- Gini impurity helps determine the best feature to split the data at
each node. The algorithm
selects the feature that results in the largest reduction in impurity
(i.e., the most homogenous
subsets) after the split.
- A Gini impurity of 0 indicates a pure node (all samples belong to a
single class), while
higher values indicate more mixed classes. The goal is to minimize
Gini impurity as the tree
```

```
grows.



50.27
Advantages of Decision Trees:
1. Easy to Interpret: Simple and intuitive to understand and
visualize.
2. Non-linear Relationships: Can model complex non-linear
relationships between features.
3. No Need for Feature Scaling: Works well without the need for
scaling or normalization of
data.
Disadvantages of Decision Trees:
1. Overfitting: Prone to overfitting, especially with deep trees and
noisy data.
2. Instability: Small changes in the data can lead to different tree
structures.
3. Bias towards Features: Can be biased towards features with more
levels, potentially
ignoring important predictors
'''



'''
50.28
Random forests improve upon decision trees by:
1. Ensemble Learning: Combining multiple decision trees to create a
more robust model,
reducing overfitting and improving generalization.
2. Random Feature Selection: At each split, a random subset of
features is considered,
which helps to reduce variance and improve model diversity.
3. Averaging Predictions: For regression, it averages the predictions
of individual trees, and
for classification, it uses majority voting, leading to more accurate
and stable results
compared to a single decision tree.



50.29
The Random Forest algorithm is an ensemble learning method that builds
multiple decision
trees and combines their predictions to improve accuracy and reduce
overfitting.
How It Works:
1. Bootstrap Sampling: Randomly selects subsets of the training data
(with replacement) to
```

```
create multiple decision trees.
2. Feature Randomness: At each split in a tree, only a random subset
of features is
considered, promoting diversity among trees.
3. Voting/Averaging: For classification, each tree votes for its
predicted class, and the
majority vote determines the final prediction. For regression, the
average of the predictions
from all trees is taken.
'''


'''
50.30
Bootstrapping in the context of random forests refers to the process
of creating multiple
subsets of the original dataset by randomly sampling with replacement.
Purpose: Each decision tree in the random forest is trained on a
different bootstrapped
sample, which introduces diversity among the trees. This helps improve
the model's
robustness and accuracy by reducing overfitting and variance when
aggregating predictions
(e.g., through voting or averaging).



50.31
Feature importance in Random Forests quantifies the contribution of
each feature
(predictor) to the model's predictions.
Key Points:
- It is calculated based on how much each feature reduces the impurity
(e.g., Gini impurity or
mean squared error) when used in tree splits across all trees in the
forest.
- Higher feature importance scores indicate that a feature is more
significant in predicting the
target variable, helping to identify relevant predictors and improve
model interpretability.
- Feature importance can be used for feature selection, allowing for
the elimination of less
informative features to enhance model performance and reduce
complexity
'''


'''
50.32
```

```
Key hyperparameters of a Random Forest and their effects:
1. n_estimators:
- Description: The number of decision trees in the forest.
- Effect: More trees generally improve performance and reduce
overfitting but increase
computation time.
2. max_depth:
- Description: The maximum depth of each tree.
- Effect: Limits overfitting; deeper trees can capture more complexity
but may lead to
overfitting.
3. min_samples_split:
- Description: The minimum number of samples required to split an
internal node.
- Effect: Higher values prevent the model from learning overly
specific patterns, reducing
overfitting.
4. min_samples_leaf:
- Description: The minimum number of samples required to be at a leaf
node.
- Effect: Prevents small leaf nodes, helping to smooth the model and
reduce overfitting.
5. max_features:
- Description: The number of features to consider when looking for the
best split.
- Effect: Reducing the number of features can improve generalization
and reduce
overfitting.
Tuning these hyperparameters affects the model's accuracy, complexity,
and ability to
generalize to new data.
'''


'''
50.33
**Logistic Regression Model:**
Logistic regression is a statistical model used for binary
classification that predicts the
probability of an outcome based on one or more predictor variables. It
uses the logistic
function (sigmoid) to map predicted values to probabilities between 0
and 1.
Assumptions:
1. Binary Outcome: The dependent variable should be binary (0 or 1).
2. independence: Observations should be independent of each other.
3. Linearity: There should be a linear relationship between the
independent variables and
the log odds of the dependent variable.
```

```
4. No Multicollinearity: Independent variables should not be too
highly correlated with each
other.



50.34
Logistic regression handles binary classification problems by modeling
the probability that a
given input belongs to a particular class (usually labeled as 1) using
the logistic function
(sigmoid function).
How It Works:
1. Linear Combination: It computes a linear combination of the input
features.
2. Logistic Function: The result is passed through the logistic
function, which transforms the
output into a value between 0 and 1, representing the probability of
the positive class.
3. Thresholding: A threshold (commonly 0.5) is applied to classify the
input into one of the
two classes. If the probability is greater than the threshold, it
predicts the positive class;
otherwise, it predicts the negative class.
This allows logistic regression to effectively model the relationship
between features and the
probability of class membership.

'''


'''
50.35
Usage in Logistic Regression: In logistic regression, the sigmoid
function is applied to the
linear combination of input features to convert the output into a
probability score
representing the likelihood of a binary class (e.g., 0 or 1). If the
output probability is greater
than a certain threshold (usually 0.5), the model predicts one class;
otherwise, it predicts the
other class. This enables logistic regression to handle binary
classification problems
effectively.



50.36
The cost function in logistic regression is used to measure the
```

difference between the
predicted probabilities and the actual binary outcomes (0 or 1)
50.37
1. One-vs-Rest (OvR): This approach involves training multiple binary classifiers, one
for each class. Each classifier predicts the probability of its respective class versus all
other classes. The class with the highest predicted probability is chosen as the final
output.
2. Softmax Regression (Multinomial Logistic Regression): This is a direct extension
of logistic regression that uses the softmax function to compute the probabilities for
multiple classes. It models the probabilities of each class simultaneously, ensuring
that the predicted probabilities sum to 1.
Both methods allow logistic regression to effectively handle multi-class classification tasks.

'''

'''
50.38
 L1 Regularization (Lasso): Adds the absolute value of the coefficients as a penalty
term to the loss function. It can lead to sparse models by driving some coefficients to
zero, effectively performing feature selection.
 L2 Regularization (Ridge): Adds the squared value of the coefficients as a penalty
term. It discourages large coefficients but does not set them to zero, leading to
smoother models without feature selection.
50.39
Key Differences from Other Boosting Algorithms:
1. Regularization: XGBoost includes L1 (Lasso) and L2 (Ridge) regularization, which
helps prevent overfitting and improves model generalization.
2. Parallel Processing: Utilizes parallel computation to speed up the training process,
making it faster than traditional boosting methods.
3. Tree Pruning: Implements a depth-first approach for tree construction, allowing it to
prune trees after growing, which leads to better performance.
4. Handling Missing Values: XGBoost can automatically learn how to handle missing

```
data during training.
50.40
Boosting is an ensemble learning technique that combines multiple weak
learners (often
decision trees) to create a strong learner.
Key Concept:
1. Sequential Learning: Boosting trains models sequentially, where
each subsequent
model focuses on the errors made by the previous ones.
2. Weighting: Data points that are misclassified by earlier models
receive higher
weights, making them more influential in the training of the next
model.
3. Final Prediction: The predictions from all models are combined,
typically through
weighted voting or averaging, to improve overall performance.
'''


'''
50.41
When constructing trees, XGBoost finds the optimal split for both
observed and missing
values.
It assigns a default direction (either left or right) for missing
values based on the gain from
the split, effectively treating missing values as a separate category
without requiring explicit
imputation.



50.42
Key hyperparameters in XGBoost include:
1. Learning Rate (eta): Controls the contribution of each tree. A
lower value makes the
model more robust but requires more trees to converge.
2. Max Depth: Defines the maximum depth of each tree. Deeper trees can
model more
complex patterns but are more prone to overfitting.
3. Min Child Weight: Controls the minimum sum of instance weight
(hessian) needed
in a child. Higher values prevent the model from learning overly
specific patterns.
4. Subsample: The fraction of samples used for fitting individual
trees. Lower values
help prevent overfitting but can lead to underfitting if too low.
5. Colsample_bytree: The fraction of features used for each tree.
Reducing this can
```

```
help with overfitting by introducing randomness.



50.43
Gradient Boosting in XGBoost is an ensemble learning technique that
builds models
sequentially to improve prediction accuracy.
Process:
1. Initialization: Start with a base model (e.g., a simple prediction,
such as the mean of
the target variable).
2. Iterative Training: In each iteration, fit a new decision tree to
the residuals (errors) of
the previous model's predictions.
3. Gradient Calculation: Calculate the gradient of the loss function
to determine the
direction and magnitude of improvement needed for predictions.
4. Update Model: Add the new tree to the ensemble with a specific
learning rate to
control the contribution of each tree.
5. Repeat: Continue adding trees until a specified number of trees is
reached or the
improvement becomes negligible.


'''
50.44
Advantages of XGBoost:
1. High Performance: Often provides state-of-the-art results in
classification and
regression tasks due to its boosting mechanism.
2. Regularization: Includes L1 and L2 regularization to prevent
overfitting.
3. Parallel Processing: Utilizes parallel computation for faster
training, making it
efficient for large datasets.
4. Flexibility: Supports various objective functions and custom loss
functions.
Disadvantages of XGBoost:
1. Complexity: More complex than simpler models, making it harder to
interpret.
2. Parameter Tuning: Requires careful tuning of hyperparameters for
optimal
performance, which can be time-consuming.
3. Memory Usage: Can consume significant memory resources, especially
with large
datasets.
'''
```