

Question 1.3. name = ["Mohan", "dash", "karam", "chandra", "gandhi", "Bapu"] do the following operations in this list;

- add an element "freedom\_fighter" in this list at the 0th index.
- find the output of the following, and explain how?
- add two more elements in the name ["Netaji", "Bose"] at the end of the list.
- what will be the value of temp: name = ["Bapuji", "dash", "karam", "chandra", "gandhi", "Mohan"]  
temp=name[-1] name[-1]=name[0] name[0]=temp print(name)

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

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

```
name.extend(["Netaji", "Bose"])
name
```

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

```
name = ["Bapuji", "dash", "karam", "chandra", "gandhi", "Mohan"]
temp = name[-1]
name[-1] = name[0]
name[0] = temp
print(name)
```

```
→ ['Mohan', 'dash', 'karam', 'chandra', 'gandhi', 'Bapuji']
```

Question 1.4. Find the output of the following.

```
animal = ['Human', 'cat', 'mat', 'cat', 'rat', 'Human', 'Lion']
print(animal.count('Human'))
print(animal.index('rat'))
print(len(animal))

animal = ['Human', 'cat', 'mat', 'cat', 'rat', 'Human', 'Lion']

print(animal.count('Human'))

print(animal.index('rat'))

print(len(animal))
```

```
→ 2
 4
 7
```

Question 1.5. tuple1=(10,20,"Apple", 3.4,'a',["master","ji"],("sita","geeta",22),{"roll\_no":1}, {"name":"Navneet"})

- a) print(len(tuple1))
- b) print(tuple1[-1][-1])
- c) fetch the value of roll\_no from this tuple.

File "<ipython-input-2-8752465adcb7>", line 6 <in >::

You

Please explain the error:

```
File "<ipython-input-2-8752465adcb7>", line 6
<input type="submit" value="Submit">
^
```

IndentationError: unexpected indent

Gemini

## Hypothesis

The error message "IndentationError: unexpected indent" on line 6 of your code suggests that the line `<input type="submit" value="Submit">` is unexpectedly indented. Python is very strict about indentation, and any misalignment can cause issues. It's likely that this line should be at the same indentation level as the line above it (`<input type="text" name="username" placeholder="Enter username">`).

## Suggested Changes

```
# 40. Handling Form Submissions with POST Request:
```

```
# <!-- templates/form.html -->
# <form action="/submit" method="post">
#   <input type="text" name="username" placeholder="Enter username">
#   <input type="submit" value="Submit" # Removed indentation
# </form>
```

```
from flask import Flask, request, render_template
```

```
app = Flask(__name__)

@app.route('/')
def form():
    return render_template('form.html')

@app.route('/submit', methods=['POST'])
def submit():
    username = request.form['username']
    return f'Hello, {username}!'

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

Use code with caution

### Sources

- [stack overflow.com/questions/25585839/indentationerror-unexpected-indent-when-returning-a-variable-after-form-submission](https://stackoverflow.com/questions/25585839/indentationerror-unexpected-indent-when-returning-a-variable-after-form-submission)
- [stack overflow.com/questions/1016814/what-should-i-do-with-unexpected-indent-in-python](https://stackoverflow.com/questions/1016814/what-should-i-do-with-unexpected-indent-in-python)
- [www.edureka.co/community/50861/how-do-i-resolve-the-unexpected-indent-error-in-python](https://www.edureka.co/community/50861/how-do-i-resolve-the-unexpected-indent-error-in-python)
- [careerkarma.com/blog/python-indentationerror-unexpected-indent/](https://careerkarma.com/blog/python-indentationerror-unexpected-indent/)
- [www.geeksforgeeks.org/how-to-resolve-the-unexpected-indent-error-in-python/](https://www.geeksforgeeks.org/how-to-resolve-the-unexpected-indent-error-in-python/)
- [stack overflow.com/questions/47191903/flask-post-request-returning-to-main-webpage-instead-of-returning-result](https://stackoverflow.com/questions/47191903/flask-post-request-returning-to-main-webpage-instead-of-returning-result)

Rate this answer

- d) print(tuple1[-3][1])  
e) fetch the element "22" from this tuple.

```
tuple1=(10,20,"Apple", 3.4,'a', ["master","ji"],("sita","geeta",22),[{"roll
print(len(tuple1))

print(tuple1[-1][-1]["name"])

print(tuple1[-3][1])

tuple1[-2]
```

8  
Navneet  
ji  
('sita', 'geeta', 22)

- 1.6. Write a program to display the appropriate message as per the color of signal (RED-Stop/Yellow-Stay/ Green-Go) at the road crossing.

```
# Function to display message based on signal color
def display_signal_message(signal_color):
    # Convert input to lowercase to handle case-insensitive input
    signal_color = signal_color.lower()

    # Check the color and print the corresponding message
    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.")

# Main program
if __name__ == "__main__":
    # Ask user for input
    user_input = input("Enter the color of the traffic signal (Red/Yellow/")

    # Display the appropriate message
    display_signal_message(user_input)
```

Enter the color of the traffic signal (Red/Yellow/Green): Red  
Stop

- 1.9. Write a program to find the factors of a whole number using a while loop.

- 1.10. Write a program to find the sum of all the positive numbers entered by the user. As soon as the user enters a negative number, stop taking in any further input from the user and display the sum.

- 1.11. Write a program to find prime numbers between 2 to 100 using nested for loops.

```
# Function to find and print factors of a number
def find_factors(number):
    i = 1
    print(f"Factors of {number} are:")
    while i <= number:
        if number % i == 0:
            print(i, end=" ")
        i += 1
    print()

# Main program
if __name__ == "__main__":
    try:
        num = int(input("Enter a whole number: "))
        find_factors(num)
    except ValueError:
        print("Please enter a valid whole number.")
```

→ Enter a whole number: 10  
Factors of 10 are:  
1 2 5 10

```
# Function to sum positive numbers
def sum_positive_numbers():
    total_sum = 0
    while True:
        try:
            num = float(input("Enter a positive number (or a negative number to stop): "))
            if num < 0:
                break
            else:
                total_sum += num
        except ValueError:
            print("Please enter a valid number.")
    print(f"Sum of all positive numbers entered: {total_sum}")

# Main program
if __name__ == "__main__":
    sum_positive_numbers()
```

→ Enter a positive number (or a negative number to stop): 5  
Enter a positive number (or a negative number to stop): 10  
Enter a positive number (or a negative number to stop): 15  
Enter a positive number (or a negative number to stop): 2  
Enter a positive number (or a negative number to stop):  
Please enter a valid number.  
Enter a positive number (or a negative number to stop): -1  
Sum of all positive numbers entered: 32.0

```
# Function to find prime numbers between a given range
def find_prime_numbers(start, end):
    print(f"Prime numbers between {start} and {end} are:")
    for num in range(start, end + 1):
        if num > 1:
            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=" ")
print()

# Main program
if __name__ == "__main__":
    find_prime_numbers(2, 100)
```

→ Prime numbers between 2 and 100 are:  
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. Write the programs for the following:

- Accept the marks of the student in five major subjects and display the same.
- Calculate the sum of the marks of all subjects. Divide the total marks by number of subjects (i.e. 5), calculate percentage = total marks/5 and display the percentage.
- Find the grade of the student as per the following criteria. Hint: Use Match & case for this.:

Criteria Grade

A percentage > 85  
B percentage < 75 && percentage >= 50

C percentage > 30 && percentage <= 50  
D percentage < 30  
Reappear

```

def main():
    # Accept marks for five subjects
    marks = []
    subjects = ["Subject 1", "Subject 2", "Subject 3", "Subject 4", "Subject 5"]

    print("Enter the marks for the following subjects:")
    for subject in subjects:
        while True:
            try:
                mark = float(input(f"{subject}: "))
                if mark < 0:
                    print("Marks cannot be negative. Please enter a valid number")
                else:
                    marks.append(mark)
                    break
            except ValueError:
                print("Invalid input. Please enter a valid number.")

    # Display the marks
    print("\nMarks obtained:")
    for i, mark in enumerate(marks, start=1):
        print(f"Subject {i}: {mark}")

    # Calculate the total marks and percentage
    total_marks = sum(marks)
    number_of_subjects = len(marks)
    percentage = total_marks / number_of_subjects

    # Display the percentage
    print(f"\nTotal Marks: {total_marks}")
    print(f"Percentage: {percentage:.2f}%")

    # Determine and display the grade
    match percentage:
        case p if p > 85:
            grade = "A"
        case p if 75 <= p <= 85:
            grade = "B"
        case p if 50 <= p < 75:
            grade = "C"
        case p if 30 <= p < 50:
            grade = "D"
        case _:
            grade = "Reappear"

    print(f"Grade: {grade}")

# Main entry point
if __name__ == "__main__":
    main()

```

→ Enter the marks for the following subjects:

Subject 1: 98  
 Subject 2: 85  
 Subject 3: 96  
 Subject 4: 85  
 Subject 5: 75

Marks obtained:  
 Subject 1: 98.0  
 Subject 2: 85.0  
 Subject 3: 96.0  
 Subject 4: 85.0  
 Subject 5: 75.0

Total Marks: 439.0  
 Percentage: 87.80%  
 Grade: A

1.14. Consider the gravitational interactions between the Earth, Moon, and Sun in our solar system. Given: mass\_earth = 5.972e24 # Mass of Earth in kilograms mass\_moon = 7.34767309e22 #Mass of Moon in kilograms mass\_sun = 1.989e30 # Mass of Sun in kilograms distance\_earth\_sun = 1.496e11 # Average distance between Earth and Sun in meters distance\_moon\_earth = 3.844e8 # Average distance between Moon and Earth in meters Tasks:

- Calculate the gravitational force between the Earth and the Sun.
- Calculate the gravitation force between the Moon and the Earth.
- Compare the calculated forces to determine which gravitational force is stronger.
- Explain which celestial body (Earth or Moon) is more attracted to the other based on the comparison.

```
G = 6.67430e-11 # Gravitational constant in m^3 kg^-1 s^-2
```

```
# Masses
mass_earth = 5.972e24 # kg
mass_sun = 1.989e30 # kg
```

```
# Distance
distance_earth_sun = 1.496e11 # meters
```

```
# Gravitational force between Earth and Sun
force_earth_sun = G * (mass_earth * mass_sun) / (distance_earth_sun ** 2)
```

```
# Masses
mass_moon = 7.34767309e22 # kg
```

```
# Distance
distance_moon_earth = 3.844e8 # meters
```

```
# Gravitational force between Moon and Earth
force_moon_earth = G * (mass_earth * mass_moon) / (distance_moon_earth **
```

```
# Compare the forces
if force_earth_sun > force_moon_earth:
    stronger_force = "Earth and Sun"
    weaker_force = "Moon and Earth"
else:
    stronger_force = "Moon and Earth"
    weaker_force = "Earth and Sun"
```

```
# Constants
G = 6.67430e-11 # Gravitational constant in m^3 kg^-1 s^-2

# Masses
mass_earth = 5.972e24 # kg
mass_moon = 7.34767309e22 # kg
mass_sun = 1.989e30 # kg

# Distances
distance_earth_sun = 1.496e11 # meters
distance_moon_earth = 3.844e8 # meters

# Calculate forces
force_earth_sun = G * (mass_earth * mass_sun) / (distance_earth_sun ** 2)
force_moon_earth = G * (mass_earth * mass_moon) / (distance_moon_earth ** 2)

# Print forces
print(f"Gravitational force between Earth and Sun: {force_earth_sun:.2e} N")
print(f"Gravitational force between Moon and Earth: {force_moon_earth:.2e} N")

# Compare forces
if force_earth_sun > force_moon_earth:
    print("The gravitational force between Earth and Sun is stronger.")
    print("Earth is more attracted to the Sun.")
else:
    print("The gravitational force between Moon and Earth is stronger.")
    print("The Moon is more attracted to the Earth.")
```

→ Gravitational force between Earth and Sun: 3.54e+22 N  
Gravitational force between Moon and Earth: 1.98e+20 N  
The gravitational force between Earth and Sun is stronger.  
Earth is more attracted to the Sun.

2. Design and implement a Python program for managing student information using object-oriented principles. Create a class called `Student` with encapsulated attributes for name, age, and roll number. Implement getter and setter methods for these attributes. Additionally, provide methods to display student information and update student details.

Tasks

- 3 Define the `Student` class with encapsulated attributes
- 3 Implement getter and setter methods for the attributes
- 3 Write methods to display student information and update details
- 3 Create instances of the `Student` class and test the implemented functionality

**bold text**

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

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

    def get_age(self):
        return self.__age

    def get_roll_number(self):
        return self.__roll_number

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

    def set_age(self, age):
        self.__age = age

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

    # Method to display student information
    def display_info(self):
        print(f"Name: {self.__name}")
        print(f"Age: {self.__age}")
        print(f"Roll Number: {self.__roll_number}")

    # Method to update student details
    def update_details(self, name=None, age=None, roll_number=None):
        if name:
            self.__name = name
        if age:
            self.__age = age
        if roll_number:
            self.__roll_number = roll_number
        print("Student details updated successfully.")

# Create instances of the Student class and test the implemented functions

# Creating a student instance
student1 = Student("Alice", 20, "S101")

print("Original Student Information:")
student1.display_info()

student1.update_details(name="Alice Johnson", age=21)

print("\nUpdated Student Information:")
student1.display_info()

student2 = Student("Bob", 22, "S102")
print("\nUsing Getter and Setter Methods:")
print(f"Name: {student2.get_name()}")
student2.set_name("Bob Smith")
print(f"Updated Name: {student2.get_name()}")
```

#### Original Student Information:

```
Name: Alice  
Age: 20  
Roll Number: S101  
Student details updated successfully.
```

#### Updated Student Information:

```
Name: Alice Johnson  
Age: 21  
Roll Number: S101
```

#### Using Getter and Setter Methods:

```
Name: Bob  
Updated Name: Bob Smith
```

3. Develop a Python program for managing library resources efficiently. Design a class named `LibraryBook` with attributes like `book_name`, `author`, and `availability status`. Implement methods for borrowing and returning books while ensuring proper encapsulation of attributes.

Tasks 3 1. Create the `LibraryBook` class with encapsulated attributes 3 2. Implement methods for borrowing and returning books 3 3. Ensure proper encapsulation to protect book details 3 4. Test the borrowing and returning functionality with sample data.

```

class LibraryBook:
    def __init__(self, book_name, author):
        self.__book_name = book_name
        self.__author = author
        self.__is_available = True # By default, the book is available

    # Getter methods
    def get_book_name(self):
        return self.__book_name

    def get_author(self):
        return self.__author

    def is_available(self):
        return self.__is_available

    # Method to borrow a book
    def borrow_book(self):
        if self.__is_available:
            self.__is_available = False
            print(f"You have successfully borrowed '{self.__book_name}' by {self.__author}")
        else:
            print(f"Sorry, '{self.__book_name}' is currently unavailable.")

    # Method to return a book
    def return_book(self):
        if not self.__is_available:
            self.__is_available = True
            print(f"Thank you for returning '{self.__book_name}'.")
        else:
            print(f"'{self.__book_name}' was not borrowed.")

# Test the borrowing and returning functionality with sample data

# Creating book instances
book1 = LibraryBook("The Great Gatsby", "F. Scott Fitzgerald")
book2 = LibraryBook("1984", "George Orwell")

# Test borrowing books
print("Borrowing Books:")
book1.borrow_book() # Should succeed
book1.borrow_book() # Should indicate the book is unavailable

# Test returning books
print("\nReturning Books:")
book1.return_book() # Should succeed
book1.return_book() # Should indicate the book was not borrowed

# Test another book
print("\nTesting Another Book:")
book2.borrow_book() # Should succeed
book2.return_book() # Should succeed

# Check availability status
print("\nChecking Availability:")
print(f"Is '{book1.get_book_name()}' available? {'Yes' if book1.is_available() else 'No'}")
print(f"Is '{book2.get_book_name()}' available? {'Yes' if book2.is_available() else 'No'}
```

 Borrowing Books:  
You have successfully borrowed 'The Great Gatsby' by F. Scott Fitzgerald.  
Sorry, 'The Great Gatsby' is currently unavailable.

Returning Books:  
Thank you for returning 'The Great Gatsby'.  
'The Great Gatsby' was not borrowed.

Testing Another Book:  
You have successfully borrowed '1984' by George Orwell.

```
Thank you for returning '1984'.
```

```
Checking Availability:
```

```
Is 'The Great Gatsby' available? Yes
```

```
Is '1984' available? Yes
```

4. Create a simple banking system using object-oriented concepts in Python. Design classes representing different types of bank accounts such as savings and checking. Implement methods for deposit, withdraw, and balance inquiry. Utilize inheritance to manage different account types efficiently.

Tasks

- 1. Define base class(es) for bank accounts with common attributes and methods
- 2. Implement subclasses for specific account types (e.g., SavingsAccount, CheckingAccount)
- 3. Provide methods for deposit, withdraw, and balance inquiry in each subclass
- 4. Test the banking system by creating instances of different account types and performing transactions.

```

class BankAccount:
    def __init__(self, account_number, holder_name, balance=0):
        self._account_number = account_number
        self._holder_name = holder_name
        self._balance = balance

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

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

    def inquire_balance(self):
        return self._balance

class SavingsAccount(BankAccount):
    def __init__(self, account_number, holder_name, balance=0, interest_rate=0.01):
        super().__init__(account_number, holder_name, balance)
        self.interest_rate = interest_rate

    def apply_interest(self):
        interest = self._balance * self.interest_rate
        self.deposit(interest)
        print(f"Applied interest of ${interest}. New balance: ${self._balance}.")

class CheckingAccount(BankAccount):
    def __init__(self, account_number, holder_name, balance=0, overdraft_limit=0):
        super().__init__(account_number, holder_name, balance)
        self.overdraft_limit = overdraft_limit

    def withdraw(self, amount):
        if amount > 0 and amount <= (self._balance + self.overdraft_limit):
            self._balance -= amount
            print(f"Withdrew ${amount}. New balance: ${self._balance}.")
        elif amount > (self._balance + self.overdraft_limit):
            print("Insufficient funds, even considering overdraft limit.")
        else:
            print("Withdrawal amount must be positive.")

# Test the banking system
def test_banking_system():
    # Create a savings account and perform operations
    savings_account = SavingsAccount("SA123", "Alice", balance=500)
    print("\nSavings Account Operations:")
    savings_account.deposit(200)
    savings_account.withdraw(150)
    savings_account.apply_interest()
    print(f"Balance inquiry: ${savings_account.inquire_balance()}")

    # Create a checking account and perform operations
    checking_account = CheckingAccount("CA456", "Bob", balance=300)
    print("\nChecking Account Operations:")
    checking_account.deposit(100)

```

```
checking_account.withdraw(350)
checking_account.withdraw(200) # Attempt to withdraw more than balance
print(f"Balance inquiry: ${checking_account.inquire_balance()}")
```

```
# Run the test
test_banking_system()
```

→ Savings Account Operations:  
 Deposited \$200. New balance: \$700.  
 Withdrew \$150. New balance: \$550.  
 Deposited \$5.5. New balance: \$555.5.  
 Applied interest of \$5.5. New balance: \$555.5.  
 Balance inquiry: \$555.5

Checking Account Operations:  
 Deposited \$100. New balance: \$400.  
 Withdrew \$350. New balance: \$50.  
 Insufficient funds, even considering overdraft limit.  
 Balance inquiry: \$50

5. Write a Python program that models different animals and their sounds. Design a base class called `Animal` with a method `make_sound()`. Create subclasses like `Dog` and `Cat` that override the `make_sound()` method to produce appropriate sounds.

Tasks

1. Define the `Animal` class with a method `make_sound()`
2. Create subclasses `Dog` and `Cat` that override the `make_sound()` method
3. Implement the sound generation logic for each subclass
4. Test the program by creating instances of `Dog` and `Cat` and calling the `make_sound()` method.

Double-click (or enter) to edit

```
class Animal:
    def make_sound(self):
        raise NotImplementedError("Subclass must implement abstract method")

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

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

# Test the program
def test_animal_sounds():
    # Create instances of Dog and Cat
    dog = Dog()
    cat = Cat()

    # Call the make_sound() method
    print("Dog Sound:")
    dog.make_sound()

    print("\nCat Sound:")
    cat.make_sound()

# Run the test
test_animal_sounds()
```

Dog Sound:  
Woof! Woof!

Cat Sound:  
Meow! Meow!

6. Write a code for Restaurant Management System Using OOPS & Create a MenuItem class that has attributes such as name, description, price, and category & Implement methods to add a new menu item, update menu item information, and remove a menu item from the menu & Use encapsulation to hide the menu item's unique identification number & Inherit from the MenuItem class to create a FoodItem class and a BeverageItem class, each with their own specific attributes and methods.

```
import itertools

class MenuItem:
    _id_counter = itertools.count(1) # Unique ID generator

    def __init__(self, name, description, price, category):
        self._id = next(MenuItem._id_counter)
        self.name = name
        self.description = description
        self.price = price
        self.category = category

    # Getter for ID (only if needed, but generally ID should be kept private)
    def get_id(self):
        return self._id

    # Method to update menu item information
    def update_info(self, name=None, description=None, price=None, category=None):
        if name is not None:
            self.name = name
        if description is not None:
            self.description = description
        if price is not None:
            self.price = price
        if category is not None:
            self.category = category

    # Method to display menu item information
    def display_info(self):
        print(f"ID: {self._id}")
        print(f"Name: {self.name}")
        print(f"Description: {self.description}")
        print(f"Price: ${self.price:.2f}")
        print(f"Category: {self.category}")

    # Static method to remove a menu item (example implementation)
    @staticmethod
    def remove_item(item_list, item_id):
        item_list[:] = [item for item in item_list if item.get_id() != item_id]

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

    # Method specific to food items
    def display_info(self):
        super().display_info()
        print(f"Cuisine Type: {self.cuisine_type}")

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

    # Method specific to beverage items
    def display_info(self):
        super().display_info()
        print(f"Alcoholic: {'Yes' if self.is_alcoholic else 'No'}")

# Test the Restaurant Management System
def test_restaurant_management_system():
    # Create menu items
```

```

pizza = FoodItem("Margherita Pizza", "Classic cheese pizza", 12.99, "Italian")
soda = BeverageItem("Cola", "Refreshing cola drink", 1.99, "Beverage", "Non-Alcoholic")

# Store items in a list
menu_items = [pizza, soda]

# Display all items
print("Menu Items:")
for item in menu_items:
    item.display_info()
    print()

# Update and display the pizza item
pizza.update_info(price=13.99)
print("Updated Pizza Item:")
pizza.display_info()
print()

# Remove the soda item and display remaining items
MenuItem.remove_item(menu_items, soda.get_id())
print("Remaining Menu Items:")
for item in menu_items:
    item.display_info()
    print()

# Run the test
test_restaurant_management_system()

```

```

→ Menu Items:
ID: 1
Name: Margherita Pizza
Description: Classic cheese pizza
Price: $12.99
Category: Main Course
Cuisine Type: Italian

ID: 2
Name: Cola
Description: Refreshing cola drink
Price: $1.99
Category: Beverage
Alcoholic: No

Updated Pizza Item:
ID: 1
Name: Margherita Pizza
Description: Classic cheese pizza
Price: $13.99
Category: Main Course
Cuisine Type: Italian

Remaining Menu Items:
ID: 1
Name: Margherita Pizza
Description: Classic cheese pizza
Price: $13.99
Category: Main Course
Cuisine Type: Italian

```

Q. Write a code for Hotel Management System using OOPS & Create a Room class that has attributes such as room number, room type, rate, and availability (private) & Implement methods to book a room, check in a guest, and check out a guest & Use encapsulation to hide the room's unique identification number & Inherit from the Room class to create a SuiteRoom class and a StandardRoom class, each with their own specific attributes and methods.

```

class Room:
    _id_counter = 1 # Class variable for room ID generation

    def __init__(self, room_type, rate):
        self.__room_number = Room._id_counter
        Room._id_counter += 1
        self.room_type = room_type
        self.rate = rate
        self.__is_available = True

    # Getter for room number (only if needed)
    def get_room_number(self):
        return self.__room_number

    # Method to book a room
    def book_room(self):
        if self.__is_available:
            self.__is_available = False
            print(f"Room {self.__room_number} booked successfully.")
        else:
            print(f"Room {self.__room_number} is already booked.")

    # Method to check in a guest
    def check_in(self):
        if not self.__is_available:
            print(f"Guest checked into Room {self.__room_number}.")
        else:
            print(f"Room {self.__room_number} is not booked. Cannot check in guest.")

    # Method to check out a guest
    def check_out(self):
        if not self.__is_available:
            self.__is_available = True
            print(f"Guest checked out of Room {self.__room_number}.")
        else:
            print(f"Room {self.__room_number} is not currently occupied. Cannot check out guest.")

    # Method to display room information
    def display_info(self):
        availability = "Available" if self.__is_available else "Not Available"
        print(f"Room Number: {self.__room_number}")
        print(f"Room Type: {self.room_type}")
        print(f"Rate: ${self.rate:.2f}")
        print(f"Availability: {availability}")

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

    # Method to display suite room information
    def display_info(self):
        super().display_info()
        print(f"Number of Bedrooms: {self.num_bedrooms}")

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

    # Method to display standard room information
    def display_info(self):
        super().display_info()
        print(f"Has Balcony: {'Yes' if self.has_balcony else 'No'}")

```

```
# Test the Hotel Management System
def test_hotel_management_system():
    # Create room instances
    suite = SuiteRoom(rate=250, num_bedrooms=2)
    standard = StandardRoom(rate=100, has_balcony=True)

    # Display room information
    print("Room Information:")
    suite.display_info()
    print()
    standard.display_info()
    print()

    # Test booking and check-in/check-out functionality
    print("Booking and Check-In/Check-Out Operations:")
    suite.book_room()
    suite.check_in()
    suite.check_out()
    suite.check_out() # Attempt to check out again
    print()

    standard.book_room()
    standard.check_in()
    standard.check_out()
    standard.check_in() # Attempt to check in again
    print()

# Run the test
test_hotel_management_system()
```

→ Room Information:  
 Room Number: 1  
 Room Type: Suite  
 Rate: \$250.00  
 Availability: Available  
 Number of Bedrooms: 2

Room Number: 2  
 Room Type: Standard  
 Rate: \$100.00  
 Availability: Available  
 Has Balcony: Yes

Booking and Check-In/Check-Out Operations:  
 Room 1 booked successfully.  
 Guest checked into Room 1.  
 Guest checked out of Room 1.  
 Room 1 is not currently occupied. Cannot check out.

Room 2 booked successfully.  
 Guest checked into Room 2.  
 Guest checked out of Room 2.  
 Room 2 is not booked. Cannot check in.

8. Write a code for Fitness Club Management System using OOPS & Create a Member class that has attributes such as name, age, membership type, and membership status (private) & Implement methods to register a new member, renew a membership, and cancel a membership & Use encapsulation to hide the member's unique identification number & Inherit from the Member class to create a FamilyMember class and an IndividualMember class, each with their own specific attributes and methods.

```
import itertools

class Member:
    _id_counter = itertools.count(1) # Unique ID generator

    def __init__(self, name, age, membership_type):
        self.__id = next(Member._id_counter)
        self.name = name
        self.age = age
        self.membership_type = membership_type
        self.__membership_status = "Active" # Membership is active by def

    # Getter for ID (only if needed)
    def get_id(self):
        return self.__id

    # Method to register a new member
    def register_member(self):
        print(f"Member {self.name} registered successfully with ID {self._id}.")

    # Method to renew a membership
    def renew_membership(self):
        if self.__membership_status == "Cancelled":
            self.__membership_status = "Active"
            print(f"Membership for {self.name} renewed successfully.")
        else:
            print(f"Membership for {self.name} is already active.")

    # Method to cancel a membership
    def cancel_membership(self):
        if self.__membership_status == "Active":
            self.__membership_status = "Cancelled"
            print(f"Membership for {self.name} has been cancelled.")
        else:
            print(f"Membership for {self.name} is already cancelled.")

    # Method to display member information
    def display_info(self):
        print(f"Member ID: {self.__id}")
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Membership Type: {self.membership_type}")
        print(f"Membership Status: {self.__membership_status}")

class FamilyMember(Member):
    def __init__(self, name, age, membership_type, num_family_members):
        super().__init__(name, age, membership_type)
        self.num_family_members = num_family_members

    # Method to display family member information
    def display_info(self):
        super().display_info()
        print(f"Number of Family Members: {self.num_family_members}")

class IndividualMember(Member):
    def __init__(self, name, age, membership_type, personal_trainer):
        super().__init__(name, age, membership_type)
        self.personal_trainer = personal_trainer

    # Method to display individual member information
    def display_info(self):
        super().display_info()
        print(f"Personal Trainer: {self.personal_trainer}")
```

```
# Test the Fitness Club Management System
def test_fitness_club_management_system():
    # Create member instances
    family_member = FamilyMember(name="John Doe", age=40, membership_type=
    individual_member = IndividualMember(name="Jane Smith", age=30, member

    # Register and display information
    print("Registering and Displaying Members:")
    family_member
```

9. Write a code for Event Management System using OOPS & Create an Event class that has attributes such as name, date, time, location, and list of attendees (private) & Implement methods to create a new event, add or remove attendees, and get the total number of attendees & Use encapsulation to hide the event's unique identification number & Inherit from the Event class to create a PrivateEvent class and a PublicEvent class, each with their own specific attributes and methods.

```
import itertools

class Event:
    _id_counter = itertools.count(1) # Unique ID generator

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

    # Getter for event ID (only if needed)
    def get_id(self):
        return self.__id

    # Method to create a new event
    def create_event(self):
        print(f"Event '{self.name}' created with ID {self.__id}.")

    # Method to add an attendee
    def add_attendee(self, attendee_name):
        if attendee_name not in self.__attendees:
            self.__attendees.append(attendee_name)
            print(f"Added {attendee_name} to the attendee list.")
        else:
            print(f"{attendee_name} is already an attendee.")

    # Method to remove an attendee
    def remove_attendee(self, attendee_name):
        if attendee_name in self.__attendees:
            self.__attendees.remove(attendee_name)
            print(f"Removed {attendee_name} from the attendee list.")
        else:
            print(f"{attendee_name} is not in the attendee list.")

    # Method to get the total number of attendees
    def get_total_attendees(self):
        return len(self.__attendees)

    # Method to display event information
    def display_info(self):
        print(f"Event ID: {self.__id}")
        print(f"Name: {self.name}")
        print(f"Date: {self.date}")
```

```
print(self.date, self.location, )
print(f"Time: {self.time}")
print(f"Location: {self.location}")
print(f"Total Attendees: {self.get_total_attendees()}")
```

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

    # Method to display private event information
    def display_info(self):
        super().display_info()
        print(f"Host Name: {self.host_name}")
```

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

    # Method to display public event information
    def display_info(self):
        super().display_info()
        print(f"Ticket Price: ${self.ticket_price:.2f}")
```

```
# Test the Event Management System
def test_event_management_system():
    # Create event instances
    private_event = PrivateEvent(name="Private Party", date="2024-09-15", t
    public_event = PublicEvent(name="Music Festival", date="2024-10-20", ti

    # Create events
    print("Creating Events:")
    private_event.create_event()
    public_event.create_event()
    print()

    # Add and remove attendees
    print("Managing Attendees for Private Event:")
    private_event.add_attendee("John Doe")
    private_event.add_attendee("Jane Smith")
    private_event.remove_attendee("John Doe")
    print()

    print("Managing Attendees for Public Event:")
    public_event.add_attendee("Alice Johnson")
    public_event.add_attendee("Bob Brown")
    public_event.remove_attendee("Alice Johnson")
    print()

    # Display event information
    print("Event Information:")
    private_event.display_info()
    print()
    public_event.display_info()

# Run the test
test_event_management_system()
```

→ Creating Events:  
Event 'Private Party' created with ID 1.  
Event 'Music Festival' created with ID 2.

```
Managing Attendees for Private Event:  
Added John Doe to the attendee list.  
Added Jane Smith to the attendee list.  
Removed John Doe from the attendee list.
```

```
Managing Attendees for Public Event:  
Added Alice Johnson to the attendee list.  
Added Bob Brown to the attendee list.  
Removed Alice Johnson from the attendee list.
```

```
Event Information:  
Event ID: 1  
Name: Private Party  
Date: 2024-09-15  
Time: 18:00  
Location: Rooftop Lounge  
Total Attendees: 1  
Host Name: Alice
```

```
Event ID: 2  
Name: Music Festival  
Date: 2024-10-20  
Time: 14:00  
Location: City Park  
Total Attendees: 1  
Ticket Price: $25.00
```

10. Write a code for Airline Reservation System using OOPS & Create a Flight class that has attributes such as flight number, departure and arrival airports, departure and arrival times, and available seats (private) & Implement methods to book a seat, cancel a reservation, and get the remaining available seats & Use encapsulation to hide the flight's unique identification number & Inherit from the Flight class to create a DomesticFlight class and an InternationalFlight class, each with their own specific attributes and methods.

```
import itertools

class Flight:
    _id_counter = itertools.count(1) # Unique ID generator

    def __init__(self, departure_airport, arrival_airport, departure_time,
                 self._id = next(Flight._id_counter)
                 self.departure_airport = departure_airport
                 self.arrival_airport = arrival_airport
                 self.departure_time = departure_time
                 self.arrival_time = arrival_time
                 self.__available_seats = total_seats

    # Getter for flight ID (only if needed)
    def get_id(self):
        return self._id

    # Method to book a seat
    def book_seat(self):
        if self.__available_seats > 0:
            self.__available_seats -= 1
            print(f"Seat booked successfully. Remaining seats: {self.__available_seats}")
        else:
            print("No available seats to book.")

    # Method to cancel a reservation
    def cancel_reservation(self):
        if self.__available_seats < 100: # Assuming a max capacity of 100
            self.__available_seats += 1
            print(f"Reservation cancelled successfully. Remaining seats: {self.__available_seats}")
        else:
            print("No reservations to cancel.")

    # Method to get the remaining available seats
    def get_remaining_seats(self):
        return self.__available_seats

    # Method to display flight information
    def display_info(self):
        print(f"Flight ID: {self._id}")
        print(f"Departure Airport: {self.departure_airport}")
        print(f"Arrival Airport: {self.arrival_airport}")
        print(f"Departure Time: {self.departure_time}")
        print(f"Arrival Time: {self.arrival_time}")
        print(f"Available Seats: {self.__available_seats}")

class DomesticFlight(Flight):
    def __init__(self, departure_airport, arrival_airport, departure_time,
                 super().__init__(departure_airport, arrival_airport, departure_time)
                 self.domestic_airline = domestic_airline

    # Method to display domestic flight information
    def display_info(self):
        super().display_info()
        print(f"Domestic Airline: {self.domestic_airline}")

class InternationalFlight(Flight):
    def __init__(self, departure_airport, arrival_airport, departure_time,
                 super().__init__(departure_airport, arrival_airport, departure_time)
                 self.international_airline = international_airline
                 self.visa_required = visa_required

    # Method to display international flight information
    def display_info(self):
```

```
super().display_info()
print(f"International Airline: {self.international_airline}")
print(f"Visa Required: {'Yes' if self.visa_required else 'No'}")

# Test the Airline Reservation System
def test_airline_reservation_system():
    # Create flight instances
    domestic_flight = DomesticFlight(departure_airport="JFK", arrival_airport="LAX", departure_time="10:00", arrival_time="13:00", available_seats=50, airline="American Airlines")
    international_flight = InternationalFlight(departure_airport="JFK", arrival_airport="LHR", departure_time="18:00", arrival_time="06:00", available_seats=200, airline="British Airways", visa_required=True)

    # Display flight information
    print("Flight Information:")
    domestic_flight.display_info()
    print()
    international_flight.display_info()
    print()

    # Book and cancel reservations
    print("Booking and Canceling Reservations:")
    domestic_flight.book_seat()
    domestic_flight.cancel_reservation()
    domestic_flight.display_info()
    print()

    international_flight.book_seat()
    international_flight.cancel_reservation()
    international_flight.display_info()

# Run the test
test_airline_reservation_system()
```

```
Flight Information:
Flight ID: 1
Departure Airport: JFK
Arrival Airport: LAX
Departure Time: 10:00
Arrival Time: 13:00
Available Seats: 50
Domestic Airline: American Airlines

Flight ID: 2
Departure Airport: JFK
Arrival Airport: LHR
Departure Time: 18:00
Arrival Time: 06:00
Available Seats: 200
International Airline: British Airways
Visa Required: Yes

Booking and Canceling Reservations:
Seat booked successfully. Remaining seats: 49.
Reservation cancelled successfully. Remaining seats: 50.
Flight ID: 1
Departure Airport: JFK
Arrival Airport: LAX
Departure Time: 10:00
Arrival Time: 13:00
Available Seats: 50
Domestic Airline: American Airlines

Seat booked successfully. Remaining seats: 199.
No reservations to cancel.
Flight ID: 2
Departure Airport: JFK
Arrival Airport: LHR
Departure Time: 18:00
Arrival Time: 06:00
Available Seats: 199
International Airline: British Airways
Visa Required: Yes
```

11. Define a Python module named constants.py containing constants like pi and the speed of light.

```
# constants.py

# Mathematical constant for pi
PI = 3.141592653589793

# Speed of light in meters per second
SPEED_OF_LIGHT = 299792458
import constants

print("Value of PI:", constants.PI)
print("Speed of Light:", constants.SPEED_OF_LIGHT, "m/s")

Value of PI: 3.141592653589793
Speed of Light: 299792458 m/s
```

12. Write a Python module named calculator.py containing functions for addition, subtraction, multiplication, and division.

```
# calculator.py

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

def subtract(a, b):
    """Return the difference between a and b."""
    return a - b

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

def divide(a, b):
    """Return the quotient of a divided by b.

    Raises:
        ValueError: If b is zero, as division by zero is not allowed.
    """
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a / b
```

```
# import calculator

a = 10
b = 5

13. Implement a Python package structure for a project named ecommerce, containing
modules for product management and order processing.

print('Multiplication: ', calculator.multiply(a, b))

# ecommerce/product_management.py

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

Enter a prompt here



0 / 100