problems-1

November 2, 2024

[ ]:

```
#Name : G.Vishal sai
#class : K24BD
#Roll no. : 10
#reg.no. : 12406911
```

[1]:

```
#1.        Create a function that merges two lists and removes duplicates.
def merge_and_remove_duplicates(list1, list2):
    merged_list = list(set(list1 + list2))
    return merged_list

# Example usage
list1 = [1, 2, 3, 4]
list2 = [3, 4, 5, 6]
result = merge_and_remove_duplicates(list1, list2)
print(result)
```

```
[1, 2, 3, 4, 5, 6]
```

[3]:

```
#2.        Write a program that takes a list of names and counts how many names␣
 ↪start with a vowel
def count_names_starting_with_vowel(names):
    vowels = {'A', 'E', 'I', 'O', 'U'}
    count = sum(1 for name in names if name[0].upper() in vowels)
    return count

names = ["Alice", "Bob", "Eve", "Oscar", "Uma", "Charlie"]
result = count_names_starting_with_vowel(names)
print(f"Number of names starting with a vowel: {result}")
```

```
Number of names starting with a vowel: 4
```

[7]:

```
#3.        Create a function that returns the index of the first repeated␣
 ↪element in a tuple.
def first_repeated_index(t):
    seen = {}
    for index, value in enumerate(t):
        if value in seen:
            return seen[value]
        seen[value] = index
    return -1
t = (1, 2, 3, 2, 4, 5)
result = first_repeated_index(t)
print(f"The index of the first repeated element is: {result}")
```

```
The index of the first repeated element is: 1
```

[9]:

```
#4.         Write a program to remove empty strings from a list.
def remove_empty_strings(lst):
    return [string for string in lst if string]

list_with_empty_strings = ["hello", "", "world", "", "!", ""]
clean_list = remove_empty_strings(list_with_empty_strings)
print(clean_list)  # Output: ['hello', 'world', '!']
```

```
['hello', 'world', '!']
```

[11]:

```
#5.         Create a function that returns all unique elements from a list.
def get_unique_elements(lst):
    return list(set(lst))

sample_list = [1, 2, 2, 3, 4, 4, 5]
unique_elements = get_unique_elements(sample_list)
print(unique_elements)  # Output: [1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5]
```

[13]:

```
#6.         Write a function to convert a tuple of words into a set of words.
def tuple_to_set(words):
    return set(words)

words_tuple = ("apple", "banana", "cherry", "apple", "banana")
unique_words_set = tuple_to_set(words_tuple)
print(unique_words_set)
```

```
{'apple', 'cherry', 'banana'}
```

[15]:

```
#7.         Create a function that reverses the order of elements in a list␣
 ↪using loops.
def reverse_list(lst):
    reversed_list = []
    for i in range(len(lst) - 1, -1, -1):
        reversed_list.append(lst[i])
    return reversed_list

sample_list = [1, 2, 3, 4, 5]
reversed_list = reverse_list(sample_list)
```

```
print(reversed_list)
```

```
[5, 4, 3, 2, 1]
```

[17]:

```
#8.         Write a program to find all common elements between two sets
def find_common_elements(set1, set2):
    return set1.intersection(set2)

set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
common_elements = find_common_elements(set1, set2)
print(common_elements)
```

```
{4, 5}
```

[21]:

```
#9.         Create a function that checks if all elements in a list are strings.
def all_strings(lst):
    return all(isinstance(item, str) for item in lst)

sample_list = ["apple", "banana", "cherry"]
result = all_strings(sample_list)
print(result)
sample_list2 = ["apple", 42, "cherry"]
result2 = all_strings(sample_list2)
print(result2)
```

```
True
False
```

[23]:

```
#10.         Write a function to flatten a list of lists into a single list
def flatten_list(lst_of_lsts):
    flat_list = []
    for sublist in lst_of_lsts:
        for item in sublist:
            flat_list.append(item)
    return flat_list

nested_list = [[1, 2, 3], [4, 5], [6, 7, 8, 9]]
flattened = flatten_list(nested_list)
print(flattened)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[25]:

```
#11.         Write a program to concatenate multiple tuples into a single tuple.
def concatenate_tuples(*tuples):
    return sum(tuples, ())

tuple1 = (1, 2, 3)
```

```
tuple2 = (4, 5)
tuple3 = (6, 7, 8, 9)
result = concatenate_tuples(tuple1, tuple2, tuple3)
print(result)
```

```
(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

[27]:

```
#12.create a function that takes a set and returns a list of all elements␣
 ↪sorted alphabetically.
def sorted_list_from_set(s):
    return sorted(s)

sample_set = {'banana', 'apple', 'cherry', 'date'}
sorted_list = sorted_list_from_set(sample_set)
print(sorted_list)
```

```
['apple', 'banana', 'cherry', 'date']
```

[29]:

```
#13.        Write a program to remove the first and last elements from a list␣
 ↪and return the modified list.
def remove_first_last(lst):
    return lst[1:-1]

sample_list = [1, 2, 3, 4, 5]
modified_list = remove_first_last(sample_list)
print(modified_list)
```

```
[2, 3, 4]
```

[31]:

```
#14.        Create a function that converts a list of tuples into a list of␣
 ↪lists.
def tuples_to_lists(tuples_list):
    return [list(tup) for tup in tuples_list]

sample_list = [(1, 2), (3, 4), (5, 6)]
converted_list = tuples_to_lists(sample_list)
print(converted_list)
```

```
[[1, 2], [3, 4], [5, 6]]
```

[33]:

```
#15.        Write a function that counts the number of unique characters in a␣
 ↪list of strings.
def count_unique_characters(strings):
    unique_chars = set(''.join(strings))
    return len(unique_chars)

string_list = ["apple", "banana", "cherry"]
```

```
unique_count = count_unique_characters(string_list)
print(f"Number of unique characters: {unique_count}")
```

```
Number of unique characters: 10
```

[35]:

```
#16.        Create a function that checks if two sets are disjoint (i.e., have␣
 ↪no common elements).
def are_disjoint(set1, set2):
    return set1.isdisjoint(set2)

set1 = {1, 2, 3}
set2 = {4, 5, 6}
result = are_disjoint(set1, set2)
print(f"Are the sets disjoint? {result}")

set3 = {1, 2, 3}
set4 = {3, 4, 5}
result2 = are_disjoint(set3, set4)
print(f"Are the sets disjoint? {result2}")
```

```
Are the sets disjoint? True
Are the sets disjoint? False
```

[37]:

```
#17.        Write a function that returns the length of the longest string in a␣
 ↪list of strings.
def longest_string_length(strings):
    return max(len(string) for string in strings)

string_list = ["apple", "banana", "cherry", "date"]
longest_length = longest_string_length(string_list)
print(f"Length of the longest string: {longest_length}")
```

```
Length of the longest string: 6
```

[39]:

```
#18.        Create a function that checks if a tuple contains any duplicates.
def contains_duplicates(t):
    return len(t) != len(set(t))

sample_tuple = (1, 2, 3, 4, 2)
result = contains_duplicates(sample_tuple)
print(f"Contains duplicates? {result}")

sample_tuple2 = (1, 2, 3, 4, 5)
result2 = contains_duplicates(sample_tuple2)
print(f"Contains duplicates? {result2}")
```

```
Contains duplicates? True
Contains duplicates? False
```

[41]:

```
#19.        Write a program to extract the last element from each tuple in a␣
 ↪list of tuples.
def extract_last_elements(tuples_list):
    return [t[-1] for t in tuples_list]

sample_list = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
last_elements = extract_last_elements(sample_list)
print(last_elements)
```

```
[3, 6, 9]
```

[43]:

```
#20.        Create a function that returns the intersection of three sets.
def intersection_of_three_sets(set1, set2, set3):
    return set1.intersection(set2, set3)
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
set3 = {4, 6, 7, 8}
result = intersection_of_three_sets(set1, set2, set3)
print(f"The intersection of the three sets is: {result}")
```

```
The intersection of the three sets is: {4}
```

[45]:

```
#21.        Write a program to merge two dictionaries and remove duplicate␣
 ↪values.
def merge_and_remove_duplicates(dict1, dict2):
    merged_dict = {}
    for key, value in {**dict1, **dict2}.items():
        if value not in merged_dict.values():
            merged_dict[key] = value
    return merged_dict

dict1 = {'a': 1, 'b': 2, 'c': 3}
dict2 = {'d': 3, 'e': 5, 'f': 1}
result = merge_and_remove_duplicates(dict1, dict2)
print(result)
```

```
{'a': 1, 'b': 2, 'c': 3, 'e': 5}
```

[47]:

```
#22.        Create a function that finds the longest word in a list of strings.
def longest_word(words):
    return max(words, key=len)

words_list = ["apple", "banana", "cherry", "watermelon"]
longest = longest_word(words_list)
print(f"The longest word is: {longest}")
```

```
The longest word is: watermelon
```

[49]:

```
#23.        Write a program to find the difference between two sets (i.e.,␣
 ↪elements in the first set but not in the second).

def difference_between_sets(set1, set2):
    return set1 - set2

set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
result = difference_between_sets(set1, set2)
print(f"The difference between the two sets is: {result}")
```

```
The difference between the two sets is: {1, 2}
```

[51]:

```
#24.        Create a function that returns a list of all even-length words from␣
 ↪a tuple of words.
def even_length_words(words):
    return [word for word in words if len(word) % 2 == 0]

words_tuple = ("apple", "banana", "cherry", "date", "fig")
even_words = even_length_words(words_tuple)
print(f"Even-length words: {even_words}")
```

```
Even-length words: ['banana', 'cherry', 'date']
```

[53]:

```
#25.        Write a function to count how many times a specific word appears in␣
 ↪a list of sentences.
def count_word_in_sentences(word, sentences):
    word = word.lower()
    count = 0
    for sentence in sentences:
        words = sentence.lower().split()
        count += words.count(word)
    return count

sentences_list = [
    "The quick brown fox jumps over the lazy dog",
    "The quick brown fox is quick",
    "Lazy dogs are not quick like the fox"
]
word_to_count = "quick"
count = count_word_in_sentences(word_to_count, sentences_list)
print(f"The word '{word_to_count}' appears {count} times in the list of␣
 ↪sentences.")
```

```
The word 'quick' appears 4 times in the list of sentences.
```

[55]:

```python
#1. Create a class Car with attributes make, model, and year, and a method start_engine that prints a message.
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def start_engine(self):
        print(f"The {self.year} {self.make} {self.model}'s engine has started!")

my_car = Car("Toyota", "Corolla", 2022)
my_car.start_engine()
```

```
The 2022 Toyota Corolla's engine has started!
```

[57]:

```python
#2. Design a BankAccount class that has methods for deposit, withdraw, and check_balance.
class BankAccount:
    def __init__(self, account_holder, balance=0.0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited {amount}. New balance is {self.balance}.")
        else:
            print("Invalid deposit amount.")

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew {amount}. New balance is {self.balance}.")
        else:
            print("Invalid withdrawal amount or insufficient funds.")

    def check_balance(self):
        print(f"Account holder: {self.account_holder}")
        print(f"Current balance: {self.balance}")

my_account = BankAccount("John Doe", 1000)
my_account.deposit(500)
my_account.withdraw(200)
my_account.check_balance()
```

```
Deposited 500. New balance is 1500.
```

```
Withdrew 200. New balance is 1300.
Account holder: John Doe
Current balance: 1300
```

[59]:

```
#3. Create a Person class with attributes name and age, and a method introduce␣
 ↪that prints a personal introduction
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

person1 = Person("Alice", 30)
person1.introduce()
```

```
Hello, my name is Alice and I am 30 years old.
```

[61]:

```
#4. Implement an Employee class with attributes name and salary, and a method␣
 ↪to give a raise to the employee.
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def give_raise(self, amount):
        if amount > 0:
            self.salary += amount
            print(f"{self.name} received a raise of {amount}. New salary is␣
 ↪{self.salary}.")
        else:
            print("Raise amount must be positive.")

employee1 = Employee("John Doe", 50000)
employee1.give_raise(5000)
employee1.give_raise(-1000)
```

```
John Doe received a raise of 5000. New salary is 55000.
Raise amount must be positive.
```

[63]:

```
#5. Create a Library class that can add_book, remove_book, and list_books␣
 ↪available in the library.
class Library:
    def __init__(self):
        self.books = []
    def add_book(self, book):
        self.books.append(book)
        print(f"'{book}' has been added to the library.")

    def remove_book(self, book):
        if book in self.books:
            self.books.remove(book)
            print(f"'{book}' has been removed from the library.")
        else:
            print(f"'{book}' is not available in the library.")

    def list_books(self):
        if self.books:
            print("Books available in the library:")
            for book in self.books:
                print(f" - {book}")
        else:
            print("No books are currently available in the library.")

library = Library()
library.add_book("1984 by George Orwell")
library.add_book("To Kill a Mockingbird by Harper Lee")
library.list_books()
library.remove_book("1984 by George Orwell")
library.list_books()
```

```
'1984 by George Orwell' has been added to the library.
'To Kill a Mockingbird by Harper Lee' has been added to the library.
Books available in the library:
 - 1984 by George Orwell
 - To Kill a Mockingbird by Harper Lee
'1984 by George Orwell' has been removed from the library.
Books available in the library:
 - To Kill a Mockingbird by Harper Lee
```

[65]:

```
#6. Design a Student class that stores a student's name, roll number, and␣
 ↪grades. Add methods to calculate and display the GPA.
class Student:
    def __init__(self, name, roll_number):
        self.name = name
        self.roll_number = roll_number
        self.grades = []

    def add_grade(self, grade):
        self.grades.append(grade)

    def calculate_gpa(self):
        if not self.grades:
            return 0
        return sum(self.grades) / len(self.grades)

    def display_gpa(self):
        gpa = self.calculate_gpa()
        print(f"Student: {self.name}, Roll Number: {self.roll_number}, GPA:␣
 ↪{gpa:.2f}")

student1 = Student("Alice", 101)
student1.add_grade(4.0)
student1.add_grade(3.7)
student1.add_grade(3.8)
student1.display_gpa()
```

```
Student: Alice, Roll Number: 101, GPA: 3.83
```

[67]:

```
#7. Implement a Laptop class that has attributes brand, model, and price, and␣
 ↪methods to display the details of the laptop.
class Laptop:
    def __init__(self, brand, model, price):
        self.brand = brand
        self.model = model
        self.price = price

    def display_details(self):
        print(f"Laptop Details:\nBrand: {self.brand}\nModel: {self.
 ↪model}\nPrice: ${self.price:.2f}")

my_laptop = Laptop("Dell", "XPS 13", 999.99)
my_laptop.display_details()
```

```
Laptop Details:
Brand: Dell
Model: XPS 13
Price: $999.99
```

[69]:

```
#8. Create a Restaurant class with methods to add_menu_item, remove_menu_item,␣
 ↪and display_menu.
class Restaurant:
    def __init__(self, name):
        self.name = name
        self.menu = {}

    def add_menu_item(self, item, price):
        self.menu[item] = price
        print(f"'{item}' has been added to the menu at ${price:.2f}.")
    def remove_menu_item(self, item):
        if item in self.menu:
            del self.menu[item]
            print(f"'{item}' has been removed from the menu.")
        else:
            print(f"'{item}' is not available in the menu.")

    def display_menu(self):
        if self.menu:
            print(f"Menu for {self.name}:")
            for item, price in self.menu.items():
                print(f" - {item}: ${price:.2f}")
        else:
            print("The menu is currently empty.")

restaurant = Restaurant("Gourmet Paradise")
restaurant.add_menu_item("Pasta", 12.99)
restaurant.add_menu_item("Pizza", 15.99)
restaurant.display_menu()
restaurant.remove_menu_item("Pasta")
restaurant.display_menu()
```

```
'Pasta' has been added to the menu at $12.99.
'Pizza' has been added to the menu at $15.99.
Menu for Gourmet Paradise:
 - Pasta: $12.99
 - Pizza: $15.99
'Pasta' has been removed from the menu.
Menu for Gourmet Paradise:
 - Pizza: $15.99
```

[71]:

```
#9. Write a Teacher class that stores the teacher's name, subject, and years of␣
 ↪experience. Add a method to display the teacher's details.
class Teacher:
    def __init__(self, name, subject, years_of_experience):
        self.name = name
        self.subject = subject
        self.years_of_experience = years_of_experience

    def display_details(self):
        print(f"Teacher Details:\nName: {self.name}\nSubject: {self.
 ↪subject}\nYears of Experience: {self.years_of_experience}")

teacher1 = Teacher("Mr. Smith", "Mathematics", 10)
teacher1.display_details()
```

```
Teacher Details:
Name: Mr. Smith
Subject: Mathematics
Years of Experience: 10
```

[1]:

```python
#10. Design a Shape class with methods for calculating the area and perimeter. Create subclasses Rectangle and Circle that inherit from Shape.
import math

class Shape:
    def area(self):
        raise NotImplementedError("Subclasses should implement this method.")

    def perimeter(self):
        raise NotImplementedError("Subclasses should implement this method.")

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius ** 2

    def perimeter(self):
        return 2 * math.pi * self.radius

rect = Rectangle(4, 5)
print(f"Rectangle Area: {rect.area()}")
print(f"Rectangle Perimeter: {rect.perimeter()}")

circle = Circle(3)
print(f"Circle Area: {circle.area():.2f}")
print(f"Circle Perimeter: {circle.perimeter():.2f}")
```

```
Rectangle Area: 20
Rectangle Perimeter: 18
Circle Area: 28.27
Circle Perimeter: 18.85
```

[7]:

```
#11. Create a Book class that stores the title, author, and ISBN number. Add␣
↪methods to check availability and issue a book.

class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.is_available = True

    def check_availability(self):
        return self.is_available

    def issue_book(self):
        if self.is_available:
            self.is_available = False
            return f"The book '{self.title}' has been issued."
        else:
            return f"Sorry, the book '{self.title}' is currently unavailable."

    def return_book(self):
        self.is_available = True
        return f"The book '{self.title}' has been returned and is now available.
↪"


book1 = Book("1984", "George Orwell", "1234567890")
print(book1.check_availability())
print(book1.issue_book())
print(book1.check_availability())
print(book1.return_book())
print(book1.check_availability())
```

```
True
The book '1984' has been issued.
False
The book '1984' has been returned and is now available.
True
```

[9]:

```
#12. Write a Pet class with attributes for name, species, and age. Add a method␣
↪to simulate the pet making a sound.
class Pet:
    def __init__(self, name, species, age):
        self.name = name
        self.species = species
        self.age = age

    def make_sound(self):
        if self.species == "dog":
            return "Woof!"
        elif self.species == "cat":
            return "Meow!"
        elif self.species == "bird":
            return "Tweet!"
        else:
            return "Unknown sound"

pet1 = Pet("Buddy", "dog", 3)
print(f"{pet1.name} the {pet1.species} says: {pet1.make_sound()}")

pet2 = Pet("Whiskers", "cat", 2)
print(f"{pet2.name} the {pet2.species} says: {pet2.make_sound()}")

pet3 = Pet("Tweety", "bird", 1)
print(f"{pet3.name} the {pet3.species} says: {pet3.make_sound()}")
```

```
Buddy the dog says: Woof!
Whiskers the cat says: Meow!
Tweety the bird says: Tweet!
```

[11]:

```
#13. Design a Movie class that stores the title, director, and rating. Add␣
 ↪methods to display the movie details and check its rating.
class Movie:
    def __init__(self, title, director, rating):
        self.title = title
        self.director = director
        self.rating = rating

    def display_details(self):
        print(f"Title: {self.title}\nDirector: {self.director}\nRating: {self.
 ↪rating}")

    def check_rating(self):
        if self.rating >= 7:
            return "Highly Rated"
        elif 4 <= self.rating < 7:
            return "Moderately Rated"
        else:
            return "Poorly Rated"

movie1 = Movie("Inception", "Christopher Nolan", 8.8)
movie1.display_details()
```

```
print(movie1.check_rating())

movie2 = Movie("The Room", "Tommy Wiseau", 3.7)
movie2.display_details()
print(movie2.check_rating())
```

```
Title: Inception
Director: Christopher Nolan
Rating: 8.8
Highly Rated
Title: The Room
Director: Tommy Wiseau
Rating: 3.7
Poorly Rated
```

[13]:

```
#14. Implement a Ticket class for a train ticket booking system with attributes␣
 ↪passenger_name, train_number, and seat_number. Add methods for ticket␣
 ↪booking and cancellation.
class Ticket:
    def __init__(self, passenger_name, train_number, seat_number):
        self.passenger_name = passenger_name
        self.train_number = train_number
        self.seat_number = seat_number
        self.is_booked = False

    def book_ticket(self):
        if not self.is_booked:
            self.is_booked = True
            return f"Ticket for {self.passenger_name} on train {self.
 ↪train_number} seat {self.seat_number} has been booked."
        else:
            return "This ticket is already booked."

    def cancel_ticket(self):
        if self.is_booked:
            self.is_booked = False
            return f"Ticket for {self.passenger_name} on train {self.
 ↪train_number} seat {self.seat_number} has been cancelled."
        else:
            return "This ticket is not booked yet."


ticket1 = Ticket("Alice", "12345", "A1")
print(ticket1.book_ticket())
print(ticket1.cancel_ticket())
print(ticket1.book_ticket())
print(ticket1.book_ticket())
print(ticket1.cancel_ticket())
```

```
Ticket for Alice on train 12345 seat A1 has been booked.
Ticket for Alice on train 12345 seat A1 has been cancelled.
Ticket for Alice on train 12345 seat A1 has been booked.
This ticket is already booked.
Ticket for Alice on train 12345 seat A1 has been cancelled.
```

[15]:

```
#15. Create a Product class with attributes name, price, and stock_quantity.␣
 ↪Add methods for purchase and checking stock levels.
class Product:
    def __init__(self, name, price, stock_quantity):
        self.name = name
        self.price = price
        self.stock_quantity = stock_quantity

    def purchase(self, quantity):
        if quantity <= self.stock_quantity:
            self.stock_quantity -= quantity
            total_cost = self.price * quantity
            return f"Purchased {quantity} units of {self.name}. Total cost:␣
 ↪${total_cost:.2f}. Stock left: {self.stock_quantity}."
        else:
            return f"Insufficient stock for {self.name}. Available stock: {self.
 ↪stock_quantity}."

    def check_stock(self):
        return f"Current stock level for {self.name}: {self.stock_quantity}␣
 ↪units."

product1 = Product("Laptop", 999.99, 10)
print(product1.check_stock())
print(product1.purchase(3))
print(product1.check_stock())
print(product1.purchase(8))
```

```
Current stock level for Laptop: 10 units.
Purchased 3 units of Laptop. Total cost: $2999.97. Stock left: 7.
Current stock level for Laptop: 7 units.
Insufficient stock for Laptop. Available stock: 7.
```

[17]:

```
#16. Design a Course class that stores the course name, course code, and a list␣
 ↪of students enrolled. Add methods to enroll a student and display all␣
 ↪enrolled students.
class Course:
    def __init__(self, course_name, course_code):
        self.course_name = course_name
        self.course_code = course_code
        self.students = []

    def enroll_student(self, student_name):
        if student_name not in self.students:
            self.students.append(student_name)
            return f"{student_name} has been enrolled in {self.course_name}."
        else:
            return f"{student_name} is already enrolled in {self.course_name}."

    def display_students(self):
        if self.students:
            print(f"Students enrolled in {self.course_name}:")
            for student in self.students:
                print(f" - {student}")
        else:
            print(f"No students are currently enrolled in {self.course_name}.")

course1 = Course("Introduction to Python", "CS101")
print(course1.enroll_student("Alice"))
print
```

```
Alice has been enrolled in Introduction to Python.
```

[17]:

```
<function print(*args, sep=' ', end='\n', file=None, flush=False)>
```

[19]:

```
#17. Implement a Vehicle class with attributes make, model, and mileage, and␣
 ↪methods for driving and refueling the vehicle.
class Vehicle:
    def __init__(self, make, model, mileage):
        self.make = make
        self.model = model
        self.mileage = mileage
        self.fuel = 100

    def drive(self, distance):
        fuel_needed = distance / 10
        if self.fuel >= fuel_needed:
            self.mileage += distance
            self.fuel -= fuel_needed
            return f"Drove {distance} miles. New mileage: {self.mileage}. Fuel␣
 ↪left: {self.fuel}%."
        else:
            return "Not enough fuel to drive the distance."

    def refuel(self, amount):
        if amount > 0:
```

```python
            self.fuel = min(self.fuel + amount, 100)
            return f"Refueled {amount}%. Fuel level: {self.fuel}%."
        else:
            return "Invalid amount. Please refuel with a positive amount."


vehicle = Vehicle("Toyota", "Camry", 15000)
print(vehicle.drive(50))
print(vehicle.refuel(20))
print(vehicle.drive(500))
```

```
Drove 50 miles. New mileage: 15050. Fuel left: 95.0%.
Refueled 20%. Fuel level: 100%.
Drove 500 miles. New mileage: 15550. Fuel left: 50.0%.
```

[21]:

```python
#18. Write a Shop class that has a list of products. Add methods to add a product, remove a product, and display all available products.
class Shop:
    def __init__(self):
        self.products = []

    def add_product(self, product):
        self.products.append(product)
        print(f"'{product}' has been added to the shop.")

    def remove_product(self, product):
        if product in self.products:
            self.products.remove(product)
            print(f"'{product}' has been removed from the shop.")
        else:
            print(f"'{product}' is not available in the shop.")

    def display_products(self):
        if self.products:
            print("Products available in the shop:")
            for product in self.products:
                print(f" - {product}")
        else:
            print("No products are currently available in the shop.")

shop = Shop()
shop.add_product("Laptop")
shop.add_product("Smartphone")
shop.display_products()
shop.remove_product("Laptop")
shop.display_products()
shop.remove_product("Tablet")
```

```
'Laptop' has been added to the shop.
'Smartphone' has been added to the shop.
Products available in the shop:
 - Laptop
 - Smartphone
'Laptop' has been removed from the shop.
Products available in the shop:
 - Smartphone
'Tablet' is not available in the shop.
```

[23]:

```
#19. Create a Department class that stores the department name and a list of␣
 ↪employees. Add methods to add an employee and display the details of all␣
 ↪employees.
class Department:
    def __init__(self, department_name):
        self.department_name = department_name
        self.employees = []

    def add_employee(self, employee_name):
        self.employees.append(employee_name)
        print(f"{employee_name} has been added to the {self.department_name}␣
 ↪department.")

    def display_employees(self):
        if self.employees:
            print(f"Employees in {self.department_name} department:")
            for employee in self.employees:
                print(f" - {employee}")
        else:
            print(f"No employees are currently in the {self.department_name}␣
 ↪department.")

dept = Department("Human Resources")
dept.add_employee("Alice Johnson")
dept.add_employee("Bob Smith")
dept.display_employees()
```

```
Alice Johnson has been added to the Human Resources department.
Bob Smith has been added to the Human Resources department.
Employees in Human Resources department:
 - Alice Johnson
 - Bob Smith
```

[25]:

```
#20. Design a User class that stores the user's username, email, and password.␣
 ↪Add methods for user registration, login, and logout.
class User:
    def __init__(self, username, email, password):
        self.username = username
        self.email = email
        self.password = password
        self.is_logged_in = False

    def register(self):
        # Registration logic would go here
        print(f"User {self.username} registered with email {self.email}.")

    def login(self, password):
        if self.password == password:
            self.is_logged_in = True
            print(f"User {self.username} logged in successfully.")
        else:
            print("Incorrect password. Login failed.")

    def logout(self):
        if self.is_logged_in:
            self.is_logged_in = False
            print(f"User {self.username} logged out successfully.")
        else:
            print(f"User {self.username} is not logged in.")

user1 = User("john_doe", "john@example.com", "securepassword123")
user1.register()
user1.login("securepassword123")
user1.logout()
```

```
User john_doe registered with email john@example.com.
User john_doe logged in successfully.
User john_doe logged out successfully.
```

[27]:

```
#21. Implement an Order class for an online shopping system. The class should␣
 ↪store the order ID, product list, and order status. Add methods to place an␣
 ↪order and check order status.
class Order:
    def __init__(self, order_id, product_list):
        self.order_id = order_id
        self.product_list = product_list
        self.order_status = "Pending"

    def place_order(self):
        self.order_status = "Placed"
        return f"Order ID {self.order_id} has been placed. Products: {', '.
 ↪join(self.product_list)}"

    def check_order_status(self):
        return f"Order ID {self.order_id} is currently {self.order_status}."

order1 = Order("12345", ["Laptop", "Smartphone", "Headphones"])
print(order1.place_order())
print(order1.check_order_status())
```

```
Order ID 12345 has been placed. Products: Laptop, Smartphone, Headphones
Order ID 12345 is currently Placed.
```

[29]:

```
#22. Write a Game class that stores the game's title and genre. Add a method to␣
 ↪start the game and display the game's details.
class Game:
    def __init__(self, title, genre):
        self.title = title
        self.genre = genre

    def start_game(self):
        print(f"Starting the game: {self.title}")

    def display_details(self):
        print(f"Game Details:\nTitle: {self.title}\nGenre: {self.genre}")

game1 = Game("The Legend of Zelda", "Adventure")
game1.display_details()
game1.start_game()
```

```
Game Details:
Title: The Legend of Zelda
Genre: Adventure
Starting the game: The Legend of Zelda
```

[31]:

```
#23. Create a Reservation class for a restaurant reservation system with␣
 ↪attributes customer_name, table_number, and time. Add methods to reserve and␣
 ↪cancel a reservation
class Reservation:
    def __init__(self, customer_name, table_number, time):
        self.customer_name = customer_name
        self.table_number = table_number
        self.time = time
        self.is_reserved = False

    def reserve(self):
        if not self.is_reserved:
            self.is_reserved = True
            return f"Reservation confirmed for {self.customer_name} at table␣
 ↪{self.table_number} at {self.time}."
        else:
            return "This table is already reserved."

    def cancel(self):
        if self.is_reserved:
            self.is_reserved = False
            return f"Reservation for {self.customer_name} at table {self.
 ↪table_number} at {self.time} has been cancelled."
        else:
            return "No reservation to cancel."

reservation1 = Reservation("John Doe", 5, "7:00 PM")
print(reservation1.reserve())
print(reservation1.cancel())
print(reservation1.reserve())
print(reservation1.reserve())
```

```
Reservation confirmed for John Doe at table 5 at 7:00 PM.
Reservation for John Doe at table 5 at 7:00 PM has been cancelled.
Reservation confirmed for John Doe at table 5 at 7:00 PM.
This table is already reserved.
```

[33]:

```
#24. Design a Building class that has a name, location, and number_of_floors.␣
 ↪Add methods to display building details and calculate the total area of the␣
 ↪building.
class Building:
    def __init__(self, name, location, number_of_floors, floor_area):
        self.name = name
        self.location = location
        self.number_of_floors = number_of_floors
        self.floor_area = floor_area

    def display_details(self):
        print(f"Building Details:\nName: {self.name}\nLocation: {self.
 ↪location}\nNumber of Floors: {self.number_of_floors}")

    def calculate_total_area(self):
        total_area = self.number_of_floors * self.floor_area
        return total_area

building1 = Building("Sky Tower", "Downtown", 10, 500)
building1.display_details()
print(f"Total Area: {building1.calculate_total_area()} sq ft")
```

```
Building Details:
Name: Sky Tower
Location: Downtown
Number of Floors: 10
```

```
Total Area: 5000 sq ft
```

[35]:

```python
#25. Implement a Warehouse class that stores a list of items with their␣
 ↪quantities. Add methods to add items, remove items, and display current␣
 ↪stock levels.
class Warehouse:
    def __init__(self):
        self.inventory = {}

    def add_item(self, item, quantity):
        if item in self.inventory:
            self.inventory[item] += quantity
        else:
            self.inventory[item] = quantity
        print(f"Added {quantity} of {item}. Current stock: {self.
 ↪inventory[item]}")

    def remove_item(self, item, quantity):
        if item in self.inventory:
            if self.inventory[item] >= quantity:
                self.inventory[item] -= quantity
                if self.inventory[item] == 0:
                    del self.inventory[item]
                print(f"Removed {quantity} of {item}.")
            else:
                print(f"Cannot remove {quantity} of {item}. Only {self.
 ↪inventory[item]} available.")
        else:
            print(f"{item} not found in inventory.")

    def display_stock(self):
        if self.inventory:
            print("Current stock levels:")
            for item, quantity in self.inventory.items():
                print(f" - {item}: {quantity}")
        else:
            print("The warehouse is empty.")

warehouse = Warehouse()
warehouse.add_item("Laptop", 10)
warehouse.add_item("Smartphone", 5)
warehouse.display_stock()
warehouse.remove_item("Laptop", 3)
warehouse.display_stock()
warehouse.remove_item("Smartphone", 6)
warehouse.display_stock()
```

```
Added 10 of Laptop. Current stock: 10
Added 5 of Smartphone. Current stock: 5
Current stock levels:
 - Laptop: 10
 - Smartphone: 5
Removed 3 of Laptop.
Current stock levels:
 - Laptop: 7
 - Smartphone: 5
Cannot remove 6 of Smartphone. Only 5 available.
Current stock levels:
 - Laptop: 7
 - Smartphone: 5
```

[ ]: