T. chandana

2403A51238

AI ASSISTED CODING – ASSIGNMENT 6.1

Task 1 :

(Classes – Employee Management)

Use AI to create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary.

Prompt –

create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary.

Code –

Employee Class

```python
class Employee:
    """
    A class to represent an employee.
    """

    def __init__(self, name, id_number, salary):
        """
        Initializes the Employee object with a name, ID, and salary.

        Args:
            name (str): The employee's name.
            id_number (str): The employee's ID number.
            salary (float): The employee's monthly salary.
        """
        self.name = name
        self.id = id_number
        self.salary = salary

    def calculate_yearly_salary(self):
        """
        Calculates and returns the employee's yearly salary.

        Returns:
            float: The calculated yearly salary.
        """
        return self.salary * 12

# Example Usage:
if __name__ == "__main__":
    # Create an instance of the Employee class
    employee1 = Employee("Alice Johnson", "E12345", 5000)
```

```python
# Calculate and print the yearly salary
yearly_salary = employee1.calculate_yearly_salary()
print(f"Yearly Salary: ${yearly_salary:,.2f}")

print("\n")

# Another example
employee2 = Employee("Bob Smith", "E67890", 7500)
print(f"Employee Name: {employee2.name}")
print(f"Employee ID: {employee2.id}")
print(f"Monthly Salary: ${employee2.salary:,.2f}")
print(f"Yearly Salary: ${employee2.calculate_yearly_salary():,.2f}")
```

Output –

```
[1]  ✓ 0.0s
...  Employee Name: Alice Johnson
     Employee ID: E12345
     Monthly Salary: $5,000.00
     Yearly Salary: $60,000.00


     Employee Name: Bob Smith
     Employee ID: E67890
     Monthly Salary: $7,500.00
     Yearly Salary: $90,000.00
```

Asking ai to Add a method to give a bonus and recalculate salary.

Library Class

```python
class Library:
    """
    A class to represent a simple library management system.

    Attributes:
        books (list): A list to store the titles of books available in the library.
    """
    def __init__(self):
        """
        Initializes the Library class with an empty list of books.
        """
        self.books = []

    def add_book(self, book_title):
        """
        Adds a new book to the library's collection.

        Args:
            book_title (str): The title of the book to add.
        """
        self.books.append(book_title)
        print(f"'{book_title}' has been added to the library.")
```

```python
    def issue_book(self, book_title):
        """
        Issues a book to a user by removing it from the collection.

        Args:
            book_title (str): The title of the book to issue.
        """
        if book_title in self.books:
            self.books.remove(book_title)
            print(f"'{book_title}' has been successfully issued.")
        else:
            print(f"Sorry, '{book_title}' is not available in the library.")

    def display_books(self):
        """
        Displays all the books currently available in the library.
        """
        if not self.books:
            print("The library currently has no books.")
        else:
            print("Current books in the library:")
            for book in self.books:
                print(f"- {book}")

# Example usage of the Library class
if __name__ == "__main__":
    # Create an instance of the Library
    my_library = Library()

    # Add some books
    my_library.add_book("The Hobbit")
    my_library.add_book("1984")
    my_library.add_book("Dune")

    # Display the current books
    print("\n--- Displaying Books ---")
    my_library.display_books()

    # Issue a book that is available
    print("\n--- Issuing '1984' ---")
    my_library.issue_book("1984")

    # Try to issue a book that is not available
    print("\n--- Issuing a Non-existent Book ---")
    my_library.issue_book("Brave New World")

    # Display the updated list of books
    print("\n--- Displaying Updated Books ---")
    my_library.display_books()
```

Task 2 :

(Loops – Automorphic Numbers in a Range)

Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.

Prompt –

Asking ai generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.

Code –

Automorphic Numbers

```python
def is_automorphic(n):
    """
    Checks if a number is an automorphic number.

    An automorphic number is a number whose square ends in the number itself.
    For example, 5 is an automorphic number because 5*5 = 25, and 25 ends in 5.
    25 is an automorphic number because 25*25 = 625, and 625 ends in 25.

    Args:
        n (int): The number to check.

    Returns:
        bool: True if the number is automorphic, False otherwise.
    """
    if n < 0:
        return False

    square = n * n

    # Convert the number and its square to strings to easily check the ending
    str_n = str(n)
    str_square = str(square)

    # Check if the string representation of the square ends with the number's string
    return str_square.endswith(str_n)

def display_automorphic_numbers(start, end):
    """
    Displays all automorphic numbers within a given range using a for loop.

    Args:
        start (int): The starting number of the range (inclusive).
        end (int): The ending number of the range (inclusive).
    """
    print(f"Automorphic numbers between {start} and {end}:")
    automorphic_found = False

    # Use a for loop to iterate through the range
    for number in range(start, end + 1):
        if is_automorphic(number):
            print(number)
            automorphic_found = True

    if not automorphic_found:
        print("No automorphic numbers found in this range.")

# Main part of the script
if __name__ == "__main__":
    display_automorphic_numbers(1, 1000)
```

Output –

```
[10]  ✓ 0.0s
...  Automorphic numbers between 1 and 1000:
     1
     5
     6
     25
     76
     376
     625
```

regenerateing using a while loop and compare both implementations.

Automorphic Numbers

```python
def is_automorphic(n):
    """
    Checks if a number is an automorphic number.

    An automorphic number is a number whose square ends in the number itself.
    For example, 5 is an automorphic number because 5*5 = 25, and 25 ends in 5.
    25 is an automorphic number because 25*25 = 625, and 625 ends in 25.

    Args:
        n (int): The number to check.

    Returns:
        bool: True if the number is automorphic, False otherwise.
    """
    if n < 0:
        return False

    square = n * n

    # Convert the number and its square to strings to easily check the ending
    str_n = str(n)
    str_square = str(square)

    # Check if the string representation of the square ends with the number's string
    return str_square.endswith(str_n)

def display_automorphic_numbers(start, end):
    """
    Displays all automorphic numbers within a given range using a while loop.

    Args:
        start (int): The starting number of the range (inclusive).
        end (int): The ending number of the range (inclusive).
    """
    print(f"Automorphic numbers between {start} and {end}:")
    automorphic_found = False

    # Use a while loop instead of a for loop
    number = start
    while number <= end:
        if is_automorphic(number):
            print(number)
            automorphic_found = True
        number += 1

    if not automorphic_found:
        print("No automorphic numbers found in this range.")

# Main part of the script
if __name__ == "__main__":
    display_automorphic_numbers(1, 1000)

```

Both give same output and work almost the same.

Task 3:

(Conditional Statements – Online Shopping Feedback Classification)

Ask AI to write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5).

Prompt –

write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5).

Code –

Online Feedback Classifier

```python
def classify_feedback(rating):
    """
    Classifies online shopping feedback based on a numerical rating (1-5)
    using nested if-elif-else conditions.

    Args:
        rating (int): The numerical rating from 1 to 5.

    Returns:
        str: The classification of the feedback ("Positive", "Neutral", "Negative"),
             or "Invalid Rating" if the input is outside the expected range.
    """
    if rating >= 1 and rating <= 5:
        if rating >= 4:
            # Ratings of 4 and 5 are considered positive
            return "Positive"
        elif rating == 3:
            # A rating of 3 is considered neutral
            return "Neutral"
        else:
            # Ratings of 1 and 2 are considered negative
            return "Negative"
    else:
        return "Invalid Rating"

# Example usage to demonstrate the function
if __name__ == "__main__":
    test_ratings = [5, 4, 3, 2, 1, 0, 6]

    print("Classifying feedback based on different ratings:")
    for r in test_ratings:
        classification = classify_feedback(r)
        print(f"Rating {r} is classified as: {classification}")
```

Output –

[3] ✓ 0.0s

```
Classifying feedback based on different ratings:
Rating 5 is classified as: Positive
Rating 4 is classified as: Positive
Rating 3 is classified as: Neutral
Rating 2 is classified as: Negative
Rating 1 is classified as: Negative
Rating 0 is classified as: Invalid Rating
Rating 6 is classified as: Invalid Rating
```

Asking ai to rewrite the code using dictionary-based or match-case structure.

Online Feedback Classifier

```python
def classify_feedback(rating):
    """
    Classifies online shopping feedback as 'Positive', 'Neutral', or 'Negative'
    based on a numerical rating (1-5) using a match-case structure.

    Args:
        rating (int): The numerical rating from 1 to 5.

    Returns:
        str: The classification of the feedback ('Positive', 'Neutral', 'Negative',
             or 'Invalid Rating' if the input is out of range).
    """
    match rating:
        case 5 | 4:
            return "Positive"
        case 3:
            return "Neutral"
        case 2 | 1:
            return "Negative"
        case _:
            return "Invalid Rating"

# Example usage
if __name__ == "__main__":
    ratings_to_test = [5, 4, 3, 2, 1, 0, 6]

    print("Classifying feedback ratings:")
    for rating in ratings_to_test:
        classification = classify_feedback(rating)
        print(f"Rating {rating}: {classification}")

```

Task 4:

(Loops – Prime Numbers in a Range)

Generate a function using AI that displays all prime numbers within a user-specified range (e.g., 1 to 500).

Prompt –

Generate a function using AI that displays all prime numbers within a user-specified range (e.g., 1 to 500).

Code –

Prime Numbers Finder

```python
def is_prime(n):
    """
    Checks if a number is a prime number.

    A prime number is a natural number greater than 1 that has no
    positive divisors other than 1 and itself.

    Args:
        n (int): The number to check.

    Returns:
        bool: True if the number is prime, False otherwise.
    """
    # Prime numbers are greater than 1
    if n <= 1:
        return False
    # 2 is the only even prime number
    if n == 2:
        return True
    # All other even numbers are not prime
    if n % 2 == 0:
        return False
    # Check for divisors from 3 up to the square root of n
    # (stepping by 2 to check only odd numbers)
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2
    return True

def display_prime_numbers_in_range(start, end):
    """
    Displays all prime numbers within a given range.

    Args:
        start (int): The starting number of the range (inclusive).
        end (int): The ending number of the range (inclusive).
    """
```

```python
    if start > end:
        print("Invalid range: start number must be less than or equal to the end number.")
        return

    print(f"Prime numbers between {start} and {end}:")
    primes_found = False
    for number in range(start, end + 1):
        if is_prime(number):
            print(number, end=" ")
            primes_found = True

    if not primes_found:
        print("No prime numbers found in this range.")
    print("\n")

# Example Usage:
if __name__ == "__main__":
    # Display primes from 1 to 500
    display_prime_numbers_in_range(1, 500)

    # You can also test with a different range
    display_prime_numbers_in_range(500, 600)
```

Output –

```
[7]  ✓ 0.0s                                                                    Python
...  Prime numbers between 1 and 500:
     2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223

     Prime numbers between 500 and 600:
     503 509 521 523 541 547 557 563 569 571 577 587 593 599
```

Asking ai to regenerate an optimized version (e.g., using the square root method).

Prime Numbers Finder

```python
def is_prime(number):
    """
    Checks if a number is a prime number.

    A prime number is a natural number greater than 1 that has no positive
    divisors other than 1 and itself.

    Args:
        number (int): The number to check for primality.

    Returns:
        bool: True if the number is prime, False otherwise.
    """
    # Prime numbers are greater than 1. Any number less than or equal to 1 is not prime.
    if number <= 1:
        return False

    # We only need to check for divisors up to the square root of the number.
    # If a number has a divisor greater than its square root, it must also
    # have a divisor smaller than its square root.
    for i in range(2, int(number**0.5) + 1):
        # If the number is divisible by any 'i', it is not prime.
        if number % i == 0:
            return False

    # If the loop completes without finding any divisors, the number is prime.
    return True

def display_prime_numbers_in_range(start, end):
    """
    Finds and displays all prime numbers within a user-specified range.

    The function iterates through each number in the range and uses a helper
    function, is_prime(), to determine if the number should be printed.

    Args:
        start (int): The beginning of the range (inclusive).
        end (int): The end of the range (inclusive).
    """
    print(f"Finding prime numbers between {start} and {end}...")
    found_primes = []

    # Iterate through every number from the start of the range to the end.
    for num in range(start, end + 1):
        if is_prime(num):
            found_primes.append(num)

    # Display the results in a clear, formatted way.
    if found_primes:
        print("The prime numbers are:")
        print(found_primes)
    else:
        print("No prime numbers were found in this range.")

# Example usage: The main execution block of the script.
if __name__ == "__main__":
    # You can change these values to test different ranges.
    range_start = 1
    range_end = 500
    display_prime_numbers_in_range(range_start, range_end)
```

Task 5:

(Classes – Library System)

Use AI to build a Library class with methods to add_book(), issue_book(), and display_books().

Prompt –

build a Library class with methods to add_book(), issue_book(), and display_books().

Code –

Library Class

```python
class Library:
    """
    A class to represent a simple library management system.

    Attributes:
        books (list): A list to store the titles of books available in the library.
    """
    def __init__(self):
        """
        Initializes the Library class with an empty list of books.
        """
        self.books = []
```

```python

    def add_book(self, book_title):
        """
        Adds a new book to the library's collection.

        Args:
            book_title (str): The title of the book to add.
        """
        self.books.append(book_title)
        print(f"'{book_title}' has been added to the library.")

    def issue_book(self, book_title):
        """
        Issues a book to a user by removing it from the collection.

        Args:
            book_title (str): The title of the book to issue.
        """
        if book_title in self.books:
            self.books.remove(book_title)
            print(f"'{book_title}' has been successfully issued.")
        else:
            print(f"Sorry, '{book_title}' is not available in the library.")

    def display_books(self):
        """
        Displays all the books currently available in the library.
        """
        if not self.books:
            print("The library currently has no books.")
        else:
            print("Current books in the library:")
            for book in self.books:
                print(f"- {book}")

# Example usage of the Library class
if __name__ == "__main__":
    # Create an instance of the Library
    my_library = Library()

```

```python
    # Add some books
    my_library.add_book("The Hobbit")
    my_library.add_book("1984")
    my_library.add_book("Dune")

    # Display the current books
    print("\n--- Displaying Books ---")
    my_library.display_books()

    # Issue a book that is available
    print("\n--- Issuing '1984' ---")
    my_library.issue_book("1984")

    # Try to issue a book that is not available
    print("\n--- Issuing a Non-existent Book ---")
    my_library.issue_book("Brave New World")

    # Display the updated list of books
    print("\n--- Displaying Updated Books ---")
    my_library.display_books()

```

Output –

```
[9]  ✓ 0.0s
...  'The Hobbit' has been added to the library.
     '1984' has been added to the library.
     'Dune' has been added to the library.

     --- Displaying Books ---
     Current books in the library:
     - The Hobbit
     - 1984
     - Dune

     --- Issuing '1984' ---
     '1984' has been successfully issued.

     --- Issuing a Non-existent Book ---
     Sorry, 'Brave New World' is not available in the library.

     --- Displaying Updated Books ---
     Current books in the library:
     - The Hobbit
     - Dune
```

Asing ai to add comments and documentation.

Library Class

```python
class Library:
    """
    A class to represent a simple library management system.

    Attributes:
        books (list): A list to store the titles of books available in the library.
    """
    def __init__(self):
        """
        Initializes the Library class with an empty list of books.
        """
        # The 'self.books' attribute is a list that will hold all the book titles.
        # It is initialized as an empty list when a new Library object is created.
        self.books = []
```

Library Class

```python
class Library:
    def __init__(self):

    def add_book(self, book_title):
        """
        Adds a new book to the library's collection.

        Args:
            book_title (str): The title of the book to add.
        """
        # Use the list's 'append()' method to add the new book title to the end of the list.
        self.books.append(book_title)
        # Provide a confirmation message to the user.
        print(f"'{book_title}' has been added to the library.")

    def issue_book(self, book_title):
        """
        Issues a book to a user by removing it from the collection.

        Args:
            book_title (str): The title of the book to issue.
        """
        # Check if the requested book title exists in the 'self.books' list.
        if book_title in self.books:
            # If the book is found, use the list's 'remove()' method to take it out.
            self.books.remove(book_title)
            # Confirm that the book has been successfully issued.
            print(f"'{book_title}' has been successfully issued.")
        else:
            # If the book is not in the list, inform the user it's unavailable.
            print(f"Sorry, '{book_title}' is not available in the library.")

    def display_books(self):
        """
        Displays all the books currently available in the library.
        """
        # Check if the list of books is empty.
        if not self.books:
            # If the list is empty, print a message indicating no books are available.
            print("The library currently has no books.")
        else:
            # If there are books, print a header and then loop through the list.
            print("Current books in the library:")
            # Iterate through each book title in the 'self.books' list.
            for book in self.books:
                # Print each book title with a bullet point for readability.
                print(f"- {book}")

# Example usage of the Library class
if __name__ == "__main__":
    # Create an instance of the Library
    my_library = Library()

    # Add some books to the library's collection
    my_library.add_book("The Hobbit")
    my_library.add_book("1984")
    my_library.add_book("Dune")

    # Display the current books in the library
    print("\n--- Displaying Books ---")
    my_library.display_books()

    # Issue a book that is available in the library
    print("\n--- Issuing '1984' ---")
    my_library.issue_book("1984")
```

```python
    # Try to issue a book that is not available (this will trigger the 'else' block)
    print("\n--- Issuing a Non-existent Book ---")
    my_library.issue_book("Brave New World")

    # Display the updated list of books to show the effect of the issue operation
    print("\n--- Displaying Updated Books ---")
    my_library.display_books()
```