AI ASSISTED CODING

ASSIGNMENT-6

NAME: G.DEEKSHITH RAJ

HT NO.: 2403A51248

Batch no.: 11

Prompt 1:

Create a Python class named Employee with attributes: name, id, and salary. Include a method to calculate the yearly salary.

Test 1:

- Use AI to create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary.
- Instructions:
  - Prompt AI to generate the Employee class.
  - Analyze the generated code for correctness and structure.
  - Ask AI to add a method to give a bonus and recalculate salary.

AI_ASS6 > TEST1 > ...

```python
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name
        self.emp_id = emp_id
        self.salary = salary  # Monthly salary

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"ID: {self.emp_id}")
        print(f"Monthly Salary: ${self.salary}")

    def calculate_yearly_salary(self):
        return self.salary * 12
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name
        self.emp_id = emp_id
        self.salary = salary  # Monthly salary

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"ID: {self.emp_id}")
        print(f"Monthly Salary: ${self.salary}")

    def calculate_yearly_salary(self):
        return self.salary * 12

    def give_bonus(self, bonus_amount):
        self.salary += bonus_amount
        print(f"New Monthly Salary after bonus: ${self.salary}")
        print(f"New Yearly Salary: ${self.calculate_yearly_salary()}")
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name
        self.emp_id = emp_id
        self.salary = salary  # Monthly salary
```

AI_ASS6 > TEST1 > Employee > display_details

```python
    def calculate_yearly_salary(self):
        return self.salary * 12

    def give_bonus(self, bonus_amount):
        self.salary += bonus_amount
        print(f"New Monthly Salary after bonus: ${self.salary}")
        print(f"New Yearly Salary: ${self.calculate_yearly_salary()}")
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name
        self.emp_id = emp_id
        self.salary = salary  # Monthly salary

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"ID: {self.emp_id}")
        print(f"Monthly Salary: ${self.salary}")

    def calculate_yearly_salary(self):
        return self.salary * 12

    def give_bonus(self, bonus_amount):
        self.salary += bonus_amount
        print(f"New Monthly Salary after bonus: ${self.salary}")
        print(f"New Yearly Salary: ${self.calculate_yearly_salary()}")
emp1 = Employee("Alice", 101, 5000)
emp1.display_details()
print("Yearly Salary:", emp1.calculate_yearly_salary())
emp1.give_bonus(500)
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\DAV> & C:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/DAV/AI_ASS6/TEST1
Name: Alice
ID: 101
Monthly Salary: $5000
Yearly Salary: 60000
New Monthly Salary after bonus: $5500
New Yearly Salary: $66000
PS C:\DAV>
```

Prompt 2:

Write a Python function that displays all Automorphic numbers between 1 and 1000 using a for loop.

Test 2:

- Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.
- Instructions:
  - Get AI-generated code to list Automorphic numbers using a for loop.
  - Analyze the correctness and efficiency of the generated logic.
  - Ask AI to regenerate using a while loop and compare both implementations.

AI_ASS6 > TEST2 > ...

```python
def is_automorphic(num):
    square = num ** 2
    return str(square).endswith(str(num))

def find_automorphic_numbers_for():
    print("Automorphic numbers between 1 and 1000 (using for loop):")
    for i in range(1, 1001):
        if is_automorphic(i):
            print(i, end=' ')

find_automorphic_numbers_for()
def find_automorphic_numbers_while():
    print("\nAutomorphic numbers between 1 and 1000 (using while loop):")
    i = 1
    while i <= 1000:
        if is_automorphic(i):
            print(i, end=' ')
        i += 1

find_automorphic_numbers_while()
def is_automorphic(num):
    square = num ** 2
    return str(square).endswith(str(num))

def find_automorphic_numbers_for():
    print("Automorphic numbers between 1 and 1000 (using for loop):")
    for i in range(1, 1001):
        if is_automorphic(i):
            print(i, end=' ')

def find_automorphic_numbers_while():
    print("\nAutomorphic numbers between 1 and 1000 (using while loop):")
    i = 1
    while i <= 1000:
        if is_automorphic(i):
            print(i, end=' ')
```

AI_ASS6 > TEST2 > ...

```python
        if is_automorphic(i):
            print(i, end=' ')
        i += 1

find_automorphic_numbers_while()
def is_automorphic(num):
    square = num ** 2
    return str(square).endswith(str(num))

def find_automorphic_numbers_for():
    print("Automorphic numbers between 1 and 1000 (using for loop):")
    for i in range(1, 1001):
        if is_automorphic(i):
            print(i, end=' ')

def find_automorphic_numbers_while():
    print("\nAutomorphic numbers between 1 and 1000 (using while loop):")
    i = 1
    while i <= 1000:
        if is_automorphic(i):
            print(i, end=' ')
        i += 1

# Run both
find_automorphic_numbers_for()
find_automorphic_numbers_while()
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\DAV> & C:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/DAV/AI_ASS6/TEST2
Automorphic numbers between 1 and 1000 (using for loop):
1 5 6 25 76 376 625
Automorphic numbers between 1 and 1000 (using while loop):
1 5 6 25 76 376 625 Automorphic numbers between 1 and 1000 (using for loop):
1 5 6 25 76 376 625
Automorphic numbers between 1 and 1000 (using while loop):
1 5 6 25 76 376 625
PS C:\DAV>
```

Prompt 3:

Write a Python function using nested if-elif-else statements that classifies user feedback (1–5 rating) into Positive, Neutral, or Negative.

Test 3:

- Task: 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).
- Instructions:
  - Generate initial code using nested if-elif-else.
  - Analyze correctness and readability.
  - Ask AI to rewrite using dictionary-based or match-case structure.

AI_ASS6 > TEST3 > ...

```python
def classify_feedback(rating):
    if rating >= 1 and rating <= 5:
        if rating >= 4:
            return "Positive"
        elif rating == 3:
            return "Neutral"
        else:
            return "Negative"
    else:
        return "Invalid rating"
def classify_feedback_match(rating):
    match rating:
        case 5 | 4:
            return "Positive"
        case 3:
            return "Neutral"
        case 2 | 1:
            return "Negative"
        case _:
            return "Invalid rating"
def classify_feedback_dict(rating):
    feedback_map = {
        1: "Negative",
        2: "Negative",
        3: "Neutral",
        4: "Positive",
        5: "Positive"
    }
    return feedback_map.get(rating, "Invalid rating")
def classify_feedback(rating):
    if rating >= 1 and rating <= 5:
        if rating >= 4:
            return "Positive"
        elif rating == 3:
            return "Neutral"
        else:
```

AI_ASS6 > TEST3 > ...

```python
def classify_feedback(rating):
            return "Positive"
        elif rating == 3:
            return "Neutral"
        else:
            return "Negative"
    else:
        return "Invalid rating"

def classify_feedback_match(rating):
    match rating:
        case 5 | 4:
            return "Positive"
        case 3:
            return "Neutral"
        case 2 | 1:
            return "Negative"
        case _:
            return "Invalid rating"

def classify_feedback_dict(rating):
    feedback_map = {
        1: "Negative",
        2: "Negative",
        3: "Neutral",
        4: "Positive",
        5: "Positive"
    }
    return feedback_map.get(rating, "Invalid rating")
for r in [5, 3, 1, 0, 6]:
    print(f"Rating: {r}")
    print("  if-elif:", classify_feedback(r))
    print("  match-case:", classify_feedback_match(r))
    print("  dictionary:", classify_feedback_dict(r))
```

OUTPUT:

```
PS C:\DAV> & C:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/DAV/AI_ASS6/TEST3
Rating: 5
  if-elif: Positive
  match-case: Positive
  dictionary: Positive
Rating: 3
  if-elif: Neutral
  match-case: Neutral
  dictionary: Neutral
Rating: 1
  if-elif: Negative
  match-case: Negative
  dictionary: Negative
Rating: 0
  if-elif: Invalid rating
  match-case: Invalid rating
  dictionary: Invalid rating
Rating: 6
  if-elif: Invalid rating
  match-case: Invalid rating
  dictionary: Invalid rating
PS C:\DAV>
```

Prompt 4:

Write a Python function that prints all prime numbers between two user-specified numbers using a for loop.

Test 4:

- Task: Generate a function using AI that displays all prime numbers within a user-specified range (e.g., 1 to 500).
- Instructions:
  - Get AI-generated code to list all primes using a for loop.
  - Analyze the correctness and efficiency of the prime-checking logic.
  - Ask AI to regenerate an optimized version (e.g., using the square root method).

AI_ASS6 > TEST4 > find_primes_basic

```python
def is_prime_basic(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

def find_primes_in_range(start, end):
    print(f"Prime numbers between {start} and {end}:")
    for num in range(start, end + 1):
        if is_prime_basic(num):
            print(num, end=' ')
find_primes_in_range(1, 50)
import math

def is_prime_optimized(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    sqrt_n = int(math.sqrt(n)) + 1
    for i in range(3, sqrt_n, 2):  # check only odd numbers
        if n % i == 0:
            return False
    return True

def find_primes_optimized(start, end):
    print(f"\nOptimized Prime numbers between {start} and {end}:")
    for num in range(start, end + 1):
        if is_prime_optimized(num):
            print(num, end=' ')
find_primes_optimized(1, 50)
import math
```

AI_ASS6 > TEST4 > find_primes_basic

```python
find_primes_optimized(1, 50)
import math

def is_prime_basic(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

def find_primes_basic(start, end):
    print(f"\nPrime numbers between {start} and {end} (Basic):")
    for num in range(start, end + 1):
        if is_prime_basic(num):
            print(num, end=' ')

def is_prime_optimized(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    sqrt_n = int(math.sqrt(n)) + 1
    for i in range(3, sqrt_n, 2):
        if n % i == 0:
            return False
    return True

def find_primes_optimized(start, end):
    print(f"\nOptimized Prime numbers between {start} and {end}:")
    for num in range(start, end + 1):
        if is_prime_optimized(num):
            print(num, end=' ')
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\DAV> & C:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/DAV/AI_ASS6/TEST4
Prime numbers between 1 and 50:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
Optimized Prime numbers between 1 and 50:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
Prime numbers between 1 and 50 (Basic):
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
Optimized Prime numbers between 1 and 50:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
PS C:\DAV>
```

Prompt 5:

Create a Python class called Library with methods to add_book(), issue_book(), and display_books().

Test 5:

- Task: Use AI to build a Library class with methods to add_book(), issue_book(), and display_books().
- Instructions:
  - Generate Library class code using AI.
  - Analyze if methods handle edge cases (e.g., issuing unavailable books).
  - Ask AI to add comments and documentation.

AI_ASS6 > TEST5 > ...

```python
class Library:
    def __init__(self):
        # Initialize an empty dictionary to store book titles and their quantities
        self.books = {}

    def add_book(self, title, quantity=1):
        """
        Add a book or increase its quantity in the library.
        """
        if title in self.books:
            self.books[title] += quantity
        else:
            self.books[title] = quantity
        print(f"{quantity} copy/copies of '{title}' added to the library.")

    def issue_book(self, title):
        """
        Issue a book to a user if available.
        """
        if title not in self.books:
            print(f"'{title}' is not available in the library.")
        elif self.books[title] == 0:
            print(f"'{title}' is currently out of stock.")
        else:
            self.books[title] -= 1
            print(f"'{title}' has been issued. Remaining copies: {self.books[title]}")

    def display_books(self):
        """
        Display all books with their available quantities.
        """
        if not self.books:
            print("The library has no books.")
        else:
            print("Books available in the library:")
            for title, quantity in self.books.items():
```

AI_ASS6 > TEST5 > ...

```python
            print("Books available in the library:")
            for title, quantity in self.books.items():
                print(f"• {title} (Available: {quantity})")
class Library:
    def __init__(self):
        # Dictionary to hold book titles and their quantities
        self.books = {}

    def add_book(self, title, quantity=1):
        """
        Add a book to the library or increase the quantity if it already exists.

        Parameters:
        title (str): The title of the book to add.
        quantity (int): Number of copies to add. Defaults to 1.
        """
        if title in self.books:
            self.books[title] += quantity
        else:
            self.books[title] = quantity
        print(f"{quantity} copy/copies of '{title}' added to the library.")

    def issue_book(self, title):
        """
        Issue a book to a user if available.

        Parameters:
        title (str): The title of the book to issue.
        """
        if title not in self.books:
            print(f"'{title}' is not available in the library.")  # Book doesn't exist
        elif self.books[title] == 0:
            print(f"'{title}' is currently out of stock.")  # No copies left
        else:
            self.books[title] -= 1  # Reduce available count
            print(f"'{title}' has been issued. Remaining copies: {self.books[title]}")
```

```
AI_ASS6 > TEST5 > ...
class Library:
    def display_books(self):
        """
        Display all books in the library with their available quantities.
        """
        if not self.books:
            print("The library has no books.")  # Empty library
        else:
            print("Books available in the library:")
            for title, quantity in self.books.items():
                print(f"• {title} (Available: {quantity})")
# Create library object
my_library = Library()

# Add books
my_library.add_book("Python Programming", 3)
my_library.add_book("Data Structures", 2)
my_library.add_book("Python Programming", 2)  # Adding more to existing

# Display books
my_library.display_books()

# Issue a book
my_library.issue_book("Python Programming")

# Try to issue a book not in library
my_library.issue_book("Machine Learning")

# Try to issue a book until it's out of stock
my_library.issue_book("Data Structures")
my_library.issue_book("Data Structures")
my_library.issue_book("Data Structures")  # Out of stock now

# Display updated book list
my_library.display_books()
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\DAV> & C:\Users\HP\AppData\Local\Programs\Python\Python313\python.exe c:/DAV/AI_ASS6/TEST5
3 copy/copies of 'Python Programming' added to the library.
2 copy/copies of 'Data Structures' added to the library.
2 copy/copies of 'Python Programming' added to the library.
Books available in the library:
• Python Programming (Available: 5)
• Data Structures (Available: 2)
'Python Programming' has been issued. Remaining copies: 4
'Machine Learning' is not available in the library.
'Data Structures' has been issued. Remaining copies: 1
'Data Structures' has been issued. Remaining copies: 0
'Data Structures' is currently out of stock.
Books available in the library:
• Python Programming (Available: 4)
• Data Structures (Available: 0)
PS C:\DAV>
```