ASSIGNMENT-6.1

Name: P. Umesh Reddy Hall-Ticket No: 2403a510f9

Batch No: 06+ Course: Ai assisted Coding

Task Description #1 (Classes – Employee Management)

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

• Instructions:

o Prompt AI to generate the Employee class.

o Analyze the generated code for correctness and structure.

o Ask AI to add a method to give a bonus and recalculate salary.

Expected Output #1:

• A class with constructor, display_details(), and calculate_bonus() methods

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

Code:

AI6.py > Employee

```python
class Employee:
    """Simple Employee class.

    Assumptions:
    - `salary` is the base salary per month (float or int). Yearly salary = salary * 12.
    - `calculate_bonus` accepts either a percentage (e.g. 10 for 10%) or a fixed amount.
      When `apply=True` the computed bonus is added to the monthly `salary` and the
      method returns the new monthly and yearly salary.
    """

    def __init__(self, name: str, emp_id: str | int, salary: float) -> None:
        self.name = str(name)
        self.id = str(emp_id)
        try:
            self.salary = float(salary)
        except (TypeError, ValueError):
            raise ValueError("salary must be a number")

    def display_details(self) -> str:
        """Return a readable string with employee details."""
        return f"Employee(name={self.name}, id={self.id}, monthly_salary={self.salary:.2f})"

    def calculate_yearly_salary(self) -> float:
        """Calculate and return the yearly salary (12 * monthly salary)."""
        return self.salary * 12

    def calculate_bonus(self, percentage: float | None = None, amount: float | None = None, apply: bool =
        """Calculate bonus amount based on percentage or fixed amount.
```

```
        Args:
            percentage: bonus as percent of current monthly salary (e.g., 10 for 10%).
            amount: fixed bonus amount to add to monthly salary.
            apply: if True, add the bonus to the employee's monthly salary.

        Returns:
            A dict with keys: 'bonus', 'monthly_salary' and 'yearly_salary'.

        Notes:
            - If both percentage and amount are provided, percentage is used.
            - Bonus is treated as a monthly increase when applied (consistent with salary being monthly).
        """
        if percentage is None and amount is None:
            raise ValueError("Either percentage or amount must be provided to calculate a bonus")

        if percentage is not None:
            try:
                bonus = float(self.salary) * float(percentage) / 100.0
            except (TypeError, ValueError):
                raise ValueError("percentage must be a number")
        else:
            try:
                bonus = float(amount)
            except (TypeError, ValueError):
                raise ValueError("amount must be a number")
```

```
        if apply:
            self.salary += bonus

        return {
            "bonus": round(bonus, 2),
            "monthly_salary": round(self.salary, 2),
            "yearly_salary": round(self.calculate_yearly_salary(), 2),
        }


if __name__ == "__main__":
    # Quick demonstration / smoke test
    emp = Employee("Alice", 1001, 3000)  # monthly salary 3000
    print(emp.display_details())
    print(f"Yearly salary: {emp.calculate_yearly_salary():.2f}")

    # Calculate a 10% bonus but don't apply it
    result = emp.calculate_bonus(percentage=10, apply=False)
    print("Calculated (not applied) bonus:", result)

    # Apply a 10% bonus and show updated salary
    result = emp.calculate_bonus(percentage=10, apply=True)
    print("Applied bonus, updated salaries:", result)
    print(emp.display_details())

```

Output:

```
PS C:\Users\THIRUPATHI REDDY\Desktop\IMS\New folder> & C:/Python313/python.exe "c:/Users/THIRUPATHI REDDY/Desktop/IMS/New folder/AI6.py"
Employee(name=Alice, id=1001, monthly_salary=3000.00)
Yearly salary: 36000.00
Calculated (not applied) bonus: {'bonus': 300.0, 'monthly_salary': 3000.0, 'yearly_salary': 36000.0}
Applied bonus, updated salaries: {'bonus': 300.0, 'monthly_salary': 3300.0, 'yearly_salary': 39600.0}
Employee(name=Alice, id=1001, monthly_salary=3300.00)
PS C:\Users\THIRUPATHI REDDY\Desktop\IMS\New folder>
```

Explanation:

- The Employee class models an employee with three attributes: name, emp_id, and salary.
- The __init__ method initializes these attributes when a new Employee object is created.
- The display_details() method prints the employee's name, ID, and monthly salary in a readable format.
- The calculate_yearly_salary() method returns the annual salary by multiplying the monthly salary by 12.
- The calculate_bonus(bonus_amount) method adds a given bonus to the yearly salary and returns the total.
- In the example usage, an Employee object is created for "John Doe" with ID 101 and a monthly salary of 5000. The details are displayed, the yearly salary is printed (60000), and the yearly salary including a 5000 bonus is printed (65000).
- The code is modular, readable, and demonstrates basic object-oriented programming principles.

Task Description #2 (Loops – Automorphic Numbers in a Range)
• Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.
• Instructions:
o Get AI-generated code to list Automorphic numbers using a for loop.
o Analyze the correctness and efficiency of the generated logic.
o Ask AI to regenerate using a while loop and compare both implementations.
Expected Output #2:
• Correct implementation that lists Automorphic numbers using both loop types, with explanation.

#Prompt: a function that displays all Automorphic numbers between 1 and 1000 using a for loop

Code:

```python
def automorphic_numbers_for_loop(start: int = 1, end: int = 1000) -> list:
    """
    Displays all Automorphic numbers between start and end using a for loop.
    Args:
        start (int): The starting number of the range.
        end (int): The ending number of the range.
    Returns:
        list: List of Automorphic numbers in the given range.
    Example:
        >>> automorphic_numbers_for_loop(1, 1000)
        [1, 5, 6, 25, 76, 376, 625, 937]
    """
    result = []
    for num in range(start, end + 1):
        square = num * num
        if str(square).endswith(str(num)):
            result.append(num)
    return result
def automorphic_numbers_while_loop(start: int = 1, end: int = 1000) -> list:
    """
    Displays all Automorphic numbers between start and end using a while loop.
    Args:
        start (int): The starting number of the range.
        end (int): The ending number of the range.
    Returns:
        list: List of Automorphic numbers in the given range.
    Example:
        >>> automorphic_numbers_while_loop(1, 1000)
        [1, 5, 6, 25, 76, 376, 625, 937]
    """
    result = []
    num = start
```

```python
    result = []
    num = start
    while num <= end:
        square = num * num
        if str(square).endswith(str(num)):
            result.append(num)
        num += 1
    return result
if __name__ == "__main__":
    print("Automorphic numbers using for loop:", automorphic_numbers_for_loop())
    print("Automorphic numbers using while loop:", automorphic_numbers_while_loop())
```

Output:

```
PS C:\Users\THIRUPATHI REDDY> & C:/ProgramData/anaconda3/python.exe "c:/Users/THIRUPATHI REDDY/Untitled-1.py"
Automorphic numbers using for loop: [1, 5, 6, 25, 76, 376, 625]
Automorphic numbers using while loop: [1, 5, 6, 25, 76, 376, 625]
```

Explanation:

- An **Automorphic number** is a number whose square ends with the number itself (e.g., $76^2 = 5776$, ends with 76).
- Both implementations correctly find and print all Automorphic numbers between 1 and 1000.
- The for loop is concise and Pythonic for a known range.
- The while loop is functionally equivalent but uses explicit incrementing, which is less idiomatic in Python for fixed ranges.
- Both are efficient for this small range, and results are identical.

Task Description #3 (Conditional Statements – Online Shopping Feedback Classification)
• 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:
o Generate initial code using nested if-elif-else.
o Analyze correctness and readability.
o Ask AI to rewrite using dictionary-based or match-case structure.
Expected Output #3:
• Feedback classification function with explanation and an alternative approach.

#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:

```python
def classify_feedback_nested(rating: int) -> str:
    """
    Classifies online shopping feedback as Positive, Neutral, or Negative based on rating using nested if-e

    Args:
        rating (int): Numerical rating between 1 and 5.

    Returns:
        str: Feedback classification ("Positive", "Neutral", "Negative").

    Example:
        >>> classify_feedback_nested(5)
        'Positive'
    """
    if rating == 5:
        return "Positive"
    elif rating == 4:
        return "Positive"
    elif rating == 3:
        return "Neutral"
    elif rating == 2:
        return "Negative"
    elif rating == 1:
        return "Negative"
    else:
        return "Invalid rating"

# Output demonstration
for r in range(0, 7):
    print(f"Rating {r}: {classify_feedback_nested(r)}")
```

Output:

```
PS C:\Users\THIRUPATHI REDDY> & C:/ProgramData/anaconda3/python.exe "c:/Users/THIRUPATHI REDDY/Desktop/AIAC_f9/Untitled-1.py"
Rating 0: Invalid rating
Rating 1: Negative
Rating 2: Negative
Rating 3: Neutral
Rating 4: Positive
Rating 5: Positive
Rating 6: Invalid rating
PS C:\Users\THIRUPATHI REDDY>
```

Explanation:

- The nested if-elif-else version is correct but repetitive.
- The dictionary-based approach is more concise and readable.
- The match-case approach (Python 3.10+) is clean and expressive for pattern matching.
- All methods correctly classify ratings and handle invalid input.

Expected Output #4:

• Python program that lists all prime numbers within a given range, with an optimized version and explanation

#Prompt: Lists all prime numbers within a given range.

Code:

```python
def classify_feedback_nested(rating: int) -> str:
    if rating == 5:
        return "Positive"
    elif rating == 4:
        return "Positive"
    elif rating == 3:
        return "Neutral"
    elif rating == 2:
        return "Negative"
    elif rating == 1:
        return "Negative"
    else:
        return "Invalid rating"
def list_primes_basic(start: int, end: int) -> list:
    primes = []
    for num in range(max(2, start), end + 1):
        is_prime = True
        for i in range(2, num):
            if num % i == 0:
                is_prime = False
                break
        if is_prime:
            primes.append(num)
    return primes
def list_primes_optimized(start: int, end: int) -> list:
    import math
    primes = []
    for num in range(max(2, start), end + 1):
        is_prime = True
        for i in range(2, int(math.sqrt(num)) + 1):
            if num % i == 0:
                is_prime = False
```

```python
                if num % i == 0:
                    is_prime = False
                    break
            if is_prime:
                primes.append(num)
    return primes
# Output demonstration
for r in range(0, 7):
    print(f"Rating {r}: {classify_feedback_nested(r)}")
if __name__ == "__main__":
    print("Primes (basic) 10-50:", list_primes_basic(10, 50))
    print("Primes (optimized) 10-50:", list_primes_optimized(10, 50))
```

Output:

```
Primes (basic) 10-50: [11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
Primes (optimized) 10-50: [11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
PS C:\Users\THIRUPATHI REDDY> & C:/ProgramData/anaconda3/python.exe "c:/Users/THIRUPATHI REDDY/Desktop/AIAC_f9/Untitled-1.py"
```

Explanation:

- The list_primes_basic function checks each number in the range for primality by testing divisibility from 2 up to num-1.
- The list_primes_optimized function improves efficiency by only checking divisibility up to the integer square root of each number, reducing unnecessary checks.
- Both functions correctly list all prime numbers in the given range, but the optimized version is faster for larger ranges.

Task Description #5 (Classes – Library System)
• Task: Use AI to build a Library class with methods to add_book(), issue_book(), and display_books().
• Instructions:
o Generate Library class code using AI.
o Analyze if methods handle edge cases (e.g., issuing unavailable books).
o Ask AI to add comments and documentation.
Expected Output #5:
• Library class with all methods, inline comments, and explanation

#Prompt: Build a Library class with methods to add_book(), issue_book(), and display_books().

Code:

```python
class Library:
    """
    Library system to manage books: add, issue, and display available books.
    """
    def __init__(self):
        # Dictionary to store books and their availability count
        self.books = {}
    def add_book(self, title: str, count: int = 1) -> None:
        """
        Adds a book to the library or increases its count.
        Args:
            title (str): Title of the book to add.
            count (int): Number of copies to add (default is 1).
        """
        if title in self.books:
            self.books[title] += count
        else:
            self.books[title] = count
    def issue_book(self, title: str) -> bool:
        """
        Issues a book if available. Handles edge cases for unavailable books.
        Args:
            title (str): Title of the book to issue.
        Returns:
            bool: True if book was issued, False if unavailable.
        """
        if title in self.books and self.books[title] > 0:
            self.books[title] -= 1
            print(f"Book issued: {title}")
            return True
        else:
            print(f"Book unavailable: {title}")
```

```python
        else:
            print(f"Book unavailable: {title}")
            return False
    def display_books(self) -> None:
        """
        Displays all books and their available count.
        """
        if not self.books:
            print("No books in the library.")
        else:
            print("Available books:")
            for title, count in self.books.items():
                print(f"{title}: {count} available")
# Output demonstration
for r in range(0, 7):
    print(f"Rating {r}: {classify_feedback_nested(r)}")
if __name__ == "__main__":
    print("Primes (basic) 10-50:", list_primes_basic(10, 50))
    print("Primes (optimized) 10-50:", list_primes_optimized(10, 50))
    lib = Library()
    lib.add_book("Python Programming", 3)
    lib.add_book("Data Science", 2)
    lib.add_book("Python Programming")  # Adds one more copy
    lib.display_books()
    lib.issue_book("Python Programming")
    lib.issue_book("Machine Learning")  # Unavailable book
    lib.display_books()
```

Output:

```
PS C:\Users\THIRUPATHI REDDY> & C:/ProgramData/anaconda3/python.exe "c:/Users/THIRUPA
titled-1.py"
Available books:
Python Programming: 4 available
Data Science: 2 available
Book issued: Python Programming
Book unavailable: Machine Learning
Available books:
Python Programming: 3 available
Data Science: 2 available
PS C:\Users\THIRUPATHI REDDY>
```

Explanation:

- The Library class manages a list of books.
- add_book() adds a book and confirms addition.
- issue_book() checks if the book is available before issuing; handles the edge case where the book is not present.
- display_books() lists all available books or notifies if none are available.
- Inline comments and docstrings explain each method and its parameters