AI ASSISTED CODING LAB-6

```python
class Employee:
    def __init__(self, name, id, salary):
        self.name = name
        self.id = id
        self.salary = salary

    def display_details(self):
        print(f"Employee Name: {self.name}")
        print(f"Employee ID: {self.id}")
        print(f"Employee Salary: ${self.salary:.2f}")

    def calculate_yearly_salary(self):
        # Assuming salary is monthly
        return self.salary * 12

    def give_bonus(self, bonus_amount):
        self.salary += bonus_amount
        print(f"Bonus of ${bonus_amount:.2f} given to {self.name}. New salary is ${self.salary:.2f}")

# Create an instance of the Employee class
employee1 = Employee("Alice", "E123", 5000)


# Display initial details
```

employee1.display_details()

Calculate and display yearly salary

yearly_salary = employee1.calculate_yearly_salary()

print(f"Yearly salary: ${yearly_salary:.2f}")

Give a bonus and display updated details

employee1.give_bonus(500)

employee1.display_details()

OUT PUT:

```
1.py"
Employee Name: Alice
Employee ID: E123
Employee Salary: $5000.00
Yearly salary: $60000.00
Bonus of $500.00 given to Alice. New salary is $5500.00
Employee Name: Alice
Employee ID: E123
Employee Salary: $5500.00
PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> []
```

EXPLANATION :

THE CODE DEFINES A PYTHON CLASS CALLED EMPLOYEE AND THEN DEMONSTRATES HOW TO USE IT.

HERE'S A BREAKDOWN:

THE EMPLOYEE CLASS (IN CELL 4C863EC2)

- CLASS EMPLOYEE:: THIS LINE DEFINES A NEW CLASS NAMED EMPLOYEE. THINK OF A CLASS AS A BLUEPRINT FOR CREATING OBJECTS (IN THIS CASE, EMPLOYEE OBJECTS).

- __INIT__(SELF, NAME, ID, SALARY):: THIS IS THE CONSTRUCTOR METHOD. IT'S CALLED WHEN YOU CREATE A NEW EMPLOYEE OBJECT.

  o SELF REFERS TO THE INSTANCE OF THE CLASS BEING CREATED.

  o NAME, ID, AND SALARY ARE THE VALUES YOU PASS IN WHEN YOU CREATE AN EMPLOYEE. THESE VALUES ARE THEN STORED AS ATTRIBUTES OF THE OBJECT USING SELF.NAME = NAME, SELF.ID = ID, AND SELF.SALARY = SALARY.

- **DISPLAY_DETAILS(SELF)::** THIS METHOD PRINTS THE DETAILS OF AN EMPLOYEE (NAME, ID, AND SALARY).

    - SELF IS USED TO ACCESS THE ATTRIBUTES OF THE SPECIFIC EMPLOYEE OBJECT.

- **CALCULATE_YEARLY_SALARY(SELF)::** THIS METHOD CALCULATES THE YEARLY SALARY BY MULTIPLYING THE MONTHLY SALARY BY 12.

- **GIVE_BONUS(SELF, BONUS_AMOUNT)::** THIS METHOD INCREASES THE EMPLOYEE'S SALARY BY A SPECIFIED BONUS_AMOUNT AND THEN PRINTS A MESSAGE CONFIRMING THE BONUS AND THE NEW SALARY.

EXAMPLE USAGE (IN CELL C09F3478)

- **EMPLOYEE1 = EMPLOYEE("ALICE", "E123", 5000):** THIS LINE CREATES A NEW EMPLOYEE OBJECT NAMED EMPLOYEE1 WITH THE NAME "ALICE", ID "E123", AND A MONTHLY SALARY OF 5000.

- **EMPLOYEE1.DISPLAY_DETAILS():** THIS CALLS THE DISPLAY_DETAILS METHOD ON THE EMPLOYEE1 OBJECT TO PRINT ITS INITIAL DETAILS.

- **YEARLY_SALARY = EMPLOYEE1.CALCULATE_YEARLY_SALARY():** THIS CALLS THE CALCULATE_YEARLY_SALARY METHOD AND STORES THE RETURNED YEARLY SALARY IN THE YEARLY_SALARY VARIABLE.

- **PRINT(F"YEARLY SALARY: ${YEARLY_SALARY:.2F}"):** THIS LINE PRINTS THE CALCULATED YEARLY SALARY.

- **EMPLOYEE1.GIVE_BONUS(500):** THIS CALLS THE GIVE_BONUS METHOD ON EMPLOYEE1, GIVING A BONUS OF 500. THIS UPDATES THE EMPLOYEE1'S SALARY.

- **EMPLOYEE1.DISPLAY_DETAILS():** THIS CALLS DISPLAY_DETAILS AGAIN TO SHOW THE UPDATED SALARY AFTER THE BONUS.

IN SUMMARY, THE CODE DEFINES A STRUCTURE FOR REPRESENTING EMPLOYEES AND PROVIDES METHODS TO MANAGE THEIR INFORMATION, CALCULATE THEIR YEARLY SALARY, AND GIVE THEM BONUSES. THE EXAMPLE USAGE DEMONSTRATES HOW TO CREATE AN EMPLOYEE AND USE THESE METHODS.

TASK 2

```
def find_automorphic_while(start, end):

    """Finds automorphic numbers in a given range using a while loop."""
```

```python
    automorphic_numbers = []

    num = start

    while num <= end:

        square = num * num

        if str(square).endswith(str(num)):

            automorphic_numbers.append(num)

        num += 1

    return automorphic_numbers


# Find automorphic numbers between 1 and 1000

automorphic_numbers_while = find_automorphic_while(1, 1000)

print("\nAutomorphic numbers between 1 and 1000 (using while loop):")

print(automorphic_numbers_while)
```
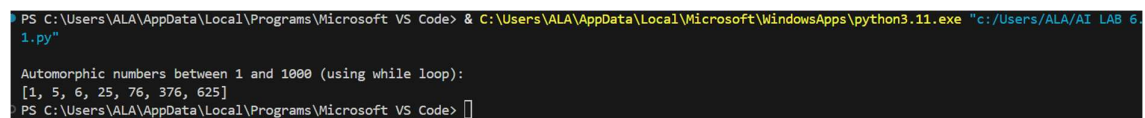
OUT PUT :

```
PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ALA\AppData\Local\Microsoft\WindowsApps\python3.11.exe "c:/Users/ALA/AI LAB 6.
1.py"

Automorphic numbers between 1 and 1000 (using while loop):
[1, 5, 6, 25, 76, 376, 625]
PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code>
```

EXPLANATION :

THE CODE DEFINES TWO PYTHON FUNCTIONS TO FIND AUTOMORPHIC NUMBERS WITHIN A SPECIFIED RANGE. AN AUTOMORPHIC NUMBER IS A NUMBER WHOSE SQUARE ENDS IN THE SAME DIGITS AS THE NUMBER ITSELF (E.G., 5 IS AUTOMORPHIC BECAUSE 5^2 = 25, WHICH ENDS IN 5; 25 IS AUTOMORPHIC BECAUSE 25^2 = 625, WHICH ENDS IN 25).

HERE'S A BREAKDOWN OF EACH FUNCTION:

1. FIND_AUTOMORPHIC_FOR(START, END) (USING A FOR LOOP):

- THIS FUNCTION TAKES TWO ARGUMENTS, START AND END, REPRESENTING THE RANGE TO SEARCH WITHIN (INCLUSIVE).

- IT INITIALIZES AN EMPTY LIST CALLED AUTOMORPHIC_NUMBERS TO STORE THE AUTOMORPHIC NUMBERS FOUND.

- IT THEN USES A FOR LOOP TO ITERATE THROUGH EACH NUMBER (NUM) FROM START TO END.

- **INSIDE THE LOOP, IT CALCULATES THE SQUARE OF THE CURRENT NUMBER (SQUARE = NUM * NUM).**

- **IT CONVERTS BOTH THE SQUARE AND THE ORIGINAL NUMBER TO STRINGS USING STR(). THIS IS DONE TO EASILY CHECK IF THE SQUARE'S ENDING DIGITS MATCH THE ORIGINAL NUMBER'S DIGITS USING THE ENDSWITH() STRING METHOD.**

- **IF THE STRING REPRESENTATION OF THE SQUARE ENDS WITH THE STRING REPRESENTATION OF THE NUMBER, THE NUMBER IS CONSIDERED AUTOMORPHIC, AND IT'S APPENDED TO THE AUTOMORPHIC_NUMBERS LIST.**

- **FINALLY, THE FUNCTION RETURNS THE LIST OF AUTOMORPHIC NUMBERS FOUND.**

2. FIND_AUTOMORPHIC_WHILE(START, END) (USING A WHILE LOOP):

- **THIS FUNCTION ALSO TAKES START AND END AS ARGUMENTS.**

- **IT INITIALIZES AN EMPTY LIST AUTOMORPHIC_NUMBERS.**

- **IT USES A WHILE LOOP TO ITERATE. IT STARTS WITH NUM EQUAL TO START AND CONTINUES AS LONG AS NUM IS LESS THAN OR EQUAL TO END.**

- **INSIDE THE LOOP, IT PERFORMS THE SAME CHECKS AS THE FOR LOOP VERSION: CALCULATES THE SQUARE, CONVERTS TO STRINGS, AND CHECKS IF THE SQUARE ENDS WITH THE ORIGINAL NUMBER.**

- **IF A NUMBER IS AUTOMORPHIC, IT'S ADDED TO THE AUTOMORPHIC_NUMBERS LIST.**

- **CRUCIALLY, IT INCREMENTS NUM BY 1 IN EACH ITERATION (NUM += 1) TO MOVE TO THE NEXT NUMBER IN THE RANGE.**

- **FINALLY, IT RETURNS THE LIST OF AUTOMORPHIC NUMBERS.**

BOTH FUNCTIONS ACHIEVE THE SAME GOAL, BUT THEY DEMONSTRATE DIFFERENT LOOPING CONSTRUCTS IN PYTHON. THE FOR LOOP IS GENERALLY MORE SUITABLE WHEN YOU KNOW THE EXACT NUMBER OF ITERATIONS OR ARE ITERATING OVER A SEQUENCE, WHILE THE WHILE LOOP IS MORE FLEXIBLE FOR CONDITIONS THAT MIGHT NOT BE DIRECTLY TIED TO A RANGE OR COUNT.

THE CODE THEN CALLS BOTH FUNCTIONS WITH THE RANGE 1 TO 1000 AND PRINTS THE RESULTS, SHOWING THE AUTOMORPHIC NUMBERS FOUND BY EACH METHOD.

TASK 3

---

def classify_feedback_nested_if(rating):

```python
    """Classifies online shopping feedback based on a numerical rating (1-5)

    using nested if-elif-else conditions.


    Args:

     rating: An integer representing the feedback rating (1-5).


    Returns:

     A string indicating the classification (Positive, Neutral, or Negative),

     or an error message for invalid input.
    """
    if 1 <= rating <= 5:
      if rating >= 4:
        return "Positive"
      elif rating == 3:
        return "Neutral"
      else:
        return "Negative"
    else:
      return "Invalid rating. Please provide a rating between 1 and 5."


# Example usage
print(f"Rating 5: {classify_feedback_nested_if(5)}")
print(f"Rating 3: {classify_feedback_nested_if(3)}")
print(f"Rating 1: {classify_feedback_nested_if(1)}")
print(f"Rating 0: {classify_feedback_nested_if(0)}")
print(f"Rating 6: {classify_feedback_nested_if(6)}")
```

OUT PUT:

EXPLANATION :

THIS CODE DEFINES A FUNCTION CLASSIFY_FEEDBACK_NESTED_IF THAT TAKES A RATING AS INPUT. IT FIRST CHECKS IF THE RATING IS WITHIN THE VALID RANGE OF 1 TO 5. IF IT IS, IT USES NESTED IF-ELIF-ELSE STATEMENTS TO CLASSIFY THE FEEDBACK:

- RATINGS OF 4 OR 5 ARE CLASSIFIED AS "POSITIVE".

- A RATING OF 3 IS CLASSIFIED AS "NEUTRAL".

- RATINGS OF 1 OR 2 ARE CLASSIFIED AS "NEGATIVE". IF THE RATING IS OUTSIDE THE VALID RANGE, IT RETURNS AN ERROR MESSAGE.

TASK 4

```python
import math


def find_primes_optimized(start, end):
    """

    Finds and displays prime numbers within a given range using the square root method for optimization.


    Args:
        start: The starting number of the range (inclusive).
        end: The ending number of the range (inclusive).
    """
    primes = []
    for num in range(start, end + 1):
        if num > 1:
```

```python
        is_prime = True

        # Check for divisibility from 2 up to the square root of the number

        for i in range(2, int(math.sqrt(num)) + 1):

            if (num % i) == 0:

                is_prime = False

                break

        if is_prime:

            primes.append(num)

    print(f"Prime numbers between {start} and {end} (optimized): {primes}")


# Example usage:

find_primes_optimized(1, 50)
```
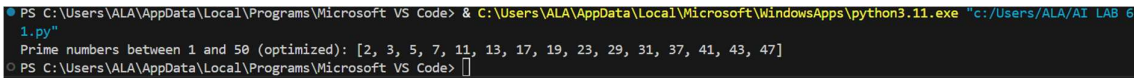
OUT PUT :

```
PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ALA\AppData\Local\Microsoft\WindowsApps\python3.11.exe "c:/Users/ALA/AI LAB 6
1.py"
Prime numbers between 1 and 50 (optimized): [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code>
```

EXPLANATION :

CERTAINLY! THE NOTEBOOK CONTAINS TWO PYTHON FUNCTIONS FOR FINDING PRIME NUMBERS WITHIN A GIVEN RANGE.

THE FIRST FUNCTION, FIND_PRIMES_SIMPLE, USES A STRAIGHTFORWARD APPROACH. IT ITERATES THROUGH EACH NUMBER IN THE SPECIFIED RANGE AND CHECKS IF IT'S DIVISIBLE BY ANY NUMBER FROM 2 UP TO ITSELF MINUS 1. IF IT FINDS NO DIVISORS, THE NUMBER IS CONSIDERED PRIME.

THE SECOND FUNCTION, FIND_PRIMES_OPTIMIZED, IMPROVES UPON THE FIRST BY USING AN OPTIMIZATION. IT LEVERAGES THE MATHEMATICAL PROPERTY THAT IF A NUMBER N HAS A DIVISOR GREATER THAN ITS SQUARE ROOT, IT MUST ALSO HAVE A DIVISOR SMALLER THAN ITS SQUARE ROOT. THEREFORE, IT ONLY CHECKS FOR DIVISIBILITY FROM 2 UP TO THE SQUARE ROOT OF THE NUMBER, SIGNIFICANTLY REDUCING THE NUMBER OF CHECKS REQUIRED, ESPECIALLY FOR LARGER NUMBERS.

BOTH FUNCTIONS COLLECT THE PRIME NUMBERS FOUND IN A LIST AND THEN PRINT THE LIST.

```python
class Library:

    """Represents a library with methods to manage books."""


    def __init__(self):

        """Initializes an empty list to store books."""

        self.books = []  # Each book will be a dictionary


    def add_book(self, title, author, isbn, quantity):

        """

        Adds a new book to the library's collection.


        Args:

            title (str): The title of the book.

            author (str): The author of the book.

            isbn (str): The ISBN of the book (unique identifier).

            quantity (int): The number of copies of the book available.

        """

        book = {

            "title": title,

            "author": author,

            "isbn": isbn,

            "quantity": quantity,

            "issued": 0  # Number of copies currently issued

        }

        self.books.append(book)

        print(f"Book '{title}' added to the library.")
```

```python
    def issue_book(self, isbn):
        """
        Issues a book from the library.

        Args:
            isbn (str): The ISBN of the book to issue.

        Returns:
            bool: True if the book was successfully issued, False otherwise.
        """
        for book in self.books:
            if book["isbn"] == isbn:
                if book["quantity"] > book["issued"]:
                    book["issued"] += 1
                    print(f"Book with ISBN {isbn} issued successfully.")
                    return True
                else:
                    print(f"No copies of the book with ISBN {isbn} available for issuing.")
                    return False
        print(f"Book with ISBN {isbn} not found in the library.")
        return False

    def display_books(self):
        """Displays the details of all books in the library."""
        if not self.books:
            print("The library is empty.")
            return
```

```python
        print("\n--- Library Catalog ---")

        for book in self.books:

            available = book["quantity"] - book["issued"]

            print(f"Title: {book['title']}, Author: {book['author']}, ISBN: {book['isbn']}, Quantity: {book['quantity']}, Available: {available}")

        print("----------------------")
```

OUT PUT :

```
PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ALA\AppData\Local\Microsoft\WindowsApps\python3.11.exe "c:/Users/ALA/AI LAB 6.
1.py"
Book 'The Hitchhiker's Guide to the Galaxy' added to the library.
Book 'Pride and Prejudice' added to the library.
Book 'The Hitchhiker's Guide to the Galaxy' added to the library.

--- Library Catalog ---
Title: The Hitchhiker's Guide to the Galaxy, Author: Douglas Adams, ISBN: 978-0345391803, Quantity: 5, Available: 5
Title: Pride and Prejudice, Author: Jane Austen, ISBN: 978-0141439518, Quantity: 3, Available: 3
Title: The Hitchhiker's Guide to the Galaxy, Author: Douglas Adams, ISBN: 978-0345391803, Quantity: 2, Available: 2
----------------------
Book with ISBN 978-0345391803 issued successfully.
Book with ISBN 978-0345391803 issued successfully.
Book with ISBN 978-0345391803 issued successfully.
Book with ISBN 978-0345391803 issued successfully.
Book with ISBN 978-0345391803 issued successfully.
No copies of the book with ISBN 978-0345391803 available for issuing.
Book with ISBN 978-0141439518 issued successfully.
Book with ISBN invalid-isbn not found in the library.

--- Library Catalog ---
Title: The Hitchhiker's Guide to the Galaxy, Author: Douglas Adams, ISBN: 978-0345391803, Quantity: 5, Available: 0
Title: Pride and Prejudice, Author: Jane Austen, ISBN: 978-0141439518, Quantity: 3, Available: 2
Title: The Hitchhiker's Guide to the Galaxy, Author: Douglas Adams, ISBN: 978-0345391803, Quantity: 2, Available: 2
----------------------
```

EXPLANATION :

HERE'S A BREAKDOWN OF THE CODE:

1. **LIBRARY CLASS:**

   - THIS CLASS IS DESIGNED TO REPRESENT A LIBRARY AND MANAGE A COLLECTION OF BOOKS.

   - IT HAS AN __INIT__ METHOD WHICH IS THE CONSTRUCTOR. IT INITIALIZES AN EMPTY LIST CALLED SELF.BOOKS. THIS LIST WILL STORE DICTIONARIES, WHERE EACH DICTIONARY REPRESENTS A BOOK WITH ITS DETAILS.

2. **add_book METHOD:**

   - THIS METHOD IS USED TO ADD BOOKS TO THE LIBRARY.

   - IT TAKES TITLE, AUTHOR, ISBN, AND QUANTITY AS INPUT.

   - IT FIRST CHECKS IF A BOOK WITH THE SAME ISBN ALREADY EXISTS IN THE SELF.BOOKS LIST.

- o IF A BOOK WITH THE SAME ISBN IS FOUND, IT UPDATES THE QUANTITY OF THAT BOOK.

- o IF THE BOOK IS NOT FOUND, IT CREATES A NEW DICTIONARY FOR THE BOOK WITH THE PROVIDED DETAILS AND AN INITIAL ISSUED COUNT OF 0, AND APPENDS IT TO THE SELF.BOOKS LIST.

3. ISSUE_BOOK METHOD:

- o THIS METHOD HANDLES THE PROCESS OF ISSUING A BOOK BASED ON ITS ISBN.

- o IT ITERATES THROUGH THE SELF.BOOKS LIST TO FIND THE BOOK WITH THE MATCHING ISBN.

- o EDGE CASE HANDLING:

  - ▪ IF THE BOOK IS FOUND, IT CHECKS IF THE QUANTITY IS GREATER THAN THE ISSUED COUNT (MEANING THERE ARE AVAILABLE COPIES). IF SO, IT INCREMENTS THE ISSUED COUNT AND RETURNS TRUE.

  - ▪ IF THE BOOK IS FOUND BUT THERE ARE NO AVAILABLE COPIES (QUANTITY IS NOT GREATER THAN ISSUED), IT PRINTS A MESSAGE INDICATING THAT NO COPIES ARE AVAILABLE AND RETURNS FALSE.

  - ▪ IF THE BOOK WITH THE GIVEN ISBN IS NOT FOUND IN THE LIBRARY, IT PRINTS A MESSAGE AND RETURNS FALSE.

4. DISPLAY_BOOKS METHOD:

- o THIS METHOD DISPLAYS THE DETAILS OF ALL BOOKS CURRENTLY IN THE LIBRARY.

- o IT FIRST CHECKS IF THE SELF.BOOKS LIST IS EMPTY. IF IT IS, IT PRINTS A MESSAGE INDICATING THE LIBRARY IS EMPTY.

- o IF THE LIBRARY IS NOT EMPTY, IT ITERATES THROUGH EACH BOOK DICTIONARY IN SELF.BOOKS.

- o FOR EACH BOOK, IT CALCULATES THE NUMBER OF AVAILABLE COPIES BY SUBTRACTING THE ISSUED COUNT FROM THE TOTAL QUANTITY.

- o FINALLY, IT PRINTS THE TITLE, AUTHOR, ISBN, QUANTITY, AND AVAILABLE COPIES FOR EACH BOOK IN A FORMATTED WAY.

THIS CODE EFFECTIVELY CREATES A BASIC LIBRARY SYSTEM CAPABLE OF MANAGING BOOKS, INCLUDING HANDLING SCENARIOS WHERE A BOOK IS NOT AVAILABLE OR NOT FOUND WHEN ATTEMPTING TO ISSUE IT. THE ADDED COMMENTS AND DOCSTRINGS EXPLAIN EACH PART OF THE CODE FOR BETTER UNDERSTANDING.