

	<p style="text-align: center;">AI ASSISTED CODING</p> <p>NAME: J.KEERTHIPRIYA ROLL NO : 2403A510G4 ASSIGNMENT:6.1</p>	
1	<p>Task Description #1 (Classes – Employee Management)</p> <ul style="list-style-type: none">• Task: Use AI to create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary.• Instructions:<ul style="list-style-type: none">○ 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. <p>PROMPT: Generate a Python class named Employee with attributes name, id, and salary. Add a method to calculate the yearly salary.</p> <p>CODE:</p> <pre>employee.py class Employee: def __init__(self, name, id, salary): self.name = name self.id = id self.salary = salary def calculate_yearly_salary(self): return self.salary * 12 def display_info(self): print(f"Employee Name: {self.name}") print(f"Employee ID: {self.id}") print(f"Monthly Salary: \${self.salary}") print(f"Yearly Salary: \${self.calculate_yearly_salary()}") # Example usage and output if __name__ == "__main__": # Create employee objects emp1 = Employee("John Doe", "E001", 5000) emp2 = Employee("Jane Smith", "E002", 6000) print("Employee 1:") emp1.display_info() print() print("Employee 2:") emp2.display_info() print() # Direct method calls print(f"John's yearly salary: \${emp1.calculate_yearly_salary()}")</pre>	

```
print()

# Direct method calls
print(f"John's yearly salary: ${emp1.calculate_yearly_salary()}")
print(f"Jane's yearly salary: ${emp2.calculate_yearly_salary()}")
```

Expected Output #1:

- A class with constructor, display_details(), and calculate_bonus() methods.

```
≡ text

Employee 1:
Employee Name: John Doe
Employee ID: E001
Monthly Salary: $5000
Yearly Salary: $60000

Employee 2:
Employee Name: Jane Smith
Employee ID: E002
Monthly Salary: $6000
Yearly Salary: $72000

John's yearly salary: $60000
Jane's yearly salary: $72000
```

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:
 - 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.

PROMPT:

Generate a Python function that displays all Automorphic numbers

between 1 and 1000 using a for loop.

CODE:

```
def is_automorphic(n):
    """Check if a number is automorphic"""
    square = n * n
    # Convert both numbers to strings to check if square ends with the number
    return str(square).endswith(str(n))

def find_automorphic_for_loop():
    """Find automorphic numbers using for loop"""
    automorphic_numbers = []
    for i in range(1, 1001):
        if is_automorphic(i):
            automorphic_numbers.append(i)
    return automorphic_numbers

def find_automorphic_while_loop():
    """Find automorphic numbers using while loop"""
    automorphic_numbers = []
    i = 1
    while i <= 1000:
        if is_automorphic(i):
            automorphic_numbers.append(i)
        i += 1
    return automorphic_numbers

# Test both implementations
if __name__ == "__main__":
    print("Finding Automorphic Numbers between 1 and 1000")
    print("=" * 50)
```

```
# Using for loop
print("\nUsing FOR loop:")
for_result = find_automorphic_for_loop()
print(f"Automorphic numbers found: {for_result}")
print(f"Count: {len(for_result)}")

# Using while loop
print("\nUsing WHILE loop:")
while_result = find_automorphic_while_loop()
print(f"Automorphic numbers found: {while_result}")
print(f"Count: {len(while_result)}")

# Verify both methods give same results
print(f"\nBoth methods give same results: {for_result == while_result}")

# Display individual automorphic numbers with their squares
print("\nDetailed verification:")
print("Number | Square | Is Automorphic?")
print("-" * 35)
for num in for_result:
    square = num * num
    print(f"{num:6} | {square:6} | Yes")
```

Expected Output #2:

- Correct implementation that lists Automorphic numbers using both loop types, with explanation.

```
Finding Automorphic Numbers between 1 and 1000
=====

Using FOR loop:
Automorphic numbers found: [1, 5, 6, 25, 76, 376, 625]
Count: 7

Using WHILE loop:
Automorphic numbers found: [1, 5, 6, 25, 76, 376, 625]
Count: 7

Both methods give same results: True

Detailed verification:
Number | Square | Is Automorphic?
-----
1 | 1 | Yes
5 | 25 | Yes
6 | 36 | Yes
25 | 625 | Yes
76 | 5776 | Yes
376 | 141376 | Yes
625 | 390625 | Yes
```

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

PROMPT:

Write Python code using nested if-elif-else statements to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating from 1 to 5.

CODE:

```

def classify_feedback(rating):
    """Classify feedback based on numerical rating using nested if-elif-else"""
    if rating >= 1 and rating <= 5:
        if rating == 5:
            return "Positive - Excellent service and experience!"
        elif rating == 4:
            return "Positive - Good service with minor issues"
        elif rating == 3:
            return "Neutral - Average experience, room for improvement"
        elif rating == 2:
            return "Negative - Below average, significant issues"
        elif rating == 1:
            return "Negative - Poor experience, major problems"
    else:
        return "Invalid rating - Please provide a rating between 1 and 5"

def analyze_feedback_detailed(rating):
    """More detailed feedback analysis with nested conditions"""
    if rating >= 1 and rating <= 5:
        if rating >= 4:
            if rating == 5:
                return "Positive - Outstanding! Highly recommend"
            else: # rating == 4
                return "Positive - Satisfactory experience"
        elif rating == 3:
            return "Neutral - Acceptable but could be better"
        else: # rating <= 2
            if rating == 2:
                return "Negative - Disappointing experience"
            else: # rating == 1

```

```

                return "Negative - Very poor, avoid this service"
    else:
        return "Invalid rating - Must be between 1 and 5"

# Test the functions
if __name__ == "__main__":
    print("Online Shopping Feedback Classifier")
    print("=" * 40)

    # Test all valid ratings
    test_ratings = [1, 2, 3, 4, 5, 0, 6]

    print("\nBasic Classification:")
    print("-" * 20)
    for rating in test_ratings:
        result = classify_feedback(rating)
        print(f"Rating {rating}: {result}")

    print("\nDetailed Classification:")
    print("-" * 20)
    for rating in test_ratings:
        result = analyze_feedback_detailed(rating)
        print(f"Rating {rating}: {result}")

    # Interactive testing
    print("\n" + "=" * 40)
    print("Interactive Testing:")
    print("Enter 'quit' to exit")

    while True:
        user_input = input("\nEnter a rating (1-5): ")
        if user_input.lower() == 'quit':
            break

```

```

if user_input.lower() == 'quit':
    break

try:
    rating = int(user_input)
    basic_result = classify_feedback(rating)
    detailed_result = analyze_feedback_detailed(rating)

    print(f"\nBasic Classification: {basic_result}")
    print(f"Detailed Classification: {detailed_result}")

except ValueError:
    print("Please enter a valid number between 1 and 5")

```

Expected Output #3:

- Feedback classification function with explanation and an alternative approach.

```

Online Shopping Feedback Classifier
=====

Basic Classification:
-----
Rating 1: Negative - Poor experience, major problems
Rating 2: Negative - Below average, significant issues
Rating 3: Neutral - Average experience, room for improvement
Rating 4: Positive - Good service with minor issues
Rating 5: Positive - Excellent service and experience!
Rating 0: Invalid rating - Please provide a rating between 1 and 5
Rating 6: Invalid rating - Please provide a rating between 1 and 5

Detailed Classification:
-----
Rating 1: Negative - Very poor, avoid this service
Rating 2: Negative - Disappointing experience
Rating 3: Neutral - Acceptable but could be better
Rating 4: Positive - Satisfactory experience
Rating 5: Positive - Outstanding! Highly recommend
Rating 0: Invalid rating - Must be between 1 and 5
Rating 6: Invalid rating - Must be between 1 and 5

=====

Interactive Testing:
Enter 'quit' to exit

Enter a rating (1-5): 4

```

Basic Classification: Positive - Good service with minor issues
Detailed Classification: Positive - Satisfactory experience

Enter a rating (1-5): 2

Basic Classification: Negative - Below average, significant issues
Detailed Classification: Negative - Disappointing experience

Enter a rating (1-5): quit

Task Description #4 (Loops – Prime Numbers in a Range)

- 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).

PROMPT:

Write a Python function that displays all prime numbers within a user-specified range (for example, 1 to 500) using a for loop.

CODE:

```
def is_prime(n):  
    """Check if a number is prime"""  
    if n < 2:  
        return False  
    if n == 2:  
        return True  
    if n % 2 == 0:  
        return False  
  
    # Check only up to square root for efficiency  
    for i in range(3, int(n ** 0.5) + 1, 2):  
        if n % i == 0:  
            return False  
    return True  
  
def find_primes(start, end):  
    """Find all prime numbers in the given range"""  
    primes = []  
    for num in range(start, end + 1):  
        if is_prime(num):  
            primes.append(num)  
    return primes
```

```
# Test the function
if __name__ == "__main__":
    # Find primes from 1 to 100
    start = 1
    end = 100

    print(f"Prime numbers from {start} to {end}:")
    print("-" * 30)

    prime_list = find_primes(start, end)

    for prime in prime_list:
        print(prime, end=" ")

    print(f"\n\nTotal prime numbers found: {len(prime_list)}")
```

Expected Output #4:

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

```
Prime numbers from 1 to 100:
-----
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

Total prime numbers found: 25
```

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:
 - Generate Library class code using AI.
 - Analyze if methods handle edge cases (e.g., issuing unavailable books).
 - Ask AI to add comments and documentation.

PROMPT:

Generate a Python class named Library with methods add_book(), issue_book(), and display_books()

CODE:


```

class Book:
    def __init__(self, book_id, title, author):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.available = True

class Library:
    def __init__(self):
        self.books = {}
        self.next_id = 1

    def add_book(self, title, author):
        """Add a new book to the library"""
        book_id = self.next_id
        self.books[book_id] = Book(book_id, title, author)
        self.next_id += 1
        print(f"Added: {title} by {author}")

    def issue_book(self, book_id, user_name):
        """Issue a book to a user"""
        if book_id not in self.books:
            print("Book not found!")
            return False

        book = self.books[book_id]
        if not book.available:
            print(f"'{book.title}' is not available")
            return False

        book.available = False
        print(f"Issued '{book.title}' to {user_name}")
        return True

    def return_book(self, book_id):

```

```

    def return_book(self, book_id):
        """Return a book to the library"""
        if book_id not in self.books:
            print("Book not found!")
            return False

        book = self.books[book_id]
        if book.available:
            print(f"'{book.title}' is already in library")
            return False

        book.available = True
        print(f"Returned '{book.title}'")
        return True

    def display_books(self):
        """Show all books and their status"""
        if not self.books:
            print("No books in library")
            return

        print("\nLibrary Books:")
        print("-" * 40)
        for book in self.books.values():
            status = "Available" if book.available else "Issued"
            print(f"ID: {book.book_id} | {book.title} | {book.author} | {status}")

Test the Library
__name__ == "__main__":
    # Create Library
    library = Library()

```

```
# Add books
print("Adding books:")
library.add_book("Python Basics", "John Smith")
library.add_book("Data Science", "Jane Doe")
library.add_book("Web Development", "Bob Johnson")

# Display all books
library.display_books()

# Issue books
print("\nIssuing books:")
library.issue_book(1, "Rithika")
library.issue_book(2, "Alex")

# Try to issue unavailable book
library.issue_book(1, "Tom")

# Display current status
library.display_books()

# Return a book
print("\nReturning book:")
library.return_book(1)

# Final display
library.display_books()
```

Expected Output #5:

- Library class with all methods, inline comments, and explanation.

Adding books:

Added: Python Basics by John Smith

Added: Data Science by Jane Doe

Added: Web Development by Bob Johnson

Library Books:

ID: 1 | Python Basics | John Smith | Available

ID: 2 | Data Science | Jane Doe | Available

ID: 3 | Web Development | Bob Johnson | Available

Issuing books:

Issued 'Python Basics' to Rithika

Issued 'Data Science' to Alex

'Python Basics' is not available

Library Books:

ID: 1 | Python Basics | John Smith | Issued

ID: 2 | Data Science | Jane Doe | Issued

ID: 3 | Web Development | Bob Johnson | Available

Returning book:

Returned 'Python Basics'

Library Books:

ID: 1 | Python Basics | John Smith | Available

ID: 2 | Data Science | Jane Doe | Issued

ID: 3 | Web Development | Bob Johnson | Available