AI ASSISTED CODING-6.3

LAB-ASSIGNMENT-6

A.SRICHALA

2303A52435

BT-31

Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals
Lab Objectives:
• To explore AI-powered auto-completion features for core Python constructs.
• To analyze how AI suggests logic for class definitions, loops, and conditionals.
• To evaluate the completeness and correctness of code generated by AI assistants.
Lab Outcomes (LOs):
After completing this lab, students will be able to:
• Use AI tools to generate and complete class definitions and methods.
• Understand and assess AI-suggested loops for iterative tasks.
• Generate conditional statements through prompt-driven suggestions.
• Critically evaluate AI-assisted code for correctness and clarity.

Task Description #1 (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 #1:
• Correct implementation that lists Automorphic numbers using both loop types, with explanation.

Prompt:-

Generate a Python program to display all Automorphic numbers between 1 and 1000 using a for loop and a while loop. Use a function to check whether a number is Automorphic and compare the execution time of both implementations."

CODE:-

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

import time as t

# -------- Using for loop --------
start_for = t.time()

automorphic_numbers = []
for i in range(1, 1001):
    if is_automorphic(i):
        automorphic_numbers.append(i)

end_for = t.time()

# -------- Using while loop (only for timing) --------
start_while = t.time()

i = 1
while i <= 1000:
    is_automorphic(i)
    i += 1

end_while = t.time()

# -------- Output --------
for num in automorphic_numbers:
    print(num, "is an Automorphic Number")

print("\nExecution Time:")
print("For loop time   :", end_for - start_for, "seconds")
print("While loop time :", end_while - start_while, "seconds")

print("\nComparison:")
if (end_for - start_for) < (end_while - start_while):
    print("For loop is faster than while loop")
else:
    print("While loop is faster than for loop")
```

OUTPUT:-

```
C:\Users\ankam>C:/Users/ankam/AppData/Local/Programs/P
1 is an Automorphic Number
5 is an Automorphic Number
6 is an Automorphic Number
25 is an Automorphic Number
76 is an Automorphic Number
376 is an Automorphic Number
625 is an Automorphic Number

Execution Time:
For loop time   : 0.0009987354278564453 seconds
While loop time : 0.0 seconds

Comparison:
While loop is faster than for loop
```

Explanation:-

1.The program uses a function to check whether a number is Automorphic by comparing the number with the last digits of its square.
2.A for loop is used to iterate from 1 to 1000 and display all Automorphic numbers.
3.The same logic is implemented using a while loop to achieve the same result.
4.The time module is used to calculate the execution time of both loop implementations.
5.Both loops produce identical output, with the for loop executing slightly faster than the while loop.

Task Description #2 (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 #2:
• Feedback classification function with explanation and an alternative approach.

Prompt:-

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

```python
#Generate a Python program to classify online shopping feedback as Positiv
def classify_feedback(rating):
    if rating >= 4 and rating <= 5:
        return "Positive"
    elif rating == 3:
        return "Neutral"
    elif rating >= 1 and rating <= 2:
        return "Negative"
    else:
        return "Invalid rating. Please provide a rating between 1 and 5."
def main():
    try:
        rating = int(input("Enter your rating (1-5): "))
        feedback = classify_feedback(rating)
        print(f"Your feedback is classified as: {feedback}")
    except ValueError:
        print("Invalid input. Please enter an integer between 1 and 5.")
if __name__ == "__main__":
    main()

```

Output:-

```
C:\Users\ankam>C:/Users/ankam/AppData/Local/Pr
Enter your rating (1-5): 5
Your feedback is classified as: Positive
```

Explanation:-

1.The classify_feedback(rating) function uses nested if-elif-else to classify ratings as Positive (4–5), Neutral (3), or Negative (1–2).

2.Any rating outside 1–5 returns an “Invalid rating” message.

3.The main() function takes user input, converts it to an integer, and calls the classification function.

4.try-except handles invalid (non-integer) input gracefully.

5.The program prints the feedback classification and runs only when executed directly (if __name__ == "__main__":).

Task 3: Statistical_operations

Define a function named statistical_operations(tuple_num) that performs the following statistical operations on a tuple of numbers:

• Minimum, Maximum

• Mean, Median, Mode
• Variance, Standard Deviation
While writing the function, observe the code suggestions provided by GitHub Copilot.Make decisions to accept, reject, or modify the suggestions based on their relevance and correctness

Prompt:-

#Define a function named statistical_operations(tuple_num) that performs the following statistical operations on a tuple of numbers: Minimum, MaximumMean, Median, Mode Variance, Standard Deviation While writing the function, observe the code suggestions provided by GitHub Copilot.Make decisions to accept, reject, or modify the suggestions based on their relevance and correctness

```python
#Define a function named statistical_operations(tuple_num) that performs the following stati
import statistics
import math
def statistical_operations(tuple_num):
    """
    Perform statistical operations on a tuple of numbers.

    Args:
        tuple_num (tuple): A tuple containing numerical values.

    Returns:
        dict: A dictionary containing the results of various statistical operations.
    """
    if not tuple_num:
        return "The input tuple is empty."

    stats = {}
    stats['Minimum'] = min(tuple_num)
    stats['Maximum'] = max(tuple_num)
    stats['Mean'] = statistics.mean(tuple_num)
    stats['Median'] = statistics.median(tuple_num)

    try:
        stats['Mode'] = statistics.mode(tuple_num)
    except statistics.StatisticsError:
        stats['Mode'] = "No unique mode found."

    stats['Variance'] = statistics.variance(tuple_num) if len(tuple_num) > 1 else 0
    stats['Standard Deviation'] = statistics.stdev(tuple_num) if len(tuple_num) > 1 else 0

    return stats
def main():
    # Example usage
    data = (1, 2, 2, 3, 4, 5, 5, 5)
    results = statistical_operations(data)
    for key, value in results.items():
        print(f"{key}: {value}")
if __name__ == "__main__":
    main()
```

Output:-

```
C:\Users\ankam>C:/Users/ankam/AppData/Local/Program
Minimum: 1
Maximum: 5
Mean: 3.375
Median: 3.5
Mode: 5
Variance: 2.5535714285714284
Standard Deviation: 1.5979898086569353
```

Explanation:-

1. The classify_feedback(rating) function uses nested if-elif-else to classify ratings as Positive (4–5), Neutral (3), or Negative (1–2).
2. Any rating outside 1–5 returns an “Invalid rating” message.
3. The main() function takes user input, converts it to an integer, and calls the classification function.
4. try-except handles invalid (non-integer) input gracefully.
5. The program prints the feedback classification and runs only when executed directly (if __name__ == "__main__":).

Task 4: Teacher Profile

• Prompt: Create a class Teacher with attributes teacher_id, name, subject, and experience. Add a method to display teacher details.

Expected Output: Class with initializer, method, and object creation.

Prompt:- Create a class named Teacher with attributes teacher_id, name, subject, and experience, and add a method to display teacher details.

Code:-

```
1
2   class Teacher:
3       """
4       Class to represent a Teacher profile
5       """
6       def __init__(self, teacher_id, name, subject, experience):
7           # Initialize attributes
8           self.teacher_id = teacher_id
9           self.name = name
10          self.subject = subject
11          self.experience = experience  # in years
12
13      def display_details(self):
14          """
15          Display teacher details
16          """
17          print(f"Teacher ID   : {self.teacher_id}")
18          print(f"Name         : {self.name}")
19          print(f"Subject      : {self.subject}")
20          print(f"Experience   : {self.experience} years")
21          print("-" * 30)
22
23  # Creating Teacher objects
24  teacher1 = Teacher(101, "Mr. Anil Kumar", "Mathematics", 8)
25  teacher2 = Teacher(102, "Ms. Priya Reddy", "Physics", 5)
26
27  # Displaying details
28  print("Teacher Profiles:")
29  teacher1.display_details()
30  teacher2.display_details()
31
```

Output:

```
C:\Users\ankam>C:/Users/ankam/AppData/
Teacher Profiles:
Teacher ID   : 101
Name         : Mr. Anil Kumar
Subject      : Mathematics
Experience   : 8 years
------------------------------
Teacher ID   : 102
Name         : Ms. Priya Reddy
Subject      : Physics
Experience   : 5 years
------------------------------
```

Explanation:-

1. Teacher class defines a blueprint for teacher profiles with attributes teacher_id, name, subject, and experience.
2. __init__ initializes these attributes when a new Teacher object is created.
3. display_details() prints all teacher information neatly using formatted strings.
4. Two Teacher objects (teacher1 and teacher2) are created with different details.
5. Calling display_details() on each object shows their profiles in a readable format.

Task #5 – Zero-Shot Prompting with Conditional Validation

Use zero-shot prompting to instruct an AI tool to generate a function that validates an Indian mobile number.

Requirements

• The function must ensure the mobile number:

o Starts with 6, 7, 8, or 9

o Contains exactly 10 digits

Expected Output

• A valid Python function that performs all required validations without using any input-output examples in the prompt.

Prompt:- Generate a Python function that validates an Indian mobile number with the following rules: Must start with 6, 7, 8, or 9 and Must contain exactly 10 digits.Do not use any input-output examples in the explanation.

Code:-

```python
def validate_indian_mobile(number):
    """
    Validate an Indian mobile number.

    Requirements:
    - Starts with 6, 7, 8, or 9
    - Exactly 10 digits

    Args:
        number (str): Mobile number to validate

    Returns:
        bool: True if valid, False otherwise
    """
    # Check if number has exactly 10 digits and all are digits
    if len(number) == 10 and number.isdigit():
        # Check if first digit is 6, 7, 8, or 9
        if number[0] in '6789':
            return True
    return False

def main():
    # Ask user for input
    mobile_number = input("Enter an Indian mobile number to validate: ")
    if validate_indian_mobile(mobile_number):
        print(f"{mobile_number} is a valid Indian mobile number.")
    else:
        print(f"{mobile_number} is NOT a valid Indian mobile number.")

if __name__ == "__main__":
    main()
```

Output:-

```
C:\Users\ankam>C:/Users/ankam/AppData/Local/Programs/Python/Python31
Enter an Indian mobile number to validate: 1234567890
1234567890 is NOT a valid Indian mobile number.
```

Explanation:-

1. validate_indian_mobile checks if the input is exactly 10 digits and all characters are numbers.

2. It then ensures the first digit is 6, 7, 8, or 9, as required for Indian mobile numbers.
3. If both conditions are satisfied, it returns True; otherwise, False.
4. main() prompts the user to enter a mobile number and calls the validation function.
5. It prints whether the entered number is valid or not.

Task Description #6 (Loops – Armstrong Numbers in a Range)
Task: Write a function using AI that finds all Armstrong numbers in a user-specified range (e.g., 1 to 1000).
Instructions:
• Use a for loop and digit power logic.
• Validate correctness by checking known Armstrong numbers (153, 370, etc.).
• Ask AI to regenerate an optimized version (using list comprehensions).
Expected Output #7:
• Python program listing Armstrong numbers in the range.
• Optimized version with explanation.

Prompt:-

Write a Python function that finds and prints all Armstrong numbers in a user-specified range (for example, 1 to 1000) using a for loop.in an optimized approach.

```
def find_armstrong_numbers_optimized(start, end):
    return [
        num for num in range(start, end + 1)
        if sum(int(digit) ** len(str(num)) for digit in str(num)) == num
    ]

def main_optimized():
    start = int(input("Enter start of range: "))
    end = int(input("Enter end of range: "))

    result = find_armstrong_numbers_optimized(start, end)

    print(f"Armstrong numbers between {start} and {end}:")
    print(result)

if __name__ == "__main__":
    main_optimized()

```

Output:-

```
C:\Users\ankam>C:/Users/ankam/AppData/
Enter start of range: 0
Enter end of range: 9
Armstrong numbers between 0 and 9:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

C:\Users\ankam>
```

Explanation:-

- The program takes a starting and ending number from the user.
- It checks each number in that range one by one.
- For each number, it calculates the sum of its digits raised to the power of the total digits.
- If this sum is equal to the original number, it is an Armstrong number.
- All such Armstrong numbers are collected and displayed as output.

Task Description #7 (Loops – Happy Numbers in a Range)
Task: Generate a function using AI that displays all Happy Numbers within a user-specified range (e.g., 1 to 500).
Instructions:
• Implement the logic using a loop: repeatedly replace a number with the sum of the squares of its digits until the result is either 1 (Happy Number) or enters a cycle (Not Happy).
• Validate correctness by checking known Happy Numbers (e.g., 1, 7, 10, 13, 19, 23, 28…).
• Ask AI to regenerate an optimized version (e.g., by using a set to detect cycles instead of infinite loops).
Expected Output #8:
• Python program that prints all Happy Numbers within a range.
• Optimized version using cycle detection with explanation.

Prompt:- Write a Python program to print all Happy Numbers in a user-defined range using loops, then generate an optimized version using a set for cycle detection and explain it briefly.

Code:-

```
def is_happy_number_optimized(n):
    """
    Determine if a number is Happy using a set to detect cycles.
    """
    seen = set()  # Using a set for faster lookup
    while n != 1 and n not in seen:
        seen.add(n)
        n = sum(int(digit)**2 for digit in str(n))
    return n == 1

def happy_numbers_in_range_optimized(start, end):
    """
    Generate Happy Numbers in a range using the optimized function.
    """
    return [num for num in range(start, end + 1) if is_happy_number_optimized(num)]

def main_optimized():
    start = int(input("Enter start of range: "))
    end = int(input("Enter end of range: "))

    result = happy_numbers_in_range_optimized(start, end)
    print(f"Happy Numbers between {start} and {end}:")
    print(result)

if __name__ == "__main__":
    main_optimized()
```

Output:-

```
C:\Users\ankam>C:/Users/ankam/AppData/Local/Programs/Python/Python312/pyt
Enter start of range: 0
Enter end of range: 9
Happy Numbers between 0 and 9:
[1, 7]
```

Explanation:- 1 The program checks each number within a user-given range.

2 .For each number, it replaces the number with the sum of the squares of its digits.

3.This process repeats until the number becomes 1 or a cycle is detected.

4.A set is used to store seen values and avoid infinite loops.

5.All numbers that reach 1 are printed as Happy Numbers.

Task Description #8 (Loops – Strong Numbers in a Range)
Task: Generate a function using AI that displays all Strong Numbers (sum offactorial of digits equals the number, e.g., 145 = 1!+4!+5!) within a given range.

Instructions:
• Use loops to extract digits and calculate factorials.
• Validate with examples (1, 2, 145).
• Ask AI to regenerate an optimized version (precompute digit factorials).
Expected Output #9:
• Python program that lists Strong Numbers.
• Optimized version with explanation.

Prompt:- Write a Python program that accepts student details from the user, stores them in a nested dictionary, and extracts specific information such as Full Name, Branch, and SGPA. The program should safely access nested keys and handle missing data using exception handling.

Code:-

```python
import math

def is_strong_number(n):
    """
    Check if a number is a Strong Number.

    A Strong Number is a number whose sum of factorials of digits equals the number itself.
    """
    # Precompute factorials of digits 0-9
    factorials = {str(i): math.factorial(i) for i in range(10)}

    sum_fact = sum(factorials[digit] for digit in str(n))
    return sum_fact == n

def strong_numbers_in_range(start, end):
    """
    Return a list of all Strong Numbers within the range [start, end].
    """
    return [num for num in range(start, end + 1) if is_strong_number(num)]

def main():
    start = int(input("Enter start of range: "))
    end = int(input("Enter end of range: "))

    result = strong_numbers_in_range(start, end)
    print(f"Strong Numbers between {start} and {end}:")
    print(result)

if __name__ == "__main__":
    main()
```

```
C:\Users\ankam>C:/Users/ankam/AppData/Local/Programs
Enter start of range: 0
Enter end of range: 9
Strong Numbers between 0 and 9:
[1, 2]
```

1. **Explanation:-** A Strong Number is a number equal to the sum of the factorials of its digits.
2. The function is_strong_number() calculates factorials of each digit and checks if their sum equals the number.
3. Factorials of digits 0–9 are precomputed using the math module for efficiency.
4. strong_numbers_in_range() finds all Strong Numbers between the given start and end values.
5. The main() function takes user input and displays all Strong Numbers in the specified range.

Task #9 – Few-Shot Prompting for Nested Dictionary Extraction
Objective
Use few-shot prompting (2–3 examples) to instruct the AI to create a function that parses a nested dictionary representing student information.
Requirements
• The function should extract and return:
o Full Name
o Branch
o SGPA
Expected Output
A reusable Python function that correctly navigates and extracts values from nested dictionaries based on the provided examples

Prompt:- Write a Python program that accepts student details from the user, stores them in a nested dictionary, and extracts specific information such as Full Name, Branch, and SGPA. The program should safely access nested keys and handle missing data using exception handling.

Code:-

```python
def extract_student_info(student_dict):
    """
    Extract Full Name, Branch, and SGPA from a nested student dictionary.
    """
    try:
        full_name = student_dict['personal']['full_name']
        branch = student_dict['academic']['branch']
        sgpa = student_dict['academic']['sgpa']
        return {
            'Full Name': full_name,
            'Branch': branch,
            'SGPA': sgpa
        }
    except KeyError as e:
        return f"Missing key in dictionary: {e}"

def main():
    # Manually input student details
    print("Enter student details:")
    full_name = input("Full Name: ")
    age = input("Age: ")  # optional
    branch = input("Branch: ")
    sgpa = input("SGPA: ")

    # Create nested dictionary manually
    student = {
        'personal': {'full_name': full_name, 'age': age},
        'academic': {'branch': branch, 'sgpa': float(sgpa)}
    }

    # Extract required info
    info = extract_student_info(student)
    print("\nExtracted Student Information:")
    print(info)

if __name__ == "__main__":
    main()
```

Output:-

```
C:\Users\ankam>C:/Users/ankam/AppData/Local/Programs/Python/Python312/python.exe c:/Users/
Enter student details:
Full Name: Priya
Age: 25
Branch: EEE
SGPA: 8.5

Extracted Student Information:
{'Full Name': 'Priya', 'Branch': 'EEE', 'SGPA': 8.5}
```

1. **Explanation:-** The program takes student details from the user and stores them in a nested dictionary.
2. Personal details are kept under the personal key, and academic details under the academic key.
3. The extract_student_info() function accesses nested keys to retrieve full name, branch, and SGPA.
4. A try–except block handles missing keys to prevent runtime errors.
5. The extracted information is displayed in a clean and readable format.

Task Description #10 (Loops – Perfect Numbers in a Range)

Task: Generate a function using AI that displays all Perfect Numbers within a user-specified range (e.g., 1 to 1000).

Instructions:

• A Perfect Number is a positive integer equal to the sum of its proper divisors (excluding itself).

o Example: 6 = 1 + 2 + 3, 28 = 1 + 2 + 4 + 7 + 14.

• Use a for loop to find divisors of each number in the range.

• Validate correctness with known Perfect Numbers (6, 28, 496…).

• Ask AI to regenerate an optimized version (using divisor check only up to √n).

Expected Output #12:

- Python program that lists Perfect Numbers in the given range.
- Optimized version with explanation.

Prompt:- Generate a Python program to find and display all Perfect Numbers in a user-defined range using loops. A Perfect Number equals the sum of its proper divisors (e.g., 6, 28). Optimize by checking divisors only up to √n.

Code:-

```python
import math

def is_perfect_number(n):
    """
    Check if a number is a Perfect Number.

    A Perfect Number is equal to the sum of its proper divisors (excluding itself).
    """
    if n < 2:
        return False

    sum_divisors = 1  # 1 is always a divisor
    # Check divisors from 2 up to sqrt(n)
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            sum_divisors += i
            if i != n // i:  # Add the complement divisor if not the square root
                sum_divisors += n // i
    return sum_divisors == n

def perfect_numbers_in_range(start, end):
    """
    Return a list of all Perfect Numbers in the range [start, end].
    """
    return [num for num in range(start, end + 1) if is_perfect_number(num)]

def main():
    start = int(input("Enter start of range: "))
    end = int(input("Enter end of range: "))

    result = perfect_numbers_in_range(start, end)
    print(f"Perfect Numbers between {start} and {end}:")
    print(result)

if __name__ == "__main__":
    main()

```

Output:-

```
C:\Users\ankam>C:/Users/ankam/AppData/Local/Programs/Python/Python312/
Enter start of range: 0
Enter end of range: 9
Perfect Numbers between 0 and 9:
[6]
```

Explanation:-

1. A Perfect Number equals the sum of its proper divisors (excluding itself).

2.The program loops through each number in the user-specified range.

3.For each number, it finds divisors only up to √n to optimize performance.

4.It sums the divisors (and their complements) and checks if the sum equals the number.

5.All numbers satisfying this condition are collected and displayed as Perfect Numbers.