

AI Assisted Coding

Assignment Lab - 6.3

Name : Shivanand Rama

Hall.t.No. : 2303A51037

Batch No. : 01

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:**
 - 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.

Expected Output #1:

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

For Loop Code :

```
1 # generate all automorphic numbers within a given range 1 to 1000 using for loop.
2
3 def is_automorphic(num):
4     square = num * num
5     return str(square).endswith(str(num))
6 automorphic_numbers = []
7 for i in range(1, 1001):
8     if is_automorphic(i):
9         automorphic_numbers.append(i)
10 print("Automorphic numbers between 1 and 1000 are:", automorphic_numbers)
```

Output :

```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● Automorphic numbers between 1 and 1000 are: [1, 5, 6, 25, 76, 376, 625]
○ PS C:\AI Assist> █
```

While Loop Code :

```
1 # generate all automorphic numbers within a given range 1 to 1000 using while loop.
2
3 def is_automorphic(num):
4     square = num * num
5     return str(square).endswith(str(num))
6 def generate_automorphic_numbers(start, end):
7     automorphic_numbers = []
8     num = start
9     while num <= end:
10         if is_automorphic(num):
11             automorphic_numbers.append(num)
12         num += 1
13     return automorphic_numbers
14 automorphic_numbers = generate_automorphic_numbers(1, 1000)
15 print("Automorphic numbers between 1 and 1000 are:", automorphic_numbers)
```

Output :

```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● Automorphic numbers between 1 and 1000 are: [1, 5, 6, 25, 76, 376, 625]
○ PS C:\AI Assist> █
```

Comparison of for loop & while loop :

```
on automorphic_timing_comparison.py
1 5 6 25 76 376 625

1 5 6 25 76 376 625

=====
TIMING COMPARISON
=====
For Loop Time: 0.00000000 seconds
While Loop Time: 0.00103951 seconds
Difference: 0.00103951 seconds
Winner: For 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:

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

Expected Output #2:

- Feedback classification function with explanation and an alternative Approach.

Code :

```
1  # Generate initial code using nested if-elif-else and givr the numerical rating (1-5).
2
3  def classify_feedback(rating):
4      if rating >= 4:
5          if rating == 5:
6              return "Positive: Excellent service!"
7          else:
8              return "Positive: Good service."
9      elif rating == 3:
10         return "Neutral: Average service."
11     else:
12         if rating == 2:
13             return "Negative: Poor service."
14         else:
15             return "Negative: Very poor service!"
16
17 # Example usage
18 print(classify_feedback(5)) # Positive: Excellent service!
19 print(classify_feedback(3)) # Neutral: Average service.
20 print(classify_feedback(1)) # Negative: Very poor service!
```

Output :

```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● Positive: Excellent service!
  Neutral: Average service.
  Negative: Very poor service!
○ PS C:\AI Assist> █
```

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.

Code :

```
1  # Define a function named statistical_operations(tuple_num) that calculates minimum, maximum, mean, median, mode, variance,  
2  # and standard deviation for a tuple of numbers.  
3  
4  import statistics  
5  def statistical_operations(tuple_num):  
6      if not tuple_num:  
7          return "The tuple is empty."  
8  
9      minimum = min(tuple_num)  
10     maximum = max(tuple_num)  
11     mean = statistics.mean(tuple_num)  
12     median = statistics.median(tuple_num)  
13  
14     try:  
15         mode = statistics.mode(tuple_num)  
16     except statistics.StatisticsError:  
17         mode = "No unique mode found"  
18  
19     variance = statistics.variance(tuple_num)  
20     std_dev = statistics.stdev(tuple_num)  
21  
22     return {  
23         "Minimum": minimum,  
24         "Maximum": maximum,  
25         "Mean": mean,  
26         "Median": median,  
27         "Mode": mode,  
28         "Variance": variance,  
29         "Standard Deviation": std_dev  
30     }  
31  
32 # Example usage  
33 data = (1, 2, 2, 3, 4)  
34 result = statistical_operations(data)  
35 for key, value in result.items():  
36     print(f"{key}: {value}")
```

Output :

```
● PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"  
Minimum: 1  
Maximum: 4  
Mean: 2.4  
Median: 2  
Mode: 2  
Variance: 1.3  
Standard Deviation: 1.140175425099138  
○ PS C:\AI Assist> █
```

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.

Code :

```
1  # Create a class Teacher with attributes teacher_id, name,  
2  # subject, and experience. Add a method to display teacher details.  
3  
4  class Teacher:  
5      def __init__(self, teacher_id, name, subject, experience):  
6          self.teacher_id = teacher_id  
7          self.name = name  
8          self.subject = subject  
9          self.experience = experience  
10  
11     def display_details(self):  
12         print(f"Teacher ID: {self.teacher_id}")  
13         print(f"Name: {self.name}")  
14         print(f"Subject: {self.subject}")  
15         print(f"Experience: {self.experience} years")  
16 # Creating an object of the Teacher class  
17 teacher1 = Teacher(101, "Alice Johnson", "Mathematics", 10)  
18 # Displaying teacher details  
19 teacher1.display_details()
```

Output :

```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"  
● Teacher ID: 101  
  Name: Alice Johnson  
  Subject: Mathematics  
  Experience: 10 years  
○ PS C:\AI Assist> █
```

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:
- Starts with 6, 7, 8, or 9
- Contains exactly 10 digits

Expected Output

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

Code :

```
1  # Write a Python function to validate an Indian mobile number. The number should start
2  # with 6, 7, 8, or 9 and must contain exactly 10 digits.
3
4  def validate_indian_mobile_number(mobile_number):
5
6      # Check if the length of the mobile number is exactly 10 digits
7      if len(mobile_number) != 10:
8          return False
9
10     # Check if the first digit is one of 6, 7, 8, or 9
11     if mobile_number[0] not in '6789':
12         return False
13
14     # Check if all characters in the mobile number are digits
15     if not mobile_number.isdigit():
16         return False
17
18     return True
19
20 # Example usage:
21 print(validate_indian_mobile_number("9876543210")) # Should return True
22 print(validate_indian_mobile_number("1234567890")) # Should return False
23 print(validate_indian_mobile_number("987654321"))  # Should return False
```

Output :

```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● True
  False
  False
○ PS C:\AI Assist> █
```

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 :

- Python program listing Armstrong numbers in the range.
- Optimized version with explanation.

Code :

```
1 # Write a Python function to find all Armstrong numbers in a given range using a for loop.
2 # Then generate an optimized version using list comprehension.
3
4 def find_armstrong_numbers_for_loop(start, end):
5     armstrong_numbers = []
6     for num in range(start, end + 1):
7         order = len(str(num))
8         sum_of_powers = 0
9         temp = num
10        while temp > 0:
11            digit = temp % 10
12            sum_of_powers += digit ** order
13            temp //= 10
14        if sum_of_powers == num:
15            armstrong_numbers.append(num)
16    return armstrong_numbers
17
18 def find_armstrong_numbers_list_comprehension(start, end):
19     return [num for num in range(start, end + 1) if sum(int(digit) ** len(str(num)) for digit in str(num)) == num]
20
21 # Example usage:
22 start_range = 100
23 end_range = 999
24 armstrong_numbers_for_loop = find_armstrong_numbers_for_loop(start_range, end_range)
25 armstrong_numbers_list_comprehension = find_armstrong_numbers_list_comprehension(start_range, end_range)
26 print("Armstrong numbers using for loop:", armstrong_numbers_for_loop)
27 print("Armstrong numbers using list comprehension:", armstrong_numbers_list_comprehension)
```

Output :

```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● Armstrong numbers using for loop: [153, 370, 371, 407]
  Armstrong numbers using list comprehension: [153, 370, 371, 407]
○ PS C:\AI Assist> █
```

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.

Code :

```
1 # Generate a Python function to print all Happy Numbers within a given range.
2 # Use loop logic and then optimize it using a set to detect cycles.
3
4 def is_happy_number(n):
5     seen = set()
6     while n != 1 and n not in seen:
7         seen.add(n)
8         n = sum(int(digit) ** 2 for digit in str(n))
9     return n == 1
10 def happy_numbers_in_range(start, end):
11     happy_numbers = []
12     for num in range(start, end + 1):
13         if is_happy_number(num):
14             happy_numbers.append(num)
15     return happy_numbers
16 # Example usage
17 start_range = 1
18 end_range = 100
19 happy_numbers = happy_numbers_in_range(start_range, end_range)
20 print(f"Happy numbers between {start_range} and {end_range}: {happy_numbers}")
```

Output :

```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● Happy numbers between 1 and 100: [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
○ PS C:\AI Assist>
```

Task Description #8 (Loops – Strong Numbers in a Range)

Task: Generate a function using AI that displays all Strong Numbers (sum of factorial 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.

Code :

```
1  # Write a Python function to display all Strong Numbers within a given range.
2  # Then optimize the program by precomputing factorials of digits.
3
4  import math
5  def is_strong_number(num):
6      """Check if a number is a Strong Number."""
7      sum_of_factorials = 0
8      temp = num
9      while temp > 0:
10         digit = temp % 10
11         sum_of_factorials += math.factorial(digit)
12         temp //= 10
13     return sum_of_factorials == num
14 def find_strong_numbers_in_range(start, end):
15     """Find all Strong Numbers in a given range."""
16     strong_numbers = []
17     for num in range(start, end + 1):
18         if is_strong_number(num):
19             strong_numbers.append(num)
20     return strong_numbers
21 # Example usage
22 start_range = 1
23 end_range = 1000
24 strong_numbers = find_strong_numbers_in_range(start_range, end_range)
25 print(f"Strong Numbers between {start_range} and {end_range}: {strong_numbers}")
```

Output :

```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● Strong Numbers between 1 and 1000: [1, 2, 145]
○ PS C:\AI Assist> █
```

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:
- Full Name
- Branch
- SGPA

Expected Output

A reusable Python function that correctly navigates and extracts values from nested dictionaries based on the provided examples

Code :

```
1  # Using a few examples, write a Python function that extracts full name, branch,  
2  # and SGPA from a nested student dictionary.  
3  
4  def extract_student_info(student_dict):  
5  
6      try:  
7          full_name = student_dict['personal_info']['name']['full_name']  
8          branch = student_dict['academic_info']['branch']  
9          sgpa = student_dict['academic_info']['grades']['SGPA']  
10  
11         return {  
12             'Full Name': full_name,  
13             'Branch': branch,  
14             'SGPA': sgpa  
15         }  
16     except KeyError as e:  
17         print(f"Key error: {e}")  
18         return None  
19  
20 # Example usage  
21 if __name__ == "__main__":  
22     student_data = {  
23         'personal_info': {  
24             'name': {  
25                 'full_name': 'John Doe',  
26                 'first_name': 'John',  
27                 'last_name': 'Doe'  
28             },  
29             'age': 20  
30         },  
31         'academic_info': {  
32             'branch': 'Computer Science',  
33             'grades': {  
34                 'SGPA': 8.5,  
35                 'CGPA': 8.3  
36             }  
37         }  
38     }  
39  
40     extracted_info = extract_student_info(student_data)  
41     print(extracted_info)
```

Output :

```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● {'Full Name': 'John Doe', 'Branch': 'Computer Science', 'SGPA': 8.5}
○ PS C:\AI Assist> █
```

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).
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 \sqrt{n}).

Code :

```
1  # Write a Python function to find all Perfect Numbers within a given range.
2  # Then generate an optimized version by checking divisors only up to the square root of the number.
3
4  def is_perfect_number(n):
5      if n < 2:
6          return False
7      divisors_sum = 1 # 1 is a proper divisor of any n > 1
8      for i in range(2, int(n**0.5) + 1):
9          if n % i == 0:
10             divisors_sum += i
11             if i != n // i:
12                 divisors_sum += n // i
13      return divisors_sum == n
14
15 def find_perfect_numbers_in_range(start, end):
16     perfect_numbers = []
17     for num in range(start, end + 1):
18         if is_perfect_number(num):
19             perfect_numbers.append(num)
20     return perfect_numbers
21
22 # User-specified range
23 start_range = 1
24 end_range = 1000
25 perfect_numbers = find_perfect_numbers_in_range(start_range, end_range)
26 print(f"Perfect numbers between {start_range} and {end_range}: {perfect_numbers}")
```

Output :

```
PS C:\AI Assist> & "C:\Program Files\Python311\python.exe" "c:/AI Assist/demo.py"
● Perfect numbers between 1 and 1000: [6, 28, 496]
○ PS C:\AI Assist> █
```