AI ASSISTED CODING

HALL TICKET : 2403A51270

BATCH :12

Question:

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 (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.

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.

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.

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:

o Get AI-generated code to list all primes using a for loop.

o Analyze the correctness and efficiency of the prime-checking logic.

o Ask AI to regenerate an optimized version (e.g., using the square root method).

Expected Output #4:

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

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.

Task 1 – Classes (Employee Management)

Prompt :

"Write a Python class Employee with attributes (name, id, salary). Add methods to display details, calculate yearly salary, and give a bonus."

Code:

class Employee:

def \_init\_(self, name, emp\_id, salary):

self.name = name

self.emp\_id = emp\_id

self.salary = salary

def display\_details(self):

print(f"Employee Name: {self.name}")

print(f"Employee ID: {self.emp\_id}")

print(f"Monthly Salary: {self.salary}")

def yearly\_salary(self):

return self.salary \* 12

def calculate\_bonus(self, bonus\_percent):

bonus = (self.salary \* bonus\_percent) / 100

self.salary += bonus

return self.salary

emp = Employee("John", 101, 50000)

emp.display\_details()

print("Yearly Salary:", emp.yearly\_salary())

print("Updated Salary after 10% bonus:", emp.calculate\_bonus(10))

output:

Employee Name: John

Employee ID: 101

Monthly Salary: 50000

Yearly Salary: 600000

Updated Salary after 10% bonus: 55000.0

Task 2 – Loops (Automorphic Numbers)

Prompt:

"Generate a function in Python that prints all Automorphic numbers between 1 and 1000 using a for loop."

Code:

def automorphic\_for():

print("Automorphic numbers between 1 and 1000 (for loop):")

for num in range(1, 1001):

sq = num \* num

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

print(num, end=" ")

automorphic\_for()

automorphic\_while()

output:

Automorphic numbers between 1 and 1000 (for loop):

1 5 6 25 76 376 625

Automorphic numbers between 1 and 1000 (while loop):

1 5 6 25 76 376 625

Task 3 – Conditional Statements (Feedback Classification)

Prompt:

"Write a Python function that classifies online shopping feedback (1–5) using nested if-elif-else: Positive, Neutral, or Negative."

Code:

def feedback\_classification(rating):

if rating == 5:

return "Positive Feedback"

elif rating == 4:

return "Positive Feedback"

elif rating == 3:

return "Neutral Feedback"

elif rating == 2:

return "Negative Feedback"

elif rating == 1:

return "Negative Feedback"

else:

return "Invalid Rating"

print("Rating: 5 →", feedback\_classification(5))

print("Rating: 3 →", feedback\_classification(3))

print("Rating: 1 →", feedback\_dict(1))

print("Rating: 7 →", feedback\_dict(7))

output:

Rating: 5 → Positive Feedback

Rating: 3 → Neutral Feedback

Rating: 1 → Negative Feedback

Rating: 7 → Invalid Rating

Task 4 – Loops (Prime Numbers)

Prompt:

"Write a Python function that prints all prime numbers between 1 and 500 using a for loop."

Code:

def primes\_in\_range(start, end):

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

for num in range(start, end + 1):

if num > 1:

for i in range(2, num):

if num % i == 0:

break

else:

print(num, end=" ")

primes\_in\_range(1, 50)

optimized\_primes(1, 50)

output:

Prime numbers between 1 and 50:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47

Optimized prime numbers between 1 and 50:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47

Task 5 – Classes (Library System)

Prompt:

"Create a Python class Library with methods add\_book(), issue\_book(), and display\_books(). Include comments and handle cases where a book is not available."

Code:

class Library:

def \_init\_(self):

self.books = []

def add\_book(self, book\_name):

"""Adds a book to the library collection"""

self.books.append(book\_name)

print(f'"{book\_name}" has been added to the library.')

def display\_books(self):

"""Displays all books in the library"""

if self.books:

print("Books available in the library:")

for book in self.books:

print(f"- {book}")

else:

print("No books available in the library.")

def issue\_book(self, book\_name):

"""Issues a book if available, else shows an error"""

if book\_name in self.books:

self.books.remove(book\_name)

print(f'"{book\_name}" has been issued.')

else:

print(f'Sorry, "{book\_name}" is not available.')

lib = Library()

lib.add\_book("Python Programming")

lib.add\_book("Data Science")

lib.display\_books()

lib.issue\_book("Python Programming")

lib.issue\_book("Machine Learning")

lib.display\_books()

output:

"Python Programming" has been added to the library.

"Data Science" has been added to the library.

Books available in the library:

- Python Programming

- Data Science

"Python Programming" has been issued.

Sorry, "Machine Learning" is not available.

Books available in the library:

- Data Science