

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	Academic Year:2025-2026
Course Coordinator Name		Venkataramana Veeramsetty	
Instructor(s) Name		Dr. V. Venkataramana (Co-ordinator)	
		Dr. T. Sampath Kumar	
		Dr. Pramoda Patro	
		Dr. Brij Kishor Tiwari	
		Dr.J.Ravichander	
		Dr. Mohammand Ali Shaik	
		Dr. Anirodh Kumar	
		Mr. S.Naresh Kumar	
		Dr. RAJESH VELPULA	
		Mr. Kundhan Kumar	
		Ms. Ch.Rajitha	
		Mr. M Prakash	
		Mr. B.Raju	
		Intern 1 (Dharma teja)	
		Intern 2 (Sai Prasad)	
		Intern 3 (Sowmya)	
		NS_2 (Mounika)	
Course Code	24CS002PC215	Course Title	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week5 - Monday	Time(s)	
Duration	2 Hours	Applicable to Batches	
AssignmentNumber:10.1(Present assignment number)/24(Total number of assignments)			
Q.No.	Question		Expected Time to complete
1	Lab 10 – Code Review and Quality: Using AI to Improve Code Quality and Readability Lab Objectives <ul style="list-style-type: none">Use AI for automated code review and quality enhancement.Identify and fix syntax, logical, performance, and security issues in Python code.Improve readability and maintainability through structured refactoring and comments.		Week5 - Monday

- Apply prompt engineering for targeted improvements.
- Evaluate AI-generated suggestions against PEP 8 standards and software engineering best practices

Task Description #1 – Syntax and Logic Errors

Task: Use AI to identify and fix syntax and logic errors in a faulty Python script.

Sample Input Code:

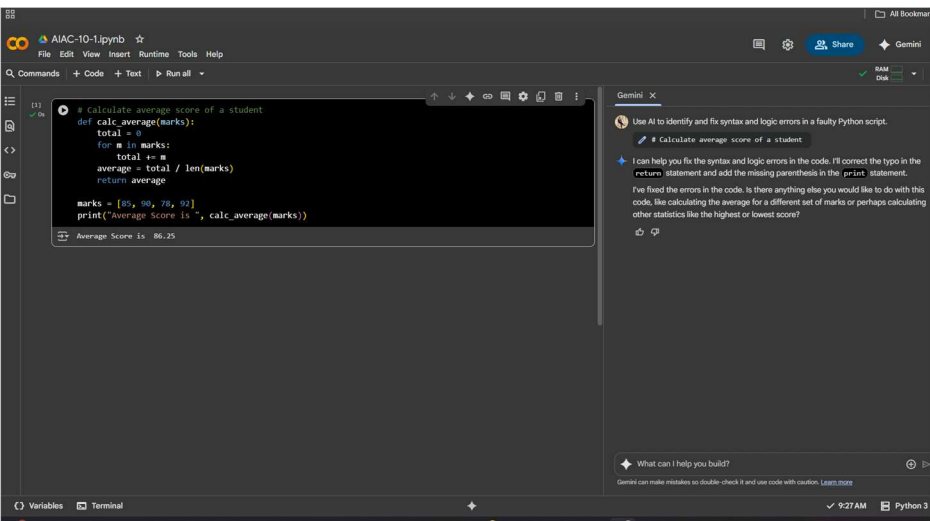
```
# Calculate average score of a student
def calc_average(marks):
    total = 0
    for m in marks:
        total += m
    average = total / len(marks)
    return avrage # Typo here
```

```
marks = [85, 90, 78, 92]
print("Average Score is ", calc_average(marks))
```

Expected Output:

- Corrected and runnable Python code with explanations of the fixes.

OUTPUT:



Task Description #2 – PEP 8 Compliance

Task: Use AI to refactor Python code to follow PEP 8 style guidelines.

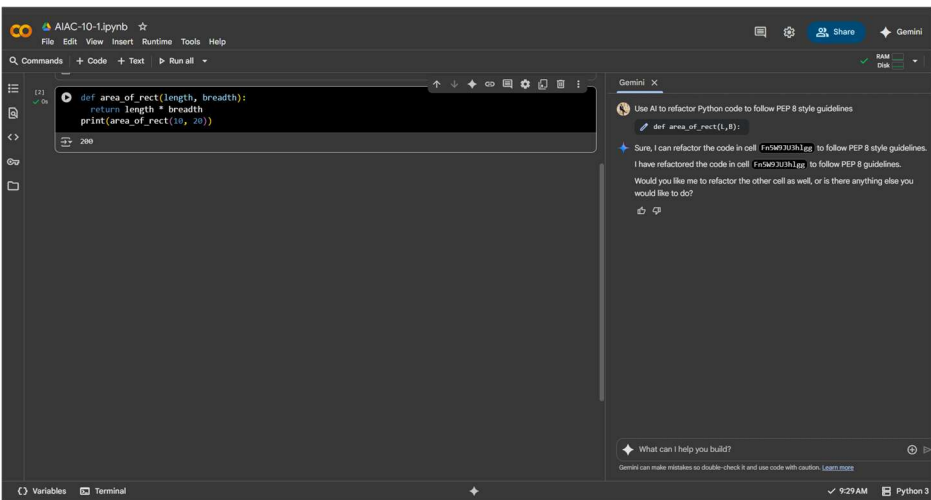
Sample Input Code:

```
def area_of_rect(L,B):return L*B
print(area_of_rect(10,20))
```

Expected Output:

- Well-formatted PEP 8-compliant Python code.

OUTPUT:



Task Description #3 – Readability Enhancement

Task: Use AI to make code more readable without changing its logic.

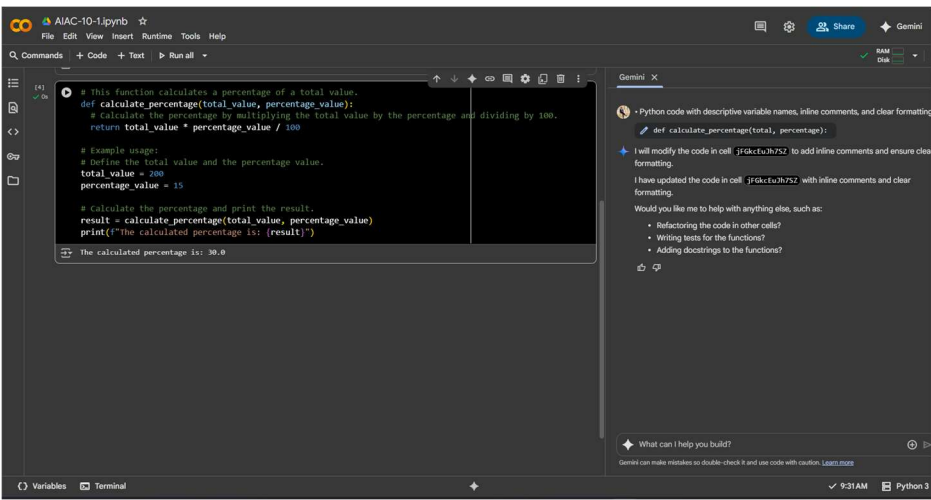
Sample Input Code:

```
def c(x,y):  
    return x*y/100  
a=200  
b=15  
print(c(a,b))
```

Expected Output:

- Python code with descriptive variable names, inline comments, and clear formatting.

OUTPUT:



Task Description #4 – Refactoring for Maintainability

Task: Use AI to break repetitive or long code into reusable functions.

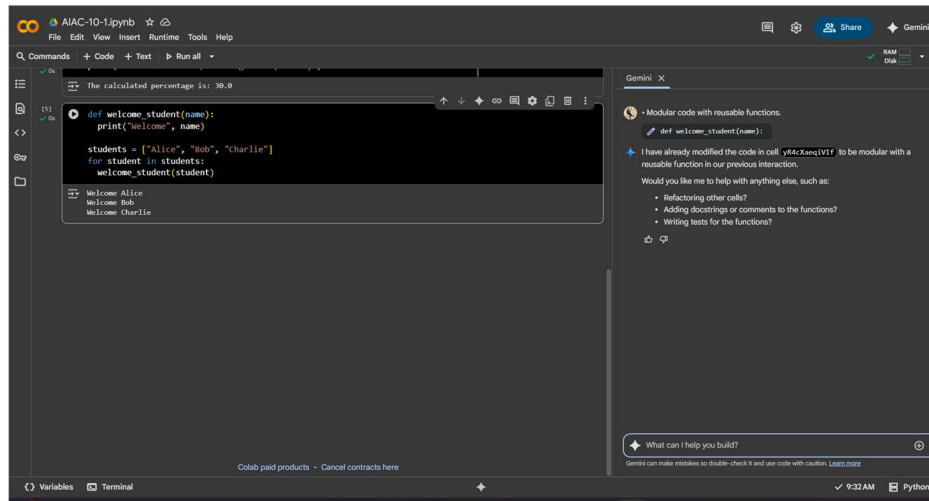
Sample Input Code:

```
students = ["Alice", "Bob", "Charlie"]  
print("Welcome", students[0])  
print("Welcome", students[1])  
print("Welcome", students[2])
```

Expected Output:

- Modular code with reusable functions.

OUTPUT:



The screenshot shows a Jupyter Notebook interface with a code cell containing the refactored Python code. The code defines a function `welcome_student(name)` that prints a welcome message for a given name, and then iterates over a list of student names, calling the function for each. The output of the code cell shows the expected output: "Welcome Alice", "Welcome Bob", and "Welcome Charlie". To the right of the code cell, the Gemini AI assistant's response is visible, showing a message that says "Modular code with reusable functions." and a code snippet for the `welcome_student(name)` function. The assistant also mentions that it has already modified the code in cell y44Xaeq1V17 to be modular with a reusable function in our previous interaction. Below this, it asks "Would you like me to help with anything else, such as:" and lists three suggestions: "Refactoring other cells?", "Adding docstrings or comments to the functions?", and "Writing tests for the functions?". At the bottom of the Gemini panel, there is a prompt "What can I help you build?" and a disclaimer "Gemini can make mistakes so double-check it and use code with caution. Learn more".

Task Description #5 – Performance Optimization

Task: Use AI to make the code run faster.

Sample Input Code:

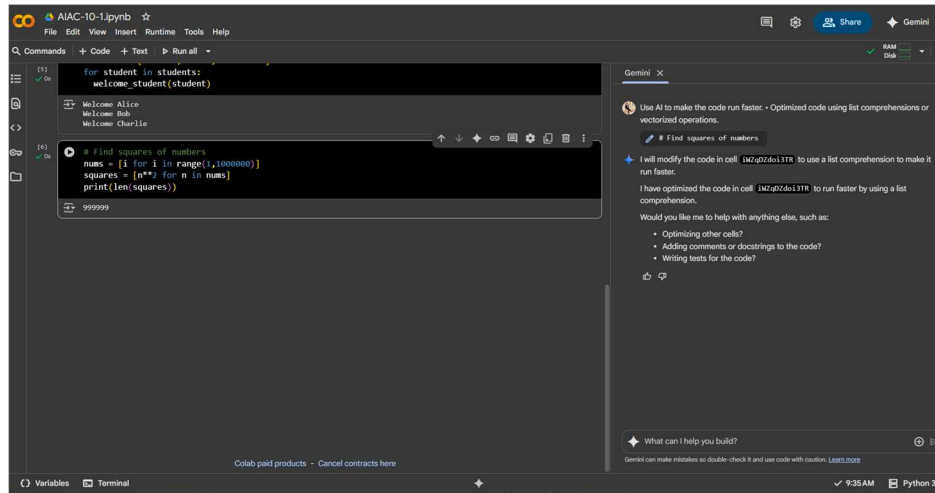
```
# Find squares of numbers  
nums = [i for i in range(1,1000000)]  
squares = []  
for n in nums:  
    squares.append(n**2)  
print(len(squares))
```

Expected Output:

- Optimized code using list comprehensions or vectorized

operations.

OUTPUT:



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell contains a loop that prints 'Welcome' for each student in a list. The second cell contains a loop that calculates the squares of numbers from 1 to 1,000,000. The output of the second cell is '999999'. On the right side, there is a Gemini chat window with suggestions for optimizing the code using list comprehensions and vectorized operations. The chat window also includes a prompt 'What can I help you build?' and a disclaimer 'Gemini can make mistakes so double-check it and use code with caution. L8870.0302'.

```
for student in students:
    welcome_student(student)

Welcome Alice
Welcome Bob
Welcome Charlie
```

```
# Find squares of numbers
nums = [i for i in range(1,1000000)]
squares = [n**2 for n in nums]
print(len(squares))

999999
```

Task Description #6 – Complexity Reduction

Task: Use AI to simplify overly complex logic.

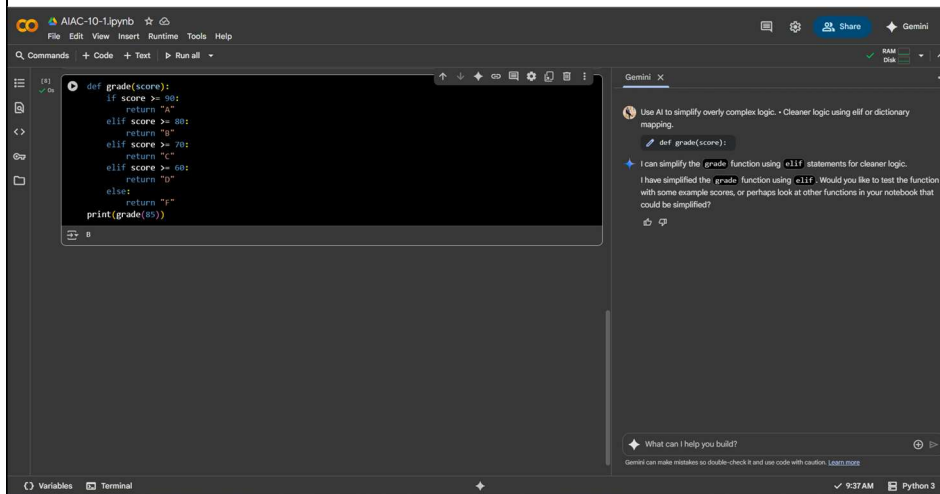
Sample Input Code:

```
def grade(score):
    if score >= 90:
        return "A"
    else:
        if score >= 80:
            return "B"
        else:
            if score >= 70:
                return "C"
            else:
                if score >= 60:
                    return "D"
                else:
                    return "F"
```

Expected Output:

- Cleaner logic using elif or dictionary mapping.

OUTPUT:



The screenshot shows a Jupyter Notebook environment with a dark theme. The main code cell contains a Python function named `grade` that takes a score as input and returns a letter grade based on the following logic: `score >= 90` returns "A", `score >= 80` returns "B", `score >= 70` returns "C", `score >= 60` returns "D", and otherwise returns "F". The function is tested with `print(grade(85))`. To the right, a Gemini AI chat window is open, displaying a suggestion to simplify the logic using `elif` or dictionary mapping. The chat window also shows a simplified version of the `grade` function using `elif` statements. The bottom status bar indicates the environment is Python 3.

```
def grade(score):  
    if score >= 90:  
        return "A"  
    elif score >= 80:  
        return "B"  
    elif score >= 70:  
        return "C"  
    elif score >= 60:  
        return "D"  
    else:  
        return "F"  
print(grade(85))
```

Use AI to simplify overly complex logic. • Cleaner logic using `elif` or dictionary mapping.

`def grade(score):`

I can simplify the `grade` function using `elif` statements for cleaner logic. I have simplified the `grade` function using `elif`. Would you like to test the function with some example scores, or perhaps look at other functions in your notebook that could be simplified?

What can I help you build?

2403A51399
Akshar Ganji
Batch -18