

Assignment 10.1 Ai Assisted Coding

Htno:2303a51305

Btno:05

Task 1: Syntax and Logic Errors Use AI to identify and fix syntax and logic errors in a faulty Python script.

Prompt:

Act as a Python debugging expert. Analyze the faulty Python script below and:

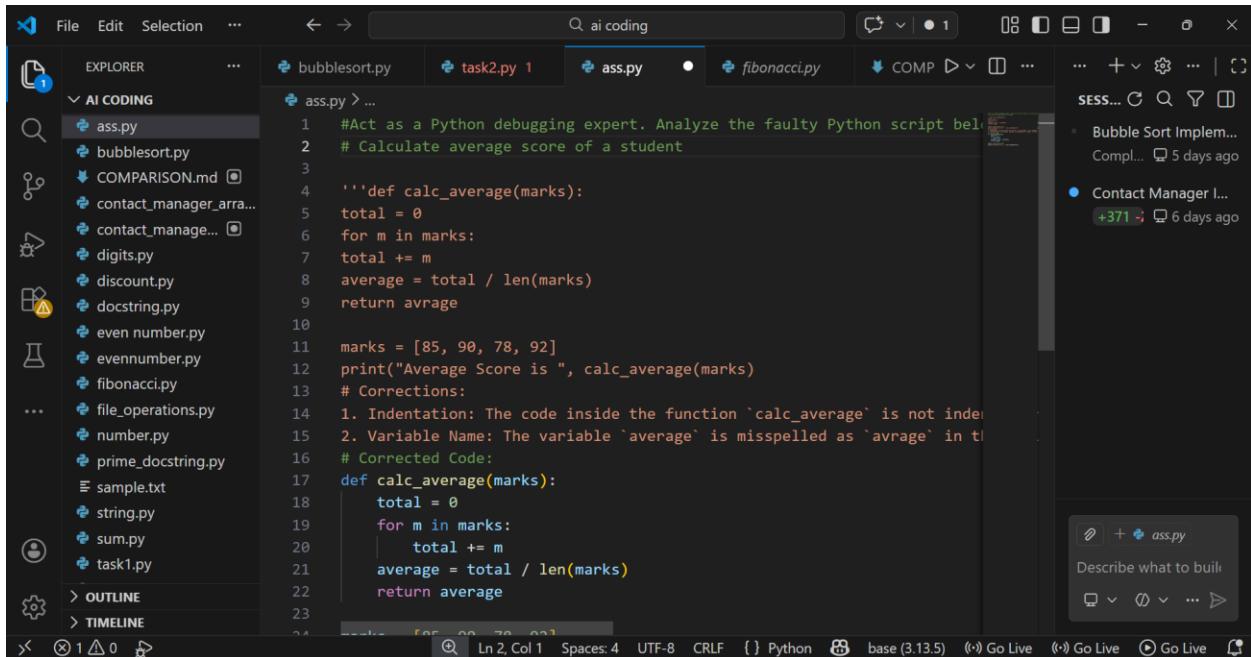
- Identify syntax errors and logic errors
- Fix the code so it runs correctly
- Explain the corrections briefly
- Show the corrected runnable code

Faulty code:

Calculate average score of a student

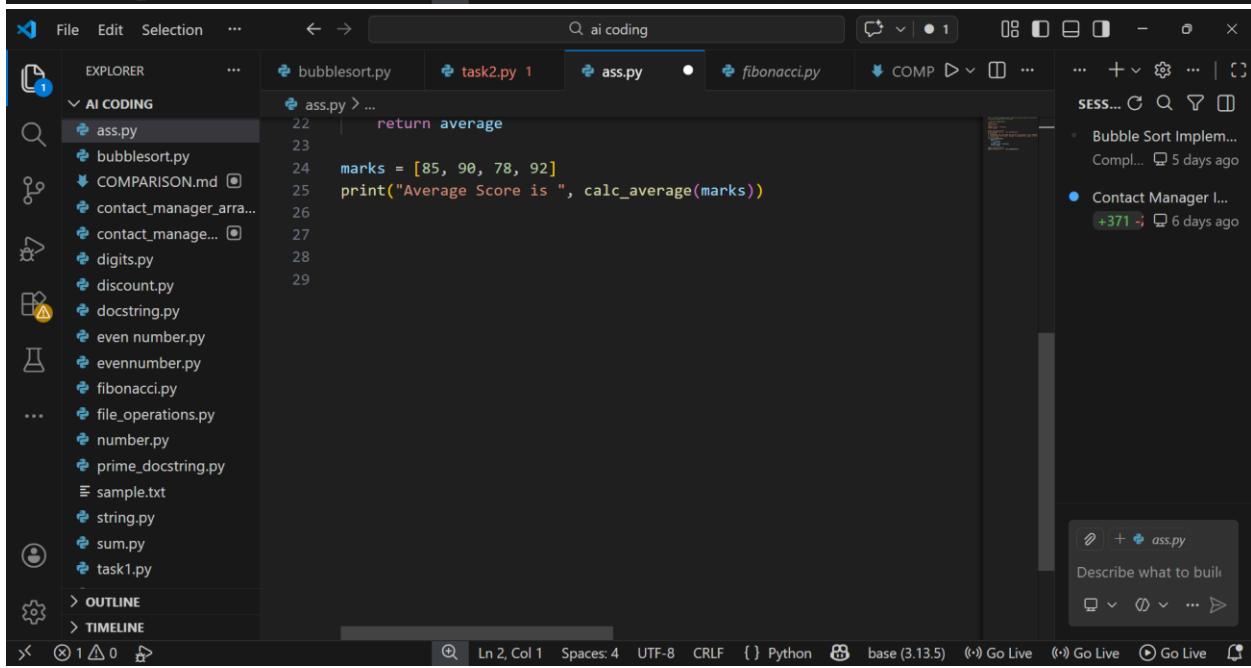
```
def calc_average(marks):
    total = 0
    for m in marks:
        total += m
    average = total / len(marks)
    return avgage
marks = [85, 90, 78, 92]
print("Average Score is ", calc_average(marks))
```

Code:



The screenshot shows the VS Code interface with the 'AI CODING' extension active. The Explorer sidebar on the left lists various Python files. The main editor area displays the content of 'ass.py'. The code is annotated with several red underlines and error messages, indicating issues such as indentation and variable spelling.

```
#Act as a Python debugging expert. Analyze the faulty Python script below.
# Calculate average score of a student
'''def calc_average(marks):
total = 0
for m in marks:
total += m
average = total / len(marks)
return avrage
marks = [85, 90, 78, 92]
print("Average Score is ", calc_average(marks))
# Corrections:
1. Indentation: The code inside the function `calc_average` is not indented correctly.
2. Variable Name: The variable `average` is misspelled as `avrage` in the return statement.
# Corrected Code:
def calc_average(marks):
    total = 0
    for m in marks:
        total += m
    average = total / len(marks)
    return average
```



The screenshot shows the same VS Code interface after the corrections have been applied. The code now runs correctly, calculating the average of the marks list.

```
def calc_average(marks):
    total = 0
    for m in marks:
        total += m
    average = total / len(marks)
    return average
marks = [85, 90, 78, 92]
print("Average Score is ", calc_average(marks))
```

Output:

The screenshot shows a VS Code interface with the following details:

- Explorer:** Shows a tree view of files in the "AI CODING" folder, including "ass.py", "bubblesort.py", "COMPARISON.md", "contact_manager_arr...", "digits.py", "discount.py", "docstring.py", "even number.py", "evennumber.py", "fibonacci.py", "file_operations.py", "number.py", "prime_docstring.py", "sample.txt", "string.py", "sum.py", and "task1.py".
- Terminal:** Displays a command-line session where a user is trying to run a script. The error message is:


```
+ (C:\Users\sneha\miniconda3\shell\condabin\conda-hook.ps1) ; (conda ac ...
+ ~~~~~
+ CategoryInfo          : SecurityError: () [], PSNotSupportedException
+ FullyQualifiedErrorId : UnauthorizedAccess
conda : The term 'conda' is not recognized as the name of a cmdlet, function,
script file, or operable program. Check the spelling of the name, or if a
path was included, verify that the path is correct and try again.
At line:1 char:62
+ ... sers\sneha\miniconda3\shell\condabin\conda-hook.ps1) ; (conda activat ...
...
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (conda:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException
```
- Status Bar:** Shows the current file is "ass.py", the language mode is "Python", and the base version is "3.13.5".

Explanation:

-->Removed invalid characters — Bullet symbol (•) was causing a syntax error because Python accepts only valid code characters.

-->Fixed typo — Changed avgare to average so the correct variable is returned.

-->Corrected indentation — Proper indentation was added inside the function and loop (Python requires indentation).

-->Closed parenthesis — Added the missing) in the print() statement to avoid syntax error.

-->Ensured correct logic — Total marks are summed and divided by number of subjects to correctly compute the average.

Task 2: PEP 8 Compliance .Use AI to refactor Python code to follow PEP 8 style

Guidelines.

Prompt

Act as a Python code reviewer. Refactor the following Python code to follow PEP 8 style guidelines.

Code:

```
def area_of_rect(L,B) : return L*B
```

```
print(area_of_rect(10,20))
```

Code:

The screenshot shows a Visual Studio Code (VS Code) interface. The title bar includes 'File', 'Edit', 'Selection', and a search bar containing 'ai coding'. The left sidebar has icons for Explorer, AI Coding, Search, Find, Contact Manager, Docstring, Even Number, Fibonacci, File Operations, Number, Prime Docstring, Sample Text, String, Sum, Task 1, Outline, and Timeline. The main area displays a Python file named 'ass.py' with the following code:

```
1  #Act as a Python code reviewer. Refactor the following Python code to fit the requirements.
2  def area_of_rect(L,B) : return L*B
3  print(area_of_rect(10,20))
4  def area_of_rect(length, breadth):
5
6      """Calculate the area of a rectangle given its length and breadth."""
7      return length * breadth
8  # Example usage
9  print(area_of_rect(10, 20))
```

The right sidebar shows a 'SESSIONS' section with a list of items: 'Bubble Sort Implement...' (Completed 5 days ago), 'Contact Manager I...' (Completed 6 days ago with 371 changes), and a 'Create New Session' button. A bottom right panel says 'Describe what to build'.

Output:

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top bar includes 'File', 'Edit', 'Selection', '...', '←', '→', and a search bar containing 'ai coding'. Below the search bar is the Explorer sidebar with sections like 'EXPLORER', 'AI CODING' (containing files like 'ass.py', 'bubblesort.py', 'task2.py'), 'COMPARISON.md', 'contact_manager_array.py', 'contact_manager.py', 'digits.py', 'discount.py', 'docstring.py', 'even number.py', 'evennumber.py', 'fibonacci.py', 'file_operations.py', 'number.py', 'prime_docstring.py', 'sample.txt', 'string.py', 'sum.py', and 'task1.py'), 'OUTLINE', and 'TIMELINE'. The main workspace has tabs for 'bubblesort.py', 'task2.py', 'ass.py' (which is currently active), and 'fibonaccipy'. Below the tabs are 'PROBLEMS' (with 1 error), 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'POSTMAN CONSOLE'. The 'TERMINAL' tab is selected, showing a Python session. The terminal output is as follows:

```
+ The ampersand (&) character is not allowed. The & operator is reserved for future use; wrap an ampersand in double quotation marks ("&") to pass it as part of a string.  
+ CategoryInfo          : ParserError: () [], ParentContainsErrorRecordException  
+ FullyQualifiedErrorId : AmpersandNotAllowed  
  
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding> python ass.py  
Python was not found; run without arguments to install from the Microsoft Store , or disable this shortcut from Settings > Apps > Advanced app settings > App execution aliases.  
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding> python ass.py  
Python was not found; run without arguments to install from the Microsoft Store , or disable this shortcut from Settings > Apps > Advanced app settings > App execution aliases.  
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding> & C:\Users\sneha\anaconda3\python.exe "c:/Users/sneha/OneDrive/Documents/Desktop/ai coding/ass.py"  
200  
200  
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding>
```

Explanation:

-->Used proper function naming — Changed variable names to lowercase with underscores (e.g., length, breadth) for readability.

-->Added correct spacing — Included spaces after commas and around operators (*) as per PEP 8 rules.

-->Split into multiple lines — Avoided writing the function in one line to improve clarity.

-->Removed extra spaces before colon — Ensured correct syntax formatting (def func()):).

-->Improved readability — Structured the code neatly so it is easier to understand and maintain.

Task 3: Readability Enhancement. Use AI to make code more readable without changing its Logic.

Prompt:

You are given a Python program that works correctly but is poorly readable due to unclear variable names,

lack of comments, and improper formatting.

Improve the readability of the code without changing its logic or output.

Replace unclear variable and function names with descriptive names.

Format the code clearly using proper indentation and spacing according to PEP 8 standards.

Do not modify the logic, calculations, or output of the program.

Input Code:

```
def c(x,y):
```

```
    return x*y/100
```

```
a=200
```

```
b=15
```

```
print(c(a,b))
```

Output:

Readable, well-formatted Python code

Same functionality and output as the original program.

Code:

The screenshot shows a code editor window titled "ai coding". The tabs at the top include "bubblesort.py", "task2.py 1", "ass.py", "fibonacci.py", "COMP", and "...". The current file is "ass.py". The code itself is as follows:

```
1  #You are given a Python program that works correctly but is poorly read
2  '''lack of comments, and improper formatting.
3  Improve the readability of the code without changing its logic or output.
4  Replace unclear variable and function names with descriptive names.
5  Format the code clearly using proper indentation and spacing according
6  | Do not modify the logic, calculations, or output of the program.
7  Input Code:
8  def c(x,y):
9      return x*y/100
10 a=200
11 b=15
12 print(c(a,b))
13 Output:
14 Readable, well-formatted Python code
15 Same functionality and output as the original program'''
16 def calculate_percentage(value, percentage):
17     """Calculate the percentage of a given value."""
18     return value * percentage / 100
19 # Sample values
20 total_amount = 200
21 percentage_value = 15
22 # Calculate and print the result
23 result = calculate_percentage(total_amount, percentage_value)
```

At the bottom of the editor, status indicators show "Ln 8, Col 1", "Spaces: 4", "UTF-8", "CRLF", "Python", "base (3.13.5)", and "Go Live".

Output:

```
ed\libs\debugpy\launcher' '53325' '--' 'c:\Users\sneha\OneDrive\Documents\Desktop\ai coding\ass.py'
30.0
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding> []
```

Explanation:

-->Improved indentation — Proper spacing was added so Python can correctly understand the function block.

-->Used meaningful names — Replaced short names like c, x, y with descriptive names to make the code easier to understand.

-->Added comments — Inline comments explain what each part of the code does.

-->Formatted clearly — Code was arranged neatly with proper line breaks and spacing.

Task 4: Refactoring for Maintainability. Use AI to break repetitive or long code into reusable Functions.

Prompt:

Act as a Python refactoring assistant. Refactor the following code to improve maintainability by removing repetition and creating reusable functions.

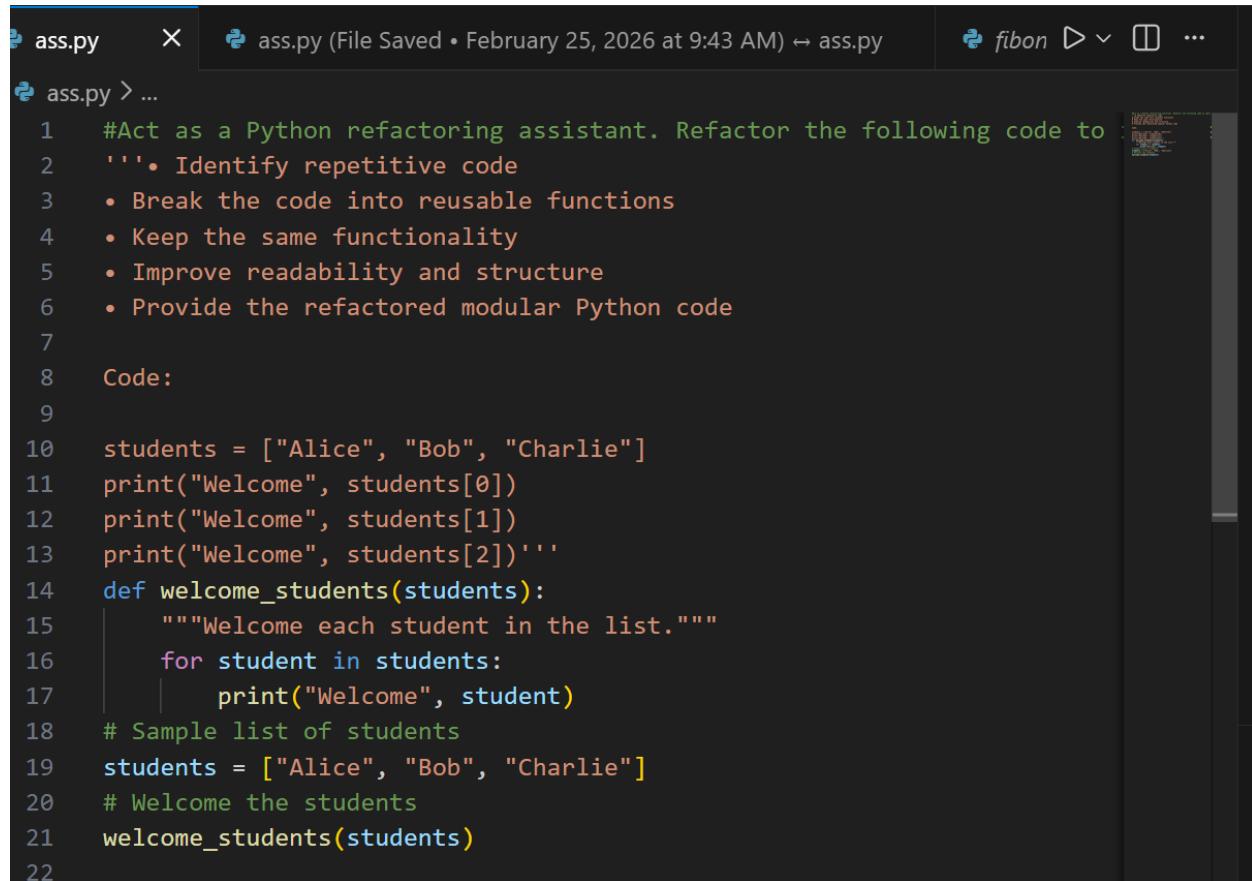
- Identify repetitive code
- Break the code into reusable functions
- Keep the same functionality
- Improve readability and structure
- Provide the refactored modular Python code

Code:

```
students = ["Alice", "Bob", "Charlie"]

print("Welcome", students[0])
print("Welcome", students[1])
print("Welcome", students[2])
```

Code:



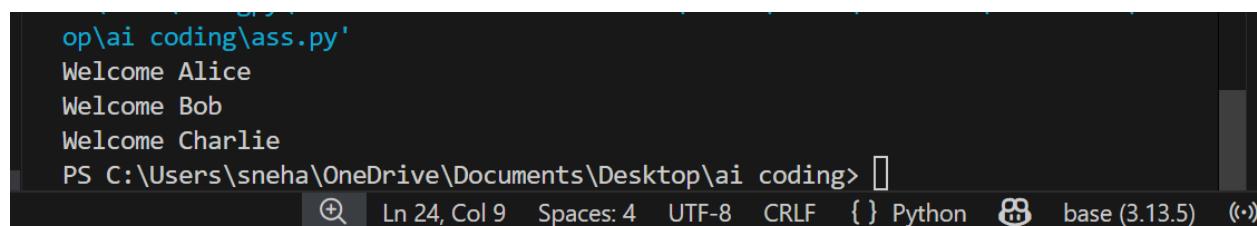
```
ass.py ass.py (File Saved • February 25, 2026 at 9:43 AM) ↔ ass.py fibon ▷ ⌂ ...
```

```
#Act as a Python refactoring assistant. Refactor the following code to:
'''• Identify repetitive code
• Break the code into reusable functions
• Keep the same functionality
• Improve readability and structure
• Provide the refactored modular Python code

Code:

students = ["Alice", "Bob", "Charlie"]
print("Welcome", students[0])
print("Welcome", students[1])
print("Welcome", students[2])
def welcome_students(students):
    """Welcome each student in the list."""
    for student in students:
        print("Welcome", student)
# Sample list of students
students = ["Alice", "Bob", "Charlie"]
# Welcome the students
welcome_students(students)
```

Output:



```
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding> op\ai coding\ass.py
Welcome Alice
Welcome Bob
Welcome Charlie
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding>
```

Explanation:

-->Removed repetition — Instead of writing multiple print statements, repetitive code is replaced with a reusable approach.

-->Created a function — A function is introduced to handle welcoming students, making the code modular.

-->Used iteration — A loop is used to process all students, reducing code length.

-->Improved maintainability — Future changes (like modifying the message) can be done in one place.

-->Enhanced readability — The structure is cleaner and easier to understand.

Task 5: Performance Optimization Task. Use AI to make the code run faster. Sample Input Code:

Prompt:

Act as a Python performance optimization expert. Improve the following code to run faster while keeping the same output.

- Identify performance bottlenecks
- Use efficient techniques like list comprehensions or built-in functions
- Keep the logic unchanged
- Provide the optimized Python code
- Briefly explain the improvements

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))
```

Code:

→ Q ai coding

py X ass.py (File Saved • February 25, 2026 at 9:43 AM) ↔ ass.py fibon ▷ ⚙

s.py > ...

```
#Act as a Python performance optimization expert. Improve the following
'''• Identify performance bottlenecks
• Use efficient techniques like list comprehensions or built-in functions
• Keep the logic unchanged
• Provide the optimized Python code
• Briefly explain the improvements

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))'''
# Optimized code using list comprehension for better performance
nums = [i for i in range(1, 1000000)]
squares = [n**2 for n in nums] # Using list comprehension to create the list
print(len(squares))
# Improvements:
```

← → 🔍 ai coding

ass.py X ass.py (File Saved • February 25, 2026 at 9:43 AM) ↔ ass.py fibon ▷ 1

ass.py > ...

```
# 1. Replaced the for loop with a list comprehension, which is generally faster.
# 2. This approach reduces the overhead of multiple append calls and allows for easier modification of the list.
```

"""
This implementation uses a singly linked list to store contacts.
Efficient deletion but requires sequential access for searching.
"""

```
class Contact:  
    """Represents a single contact with name and phone number."""  
  
    def __init__(self, name, phone):  
        self.name = name  
        self.phone = phone  
  
    def __repr__(self):  
        return f"Contact(name='{self.name}', phone='{self.phone}')"  
  
    def __str__(self):  
        return f"Name: {self.name}, Phone: {self.phone}"  
  
class Node:
```

Ln 166, Col 17 Spaces: 4 UTF-8 CRLF { } Python base (3.13.5) (↻) G

← → 🔍 ai coding

ss.py X ass.py (File Saved • February 25, 2026 at 9:43 AM) ↔ ass.py fibon ▷ ⏺

ass.py > ...

```
class LinkedListContactManager:
    def add_contact(self, name, phone):
        # Check for duplicate names
        if self.search_contact(name) is not None:
            print(f"Error: Contact '{name}' already exists.")
            return False

        contact = Contact(name, phone)
        new_node = Node(contact)

        # Add to the beginning of the list
        new_node.next = self.head
        self.head = new_node
        self.count += 1

        print(f"✓ Contact '{name}' added successfully.")
        return True

    def search_contact(self, name):
        """Search for a contact by name.

        Time Complexity: O(n) - linear search through the linked list
        """
        pass
```

Ln 166, Col 17 Spaces: 4 UTF-8 CRLF { } Python base (3.13.5) (↻) Go

The screenshot shows a VS Code interface with the following details:

- Title Bar:** Shows "ass.py" and "ass.py (File Saved • February 25, 2026 at 9:43 AM) ↔ ass.py".
- Code Editor:** Displays Python code for a `LinkedListContactManager`. The code includes methods `display_all_contacts` and `get_contact_count`, both with detailed docstrings explaining time complexity and logic.
- Status Bar:** Shows "Ln 166, Col 17" and "Spaces: 4".
- Bottom Status Bar:** Shows "UTF-8", "CRLF", "Python", "base (3.13.5)", and "(i) C".

Output:

The terminal output shows the execution of the `ass.py` script:

```
\OneDrive\Documents\Desktop\ai coding'; & 'C:\Users\sneha\anaconda3\python.exe'
  'c:\Users\sneha\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '56548' --- 'c:\Users\sneha\OneDrive\Documents\Desktop\ai coding\ass.py'
999999
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding> ^C
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding>
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding> cd 'c:\Users\sneha'
```

Explanation:

->Reduced unnecessary loops — Replaced the manual loop with a list comprehension which is faster in Python.

-->Improved execution speed — List comprehensions are optimized internally, reducing overhead.

-->Simplified code — Fewer lines make the code cleaner and easier to maintain.

-->Lower memory overhead — Direct computation avoids extra intermediate steps.

-->Maintained same output — The logic (calculating squares and counting them) remains unchanged.

Task 6: Complexity Reduction. Use AI to simplify overly complex logic.

Prompt:

Act as a Python code simplification expert. Simplify the following code by reducing unnecessary complexity while keeping the same logic and output.

- Replace deeply nested if-else statements with cleaner logic (e.g., elif or mapping)
- Improve readability and structure
- Keep functionality unchanged
- Provide the simplified Python code
- Briefly explain the improvements

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"
```

Code:

← → 🔍 ai coding ⚙️ 1 08

s.py X ass.py (File Saved • February 25, 2026 at 9:43 AM) ↔ ass.py fibon ▷ ⓘ

ss.py > ...

```
#Act as a Python code simplification expert. Simplify the following code.
'''• Replace deeply nested if-else statements with cleaner logic (e.g.,
• Improve readability and structure
• Keep functionality unchanged
• Provide the simplified Python code
• Briefly explain the improvements

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"'''
```

Ln 39, Col 30 Spaces: 4 UTF-8 CRLF { } Python ⚙️ base (3.13.5) ⓘ Go Li

The screenshot shows a code editor window with the following details:

- Title Bar:** Shows the file name "ass.py" and a status message "ass.py (File Saved • February 25, 2026 at 9:43 AM) ↔ ass.py".
- Code Area:** Displays the following Python code:

```
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"
#example usage
print(grade(95)) # Output: A
print(grade(85)) # Output: B
print(grade(75)) # Output: C
print(grade(65)) # Output: D
# Improvements:
# 1. Replaced deeply nested if-else statements with elif for better readability
# 2. The logic remains unchanged, ensuring the same output for the same inputs
```
- Status Bar:** Shows the current line and column ("Ln 39, Col 30"), character encoding ("UTF-8"), line endings ("CRLF"), and the Python version ("base (3.13.5)").

Output:

The terminal window displays the following output:

```
her' '56512' '--' 'c:\Users\sneha\OneDrive\Documents\Desktop\ai coding\ass.py
A
B
C
D
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding>
```

Explanation:

->Removed deep nesting — Replaced multiple nested if-else blocks with a simpler structure to make the code easier to follow.

->Used elif statements — This reduces unnecessary checks once a condition is satisfied.

- >Improved readability — The logic becomes clearer and easier to understand at a glance.
- >Simplified maintenance — Future changes to grading rules can be made more easily.
- >Kept functionality same — The grading output remains exactly the same for all scores.