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

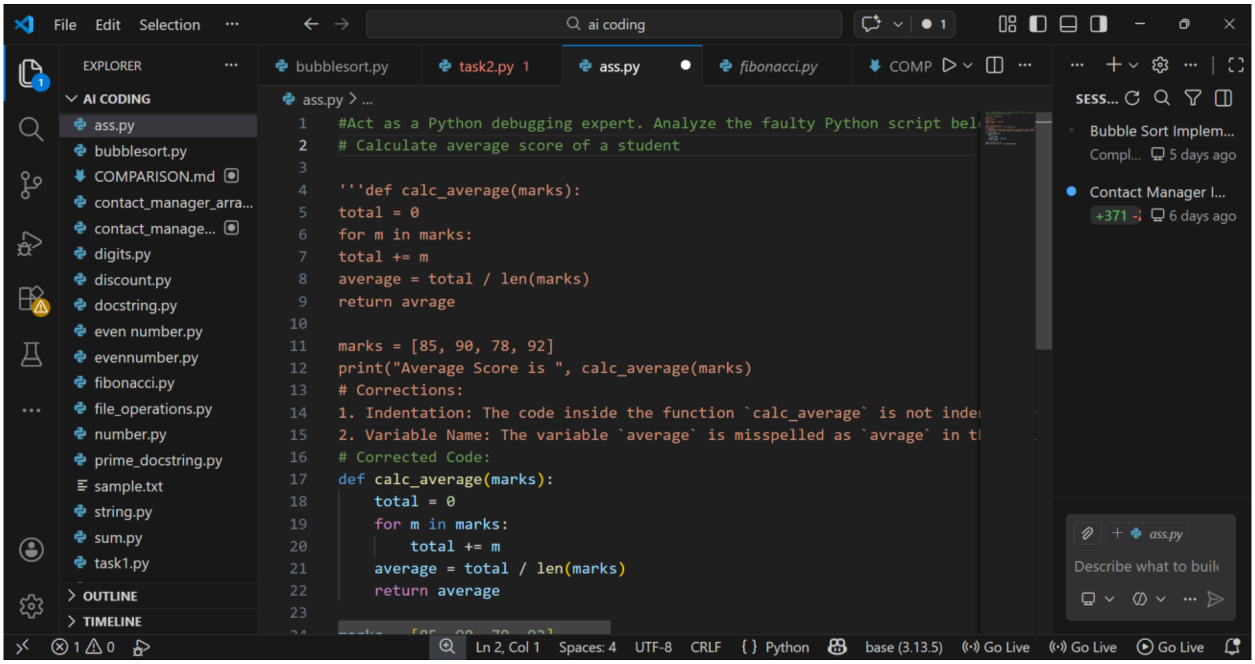
• Explain the corrections briefly

• Show the corrected runnable code

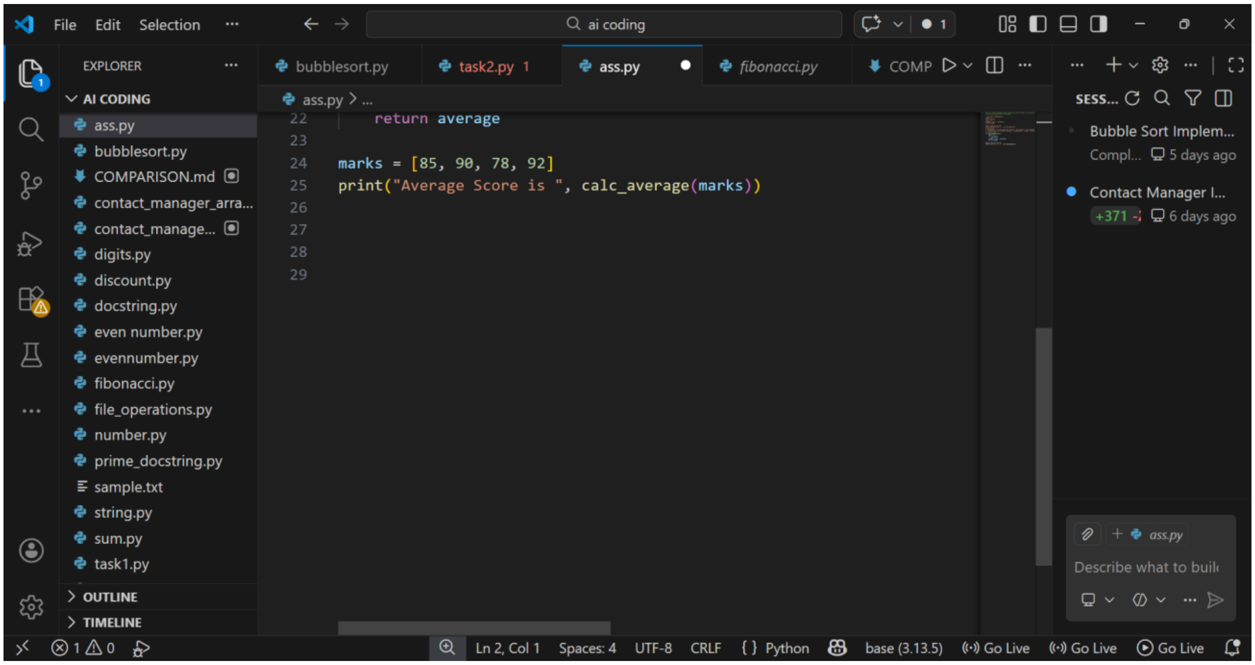
Faulty code:

```
Calculate average score of a student

def calc_average(marks):

total = 0

for m in marks:

total += m

average = total / len(marks)

return avrage

marks = [85, 90, 78, 92]

print("Average Score is ", calc_average(marks)
```

Code:

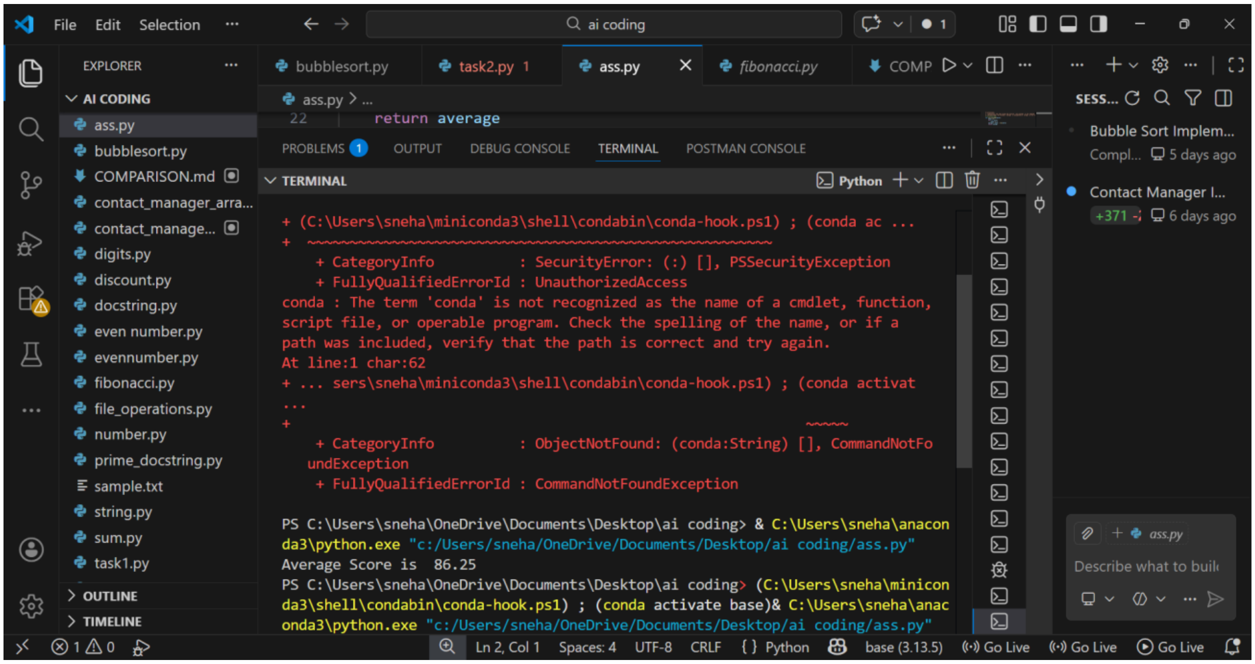


```
#Act as a Python debugging expert. Analyze the faulty Python script bel
# Calculate average score of a student

'''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 inde
2. Variable Name: The variable `average` is misspelled as `avrage` in tl
# Corrected Code:
def calc_average(marks):
    total = 0
    for m in marks:
        total += m
    average = total / len(marks)
    return average

```



```
    return average

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

Output:



```python
    return average
```

```
+ (C:\Users\sneha\miniconda3\shell\condabin\conda-hook.ps1) ; (conda ac ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + 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) [], CommandNotFo
   undException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding> & C:\Users\sneha\anacon
da3\python.exe "c:/Users/sneha/OneDrive/Documents/Desktop/ai coding/ass.py"
Average Score is  86.25
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding> (C:\Users\sneha\minicon
da3\shell\condabin\conda-hook.ps1) ; (conda activate base)& C:\Users\sneha\anac
onda3\python.exe "c:/Users/sneha/OneDrive/Documents/Desktop/ai coding/ass.py"
```

Explanation:

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

-->Fixed typo — Changed avrage 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

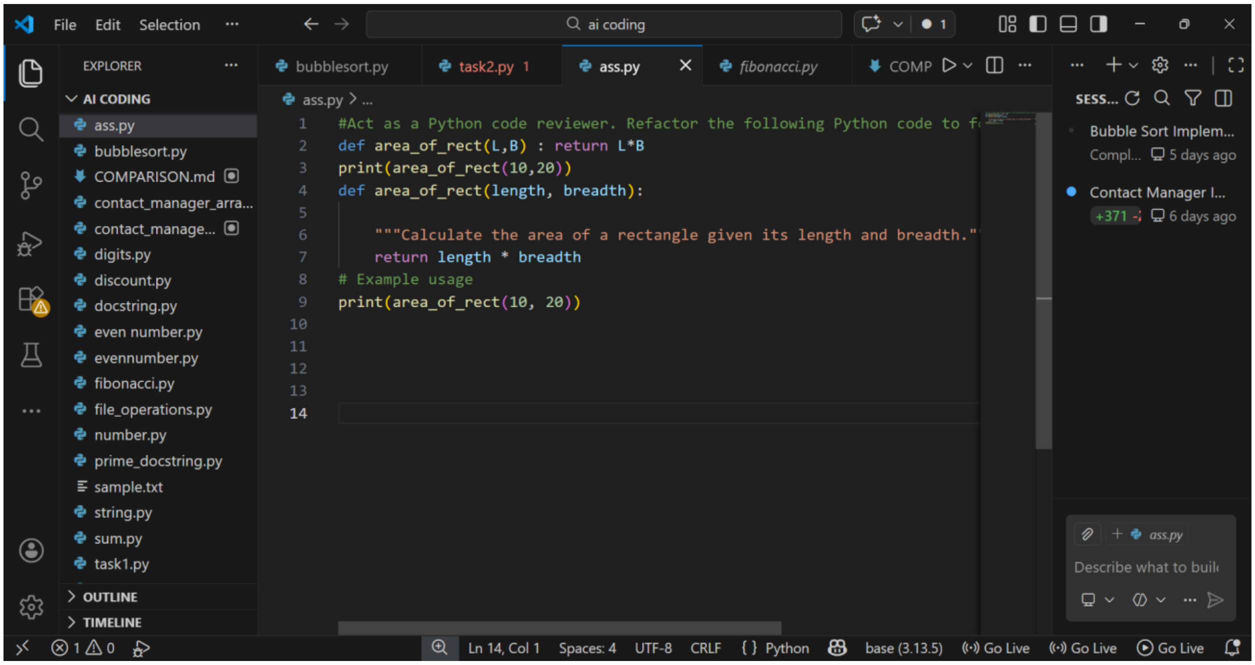
Act as a Python code reviewer. Refactor the following Python code to follow PEP 8 style guidelines.

Code:

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

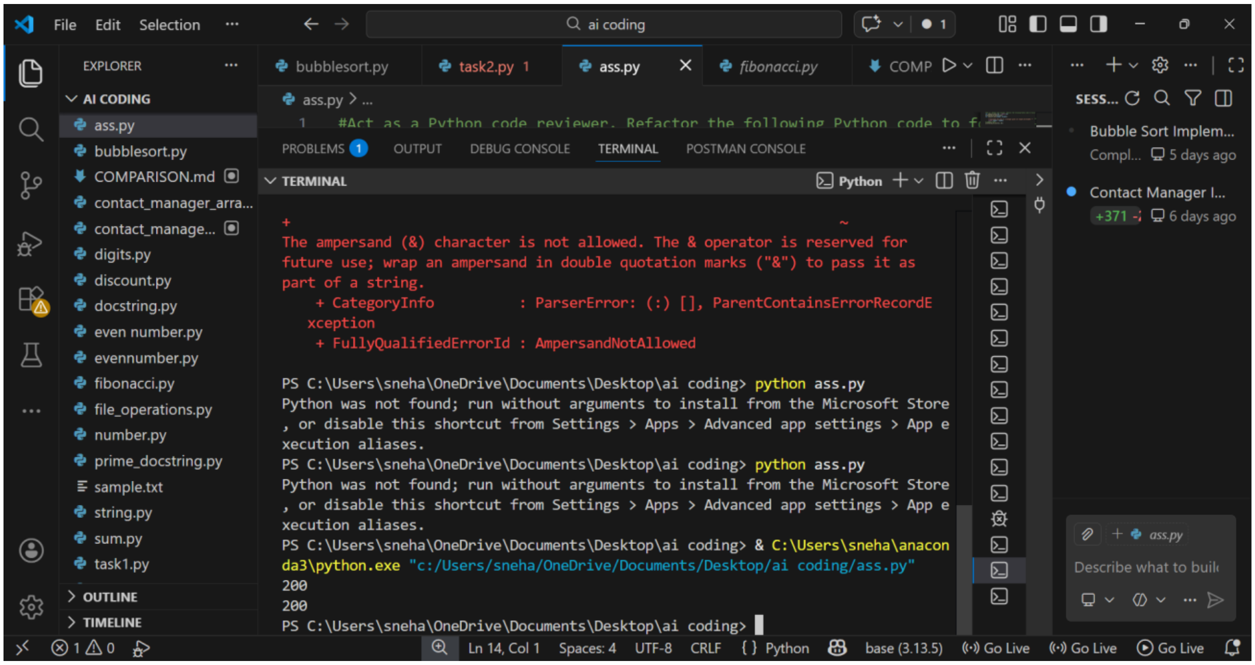
print(area_of_rect(10,20))

Code:



```
#Act as a Python code reviewer. Refactor the following Python code to f
def area_of_rect(L,B) : return L*B
print(area_of_rect(10,20))
def area_of_rect(length, breadth):

    """Calculate the area of a rectangle given its length and breadth."
    return length * breadth
# Example usage
print(area_of_rect(10, 20))
```

Output:



```
+                                                                    ~
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: (:) [], ParentContainsErrorRecordE
   xception
    + 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 e
xecution 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 e
xecution aliases.
PS C:\Users\sneha\OneDrive\Documents\Desktop\ai coding> & C:\Users\sneha\anacon
da3\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:

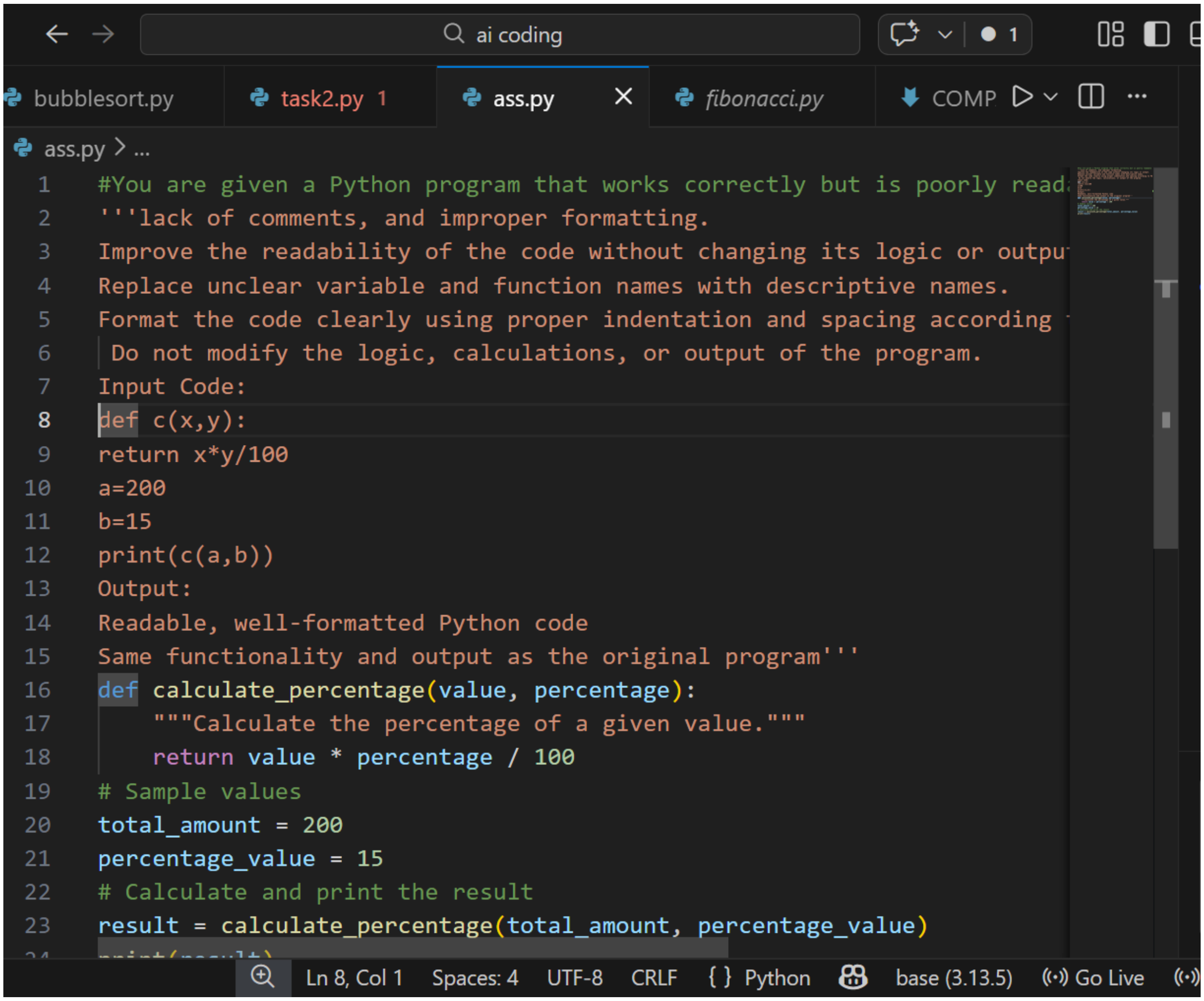
```
def c(x,y):

return x*y/100

a=200

b=15

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

Output:

Readable, well-formatted Python code

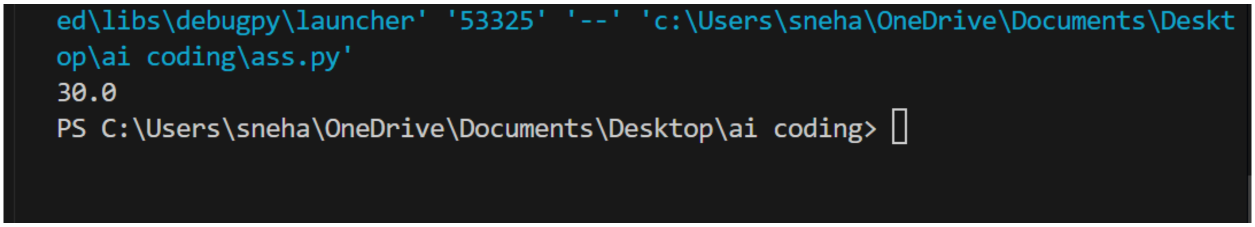
Same functionality and output as the original program.

Code:



```
#You are given a Python program that works correctly but is poorly read
'''lack of comments, and improper formatting.
Improve the readability of the code without changing its logic or outpu
Replace unclear variable and function names with descriptive names.
Format the code clearly using proper indentation and spacing according
 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'''
def calculate_percentage(value, percentage):
    """Calculate the percentage of a given value."""
    return value * percentage / 100
# Sample values
total_amount = 200
percentage_value = 15
# Calculate and print the result
result = calculate_percentage(total_amount, percentage_value)
print(result)
```

Output:



```
ed\libs\debugpy\launcher' '53325' '--' 'c:\Users\sneha\OneDrive\Documents\Deskt
op\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:

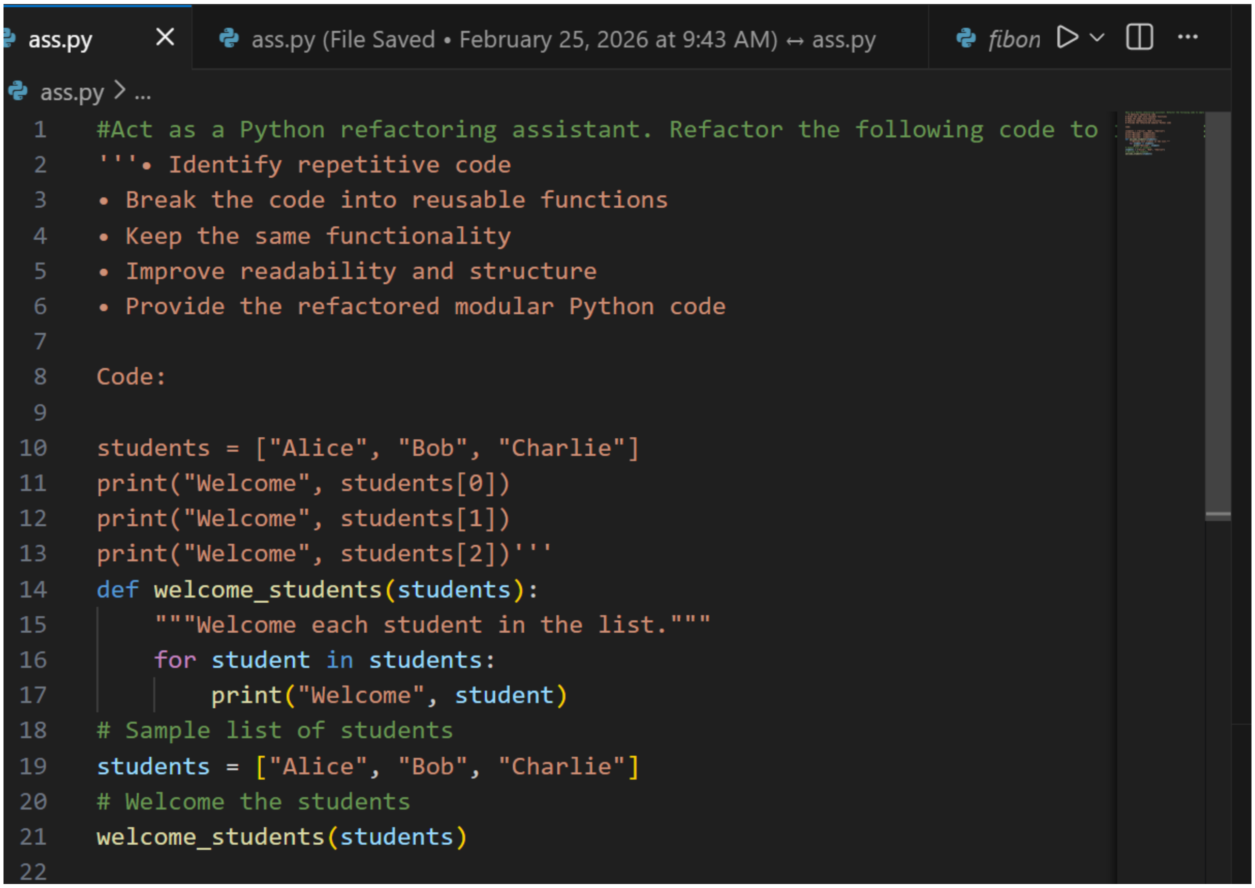
Act as a Python refactoring assistant. Refactor the following code to improve maintainability by removing repetition and creating reusable functions.

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

Code:

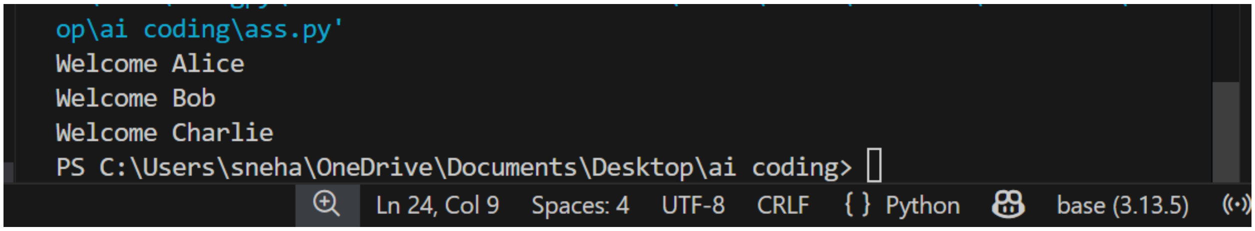
```
students = ["Alice", "Bob", "Charlie"]

print("Welcome", students[0])

print("Welcome", students[1])

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

Code:



```
#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:



```
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.

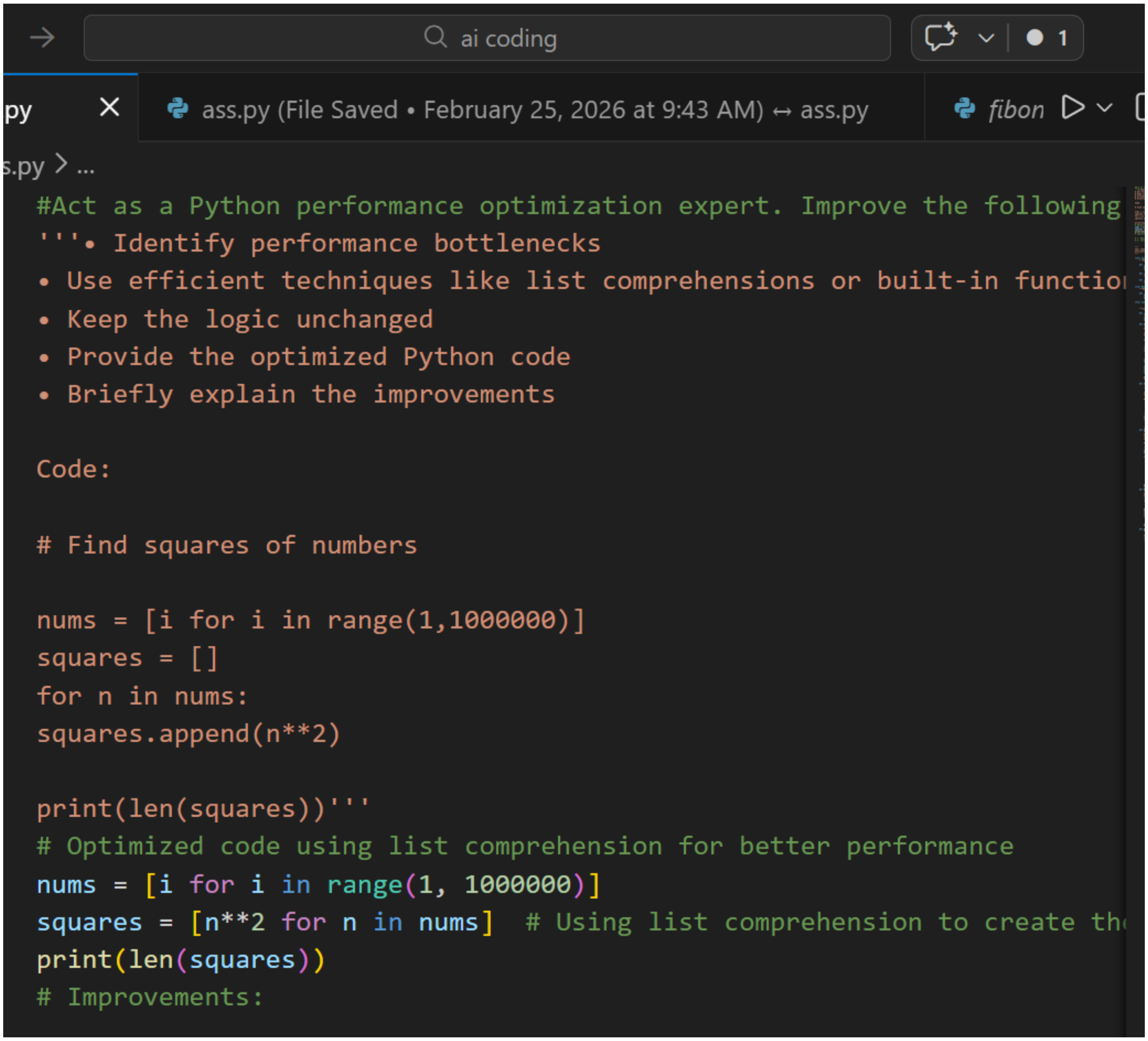
- Identify performance bottlenecks
- Use efficient techniques like list comprehensions or built-in functions
- Keep the logic unchanged
- Provide the optimized Python code
- Briefly explain the improvements

Code:

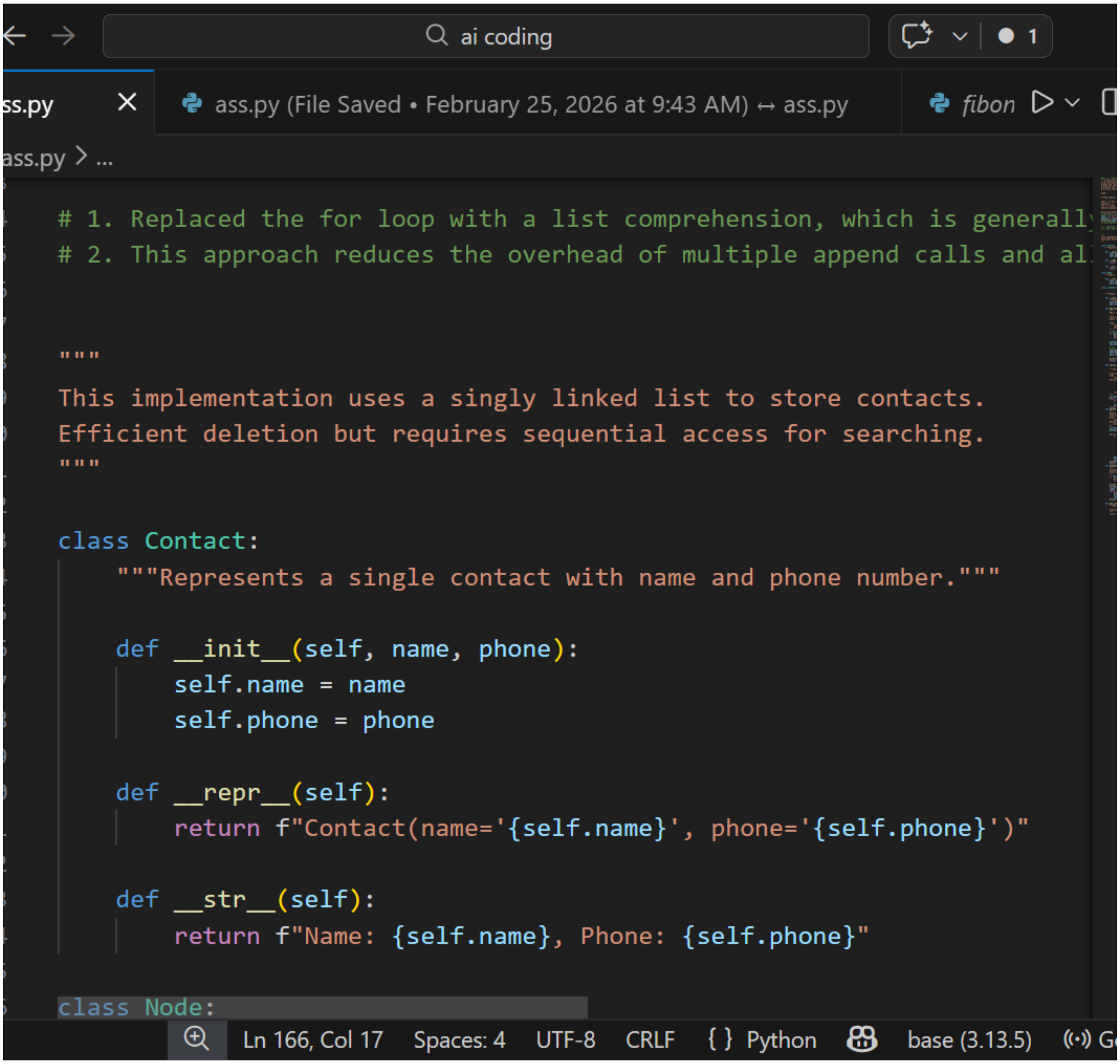
Find squares of numbers

```
nums = [i for i in range(1,1000000)]

squares = []

for n in nums:

squares.append(n**2)

print(len(squares))
```

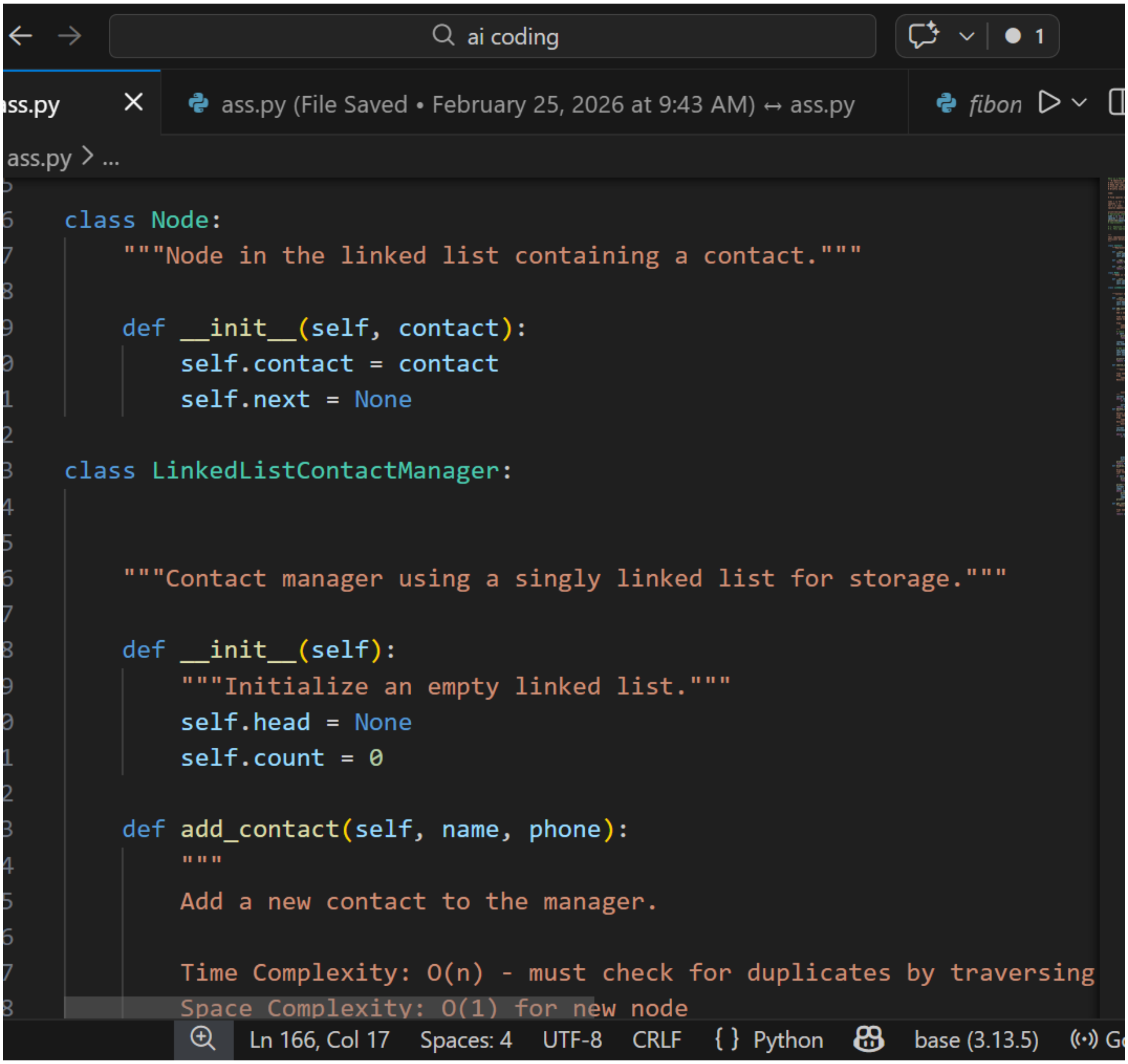
Code:



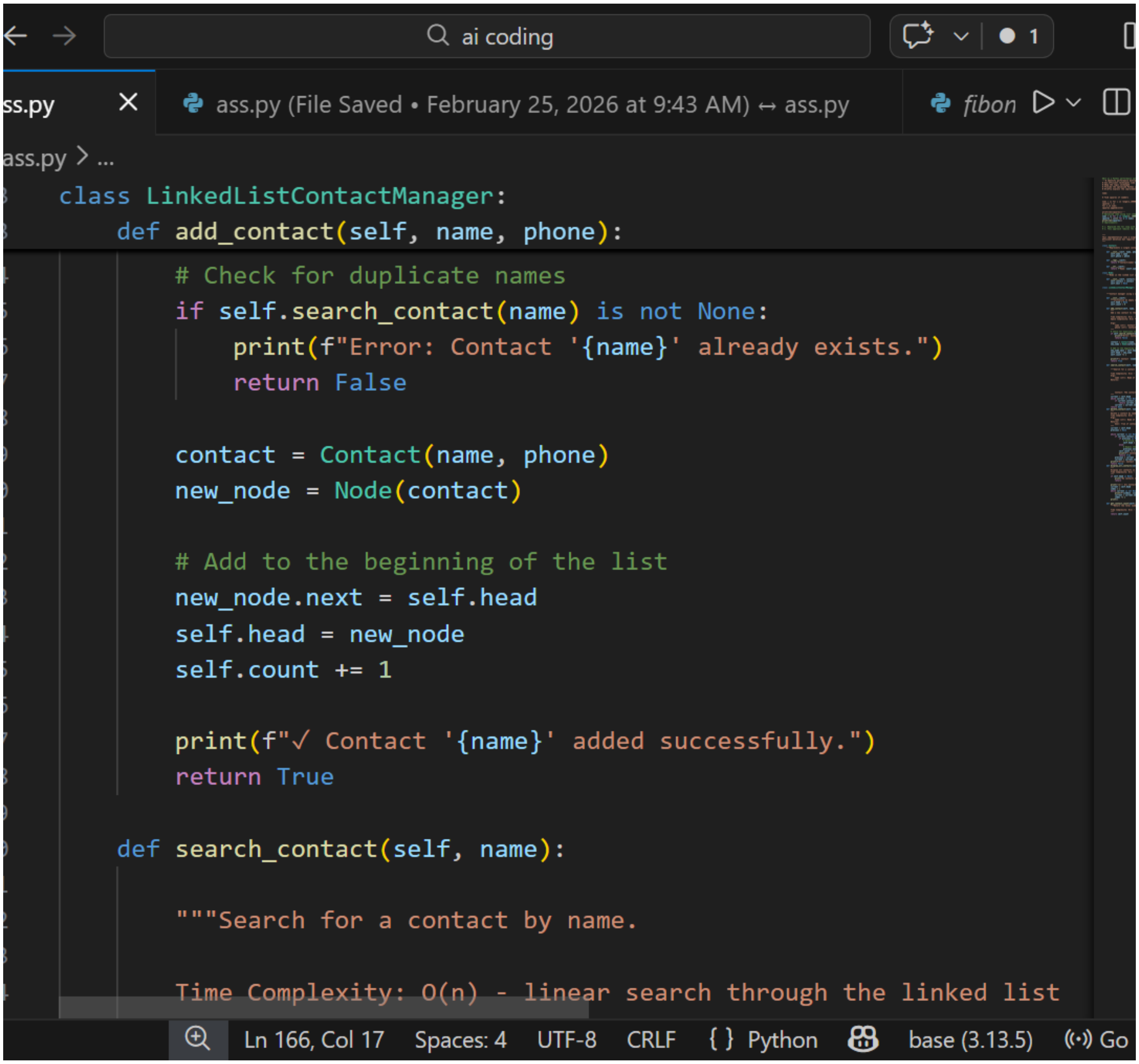
```
#Act as a Python performance optimization expert. Improve the following
'''• Identify performance bottlenecks
• Use efficient techniques like list comprehensions or built-in functio
• Keep the logic unchanged
• Provide the optimized Python code
• Briefly explain the improvements

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 th
print(len(squares))
# Improvements:
```



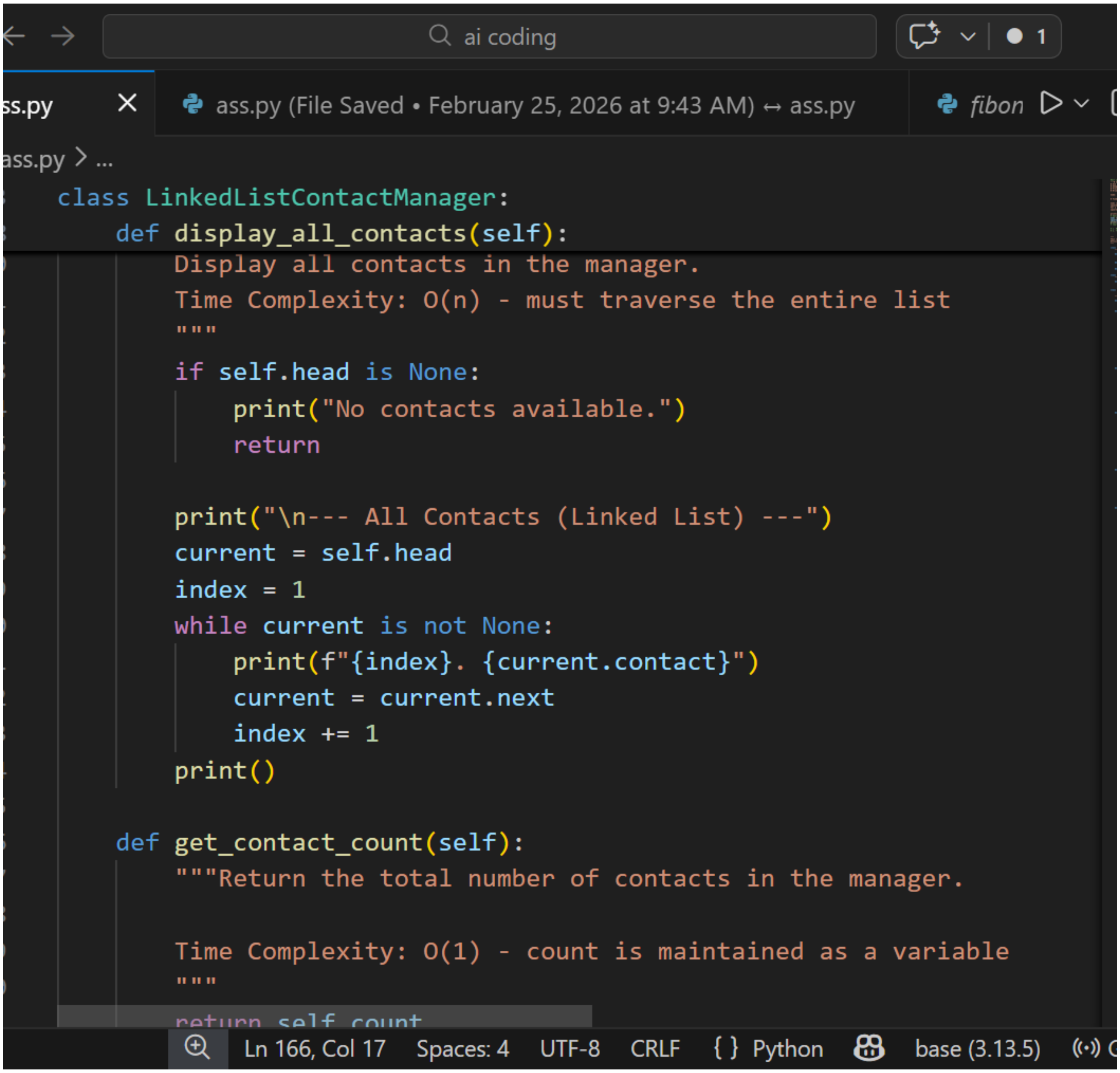
```python
# 1. Replaced the for loop with a list comprehension, which is generally
# 2. This approach reduces the overhead of multiple append calls and al


"""
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:
```



```python
class Node:
    """Node in the linked list containing a contact."""

    def __init__(self, contact):
        self.contact = contact
        self.next = None

class LinkedListContactManager:


    """Contact manager using a singly linked list for storage."""

    def __init__(self):
        """Initialize an empty linked list."""
        self.head = None
        self.count = 0

    def add_contact(self, name, phone):
        """
        Add a new contact to the manager.

        Time Complexity: O(n) - must check for duplicates by traversing
        Space Complexity: O(1) for new node
```

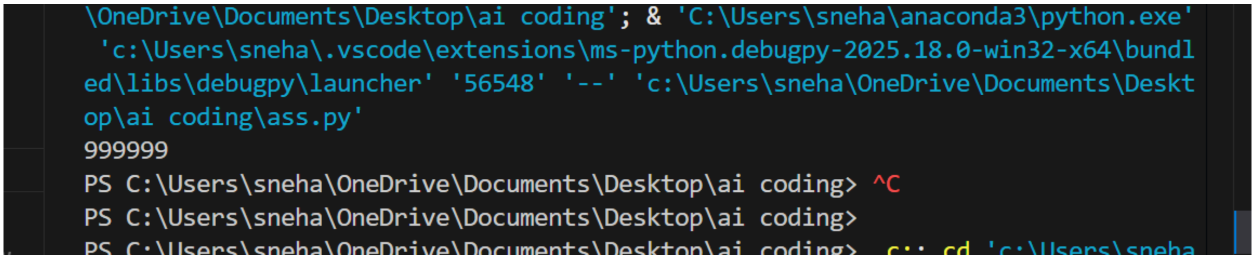


```python
class LinkedListContactManager:
    def add_contact(self, name, phone):
        # Check for duplicate names
        if self.search_contact(name) is not None:
            print(f"Error: Contact '{name}' already exists.")
            return False

        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
```



```python
class LinkedListContactManager:
    def display_all_contacts(self):
        Display all contacts in the manager.
        Time Complexity: O(n) - must traverse the entire list
        """
        if self.head is None:
            print("No contacts available.")
            return

        print("\n--- All Contacts (Linked List) ---")
        current = self.head
        index = 1
        while current is not None:
            print(f"{index}. {current.contact}")
            current = current.next
            index += 1
        print()

    def get_contact_count(self):
        """Return the total number of contacts in the manager.

        Time Complexity: O(1) - count is maintained as a variable
        """
        return self.count
```

Output:



```
\OneDrive\Documents\Desktop\ai coding'; & 'C:\Users\sneha\anaconda3\python.exe'
 'c:\Users\sneha\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundl
ed\libs\debugpy\launcher' '56548' '--' 'c:\Users\sneha\OneDrive\Documents\Deskt
op\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>  c:; 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

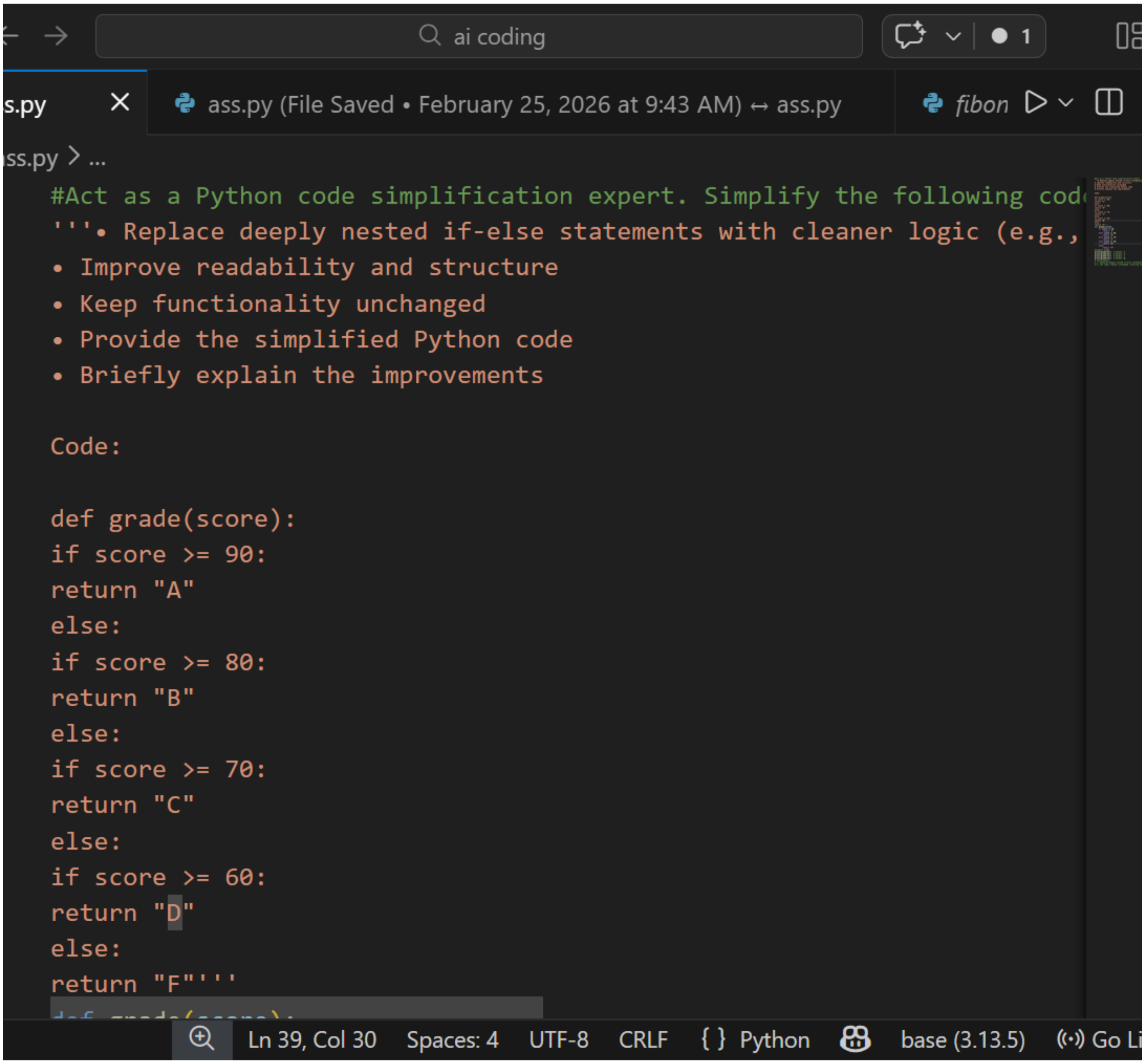
- Briefly explain the improvements

Code:

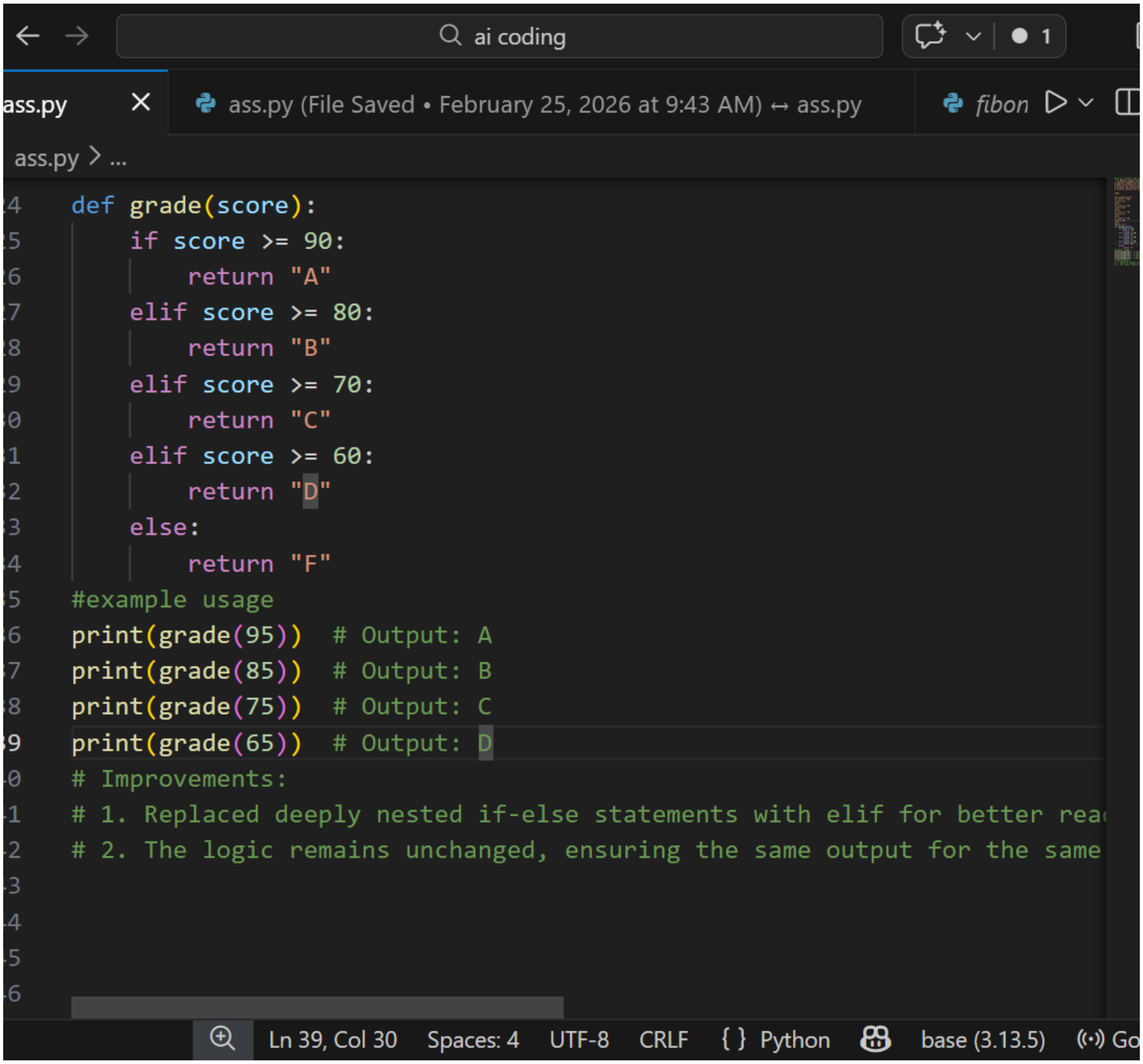
```
def grade(score):

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:

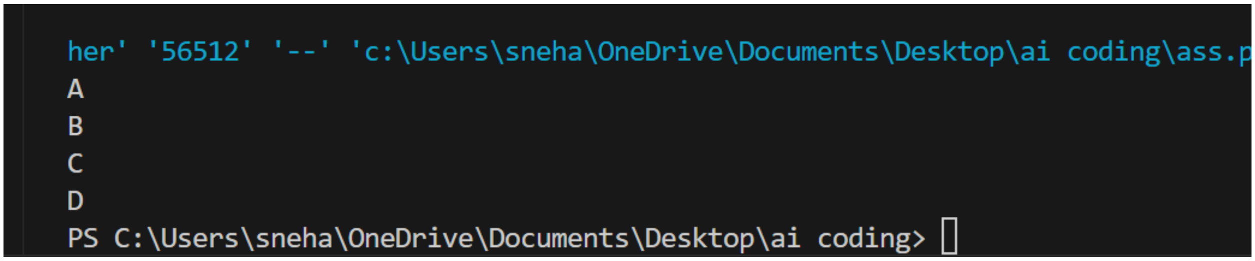


```
#Act as a Python code simplification expert. Simplify the following code
'''• Replace deeply nested if-else statements with cleaner logic (e.g.,
• Improve readability and structure
• Keep functionality unchanged
• Provide the simplified Python code
• Briefly explain the improvements

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"'''
def grade(score):
```



```
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 rea
# 2. The logic remains unchanged, ensuring the same output for the same
```

Output:



```
her' '56512' '--' 'c:\Users\sneha\OneDrive\Documents\Desktop\ai coding\ass.p
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.