ASSIGNMENT-6.2

Name : P.Reshmitha Reddy

Rollno : 2403A510A7

Batch : 05

Course : AI Assisted Coding

Task Description#1 (Classes)

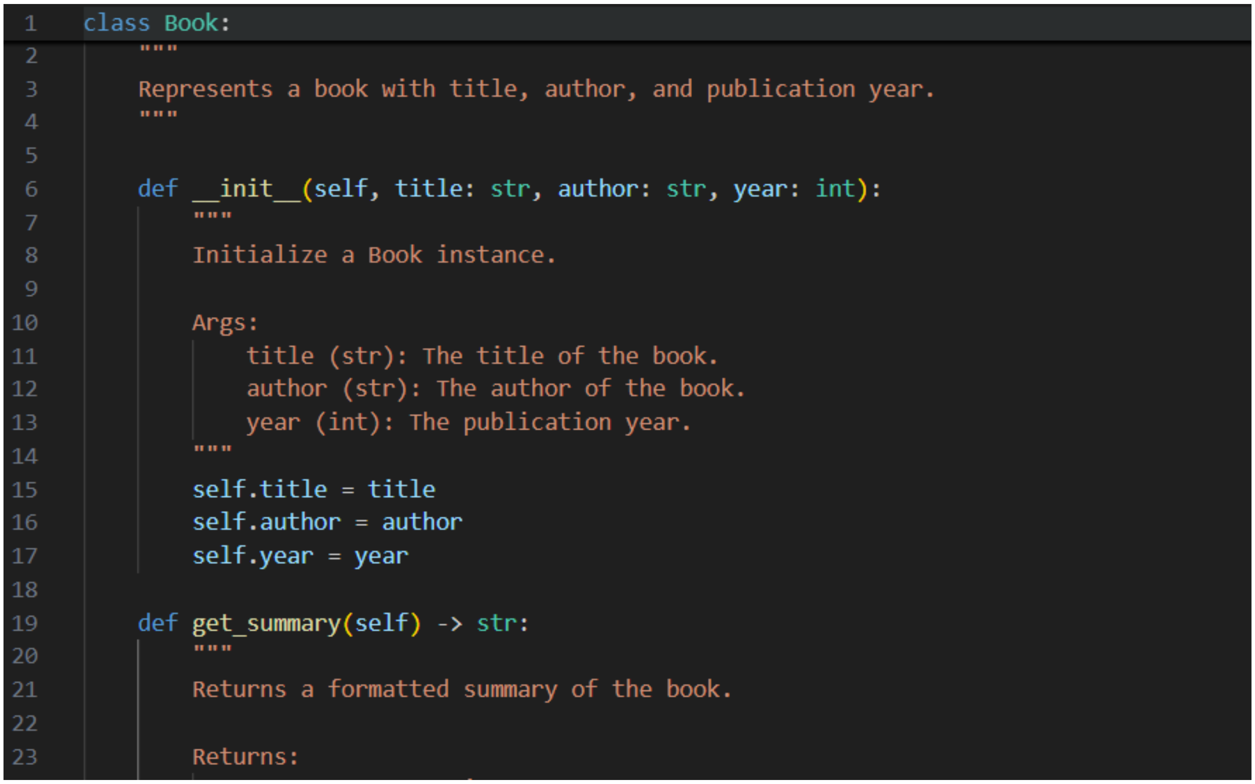
- Prompt AI to generate a Book class with attributes: title, author, year. Add a method get_summary() to return a formatted string.

Expected Output#1

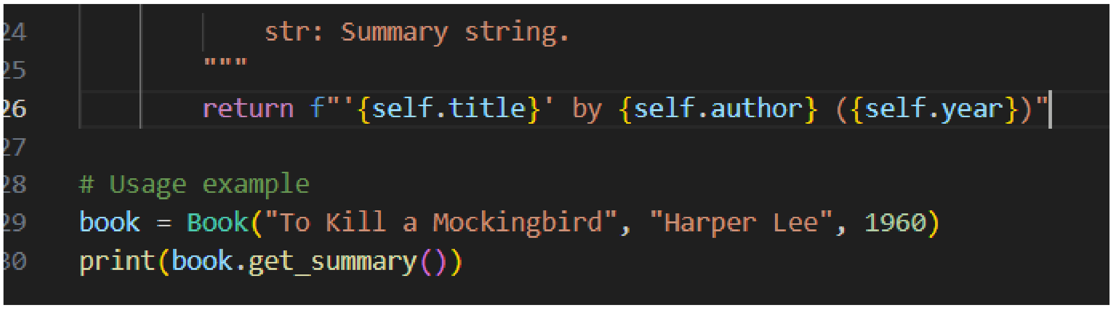
- Functional class with constructor and summary method. Students should analyze code readability and style

Prompt :# generate a Book class with attributes: title, author, year.

Code :

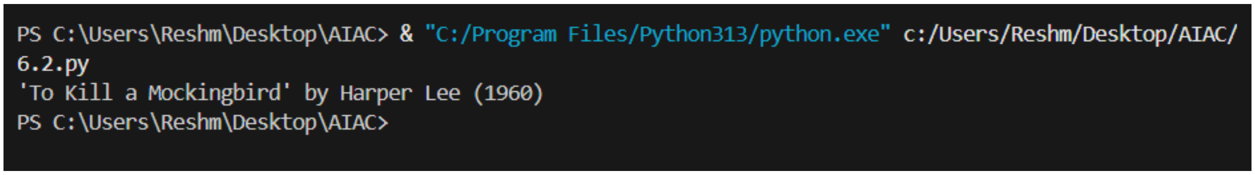


```python
class Book:
    """
    Represents a book with title, author, and publication year.
    """

    def __init__(self, title: str, author: str, year: int):
        """
        Initialize a Book instance.

        Args:
            title (str): The title of the book.
            author (str): The author of the book.
            year (int): The publication year.
        """
        self.title = title
        self.author = author
        self.year = year

    def get_summary(self) -> str:
        """
        Returns a formatted summary of the book.

        Returns:
```



```python
            str: Summary string.
        """
        return f"'{self.title}' by {self.author} ({self.year})"

# Usage example
book = Book("To Kill a Mockingbird", "Harper Lee", 1960)
print(book.get_summary())
```

Output :



```
PS C:\Users\Reshm\Desktop\AIAC> & "C:/Program Files/Python313/python.exe" c:/Users/Reshm/Desktop/AIAC/
6.2.py
'To Kill a Mockingbird' by Harper Lee (1960)
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

The Book class is well-structured, with clear attribute names (title, author, year) and type hints for readability.The constructor (__init__) properly initializes all attributes, following good coding style and documentation practices.The get_summary method returns a concise, formatted string summarizing the book’s details.The usage example demonstrates how to create a Book instance and print its summary.The code is readable, maintainable, and follows Python conventions for class design and docstrings.

Task Description#2 (Loops)

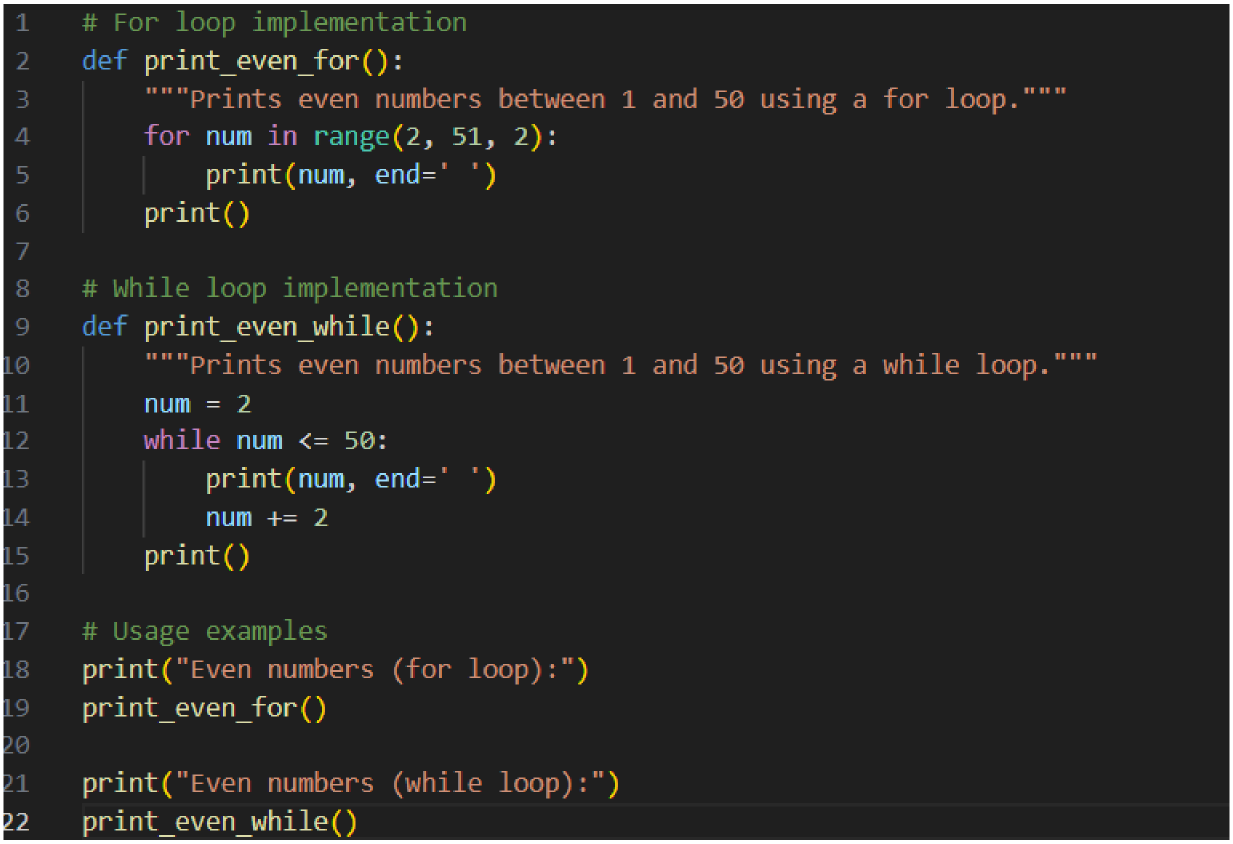
- Ask AI to generate a function that prints the even numbers between 1 and 50 using a
for loop. Then regenerate using a while loop.

Expected Output#2

- Correct loop-based implementation.

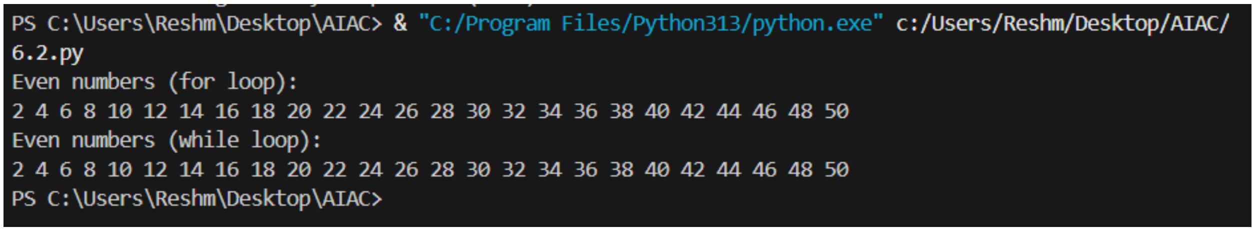
Prompt :# Generate Python functions to print even numbers from 1 to 50 using both a for loop and a while loop.

Code :



```python
# For loop implementation
def print_even_for():
    """Prints even numbers between 1 and 50 using a for loop."""
    for num in range(2, 51, 2):
        print(num, end=' ')
    print()

# While loop implementation
def print_even_while():
    """Prints even numbers between 1 and 50 using a while loop."""
    num = 2
    while num <= 50:
        print(num, end=' ')
        num += 2
    print()

# Usage examples
print("Even numbers (for loop):")
print_even_for()

print("Even numbers (while loop):")
print_even_while()
```

Output :



```
PS C:\Users\Reshm\Desktop\AIAC> & "C:/Program Files/Python313/python.exe" c:/Users/Reshm/Desktop/AIAC/
6.2.py
Even numbers (for loop):
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50
Even numbers (while loop):
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

The file contains two functions: print_even_for() uses a for loop, and print_even_while() uses a while loop to print even numbers from 2 to 50.Both implementations correctly use step size 2 to iterate through even numbers, ensuring efficiency and clarity.The output is printed in a single line for each function, separated by spaces.The usage examples clearly demonstrate both approaches, making the code easy to understand and compare.The code is readable, concise, and follows good Python practices for loop constructs and function documentation.

Task Description#3 (Conditional Statements)

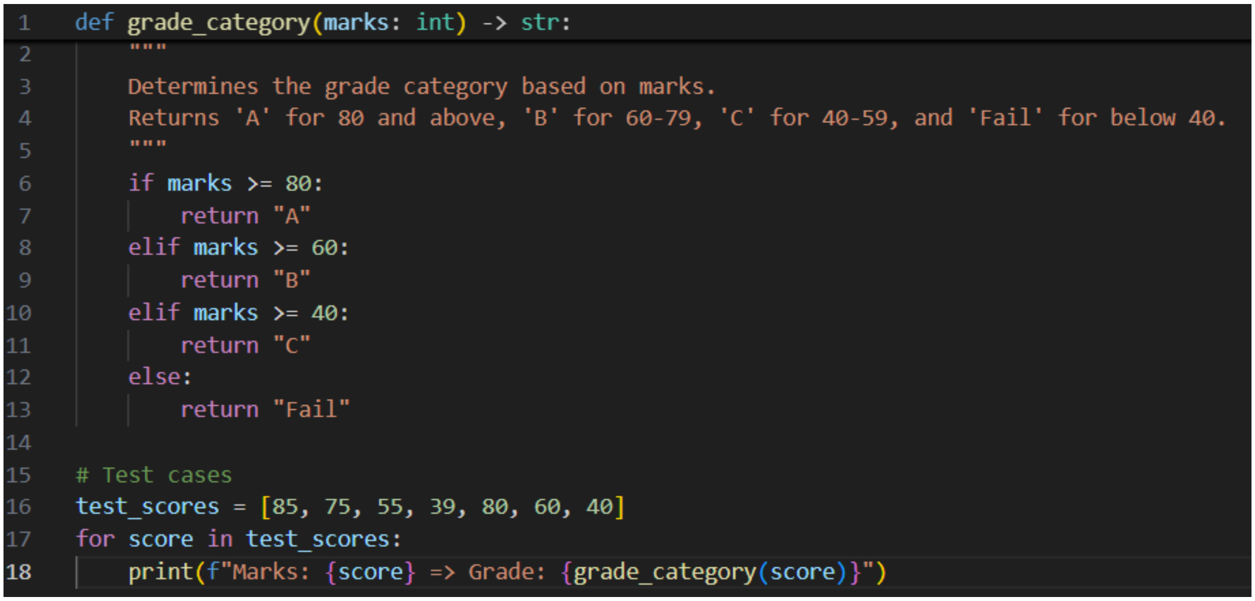
- Ask AI to write a function that determines the grade category (A, B, C, Fail) based on
marks using if-elif-else. Then test it with sample inputs.

Expected Output#3

- Function must handle score boundaries and include appropriate logic. Students
analyze boundary cases.

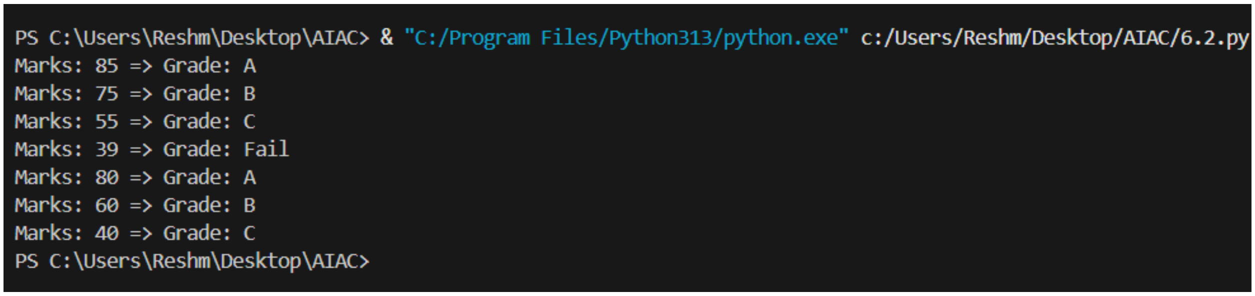
Prompt :# write a function that determines the grade category (A, B, C, Fail) based on
marks using if-elif-else

Code :



```python
def grade_category(marks: int) -> str:
    """
    Determines the grade category based on marks.
    Returns 'A' for 80 and above, 'B' for 60-79, 'C' for 40-59, and 'Fail' for below 40.
    """
    if marks >= 80:
        return "A"
    elif marks >= 60:
        return "B"
    elif marks >= 40:
        return "C"
    else:
        return "Fail"

# Test cases
test_scores = [85, 75, 55, 39, 80, 60, 40]
for score in test_scores:
    print(f"Marks: {score} => Grade: {grade_category(score)}")
```

Output :



```
PS C:\Users\Reshm\Desktop\AIAC> & "C:/Program Files/Python313/python.exe" c:/Users/Reshm/Desktop/AIAC/6.2.py
Marks: 85 => Grade: A
Marks: 75 => Grade: B
Marks: 55 => Grade: C
Marks: 39 => Grade: Fail
Marks: 80 => Grade: A
Marks: 60 => Grade: B
Marks: 40 => Grade: C
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

The function grade_category uses if-elif-else statements to categorize marks into grades: 'A' (≥80), 'B' (60–79), 'C' (40–59), and 'Fail' (<40).

The logic correctly handles score boundaries, ensuring that 80, 60, and 40 are included in their respective categories.

The test cases cover a range of scores, including boundary values, demonstrating that the function works as expected.

The code is clear, readable, and follows good Python practices for conditional statements and function documentation.

Task Description#4 (For and While loops)

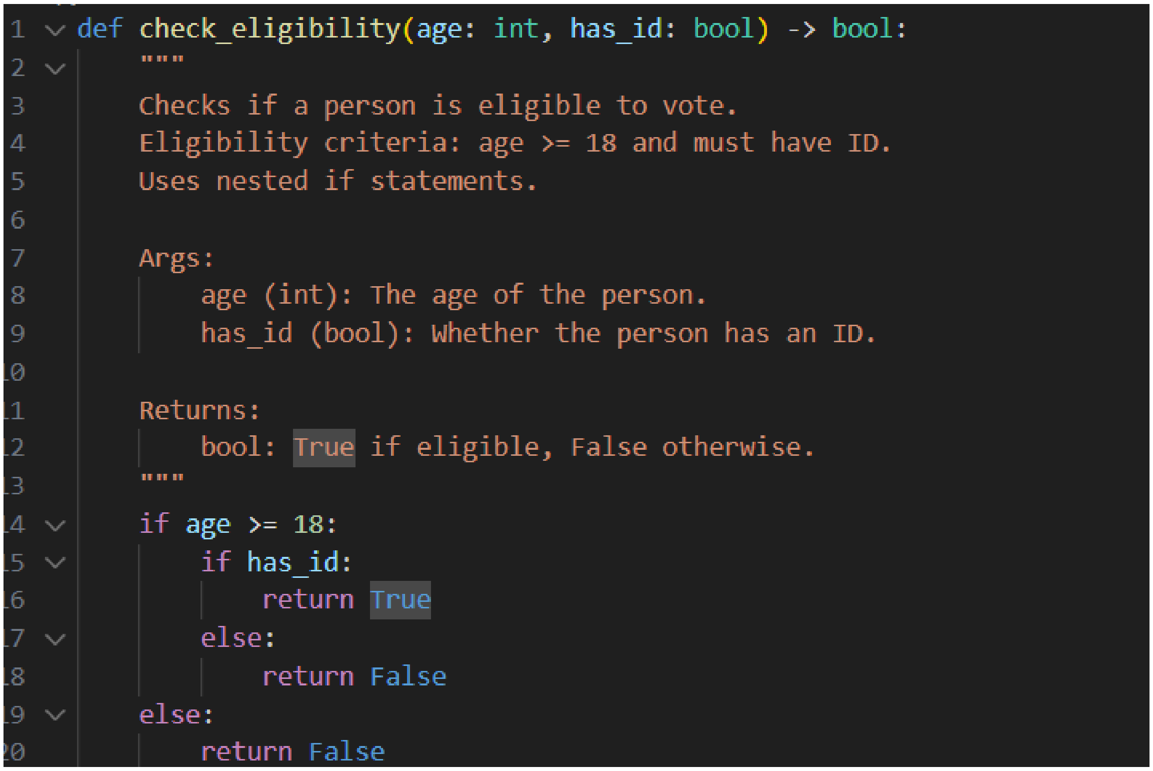
- Generate a function check_eligibility(age, has_id) that checks if a person is eligible to
vote (age ≥ 18 and must have ID). Use nested ifs.

Expected Output#4

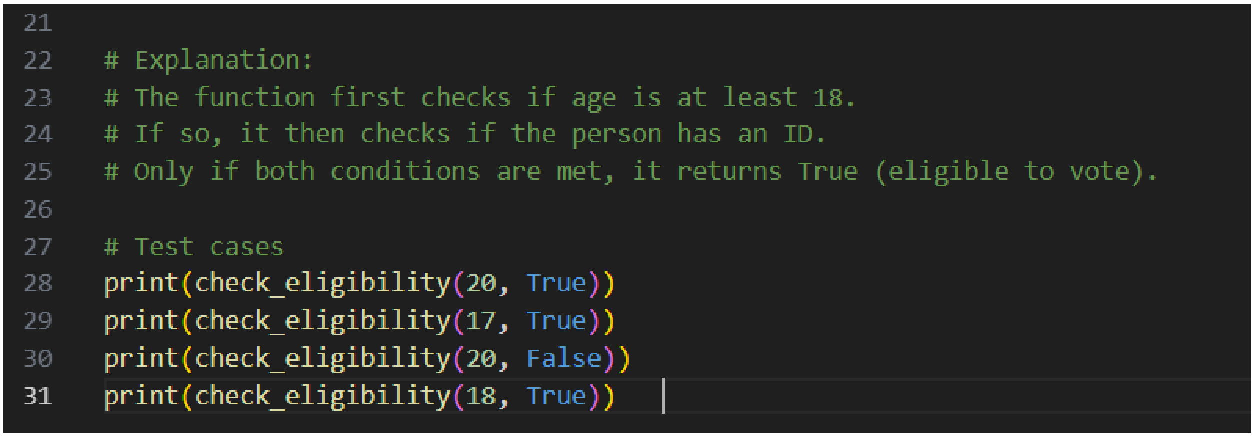
- Python code with explanation.

Prompt :# Generate a Python function with nested ifs to check voting eligibility (age ≥ 18 and has ID)

Code :

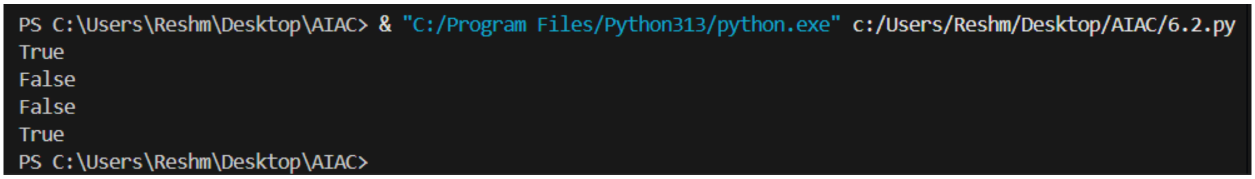


```python
def check_eligibility(age: int, has_id: bool) -> bool:
    """
    Checks if a person is eligible to vote.
    Eligibility criteria: age >= 18 and must have ID.
    Uses nested if statements.

    Args:
        age (int): The age of the person.
        has_id (bool): Whether the person has an ID.

    Returns:
        bool: True if eligible, False otherwise.
    """
    if age >= 18:
        if has_id:
            return True
        else:
            return False
    else:
        return False
```



```python

# Explanation:
# The function first checks if age is at least 18.
# If so, it then checks if the person has an ID.
# Only if both conditions are met, it returns True (eligible to vote).

# Test cases
print(check_eligibility(20, True))
print(check_eligibility(17, True))
print(check_eligibility(20, False))
print(check_eligibility(18, True))
```

Output :



```
PS C:\Users\Reshm\Desktop\AIAC> & "C:/Program Files/Python313/python.exe" c:/Users/Reshm/Desktop/AIAC/6.2.py
True
False
False
True
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

The function check_eligibility correctly uses nested if statements to check voting eligibility based on age and possession of an ID.The eligibility criteria are clear: the person must be at least 18 years old and must have an ID.The function returns True only if both conditions are satisfied; otherwise, it returns False.The code is readable, well-documented, and includes test cases that demonstrate all possible scenarios (eligible, underage, missing ID).The logic is robust and easy to follow, making it suitable for real-world eligibility checks.

Task Description#5 (Class)

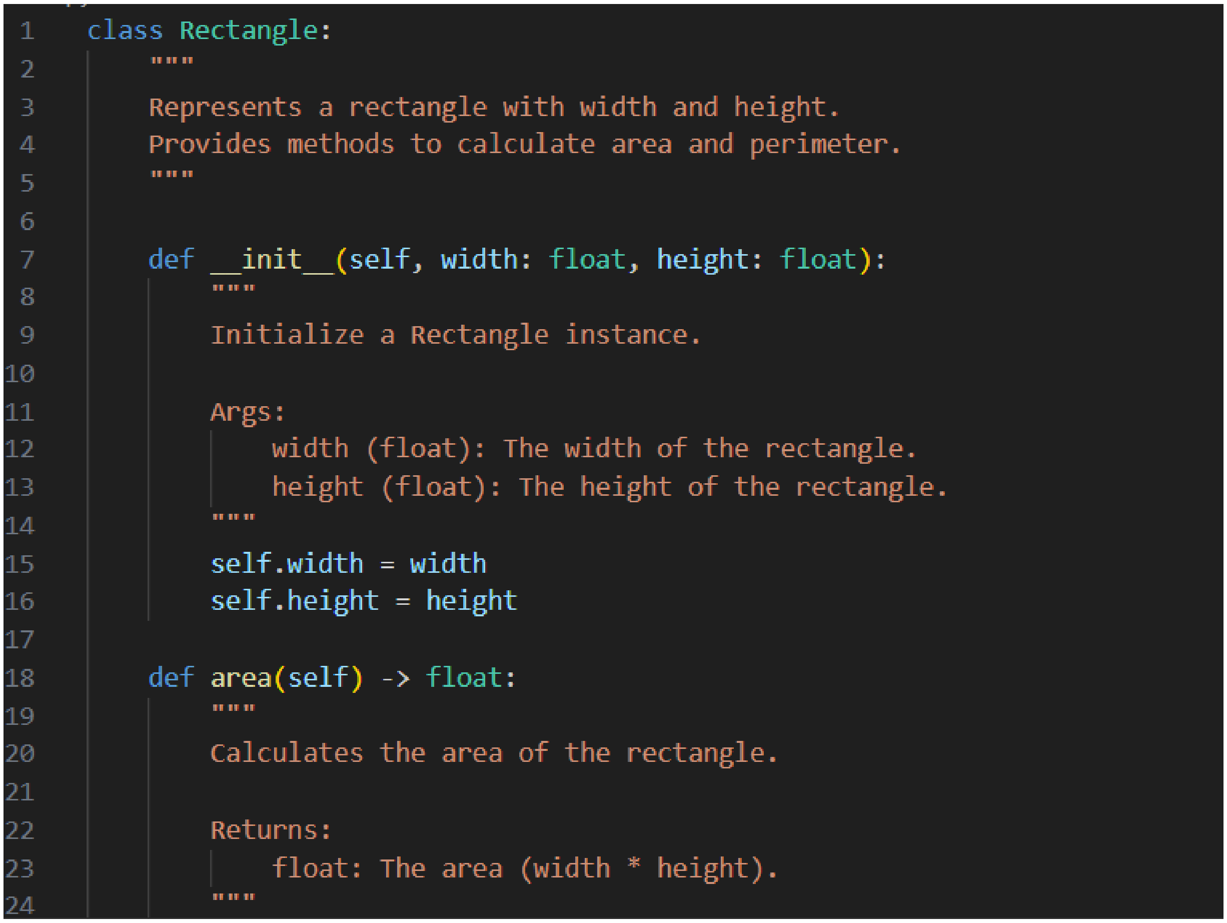
- Ask AI to create a Rectangle class with methods to calculate area() and perimeter().

Expected Output#5

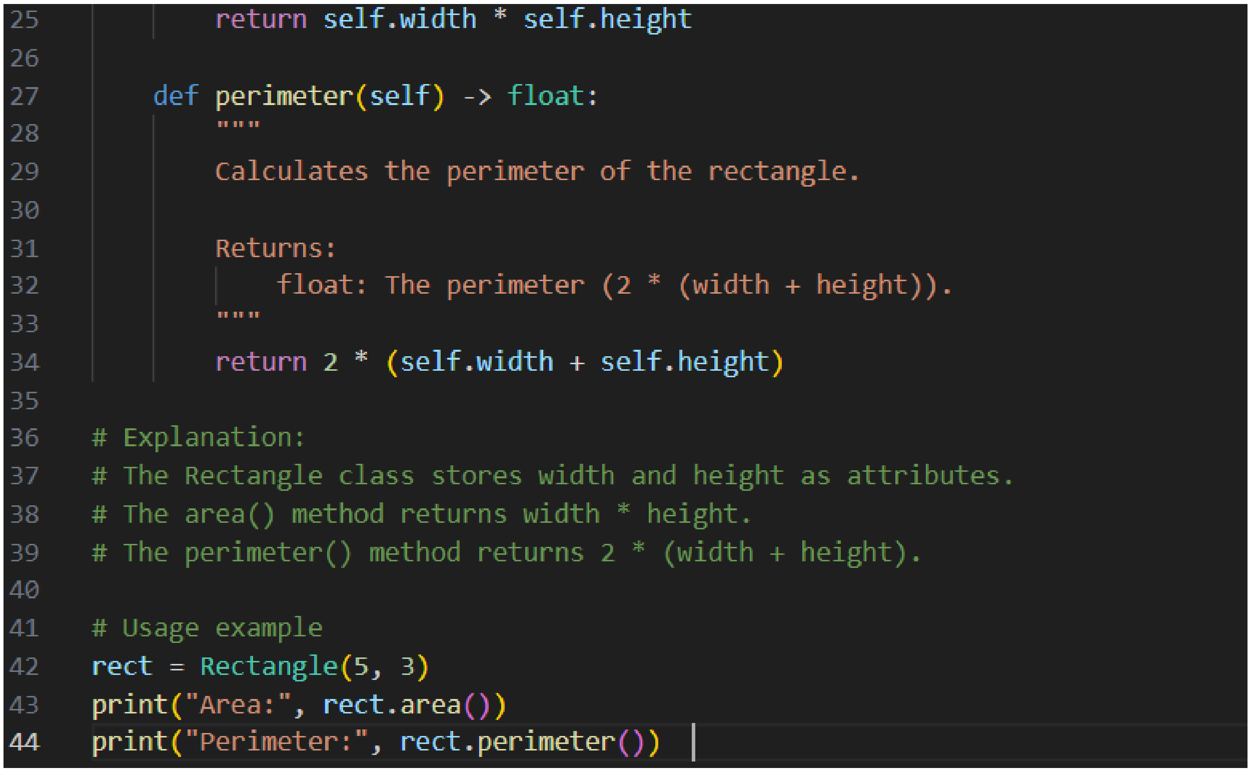
- Python code with explanation

Prompt :# Create a Python Rectangle class with methods to calculate area and perimeter.

Code :

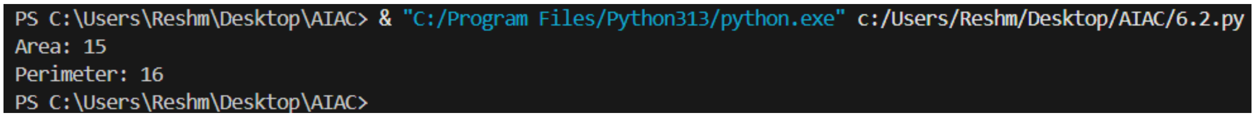


```python
class Rectangle:
    """
    Represents a rectangle with width and height.
    Provides methods to calculate area and perimeter.
    """

    def __init__(self, width: float, height: float):
        """
        Initialize a Rectangle instance.

        Args:
            width (float): The width of the rectangle.
            height (float): The height of the rectangle.
        """
        self.width = width
        self.height = height

    def area(self) -> float:
        """
        Calculates the area of the rectangle.

        Returns:
            float: The area (width * height).
        """
```



```python
        return self.width * self.height

    def perimeter(self) -> float:
        """
        Calculates the perimeter of the rectangle.

        Returns:
            float: The perimeter (2 * (width + height)).
        """
        return 2 * (self.width + self.height)

# Explanation:
# The Rectangle class stores width and height as attributes.
# The area() method returns width * height.
# The perimeter() method returns 2 * (width + height).

# Usage example
rect = Rectangle(5, 3)
print("Area:", rect.area())
print("Perimeter:", rect.perimeter())
```

Output :



```
PS C:\Users\Reshm\Desktop\AIAC> & "C:/Program Files/Python313/python.exe" c:/Users/Reshm/Desktop/AIAC/6.2.py
Area: 15
Perimeter: 16
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

he Rectangle class is well-designed, with clear docstrings and type hints for its attributes and methods.The constructor (__init__) initializes the rectangle's width and height.The area() method correctly calculates and returns the area as width * height.The perimeter() method correctly calculates and returns the perimeter as 2 * (width + height).The usage example demonstrates creating a Rectangle object and printing its area and perimeter.The code is readable, follows Python conventions, and is suitable for reuse and extension.