AIAC-ASSIGNMENT 9.1

NAME: T.SAI VARUN

Ht.no:2403A510C0

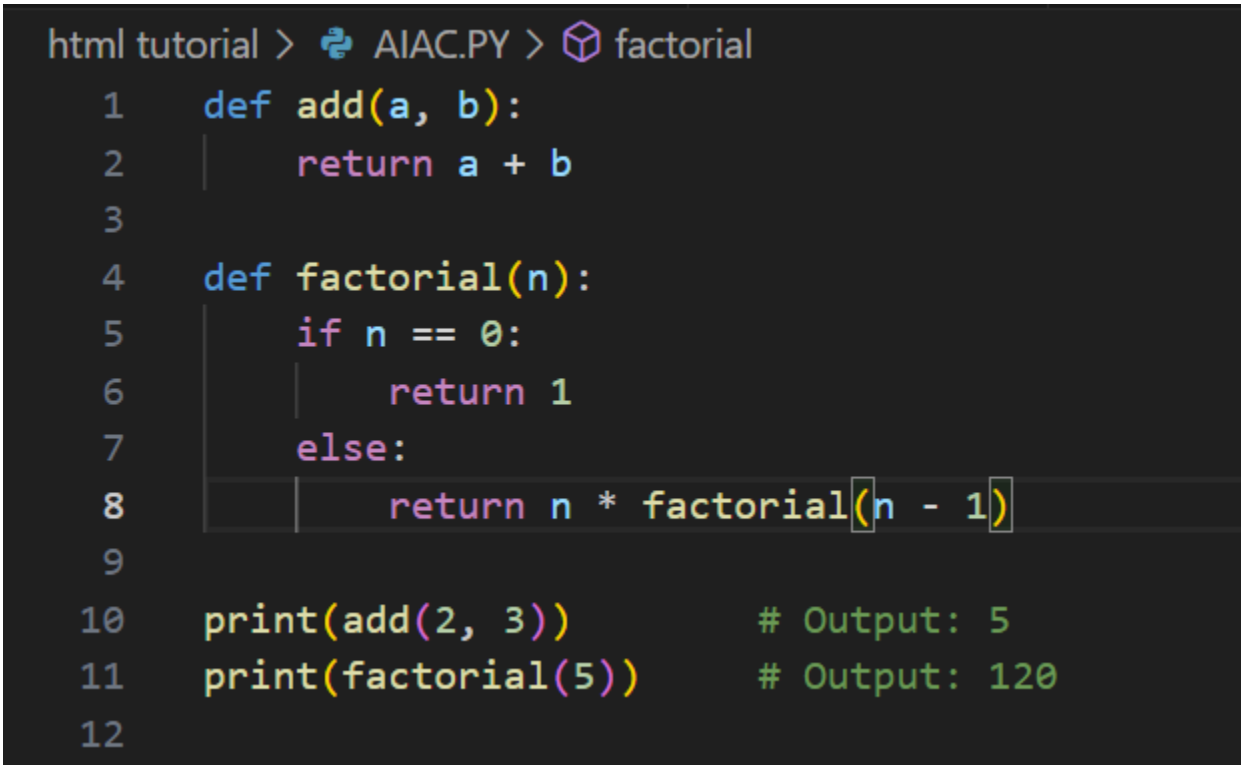
BATCH:05

Task 1: Google-Style Docstrings for Python Functions

Prompt:

Add Google-style docstrings to all functions in this Python code. Include function description, parameters with type hints, return values with type hints, and example usage.

Python code (Before docstring):



AIAC-ASSIGNMENT 9.1

Python Code(after docstring):

html tutorial > AIAC.PY > ...

```python
def add(a: int, b: int) -> int:
    """
    Add two integers.

    Args:
        a (int): First integer to add.
        b (int): Second integer to add.

    Returns:
        int: Sum of a and b.

    Example:
        >>> add(2, 3)
        5
    """
    return a + b

def factorial(n: int) -> int:
    """
    Calculate factorial of a non-negative integer using recursion.

    Args:
        n (int): Non-negative integer.

    Returns:
        int: Factorial of n.

    Example:
        >>> factorial(5)
        120
    """
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
print(add(2, 3))        # Output: 5
print(factorial(5))     # Output: 120

```

AIAC-ASSIGNMENT 9.1

Output:

```
TERMINAL                                                    powershell
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/html tutorial/AIAC.PY"
5
120
PS C:\Users\saiva\OneDrive\Documents\html tutorial>
```

Observation:

Docstrings clearly explain what each function does, its parameters, return type, and provide an example usage, improving readability and maintainability.

Task 2: Inline Comments for Complex Logic

Prompt: Add meaningful inline comments to this Python code, but only for complex logic.

Python Code(Before comments) :

```
html tutorial > AIAC.PY > ...
1   def fibonacci(n):
2       if n <= 1:
3           return n
4       else:
5           return fibonacci(n-1) + fibonacci(n-2)
6   print(fibonacci(10))
```

AIAC-ASSIGNMENT 9.1

Python Code(After comments):

```
html tutorial > AIAC.PY > ...
1   def fibonacci(n):
2       # Base case: if n is 0 or 1, return n itself
3       if n <= 1:
4           return n
5       else:
6           # Recursive call: sum of previous two Fibonacci numbers
7           return fibonacci(n-1) + fibonacci(n-2)
8   # Example usage
9   print(fibonacci(10))  # Output: 55
```

Output:

```
TERMINAL                                                    powershell
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Python313
\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/html tutorial/AIAC.PY"
55
PS C:\Users\saiva\OneDrive\Documents\html tutorial>
```

Observation:

Comments focus only on recursive logic, helping beginners understand recursion without cluttering with obvious statements.

Task 3: Module-Level Docstring

Prompt: Write a module-level docstring for this Python file, summarizing its purpose, dependencies, and main functions.

Python File (After Module Docstring):

AIAC-ASSIGNMENT 9.1

AIAC_9.py > ...

```python
"""
This module provides basic mathematical operations including addition and factorial calculation.

Dependencies:
- None (pure Python)

Functions:
- add(a, b): Returns the sum of two integers.
- factorial(n): Computes the factorial of a non-negative integer using recursion.
"""
def add(a, b):
    return a + b

def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
# Example usage:
if __name__ == "__main__":
    print("Addition of 3 and 5:", add(3, 5))
    print("Factorial of 5:", factorial(5))
```

Output:

Observation: Module-level docstring gives a high-level overview of the file, making it easy for new developers to understand its purpose quickly.

Task 4: Convert Inline Comments to Structured Docstrings

Prompt: Convert these inline comments into proper Google-style function docstrings. Keep the meaning intact.

Python Code with Inline Comments:

AIAC-ASSIGNMENT 9.1

AIAC_9.py > ...

```python
def multiply(a, b):
    # This function multiplies two numbers and returns the result
    return a * b
print(multiply(3, 4))  # Example usage
```

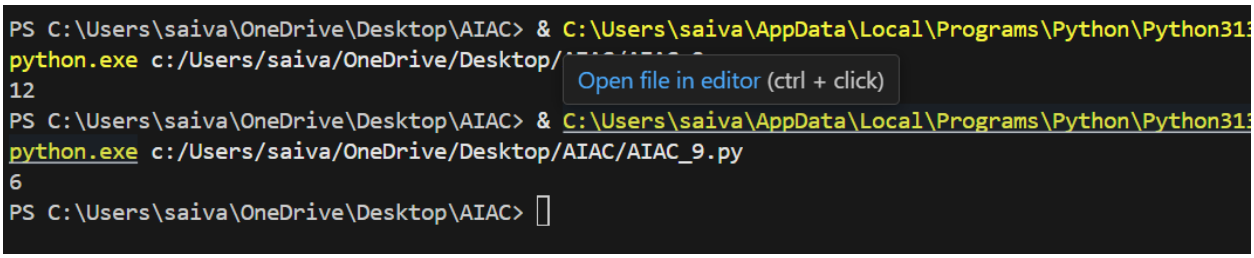
Python Code (After Docstring Conversion):

AIAC_9.py > multiply

```python
def multiply(a: int, b: int) -> int:
    """
    Multiply two integers.

    Args:
        a (int): First integer.
        b (int): Second integer.

    Returns:
        int: Product of a and b.

    Example:
        >>> multiply(2, 3)
        6
    """
    return a * b
print(multiply(2, 3))  # Output: 6
```

AIAC-ASSIGNMENT 9.1

Output:



Observation: Moving inline comments to structured docstrings standardizes documentation and allows tools to read function descriptions.

Task 5: Review and Correct Docstrings

Prompt : Review and correct the existing docstrings to match current code behavior. Follow Google-style format.

AIAC-ASSIGNMENT 9.1

Python Code (After Correction):

AIAC_9.py > ...

```python
def divide(a: float, b: float) -> float:
    """
    Divide two numbers.

    Args:
        a (float): Numerator.
        b (float): Denominator.

    Returns:
        float: Result of division.

    Example:
        >>> divide(6, 3)
        2.0
    """
    return a / b
print(divide(6, 3))  # Output: 2.0
```

Output:

```
TERMINAL                                                    powershell
PS C:\Users\saiva\OneDrive\Desktop\AIAC> & C:\Users\saiva\AppData\Local\Programs\Python\Python313\
python.exe c:/Users/saiva/OneDrive/Desktop/AIAC/AIAC_9.py
2.0
PS C:\Users\saiva\OneDrive\Desktop\AIAC>
```

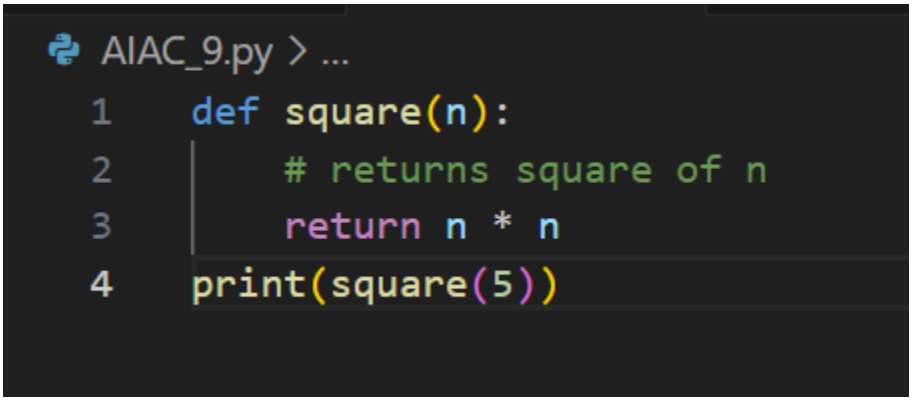
Observation: Corrected docstrings now accurately describe function behavior and improve reliability of documentation.

AIAC-ASSIGNMENT 9.1

Task 6: Prompt Comparison Experiment

Prompt 1 (Vague): Add comments to this function.

CODE:



Prompt 2 (Detailed): Add Google-style docstrings to this function including parameters, return type, and example usage.

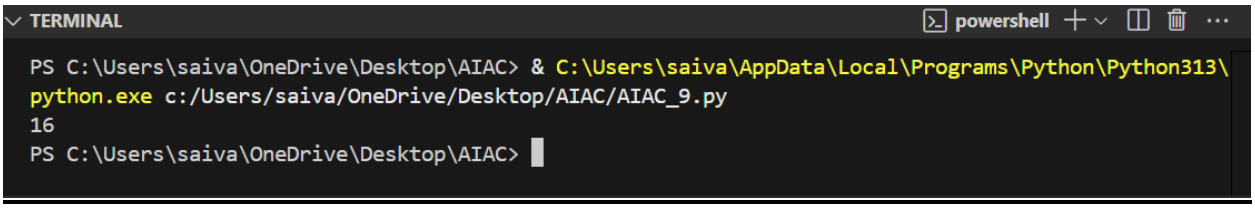
AIAC-ASSIGNMENT 9.1

Python Function (After Detailed Prompt):

AIAC_9.py > ...

```python
def square(n: int) -> int:
    """
    Calculate the square of an integer.

    Args:
        n (int): Number to be squared.

    Returns:
        int: Square of n.

    Example:
        >>> square(4)
        16
    """
    return n * n
print(square(4))  # Output: 16
```

Output:



Observation:

Detailed prompts produce better, complete, and standardized documentation, whereas vague prompts give only superficial comments.

Conclusion:

Detailed prompts produce better, complete, and standardized documentation, while vague prompts give only superficial comments.