LAB ASSIGNMENT-09

NAME:B.VISHNU VARDHAN

ROLLNO:2403A510F2

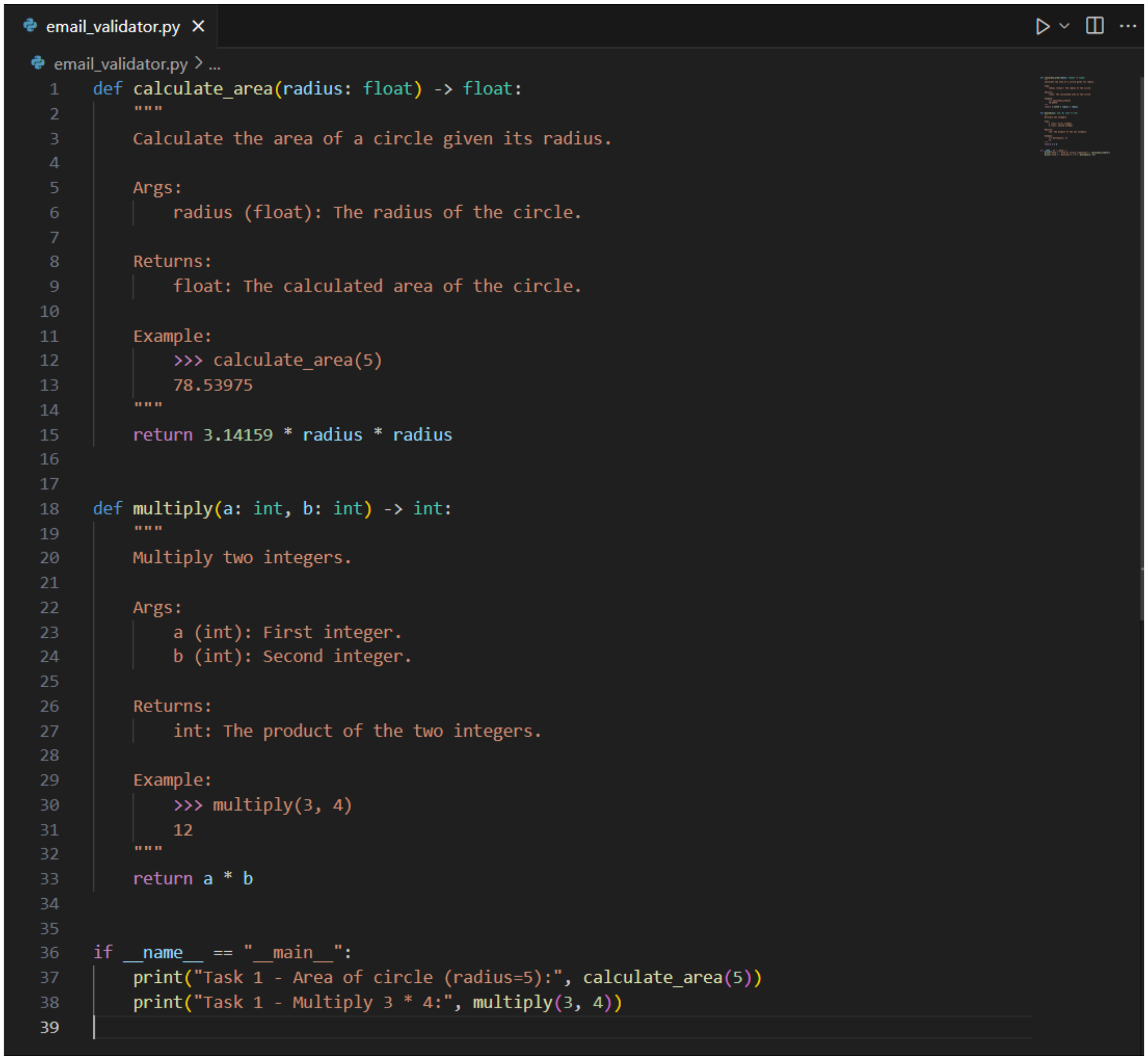
BATCH NO:06

TASK 1: GOOGLE-STYLE DOCSTRINGS FOR PYTHON FUNCTIONS

Prompt: “Add Google-style docstrings to all functions without providing any input-output examples. Ensure each docstring includes function description, parameters with type hints, return type hints, and example usage.”

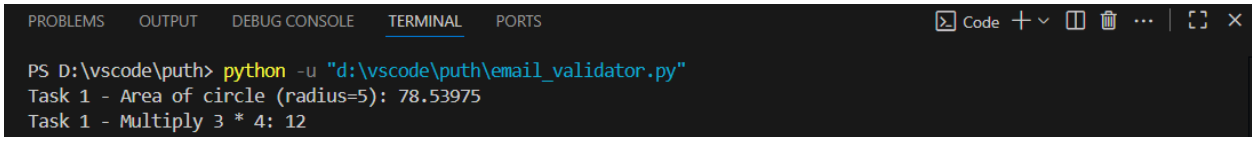
LAB ASSIGNMENT-09

Code:



```python
def calculate_area(radius: float) -> float:
    """
    Calculate the area of a circle given its radius.

    Args:
        radius (float): The radius of the circle.

    Returns:
        float: The calculated area of the circle.

    Example:
        >>> calculate_area(5)
        78.53975
    """
    return 3.14159 * radius * radius


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

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

    Returns:
        int: The product of the two integers.

    Example:
        >>> multiply(3, 4)
        12
    """
    return a * b


if __name__ == "__main__":
    print("Task 1 - Area of circle (radius=5):", calculate_area(5))
    print("Task 1 - Multiply 3 * 4:", multiply(3, 4))

```

Output:



```
PS D:\vscode\puth> python -u "d:\vscode\puth\email_validator.py"
Task 1 - Area of circle (radius=5): 78.53975
Task 1 - Multiply 3 * 4: 12
```

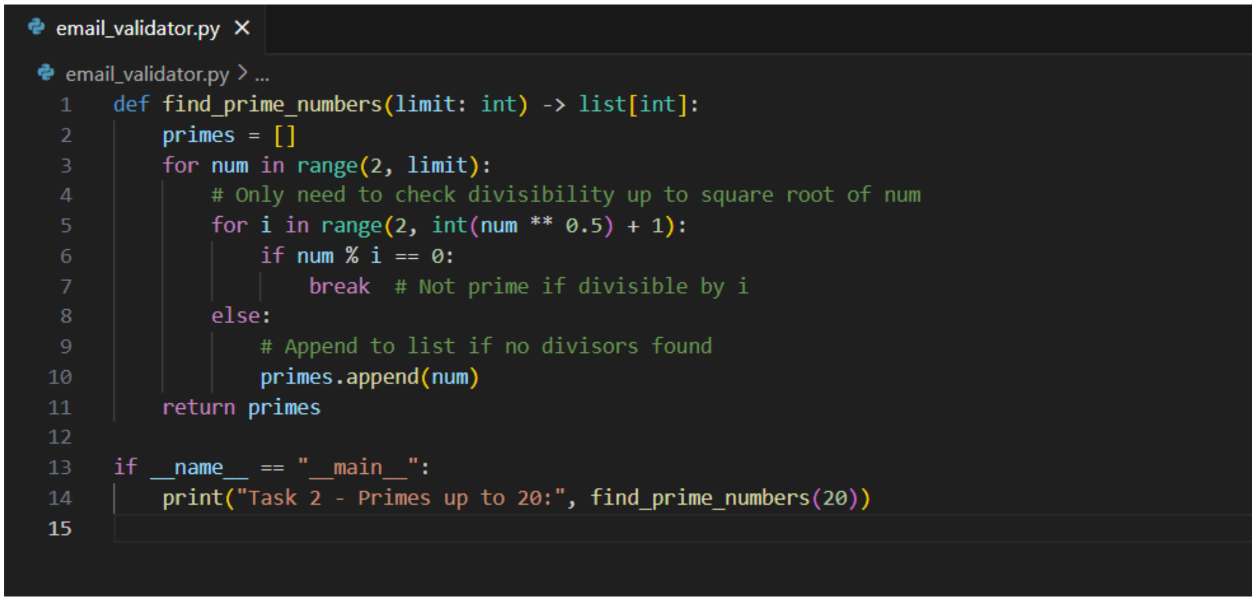
Observation: The function is documented clearly with parameter types, return type, and an example usage. This improves maintainability and understanding of the code.

LAB ASSIGNMENT-09

TASK 2: INLINE COMMENTS FOR COMPLEX LOGIC

Prompt: “Add meaningful inline comments explaining only non-intuitive logic in the function.”

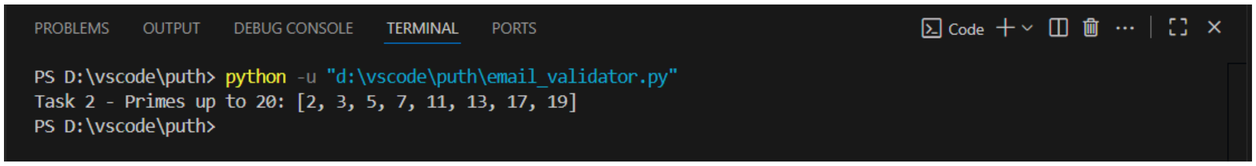
Code:



email_validator.py > ...

```python
def find_prime_numbers(limit: int) -> list[int]:
    primes = []
    for num in range(2, limit):
        # Only need to check divisibility up to square root of num
        for i in range(2, int(num ** 0.5) + 1):
            if num % i == 0:
                break  # Not prime if divisible by i
        else:
            # Append to list if no divisors found
            primes.append(num)
    return primes

if __name__ == "__main__":
    print("Task 2 - Primes up to 20:", find_prime_numbers(20))
```

Output:



```
PS D:\vscode\puth> python -u "d:\vscode\puth\email_validator.py"
Task 2 - Primes up to 20: [2, 3, 5, 7, 11, 13, 17, 19]
PS D:\vscode\puth>
```

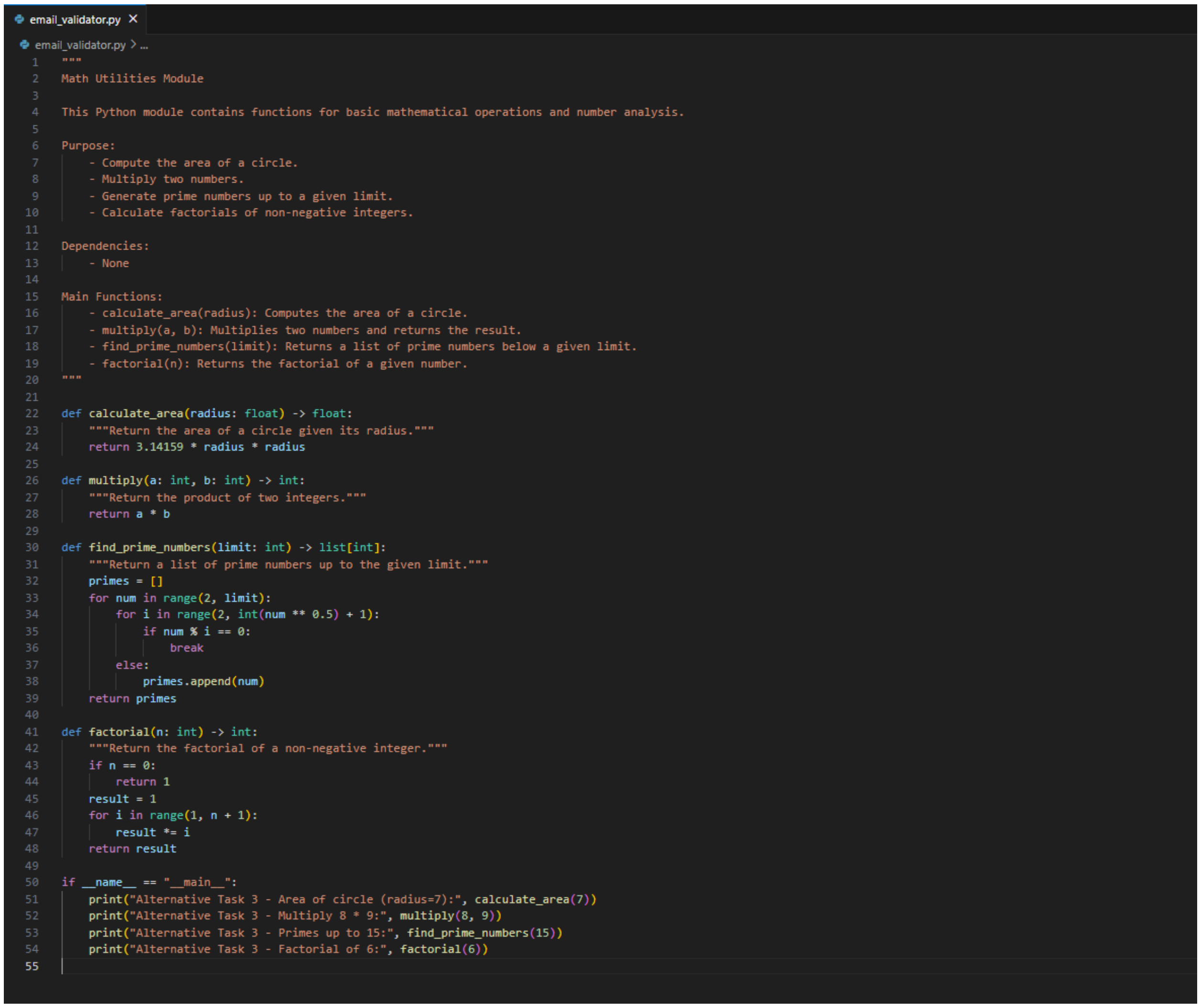
Observation: Inline comments highlight the efficient logic of checking divisibility up to the square root of the number, making the complex logic clearer.

TASK 3: MODULE-LEVEL DOCUMENTATION

Prompt: “Write a module-level docstring summarizing the purpose, dependencies, and main functions of the file.”

Code:

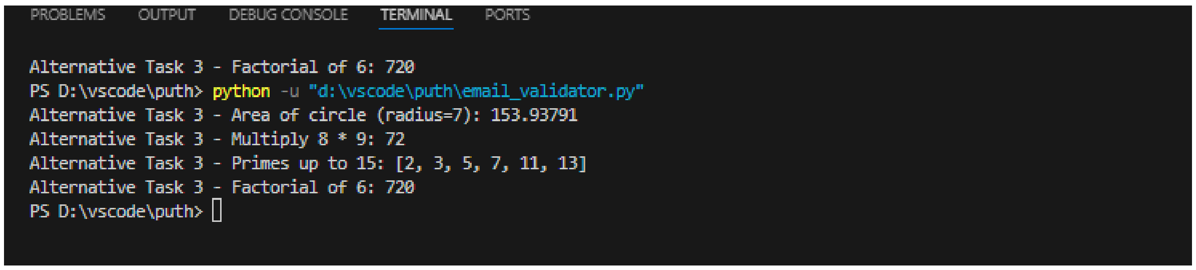
LAB ASSIGNMENT-09



email_validator.py

```python
"""
Math Utilities Module

This Python module contains functions for basic mathematical operations and number analysis.

Purpose:
    - Compute the area of a circle.
    - Multiply two numbers.
    - Generate prime numbers up to a given limit.
    - Calculate factorials of non-negative integers.

Dependencies:
    - None

Main Functions:
    - calculate_area(radius): Computes the area of a circle.
    - multiply(a, b): Multiplies two numbers and returns the result.
    - find_prime_numbers(limit): Returns a list of prime numbers below a given limit.
    - factorial(n): Returns the factorial of a given number.
"""

def calculate_area(radius: float) -> float:
    """Return the area of a circle given its radius."""
    return 3.14159 * radius * radius

def multiply(a: int, b: int) -> int:
    """Return the product of two integers."""
    return a * b

def find_prime_numbers(limit: int) -> list[int]:
    """Return a list of prime numbers up to the given limit."""
    primes = []
    for num in range(2, limit):
        for i in range(2, int(num ** 0.5) + 1):
            if num % i == 0:
                break
        else:
            primes.append(num)
    return primes

def factorial(n: int) -> int:
    """Return the factorial of a non-negative integer."""
    if n == 0:
        return 1
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

if __name__ == "__main__":
    print("Alternative Task 3 - Area of circle (radius=7):", calculate_area(7))
    print("Alternative Task 3 - Multiply 8 * 9:", multiply(8, 9))
    print("Alternative Task 3 - Primes up to 15:", find_prime_numbers(15))
    print("Alternative Task 3 - Factorial of 6:", factorial(6))
```

Output:



```
Alternative Task 3 - Factorial of 6: 720
PS D:\vscode\puth> python -u "d:\vscode\puth\email_validator.py"
Alternative Task 3 - Area of circle (radius=7): 153.93791
Alternative Task 3 - Multiply 8 * 9: 72
Alternative Task 3 - Primes up to 15: [2, 3, 5, 7, 11, 13]
Alternative Task 3 - Factorial of 6: 720
PS D:\vscode\puth>
```

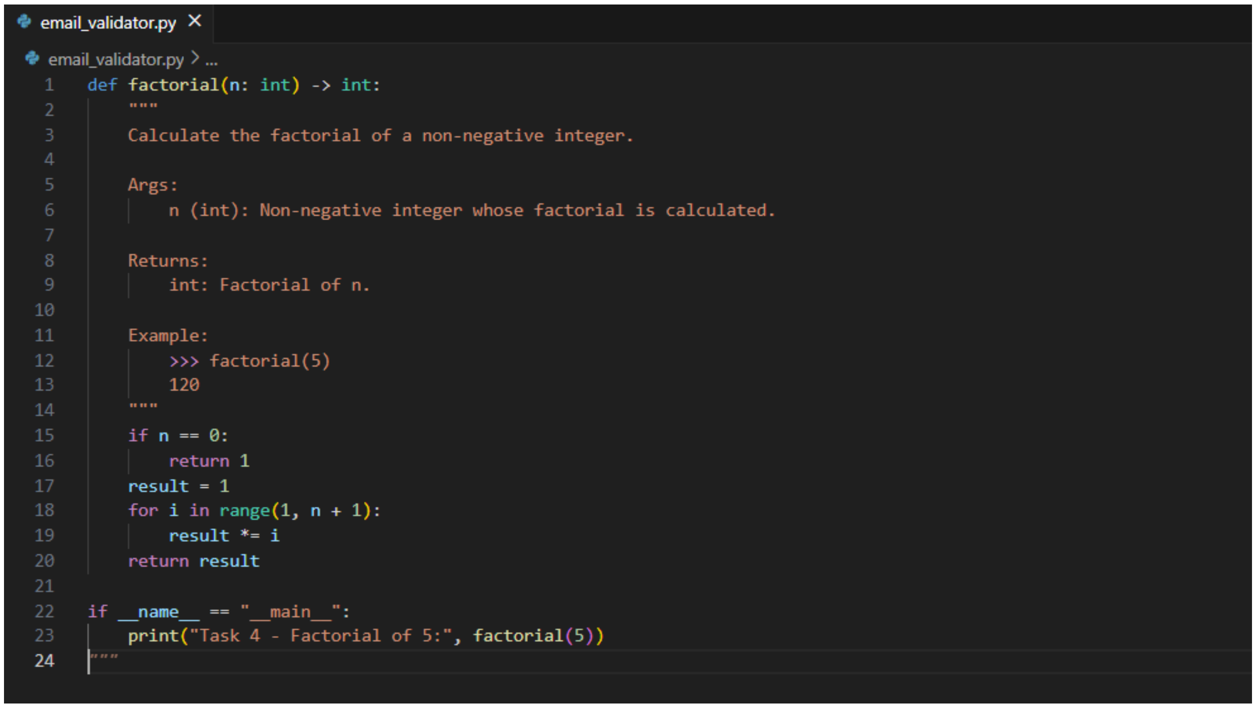
Observation: Provides a concise overview of the module, improving readability and usability by other developers.

TASK 4: CONVERT INLINE COMMENTS TO STRUCTURED DOCSTRINGS

LAB ASSIGNMENT-09

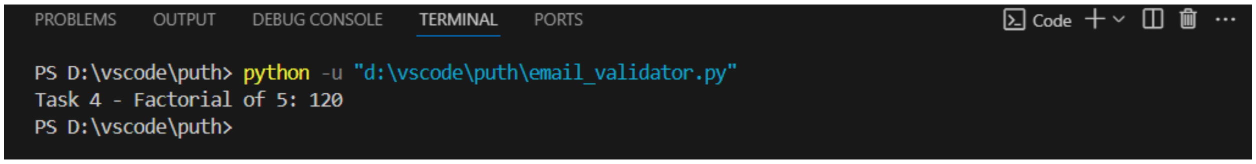
Prompt: "Transform inline comments in the factorial function into a structured Google-style docstring."

Code:



```python
def factorial(n: int) -> int:
    """
    Calculate the factorial of a non-negative integer.

    Args:
        n (int): Non-negative integer whose factorial is calculated.

    Returns:
        int: Factorial of n.

    Example:
        >>> factorial(5)
        120
    """
    if n == 0:
        return 1
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

if __name__ == "__main__":
    print("Task 4 - Factorial of 5:", factorial(5))
"""
```

Output:



```
PS D:\vscode\puth> python -u "d:\vscode\puth\email_validator.py"
Task 4 - Factorial of 5: 120
PS D:\vscode\puth>
```

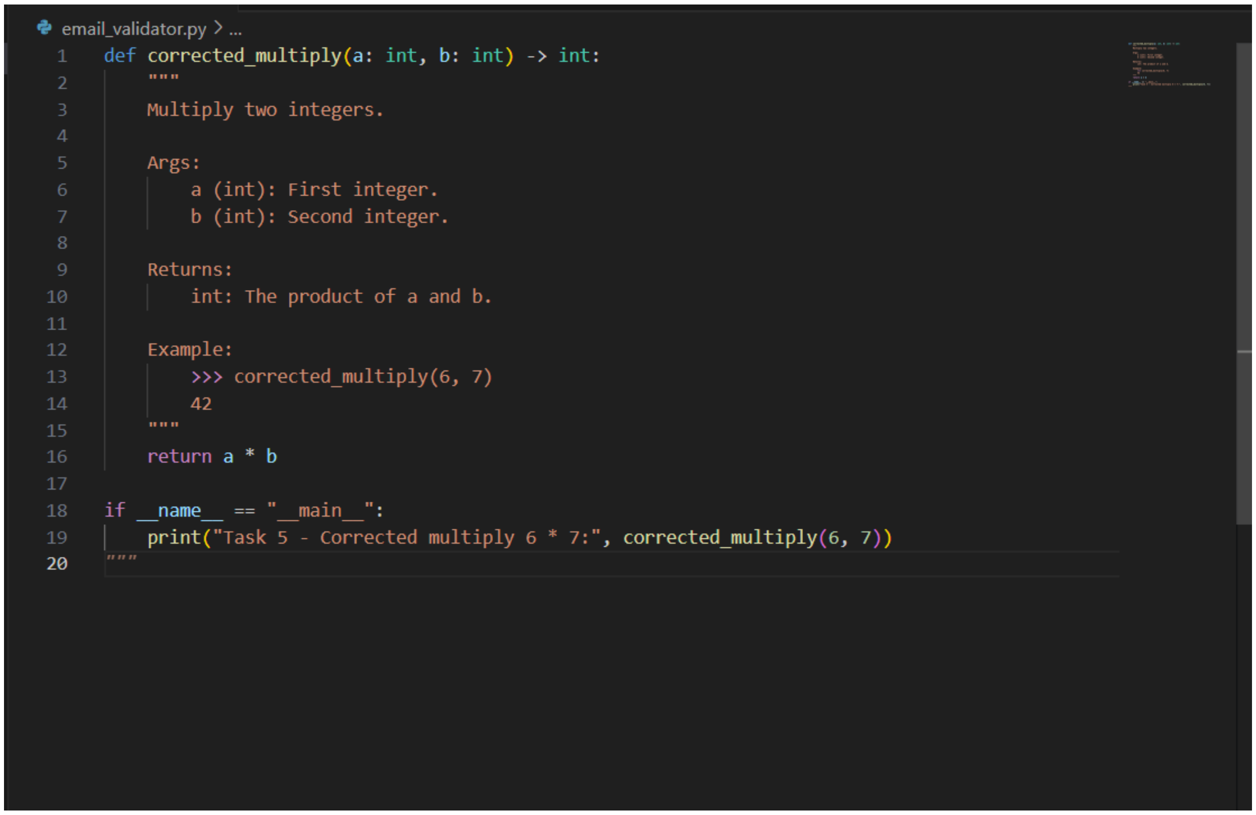
Observation: Converting inline comments to structured docstrings provides consistency and better tool support (e.g., for IDEs or documentation generators).

TASK 5: REVIEW AND CORRECT DOCSTRINGS

Prompt: "Identify and correct inaccuracies in the existing docstring of a multiplication function."

LAB ASSIGNMENT-09

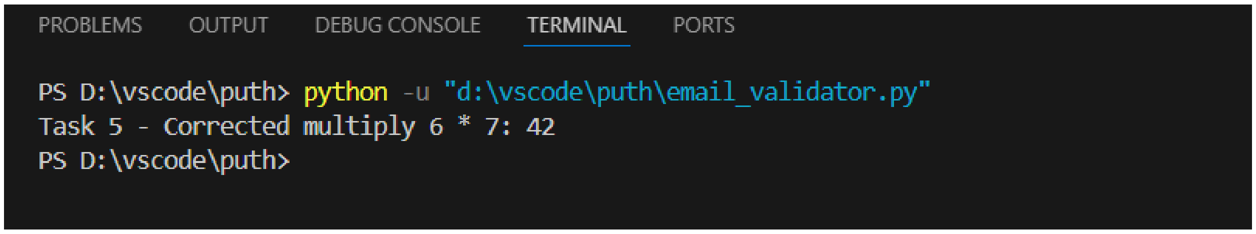
Code:



email_validator.py > ...

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

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

    Returns:
        int: The product of a and b.

    Example:
        >>> corrected_multiply(6, 7)
        42
    """
    return a * b

if __name__ == "__main__":
    print("Task 5 - Corrected multiply 6 * 7:", corrected_multiply(6, 7))
"""
```

Output:



```
PS D:\vscode\puth> python -u "d:\vscode\puth\email_validator.py"
Task 5 - Corrected multiply 6 * 7: 42
PS D:\vscode\puth>
```

Observation: The corrected docstring now correctly matches the function behavior and follows Google style.

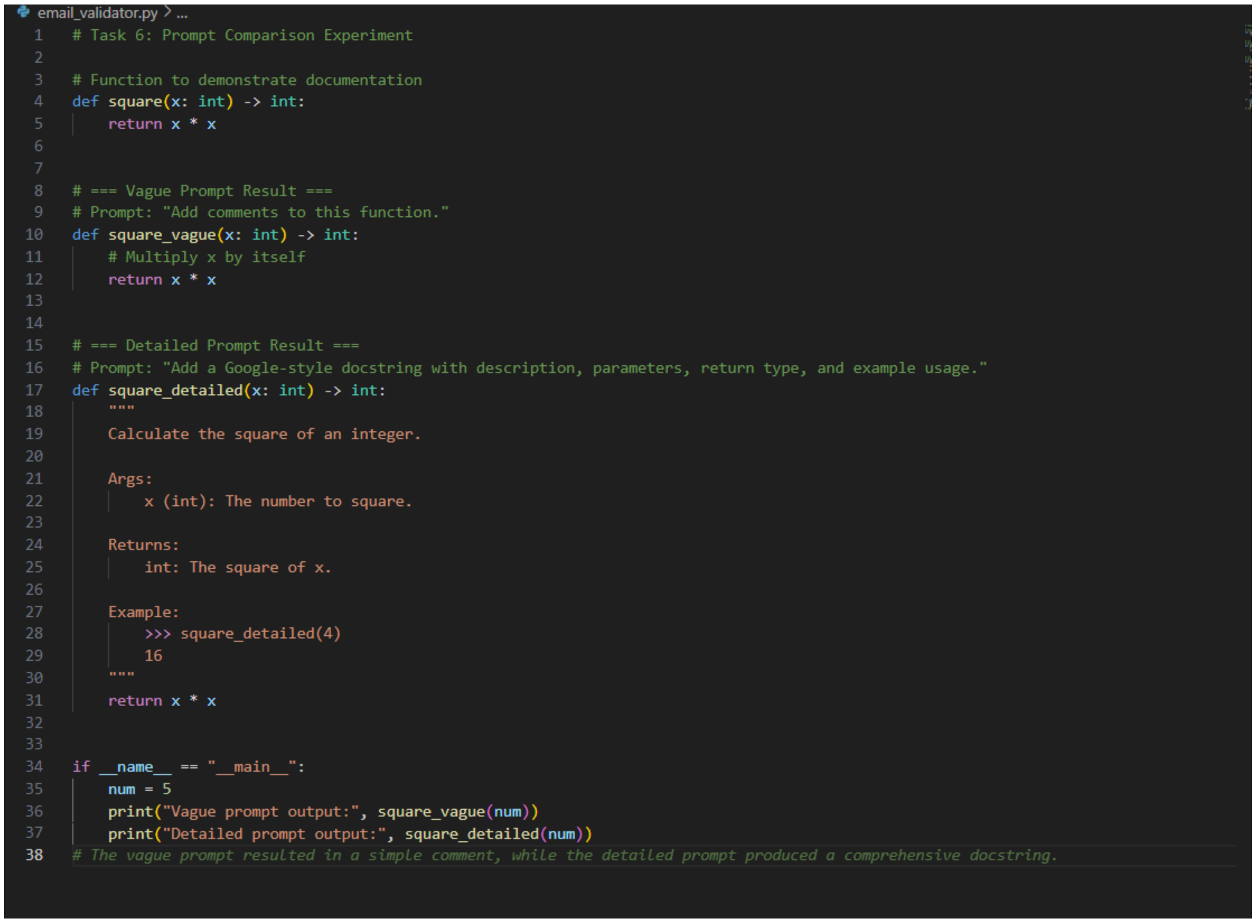
TASK 6: PROMPT COMPARISON EXPERIMENT

Prompt:

Add a Google-style docstring with description, parameters, return type, and example usage.

LAB ASSIGNMENT-09

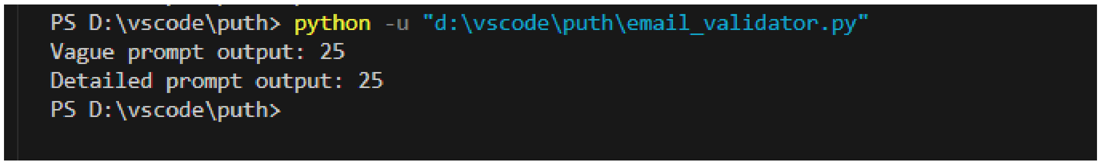
Code:



email_validator.py > ...

```python
# Task 6: Prompt Comparison Experiment

# Function to demonstrate documentation
def square(x: int) -> int:
    return x * x


# === Vague Prompt Result ===
# Prompt: "Add comments to this function."
def square_vague(x: int) -> int:
    # Multiply x by itself
    return x * x


# === Detailed Prompt Result ===
# Prompt: "Add a Google-style docstring with description, parameters, return type, and example usage."
def square_detailed(x: int) -> int:
    """
    Calculate the square of an integer.

    Args:
        x (int): The number to square.

    Returns:
        int: The square of x.

    Example:
        >>> square_detailed(4)
        16
    """
    return x * x


if __name__ == "__main__":
    num = 5
    print("Vague prompt output:", square_vague(num))
    print("Detailed prompt output:", square_detailed(num))
# The vague prompt resulted in a simple comment, while the detailed prompt produced a comprehensive docstring.
```

Output:



```
PS D:\vscode\puth> python -u "d:\vscode\puth\email_validator.py"
Vague prompt output: 25
Detailed prompt output: 25
PS D:\vscode\puth>
```

Observation: The detailed prompt produces a professional, complete docstring that improves usability and code clarity compared to a simple inline comment.