Assignment-9.2

Task-1:

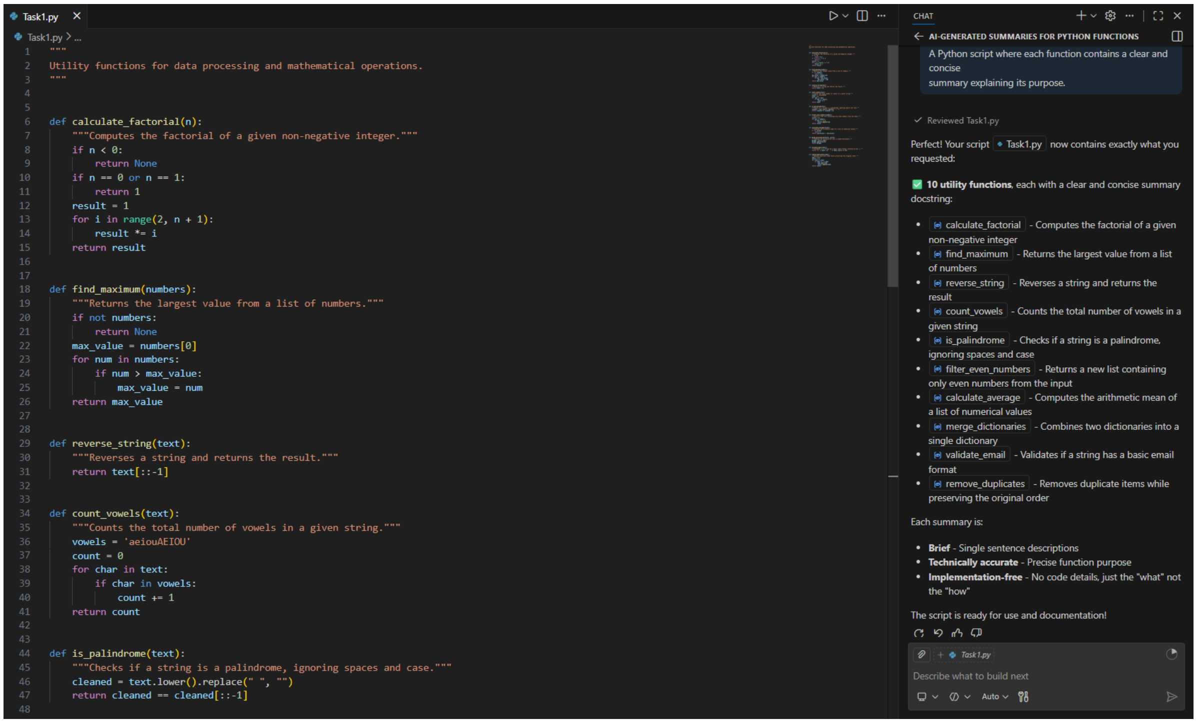
Use AI to generate concise functional summaries for each Python function in a given script.

Instructions:

- Provide a Python script to the AI.
- Ask the AI to write a short summary describing the purpose of each function.
- Ensure summaries are brief and technically accurate.
- Do not include code implementation details.

Expected Output -1:

A Python script where each function contains a clear and concise summary explaining its purpose.



```python
"""
Utility functions for data processing and mathematical operations.
"""


def calculate_factorial(n):
    """Computes the factorial of a given non-negative integer."""
    if n < 0:
        return None
    if n == 0 or n == 1:
        return 1
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result


def find_maximum(numbers):
    """Returns the largest value from a list of numbers."""
    if not numbers:
        return None
    max_value = numbers[0]
    for num in numbers:
        if num > max_value:
            max_value = num
    return max_value


def reverse_string(text):
    """Reverses a string and returns the result."""
    return text[::-1]


def count_vowels(text):
    """Counts the total number of vowels in a given string."""
    vowels = 'aeiouAEIOU'
    count = 0
    for char in text:
        if char in vowels:
            count += 1
    return count


def is_palindrome(text):
    """Checks if a string is a palindrome, ignoring spaces and case."""
    cleaned = text.lower().replace(" ", "")
    return cleaned == cleaned[::-1]

```

A Python script where each function contains a clear and concise
summary explaining its purpose.

✓ Reviewed Task1.py

Perfect! Your script Task1.py now contains exactly what you requested:

✅ **10 utility functions**, each with a clear and concise summary docstring:

- calculate_factorial - Computes the factorial of a given non-negative integer
- find_maximum - Returns the largest value from a list of numbers
- reverse_string - Reverses a string and returns the result
- count_vowels - Counts the total number of vowels in a given string
- is_palindrome - Checks if a string is a palindrome, ignoring spaces and case
- filter_even_numbers - Returns a new list containing only even numbers from the input
- calculate_average - Computes the arithmetic mean of a list of numerical values
- merge_dictionaries - Combines two dictionaries into a single dictionary
- validate_email - Validates if a string has a basic email format
- remove_duplicates - Removes duplicate items while preserving the original order

Each summary is:

- **Brief** - Single sentence descriptions
- **Technically accurate** - Precise function purpose
- **Implementation-free** - No code details, just the "what" not the "how"

The script is ready for use and documentation!

Task-2:

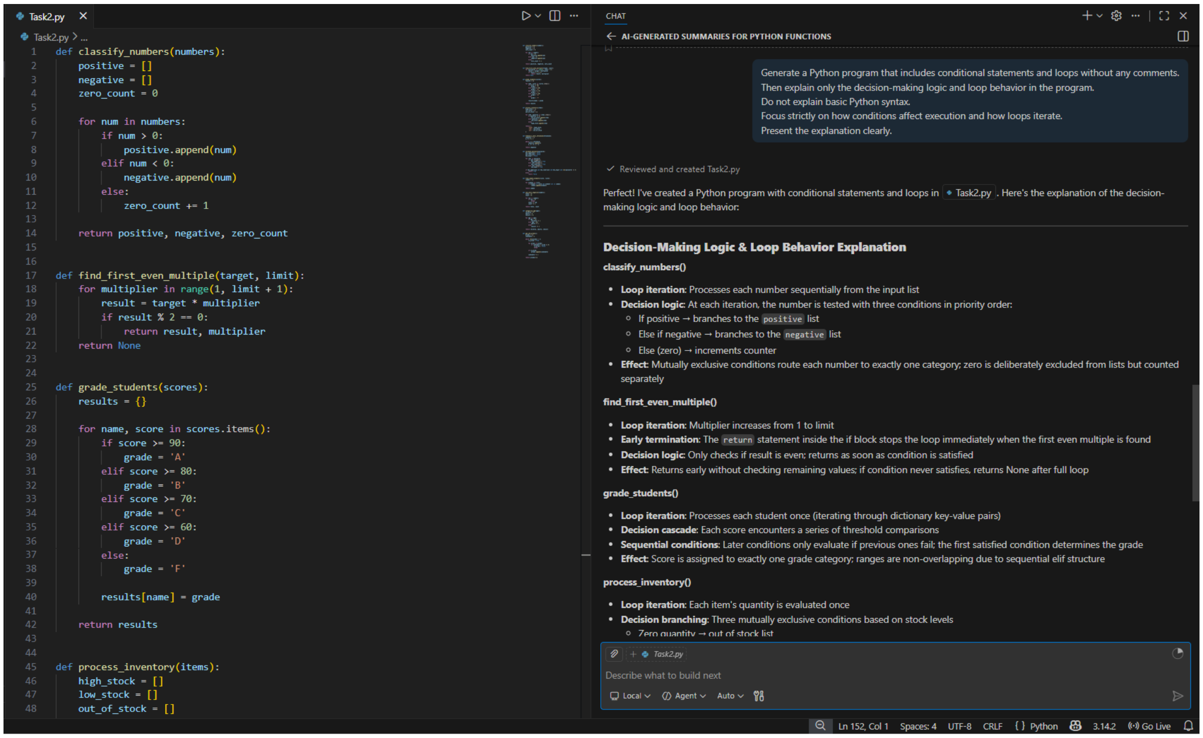
Use AI to document the logic behind conditional statements and loops in a Python program.

Instructions:

- **Provide a Python program without comments.**
- **Instruct AI to explain only decision-making logic and loop behavior.**
- **Skip basic syntax explanations.**

Expected Output -2:

Python code with clear explanations describing the logic of conditions and loops.



```python
def classify_numbers(numbers):
    positive = []
    negative = []
    zero_count = 0

    for num in numbers:
        if num > 0:
            positive.append(num)
        elif num < 0:
            negative.append(num)
        else:
            zero_count += 1

    return positive, negative, zero_count


def find_first_even_multiple(target, limit):
    for multiplier in range(1, limit + 1):
        result = target * multiplier
        if result % 2 == 0:
            return result, multiplier
    return None


def grade_students(scores):
    results = {}

    for name, score in scores.items():
        if score >= 90:
            grade = 'A'
        elif score >= 80:
            grade = 'B'
        elif score >= 70:
            grade = 'C'
        elif score >= 60:
            grade = 'D'
        else:
            grade = 'F'

        results[name] = grade

    return results


def process_inventory(items):
    high_stock = []
    low_stock = []
    out_of_stock = []
```

Generate a Python program that includes conditional statements and loops without any comments.
Then explain only the decision-making logic and loop behavior in the program.
Do not explain basic Python syntax.
Focus strictly on how conditions affect execution and how loops iterate.
Present the explanation clearly.

✓ Reviewed and created Task2.py

Perfect! I've created a Python program with conditional statements and loops in Task2.py. Here's the explanation of the decision-making logic and loop behavior:

Decision-Making Logic & Loop Behavior Explanation

classify_numbers()

- **Loop iteration:** Processes each number sequentially from the input list
- **Decision logic:** At each iteration, the number is tested with three conditions in priority order:
  - If positive → branches to the `positive` list
  - Else if negative → branches to the `negative` list
  - Else (zero) → increments counter
- **Effect:** Mutually exclusive conditions route each number to exactly one category; zero is deliberately excluded from lists but counted separately

find_first_even_multiple()

- **Loop iteration:** Multiplier increases from 1 to limit
- **Early termination:** The `return` statement inside the if block stops the loop immediately when the first even multiple is found
- **Decision logic:** Only checks if result is even; returns as soon as condition is satisfied
- **Effect:** Returns early without checking remaining values; if condition never satisfies, returns None after full loop

grade_students()

- **Loop iteration:** Processes each student once (iterating through dictionary key-value pairs)
- **Decision cascade:** Each score encounters a series of threshold comparisons
- **Sequential conditions:** Later conditions only evaluate if previous ones fail; the first satisfied condition determines the grade
- **Effect:** Score is assigned to exactly one grade category; ranges are non-overlapping due to sequential elif structure

process_inventory()

- **Loop iteration:** Each item's quantity is evaluated once
- **Decision branching:** Three mutually exclusive conditions based on stock levels
  - Zero quantity → out of stock list

Task-3:

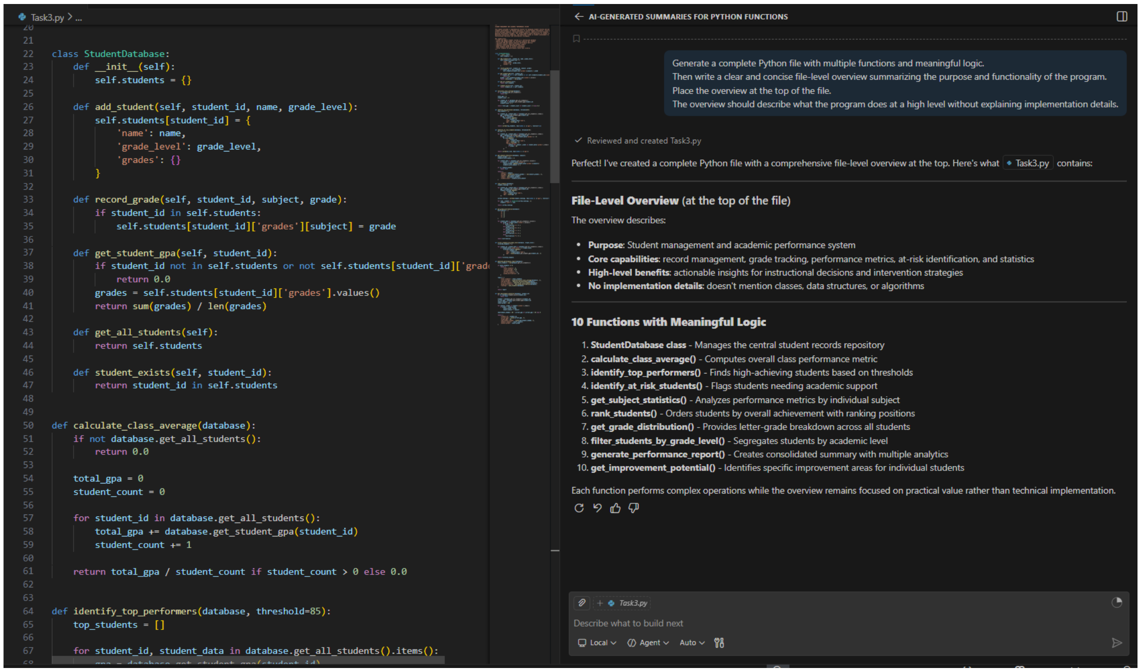
Use AI to generate a high-level overview describing the functionality of an entire Python file.

Instructions:

- **Provide the complete Python file to AI.**
- **Ask AI to write a brief overview summarizing the file's purpose and functionality.**
- **Place the overview at the top of the file.**

Expected Output -3:

A Python file with a clear and concise file-level overview at the beginning.



```python
class StudentDatabase:
    def __init__(self):
        self.students = {}

    def add_student(self, student_id, name, grade_level):
        self.students[student_id] = {
            'name': name,
            'grade_level': grade_level,
            'grades': {}
        }

    def record_grade(self, student_id, subject, grade):
        if student_id in self.students:
            self.students[student_id]['grades'][subject] = grade

    def get_student_gpa(self, student_id):
        if student_id not in self.students or not self.students[student_id]['grad
            return 0.0
        grades = self.students[student_id]['grades'].values()
        return sum(grades) / len(grades)

    def get_all_students(self):
        return self.students

    def student_exists(self, student_id):
        return student_id in self.students


def calculate_class_average(database):
    if not database.get_all_students():
        return 0.0

    total_gpa = 0
    student_count = 0

    for student_id in database.get_all_students():
        total_gpa += database.get_student_gpa(student_id)
        student_count += 1

    return total_gpa / student_count if student_count > 0 else 0.0


def identify_top_performers(database, threshold=85):
    top_students = []

    for student_id, student_data in database.get_all_students().items():
        gpa = database.get_student_gpa(student_id)
```

Generate a complete Python file with multiple functions and meaningful logic.
Then write a clear and concise file-level overview summarizing the purpose and functionality of the program.
Place the overview at the top of the file.
The overview should describe what the program does at a high level without explaining implementation details.

✓ Reviewed and created Task3.py

Perfect! I've created a complete Python file with a comprehensive file-level overview at the top. Here's what Task3.py contains:

File-Level Overview (at the top of the file)

The overview describes:

- **Purpose**: Student management and academic performance system
- **Core capabilities**: record management, grade tracking, performance metrics, at-risk identification, and statistics
- **High-level benefits**: actionable insights for instructional decisions and intervention strategies
- **No implementation details**: doesn't mention classes, data structures, or algorithms

10 Functions with Meaningful Logic

1. **StudentDatabase class** - Manages the central student records repository
2. **calculate_class_average()** - Computes overall class performance metric
3. **identify_top_performers()** - Finds high-achieving students based on thresholds
4. **identify_at_risk_students()** - Flags students needing academic support
5. **get_subject_statistics()** - Analyzes performance metrics by individual subject
6. **rank_students()** - Orders students by overall achievement with ranking positions
7. **get_grade_distribution()** - Provides letter-grade breakdown across all students
8. **filter_students_by_grade_level()** - Segregates students by academic level
9. **generate_performance_report()** - Creates consolidated summary with multiple analytics
10. **get_improvement_potential()** - Identifies specific improvement areas for individual students

Each function performs complex operations while the overview remains focused on practical value rather than technical implementation.

Task-4:

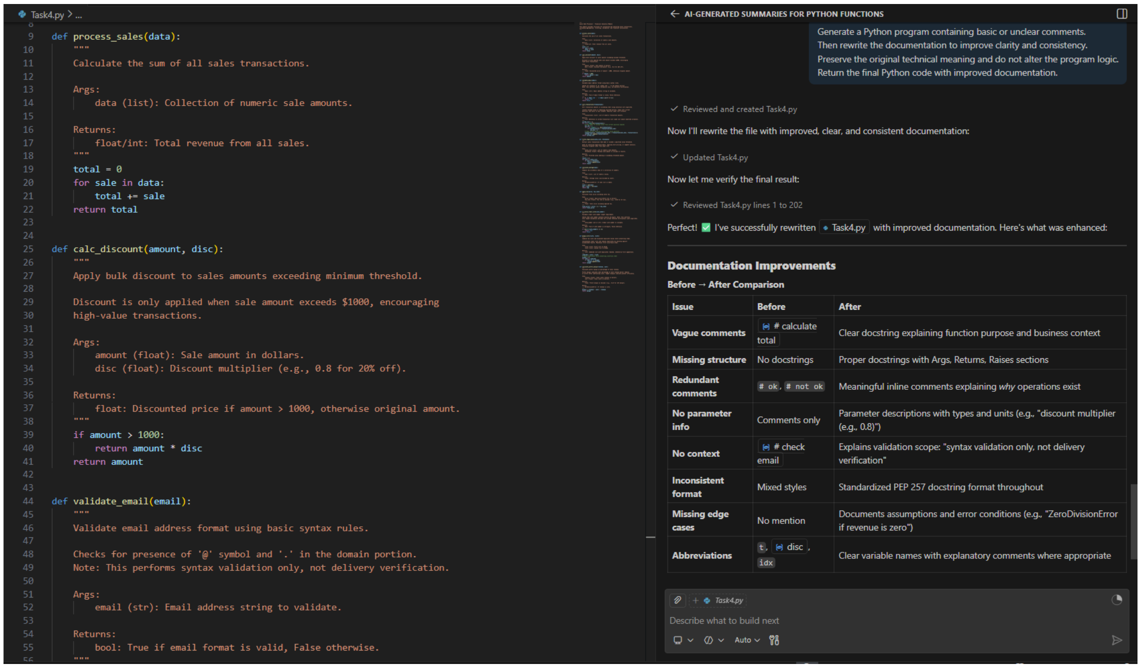
Use AI to improve clarity and consistency of existing documentation in Python code.

Instructions:

- **Provide Python code containing basic or unclear comments.**
- **Ask AI to rewrite the documentation to improve clarity and consistency.**
- **Ensure technical meaning remains unchanged.**

Expected Output -4:

Python code with refined and improved documentation that is clear and consistent.



```python
def process_sales(data):
    """
    Calculate the sum of all sales transactions.

    Args:
        data (list): Collection of numeric sale amounts.

    Returns:
        float/int: Total revenue from all sales.
    """
    total = 0
    for sale in data:
        total += sale
    return total


def calc_discount(amount, disc):
    """
    Apply bulk discount to sales amounts exceeding minimum threshold.

    Discount is only applied when sale amount exceeds $1000, encouraging
    high-value transactions.

    Args:
        amount (float): Sale amount in dollars.
        disc (float): Discount multiplier (e.g., 0.8 for 20% off).

    Returns:
        float: Discounted price if amount > 1000, otherwise original amount.
    """
    if amount > 1000:
        return amount * disc
    return amount


def validate_email(email):
    """
    Validate email address format using basic syntax rules.

    Checks for presence of '@' symbol and '.' in the domain portion.
    Note: This performs syntax validation only, not delivery verification.

    Args:
        email (str): Email address string to validate.

    Returns:
        bool: True if email format is valid, False otherwise.
    """
```

Generate a Python program containing basic or unclear comments.
Then rewrite the documentation to improve clarity and consistency.
Preserve the original technical meaning and do not alter the program logic.
Return the final Python code with improved documentation.

✓ Reviewed and created Task4.py

Now I'll rewrite the file with improved, clear, and consistent documentation:

✓ Updated Task4.py

Now let me verify the final result:

✓ Reviewed Task4.py lines 1 to 202

Perfect! ✅ I've successfully rewritten Task4.py with improved documentation. Here's what was enhanced:

Documentation Improvements

Before → After Comparison

Issue	Before	After
Vague comments	`# calculate total`	Clear docstring explaining function purpose and business context
Missing structure	No docstrings	Proper docstrings with Args, Returns, Raises sections
Redundant comments	`# ok`, `# not ok`	Meaningful inline comments explaining *why* operations exist
No parameter info	Comments only	Parameter descriptions with types and units (e.g., "discount multiplier (e.g., 0.8)")
No context	`# check email`	Explains validation scope: "syntax validation only, not delivery verification"
Inconsistent format	Mixed styles	Standardized PEP 257 docstring format throughout
Missing edge cases	No mention	Documents assumptions and error conditions (e.g., "ZeroDivisionError if revenue is zero")
Abbreviations	`t`, `disc`, `idx`	Clear variable names with explanatory comments where appropriate

Task-5:

Study the impact of prompt detail on AI-generated documentation quality.

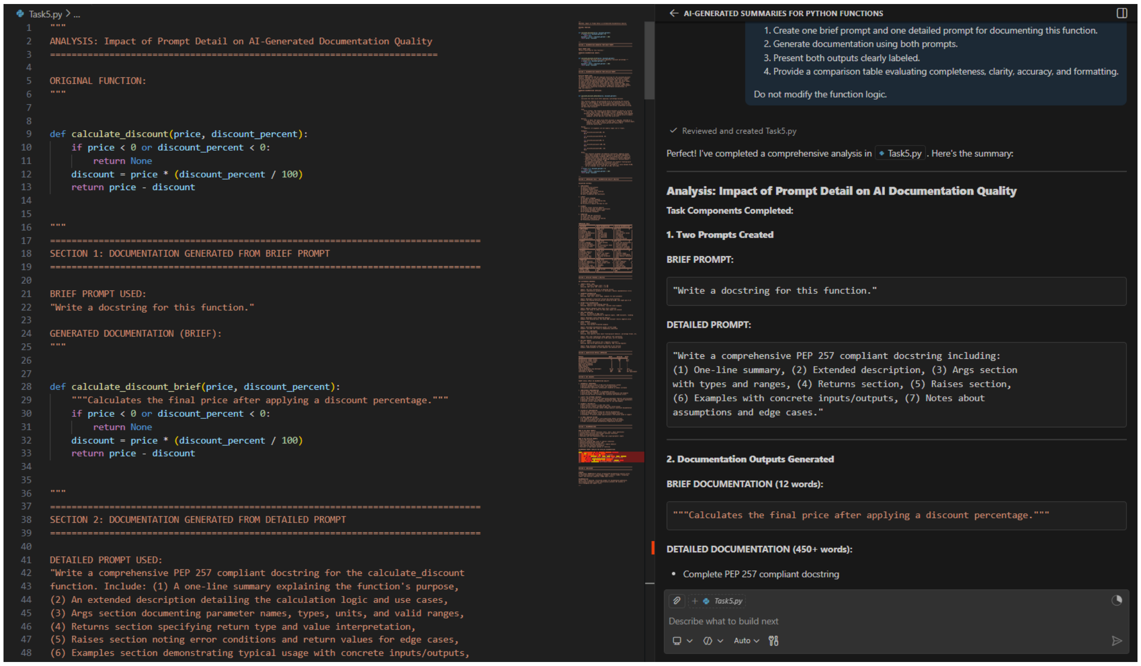
Instructions:

Create two prompts: one brief and one detailed.

- **Use both prompts to document the same Python function.**
- **Compare the generated outputs.**

Expected Output -5:

A comparison table highlighting differences in completeness, clarity, and accuracy of documentation.


Task5.py > ...
AI-GENERATED SUMMARIES FOR PYTHON FUNCTIONS
Task5.py
Describe what to build next
Auto

```python
"""
ANALYSIS: Impact of Prompt Detail on AI-Generated Documentation Quality
========================================================================

ORIGINAL FUNCTION:
"""


def calculate_discount(price, discount_percent):
    if price < 0 or discount_percent < 0:
        return None
    discount = price * (discount_percent / 100)
    return price - discount


"""
================================================================================
SECTION 1: DOCUMENTATION GENERATED FROM BRIEF PROMPT
================================================================================

BRIEF PROMPT USED:
"Write a docstring for this function."

GENERATED DOCUMENTATION (BRIEF):
"""


def calculate_discount_brief(price, discount_percent):
    """Calculates the final price after applying a discount percentage."""
    if price < 0 or discount_percent < 0:
        return None
    discount = price * (discount_percent / 100)
    return price - discount


"""
================================================================================
SECTION 2: DOCUMENTATION GENERATED FROM DETAILED PROMPT
================================================================================

DETAILED PROMPT USED:
"Write a comprehensive PEP 257 compliant docstring for the calculate_discount
function. Include: (1) A one-line summary explaining the function's purpose,
(2) An extended description detailing the calculation logic and use cases,
(3) Args section documenting parameter names, types, units, and valid ranges,
(4) Returns section specifying return type and value interpretation,
(5) Raises section noting error conditions and return values for edge cases,
(6) Examples section demonstrating typical usage with concrete inputs/outputs,
```

1. Create one brief prompt and one detailed prompt for documenting this function.
2. Generate documentation using both prompts.
3. Present both outputs clearly labeled.
4. Provide a comparison table evaluating completeness, clarity, accuracy, and formatting.

Do not modify the function logic.

✓ Reviewed and created Task5.py

Perfect! I've completed a comprehensive analysis in Task5.py. Here's the summary:

Analysis: Impact of Prompt Detail on AI Documentation Quality

Task Components Completed:

1. Two Prompts Created

BRIEF PROMPT:

```
"Write a docstring for this function."
```

DETAILED PROMPT:

```
"Write a comprehensive PEP 257 compliant docstring including:
(1) One-line summary, (2) Extended description, (3) Args section
with types and ranges, (4) Returns section, (5) Raises section,
(6) Examples with concrete inputs/outputs, (7) Notes about
assumptions and edge cases."
```

---

2. Documentation Outputs Generated

BRIEF DOCUMENTATION (12 words):

```
"""Calculates the final price after applying a discount percentage."""
```

DETAILED DOCUMENTATION (450+ words):

- Complete PEP 257 compliant docstring