

AI ASSISTED CODING

LAB-9: Documentation Generation: Automatic Documentation and Code Comments

Roll no: 2503A51L17

Name: Simra Tahseen

Batch: 24BTCAICSB19

Task-1 Description: (Automatic Code Commenting)

Scenario: You have been given a Python function without comments.

```
def calculate_discount(price, discount_rate):
    return price - (price * discount_rate / 100)
```

- Use an AI tool (or manually simulate it) to generate line-by-line comments for the function.
- Modify the function so that it includes a docstring in Google-style or NumPy-style format.
- Compare the auto-generated comments with your manually written version

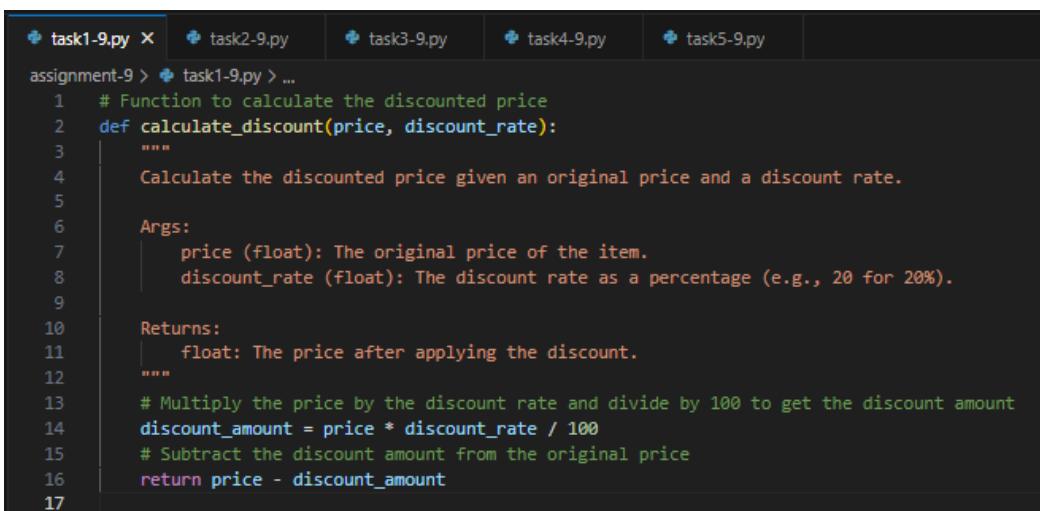
Prompt: def calculate_discount(price, discount_rate):

```
    return price - (price * discount_rate / 100)
```

- Generate line-by-line comments for the function.
- Modify the function so that it includes a docstring in Google-style or NumPy-style format.

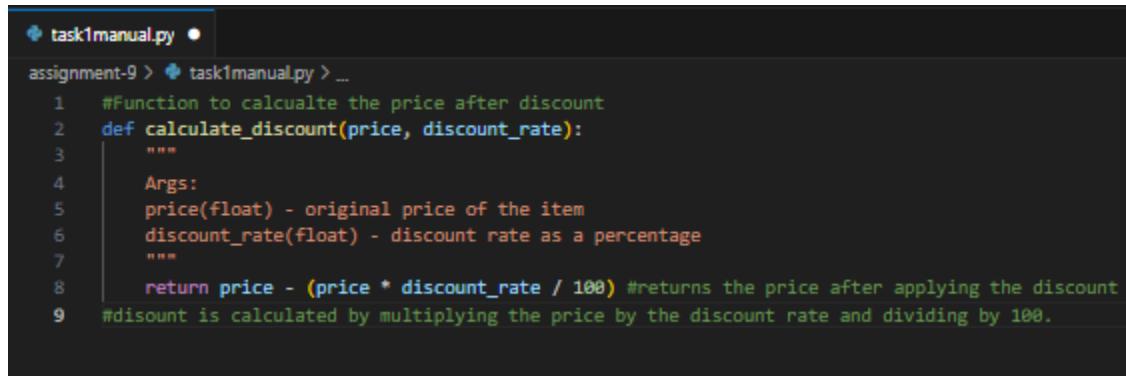
Code Generated:

Copilot Generated comments:



```
task1-9.py X task2-9.py task3-9.py task4-9.py task5-9.py
assignment-9 > task1-9.py > ...
1  # Function to calculate the discounted price
2  def calculate_discount(price, discount_rate):
3      """
4          Calculate the discounted price given an original price and a discount rate.
5
6          Args:
7              price (float): The original price of the item.
8              discount_rate (float): The discount rate as a percentage (e.g., 20 for 20%).
9
10         Returns:
11             float: The price after applying the discount.
12         """
13
14         # Multiply the price by the discount rate and divide by 100 to get the discount amount
15         discount_amount = price * discount_rate / 100
16
17         # Subtract the discount amount from the original price
18         return price - discount_amount
```

Manually Written comments:



```
task1manual.py •
assignment-9 > task1manual.py > ...
1  #Function to calculate the price after discount
2  def calculate_discount(price, discount_rate):
3      """
4          Args:
5              price(float) - original price of the item
6              discount_rate(float) - discount rate as a percentage
7          """
8      return price - (price * discount_rate / 100) #returns the price after applying the discount
9      #discount is calculated by multiplying the price by the discount rate and dividing by 100.
```

Observation: I observed how comments and docstrings improve code readability. Adding both AI-generated and manual comments showed the difference in clarity. AI-generated comments were quick but sometimes generic, while manually written comments were more accurate and contextual. The use of a structured docstring format (Google or NumPy style) made the function self-explanatory for future developers.

Task-2 Description: (API Documentation Generator)

Scenario: A team is building a Library Management System with multiple functions.

```
def add_book(title, author, year):
    # code to add book
    pass
def issue_book(book_id, user_id):
    # code to issue book
    Pass
```

- Write a Python script that uses docstrings for each function (with input, output, and description).
- Use a documentation generator tool (like pdoc, Sphinx, or MkDocs) to automatically create HTML documentation.
- Submit both the code and the generated documentation as output.

Prompt:

```
def add_book(title, author, year):
    # code to add book
    pass
def issue_book(book_id, user_id):
    # code to issue book
    Pass
```

- Write a Python script that uses docstrings for each function (with input, output, and description).
- Use a documentation generator tool (like pdoc, Sphinx, or MkDocs) to automatically create HTML documentation.

Code Generated:

Python Code:

```
task2-9.py X
assignment-9 > task2-9.py > ...
1 def add_book(title, author, year):
2     """
3         Add a new book to the library system.
4
5     Args:
6         title (str): The title of the book.
7         author (str): The author of the book.
8         year (int): The year the book was published.
9
10    Returns:
11        bool: True if the book was added successfully, False otherwise.
12    """
13    # code to add book
14    pass
15
16 def issue_book(book_id, user_id):
17     """
18         Issue a book to a user.
19
20     Args:
21         book_id (int): The unique identifier for the book.
22         user_id (int): The unique identifier for the user.
23
24     Returns:
25        bool: True if the book was issued successfully, False otherwise.
26    """
27    # code to issue book
28    pass
29
```

HTML Code:

```
index.html X
docs > index.html > html
1 <!doctype html>
2 <html>
3 <head>
4     <meta charset="utf-8">
5     <meta http-equiv="refresh" content="0; url=../task2-9.html"/>
6 </head>
7 </html>
8
```

```

docs > t2.html > html > body > main.pdoc > section#add_book > div.docstring > p
  1   <!doctype html>
  2   <html lang="en">
  3   <head>...
  4   <style>/*! custom.css */</style></head>
  5   <body>
  6   <nav class="pdoc">...
  7   </nav>
  8   <main class="pdoc">
  9     <section class="module-info">...
 10     <section id="add_book">
 11       <input id="add_book-view-source" class="view-source-toggle-state" type="checkbox" aria-hidden="true" checked="" data-pdoc="true" data-pdoc-type="checkbox" data-pdoc-value="checked" data-pdoc-name="add_book-view-source" data-pdoc-args="[]"/>
 12       <a href="#add_book">Add book</a>
 13       <div class="pdoc-code codehilite"><pre><span><span id="add_book-2"><a href="#add_book-2"><span class="linenos">1</span></a></span></pre></div>
 14
 15       <div class="docstring"><p>Add a new book to the library system.</p>
 16
 17     <p>Args:</p>
 18       title (str): The title of the book.
 19       author (str): The author of the book.
 20       year (int): The publication year of the book.</p>
 21
 22     <p>Returns:</p>
 23       bool: True if the book was added successfully, False otherwise.</p>
 24   </div>
 25
 26   </section>
 27   <section id="issue_book">...
 28   </section>
 29 </main>
 30 </body>
 31 </html>

```

Javascript Code:

```

search.js | 46
docs > search.js > ...
  1 window.pdocSearch = (function() {
  2   /* elasticsearch - http://weissong.github.io */
  3   /* Copyright (C) 2017 Oliver Nightingale */
  4   /* Copyright (C) 2017 Wei Song */
  5   /* MIT Licensed */
  6   /* */
  7   /* elasticsearch - https://github.com/elastic/elasticsearch */
  8
  9   let searchIndex;
 10  if (docs._isPrebuiltIndex) {
 11    console.info("using precompiled search index");
 12    searchIndex = elasticsearch.Index.load(docs);
 13  } else {
 14    console.time("Building search index");
 15    // mirrored in buildSearchIndex.js (part 1)
 16    searchIndex = elasticsearch.Index.create();
 17    this.pipeline = elasticsearch.Pipeline.create();
 18    this.pipeline.remove(elasticsearch.StopWordFilter);
 19    this.addField("fullname");
 20    this.addField("annotation");
 21    this.addField("default_value");
 22    this.addField("signature");
 23    this.addField("title");
 24    this.addField("doc");
 25    this.setRef("fullname");
 26  }
 27  for (let doc of docs) {
 28    searchIndex.addDoc(doc);
 29  }
 30  console.timeEnd("Building search index");
 31
 32  return (term) => searchIndex.search(term, {
 33    fields: [
 34      { fullname: { boost: 4 } },
 35      { title: { boost: 2 } },
 36      { annotation: { boost: 2 } },
 37      { default_value: { boost: 2 } },
 38      { signature: { boost: 2 } },
 39      { bases: { boost: 2 } },
 40      { doc: { boost: 1 } },
 41    ],
 42    expand: true
 43  });
 44}
 45
 46 })([]);

```

Output:

The screenshot shows the API Documentation interface for the task2-9 module. It contains two main sections: `add_book` and `issue_book`.

add_book:

- Docstring:** Add a new book to the library system.
- Arguments:**
 - `title` (str): The title of the book.
 - `author` (str): The author of the book.
 - `year` (int): The publication year of the book.
- Returns:** bool: True if the book was added successfully, False otherwise.

issue_book:

- Docstring:** Issue a book to a user.
- Arguments:**
 - `book_id` (int): The unique identifier for the book.
 - `user_id` (int): The unique identifier for the user.
- Returns:** bool: True if the book was issued successfully, False otherwise.

Observation: I observed that the importance of proper docstrings in functions. By writing detailed docstrings for the library management system, I saw how documentation tools (like pdoc, Sphinx, or MkDocs) can automatically create HTML files for easy reference. This task highlighted how automated documentation makes projects more professional, user-friendly, and maintainable, especially in team environments.

Task-3 Description: (AI-Assisted Code Summarization)

Scenario: You are reviewing a colleague's codebase containing long functions.

```
def process_sensor_data(data):
    cleaned = [x for x in data if x is not None]
    avg = sum(cleaned)/len(cleaned)
    anomalies = [x for x in cleaned if abs(x - avg) > 10]
    return {"average": avg, "anomalies": anomalies}
```

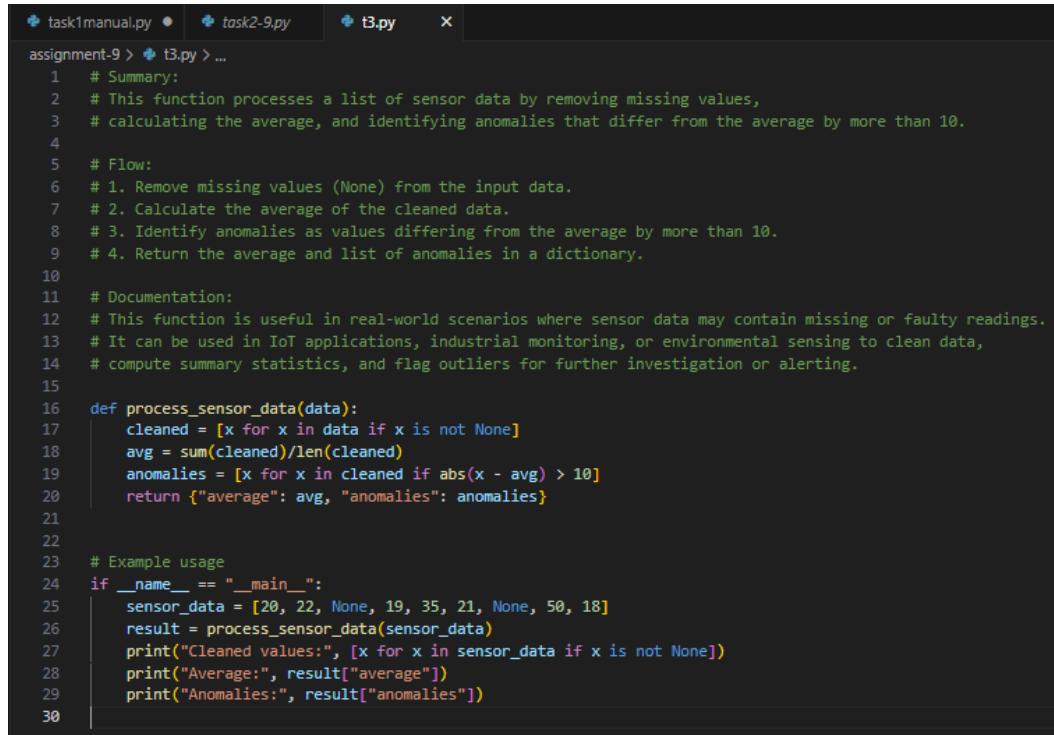
- Generate a summary comment explaining the purpose of the function in 2–3 lines.
- Create a flow-style comment (step-by-step explanation).
- Write a short paragraph of documentation describing possible use cases of this function in real-world scenarios

Prompt:

```
def process_sensor_data(data):
    cleaned = [x for x in data if x is not None]
    avg = sum(cleaned)/len(cleaned)
    anomalies = [x for x in cleaned if abs(x - avg) > 10]
    return {"average": avg, "anomalies": anomalies}
```

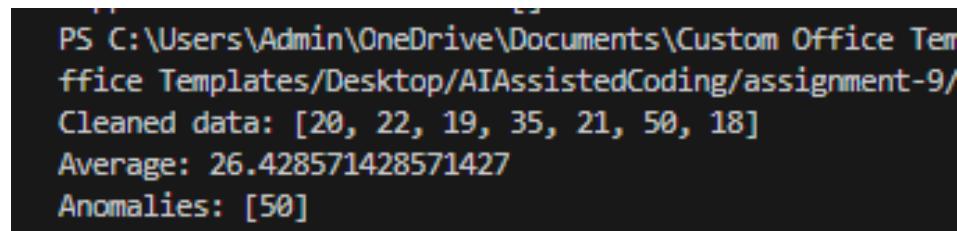
- Generate a summary comment explaining the purpose of the function in 2–3 lines.
- Create a flow-style comment (step-by-step explanation).
- Write a short paragraph of documentation describing possible use cases of this function in real-world scenarios. Also include a example and ouput which shows values, anomalies, etc.

Code Generated:



```
task1manual.py ● task2-9.py ✘ t3.py X
assignment-9 > t3.py > ...
1  # Summary:
2  # This function processes a list of sensor data by removing missing values,
3  # calculating the average, and identifying anomalies that differ from the average by more than 10.
4
5  # Flow:
6  # 1. Remove missing values (None) from the input data.
7  # 2. Calculate the average of the cleaned data.
8  # 3. Identify anomalies as values differing from the average by more than 10.
9  # 4. Return the average and list of anomalies in a dictionary.
10
11 # Documentation:
12 # This function is useful in real-world scenarios where sensor data may contain missing or faulty readings.
13 # It can be used in IoT applications, industrial monitoring, or environmental sensing to clean data,
14 # compute summary statistics, and flag outliers for further investigation or alerting.
15
16 def process_sensor_data(data):
17     cleaned = [x for x in data if x is not None]
18     avg = sum(cleaned)/len(cleaned)
19     anomalies = [x for x in cleaned if abs(x - avg) > 10]
20     return {"average": avg, "anomalies": anomalies}
21
22
23 # Example usage
24 if __name__ == "__main__":
25     sensor_data = [20, 22, None, 19, 35, 21, None, 50, 18]
26     result = process_sensor_data(sensor_data)
27     print("Cleaned values:", [x for x in sensor_data if x is not None])
28     print("Average:", result["average"])
29     print("Anomalies:", result["anomalies"])
30
```

Output:



```
PS C:\Users\Admin\OneDrive\Documents\Custom Office Templates\Desktop\AIAssistedCoding\assignment-9>
Cleaned data: [20, 22, 19, 35, 21, 50, 18]
Average: 26.428571428571427
Anomalies: [50]
```

Observation: I observed how AI or summarization techniques can explain long functions clearly. Instead online-by-line reading, summaries provided the purpose of the function in a few sentences. The flow-style comments helped me break down logic step by step, and writing use cases showed how documentation can connect code to real-world applications, such as sensor monitoring or anomaly detection.

Task-4 Description: (Real-Time Project Documentation)

Scenario:

You are part of a project team that develops a Chatbot Application. The team needs documentation for maintainability.

- Write a README.md file for the chatbot project (include project description, installation steps, usage, and example).
- Add inline comments in the chatbot's main Python script (focus on explaining logic, not trivial code).
- Use an AI-assisted tool (or simulate it) to generate a usage guide in plain English from your code comments.
- Reflect: How does automated documentation help in real-time projects compared to manual documentation?

Prompt:

Write a Python program that simulates a simple chatbot.

Create project documentation for a chatbot application:

- Write a README.md (description, installation, usage, example).
- Add inline comments in the chatbot's main Python script (explain logic).
- Generate a plain-English usage guide from those comments.

Code Generated:

```
task4-9.py  README.md
assignment-9 > task4-9.py > ...
1  # Simple Chatbot Implementation
2  # This script simulates a simple rule-based chatbot that responds to user input.
3
4  def chatbot_response(user_input):
5      """Return a response based on user input."""
6      # Convert input to lowercase for easier matching
7      user_input = user_input.lower()
8      # Respond to greetings
9      if any(greet in user_input for greet in ["hi", "hello", "hey"]):
10          return "Hello! How can I help you today?"
11      # Respond to asking about the bot
12      elif "your name" in user_input:
13          return "I'm a simple chatbot."
14      # Respond to thanks
15      elif "thank" in user_input:
16          return "You're welcome!"
17      # Respond to exit
18      elif any(bye in user_input for bye in ["bye", "exit", "quit"]):
19          return "Goodbye! Have a nice day!"
20      # Default response
21      else:
22          return "Sorry, I didn't understand that."
23
24
25  if __name__ == "__main__":
26      print("Welcome to the Simple Chatbot! Type 'bye' to exit.")
27      while True:
28          user_input = input("You: ")
29          response = chatbot_response(user_input)
30          print("Bot:", response)
31          if response.startswith("Goodbye"):
32              break
33
```

Documentation Generated(README.md file):

```
* task4-9.py | README.md X
assignment-9 > README.md > # Simple Chatbot Application
1  # Simple Chatbot Application
2
3  ## Description
4  This project is a simple rule-based chatbot written in Python. The chatbot responds to user input with basic greetings, answers, and fallback responses. It demonstrates basic natural language processing and control flow in Python.
5
6  ## Installation
7  1. Make sure you have Python 3 installed on your system.
8  2. Download or clone this repository to your local machine.
9
10 ## Usage
11 Run the chatbot script in your terminal:
12
13 ````bash
14 python task4-9.py
15 ````

16 Type your messages and the chatbot will respond. Type 'bye' to exit.
17
18
19 ## Example
20 ````

21 Welcome to the Simple Chatbot! Type 'bye' to exit.
22 You: hi
23 Bot: Hello! How can I help you today?
24 You: what's your name?
25 Bot: I'm a simple chatbot.
26 You: thank you
27 Bot: You're welcome!
28 You: bye
29 Bot: Goodbye! Have a nice day!
30 ````
```

Output:

```
PS C:\Users\Admin\OneDrive\Documents\Custom Office Templates\Desktop\ffice Templates\Desktop/AIAssistedCoding\assignment-9\task4-9.py"
Welcome to the Simple Chatbot! Type 'bye' to exit.
You: Hello
Bot: Hello! How can I help you today?
You: Your name?
Bot: I'm a simple chatbot.
You: Bye
Bot: Goodbye! Have a nice day!
```

Observation: The entire chatbot program and its documentation were generated using AI. Through this task, I observed how AI can create code, add comments, and prepare README files automatically. I learned that documentation is just as important as the code itself because it makes the project easier to understand and maintain. This showed me how AI-assisted tools can save time and effort in real projects while still ensuring clarity.