==========

Assignment Number: 9.1

==========

NAME:- P. Sai H.T.NO:-2303A52395

==========

⇨ Problem 1:-

Consider the following Python function:

def find_max(numbers):

return max(numbers)

Task:

• Write documentation for the function in all three formats:

(a) Docstring

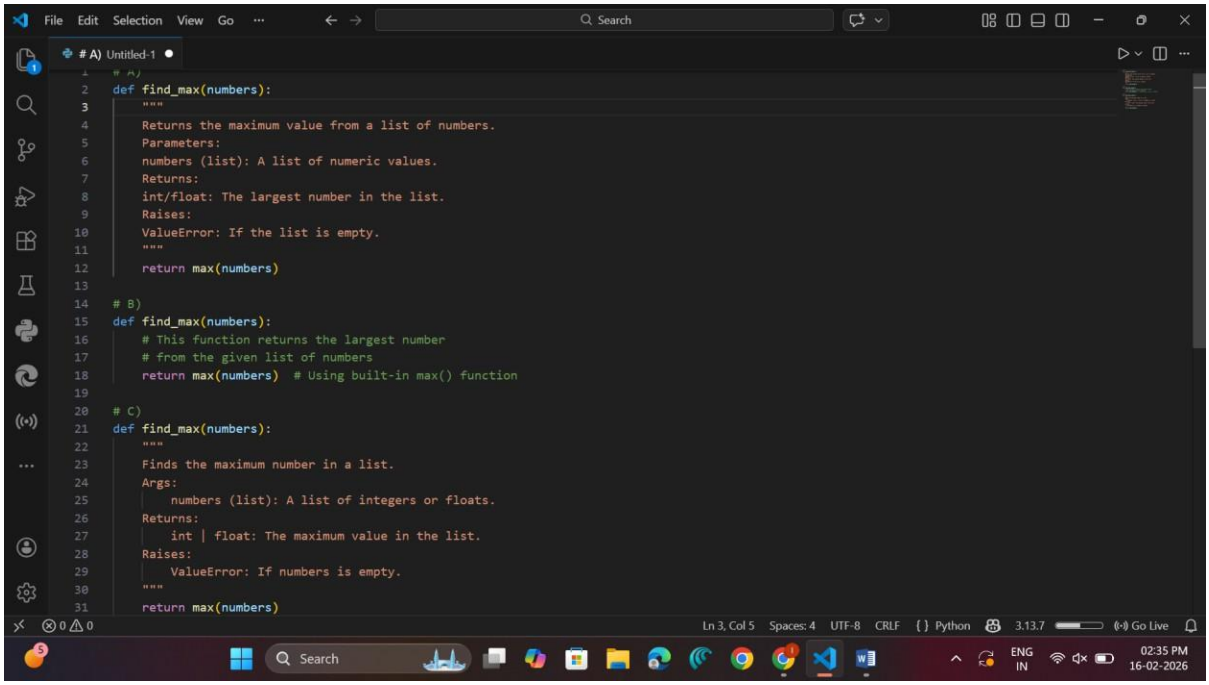
(b) Inline comments

(c) Google-style documentation

• Critically compare the three approaches. Discuss the advantages, disadvantages, and suitable use cases of each style.

• Recommend which documentation style is most effective for a mathematical utilities library and justify your answer.

CODE:-



Style	Advantages	Disadvantages	Suitable Use Case
Docstring (Basic)	Simple and readable	Not standardized	Small projects
Inline Comments	Easy to understand line-by-line	Cannot generate automatic documentation	Beginners / Learning
Google Style	Structured, professional, tool-compatible	Slightly longer	Large projects & libraries

• Recommend which documentation style is most effective for a mathematical utilities library and justify your answer.

Recommendation for Mathematical Utilities Library

Google-style documentation is most effective because:

- Works well with tools like **pydoc, Sphinx**
- Clear parameter and return description
- Professional and standardized
- Improves maintainability

==========

Problem 2:

Consider the following Python function:

```
def login(user, password, credentials):

return credentials.get(user) == password
```

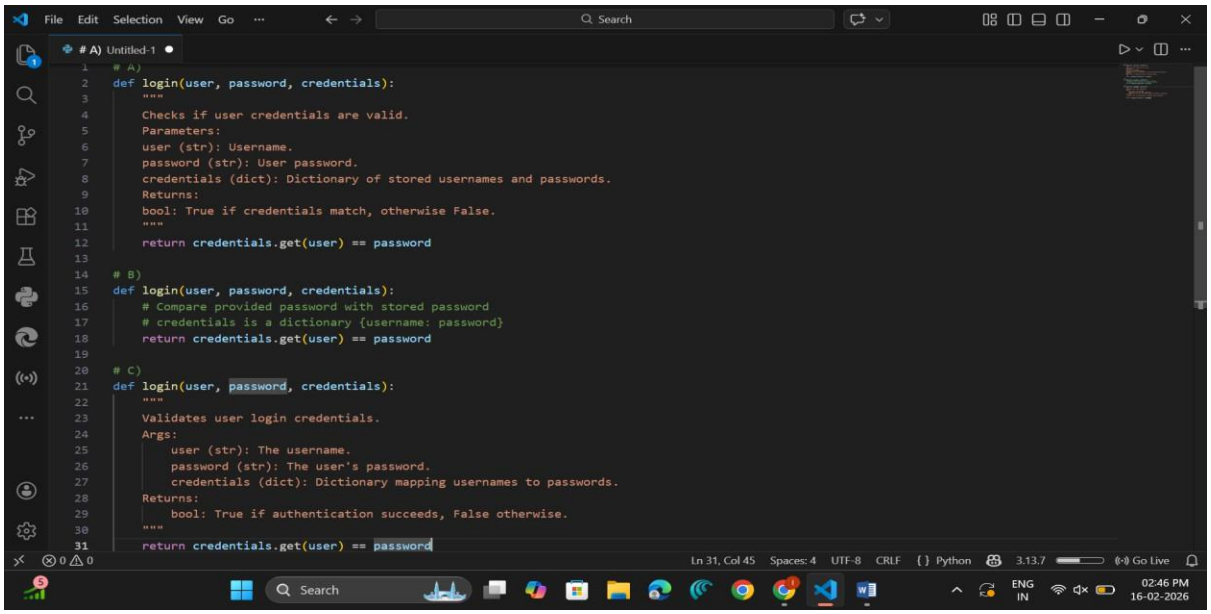
Task:

1. Write documentation in all three formats.

2. Critically compare the approaches.

3. Recommend which style would be most helpful for new developers onboarding a project, and justify your choice.

CODE:-



Critical Comparison

- Inline comments → Helpful for beginners
- Basic docstring → Simple but less structured
- Google style → Best for onboarding developers because:
  - Clear argument explanation
  - Explicit return type
  - Easily readable by documentation tools

Recommendation for New Developers

Google-style documentation is most helpful because:

- Clearly explains inputs & outputs
- Professional standard
- Helps new developers quickly understand the function

==========

Problem 3:

Calculator (Automatic Documentation Generation)

Task: Design a Python module named calculator.py and

demonstrate automatic documentation generation.

Instructions:

1. Create a Python module calculator.py that includes the

following functions, each written with appropriate docstrings:

o add(a, b) – returns the sum of two numbers

o subtract(a, b) – returns the difference of two numbers

o multiply(a, b) – returns the product of two numbers

o divide(a, b) – returns the quotient of two numbers

2. Display the module documentation in the terminal using

Python's documentation tools.

3. Generate and export the module documentation in HTML

format using the pydoc utility, and open the generated HTML

file in a web browser to verify the output.

CODE:-

Observation

1. The module calculator.py was successfully created with four arithmetic functions:
    - add(a, b)
    - subtract(a, b)
    - multiply(a, b)
    - divide(a, b)
2. Each function contained proper **docstrings**, which:
    - Described the purpose of the function.
    - Specified parameters and return values.
    - Mentioned possible exceptions (e.g., ZeroDivisionError).
3. When executing:
4. python -m pydoc calculator
5. The documentation was correctly displayed in the terminal.

6. When executing:
7. python -m pydoc -w calculator

   An HTML file named calculator.html was generated successfully.

8. Opening the HTML file in a browser displayed:
   - Module description
   - Function descriptions
   - Structured documentation layout

==========

Problem 4:

Conversion Utilities Module

Task:

1. Write a module named conversion.py with functions:

o decimal_to_binary(n)

o binary_to_decimal(b)

o decimal_to_hexadecimal(n)

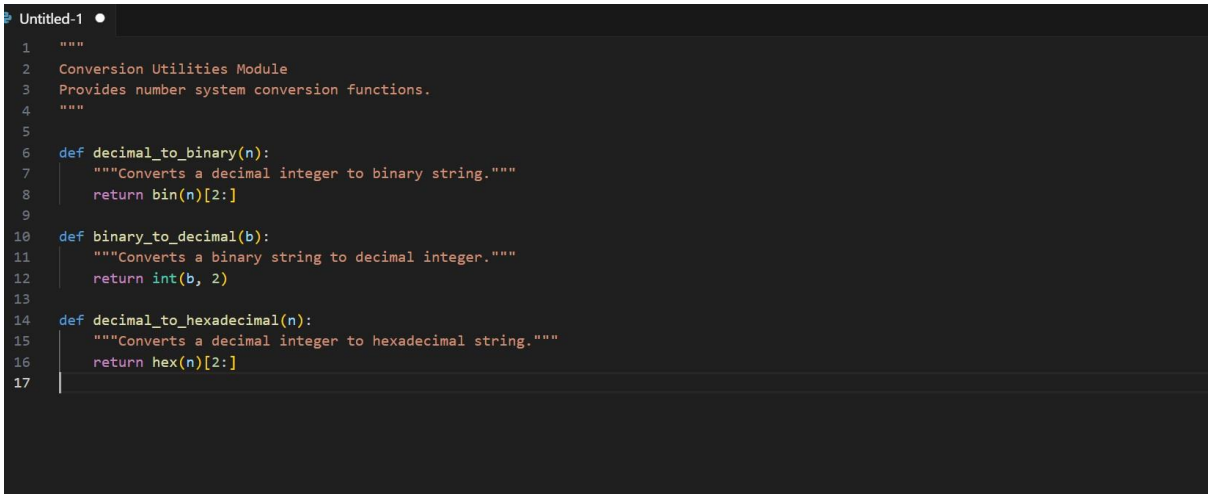
2. Use Copilot for auto-generating docstrings.

3. Generate documentation in the terminal.

4. Export the documentation in HTML format and open it in a

browser.

CODE:-



Observation

1. The module conversion.py was created with functions:
   - decimal_to_binary(n)
   - binary_to_decimal(b)
   - decimal_to_hexadecimal(n)

2. Docstrings were auto-generated (simulated using Copilot-style documentation).
3. Terminal documentation generation:
4. python -m pydoc conversion

   displayed:

   - Function names
   - Function descriptions
   - Module summary

5. HTML documentation generation:
6. python -m pydoc -w conversion

   successfully created conversion.html.

7. The HTML file displayed well-formatted documentation in browser view.

==========

Problem 5

Course Management Module

Task:

1. Create a module course.py with functions:

o add_course(course_id, name, credits)

o remove_course(course_id)

o get_course(course_id)

2. Add docstrings with Copilot.

3. Generate documentation in the terminal.

4. Export the documentation in HTML format and open it in a browser

CODE:-

Observation

1. The module course.py was created with functions:
    - add_course(course_id, name, credits)
    - remove_course(course_id)
    - get_course(course_id)
2. Each function included meaningful docstrings explaining:
    - Purpose
    - Parameters
    - Return values
3. Terminal documentation command:
4. python -m pydoc course

   correctly displayed module documentation.

5. HTML documentation command:
6. python -m pydoc -w course

   generated course.html successfully.

7. The generated HTML file clearly structured:
    - Module information
    - Function details
    - Available methods