

# AI Assisted Coding

## Assignment-9.1

K.Sai Rahul

2303A52392

## Batch-43

### TASK 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

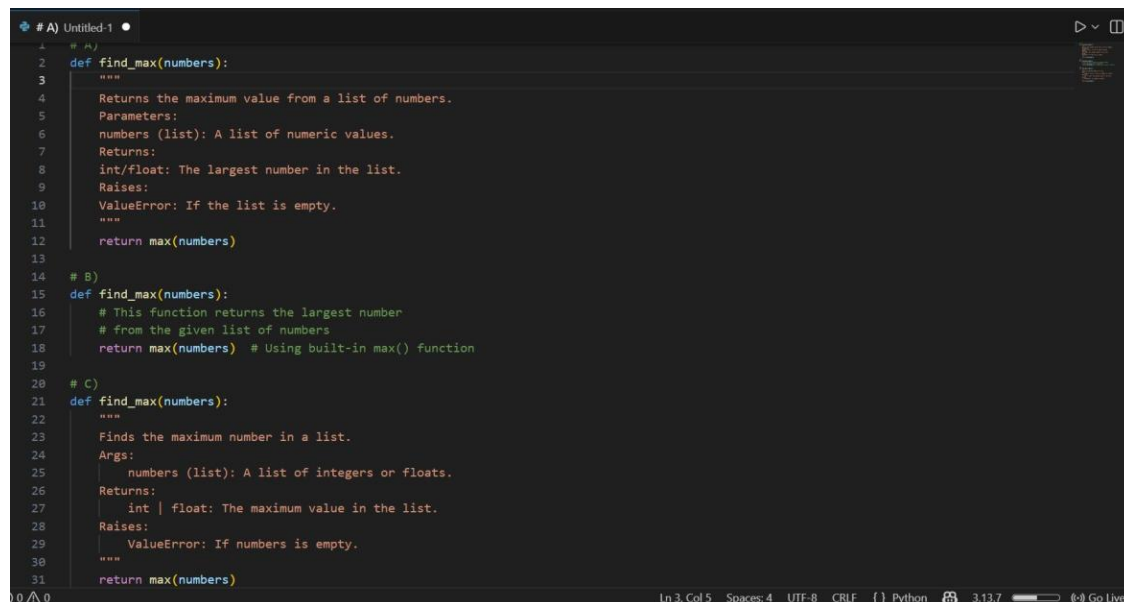
(b) Inline comments

(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:



```
1  # A) Untitled-1
2  def find_max(numbers):
3      """
4      Returns the maximum value from a list of numbers.
5      Parameters:
6      numbers (list): A list of numeric values.
7      Returns:
8      int/float: The largest number in the list.
9      Raises:
10     ValueError: If the list is empty.
11     """
12     return max(numbers)
13
14  # B)
15  def find_max(numbers):
16      # This function returns the largest number
17      # from the given list of numbers
18      return max(numbers) # Using built-in max() function
19
20  # C)
21  def find_max(numbers):
22      """
23      Finds the maximum number in a list.
24      Args:
25      | numbers (list): A list of integers or floats.
26      Returns:
27      | int | float: The maximum value in the list.
28      Raises:
29      | ValueError: If numbers is empty.
30      """
31      return max(numbers)
```

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

### TASK 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:

```
le Edit Selection View Go ... ← → Q Search
# A) Untitled-1
1 # A)
2 def login(user, password, credentials):
3     """
4     Checks if user credentials are valid.
5     Parameters:
6     user (str): Username.
7     password (str): User password.
8     credentials (dict): Dictionary of stored usernames and passwords.
9     Returns:
10    bool: True if credentials match, otherwise False.
11    """
12    return credentials.get(user) == password
13
14 # B)
15 def login(user, password, credentials):
16     # Compare provided password with stored password
17     # credentials is a dictionary {username: password}
18     return credentials.get(user) == password
19
20 # C)
21 def login(user, password, credentials):
22     """
23     Validates user login credentials.
24     Args:
25     user (str): The username.
26     password (str): The user's password.
27     credentials (dict): Dictionary mapping usernames to passwords.
28     Returns:
29     bool: True if authentication succeeds, False otherwise.
30     """
31    return credentials.get(user) == password
```

## 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

### TASK 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

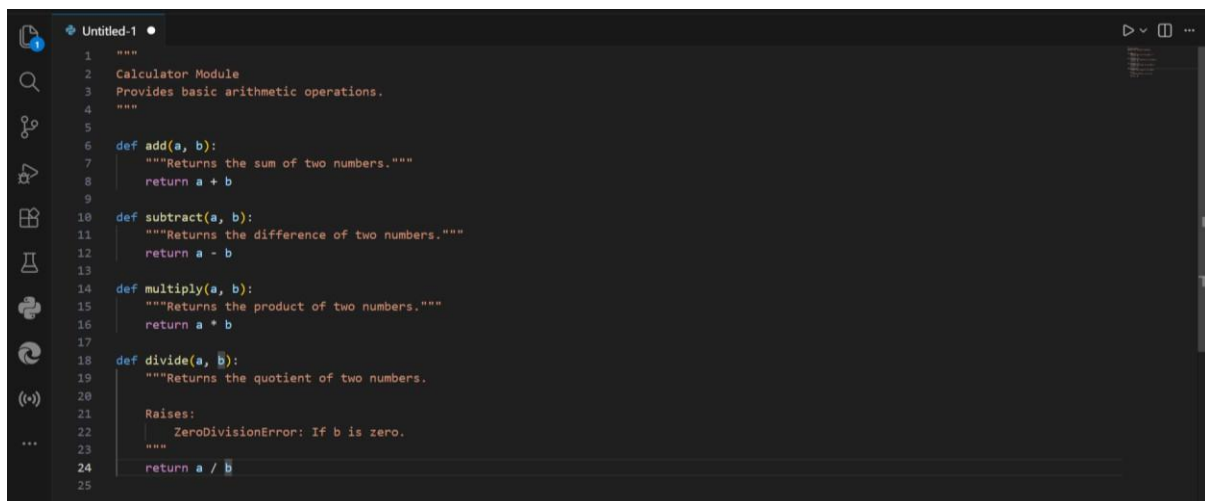
Python's documentation tools.

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:



```
1  """
2  Calculator Module
3  Provides basic arithmetic operations.
4  """
5
6  def add(a, b):
7      """Returns the sum of two numbers."""
8      return a + b
9
10 def subtract(a, b):
11     """Returns the difference of two numbers."""
12     return a - b
13
14 def multiply(a, b):
15     """Returns the product of two numbers."""
16     return a * b
17
18 def divide(a, b):
19     """Returns the quotient of two numbers.
20
21     Raises:
22         ZeroDivisionError: If b is zero.
23     """
24     return a / b
25
```

## Observation

1. The module `calculator.py` was successfully created with four arithmetic functions:
  - o `add(a, b)`
  - o `subtract(a, b)`
  - o `multiply(a, b)`
  - o `divide(a, b)`
2. Each function contained proper **docstrings**, which:
  - o Described the purpose of the function.
  - o Specified parameters and return values.
  - o 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:

○ `decimal_to_binary(n)`

○ `binary_to_decimal(b)`

○ `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:

```
Untitled-1
1  """
2  Conversion Utilities Module
3  Provides number system conversion functions.
4  """
5
6  def decimal_to_binary(n):
7      """Converts a decimal integer to binary string."""
8      return bin(n)[2:]
9
10 def binary_to_decimal(b):
11     """Converts a binary string to decimal integer."""
12     return int(b, 2)
13
14 def decimal_to_hexadecimal(n):
15     """Converts a decimal integer to hexadecimal string."""
16     return hex(n)[2:]
17
```

## 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.

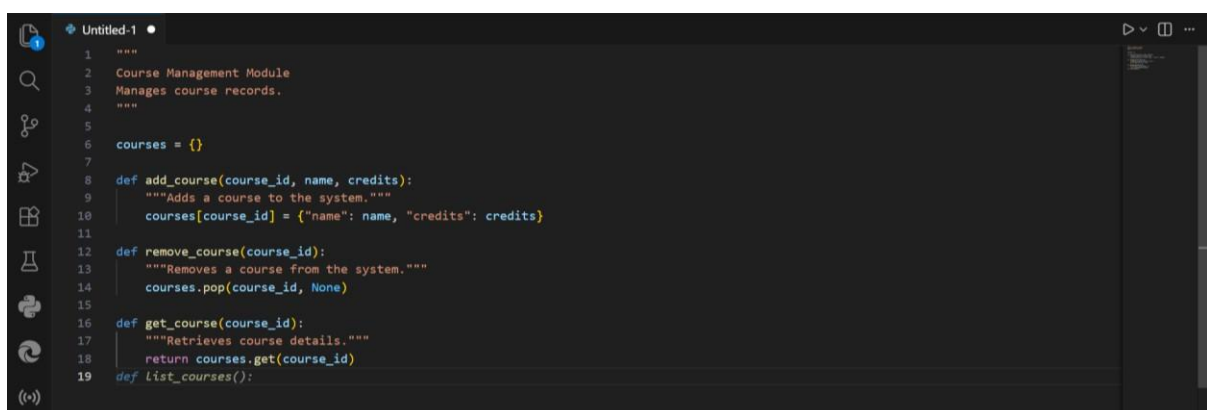
## TASK 5:

### 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:



```
1  """
2  Course Management Module
3  Manages course records.
4  """
5
6  courses = {}
7
8  def add_course(course_id, name, credits):
9      """Adds a course to the system."""
10     courses[course_id] = {"name": name, "credits": credits}
11
12  def remove_course(course_id):
13      """Removes a course from the system."""
14     courses.pop(course_id, None)
15
16  def get_course(course_id):
17      """Retrieves course details."""
18     return courses.get(course_id)
19  def list_courses():
```

# 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