

Lab Assignment 9.1

HT.No: 2303A51109

Name: M.Hasini

Batch-02

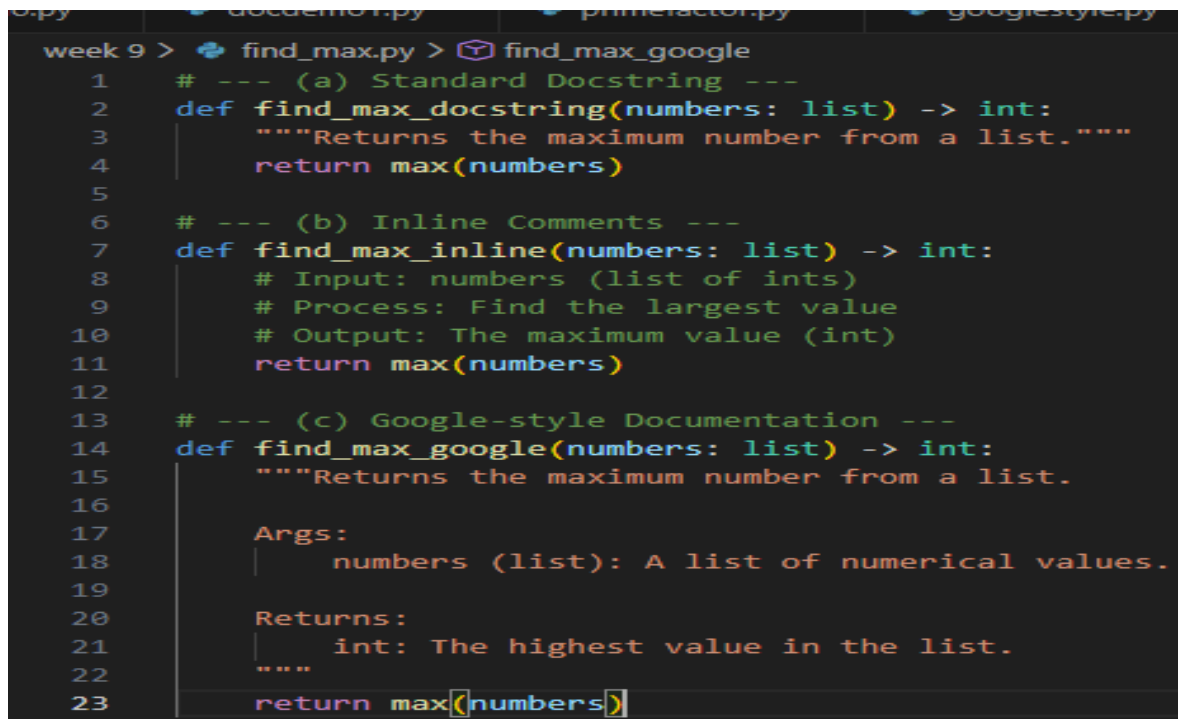
Problem 1:

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.



```
week 9 > find_max.py > find_max_google  
1  # --- (a) Standard Docstring ---  
2  def find_max_docstring(numbers: list) -> int:  
3      """Returns the maximum number from a list."""  
4      return max(numbers)  
5  
6  # --- (b) Inline Comments ---  
7  def find_max_inline(numbers: list) -> int:  
8      # Input: numbers (list of ints)  
9      # Process: Find the largest value  
10     # Output: The maximum value (int)  
11     return max(numbers)  
12  
13 # --- (c) Google-style Documentation ---  
14 def find_max_google(numbers: list) -> int:  
15     """Returns the maximum number from a list.  
16  
17     Args:  
18         numbers (list): A list of numerical values.  
19  
20     Returns:  
21         int: The highest value in the list.  
22     """  
23     return max(numbers)
```

Style	Advantages	Disadvantages	Use Case
Docstring	Easy to read, built-in Python support	Less structured	Small to medium projects
Inline Comments	Very simple	Not suitable for functions	Explaining logic
Google-Style	Professional, standardized	Slightly longer	Libraries & APIs

Google-style documentation is most effective for a **mathematical utilities library** because it is clear, structured, and widely used in professional projects

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

```

week 9 > login.py > login_google
1  # --- (a) Standard Docstring ---
2  def login_docstring(user: str, password: str, credentials: dict) -> bool:
3      """Validates user credentials."""
4      return credentials.get(user) == password
5
6  # --- (b) Inline Comments ---
7  def login_inline(user: str, password: str, credentials: dict) -> bool:
8      # user: str, username
9      # password: str, user password
10     # credentials: dict, stored user data
11     # Returns: bool, True if match else False
12     return credentials.get(user) == password
13
14  # --- (c) Google-style Documentation ---
15  def login_google(user: str, password: str, credentials: dict) -> bool:
16      """Validates user credentials against a dictionary.
17
18      Args:
19          user (str): The username to verify.
20          password (str): The password provided.
21          credentials (dict): Dictionary mapping users to passwords.
22
23      Returns:
24          bool: True if login is successful, False otherwise.
25      """
26      return credentials.get(user) == password

```

Google-style documentation is best for **new developers onboarding** because it clearly explain parameters, return values, and usage.

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.

```
PS C:\Users\User\OneDrive\Desktop\AIAC> cd "week 9"
PS C:\Users\User\OneDrive\Desktop\AIAC\week 9> python -m pydoc -w calculator
wrote calculator.html
PS C:\Users\User\OneDrive\Desktop\AIAC\week 9> python -m pydoc -p 1234
Server ready at http://localhost:1234/
Server commands: [b]rowser, [q]uit
server> b
server> 
```

C:\Users\User\OneDrive\Desktop\AIAC\week 9

[calculator](#)
[docdemo](#)

[docdemo1](#)
[find_max](#)

[googlestyle](#)
[login](#)

[primefacto](#)
[pydocdem](#)

week 9 > calculator.py > divide

```
1  """Calculator module providing basic arithmetic operations."""
2
3  def add(a: float, b: float) -> float:
4      """Returns the sum of two numbers.
5
6      Args:
7          a (float): The first number.
8          b (float): The second number.
9
10     Returns:
11         float: The sum of a and b.
12     """
13     return a + b
14
15  def subtract(a: float, b: float) -> float:
16      """Returns the difference of two numbers.
17
18     Args:
19         a (float): The first number.
20         b (float): The second number.
21
22     Returns:
23         float: The difference of a and b.
24     """
25     return a - b
26
27  def multiply(a: float, b: float) -> float:
28      """Returns the product of two numbers.
29
30     Args:
31         a (float): The first number.
32         b (float): The second number.
```

week 9 > calculator.py > divide

```
25     return a - b
26
27 def multiply(a: float, b: float) -> float:
28     """Returns the product of two numbers.
29
30     Args:
31         a (float): The first number.
32         b (float): The second number.
33
34     Returns:
35         float: The product of a and b.
36     """
37     return a * b
38
39 def divide(a: float, b: float) -> float:
40     """Returns the quotient of two numbers.
41
42     Args:
43         a (float): The numerator.
44         b (float): The denominator.
45
46     Returns:
47         float: The quotient of a and b.
48
49     Raises:
50         ValueError: If b is zero.
51     """
52     if b == 0:
53         raise ValueError("Cannot divide by zero.")
54     return a / b
```

```
IAC/week 9/calculator.py"
```

```
PS C:\Users\User\OneDrive\Desktop\AIAC> cd "week 9"
```

```
PS C:\Users\User\OneDrive\Desktop\AIAC\week 9> python -m pydoc calculator
```

```
Help on module calculator:
```

NAME

```
calculator - Calculator module providing basic arithmetic operations.
```

FUNCTIONS

```
add(a: float, b: float) -> float
```

```
Returns the sum of two numbers.
```

Args:

```
a (float): The first number.
```

```
b (float): The second number.
```

Returns:

```
float: The sum of a and b.
```

```
divide(a: float, b: float) -> float
```

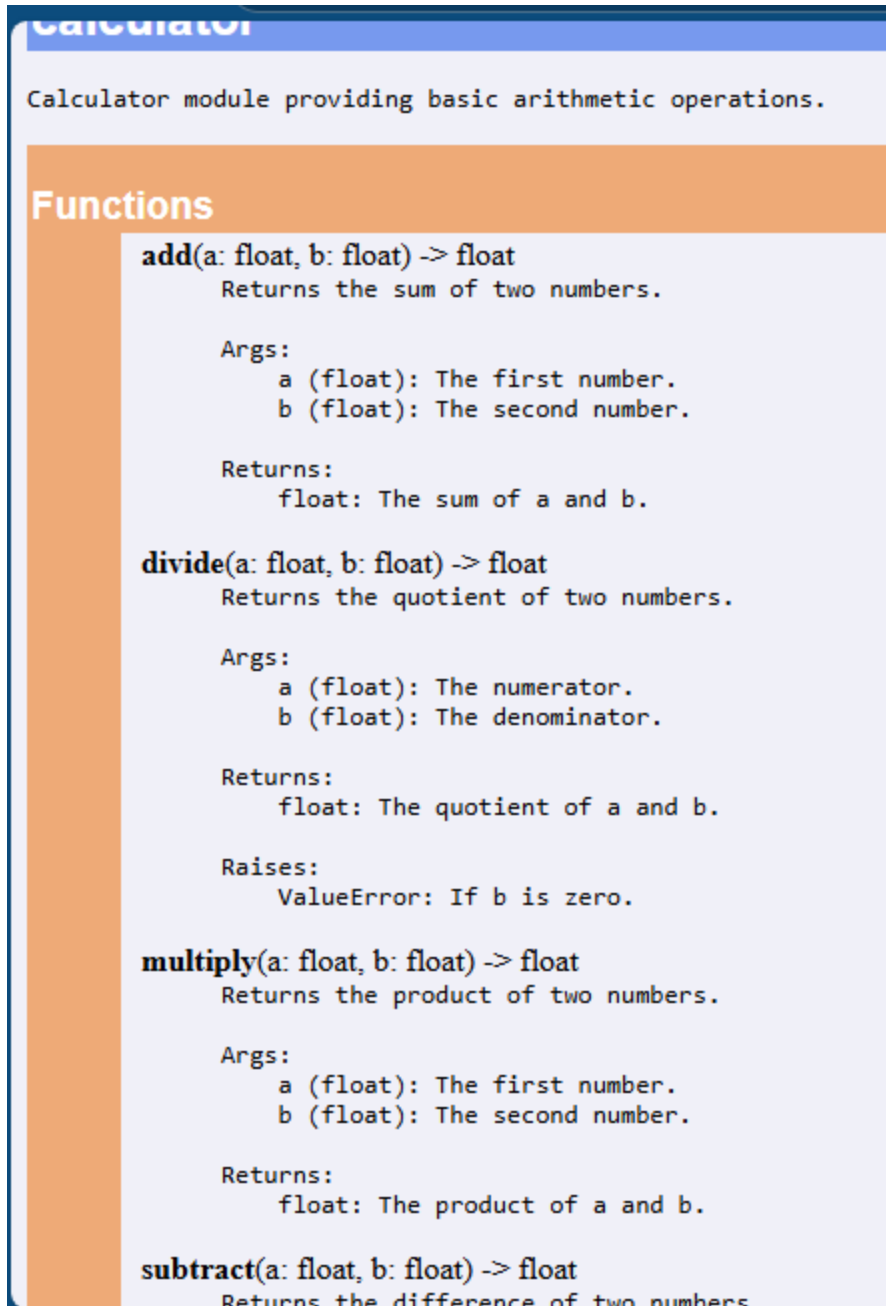
```
Returns the quotient of two numbers.
```

Args:

```
a (float): The numerator.
```

```
b (float): The denominator.
```

```
-- More --
```



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.

```
week 9 > 📁 conversion.py > 📄 decimal_to_hexadecimal
1  """Conversion utilities for number systems."""
2
3  def decimal_to_binary(n: int) -> str:
4      """Converts a decimal number to binary string.
5
6      Args:
7          n (int): The decimal number.
8
9      Returns:
10         str: The binary representation.
11     """
12     return bin(n)[2:]
13
14 def binary_to_decimal(b: str) -> int:
15     """Converts a binary string to decimal number.
16
17     Args:
18         b (str): The binary string.
19
20     Returns:
21         int: The decimal representation.
22     """
23     return int(b, 2)
24
25 def decimal_to_hexadecimal(n: int) -> str:
26     """Converts a decimal number to hexadecimal string.
27
28     Args:
29         n (int): The decimal number.
30
31     Returns:
32         str: The hexadecimal representation.
```

```
Help on module conversion:

NAME
  conversion - Conversion utilities for number systems.

FUNCTIONS
  binary_to_decimal(b: str) -> int
    Converts a binary string to decimal number.

    Args:
      b (str): The binary string.

    Returns:
      int: The decimal representation.

  decimal_to_binary(n: int) -> str
    Converts a decimal number to binary string.

    Args:
      n (int): The decimal number.
-- More --
```

```
IAC/week 9/conversion.py"
PS C:\Users\User\OneDrive\Desktop\AIAC> cd "week 9"
PS C:\Users\User\OneDrive\Desktop\AIAC\week 9> python -m pydoc -w conversion
wrote conversion.html
PS C:\Users\User\OneDrive\Desktop\AIAC\week 9> python -m pydoc -p 1234
Server ready at http://localhost:1234/
Server commands: [b]rowser, [q]uit
server> b
server>
```

conversion

Conversion utilities for number systems.

Functions

```
binary_to_decimal(b: str) -> int
    Converts a binary string to decimal number.

    Args:
        b (str): The binary string.

    Returns:
        int: The decimal representation.

decimal_to_binary(n: int) -> str
    Converts a decimal number to binary string.

    Args:
        n (int): The decimal number.

    Returns:
        str: The binary representation.

decimal_to_hexadecimal(n: int) -> str
    Converts a decimal number to hexadecimal string.

    Args:
        n (int): The decimal number.

    Returns:
        str: The hexadecimal representation.
```

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

```

Help on module course:

NAME
  course - Course management system for handling course data.

FUNCTIONS
  __annotate__(format, /)

  add_course(course_id: str, name: str, credits: int) -> None
    Adds a new course to the system.

    Args:
      course_id (str): Unique identifier for the course.
      name (str): Name of the course.
      credits (int): Number of credits for the course.

  get_course(course_id: str) -> dict
    Retrieves course details by ID.

    Args:
  -- More --

```

```

PS C:\Users\User\OneDrive\Desktop\AIAC\week 9> python -m pydoc -p 1234
Server ready at http://localhost:1234/
Server commands: [b]rowser, [q]uit
server> b
server>

```

```

use help(str) for help on the str class.
PS C:\Users\User\OneDrive\Desktop\AIAC> cd "week 9"
wrote course.html
PS C:\Users\User\OneDrive\Desktop\AIAC\week 9> pydoc -p 1234

```

course

Course management system for handling course data.

Functions

__annotate__(format, /)

add_course(course_id: str, name: str, credits: int) -> None

Adds a new course to the system.

Args:

course_id (str): Unique identifier for the course.

name (str): Name of the course.

credits (int): Number of credits for the course.

get_course(course_id: str) -> dict

Retrieves course details by ID.

Args:

course_id (str): Unique identifier for the course.

Returns:

dict: Course details or None if not found.

remove_course(course_id: str) -> None

Removes a course from the system by ID.

Args:

course_id (str): Unique identifier for the course.