

AI ASSISTED CODING

ASSIGNMENT - 9.1

ROLL NO. 2303A510J0

Problem 1:

Consider the following Python function:

```
def find_max(numbers):  
    return max(numbers)
```

Task:

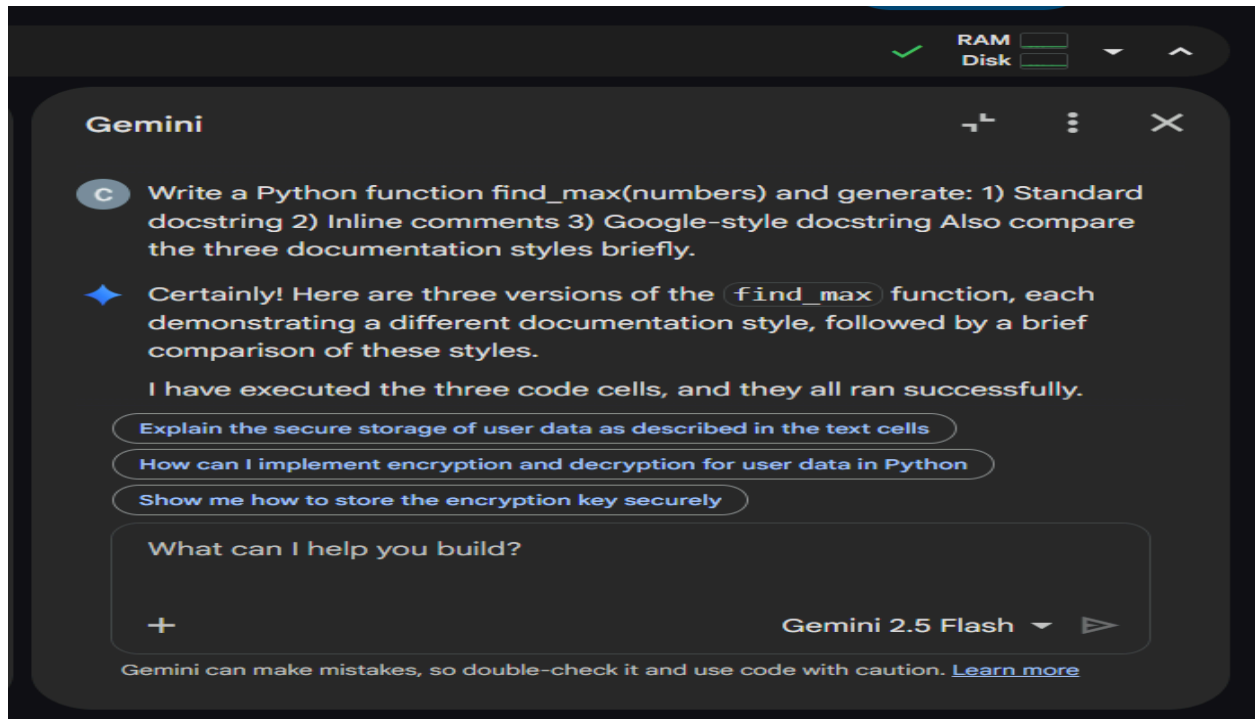
- Write documentation for the function in all three formats:
 - (a) Docstring
 - (b) Inline comments
 - (c) Google-style documentation
- Critically compare the three approaches. Discuss the advantages, disadvantages, and suitable use cases of each style.
- Recommend which documentation style is most effective for a mathematical utilities library and justify your answer.

Initial code:

```
def find_max(numbers):  
    return max(numbers)  
  
print(find_max([1, 3, 7, 9, 2]))
```

Prompt:

Write a Python function `find_max(numbers)` and generate: 1) Standard docstring 2) Inline comments 3) Google-style docstring Also compare the three documentation styles briefly.



Code:

```
def find_max_standard_docstring(numbers):  
    """  
    Finds the maximum number in a list of numbers.  
  
    Args:  
        numbers (list): A list of numerical values.  
  
    Returns:  
        int or float: The maximum value in the list.  
    """  
    if not numbers:  
        raise ValueError("Input list cannot be empty")  
    return max(numbers)  
  
print("Standard Docstring Example:", find_max_standard_docstring([1, 5, 2,  
9, 3]))
```

```
def find_max_inline_comments(numbers):
    # Checks if the input list is empty
    if not numbers:
        # Raises an error if the list is empty
        raise ValueError("Input list cannot be empty")
    # Returns the maximum value from the list using the built-in max()
    function
    return max(numbers)

print("Inline Comments Example:", find_max_inline_comments([10, 4, 7, 12, 1]))
```

```
def find_max_google_docstring(numbers):
    """
    Finds the maximum number in a list of numbers.

    Args:
        numbers (list): A list of numerical values.

    Returns:
        int or float: The maximum value in the list.

    Raises:
        ValueError: If the input list is empty.

    Example:
        >>> find_max_google_docstring([1, 5, 2, 9, 3])
        9
        >>> find_max_google_docstring([])
        ValueError: Input list cannot be empty
    """
    if not numbers:
        raise ValueError("Input list cannot be empty")
    return max(numbers)

print("Google-style Docstring Example:", find_max_google_docstring([20, 15, 25, 18]))
```

Output:

```
Standard Docstring Example: 9
Inline Comments Example: 12
Google-style Docstring Example: 25
```

Recommendation – Mathematical Utilities Library

Use Google-style docstrings

Justification:

Mathematical libraries are reused widely and often auto-documented. Google-style docstrings provide structured parameters and return descriptions that integrate well with documentation tools and improve readability.

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.

Initial code:

```
def login(user, password, credentials):
    return credentials.get(user) == password
```

Prompt:

Document a Python login() function using docstring, inline comments, and Google-style format. Compare which helps new developers most.

Code:

```

def login_standard_docstring(user, password, credentials):
    """
    Authenticates a user by checking their provided password against
    stored credentials.

    Args:
        user (str): The username attempting to log in.
        password (str): The password provided by the user.
        credentials (dict): A dictionary where keys are usernames (str)
        and values are their passwords (str).

    Returns:
        bool: True if the user's password matches the stored credentials,
        False otherwise.
    """
    return credentials.get(user) == password

# Example usage:
user_credentials = {'admin': 'password123', 'guest': 'guestpass'}
print("Standard Docstring Login (admin, correct pass):",
login_standard_docstring('admin', 'password123', user_credentials))
print("Standard Docstring Login (admin, wrong pass):",
login_standard_docstring('admin', 'wrongpass', user_credentials))

def login_inline_comments(user, password, credentials):
    # Retrieve the stored password for the given user from the credentials
    dictionary.
    # If the user does not exist, credentials.get(user) will return None.
    stored_password = credentials.get(user)

    # Compare the provided password with the stored password.
    # If both are valid and match, authentication is successful.
    return stored_password == password

# Example usage:
user_credentials = {'admin': 'password123', 'guest': 'guestpass'}
print("Inline Comments Login (guest, correct pass):",
login_inline_comments('guest', 'guestpass', user_credentials))
print("Inline Comments Login (guest, wrong pass):",
login_inline_comments('guest', 'badpass', user_credentials))

```

```

def login_google_docstring(user: str, password: str, credentials: dict) ->
bool:
    """
    Authenticates a user against a dictionary of stored credentials.

    This function checks if the provided username exists in the
    `credentials` dictionary
    and if the `password` matches the stored password for that user.

    Args:
        user: The username attempting to log in.
        password: The password provided by the user.
        credentials: A dictionary mapping usernames (str) to their
passwords (str).

    Returns:
        True if the username exists and the password matches, False
otherwise.

    Example:
        >>> user_creds = {'testuser': 'secretpass'}
        >>> login_google_docstring('testuser', 'secretpass', user_creds)
        True
        >>> login_google_docstring('testuser', 'wrongpass', user_creds)
        False
        >>> login_google_docstring('nonexistent', 'anypass', user_creds)
        False
    """
    return credentials.get(user) == password

# Example usage:
user_credentials = {'manager': 'securepwd', 'developer': 'devpass'}
print("Google-style Docstring Login (manager, correct pass):",
login_google_docstring('manager', 'securepwd', user_credentials))
print("Google-style Docstring Login (developer, wrong pass):",
login_google_docstring('developer', 'wrongpass', user_credentials))

```

Output:

```
Standard Docstring Login (admin, correct pass): True
Standard Docstring Login (admin, wrong pass): False
Inline Comments Login (guest, correct pass): True
Inline Comments Login (guest, wrong pass): False
Google-style Docstring Login (manager, correct pass): True
Google-style Docstring Login (developer, wrong pass): False
```

Recommendation – For New Developers

Best: Google-style docstrings

Justification to write:

New developers benefit from structured parameters and return descriptions. Google-style docstrings make onboarding easier because expectations and data types are clearly defined.

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

Prompt:

Create calculator.py module with add, subtract, multiply, divide functions using proper docstrings compatible with pydoc.

Code:

```
"""
Calculator module providing basic arithmetic operations.

This module contains functions for performing addition, subtraction,
multiplication, and division operations.
"""

def add(a, b):
    """
    Add two numbers.

    Args:
        a (float): The first number.
        b (float): The second number.

    Returns:
        float: The sum of a and b.
    """
    return a + b

def subtract(a, b):
    """
    Subtract two numbers.

    Args:
        a (float): The first number.
        b (float): The second number.

    Returns:
        float: The difference of a and b.
    """
```



```

    return a - b

def multiply(a, b):
    """
    Multiply two numbers.

    Args:
        a (float): The first number.
        b (float): The second number.

    Returns:
        float: The product of a and b.
    """
    return a * b

def divide(a, b):
    """
    Divide two numbers.

    Args:
        a (float): The dividend.
        b (float): The divisor (must not be zero).

    Returns:
        float: The quotient of a divided by b.

    Raises:
        ValueError: If b is zero.
    """
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b

# Example usage:
if __name__ == "__main__":
    num1 = 10
    num2 = 5

```

```
print(f"Addition: {num1} + {num2} = {add(num1, num2)}")
print(f"Subtraction: {num1} - {num2} = {subtract(num1, num2)}")
print(f"Multiplication: {num1} * {num2} = {multiply(num1, num2)}")
print(f"Division: {num1} / {num2} = {divide(num1, num2)}")
```

Output:

```
Addition: 10 + 5 = 15
Subtraction: 10 - 5 = 5
Multiplication: 10 * 5 = 50
Division: 10 / 5 = 2.0
```

Command:

```
python -m pydoc calculator
```

Output:

```
PS C:\Users\rohit\OneDrive\Documents\SRU\ai_code\lab_9.1> python
-m pydoc calculator
>>
Help on module calculator:

NAME
    calculator

FUNCTIONS
    add(a, b)
        Add two numbers.

        Args:
            a: First number
            b: Second number

        Returns:
            The sum of a and b

    divide(a, b)
        Divide two numbers.
```

Args:

a: First number (dividend)

b: Second number (divisor)

Returns:

The quotient of a divided by b

Raises:

ZeroDivisionError: If b is zero

`multiply(a, b)`

Multiply two numbers.

Args:

a: First number

b: Second number

Returns:

The product of a and b

`subtract(a, b)`

Subtract two numbers.

Args:

a: First number

b: Second number

Returns:

The difference of a and b

FILE

c:\users\rohit\onedrive\documents\sru\ai_code\lab_9.1\calculator.py

Command for html export:

python -m pydoc -w calculator

```
PS C:\Users\rohit\OneDrive\Documents\SRU\ai_code\lab_9.1> python
-m pydoc -w calculator
>>
wrote calculator.html
```

Output (calculator.html):

[index](#)
calculator c:\users\rohit\onedrive\documents\srui_code\lab_9.1\calculator.py

Functions

add(a, b)
Add two numbers.

Args:
a: First number
b: Second number

Returns:
The sum of a and b

divide(a, b)
Divide two numbers.

Args:
a: First number (dividend)
b: Second number (divisor)

Returns:
The quotient of a divided by b

Raises:
ZeroDivisionError: If b is zero

multiply(a, b)
Multiply two numbers.

Args:
a: First number
b: Second number

Returns:
The product of a and b

subtract(a, b)
Subtract two numbers.

Args:
a: First number
b: Second number

Returns:
The difference of a and b

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

Prompt:

Generate `conversion.py` module with `decimal_to_binary`, `binary_to_decimal`, `decimal_to_hexadecimal` functions and detailed docstrings.

Code:

```
"""
Conversion module for number system conversions.

This module provides functions to convert between different number systems
including decimal, binary, and hexadecimal representations.
"""

def decimal_to_binary(decimal_num):
    """
    Convert a decimal number to its binary representation.

    Args:
        decimal_num (int): A non-negative integer to convert to binary.

    Returns:
        str: The binary representation of the decimal number (without '0b'
        prefix).

    Raises:
        ValueError: If the input is negative.
    """
```

Example:

```
>>> decimal_to_binary(10)
```

```
'1010'
```

```
>>> decimal_to_binary(255)
```

```
'11111111'
```

```
"""
```

```
if decimal_num < 0:
```

```
    raise ValueError("Input must be a non-negative integer")
```

```
return bin(decimal_num)[2:]
```

```
def binary_to_decimal(binary_str):
```

```
    """
```

```
    Convert a binary string to its decimal representation.
```

Args:

```
    binary_str (str): A binary number as a string (e.g., '1010').
```

Returns:

```
    int: The decimal equivalent of the binary number.
```

Raises:

```
    ValueError: If the input contains non-binary characters.
```

Example:

```
>>> binary_to_decimal('1010')
```

```
10
```

```
>>> binary_to_decimal('11111111')
```

```
255
```

```
"""
```

```
try:
```

```
    return int(binary_str, 2)
```

```
except ValueError:
```

```
    raise ValueError("Input must contain only binary digits (0 and 1)")
```

```
def decimal_to_hexadecimal(decimal_num):
```

```
    """
```

```

Convert a decimal number to its hexadecimal representation.

Args:
    decimal_num (int): A non-negative integer to convert to hexadecimal.

Returns:
    str: The hexadecimal representation of the decimal number (without '0x'
prefix).

Raises:
    ValueError: If the input is negative.

Example:
    >>> decimal_to_hexadecimal(255)
    'ff'
    >>> decimal_to_hexadecimal(4095)
    'fff'
    """
    if decimal_num < 0:
        raise ValueError("Input must be a non-negative integer")
    return hex(decimal_num)[2:]

# Example usage:
if __name__ == "__main__":
    print(decimal_to_binary(10)) # Output: '1010'
    print(binary_to_decimal('1010')) # Output: 10
    print(decimal_to_hexadecimal(255)) # Output: 'ff'

```

Output:

```

1010
10
ff

```

Command:

```
python -m pydoc conversion
```

Output:

```
PS C:\Users\rohit\OneDrive\Documents\SRU\ai_code\lab_9.1> python
-m pydoc conversion
Help on module conversion:
```

NAME

conversion - Conversion module for number system conversions.

DESCRIPTION

This module provides functions to convert between different number systems including decimal, binary, and hexadecimal representations.

FUNCTIONS

binary_to_decimal(binary_str)

Convert a binary string to its decimal representation.

Args:

binary_str (str): A binary number as a string (e.g., '1010').

Returns:

int: The decimal equivalent of the binary number.

Raises:

ValueError: If the input contains non-binary characters.

Example:

```
>>> binary_to_decimal('1010')
10
>>> binary_to_decimal('11111111')
255
```

decimal_to_binary(decimal_num)

Convert a decimal number to its binary representation.

Args:

`decimal_num (int)`: A non-negative **integer** to convert to binary.

Returns:

`str`: The binary representation of the decimal number (without `'0b'` prefix).

Raises:

`ValueError`: If the input is negative.

Example:

```
>>> decimal_to_binary(10)
'1010'
>>> decimal_to_binary(255)
'11111111'
```

`decimal_to_hexadecimal(decimal_num)`

Convert a decimal number to its hexadecimal representation.

Args:

`decimal_num (int)`: A non-negative **integer** to convert to hexadecimal.

Returns:

`str`: The hexadecimal representation of the decimal number (without `'0x'` prefix).

Raises:

`ValueError`: If the input is negative.

Example:

```
>>> decimal_to_hexadecimal(255)
'ff'
>>> decimal_to_hexadecimal(4095)
'fff'
```

FILE

c:\users\rohit\onedrive\documents\sru\ai_code\lab_9.1\conversion.py

Command for html version:
Python -m pydoc -w conversion

Output(conversion.html):

[index](#)
conversion [c:\users\rohit\onedrive\documents\sru\ai_code\lab_9.1\conversion.py](#)

Conversion module for number system conversions.

This module provides functions to convert between different number systems including decimal, binary, and hexadecimal representations.

Functions

binary_to_decimal(binary_str)
Convert a binary string to its decimal representation.

Args:
binary_str (str): A binary number as a string (e.g., '1010').

Returns:
int: The decimal equivalent of the binary number.

Raises:
ValueError: If the input contains non-binary characters.

Example:
>>> [binary_to_decimal](#)('1010')
10
>>> [binary_to_decimal](#)('11111111')
255

decimal_to_binary(decimal_num)
Convert a decimal number to its binary representation.

Args:
decimal_num (int): A non-negative integer to convert to binary.

Returns:
str: The binary representation of the decimal number (without '0b' prefix).

Raises:
ValueError: If the input is negative.

Example:
>>> [decimal_to_binary](#)(10)
'1010'
>>> [decimal_to_binary](#)(255)
'11111111'

decimal_to_hexadecimal(decimal_num)
Convert a decimal number to its hexadecimal representation.

Args:
decimal_num (int): A non-negative integer to convert to hexadecimal.

Returns:
str: The hexadecimal representation of the decimal number (without '0x' prefix).

Raises:
ValueError: If the input is negative.

Example:
>>> [decimal_to_hexadecimal](#)(255)
'ff'
>>> [decimal_to_hexadecimal](#)(4095)
'fff'

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.

Prompt:

Create `course.py` module with `add_course`, `remove_course`, `get_course` functions using dictionary storage and proper docstrings.

Code:

```
"""
Course management module for storing and retrieving course information.
"""

# Dictionary to store courses with course_id as key
courses = {}

def add_course(course_id, course_name, credits):
    """
    Add a new course to the storage.

    Args:
        course_id (str): Unique identifier for the course
        course_name (str): Name of the course
        credits (int): Number of credit hours

    Returns:
        bool: True if course added successfully, False if course_id already exists
    """
```

```

    if course_id in courses:
        return False
    courses[course_id] = {
        "name": course_name,
        "credits": credits
    }
    return True

def remove_course(course_id):
    """
    Remove a course from the storage.

    Args:
        course_id (str): Unique identifier for the course to remove

    Returns:
        bool: True if course removed successfully, False if course_id not found
    """
    if course_id in courses:
        del courses[course_id]
        return True
    return False

def get_course(course_id):
    """
    Retrieve course information by course_id.

    Args:
        course_id (str): Unique identifier for the course

    Returns:
        dict: Course information with 'name' and 'credits' keys, or None if not
found
    """
    return courses.get(course_id)

# Example usage:

```

```

if __name__ == "__main__":
    add_course("CS101", "Introduction to Computer Science", 4)
    add_course("MATH201", "Calculus I", 3)

    print(get_course("CS101")) # Output: {'name': 'Introduction to Computer
Science', 'credits': 4}
    print(get_course("MATH201")) # Output: {'name': 'Calculus I', 'credits': 3}

    remove_course("CS101")
    print(get_course("CS101")) # Output: None

```

Output:

```

{'name': 'Introduction to Computer Science', 'credits': 4}
{'name': 'Calculus I', 'credits': 3}
None

```

Command for documentation in terminal:

`python -m pydoc course`

Output:

```

PS C:\Users\rohit\OneDrive\Documents\SRU\ai_code\lab_9.1> python
-m pydoc course
Help on module course:

NAME
    course - Course management module for storing and retrieving
    course information.

FUNCTIONS
    add_course(course_id, course_name, credits)
        Add a new course to the storage.

    Args:
        course_id (str): Unique identifier for the course
        course_name (str): Name of the course
        credits (int): Number of credit hours

```

```
Returns:
    bool: True if course added successfully, False if
course_id already exists

get_course(course_id)
    Retrieve course information by course_id.

Args:
    course_id (str): Unique identifier for the course

Returns:
    dict: Course information with 'name' and 'credits'
keys, or None if not found

remove_course(course_id)
    Remove a course from the storage.

Args:
    course_id (str): Unique identifier for the course to
remove

Returns:
    bool: True if course removed successfully, False if
course_id not found

DATA
    courses = {}

FILE

c:\users\rohit\onedrive\documents\sru\ai_code\lab_9.1\course.py
```

Command for HTML version:
`python -m pydoc -w course`

```
PS C:\Users\rohit\OneDrive\Documents\SRU\ai_code\lab_9.1> python
-m pydoc -w course
wrote course.html
```

Output(course.html):

[index](#)

course c:\users\rohit\onedrive\documents\sru\ai_code\lab_9.1\course.py

Course management module for storing and retrieving course information.

Functions

add_course(course_id, course_name, credits)

Add a new course to the storage.

Args:

course_id (str): Unique identifier for the course

course_name (str): Name of the course

credits (int): Number of credit hours

Returns:

bool: True if course added successfully, False if course_id already exists

get_course(course_id)

Retrieve course information by course_id.

Args:

course_id (str): Unique identifier for the course

Returns:

dict: Course information with 'name' and 'credits' keys, or None if not found

remove_course(course_id)

Remove a course from the storage.

Args:

course_id (str): Unique identifier for the course to remove

Returns:

bool: True if course removed successfully, False if course_id not found

Data

courses = {}