

AI Assistant Coding

Assignment 9.5

Name : P.Nagasri

HT. No : 2303A52456

Batch: 32

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:

```
def find_max(numbers: list) -> float:  
    """Find the maximum number in a list of numbers.  
    Args:  
        numbers (list): A list of numbers.  
    Returns:  
        The maximum number in the list.  
    """  
  
    if not numbers: # Check if the list is empty  
        raise ValueError("The list is empty.")  
    return max(numbers) # Use the built-in max function to find the maximum  
number
```

Advantages and Disadvantages:

Feature	Docstrings (PEP 257)	Inline Comments	Google-Style
Primary Audience	Users of the function	Maintainers of the code	Both Users and Maintainers
Advantages	Simple, fast to write, accessible via help().	Explains <i>why</i> a specific line exists; great for complex logic.	Highly structured, easy to read, supports auto-generation of docs.
Disadvantages	Lacks detail on types or specific error handling.	Can clutter code; doesn't show up in help() or IDE tooltips.	Verbose; can take up more space than the code itself.
Best Use Case	Small scripts or internal helper functions.	Explaining "hacks," workarounds, or complex algorithms.	Public APIs, libraries, and large-scale team projects.

Recommendation: Google Style is best one to use.

In Google style method we will mention what is input type and output type and who logic is goanna work.

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:

```
def login(username: str, password: str, credentials: dict) -> bool:
    """
    Logs in a user with the given username and password.
    Args:
        username (str): The username of the user.
        password (str): The password of the user.
```

```
    credentials (dict): A dictionary containing usernames as keys and
passwords as values.
    Returns:
        bool: True if login is successful, False otherwise.
    """
    return credentials.get(username) == password # Check if the username exists
and the password matches
```

Comparison:

Google style is simple and easy to read, NumPy style is more structured and detailed, and reST style is more formal but harder for beginners.

Recommendation:

Google style is best for new developers because it is clear, beginner-friendly, and easy to understand quickly.

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:

```

def add(a: float, b: float) -> float:
    """Returns the sum of a and b.
    Args:
        a: The first number.
        b: The second number.
    Returns:
        The sum of a and b.
    raises:
        TypeError: If a or b is not a number.
        ValueError: If a or b is not a finite number.
    """
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):
        raise TypeError("Both arguments must be numbers.")
    if not (isinstance(a, float) and a.is_finite()) and not (isinstance(b,
float) and b.is_finite()):
        raise ValueError("Both arguments must be finite numbers.")
    return a + b

def subtract(a: float, b: float) -> float:
    """Returns the difference of a and b.
    Args:
        a: The first number.
        b: The second number.
    Returns:
        The difference of a and b.
    raises:
        TypeError: If a or b is not a number.
        ValueError: If a or b is not a finite number.
    """
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):
        raise TypeError("Both arguments must be numbers.")
    if not (isinstance(a, float) and a.is_finite()) and not (isinstance(b,
float) and b.is_finite()):
        raise ValueError("Both arguments must be finite numbers.")
    return a - b

def multiply(a: float, b: float) -> float:
    """Returns the product of a and b.
    Args:
        a: The first number.
        b: The second number.
    Returns:
        The product of a and b.
    raises:
        TypeError: If a or b is not a number.
        ValueError: If a or b is not a finite number.
    """
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):
        raise TypeError("Both arguments must be numbers.")

```

```

    if not (isinstance(a, float) and a.is_finite()) and not (isinstance(b,
float) and b.is_finite()):
        raise ValueError("Both arguments must be finite numbers.")
    return a * b
def divide(a: float, b: float) -> float:
    """Returns the quotient of a and b.
    Args:
        a: The first number.
        b: The second number.
    Returns:
        The quotient of a and b.
    raises:
        TypeError: If a or b is not a number.
        ValueError: If a or b is not a finite number.
        ZeroDivisionError: If b is zero.
    """
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):
        raise TypeError("Both arguments must be numbers.")
    if not (isinstance(a, float) and a.is_finite()) and not (isinstance(b,
float) and b.is_finite()):
        raise ValueError("Both arguments must be finite numbers.")
    if b == 0:
        raise ZeroDivisionError("The second argument cannot be zero.")
    return a / b

```

```

# (AIAC_env) PS D:\Course\AIAC\lab9.1_25_2_2026> python -m pydoc calculator
# Help on module calculator:

# NAME
#     calculator

# FUNCTIONS
#     add(a: float, b: float) -> float
#         Returns the sum of a and b.
#         Args:
#             a: The first number.
#             b: The second number.
#         Returns:
#             The sum of a and b.
#         raises:
#             TypeError: If a or b is not a number.
#             ValueError: If a or b is not a finite number.
#
#     divide(a: float, b: float) -> float
#         Returns the quotient of a and b.
#         Args:

```

```
#         a: The first number.
#         b: The second number.
#     Returns:
#         The quotient of a and b.
#     raises:
#         TypeError: If a or b is not a number.
#         ValueError: If a or b is not a finite number.
#         ZeroDivisionError: If b is zero.

# multiply(a: float, b: float) -> float
#     Returns the product of a and b.
#     Args:
#         a: The first number.
#         b: The second number.
#     Returns:
#         The product of a and b.
#     raises:
#         TypeError: If a or b is not a number.
#         ValueError: If a or b is not a finite number.

# subtract(a: float, b: float) -> float
#     Returns the difference of a and b.
#     Args:
#         a: The first number.
#         b: The second number.
#     Returns:
#         The difference of a and b.
#     raises:
#         TypeError: If a or b is not a number.
#         ValueError: If a or b is not a finite number.

# FILE
#     d:\course\aiac\lab9.1_25_2_2026\calculator.py
```

[index](#)
calculator d:\course\aiac\lab9.1_25_2_2026\calculator.py

Functions

add(a: float, b: float) -> float
Returns the sum of a and b.
Args:
 a: The first number.
 b: The second number.
Returns:
 The sum of a and b.
raises:
 TypeError: If a or b is not a number.
 ValueError: If a or b is not a finite number.

divide(a: float, b: float) -> float
Returns the quotient of a and b.
Args:
 a: The first number.
 b: The second number.
Returns:
 The quotient of a and b.
raises:
 TypeError: If a or b is not a number.
 ValueError: If a or b is not a finite number.
 ZeroDivisionError: If b is zero.

multiply(a: float, b: float) -> float
Returns the product of a and b.
Args:
 a: The first number.
 b: The second number.
Returns:
 The product of a and b.
raises:
 TypeError: If a or b is not a number.
 ValueError: If a or b is not a finite number.

subtract(a: float, b: float) -> float
Returns the difference of a and b.
Args:
 a: The first number.
 b: The second number.
Returns:
 The difference of a and b.
raises:
 TypeError: If a or b is not a number.
 ValueError: If a or b is not a finite number.

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:

```
def decimal_to_binary(n):
    """Converts a decimal number to binary representation.
    Args:
        n: A decimal number.
    Returns:
        A string representing the binary representation of n.
    raises:
        TypeError: If n is not an integer.
        ValueError: If n is a negative integer.
    """
    if not isinstance(n, int):
        raise TypeError("Input must be an integer.")
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    if n == 0:
        return "0"
    binary = ""
    while n > 0:
        binary = str(n % 2) + binary
        n //= 2
    return binary

def binary_to_decimal(b):
    """Converts a binary string to decimal representation.
    Args:
        b: A string representing a binary number.
    Returns:
        An integer representing the decimal representation of b.
    raises:
        TypeError: If b is not a string.
```



```

        ValueError: If b is not a valid binary string.
        """
    if not isinstance(b, str):
        raise TypeError("Input must be a string.")
    if not all(char in "01" for char in b):
        raise ValueError("Input must be a valid binary string.")
    decimal = 0
    for index, char in enumerate(reversed(b)):
        decimal += int(char) * (2 ** index)
    return decimal
def decimal_to_hexadecimal(n):
    """Converts a decimal number to hexadecimal representation.
    Args:
        n: A decimal number.
    Returns:
        A string representing the hexadecimal representation of n.
    raises:
        TypeError: If n is not an integer.
        ValueError: If n is a negative integer.
    """
    if not isinstance(n, int):
        raise TypeError("Input must be an integer.")
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    if n == 0:
        return "0"
    hexadecimal = ""
    hex_digits = "0123456789ABCDEF"
    while n > 0:
        hexadecimal = hex_digits[n % 16] + hexadecimal
        n //= 16
    return hexadecimal

```

```

# (AIAC_env) PS D:\Course\AIAC\lab9.1_25_2_2026> python -m pydoc conversion
# Help on module conversion:

# NAME

```

```
# conversion

# FUNCTIONS

# binary_to_decimal(b)
#     Converts a binary string to decimal representation.
#     Args:
#         b: A string representing a binary number.
#     Returns:
#         An integer representing the decimal representation of b.
#     raises:
#         TypeError: If b is not a string.
#         ValueError: If b is not a valid binary string.

# decimal_to_binary(n)
#     Converts a decimal number to binary representation.
#     Args:
#         n: A decimal number.
#     Returns:
#         A string representing the binary representation of n.
#     raises:
#         TypeError: If n is not an integer.
#         ValueError: If n is a negative integer.

# decimal_to_hexadecimal(n)
#     Converts a decimal number to hexadecimal representation.
#     Args:
#         n: A decimal number.
#     Returns:
#         A string representing the hexadecimal representation of n.
#     raises:
#         TypeError: If n is not an integer.
#         ValueError: If n is a negative integer.

# FILE
#     d:\course\aiac\lab9.1_25_2_2026\conversion.py
```

conversion

Functions

binary_to_decimal(b)

Converts a binary string to decimal representation.

Args:

b: A string representing a binary number.

Returns:

An integer representing the decimal representation of b.

raises:

TypeError: If b is not a string.

ValueError: If b is not a valid binary string.

decimal_to_binary(n)

Converts a decimal number to binary representation.

Args:

n: A decimal number.

Returns:

A string representing the binary representation of n.

raises:

TypeError: If n is not an integer.

ValueError: If n is a negative integer.

decimal_to_hexadecimal(n)

Converts a decimal number to hexadecimal representation.

Args:

n: A decimal number.

Returns:

A string representing the hexadecimal representation of n.

raises:

TypeError: If n is not an integer.

ValueError: If n is a negative integer.

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:

```
def add_course(course_id: str, course_name: str, credits: int) -> dict:
    """Adds a course to the course catalog.
    Args:
        course_id: The unique identifier for the course.
        course_name: The name of the course.
        credits: The number of credits for the course.
    Returns:
        A dictionary representing the added course.
    raises:
        TypeError: If course_id or course_name is not a string, or if credits
is not an integer.
        ValueError: If course_id is empty or already exists, if course_name is
empty, or if credits is not a positive integer.
    """
    if not isinstance(course_id, str) or not isinstance(course_name, str) or
not isinstance(credits, int):
        raise TypeError("course_id and course_name must be strings, and
credits must be an integer.")
    if not course_id:
        raise ValueError("course_id cannot be empty.")
    if not course_name:
        raise ValueError("course_name cannot be empty.")
    if credits <= 0:
        raise ValueError("credits must be a positive integer.")

    # Assuming we have a global course catalog dictionary
    global course_catalog
    if course_id in course_catalog:
        raise ValueError("course_id already exists in the catalog.")

    course_catalog[course_id] = {
        "course_name": course_name,
        "credits": credits
    }
    return course_catalog[course_id]

def remove_course(course_id: str) -> None:
    """Removes a course from the course catalog.
    Args:
        course_id: The unique identifier for the course to be removed.
    Raises:
        TypeError: If course_id is not a string.
        ValueError: If course_id is empty or does not exist in the catalog.
    """
```

```

if not isinstance(course_id, str):
    raise TypeError("course_id must be a string.")
if not course_id:
    raise ValueError("course_id cannot be empty.")

global course_catalog
if course_id not in course_catalog:
    raise ValueError("course_id does not exist in the catalog.")

del course_catalog[course_id]

def get_course(course_id: str) -> dict:
    """Retrieves a course from the course catalog.
    Args:
        course_id: The unique identifier for the course to be retrieved.
    Returns:
        A dictionary representing the retrieved course.
    Raises:
        TypeError: If course_id is not a string.
        ValueError: If course_id is empty or does not exist in the catalog.
    """
    if not isinstance(course_id, str):
        raise TypeError("course_id must be a string.")
    if not course_id:
        raise ValueError("course_id cannot be empty.")

    global course_catalog
    if course_id not in course_catalog:
        raise ValueError("course_id does not exist in the catalog.")

    return course_catalog[course_id]

```

```

(AIAC_env) PS D:\Course\AIAC\lab9.1_25_2_2026> python -m pydoc course
# Help on module course:

# NAME
#     course

# FUNCTIONS
#     add_course(course_id: str, course_name: str, credits: int) -> dict
#         Adds a course to the course catalog.
#         Args:
#             course_id: The unique identifier for the course.
#             course_name: The name of the course.
#             credits: The number of credits for the course.
#         Returns:
#             A dictionary representing the added course.
#         raises:

```

```
#         TypeError: If course_id or course_name is not a string, or if
credits is not an integer.
#         ValueError: If course_id is empty or already exists, if
course_name is empty, or if credits is not a positive integer.

#     get_course(course_id: str) -> dict
#         Retrieves a course from the course catalog.
#         Args:
#             course_id: The unique identifier for the course to be retrieved.
#         Returns:
#             A dictionary representing the retrieved course.
#         Raises:
#             TypeError: If course_id is not a string.
#             ValueError: If course_id is empty or does not exist in the
catalog.

#     remove_course(course_id: str) -> None
#         Removes a course from the course catalog.
#         Args:
#             course_id: The unique identifier for the course to be removed.
#         Raises:
#             TypeError: If course_id is not a string.
#             ValueError: If course_id is empty or does not exist in the
catalog.

# FILE
#     d:\course\aiac\lab9.1_25_2_2026\course.py
```

[index](#)

course [d:\course\aiac\lab9.1_25_2_2026\course.py](#)

Functions

add_course(course_id: str, course_name: str, credits: int) -> dict
Adds a course to the course catalog.
Args:
 course_id: The unique identifier for the course.
 course_name: The name of the course.
 credits: The number of credits for the course.
Returns:
 A dictionary representing the added course.
raises:
 TypeError: If course_id or course_name is not a string, or if credits is not an integer.
 ValueError: If course_id is empty or already exists, if course_name is empty, or if credits is not a positive integer.

get_course(course_id: str) -> dict
Retrieves a course from the course catalog.
Args:
 course_id: The unique identifier for the course to be retrieved.
Returns:
 A dictionary representing the retrieved course.
Raises:
 TypeError: If course_id is not a string.
 ValueError: If course_id is empty or does not exist in the catalog.

remove_course(course_id: str) -> None
Removes a course from the course catalog.
Args:
 course_id: The unique identifier for the course to be removed.
Raises:
 TypeError: If course_id is not a string.
 ValueError: If course_id is empty or does not exist in the catalog.

Python 3.14.0 [tags/v3.14.0:ebf955d, MSC v.1944 64 bit (AMD64)]
Windows-11

course

Functions

add_course(course_id: str, course_name: str, credits: int) -> dict
Adds a course to the course catalog.
Args:
 course_id: The unique identifier for the course.
 course_name: The name of the course.
 credits: The number of credits for the course.
Returns:
 A dictionary representing the added course.
raises:
 TypeError: If course_id or course_name is not a string, or if credits is not an integer.
 ValueError: If course_id is empty or already exists, if course_name is empty, or if credits is not a positive integer.

get_course(course_id: str) -> dict
Retrieves a course from the course catalog.
Args:
 course_id: The unique identifier for the course to be retrieved.
Returns:
 A dictionary representing the retrieved course.
Raises:
 TypeError: If course_id is not a string.
 ValueError: If course_id is empty or does not exist in the catalog.

remove_course(course_id: str) -> None
Removes a course from the course catalog.
Args:
 course_id: The unique identifier for the course to be removed.
Raises:
 TypeError: If course_id is not a string.
 ValueError: If course_id is empty or does not exist in the catalog.