ASSIGNMENT – 9.1

Name: N SOURISH

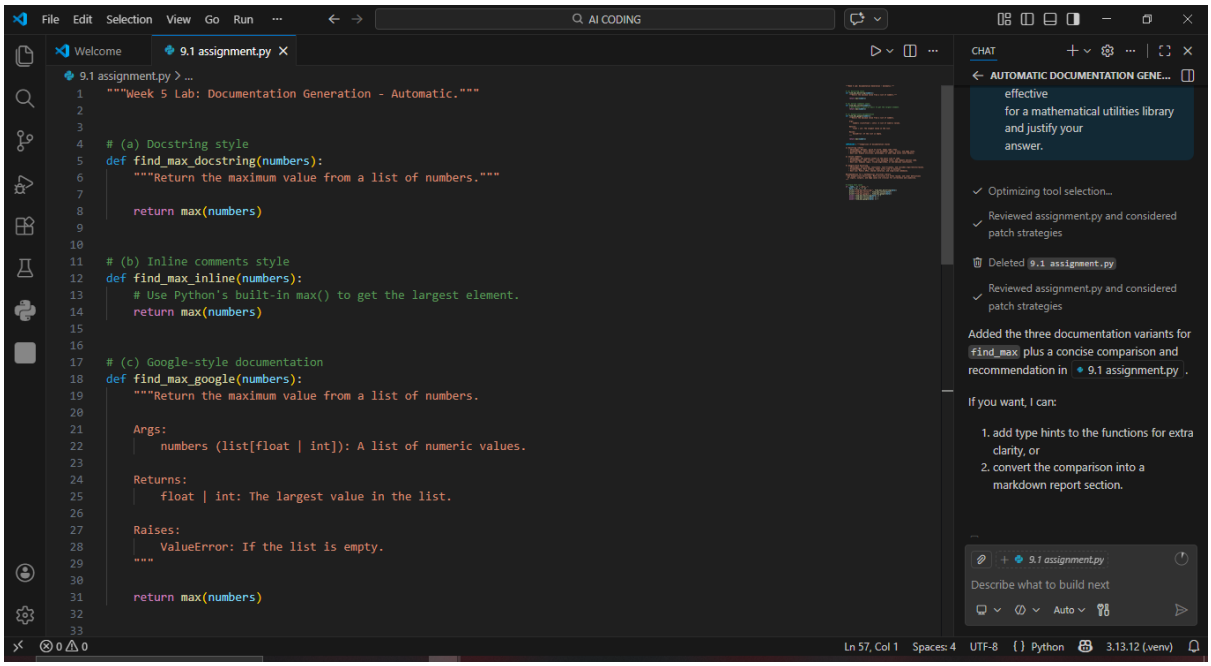
HT.NO: 2303A52360

BATCH - 44

Problem 1:

Task:

1)



2)

3)

```
Recommendation for a mathematical utilities library
- Use Google-style docstrings. Math utilities are often reused, and clear definitions
  of inputs, outputs, and edge cases are critical for correctness and usability.
"""
```

Testcase:

```python
# Simple test cases
if __name__ == "__main__":
    data = [3, 1, 7, 2]
    print("find_max_docstring:", find_max_docstring(data))
    print("find_max_inline:", find_max_inline(data))
    print("find_max_google:", find_max_google(data))
    assert find_max_docstring(data) == 7
    assert find_max_inline(data) == 7
    assert find_max_google(data) == 7
```

Testcase output:

```
find_max_docstring: 7
find_max_inline: 7
find_max_google: 7
```

Problem 2 :

Task:

1)

```python
# Problem 2
# (a) Docstring style
def login_docstring(user, password, credentials):
    """Return True if the password matches the stored credentials."""

    return credentials.get(user) == password


# (b) Inline comments style
def login_inline(user, password, credentials):
    # Compare provided password with the stored password for the user.
    return credentials.get(user) == password


# (c) Google-style documentation
def login_google(user, password, credentials):
    """Validate a user's login attempt.

    Args:
        user (str): Username to authenticate.
        password (str): Password provided by the user.
        credentials (dict[str, str]): Mapping of usernames to passwords.

    Returns:
        bool: True when the password matches, otherwise False.
    """

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

2)

```
COMPARISON_LOGIN = """Comparison of documentation styles for login()

1) Docstring (short)
   - Advantages: Quick overview, low maintenance overhead.
   - Disadvantages: Lacks detail on parameters, types, and behavior.
   - Best use: Small internal helpers or obvious behavior.

2) Inline comments
   - Advantages: Highlights intent right where logic occurs.
   - Disadvantages: Can repeat obvious code and add clutter.
   - Best use: Non-obvious logic, security checks, or edge cases.

3) Google-style docstring
   - Advantages: Clear contract for new readers and tooling support.
   - Disadvantages: More verbose and slower to write.
   - Best use: Public-facing functions and onboarding materials.
```

3)

```
Recommendation for onboarding new developers
- Use Google-style docstrings. They provide consistent, structured explanations
  of inputs and outputs, which helps new contributors understand expectations
  and reduces misinterpretation.
"""
```

Testcase:

```
# testcases for login functions
creds = {"alice": "p@ss", "bob": "1234"}
print("login_docstring:", login_docstring("alice", "p@ss", creds))
print("login_inline:", login_inline("bob", "wrong", creds))
print("login_google:", login_google("bob", "1234", creds))
assert login_docstring("alice", "p@ss", creds) is True
assert login_inline("bob", "wrong", creds) is False
assert login_google("bob", "1234", creds) is True
```

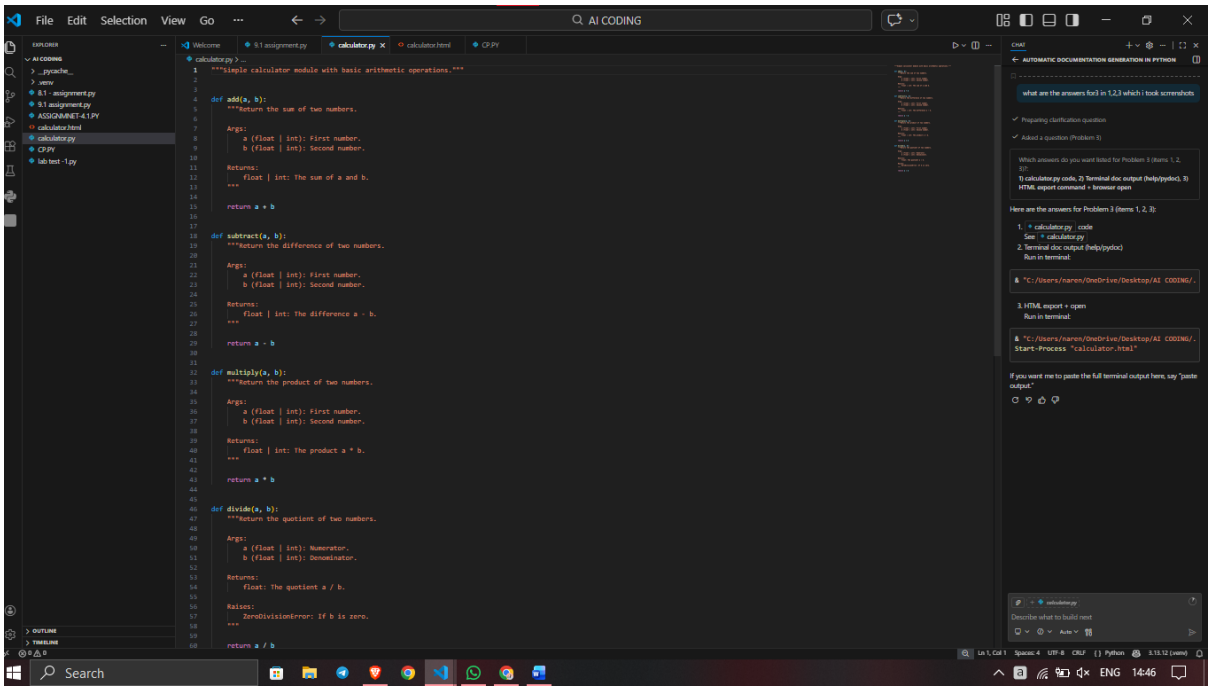
Testcase Output:

```
login_docstring: True
login_inline: False
login_google: True
```

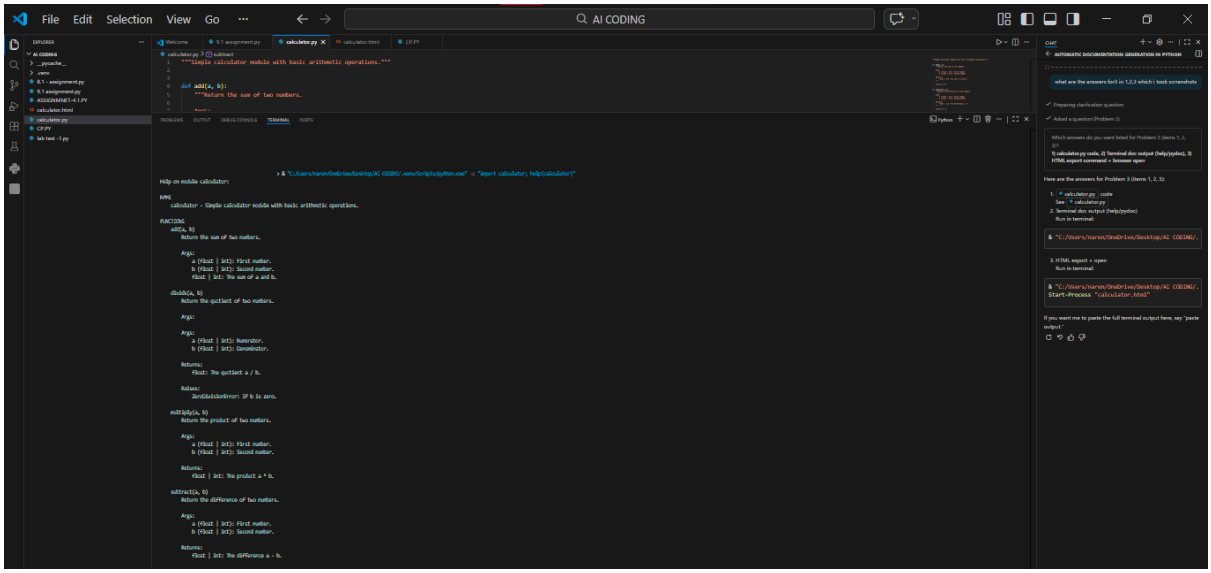
Problem 3 :

Task:

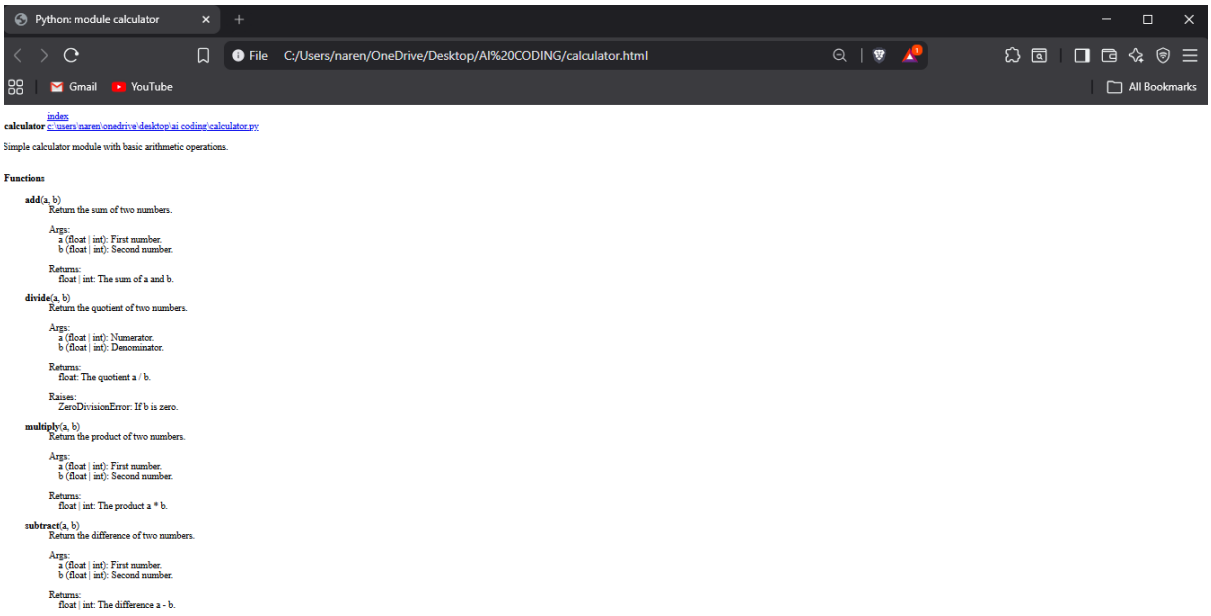
1)



2)



3)



Problem 4:

1)

```python
def decimal_to_binary(n):


    if n < 0:
        raise ValueError("n must be non-negative")
    return bin(n)[2:]


def binary_to_decimal(b):


    if not b or any(ch not in "01" for ch in b):
        raise ValueError("b must be a non-empty binary string")
    return int(b, 2)


def decimal_to_hexadecimal(n):


    if n < 0:
        raise ValueError("n must be non-negative")
    return hex(n)[2:]
```

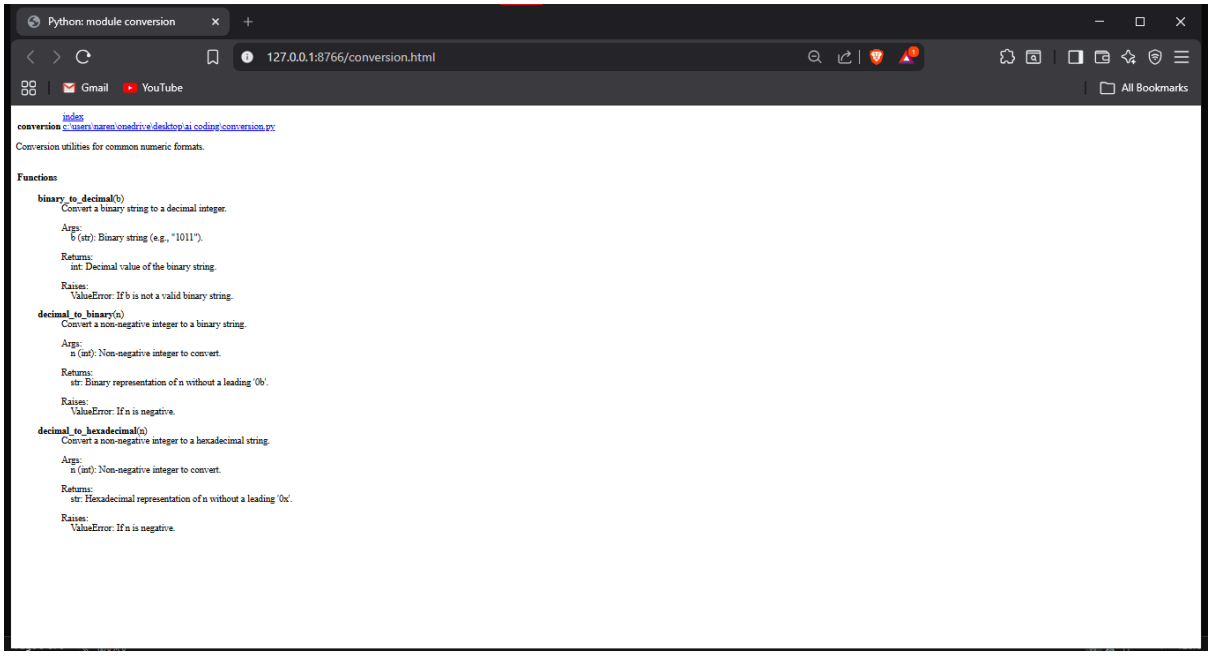
2)

```python
"""Conversion utilities for common numeric formats."""


def decimal_to_binary(n):
    """Convert a non-negative integer to a binary string.

    Args:
        n (int): Non-negative integer to convert.

    Returns:
        str: Binary representation of n without a leading '0b'.

    Raises:
        ValueError: If n is negative.
    """
    if n < 0:
        raise ValueError("n must be non-negative")
    return bin(n)[2:]


def binary_to_decimal(b):
    """Convert a binary string to a decimal integer.

    Args:
        b (str): Binary string (e.g., "1011").

    Returns:
        int: Decimal value of the binary string.

    Raises:
        ValueError: If b is not a valid binary string.
    """
    if not b or any(ch not in "01" for ch in b):
        raise ValueError("b must be a non-empty binary string")
    return int(b, 2)


def decimal_to_hexadecimal(n):
    """Convert a non-negative integer to a hexadecimal string.

    Args:
        n (int): Non-negative integer to convert.

    Returns:
        str: Hexadecimal representation of n without a leading '0x'.

    Raises:
        ValueError: If n is negative.
    """
    if n < 0:
        raise ValueError("n must be non-negative")
    return hex(n)[2:]
```

3)

```
Python Library Documentation: module conversion

NAME
    conversion - Conversion utilities for common numeric formats.

FUNCTIONS
    binary_to_decimal(b)
        Convert a binary string to a decimal integer.

        Args:
            b (str): Binary string (e.g., "1011").

        Returns:
            int: Decimal value of the binary string.

        Raises:
            ValueError: If b is not a valid binary string.

    decimal_to_binary(n)
        Convert a non-negative integer to a binary string.

        Args:
            n (int): Non-negative integer to convert.

        Returns:
            str: Binary representation of n without a leading '0b'.

        Raises:
            ValueError: If n is negative.

    decimal_to_hexadecimal(n)
        Convert a non-negative integer to a hexadecimal string.

        Args:
            n (int): Non-negative integer to convert.

        Returns:
            str: Hexadecimal representation of n without a leading '0x'.

        Raises:
            ValueError: If n is negative.
```

4)



Problem 5:

Task:

1)

```python
_courses = {}

def add_course(course_id, name, credits):

    if not course_id or not str(course_id).strip():
        raise ValueError("course_id must be a non-empty value")
    if not name or not str(name).strip():
        raise ValueError("name must be a non-empty value")
    if not isinstance(credits, int):
        raise TypeError("credits must be an integer")
    if credits < 0:
        raise ValueError("credits must be non-negative")

    record = {
        "course_id": str(course_id).strip(),
        "name": str(name).strip(),
        "credits": credits,
    }
    _courses[record["course_id"]] = record
    return record


def remove_course(course_id):

    if not course_id:
        return False
    return _courses.pop(str(course_id).strip(), None) is not None


def get_course(course_id):

    if not course_id:
        return None
    return _courses.get(str(course_id).strip())
```

2)

```python
"""Course management utilities for adding, removing, and retrieving courses."""

_courses = {}


def add_course(course_id, name, credits):
    """Add or update a course in the in-memory course catalog.

    Args:
        course_id (str): Unique identifier for the course (for example, "CS101").
        name (str): Human-readable course name.
        credits (int): Number of credits assigned to the course.

    Returns:
        dict: The stored course record with keys ``course_id``, ``name``, and ``credits``.

    Raises:
        ValueError: If ``course_id`` or ``name`` is empty, or if ``credits`` is negative.
        TypeError: If ``credits`` is not an integer.
    """

    if not course_id or not str(course_id).strip():
        raise ValueError("course_id must be a non-empty value")
    if not name or not str(name).strip():
        raise ValueError("name must be a non-empty value")
    if not isinstance(credits, int):
        raise TypeError("credits must be an integer")
    if credits < 0:
        raise ValueError("credits must be non-negative")

    record = {
        "course_id": str(course_id).strip(),
        "name": str(name).strip(),
        "credits": credits,
    }
    _courses[record["course_id"]] = record
    return record


def remove_course(course_id):
    """Remove a course from the in-memory course catalog.

    Args:
        course_id (str): Identifier of the course to remove.

    Returns:
        bool: ``True`` if a course was removed, otherwise ``False``.
    """

    if not course_id:
        return False
    return _courses.pop(str(course_id).strip(), None) is not None


def get_course(course_id):
    """Retrieve a course by its identifier.

    Args:
        course_id (str): Identifier of the course to fetch.

    Returns:
        dict | None: The course record if found, otherwise ``None``.
    """

    if not course_id:
        return None
    return _courses.get(str(course_id).strip())
```

3)

```
NAME
    conversion - Conversion utilities for common numeric formats.

FUNCTIONS
    binary_to_decimal(b)
        Convert a binary string to a decimal integer.

        Args:
            b (str): Binary string (e.g., "1011").

        Returns:
            int: Decimal value of the binary string.

        Raises:
            ValueError: If b is not a valid binary string.

    decimal_to_binary(n)
        Convert a non-negative integer to a binary string.

        Args:
            n (int): Non-negative integer to convert.

        Returns:
            str: Binary representation of n without a leading '0b'.

        Raises:
            ValueError: If n is negative.

    decimal_to_hexadecimal(n)
        Convert a non-negative integer to a hexadecimal string.

        Args:
            n (int): Non-negative integer to convert.

        Returns:
            str: Hexadecimal representation of n without a leading '0x'.

        Raises:
            ValueError: If n is negative.
```

4)