

# ASSIGNMENT – 9.1

Name: N SOURISH

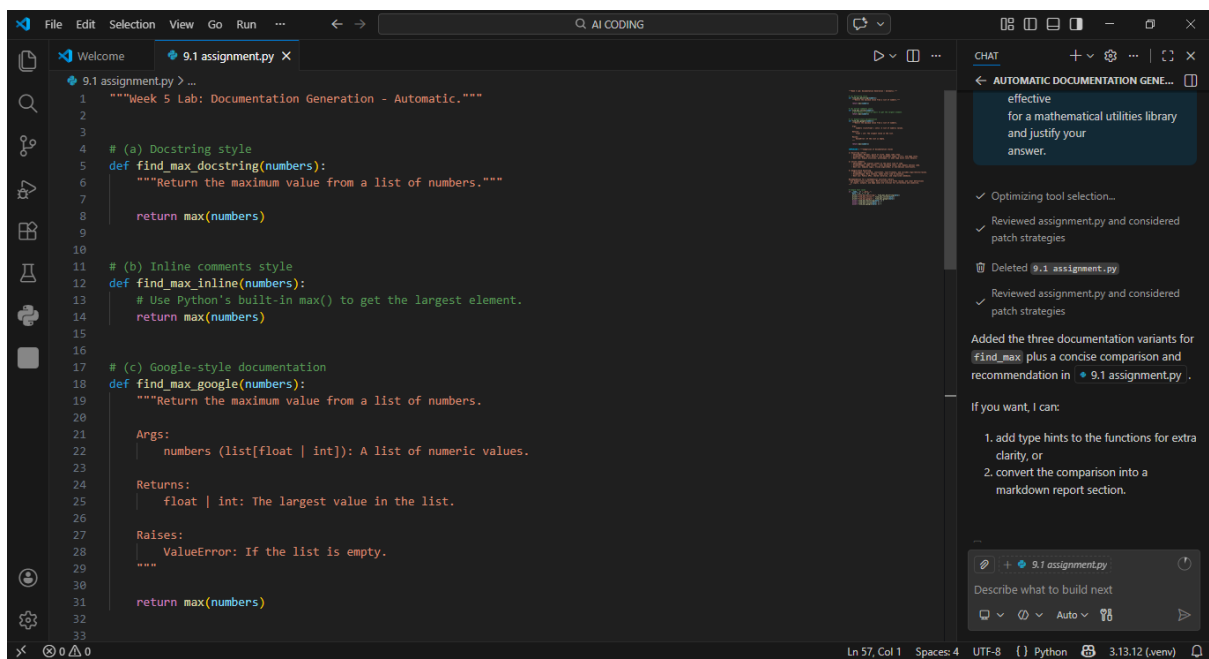
HT.NO: 2303A52360

BATCH - 44

## Problem 1:

### Task:

1)



```
1 """Week 5 Lab: Documentation Generation - Automatic."""
2
3
4 # (a) Docstring style
5 def find_max_docstring(numbers):
6     """Return the maximum value from a list of numbers."""
7
8     return max(numbers)
9
10
11 # (b) Inline comments style
12 def find_max_inline(numbers):
13     # Use Python's built-in max() to get the largest element.
14     return max(numbers)
15
16
17 # (c) Google-style documentation
18 def find_max_google(numbers):
19     """Return the maximum value from a list of numbers.
20
21     Args:
22         numbers (list[float | int]): A list of numeric values.
23
24     Returns:
25         float | int: The largest value in the list.
26
27     Raises:
28         ValueError: If the list is empty.
29     """
30
31     return max(numbers)
32
33
```

CHAT

AUTOMATIC DOCUMENTATION GENERATION

effective for a mathematical utilities library and justify your answer.

✓ Optimizing tool selection...

✓ Reviewed assignment.py and considered patch strategies

Deleted 9.1 assignment.py

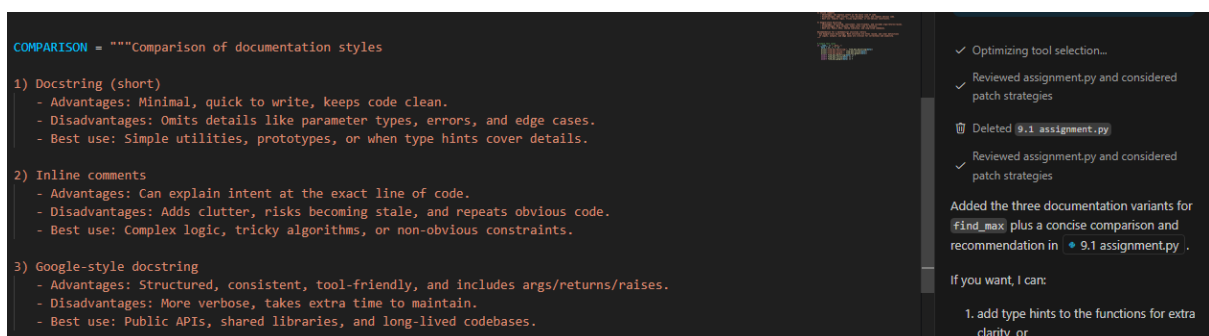
✓ Reviewed assignment.py and considered patch strategies

Added the three documentation variants for `find_max` plus a concise comparison and recommendation in `9.1 assignment.py`.

If you want, I can:

1. add type hints to the functions for extra clarity, or
2. convert the comparison into a markdown report section.

2)



```
COMPARISON = """Comparison of documentation styles

1) Docstring (short)
- Advantages: Minimal, quick to write, keeps code clean.
- Disadvantages: Omits details like parameter types, errors, and edge cases.
- Best use: Simple utilities, prototypes, or when type hints cover details.

2) Inline comments
- Advantages: Can explain intent at the exact line of code.
- Disadvantages: Adds clutter, risks becoming stale, and repeats obvious code.
- Best use: Complex logic, tricky algorithms, or non-obvious constraints.

3) Google-style docstring
- Advantages: Structured, consistent, tool-friendly, and includes args/returns/raises.
- Disadvantages: More verbose, takes extra time to maintain.
- Best use: Public APIs, shared libraries, and long-lived codebases.
```

✓ Optimizing tool selection...

✓ Reviewed assignment.py and considered patch strategies

Deleted 9.1 assignment.py

✓ Reviewed assignment.py and considered patch strategies

Added the three documentation variants for `find_max` plus a concise comparison and recommendation in `9.1 assignment.py`.

If you want, I can:

1. add type hints to the functions for extra clarity, or

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:

```
# 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)

```
# 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
```

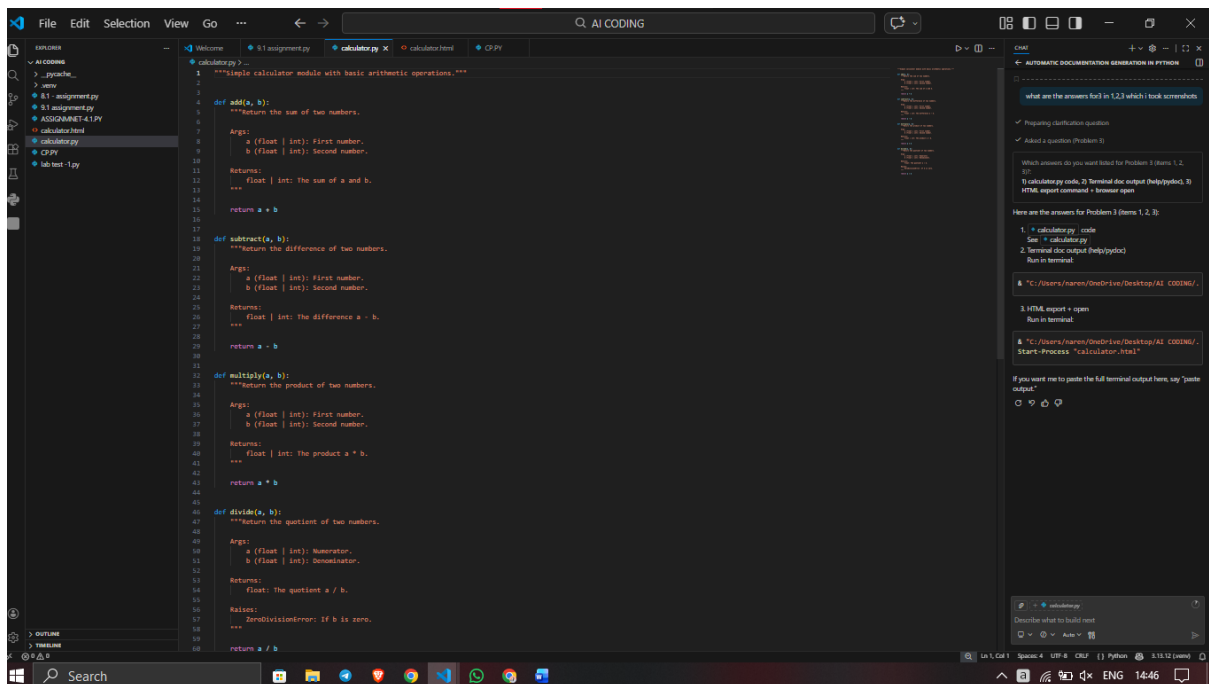
**Testcase Output:**

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

## Problem 3 :

### Task:

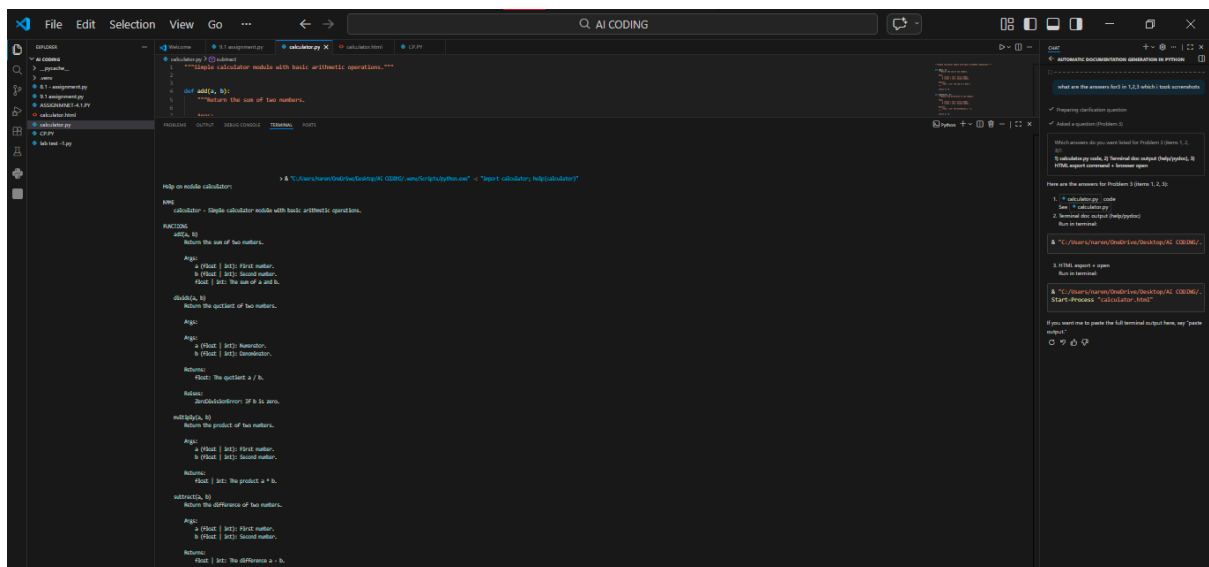
1)



The screenshot shows a VS Code editor with a file explorer on the left containing files like `calculator.py`, `calculator.html`, and `calculator.js`. The main editor displays the content of `calculator.py`, which defines four functions: `add(a, b)`, `subtract(a, b)`, `multiply(a, b)`, and `divide(a, b)`. Each function includes docstrings and type hints. The `divide` function also includes a `ZeroDivisionError` exception. On the right, an AI chat window is open, showing a conversation about the calculator module. The chat includes a question about the module's purpose and a response listing the functions and their usage.

```
calculator.py
"""Simple calculator module with basic arithmetic operations."""
def add(a, b):
    """Return the sum of two numbers.
    Args:
        a (float | int): First number.
        b (float | int): Second number.
    Returns:
        float | int: The sum of a and b.
    """
    return a + b
def subtract(a, b):
    """Return the difference of two numbers.
    Args:
        a (float | int): First number.
        b (float | int): Second number.
    Returns:
        float | int: The difference a - b.
    """
    return a - b
def multiply(a, b):
    """Return the product of two numbers.
    Args:
        a (float | int): First number.
        b (float | int): Second number.
    Returns:
        float | int: The product a * b.
    """
    return a * b
def divide(a, b):
    """Return the quotient of two numbers.
    Args:
        a (float | int): Numerator.
        b (float | int): Denominator.
    Returns:
        float: The quotient a / b.
    Raises:
        ZeroDivisionError: If b is zero.
    """
    return a / b
```

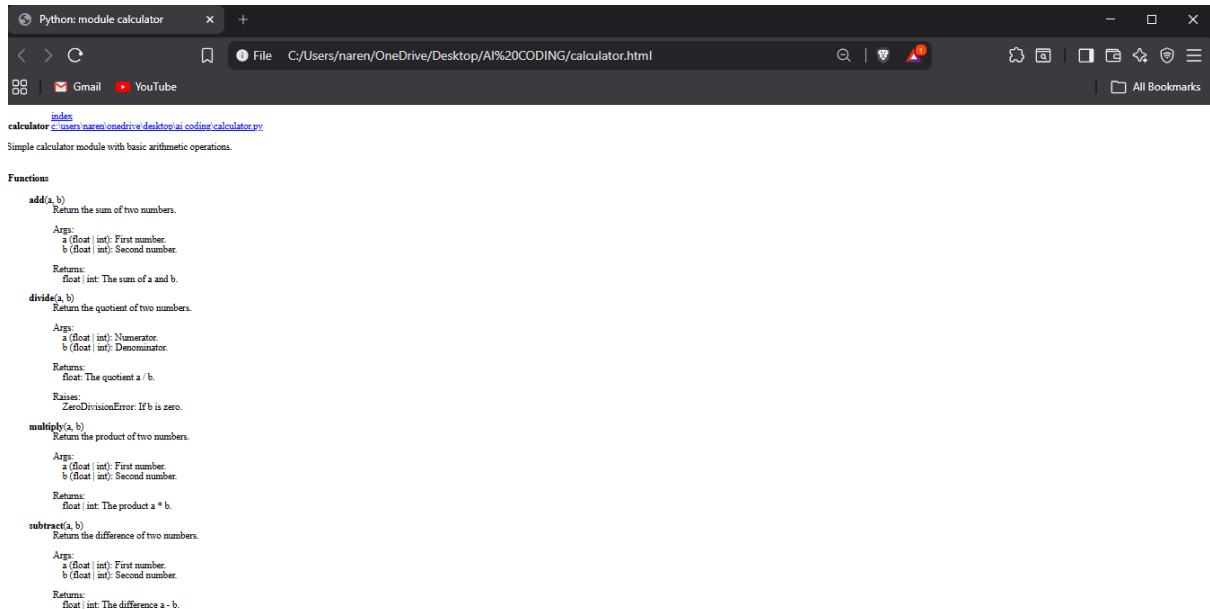
2)



The screenshot shows a VS Code editor with a file explorer on the left containing files like `calculator.py`, `calculator.html`, and `calculator.js`. The main editor displays the content of `calculator.py`, which defines four functions: `add(a, b)`, `subtract(a, b)`, `multiply(a, b)`, and `divide(a, b)`. Each function includes docstrings and type hints. The `divide` function also includes a `ZeroDivisionError` exception. On the right, an AI chat window is open, showing a conversation about the calculator module. The chat includes a question about the module's purpose and a response listing the functions and their usage.

```
calculator.py
"""Simple calculator module with basic arithmetic operations."""
def add(a, b):
    """Return the sum of two numbers.
    Args:
        a (float | int): First number.
        b (float | int): Second number.
    Returns:
        float | int: The sum of a and b.
    """
    return a + b
def subtract(a, b):
    """Return the difference of two numbers.
    Args:
        a (float | int): First number.
        b (float | int): Second number.
    Returns:
        float | int: The difference a - b.
    """
    return a - b
def multiply(a, b):
    """Return the product of two numbers.
    Args:
        a (float | int): First number.
        b (float | int): Second number.
    Returns:
        float | int: The product a * b.
    """
    return a * b
def divide(a, b):
    """Return the quotient of two numbers.
    Args:
        a (float | int): Numerator.
        b (float | int): Denominator.
    Returns:
        float: The quotient a / b.
    Raises:
        ZeroDivisionError: If b is zero.
    """
    return a / b
```

3)



The screenshot shows a web browser window with the address bar displaying the file path: C:/Users/naren/OneDrive/Desktop/AI%20CODING/calculator.html. The page content includes a title 'Python: module calculator' and a description 'Simple calculator module with basic arithmetic operations.' Below this, a section titled 'Functions:' lists four functions: `add(a, b)`, `divide(a, b)`, `multiply(a, b)`, and `subtract(a, b)`. Each function is followed by its description, arguments, and return values.

```
Python: module calculator
C:/Users/naren/OneDrive/Desktop/AI%20CODING/calculator.html
Simple calculator module with basic arithmetic operations.

Functions:

add(a, b)
Return the sum of two numbers.
Args:
  a (float | int): First number.
  b (float | int): Second number.
Returns:
  float | int: The sum of a and b.

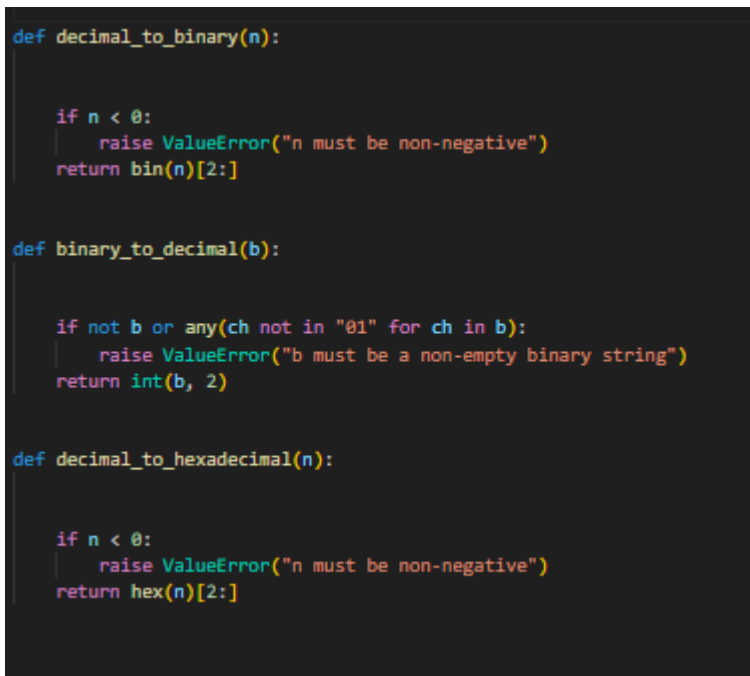
divide(a, b)
Return the quotient of two numbers.
Args:
  a (float | int): Numerator.
  b (float | int): Denominator.
Returns:
  float: The quotient a / b.
Raises:
  ZeroDivisionError: If b is zero.

multiply(a, b)
Return the product of two numbers.
Args:
  a (float | int): First number.
  b (float | int): Second number.
Returns:
  float | int: The product a * b.

subtract(a, b)
Return the difference of two numbers.
Args:
  a (float | int): First number.
  b (float | int): Second number.
Returns:
  float | int: The difference a - b.
```

## Problem 4:

1)



The screenshot shows a code editor with three Python functions. The first function, `decimal_to_binary(n)`, checks if `n` is non-negative and returns the binary representation. The second function, `binary_to_decimal(b)`, checks if `b` is a non-empty binary string and returns the decimal value. The third function, `decimal_to_hexadecimal(n)`, checks if `n` is non-negative and returns the hexadecimal representation.

```
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)

```

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.

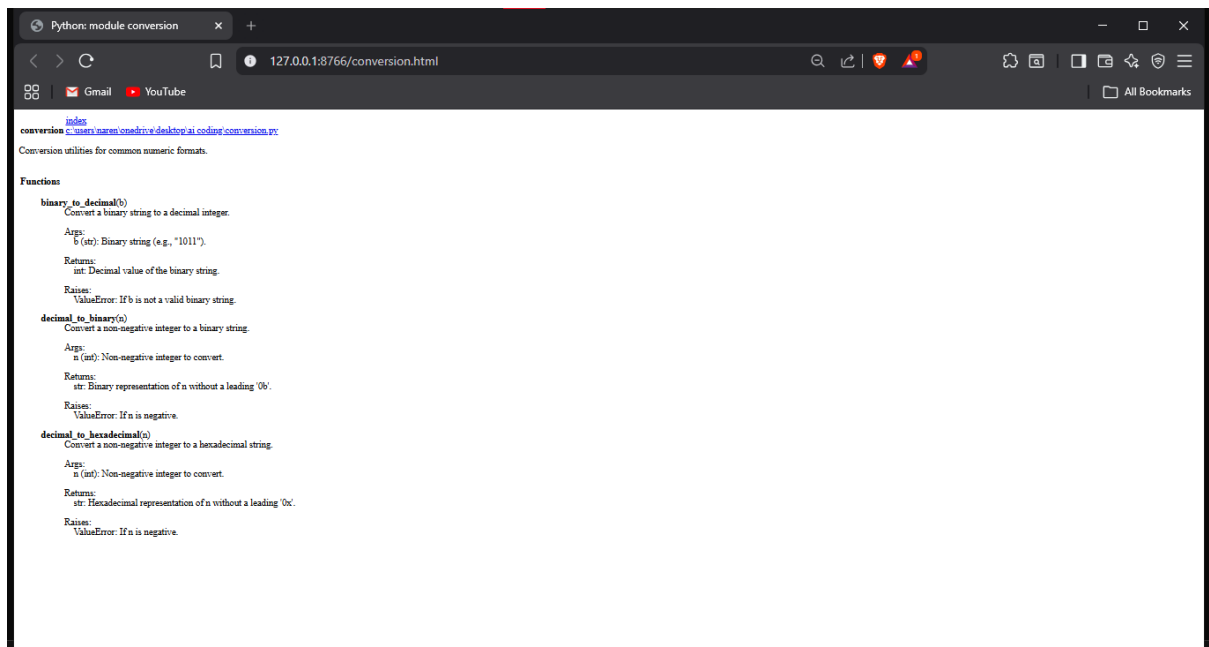
##### Returns:

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

##### Raises:

ValueError: If n is negative.

4)





## Problem 5:

### Task:

1)

```
_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)

```
"""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.

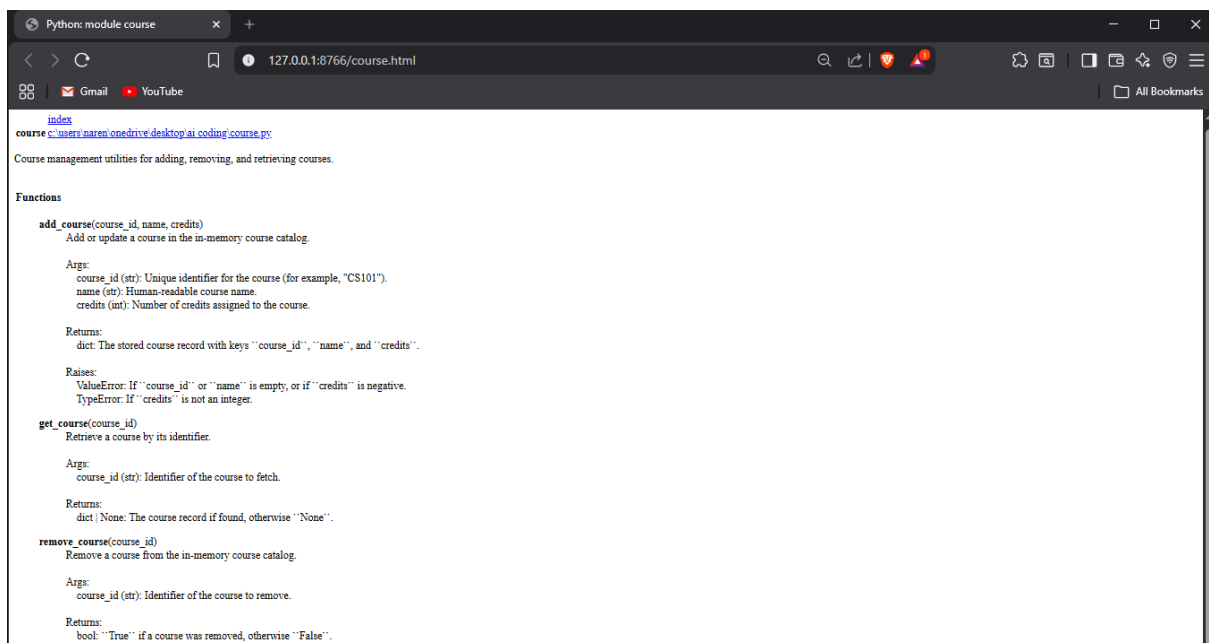
        Args:
            n (int): Non-negative integer to convert.

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

        Raises:
            ValueError: If n is negative.

```

4)



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8766/course.html". The page content is a documentation for a Python module named "course". It includes a brief description, a list of functions, and detailed documentation for each function, including arguments, return values, and exceptions.

[index](#)  
**course** `c:\users\maren\onedrive\desktop\ai coding\course.py`  
 Course management utilities for adding, removing, and retrieving courses.

**Functions**

**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.

**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".

**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".