

# AI Assisted Coding

## ASSIGNMENT 9.1

Name:T.Akshaya

HT No: 2303A52017

Batch: 31

Problem 1:

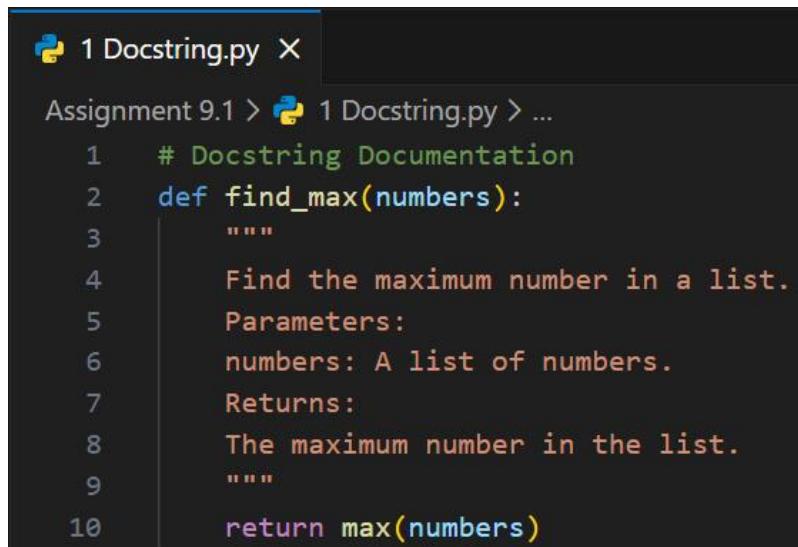
Consider the following Python function:

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

Task:

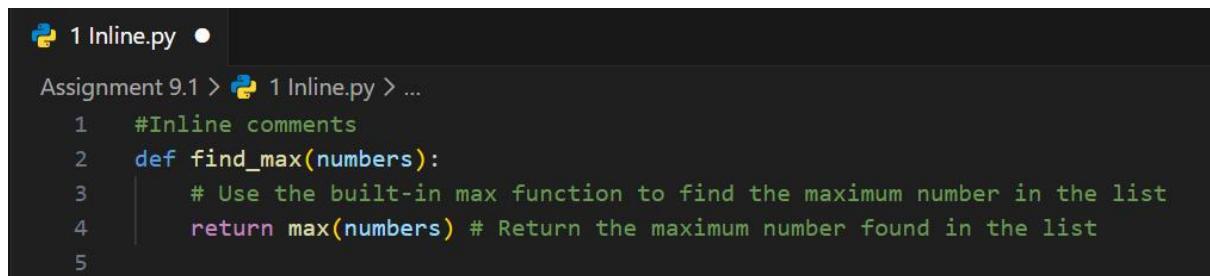
- Write documentation for the function in all three formats:
  - (a) Docstring
  - (b) Inline comments
  - (c) Google-style documentation
- Critically compare the three approaches. Discuss the advantages, disadvantages, and suitable use cases of each style.
- Recommend which documentation style is most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

### (a) Docstring



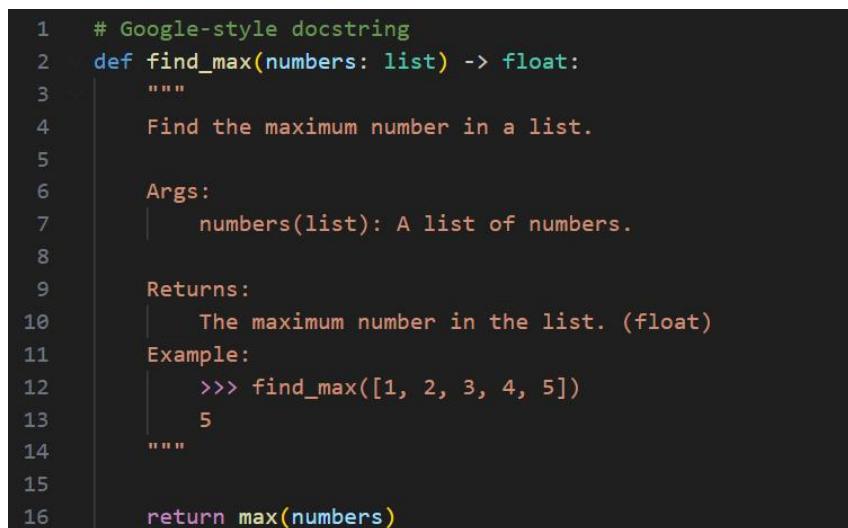
```
1 Docstring.py X
Assignment 9.1 > 1 Docstring.py > ...
1     # Docstring Documentation
2     def find_max(numbers):
3         """
4             Find the maximum number in a list.
5             Parameters:
6                 numbers: A list of numbers.
7             Returns:
8                 The maximum number in the list.
9             """
10            return max(numbers)
```

### (b) Inline comments



```
1 Inline.py ●
Assignment 9.1 > 1 Inline.py > ...
1     #Inline comments
2     def find_max(numbers):
3         # Use the built-in max function to find the maximum number in the list
4         return max(numbers) # Return the maximum number found in the list
5
```

### (c) Google-style documentation



```
1     # Google-style docstring
2     def find_max(numbers: list) -> float:
3         """
4             Find the maximum number in a list.
5
6             Args:
7                 numbers(list): A list of numbers.
8
9             Returns:
10                The maximum number in the list. (float)
11            Example:
12                >>> find_max([1, 2, 3, 4, 5])
13                5
14            """
15
16            return max(numbers)
```

Critically compare the three approaches. Discuss the advantages, disadvantages, and suitable use cases of each style.

Aspect	Docstring	Inline Comments	Google-style
Readability	Good	Minimal	Excellent
IDE Support	Native	None	Native
Formality	Standard	Casual	Professional
Maintenance	Easy	Tedious	Structured
Type Hints	Optional	Unclear	Explicit
Examples	Rare	Never	Common

Format	Advantages	Disadvantages
Docstring	IDE tooltips, help() function, simple	Lacks structure, minimal detail
Inline	Code-adjacent clarity, detailed logic	Not accessible to tools, clutters code
Google	Professional standard, type specs, examples	Verbose, requires more effort

Google-style documentation is most effective for a mathematical utilities library because it is professional, structured, and clear.

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.

```
1  # Docstring Type Documentation
2  def login(user, password, credentials):
3      """
4          Logs in a user by checking if the provided password matches the stored credentials.
5          Parameters:
6              user: The username of the user trying to log in.
7              password: The password provided by the user.
8              credentials: A dictionary containing usernames as keys and their corresponding passwords as values.
9          Returns:
10             True if the login is successful (i.e., the password matches the stored credentials), False otherwise.
11             """
12             return credentials.get(user) == password
13
14
15  # Google Style Docstring
16  def login(user: str, password: str, credentials: dict) -> bool:
17      """
18          Logs in a user by checking if the provided password matches the stored credentials.
19
20          Args:
21              user (str): The username of the user trying to log in.
22              password (str): The password provided by the user.
23              credentials (dict): A dictionary containing usernames as keys and their corresponding passwords
24              as values.
25
26          Returns:
27              bool: True if the login is successful (i.e., the password matches the stored credentials),
28                  False otherwise.
29          Example:
30              >>> credentials = {'alice': 'password123', 'bob': 'securepass'}
31              >>> login('alice', 'password123', credentials)
32              True
33          Raises ValueError: If the user is not found in the credentials.
34          """
35
36
37          if user not in credentials:
38              raise ValueError("User not found in credentials.")
39          return credentials[user] == password
40
41  # Inline Comment
42  def login(user, password, credentials):
43      return credentials.get(user) == password # This line checks if the password provided by the
44      # user matches the password stored in the credentials dictionary for that user
45      #
46      # -----
47      # The function 'login' checks if the provided password matches the stored credentials for a given user.
48      # It uses the 'get' method of the dictionary to retrieve the password associated with the user,
49      # returning True if it matches the provided password and False otherwise.
50      #
51      # -----
```

Point	Inline	Docstring	Google Style
Clear for beginners	No	Yes	Yes, very clear
Structured	No	Somewhat	Yes
Professional	No	Medium	Yes
For big projects	No	Okay	Yes
Best for onboarding	No	Good	Best

New developers need clarity, structure, and consistency.  
 Google style clearly explains inputs, outputs, and possible errors  
 Therefore, for onboarding new developers in a real project,  
 It scales well in real-world production systems.  
 It supports automated documentation generation.  
 It reduces ambiguity in collaborative projects.  
 Google-style documentation is the most helpful approach.

### Problem 3: Calculator (Automatic Documentation Generation)

Task: Design a Python module named calculator.py and demonstrate automatic documentation generation.

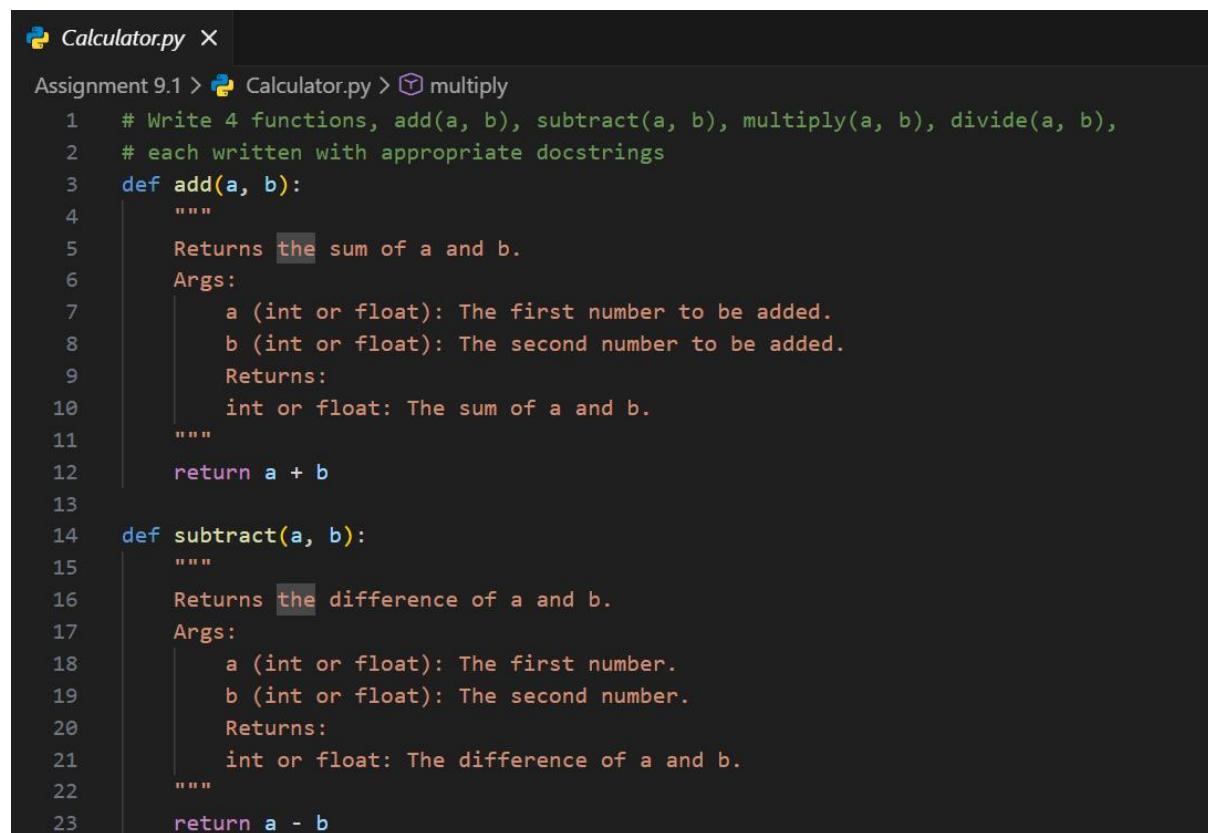
Instructions:

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.



The screenshot shows a code editor window with the file 'Calculator.py' open. The code defines four functions: add, subtract, multiply, and divide. Each function has a docstring that describes its purpose, arguments, and return value. The code editor interface includes a title bar, a toolbar with icons, and a status bar at the bottom.

```
Calculator.py X
Assignment 9.1 > Calculator.py > multiply
1  # Write 4 functions, add(a, b), subtract(a, b), multiply(a, b), divide(a, b),
2  # each written with appropriate docstrings
3  def add(a, b):
4      """
5          Returns the sum of a and b.
6          Args:
7              a (int or float): The first number to be added.
8              b (int or float): The second number to be added.
9          Returns:
10             int or float: The sum of a and b.
11         """
12     return a + b
13
14  def subtract(a, b):
15      """
16          Returns the difference of a and b.
17          Args:
18              a (int or float): The first number.
19              b (int or float): The second number.
20          Returns:
21             int or float: The difference of a and b.
22         """
23     return a - b
```

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

```
PS Z:\AIAC\Assignment 9.1> python -m pydoc .\calculator.py
```

Returns the sum of a and b.

Args:

  a (int or float): The first number to be added.  
  b (int or float): The second number to be added.

Returns:

  int or float: The sum of a and b.

Returns the difference of a and b.

Args:

  a (int or float): The first number.  
  b (int or float): The second number.

Returns:

  int or float: The difference of a and b.

Returns the product of a and b.

Args:

  a (int or float): The first number.  
  b (int or float): The second number.

Returns:

  int or float: The product of a and b.

Returns the quotient of a and b.

Args:

  a (int or float): The dividend.  
  b (int or float): The divisor.

Returns:

  int or float: The quotient of a and b.

Raises:

  ValueError: If b is zero.

```
● PS Z:\AIAC\Assignment 9.1> python -m pydoc -w .\Calculator.py
```

Returns the sum of a and b.

Args:

a (int or float): The first number to be added.

b (int or float): The second number to be added.

Returns:

int or float: The sum of a and b.

Returns the difference of a and b.

Args:

a (int or float): The first number.

b (int or float): The second number.

Returns:

int or float: The difference of a and b.

Returns the product of a and b.

Args:

a (int or float): The first number.

b (int or float): The second number.

Returns:

int or float: The product of a and b.

Returns the quotient of a and b.

Args:

a (int or float): The dividend.

b (int or float): The divisor.

Returns:

int or float: The quotient of a and b.

Raises:

ValueError: If b is zero.

```
wrote calculator.html
```

The screenshot shows a web browser window with the title "Pydoc: module Calculator". The address bar indicates the URL is "localhost:8080/Calculator.html". The page content displays Python code for a module named "Calculator" with four functions: add, subtract, multiply, and divide. Each function includes a docstring describing its purpose, arguments, and return value.

```
Python 3.13.5 [tags/v3.13.5:6cb20a2, MSC v.1943 64 bit (AMD64)]  
Windows-11  
  


## Calculator



```
# Write 4 functions, add\(a, b\), subtract\(a, b\), multiply\(a, b\), divide\(a, b\),  
# each written with appropriate docstrings
```



### Functions



```
add(a, b)  
    Returns the sum of a and b.  
    Args:  
        a (int or float): The first number to be added.  
        b (int or float): The second number to be added.  
    Returns:  
        int or float: The sum of a and b.  
  
divide(a, b)  
    Returns the quotient of a and b.  
    Args:  
        a (int or float): The dividend.  
        b (int or float): The divisor.  
    Returns:  
        int or float: The quotient of a and b.  
    Raises:  
        ValueError: If b is zero.  
  
multiply(a, b)  
    Returns the product of a and b.  
    Args:  
        a (int or float): The first number.  
        b (int or float): The second number.  
    Returns:  
        int or float: The product of a and b.  
  
subtract(a, b)  
    Returns the difference of a and b.  
    Args:  
        a (int or float): The first number.  
        b (int or float): The second number.  
    Returns:  
        int or float: 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.

```
conversion.py X
Assignment 9.1 > conversion.py ...
1  # Write 3 functions decimal_to_binary(n), binary_to_decimal(s), and decimal_to_hexadecimal(n),
2  # each written with appropriate docstrings
3  def decimal_to_binary(n):
4      """
5          Converts a decimal number to its binary representation.
6          Args:
7              n (int): The decimal number to be converted.
8          Returns:
9              str: The binary representation of the decimal number.
10         """
11        if n < 0:
12            raise ValueError("Input must be a non-negative integer.")
13        return bin(n)[2:]
14  def binary_to_decimal(s):
15      """
16          Converts a binary string to its decimal representation.
17          Args:
18              s (str): The binary string to be converted.
19          Returns:
20              int: The decimal representation of the binary string.
21         """
22        if not all(char in '01' for char in s):
23            raise ValueError("Input must be a valid binary string.")
24        return int(s, 2)
25  def decimal_to_hexadecimal(n):
26      """
27          Converts a decimal number to its hexadecimal representation.
28          Args:
29              n (int): The decimal number to be converted.
30          Returns:
31              str: The hexadecimal representation of the decimal number.
32         """
33        if n < 0:
34            raise ValueError("Input must be a non-negative integer.")
35        return hex(n)[2:].upper()
```

```
● PS Z:\AIAC\Assignment 9.1> python -m pydoc .\conversion.py

    Converts a decimal number to its binary representation.
    Args:
        n (int): The decimal number to be converted.
    Returns:
        str: The binary representation of the decimal number.

    Converts a binary string to its decimal representation.
    Args:
        s (str): The binary string to be converted.
    Returns:
        int: The decimal representation of the binary string.

    Converts a decimal number to its hexadecimal representation.
    Args:
        n (int): The decimal number to be converted.
    Returns:
        str: The hexadecimal representation of the decimal number.

Help on module conversion:

NAME
    conversion

DESCRIPTION
    # Write 3 functions decimal_to_binary(n), binary_to_decimal(s), and decimal_to_hexadecimal(n),
    # each written with appropriate docstrings

FUNCTIONS
    binary_to_decimal(s)
        Converts a binary string to its decimal representation.
        Args:
            s (str): The binary string to be converted.
        Returns:
            int: The decimal representation of the binary string.

    decimal_to_binary(n)
        Converts a decimal number to its binary representation.
        Args:
            n (int): The decimal number to be converted.
        Returns:
            str: The binary representation of the decimal number.

    decimal_to_hexadecimal(n)
        Converts a decimal number to its hexadecimal representation.
        Args:
            n (int): The decimal number to be converted.
        Returns:
            str: The hexadecimal representation of the decimal number.

FILE
    z:\aiac\assignment 9.1\conversion.py
```

```
● PS Z:\AIAC\Assignment 9.1> python -m pydoc -w .\conversion.py
```

Converts a decimal number to its binary representation.

Args:

n (int): The decimal number to be converted.

Returns:

str: The binary representation of the decimal number.

Converts a binary string to its decimal representation.

Args:

s (str): The binary string to be converted.

Returns:

int: The decimal representation of the binary string.

Converts a decimal number to its hexadecimal representation.

Args:

n (int): The decimal number to be converted.

Returns:

str: The hexadecimal representation of the decimal number.

wrote conversion.html

The screenshot shows a web browser window displaying the generated documentation for the `conversion` module. The title bar reads "Pydoc: module conversion". The address bar shows the URL `localhost:8080/conversion.html`. The page header indicates the Python version is 3.13.5 and the build date is v3.13.5:6cb20a2, MSC v.1943 64 bit (AMD64). The main content area has a blue header "conversion". Below it, a section titled "Functions" contains three function definitions: `binary_to_decimal(s)`, `decimal_to_binary(n)`, and `decimal_to_hexadecimal(n)`. Each function includes its docstring, which describes the purpose, arguments, and return value.

```
# Write 3 functions decimal\_to\_binary\(n\), binary\_to\_decimal\(s\), and decimal\_to\_hexadecimal\(n\),  
# each written with appropriate docstrings
```

## Functions

**`binary_to_decimal(s)`**  
Converts a binary string to its decimal representation.  
Args:  
s (str): The binary string to be converted.  
Returns:  
int: The decimal representation of the binary string.

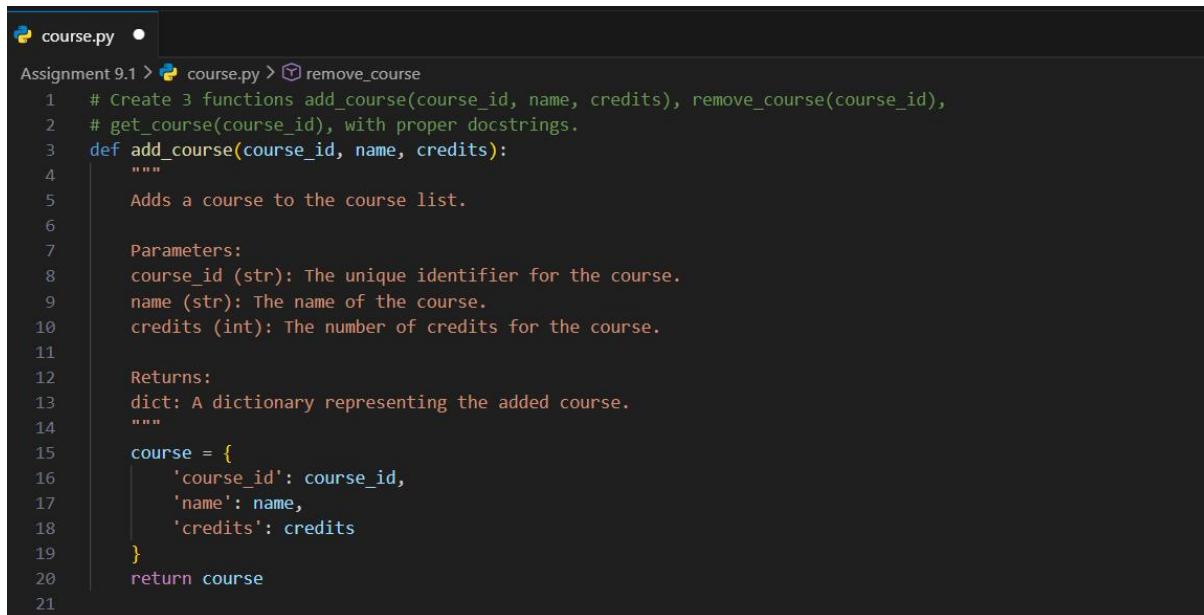
**`decimal_to_binary(n)`**  
Converts a decimal number to its binary representation.  
Args:  
n (int): The decimal number to be converted.  
Returns:  
str: The binary representation of the decimal number.

**`decimal_to_hexadecimal(n)`**  
Converts a decimal number to its hexadecimal representation.  
Args:  
n (int): The decimal number to be converted.  
Returns:  
str: The hexadecimal representation of the decimal number.

## 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.



A screenshot of a code editor window titled "course.py". The window shows Python code for a "course" module. The code defines three functions: add\_course, remove\_course, and get\_course. The add\_course function is annotated with a detailed docstring, including parameters (course\_id, name, credits) and return type (dict). The code uses triple quotes for the docstring and includes a detailed description of each parameter and the return value. The code editor interface includes a toolbar at the top and a status bar at the bottom.

```
Assignment 9.1 > course.py > remove_course
1 # Create 3 functions add_course(course_id, name, credits), remove_course(course_id),
2 # get_course(course_id), with proper docstrings.
3 def add_course(course_id, name, credits):
4     """
5     Adds a course to the course list.
6
7     Parameters:
8     course_id (str): The unique identifier for the course.
9     name (str): The name of the course.
10    credits (int): The number of credits for the course.
11
12    Returns:
13    dict: A dictionary representing the added course.
14    """
15    course = {
16        'course_id': course_id,
17        'name': name,
18        'credits': credits
19    }
20    return course
21
```

```

22     def remove_course(course_id, course_list):
23         """
24             Removes a course from the course list.
25
26             Parameters:
27                 course_id (str): The unique identifier for the course to be removed.
28                 course_list (list): The list of courses from which to remove the course.
29
30             Returns:
31                 bool: True if the course was successfully removed, False otherwise.
32                 """
33             for course in course_list:
34                 if course['course_id'] == course_id:
35                     course_list.remove(course)
36                     return True
37             return False
38
39     def get_course(course_id, course_list):
40         """
41             Retrieves a course from the course list.
42
43             Parameters:
44                 course_id (str): The unique identifier for the course to be retrieved.
45                 course_list (list): The list of courses from which to retrieve the course.
46
47             Returns:
48                 dict: A dictionary representing the retrieved course, or None if the course is not found.
49                 """
50             for course in course_list:
51                 if course['course_id'] == course_id:
52                     return course
53             return None

```

PS Z:\AIAC\Assignment 9.1> python -m pydoc .\course.py

Adds a course to the course list.

Parameters:

course\_id (str): The unique identifier for the course.

name (str): The name of the course.

credits (int): The number of credits for the course.

Returns:

dict: A dictionary representing the added course.

Removes a course from the course list.

Parameters:

course\_id (str): The unique identifier for the course to be removed.

course\_list (list): The list of courses from which to remove the course.

Returns:

bool: True if the course was successfully removed, False otherwise.

Retrieves a course from the course list.

Parameters:

course\_id (str): The unique identifier for the course to be retrieved.

course\_list (list): The list of courses from which to retrieve the course.

```
● PS Z:\AIAC\Assignment 9.1> python -m pydoc -w .\course.py
```

Adds a course to the course list.

Parameters:

course\_id (str): The unique identifier for the course.

name (str): The name of the course.

credits (int): The number of credits for the course.

Returns:

dict: A dictionary representing the added course.

Removes a course from the course list.

Parameters:

course\_id (str): The unique identifier for the course to be removed.

course\_list (list): The list of courses from which to remove the course.

Returns:

bool: True if the course was successfully removed, False otherwise.

Retrieves a course from the course list.

Parameters:

course\_id (str): The unique identifier for the course to be retrieved.

course\_list (list): The list of courses from which to retrieve the course.

Returns:

dict: A dictionary representing the retrieved course, or None if the course is not found.

wrote course.html

Pydoc: module course

localhost:8080/course.html

## course

```
# Create 3 functions add\_course(course_id, name, credits), remove\_course(course_id),
# get\_course(course_id), with proper docstrings.
```

### Functions

**[add\\_course](#)(course\_id, name, credits)**

Adds a course to the course list.

**Parameters:**

- course\_id (str): The unique identifier for the course.
- name (str): The name of the course.
- credits (int): The number of credits for the course.

**Returns:**

- dict: A dictionary representing the added course.

**[get\\_course](#)(course\_id, course\_list)**

Retrieves a course from the course list.

**Parameters:**

- course\_id (str): The unique identifier for the course to be retrieved.
- course\_list (list): The list of courses from which to retrieve the course.

**Returns:**

- dict: A dictionary representing the retrieved course, or None if the course is not found.

**[remove\\_course](#)(course\_id, course\_list)**

Removes a course from the course list.

**Parameters:**

- course\_id (str): The unique identifier for the course to be removed.
- course\_list (list): The list of courses from which to remove the course.

**Returns:**

- bool: True if the course was successfully removed, False otherwise.