Lab Assignment-9

Name:Anand

Hallticket:2303A51090

Batch-02

Problem 1:

Consider the following Python function:

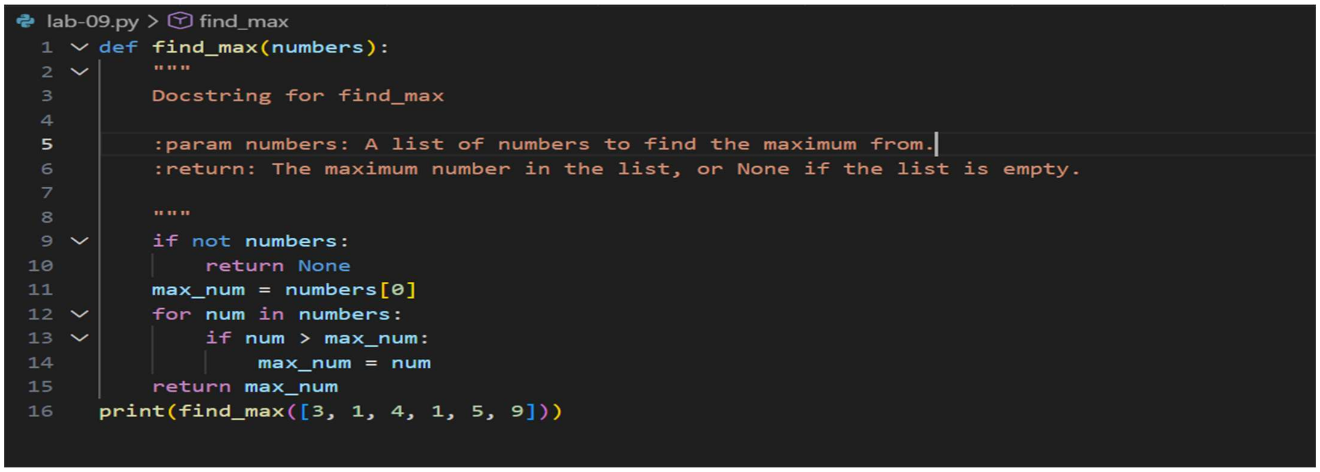
```
def find_max(numbers):
    return max(numbers)
```

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.

Screenshots:

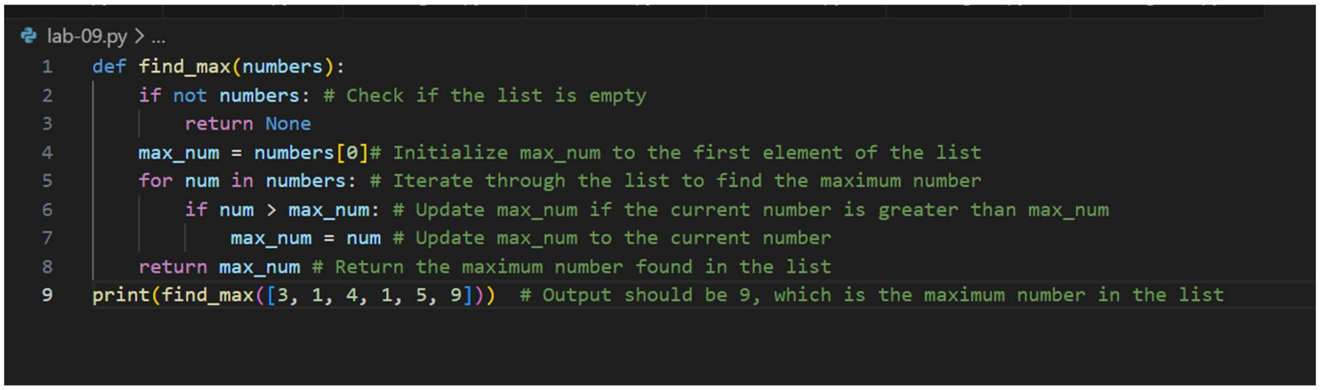
A:docstring



lab-09.py > find_max

```python
def find_max(numbers):
    """
    Docstring for find_max

    :param numbers: A list of numbers to find the maximum from.
    :return: The maximum number in the list, or None if the list is empty.

    """
    if not numbers:
        return None
    max_num = numbers[0]
    for num in numbers:
        if num > max_num:
            max_num = num
    return max_num
print(find_max([3, 1, 4, 1, 5, 9]))
```

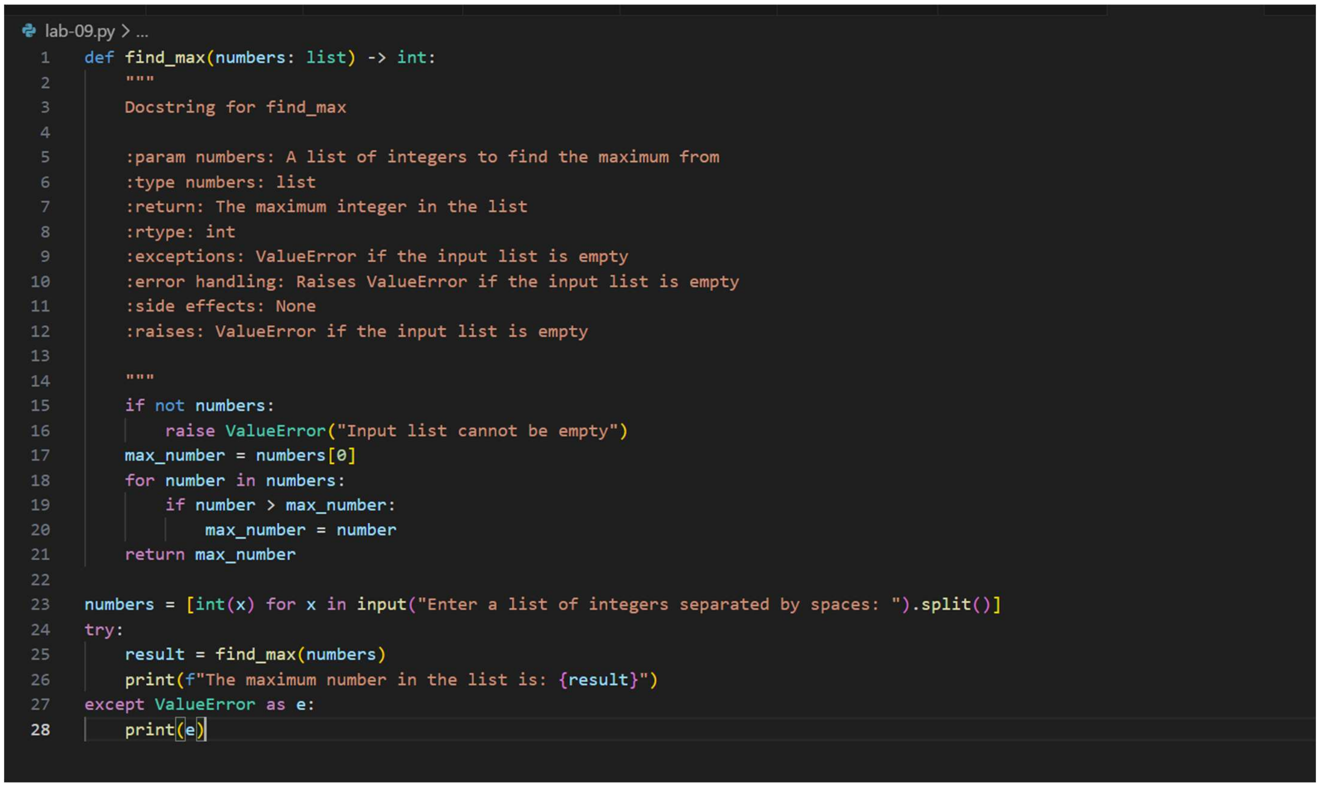
B:in line comments



lab-09.py > ...

```python
def find_max(numbers):
    if not numbers: # Check if the list is empty
        return None
    max_num = numbers[0]# Initialize max_num to the first element of the list
    for num in numbers: # Iterate through the list to find the maximum number
        if num > max_num: # Update max_num if the current number is greater than max_num
            max_num = num # Update max_num to the current number
    return max_num # Return the maximum number found in the list
print(find_max([3, 1, 4, 1, 5, 9]))  # Output should be 9, which is the maximum number in the list
```

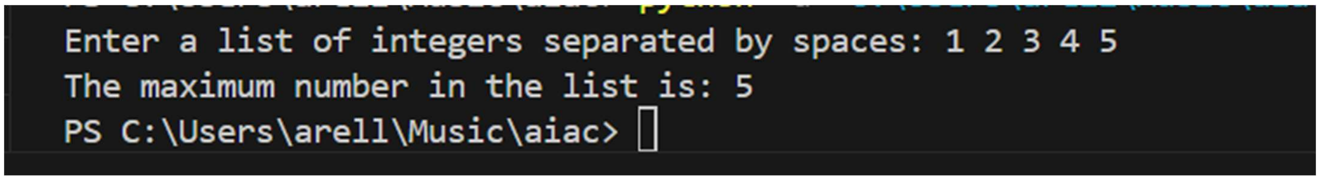
C: Google style



lab-09.py > ...

```python
def find_max(numbers: list) -> int:
    """
    Docstring for find_max

    :param numbers: A list of integers to find the maximum from
    :type numbers: list
    :return: The maximum integer in the list
    :rtype: int
    :exceptions: ValueError if the input list is empty
    :error handling: Raises ValueError if the input list is empty
    :side effects: None
    :raises: ValueError if the input list is empty

    """
    if not numbers:
        raise ValueError("Input list cannot be empty")
    max_number = numbers[0]
    for number in numbers:
        if number > max_number:
            max_number = number
    return max_number

numbers = [int(x) for x in input("Enter a list of integers separated by spaces: ").split()]
try:
    result = find_max(numbers)
    print(f"The maximum number in the list is: {result}")
except ValueError as e:
    print(e)
```

Output:



```
Enter a list of integers separated by spaces: 1 2 3 4 5
The maximum number in the list is: 5
PS C:\Users\arell\Music\aiac>
```

<u>Problem 2: Consider the following Python function:</u>

def login(user, password, credentials):

return credentials.get(user) == password

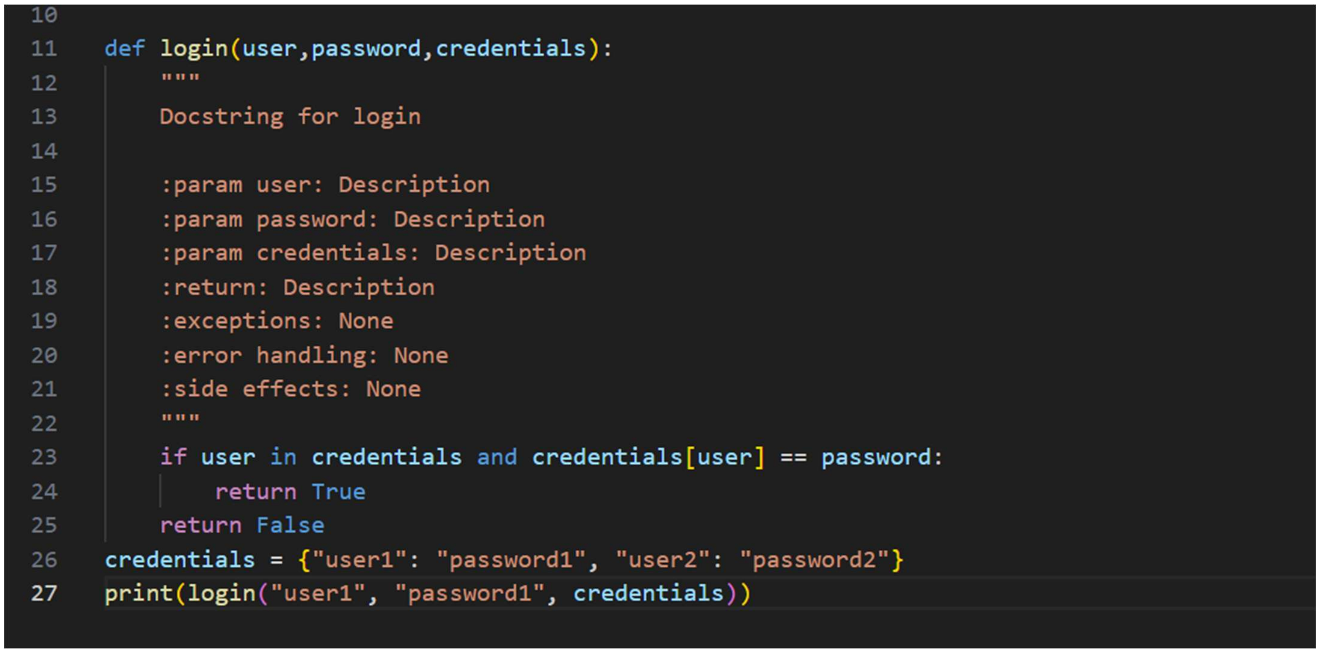
Task:

1. Write documentation in all three formats.

2. Critically compare the approaches.

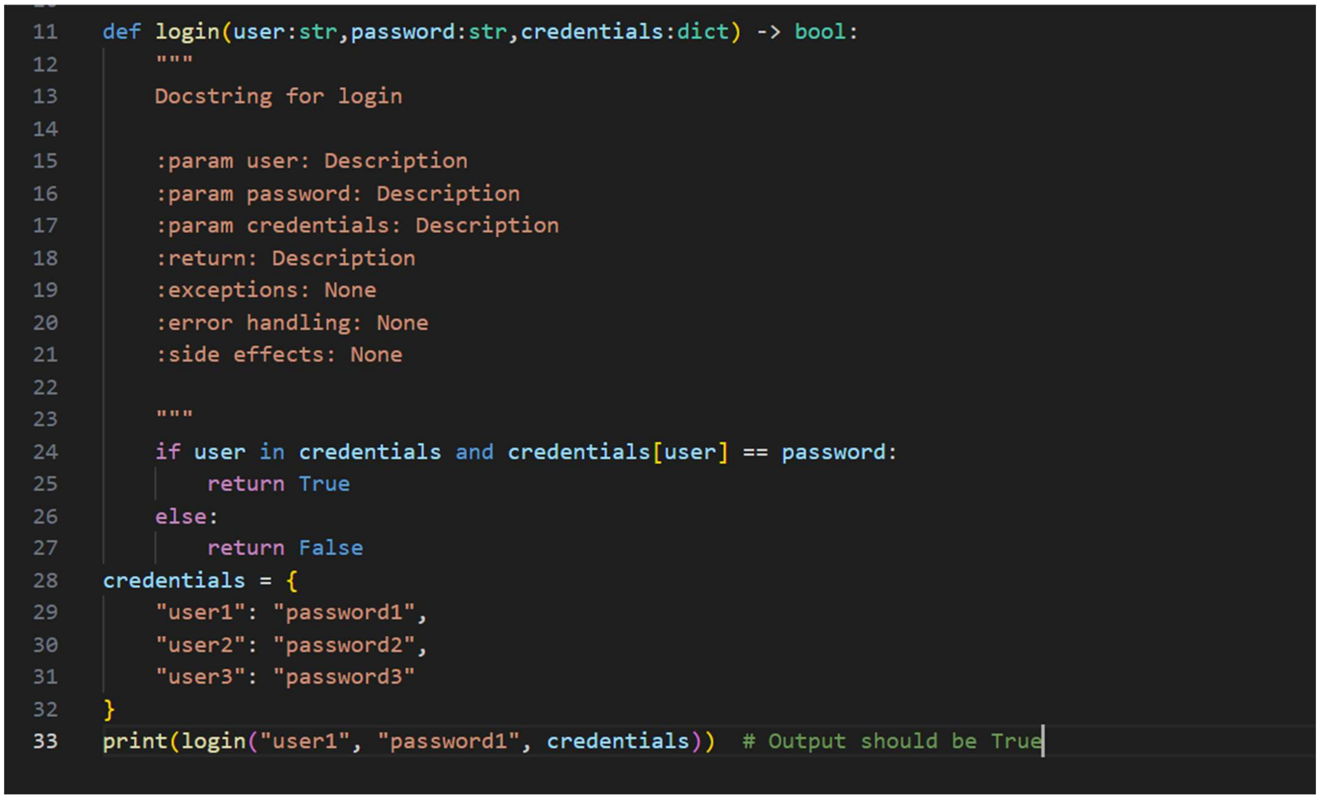
3. Recommend which style would be most helpful for new developers onboarding a project, and justify your choice.

A:docstring:



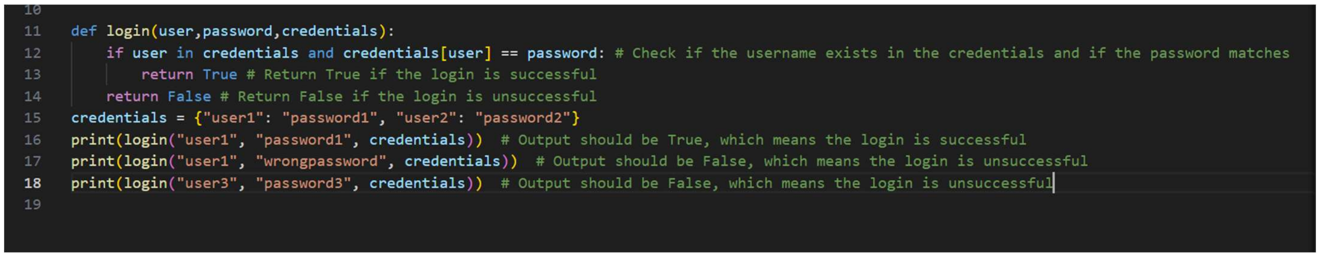
```python

def login(user,password,credentials):
    """
    Docstring for login

    :param user: Description
    :param password: Description
    :param credentials: Description
    :return: Description
    :exceptions: None
    :error handling: None
    :side effects: None
    """
    if user in credentials and credentials[user] == password:
        return True
    return False
credentials = {"user1": "password1", "user2": "password2"}
print(login("user1", "password1", credentials))
```

B: google style



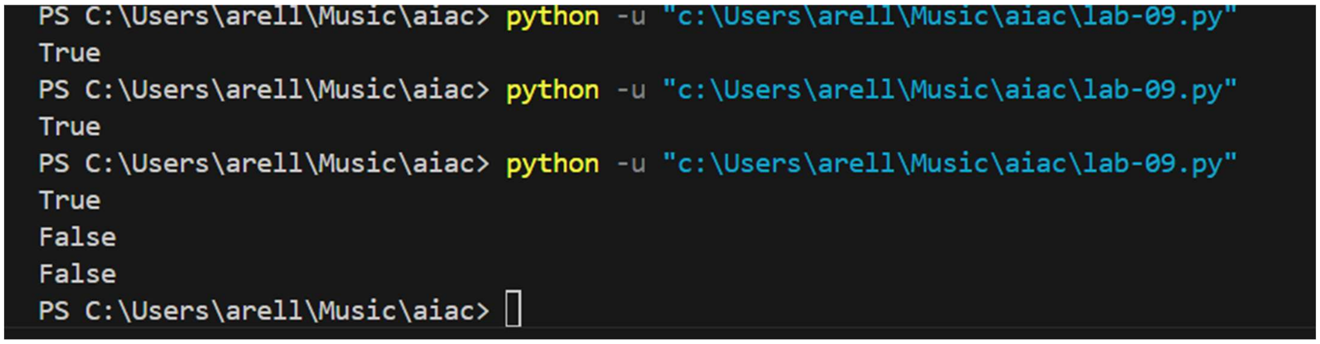
```python
def login(user:str,password:str,credentials:dict) -> bool:
    """
    Docstring for login

    :param user: Description
    :param password: Description
    :param credentials: Description
    :return: Description
    :exceptions: None
    :error handling: None
    :side effects: None

    """
    if user in credentials and credentials[user] == password:
        return True
    else:
        return False
credentials = {
    "user1": "password1",
    "user2": "password2",
    "user3": "password3"
}
print(login("user1", "password1", credentials))  # Output should be True
```

C:Inline comments



```python
def login(user,password,credentials):
    if user in credentials and credentials[user] == password: # Check if the username exists in the credentials and if the password matches
        return True # Return True if the login is successful
    return False # Return False if the login is unsuccessful
credentials = {"user1": "password1", "user2": "password2"}
print(login("user1", "password1", credentials))  # Output should be True, which means the login is successful
print(login("user1", "wrongpassword", credentials))  # Output should be False, which means the login is unsuccessful
print(login("user3", "password3", credentials))  # Output should be False, which means the login is unsuccessful
```

Output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\lab-09.py"
True
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\lab-09.py"
True
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\lab-09.py"
True
False
False
PS C:\Users\arell\Music\aiac>
```

__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

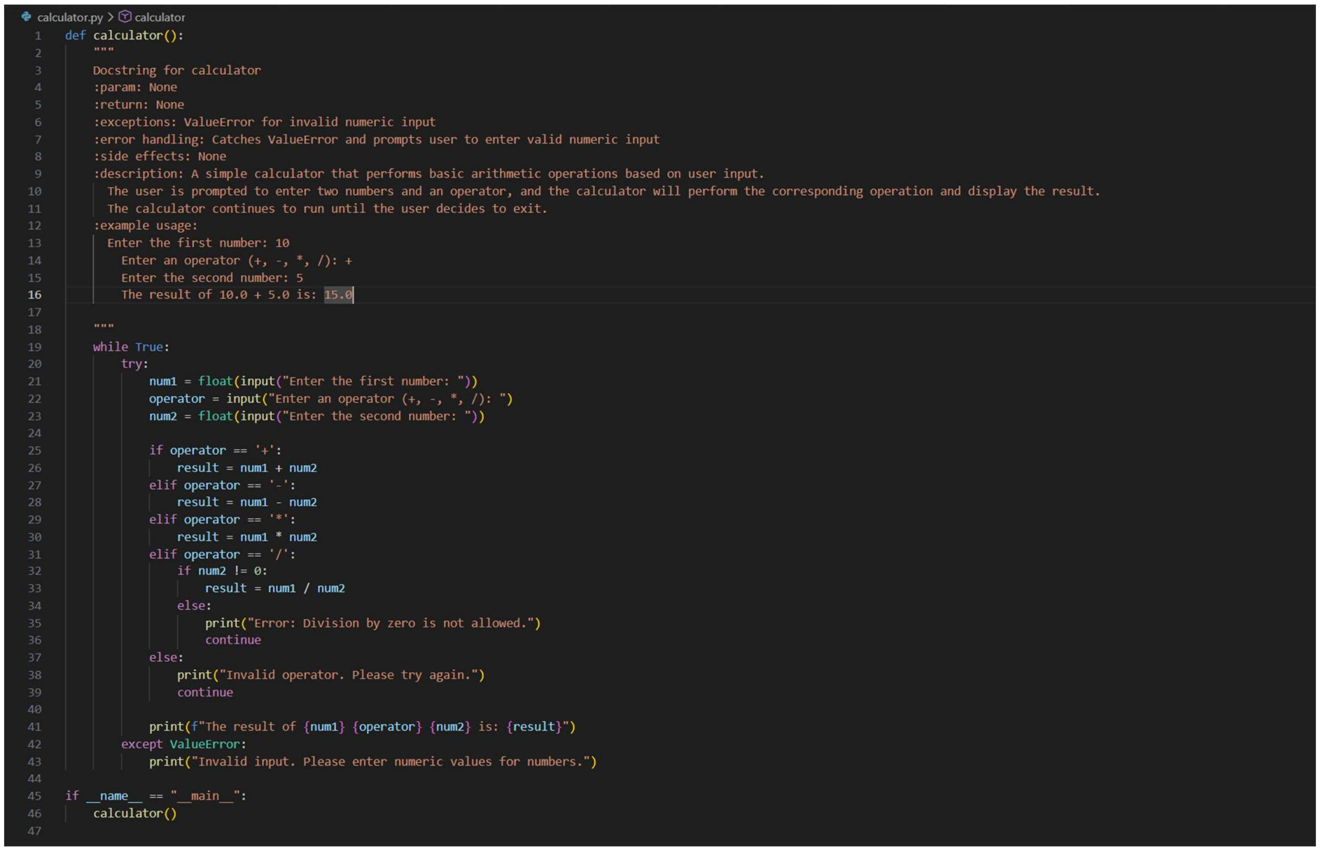
    o multiply(a, b) – returns the product of two numbers

    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.

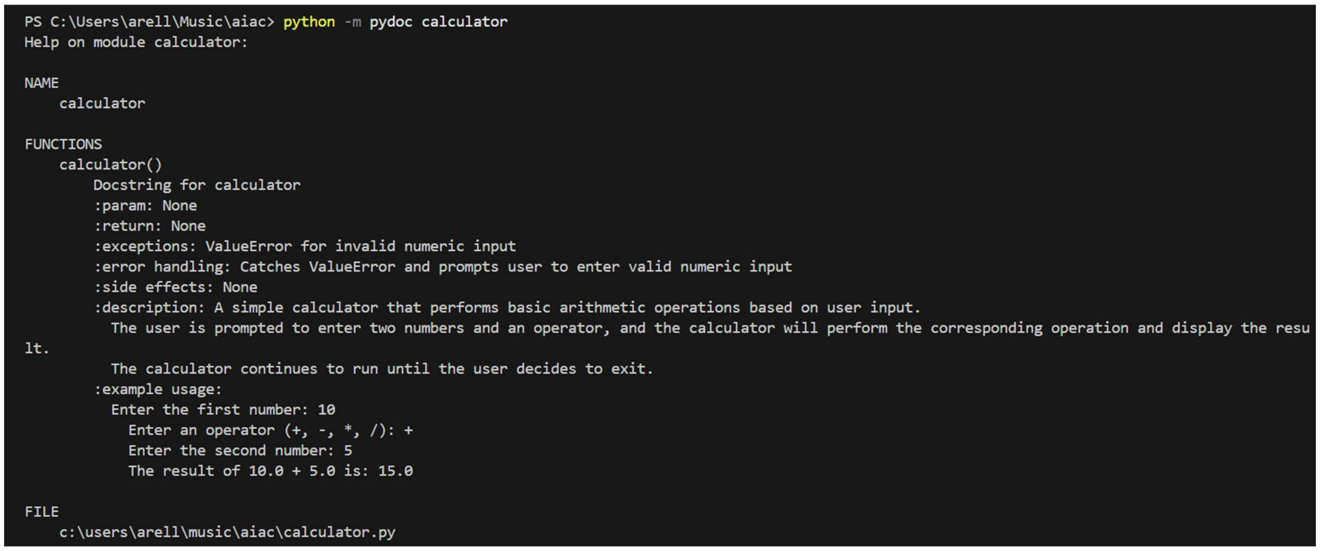
Screenshots:



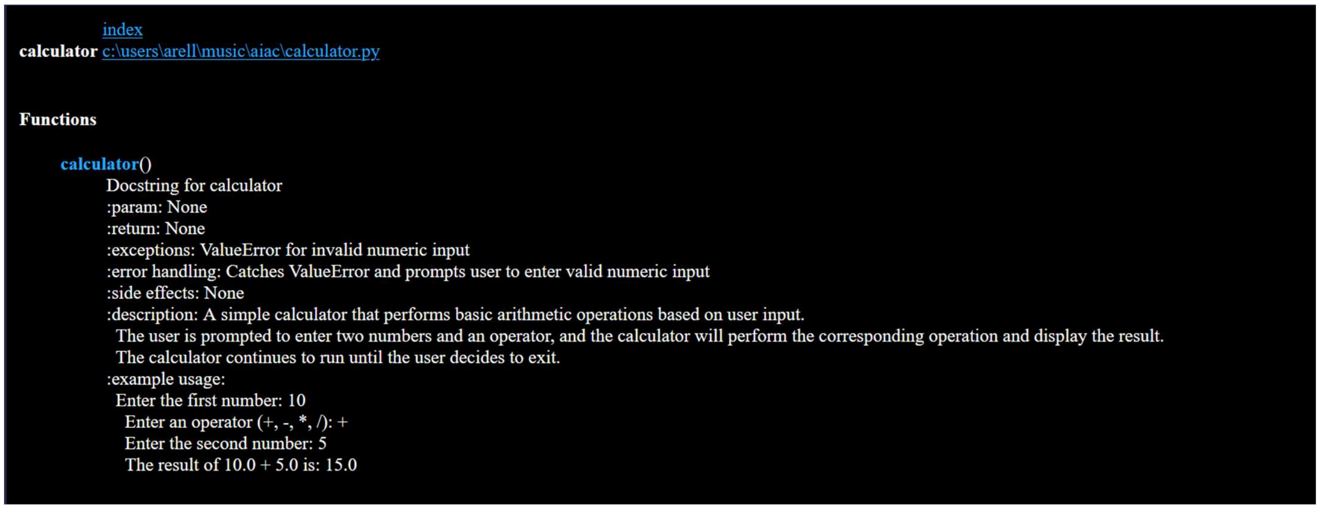
calculator.py > calculator

```python
def calculator():
    """
    Docstring for calculator
    :param: None
    :return: None
    :exceptions: ValueError for invalid numeric input
    :error handling: Catches ValueError and prompts user to enter valid numeric input
    :side effects: None
    :description: A simple calculator that performs basic arithmetic operations based on user input.
      The user is prompted to enter two numbers and an operator, and the calculator will perform the corresponding operation and display the result.
      The calculator continues to run until the user decides to exit.
    :example usage:
      Enter the first number: 10
        Enter an operator (+, -, *, /): +
        Enter the second number: 5
        The result of 10.0 + 5.0 is: 15.0

    """
    while True:
        try:
            num1 = float(input("Enter the first number: "))
            operator = input("Enter an operator (+, -, *, /): ")
            num2 = float(input("Enter the second number: "))

            if operator == '+':
                result = num1 + num2
            elif operator == '-':
                result = num1 - num2
            elif operator == '*':
                result = num1 * num2
            elif operator == '/':
                if num2 != 0:
                    result = num1 / num2
                else:
                    print("Error: Division by zero is not allowed.")
                    continue
            else:
                print("Invalid operator. Please try again.")
                continue

            print(f"The result of {num1} {operator} {num2} is: {result}")
        except ValueError:
            print("Invalid input. Please enter numeric values for numbers.")

if __name__ == "__main__":
    calculator()
```

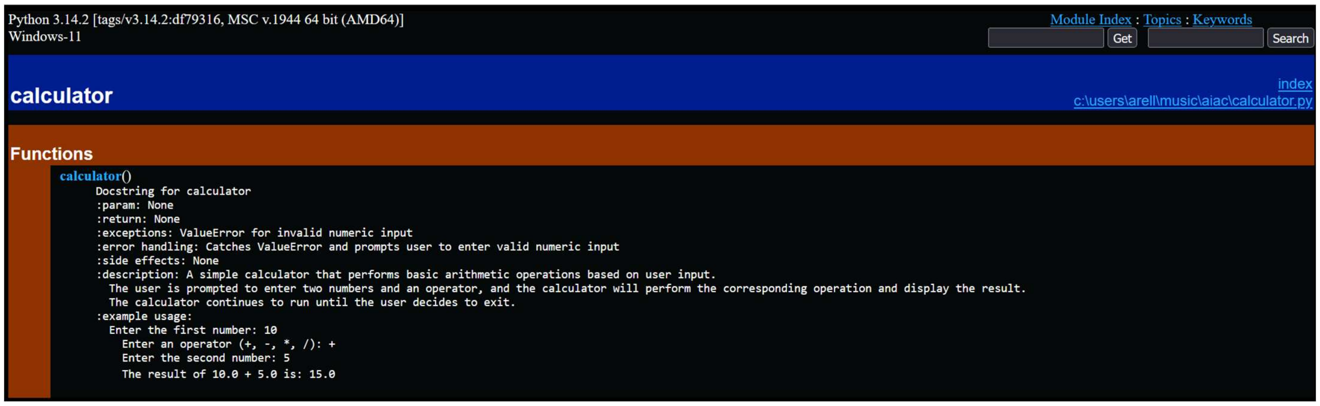
Output:



```
PS C:\Users\arell\Music\aiac> python -m pydoc calculator
Help on module calculator:

NAME
    calculator

FUNCTIONS
    calculator()
        Docstring for calculator
        :param: None
        :return: None
        :exceptions: ValueError for invalid numeric input
        :error handling: Catches ValueError and prompts user to enter valid numeric input
        :side effects: None
        :description: A simple calculator that performs basic arithmetic operations based on user input.
          The user is prompted to enter two numbers and an operator, and the calculator will perform the corresponding operation and display the resu
lt.
          The calculator continues to run until the user decides to exit.
        :example usage:
          Enter the first number: 10
            Enter an operator (+, -, *, /): +
            Enter the second number: 5
            The result of 10.0 + 5.0 is: 15.0

FILE
    c:\users\arell\music\aiac\calculator.py
```



```
        index
calculator c:\users\arell\music\aiac\calculator.py


Functions

    calculator()
        Docstring for calculator
        :param: None
        :return: None
        :exceptions: ValueError for invalid numeric input
        :error handling: Catches ValueError and prompts user to enter valid numeric input
        :side effects: None
        :description: A simple calculator that performs basic arithmetic operations based on user input.
          The user is prompted to enter two numbers and an operator, and the calculator will perform the corresponding operation and display the result.
          The calculator continues to run until the user decides to exit.
        :example usage:
          Enter the first number: 10
            Enter an operator (+, -, *, /): +
            Enter the second number: 5
            The result of 10.0 + 5.0 is: 15.0
```



Python 3.14.2 [tags/v3.14.2:df79316, MSC v.1944 64 bit (AMD64)]
Windows-11
Module Index : Topics : Keywords
Get
Search

```
calculator                                              index
                         c:\users\arell\music\aiac\calculator.py

Functions
    calculator()
        Docstring for calculator
        :param: None
        :return: None
        :exceptions: ValueError for invalid numeric input
        :error handling: Catches ValueError and prompts user to enter valid numeric input
        :side effects: None
        :description: A simple calculator that performs basic arithmetic operations based on user input.
          The user is prompted to enter two numbers and an operator, and the calculator will perform the corresponding operation and display the result.
          The calculator continues to run until the user decides to exit.
        :example usage:
          Enter the first number: 10
            Enter an operator (+, -, *, /): +
            Enter the second number: 5
            The result of 10.0 + 5.0 is: 15.0
```

__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)

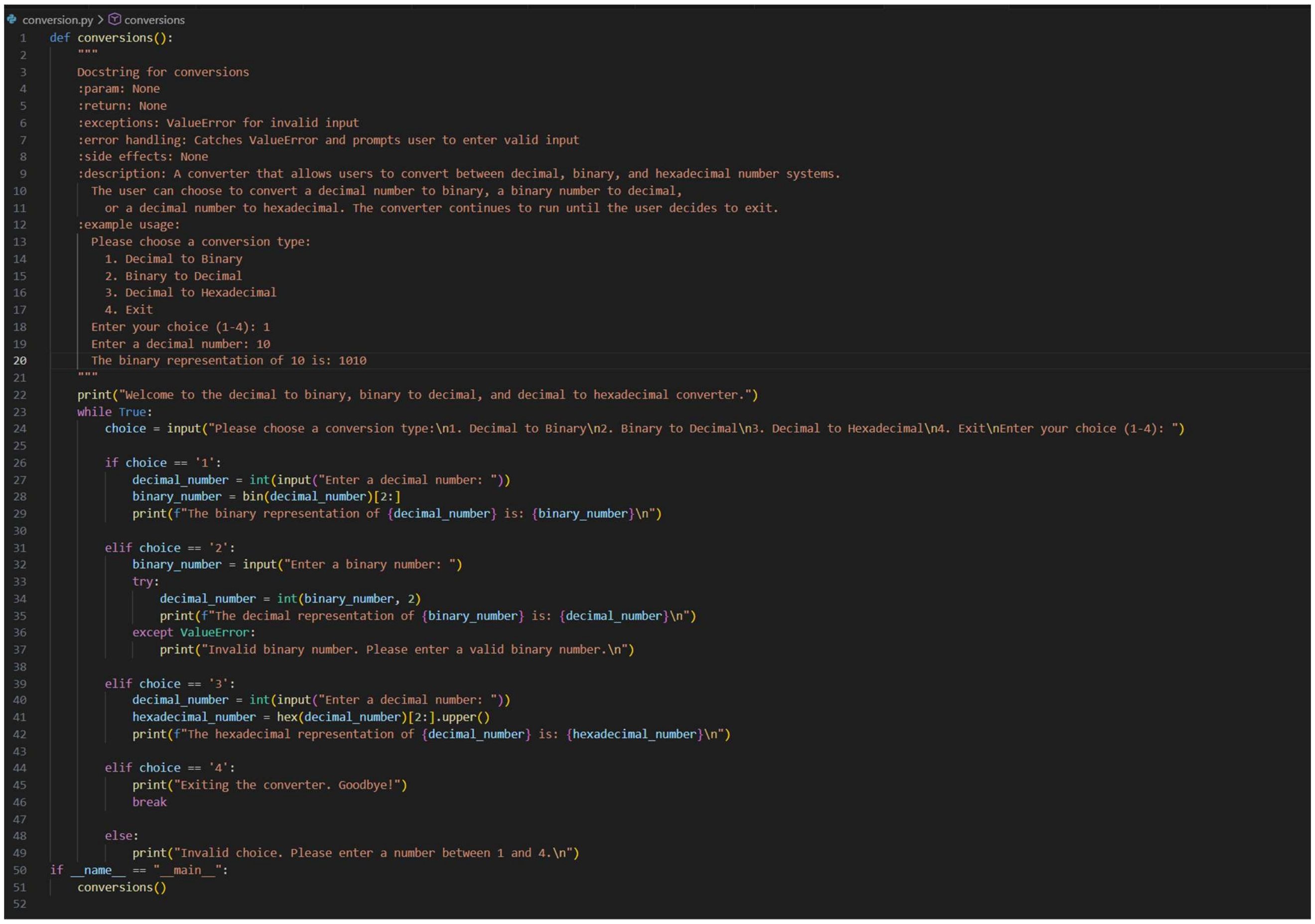
    o decimal_to_hexadecimal(n)

2. Use Copilot for auto-generating docstrings.

3. Generate documentation in the terminal.

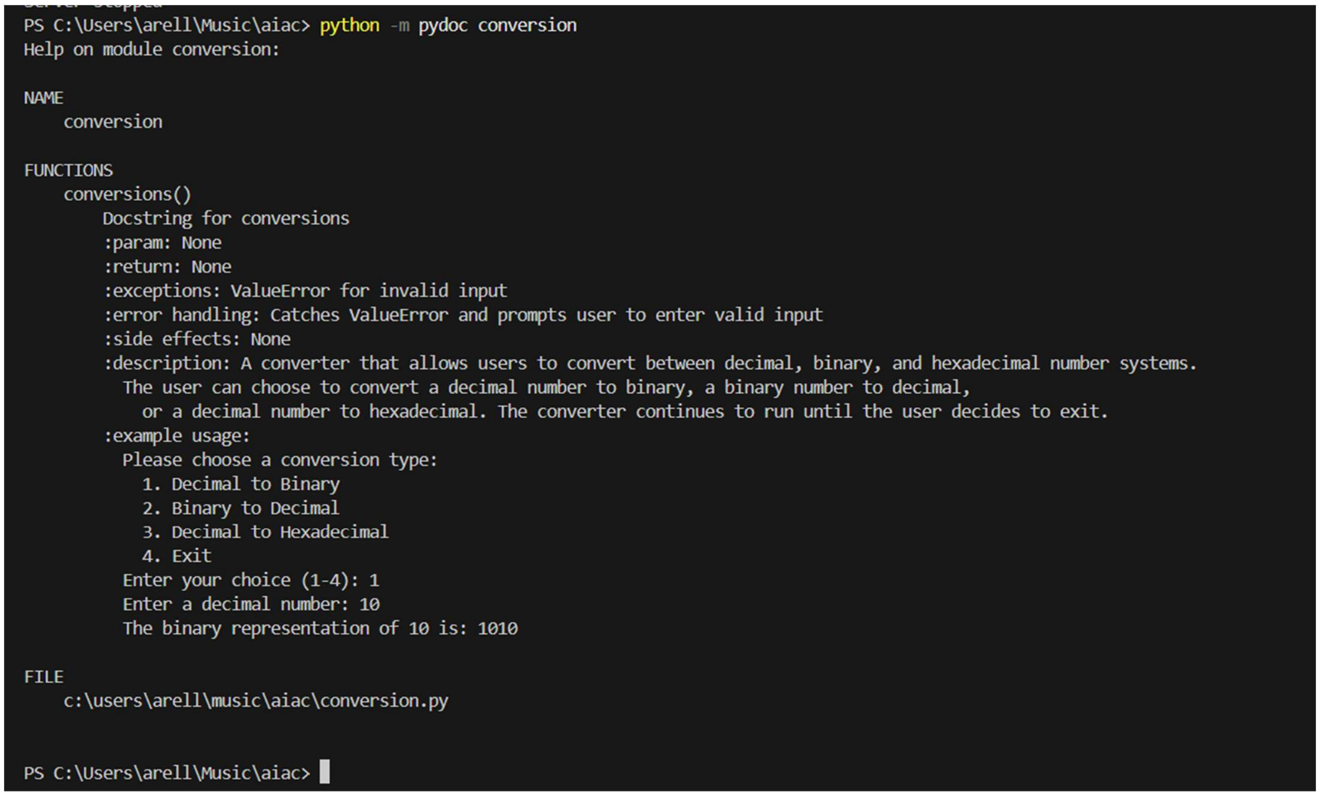
4. Export the documentation in HTML format and open it in a browser.

Screenshots:

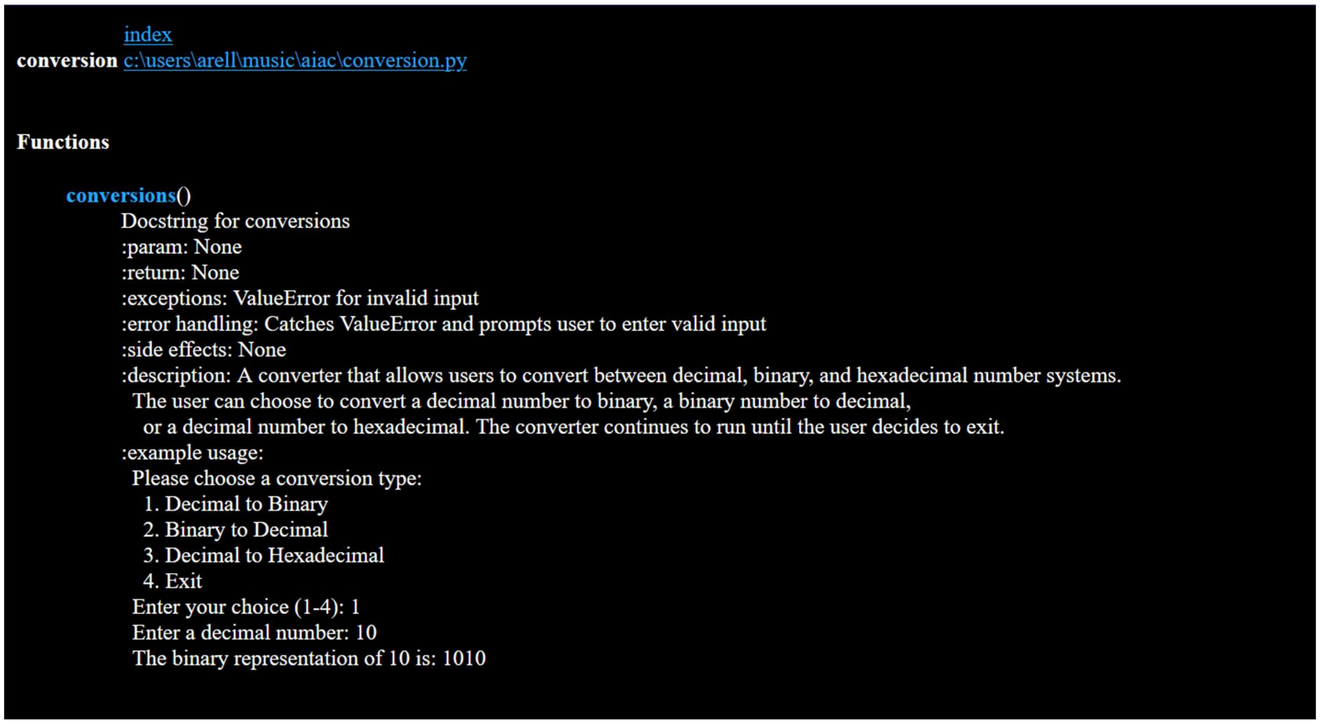


```
conversion.py > conversions
def conversions():
    """
    Docstring for conversions
    :param: None
    :return: None
    :exceptions: ValueError for invalid input
    :error handling: Catches ValueError and prompts user to enter valid input
    :side effects: None
    :description: A converter that allows users to convert between decimal, binary, and hexadecimal number systems.
      The user can choose to convert a decimal number to binary, a binary number to decimal,
        or a decimal number to hexadecimal. The converter continues to run until the user decides to exit.
    :example usage:
      Please choose a conversion type:
        1. Decimal to Binary
        2. Binary to Decimal
        3. Decimal to Hexadecimal
        4. Exit
      Enter your choice (1-4): 1
      Enter a decimal number: 10
      The binary representation of 10 is: 1010
    """
    print("Welcome to the decimal to binary, binary to decimal, and decimal to hexadecimal converter.")
    while True:
        choice = input("Please choose a conversion type:\n1. Decimal to Binary\n2. Binary to Decimal\n3. Decimal to Hexadecimal\n4. Exit\nEnter your choice (1-4): ")

        if choice == '1':
            decimal_number = int(input("Enter a decimal number: "))
            binary_number = bin(decimal_number)[2:]
            print(f"The binary representation of {decimal_number} is: {binary_number}\n")

        elif choice == '2':
            binary_number = input("Enter a binary number: ")
            try:
                decimal_number = int(binary_number, 2)
                print(f"The decimal representation of {binary_number} is: {decimal_number}\n")
            except ValueError:
                print("Invalid binary number. Please enter a valid binary number.\n")

        elif choice == '3':
            decimal_number = int(input("Enter a decimal number: "))
            hexadecimal_number = hex(decimal_number)[2:].upper()
            print(f"The hexadecimal representation of {decimal_number} is: {hexadecimal_number}\n")

        elif choice == '4':
            print("Exiting the converter. Goodbye!")
            break

        else:
            print("Invalid choice. Please enter a number between 1 and 4.\n")
if __name__ == "__main__":
    conversions()

```

Output:



```
PS C:\Users\arell\Music\aiac> python -m pydoc conversion
Help on module conversion:

NAME
    conversion

FUNCTIONS
    conversions()
        Docstring for conversions
        :param: None
        :return: None
        :exceptions: ValueError for invalid input
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: A converter that allows users to convert between decimal, binary, and hexadecimal number systems.
          The user can choose to convert a decimal number to binary, a binary number to decimal,
            or a decimal number to hexadecimal. The converter continues to run until the user decides to exit.
        :example usage:
          Please choose a conversion type:
            1. Decimal to Binary
            2. Binary to Decimal
            3. Decimal to Hexadecimal
            4. Exit
          Enter your choice (1-4): 1
          Enter a decimal number: 10
          The binary representation of 10 is: 1010

FILE
    c:\users\arell\music\aiac\conversion.py


PS C:\Users\arell\Music\aiac>
```



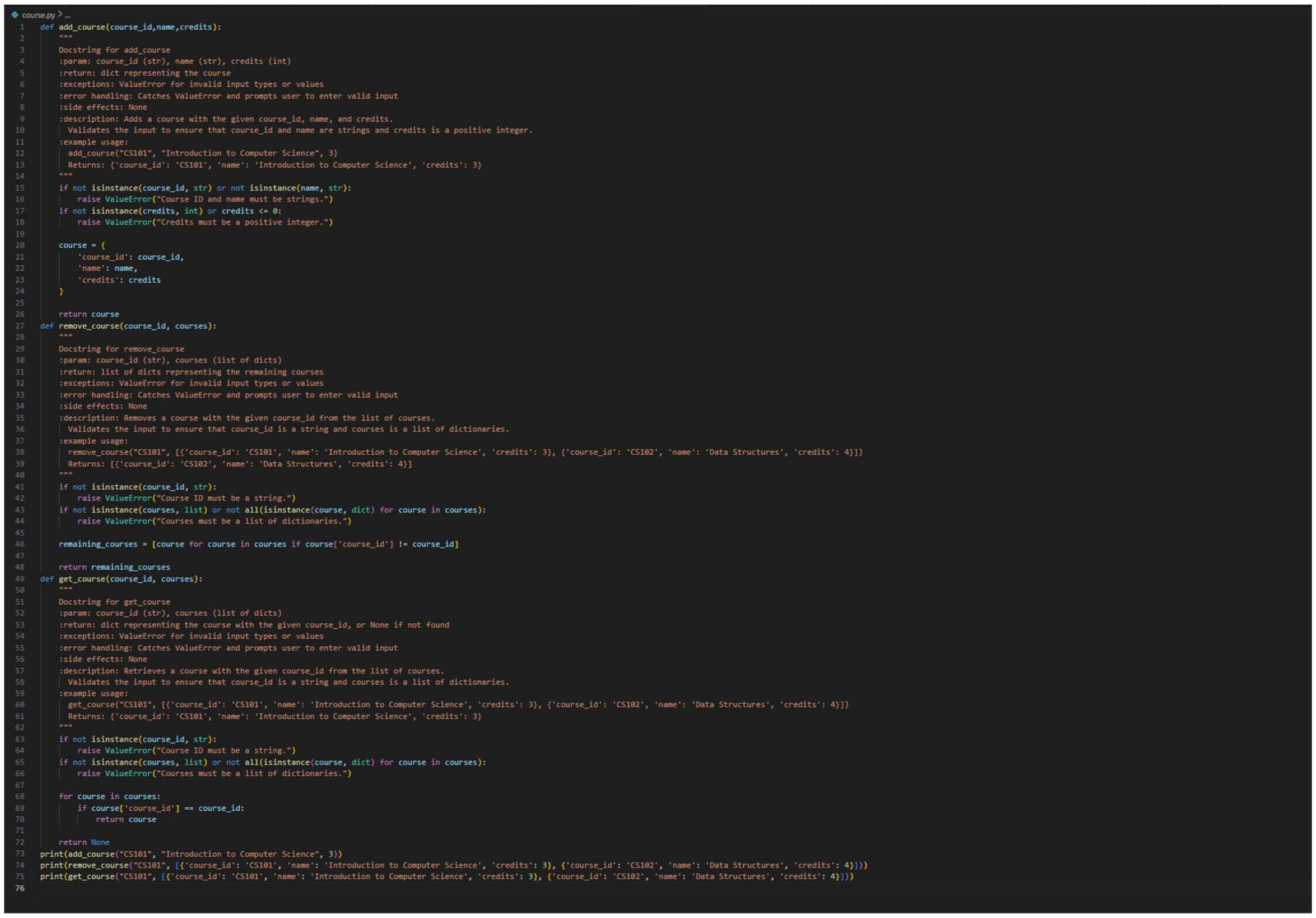
```
            index
conversion c:\users\arell\music\aiac\conversion.py


Functions

    conversions()
        Docstring for conversions
        :param: None
        :return: None
        :exceptions: ValueError for invalid input
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: A converter that allows users to convert between decimal, binary, and hexadecimal number systems.
          The user can choose to convert a decimal number to binary, a binary number to decimal,
            or a decimal number to hexadecimal. The converter continues to run until the user decides to exit.
        :example usage:
          Please choose a conversion type:
            1. Decimal to Binary
            2. Binary to Decimal
            3. Decimal to Hexadecimal
            4. Exit
          Enter your choice (1-4): 1
          Enter a decimal number: 10
          The binary representation of 10 is: 1010
```

<u>Problem 5 – Course Management Module</u>

Task:

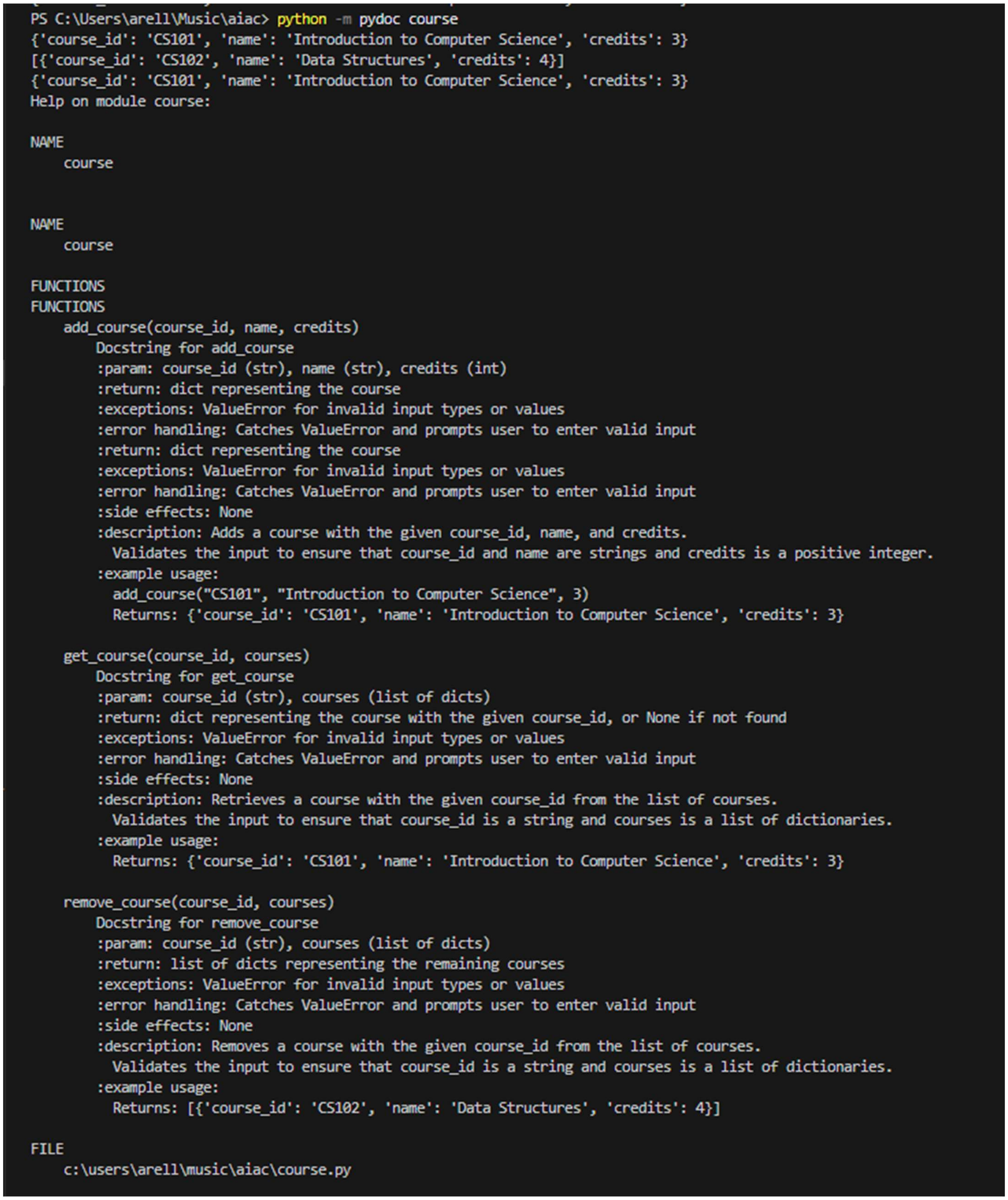
1. Create a module course.py with functions:
    - o add_course(course_id, name, credits)
    - o remove_course(course_id)
    - o get_course(course_id)
2. Add docstrings with Copilot.
3. Generate documentation in the terminal.
4. Export the documentation in HTML format and open it in a browser.

Screenshots:

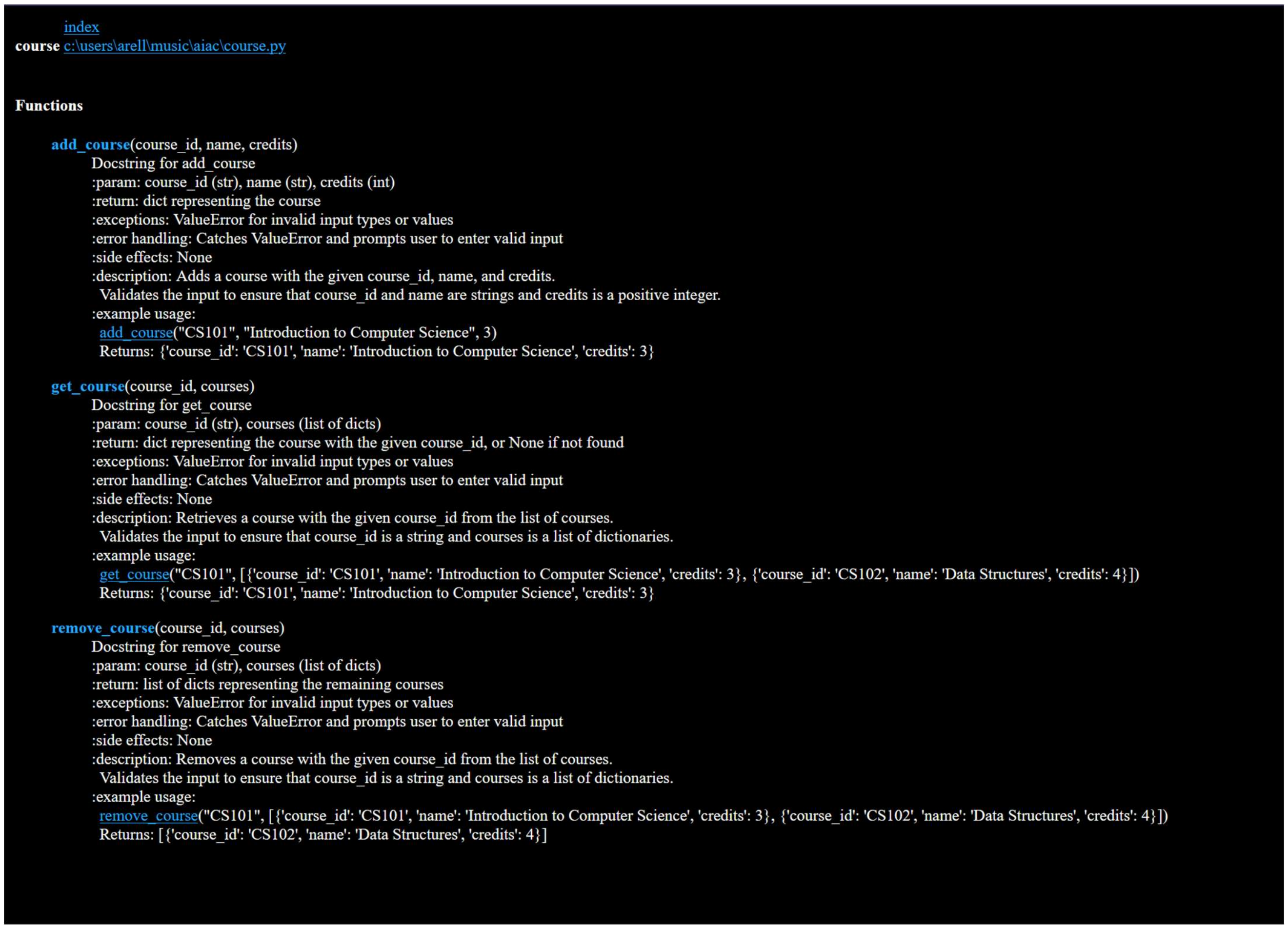


course.py > ...

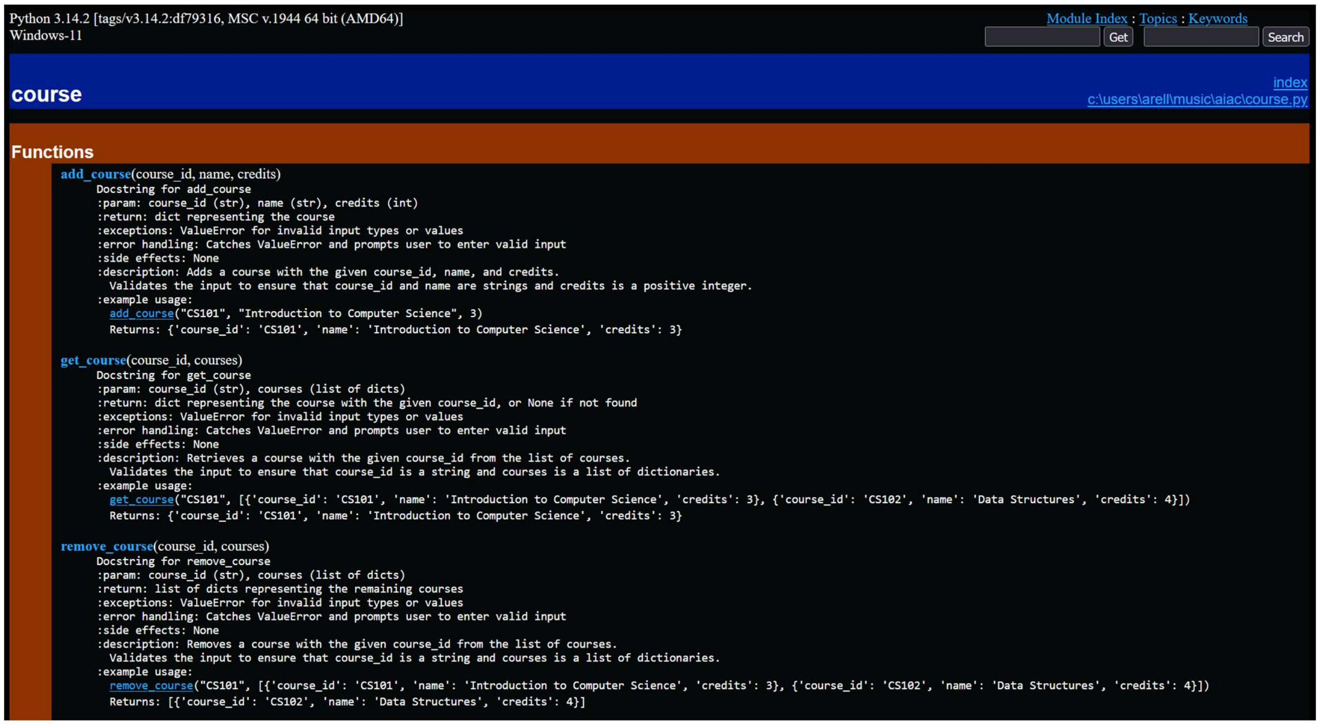
```python
def add_course(course_id,name,credits):
    """
    Docstring for add_course
    :param: course_id (str), name (str), credits (int)
    :return: dict representing the course
    :exceptions: ValueError for invalid input types or values
    :error handling: Catches ValueError and prompts user to enter valid input
    :side effects: None
    :description: Adds a course with the given course_id, name, and credits.
      Validates the input to ensure that course_id and name are strings and credits is a positive integer.
    :example usage:
      add_course("CS101", "Introduction to Computer Science", 3)
      Returns: {'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}
    """
    if not isinstance(course_id, str) or not isinstance(name, str):
        raise ValueError("Course ID and name must be strings.")
    if not isinstance(credits, int) or credits <= 0:
        raise ValueError("Credits must be a positive integer.")

    course = {
        'course_id': course_id,
        'name': name,
        'credits': credits
    }

    return course
def remove_course(course_id, courses):
    """
    Docstring for remove_course
    :param: course_id (str), courses (list of dicts)
    :return: list of dicts representing the remaining courses
    :exceptions: ValueError for invalid input types or values
    :error handling: Catches ValueError and prompts user to enter valid input
    :side effects: None
    :description: Removes a course with the given course_id from the list of courses.
      Validates the input to ensure that course_id is a string and courses is a list of dictionaries.
    :example usage:
      remove_course("CS101", [{'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}, {'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}])
      Returns: [{'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}]
    """
    if not isinstance(course_id, str):
        raise ValueError("Course ID must be a string.")
    if not isinstance(courses, list) or not all(isinstance(course, dict) for course in courses):
        raise ValueError("Courses must be a list of dictionaries.")

    remaining_courses = [course for course in courses if course['course_id'] != course_id]

    return remaining_courses
def get_course(course_id, courses):
    """
    Docstring for get_course
    :param: course_id (str), courses (list of dicts)
    :return: dict representing the course with the given course_id, or None if not found
    :exceptions: ValueError for invalid input types or values
    :error handling: Catches ValueError and prompts user to enter valid input
    :side effects: None
    :description: Retrieves a course with the given course_id from the list of courses.
      Validates the input to ensure that course_id is a string and courses is a list of dictionaries.
    :example usage:
      get_course("CS101", [{'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}, {'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}])
      Returns: {'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}
    """
    if not isinstance(course_id, str):
        raise ValueError("Course ID must be a string.")
    if not isinstance(courses, list) or not all(isinstance(course, dict) for course in courses):
        raise ValueError("Courses must be a list of dictionaries.")

    for course in courses:
        if course['course_id'] == course_id:
            return course

    return None
print(add_course("CS101", "Introduction to Computer Science", 3))
print(remove_course("CS101", [{'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}, {'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}]))
print(get_course("CS101", [{'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}, {'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}]))
```



```
PS C:\Users\arell\Music\aiac> python -m pydoc course
{'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}
[{'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}]
{'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}
Help on module course:

NAME
    course


NAME
    course

FUNCTIONS
FUNCTIONS
    add_course(course_id, name, credits)
        Docstring for add_course
        :param: course_id (str), name (str), credits (int)
        :return: dict representing the course
        :exceptions: ValueError for invalid input types or values
        :error handling: Catches ValueError and prompts user to enter valid input
        :return: dict representing the course
        :exceptions: ValueError for invalid input types or values
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: Adds a course with the given course_id, name, and credits.
          Validates the input to ensure that course_id and name are strings and credits is a positive integer.
        :example usage:
          add_course("CS101", "Introduction to Computer Science", 3)
          Returns: {'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}

    get_course(course_id, courses)
        Docstring for get_course
        :param: course_id (str), courses (list of dicts)
        :return: dict representing the course with the given course_id, or None if not found
        :exceptions: ValueError for invalid input types or values
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: Retrieves a course with the given course_id from the list of courses.
          Validates the input to ensure that course_id is a string and courses is a list of dictionaries.
        :example usage:
          Returns: {'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}

    remove_course(course_id, courses)
        Docstring for remove_course
        :param: course_id (str), courses (list of dicts)
        :return: list of dicts representing the remaining courses
        :exceptions: ValueError for invalid input types or values
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: Removes a course with the given course_id from the list of courses.
          Validates the input to ensure that course_id is a string and courses is a list of dictionaries.
        :example usage:
          Returns: [{'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}]

FILE
    c:\users\arell\music\aiac\course.py
```



```
        index
course c:\users\arell\music\aiac\course.py


Functions

    add_course(course_id, name, credits)
        Docstring for add_course
        :param: course_id (str), name (str), credits (int)
        :return: dict representing the course
        :exceptions: ValueError for invalid input types or values
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: Adds a course with the given course_id, name, and credits.
          Validates the input to ensure that course_id and name are strings and credits is a positive integer.
        :example usage:
          add_course("CS101", "Introduction to Computer Science", 3)
          Returns: {'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}

    get_course(course_id, courses)
        Docstring for get_course
        :param: course_id (str), courses (list of dicts)
        :return: dict representing the course with the given course_id, or None if not found
        :exceptions: ValueError for invalid input types or values
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: Retrieves a course with the given course_id from the list of courses.
          Validates the input to ensure that course_id is a string and courses is a list of dictionaries.
        :example usage:
          get_course("CS101", [{'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}, {'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}])
          Returns: {'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}

    remove_course(course_id, courses)
        Docstring for remove_course
        :param: course_id (str), courses (list of dicts)
        :return: list of dicts representing the remaining courses
        :exceptions: ValueError for invalid input types or values
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: Removes a course with the given course_id from the list of courses.
          Validates the input to ensure that course_id is a string and courses is a list of dictionaries.
        :example usage:
          remove_course("CS101", [{'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}, {'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}])
          Returns: [{'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}]
```



Python 3.14.2 [tags/v3.14.2:df79316, MSC v.1944 64 bit (AMD64)]
Windows-11
Module Index : Topics : Keywords
Get
Search
course
index
c:\users\arell\music\aiac\course.py
Functions

```
    add_course(course_id, name, credits)
        Docstring for add_course
        :param: course_id (str), name (str), credits (int)
        :return: dict representing the course
        :exceptions: ValueError for invalid input types or values
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: Adds a course with the given course_id, name, and credits.
          Validates the input to ensure that course_id and name are strings and credits is a positive integer.
        :example usage:
          add_course("CS101", "Introduction to Computer Science", 3)
          Returns: {'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}

    get_course(course_id, courses)
        Docstring for get_course
        :param: course_id (str), courses (list of dicts)
        :return: dict representing the course with the given course_id, or None if not found
        :exceptions: ValueError for invalid input types or values
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: Retrieves a course with the given course_id from the list of courses.
          Validates the input to ensure that course_id is a string and courses is a list of dictionaries.
        :example usage:
          get_course("CS101", [{'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}, {'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}])
          Returns: {'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}

    remove_course(course_id, courses)
        Docstring for remove_course
        :param: course_id (str), courses (list of dicts)
        :return: list of dicts representing the remaining courses
        :exceptions: ValueError for invalid input types or values
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: Removes a course with the given course_id from the list of courses.
          Validates the input to ensure that course_id is a string and courses is a list of dictionaries.
        :example usage:
          remove_course("CS101", [{'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}, {'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}])
          Returns: [{'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}]
```