LAB ASSIGNMENT – 9.1

NAME: CH VISHWANATH RAO

BATCH: 02

HT NO: 2303A51095

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:

A] DOCSTRING:

```python
def find_max_nums(nums):
    """
    This function takes a list of numbers as input and returns the maximum number in the list

    Args:
        nums (list): A list of numbers.

    Returns:
        int: The maximum number in the list.
    """

    max_num = nums[0]

    for num in nums:
        if num > max_num:
            max_num = num

    return max_num
```

B] Inline comments

```python
def find_max_numbers(nums):
    # This function takes a list of numbers as input and returns the maximum number in the list
    max_num = nums[0]
    for num in nums:
        if num > max_num:

            max_num = num # Update max_num if the current number is greater than the current max_num

            # Update max_num if the current number is greater than the current max_num
    return max_num
```

C] Google-style documentation

```python
def find_max_numbers(nums:int) -> int:
    """
    Max of numbers in a list

    Args:
        nums (list): A list of numbers.

    Returns:
        int: The maximum number in the list.
    """

    max_num = nums[0]
    for num in nums:
        if num > max_num:

            max_num = num
    return max_num
```

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:

A] *Docstring*

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

    :param user: Description
    :param password: Description
    :param credentials: Description
    """
    if user in credentials and credentials[user] == password:
        print("Login successful")
    else:
        print("Invalid credentials")
```

B] *INLINE COMMENTS*

```python
def login(user, password,credentials):
    # Docstring for login
    if user in credentials and credentials[user] == password:
        # check if user and password are in the credentials dictionary and if the password matches the one in the dictionary
        print("Login successful")
        # if the user and password are correct, print "Login successful"
    else:
        print("Invalid credentials") # if the user or password is incorrect, print "Invalid credentials"
```

C] GOOGLE STYLE DOCUMENTATION

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

    :param user: Description
    :param password: Description
    :param credentials: Description
    :return: Description
    """
    if user in credentials and credentials[user] == password:
        print("Login successful")
        return 1
    else:
        print("Invalid credentials")
        return 0
    # Testing the function
    print(login("user1", "password1", {"user1": "password1", "user2": "password2"}))
```

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

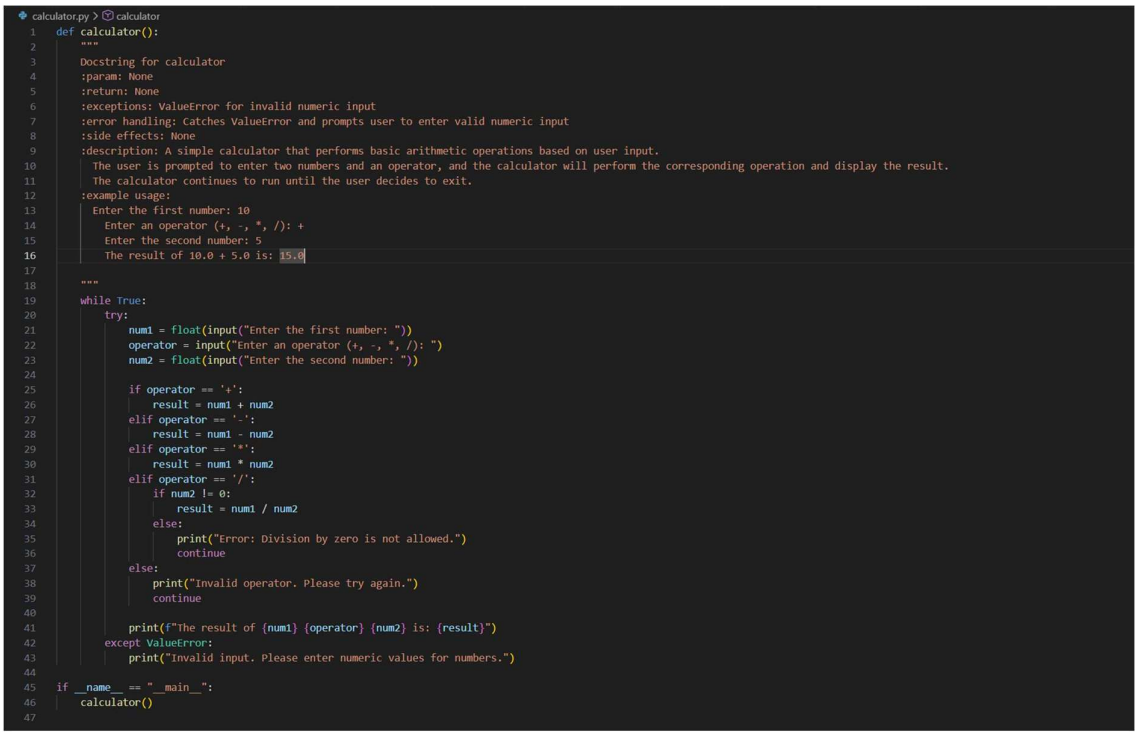
o multiply(a, b) – returns the product of two numbers

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:



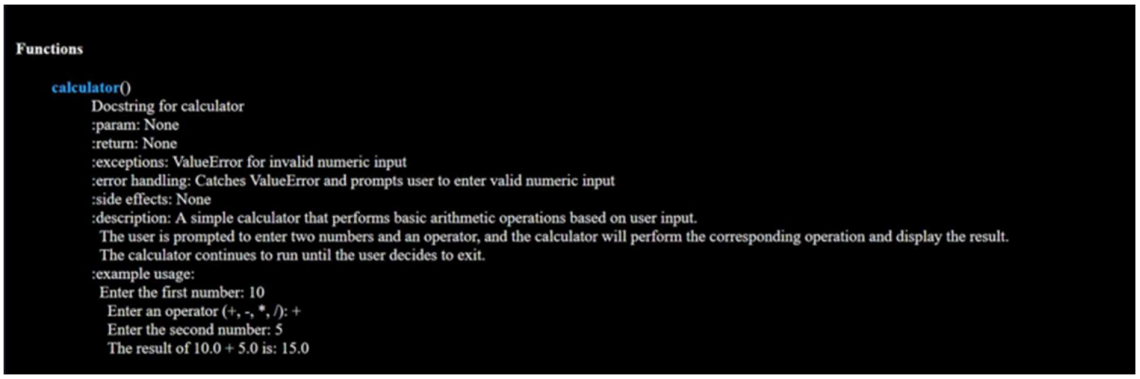
```python
def calculator():
    """
    Docstring for calculator
    :param: None
    :return: None
    :exceptions: ValueError for invalid numeric input
    :error handling: Catches ValueError and prompts user to enter valid numeric input
    :side effects: None
    :description: A simple calculator 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()
```



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:

conversion.py > conversions

```python
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()
```

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

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:

```
course.py >...
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}]))

```

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 : Get

course

c:\users

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', 'credit
      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', 'cre
      Returns: [{'course_id': 'CS102', 'name': 'Data Structures', 'credits': 4}]
```