

Lab Assignment-9

Name: D.BHARATH KUMAR

Hall ticket : 2303A51097

Batch : 02

Problem 1:

Consider the following Python function:

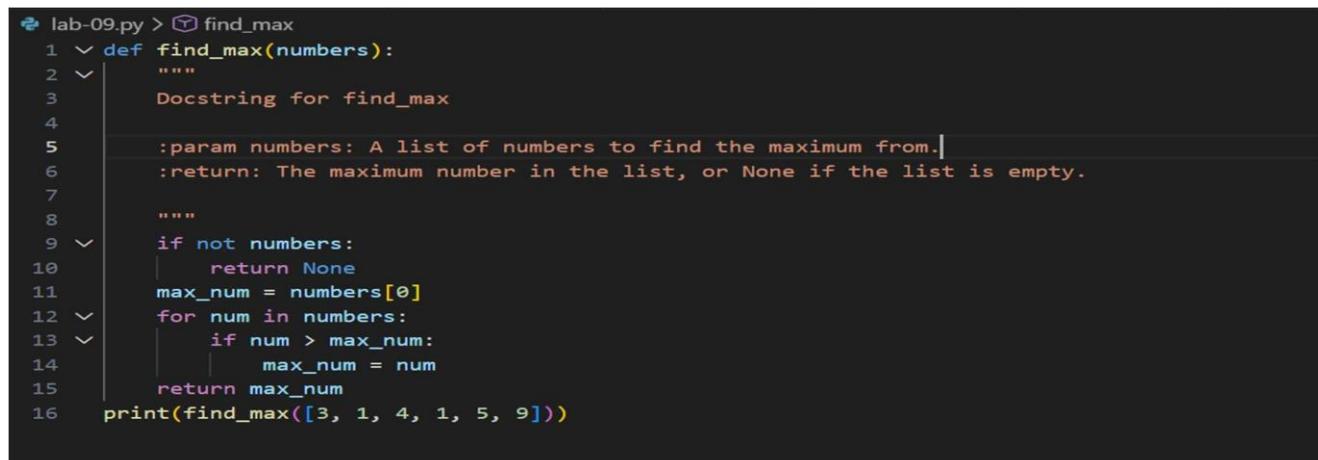
```
def find_max(numbers):
    return max(numbers)
```

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
1  def find_max(numbers):
2      """
3          Docstring for find_max
4
5      :param numbers: A list of numbers to find the maximum from.
6      :return: The maximum number in the list, or None if the list is empty.
7
8      """
9      if not numbers:
10          return None
11      max_num = numbers[0]
12      for num in numbers:
13          if num > max_num:
14              max_num = num
15      return max_num
16  print(find_max([3, 1, 4, 1, 5, 9]))
```

B: in line comments

```
lab-09.py > ...
1 def find_max(numbers):
2     if not numbers: # Check if the list is empty
3         return None
4     max_num = numbers[0]# Initialize max_num to the first element of the list
5     for num in numbers: # Iterate through the list to find the maximum number
6         if num > max_num: # Update max_num if the current number is greater than max_num
7             max_num = num # Update max_num to the current number
8     return max_num # Return the maximum number found in the list
9 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 > ...
1 def find_max(numbers: list) -> int:
2 """
3     Docstring for find_max
4
5     :param numbers: A list of integers to find the maximum from
6     :type numbers: list
7     :return: The maximum integer in the list
8     :rtype: int
9     :exceptions: ValueError if the input list is empty
10    :error handling: Raises ValueError if the input list is empty
11    :side effects: None
12    :raises: ValueError if the input list is empty
13
14 """
15 if not numbers:
16     raise ValueError("Input list cannot be empty")
17 max_number = numbers[0]
18 for number in numbers:
19     if number > max_number:
20         max_number = number
21 return max_number
22
23 numbers = [int(x) for x in input("Enter a list of integers separated by spaces: ").split()]
24 try:
25     result = find_max(numbers)
26     print(f"The maximum number in the list is: {result}")
27 except ValueError as e:
28     print(e)
```

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/lab_09.py
the maximum number is: 5
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

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.

A:docstring:

```

10
11  def login(user,password,credentials):
12      """
13          Docstring for login
14
15          :param user: Description
16          :param password: Description
17          :param credentials: Description
18          :return: Description
19          :exceptions: None
20          :error handling: None
21          :side effects: None
22      """
23
24      if user in credentials and credentials[user] == password:
25          return True
26      return False
27 credentials = {"user1": "password1", "user2": "password2"}
28 print(login("user1", "password1", credentials))

```

B: google style

```

11  def login(user:str,password:str,credentials:dict) -> bool:
12      """
13          Docstring for login
14
15          :param user: Description
16          :param password: Description
17          :param credentials: Description
18          :return: Description
19          :exceptions: None
20          :error handling: None
21          :side effects: None
22
23      """
24
25      if user in credentials and credentials[user] == password:
26          return True
27      else:
28          return False
29 credentials = {
30     "user1": "password1",
31     "user2": "password2",
32     "user3": "password3"
33 }
34 print(login("user1", "password1", credentials)) # Output should be True

```

C:Inline comments

```

10
11  def login(user,password,credentials):
12      if user in credentials and credentials[user] == password: # Check if the username exists in the credentials and if the password matches
13          return True # Return True if the login is successful
14      return False # Return False if the login is unsuccessful
15  credentials = {"user1": "password1", "user2": "password2"}
16  print(login("user1", "password1", credentials)) # Output should be True, which means the login is successful
17  print(login("user1", "wrongpassword", credentials)) # Output should be False, which means the login is unsuccessful
18  print(login("user3", "password3", credentials)) # Output should be False, which means the login is unsuccessful]
19

```

Output:

The screenshot shows a terminal window with the following content:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + 

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe c:/users/BHARATH/OneDrive/Pictures/Desktop/AIAC/lab_09.py
Login successful: True
Login failed: False
Login failed: False
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\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
1 def calculator():
2     """
3         Docstring for calculator
4         :param: None
5         :return: None
6         :exceptions: ValueError for invalid numeric input
7         :error handling: Catches ValueError and prompts user to enter valid numeric input
8         :side effects: None
9         :description: A simple calculator that performs basic arithmetic operations based on user input.
10        The user is prompted to enter two numbers and an operator, and the calculator will perform the corresponding operation and display the result.
11        The calculator continues to run until the user decides to exit.
12        :example usage:
13        Enter the first number: 10
14        Enter an operator (+, -, *, /): +
15        Enter the second number: 5
16        The result of 10.0 + 5.0 is: 15.0
17
18    """
19    while True:
20        try:
21            num1 = float(input("Enter the first number: "))
22            operator = input("Enter an operator (+, -, *, /): ")
23            num2 = float(input("Enter the second number: "))
24
25            if operator == '+':
26                result = num1 + num2
27            elif operator == '-':
28                result = num1 - num2
29            elif operator == '*':
30                result = num1 * num2
31            elif operator == '/':
32                if num2 != 0:
33                    result = num1 / num2
34                else:
35                    print("Error: Division by zero is not allowed.")
36                    continue
37            else:
38                print("Invalid operator. Please try again.")
39                continue
40
41            print(f"The result of {num1} {operator} {num2} is: {result}")
42        except ValueError:
43            print("Invalid input. Please enter numeric values for numbers.")
44
45    if __name__ == "__main__":
46        calculator()
47

```

Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> python -m pydoc calculator
Help on module calculator:

NAME
    calculator

FUNCTIONS
    calculator()
        docstring of calculator
        parameter: none
        return: none
        exceptions: value error if the user input is not a number
        error handling: try-except block to catch value error and prompt user to enter a valid number
        side effects: none
        description: This function takes user input for two numbers and performs addition, subtraction, multiplication, and division operations.
        example usage:
        >>> calculator()
        Enter first number: 5
        Enter second number: 3
        Enter operation (+, -, *, /): +
        Result: 8.0

FILE
c:\users\bharath\onedrive\pictures\desktop\aiac\calculator.py

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```

[index](#)

calculator <c:/users/bharath/onedrive/pictures/desktop/aiac/calculator.py>

Functions

```
calculator()
    docstring of calculator
    parameter: none
    return: none
    exceptions: value error if the user input is not a number
    error handling: try-except block to catch value error and prompt user to enter a valid number
    side effects: none
    description: This function takes user input for two numbers and performs addition, subtraction, multiplication, and division operations.
    example usage:
>>> calculator()
Enter first number: 5
Enter second number: 3
Enter operation (+, -, *, /): +
Result: 8.0
```

The screenshot shows a Python documentation page for the 'calculator' module. At the top, it displays the Python version (3.13.5) and build details (tags/v3.13.5:6eb20a2, MSC v.1943 64 bit (AMD64)). Below this is a navigation bar with links for 'Module Index', 'Topics', 'Keywords', 'Get', and 'Search'. The main content area has a blue header with the module name 'calculator' and its path 'c:/users/bharath/onedrive/pictures/desktop/aiac/calculator.py'. A large orange sidebar on the left is titled 'Functions' and contains the detailed docstring for the 'calculator()' function, which is identical to the one shown in the previous code block. The main content area below the sidebar shows the function signature and its behavior when run in a terminal.

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
1 def conversions():
2     """
3     Docstring for conversions
4     :param: None
5     :return: None
6     :exceptions: ValueError for invalid input
7     :error handling: Catches ValueError and prompts user to enter valid input
8     :side effects: None
9     :description: A converter that allows users to convert between decimal, binary, and hexadecimal number systems.
10    The user can choose to convert a decimal number to binary, a binary number to decimal,
11    or a decimal number to hexadecimal. The converter continues to run until the user decides to exit.
12    :example usage:
13        Please choose a conversion type:
14            1. Decimal to Binary
15            2. Binary to Decimal
16            3. Decimal to Hexadecimal
17            4. Exit
18        Enter your choice (1-4): 1
19        Enter a decimal number: 10
20        The binary representation of 10 is: 1010
21    """
22    print("Welcome to the decimal to binary, binary to decimal, and decimal to hexadecimal converter.")
23    while True:
24        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): ")
25
26        if choice == '1':
27            decimal_number = int(input("Enter a decimal number: "))
28            binary_number = bin(decimal_number)[2:]
29            print(f"The binary representation of {decimal_number} is: {binary_number}\n")
30
31        elif choice == '2':
32            binary_number = input("Enter a binary number: ")
33            try:
34                decimal_number = int(binary_number, 2)
35                print(f"The decimal representation of {binary_number} is: {decimal_number}\n")
36            except ValueError:
37                print("Invalid binary number. Please enter a valid binary number.\n")
38
39        elif choice == '3':
40            decimal_number = int(input("Enter a decimal number: "))
41            hexadeciml_number = hex(decimal_number)[2:].upper()
42            print(f"The hexadecimal representation of {decimal_number} is: {hexadeciml_number}\n")
43
44        elif choice == '4':
45            print("Exiting the converter. Goodbye!")
46            break
47
48        else:
49            print("Invalid choice. Please enter a number between 1 and 4.\n")
50    if __name__ == "__main__":
51        conversions()
52
```

Output:

```

Help on module conversions:

NAME
    conversions

FUNCTIONS
    conversions()
        Docstring for conversions function.
        parameters: None
        returns: None
        exceptions: value error if the user input is not a number
        error handling: try-except block to catch value error and prompt user to enter a valid number
        side effects: None
        description: This function converts the user input from one unit to another unit. The user is prompted to enter the value, the initial unit, and the final unit. The function then converts the value from the initial unit to the final unit and displays the result.
        example usage:
        please choose an option:
        1.decimal to binary
        2.binary to decimal
        3.decimal to hexadecimal
        4.exit
        enter your choice: 1
        enter the decimal number: 10
        binary equivalent: 1010

FILE
    c:\\users\\bharath\\onedrive\\pictures\\desktop\\aiac\\conversions.py
```

```

PS C:\\Users\\BHARATH\\OneDrive\\Pictures\\Desktop\\AIAC>
PS C:\\Users\\BHARATH\\OneDrive\\Pictures\\Desktop\\AIAC>
PS C:\\Users\\BHARATH\\OneDrive\\Pictures\\Desktop\\AIAC> █
```

Functions

```
conversions()  
Docstring for conversions function.  
parameters: None  
returns: None  
exceptions: value error if the user input is not a number  
error handling: try-except block to catch value error and prompt user to enter a valid number  
side effects: None  
description: This function converts the user input from one unit to another unit. The user is prompted to enter the value, the initial unit, and the final unit. The function then converts the value from the initial unit to the final unit  
example usage:  
please choose an option:  
1.decimal to binary  
2.binary to decimal  
3.decimal to hexadecimal  
4.exit  
enter your choice: 1  
enter the decimal number: 10  
binary equivalent: 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.

Screenshots:

```

@course.py ?
1 def add_course(course_id, name, credits):
2
3     """
4         Docstring for add_course
5         iparam: course_id (str), name (str), credits (int)
6         return: dict representing the course
7         exceptions: ValueError for invalid input types or values
8         error handling: Catches ValueError and prompts user to enter valid input
9         side effects: None
10        description: Adds a course with the given course_id, name, and credits.
11        Validates the input to ensure that course_id, name, and credits is a positive integer.
12        example usage:
13        add_course('CS101', 'Introduction to Computer Science', 3)
14        Returns: {'course_id': 'CS101', 'name': 'Introduction to Computer Science', 'credits': 3}
15        """
16
17    if not isinstance(course_id, str) or not isinstance(name, str):
18        raise ValueError("Course ID and name must be strings.")
19    if not isinstance(credits, int) or credits <= 0:
20        raise ValueError("Credits must be a positive integer.")
21
22    course = {
23        'course_id': course_id,
24        'name': name,
25        'credits': credits
26    }
27
28    return course
29
30 def remove_course(course_id, courses):
31
32     """
33         Docstring for remove_course
34         iparam: course_id (str), courses (list of dicts)
35         return: list of dicts representing the remaining courses
36         exceptions: ValueError for invalid input types or values
37         error handling: Catches ValueError and prompts user to enter valid input
38         side effects: None
39         description: Removes a course with the given course_id from the list of courses.
40         Validates the input to ensure that course_id is a string and courses is a list of dictionaries.
41         example usage:
42         remove_course('CS101', [{"course_id": "CS101", "name": "Introduction to Computer Science", "credits": 3}, {"course_id": "CS102", "name": "Data Structures", "credits": 4}])
43         Returns: [{"course_id": "CS102", "name": "Data Structures", "credits": 4}]
44
45    if not isinstance(course_id, str):
46        raise ValueError("Course ID must be a string.")
47    if not isinstance(courses, list) or not all(isinstance(course, dict) for course in courses):
48        raise ValueError("Courses must be a list of dictionaries.")
49
50    remaining_courses = [course for course in courses if course['course_id'] != course_id]
51
52    return remaining_courses
53
54 def get_course(course_id, courses):
55
56     """
57         Docstring for get_course
58         iparam: course_id (str), courses (list of dicts)
59         return: dict representing the course with the given course_id, or None if not found
60         exceptions: ValueError for invalid input types or values
61         error handling: Catches ValueError and prompts user to enter valid input
62         side effects: None
63         description: Retrieves a course with the given course_id from the list of courses.
64         Validates the input to ensure that course_id is a string and courses is a list of dictionaries.
65         example usage:
66         get_course('CS101', [{"course_id": "CS101", "name": "Introduction to Computer Science", "credits": 3}, {"course_id": "CS102", "name": "Data Structures", "credits": 4}])
67         Returns: {"course_id": "CS101", "name": "Introduction to Computer Science", "credits": 3}
68
69    if not isinstance(course_id, str):
70        raise ValueError("Course ID must be a string.")
71    if not isinstance(courses, list) or not all(isinstance(course, dict) for course in courses):
72        raise ValueError("Courses must be a list of dictionaries.")
73
74    for course in courses:
75        if course['course_id'] == course_id:
76            return course
77
78    return None
79
80 print(add_course('CS101', "Introduction to Computer Science", 3))
81 print(remove_course('CS101', [{"course_id": "CS101", "name": "Introduction to Computer Science", "credits": 3}, {"course_id": "CS102", "name": "Data Structures", "credits": 4}]))
82 print(get_course('CS101', [{"course_id": "CS101", "name": "Introduction to Computer Science", "credits": 3}, {"course_id": "CS102", "name": "Data Structures", "credits": 4}]))

```

```
course_credits must be a positive integer. Please enter a valid number.
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> python -m pydoc course
{'course_name': 'Introduction to Computer Science', 'course_code': 'CS101', 'course_credits': 3}
course_credits must be a positive integer. Please enter a valid number.
Help on module course:

NAME
    course

FUNCTIONS
    course(course_name, course_code, course_credits)
        Docstring for course function.
        parameters: course_name (str), course_code (str), course_credits (int)
        returns: dict
        exceptions: value error if course_credits is not a positive integer
        error handling: try-except block to catch value error and prompt user to enter a valid number for course_credits
        side effects: None
        description: This function takes the course name, course code, and course credits as input and returns a dictionary containing the course information. The function also validates that the course credits is a positive integer and handles any value errors that may arise.
        example usage:
        course("Introduction to Computer Science", "CS101", 3)
        output: {'course_name': 'Introduction to Computer Science', 'course_code': 'CS101', 'course_credits': 3}
        course("Data Structures", "CS102", -2)
        output: ValueError: course_credits must be a positive integer. Please enter a valid number.

DATA
```

```
course_info = {'course_code': 'CS101', 'course_credits': 3, 'course_na...
course_info_invalid = None
```

```
FILE
c:\users\bharath\onedrive\pictures\desktop\aiac\course.py
```

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

[index](#)
course c:\users\bharath\onedrive\pictures\desktop\aiac\course.py

Functions

```
course(course_name, course_code, course_credits)
Docstring for course function.
parameters: course_name (str), course_code (str), course_credits (int)
returns: dict
exceptions: value error if course_credits is not a positive integer
error handling: try-except block to catch value error and prompt user to enter a valid number for course_credits
side effects: None
description: This function takes the course name, course code, and course credits as input and returns a dictionary containing the course information. The function also validates that the course credits is a positive integer and has
example usage:
course("Introduction to Computer Science", "CS101", 3)
output: {'course_name': 'Introduction to Computer Science', 'course_code': 'CS101', 'course_credits': 3}
course("Data Structures", "CS102", -2)
output: ValueError: course_credits must be a positive integer. Please enter a valid number.
```

Data

```
course_info = {'course_code': 'CS101', 'course_credits': 3, 'course_name': 'Introduction to Computer Science'}
course_info_invalid = None
```

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

[Module Index](#) : [Topics](#) : [Keywords](#)

Get Search

[index](#)

c:\users\bharath\onedrive\pictures\desktop\aiac\course.py

course

Functions

```
course(course_name, course_code, course_credits)
Docstring for course function.
parameters: course_name (str), course_code (str), course_credits (int)
returns: dict
exceptions: value error if course_credits is not a positive integer
error handling: try-except block to catch value error and prompt user to enter a valid number for course_credits
side effects: None
description: This function takes the course name, course code, and course credits as input and returns a dictionary containing the course information. The function also validates that the course credits is a positive integer and has
example usage:
course("Introduction to Computer Science", "CS101", 3)
output: {'course_name': 'Introduction to Computer Science', 'course_code': 'CS101', 'course_credits': 3}
course("Data Structures", "CS102", -2)
output: ValueError: course_credits must be a positive integer. Please enter a valid number.
```

Data

```
course_info = {'course_code': 'CS101', 'course_credits': 3, 'course_name': 'Introduction to Computer Science'}
course_info_invalid = None
```