

# Lab Assignment-9

Name: P. Chandra Vardhan Reddy

Hall ticket:2303A51034

Batch-01

## 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:

```
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> 
```

Problem 2: Consider the following Python func on:

```
def login(user, password, creden als):
    return creden als.get(user) == password
```

Task:

1. Write documenta on in all three formats.
2. Cri cally compare the approaches.
3. Recommend which style would be most helpful for new developers onboarding a project, and jus fy 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     if user in credentials and credentials[user] == password:
24         return True
25     return False
26 credentials = {"user1": "password1", "user2": "password2"}
27 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     if user in credentials and credentials[user] == password:
25         return True
26     else:
27         return False
28     credentials = {
29         "user1": "password1",
30         "user2": "password2",
31         "user3": "password3"
32     }
33     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:

```

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
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:

```
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 result.

It.

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
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            hexadecimal_number = hex(decimal_number)[2:].upper()
42            print(f"The hexadecimal representation of {decimal_number} is: {hexadecimal_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:

```

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

```

## 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 documenta on in the terminal.
4. Export the documenta on in HTML format and open it in a browser.

Screenshots:

```

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

```

```
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}]
```