Lab Assignment-09.5

Name:Anand

Hallticket:2303A51090

Batch-02

Problem 1: String Utilities Function

Consider the following Python function:

```
def reverse_string(text):
return text[::-1]
```

Task:

1. Write documentation in:
   - o (a) Docstring
   - o (b) Inline comments
   - o (c) Google-style documentation
2. Compare the three documentation styles.
3. Recommend the most suitable style for a utility-based string library.

Docstring:

```python
def reverse_string(s):
    """
    Docstring for reverse_string
    :param: s (str)
    :return: str representing the reversed input string
    :exceptions: ValueError for invalid input types
    :error handling: Catches ValueError and prompts user to enter valid input
    :side effects: None
    :description: Reverses the input string. Validates the input to ensure that it is a string.
    :example usage:
      reverse_string("hello")
      Returns: "olleh"
    """
    if not isinstance(s, str):
        raise ValueError("Input must be a string.")

    return s[::-1]
```

```
Help on module assg_09_5:

NAME
    assg_09_5

FUNCTIONS
    reverse_string(s)
        Docstring for reverse_string
        :param: s (str)
        :return: str representing the reversed input string
        :exceptions: ValueError for invalid input types
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: Reverses the input string. Validates the input to ensure that it is a string.
        :example usage:
          reverse_string("hello")
-- More  --
```

Incline comments:

assg_09_5.py > ...

```python
def reverse_string(s):# defining a function that takes a string as input
    if not isinstance(s, str):# checking if the input is not a string
        raise ValueError("Input must be a string.")# raising a ValueError if the input is not a string
    return s[::-1]# returning the reversed string using slicing
s="Hello, World!"# defining a string variable
print(reverse_string(s))# calling the reverse_string function and printing the result
```

Google style :

assg_09_5.py > ...

```python
def reverse_string(s:str) -> str:
    """
    Docstring for reverse_string
    :param: s (str)
    :return: str representing the reversed input string
    :exceptions: ValueError for invalid input types
    :error handling: Catches ValueError and prompts user to enter valid input
    :side effects: None
    :description: Reverses the input string. Validates the input to ensure that it is a string.
    :example usage:
      reverse_string("hello")
      Returns: "olleh"
    """
    if not isinstance(s, str):
        raise ValueError("Input must be a string.")

    return s[::-1]
string_to_reverse = "hello"
print(reverse_string(string_to_reverse))
```

Problem 2: Password Strength Checker

Consider the function:

def check_strength(password):

return len(password) >= 8

Task:

1. Document the function using docstring, inline comments, and

Google style.

2. Compare documentation styles for security-related code.

3. Recommend the most appropriate style.

Doc_string and google_style:

```python

def password_strength_checker(password:str) -> str:
    """
    Docstring for password_strength_checker
    :param: password (str)
    :return: str indicating the strength of the password ("Weak", "Moderate", "Strong")
    :exceptions: ValueError for invalid input types or values
    :error handling: Catches ValueError and prompts user to enter valid input
    :side effects: None
    :description: Evaluates the strength of a given password based on its length and character composition.
      Validates the input to ensure that it is a string and meets certain criteria for strength.
    :example usage:
      password_strength_checker("P@ssw0rd")
      Returns: "Strong"
    """
    if not isinstance(password, str):
        raise ValueError("Password must be a string.")

    length = len(password)
    has_upper = any(c.isupper() for c in password)
    has_lower = any(c.islower() for c in password)
    has_digit = any(c.isdigit() for c in password)
    has_special = any(not c.isalnum() for c in password)

    if length < 6:
        return "Weak"
    elif length < 12:
        if has_upper and has_lower and has_digit:
            return "Moderate"
        else:
            return "Weak"
    else:
        if has_upper and has_lower and has_digit and has_special:
            return "Strong"
        else:
            return "Moderate"
password = "P@ssw0rd"
print(password_strength_checker(password))

```

Incline comments:

```python

def password_strength_checker(password:str) -> str: #defining the function with type hints

    if not isinstance(password, str):# if password is not a string, raise an error
        raise ValueError("Password must be a string.")# raise an error if the password is not a string
    if len(password) < 8:# if the password is less than 8 characters, return weak
        return "Weak" #return weak if the password is less than 8 characters
    if not any(char.isupper() for char in password):# if the password does not contain an uppercase letter, return weak
        return "Weak" #return weak if the password does not contain an uppercase letter
    if not any(char.islower() for char in password):# if the password does not contain a lowercase letter, return weak
        return "Weak" #return weak if the password does not contain a lowercase letter
    if not any(char.isdigit() for char in password):# if the password does not contain a digit, return weak
        return "Weak" #return weak if the password does not contain a digit
    if not any(char in "!@#$%^&*()-_=+[]{}|;:'\",.<>?/" for char in password):# if the password does not contain a special character, return weak
        return "Weak" #return weak if the password does not contain a special character
    return "Strong" #return strong if the password meets all the criteria
password_to_check = "P@ssw0rd"
print(password_strength_checker(password_to_check))

```

Problem 3: Math Utilities Module

Task:

1. Create a module math_utils.py with functions:

    o square(n)

- cube(n)
- factorial(n)

2. Generate docstrings automatically using AI tools.
3. Export documentation as an HTML file.

Code:

math_utils.py > ...

```python
def factorial(n):
    """
    Calculate the factorial of a non-negative integer n.
    :param n: A non-negative integer
    :return: The factorial of n
    :raises ValueError: If n is negative or not an integer


    Example usage:
    print(factorial(5))  # Output: 120

    """
    if not isinstance(n, int):
        raise ValueError("Input must be an integer.")
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")

    result = 1
    for i in range(2, n + 1):
        result *= i

    return result
def square(x):
    """
    Calculate the square of a number x.
    :param x: A number (int or float)
    :return: The square of x
    :raises ValueError: If x is not a number


    Example usage:
    print(square(4))  # Output: 16
    print(square(2.5))  # Output: 6.25

    """
    if not isinstance(x, (int, float)):
        raise ValueError("Input must be a number.")

    return x * x
def cube(x):
    """
    Calculate the cube of a number x.
    :param x: A number (int or float)
    :return: The cube of x
    :raises ValueError: If x is not a number


    Example usage:
    print(cube(3))  # Output: 27
    print(cube(1.5))  # Output: 3.375

    """
    if not isinstance(x, (int, float)):
        raise ValueError("Input must be a number.")

    return x * x * x
print(factorial(5))
print(square(4))
print(cube(3))

```

Output:

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


Functions

    cube(x)
        Calculate the cube of a number x.
        :param x: A number (int or float)
        :return: The cube of x
        :raises ValueError: If x is not a number


        Example usage:
        print(cube(3))  # Output: 27
        print(cube(1.5))  # Output: 3.375

    factorial(n)
        Calculate the factorial of a non-negative integer n.
        :param n: A non-negative integer
        :return: The factorial of n
        :raises ValueError: If n is negative or not an integer


        Example usage:
        print(factorial(5))  # Output: 120

    square(x)
        Calculate the square of a number x.
        :param x: A number (int or float)
        :return: The square of x
        :raises ValueError: If x is not a number


        Example usage:
        print(square(4))  # Output: 16
        print(square(2.5))  # Output: 6.25
```

```
PS C:\Users\arell\Music\aiac> python -m pydoc -w C:\Users\arell\Music\aiac\math_utils.py
120
16
27
wrote math_utils.html
PS C:\Users\arell\Music\aiac>
```

Problem 4: Attendance Management Module

Task:

1. Create a module attendance.py with functions:

    o mark_present(student)

- o mark_absent(student)
- o get_attendance(student)

2. Add proper docstrings.
3. Generate and view documentation in terminal and browse

Code:

attendance.py > ...

```python
class Attendance:
    def __init__(self, student_name, date, status):
        """Docstring for Attendance class
        :param: student_name (str), date (str), status (str)
        :return: None
        :exceptions: ValueError for invalid input types or values
        :error handling: Catches ValueError and prompts user to enter valid input
        :side effects: None
        :description: Initializes an Attendance object with the given student name, date, and status.
          Validates the input to ensure that student_name and date are strings and status is either "Present" or "Absent".
        :example usage:
          attendance = Attendance("John Doe", "2024-06-01", "Present")
        """
        if not isinstance(student_name, str) or not isinstance(date, str):
            raise ValueError("Student name and date must be strings.")
        if status not in ["Present", "Absent"]:
            raise ValueError("Status must be either 'Present' or 'Absent'.")

        self.student_name = student_name
        self.date = date
        self.status = status
    def __str__(self):
        """Docstring for __str__ method
        :param: None
        :return: str representation of the Attendance object
        :exceptions: None
        :error handling: None
        :side effects: None
        :description: Returns a string representation of the Attendance object in the format "Student Name:
            Date: YYYY-MM-DD, Status: Present/Absent".
        :example usage:
          attendance = Attendance("John Doe", "2024-06-01", "Present")
          print(attendance)  # Output: Student Name: John Doe, Date: 2024-06-01, Status: Present
        """
        return f"Student Name: {self.student_name}, Date: {self.date}, Status: {self.status}"
attendance = Attendance("John Doe", "2024-06-01", "Present")
print(attendance)

```

Output:

```
NAME
    attendance

CLASSES
    builtins.object
        Attendance

    class Attendance(builtins.object)
     |  Attendance(student_name, date, status)
     |
     |  Methods defined here:
     |
     |  __init__(self, student_name, date, status)
     |      Docstring for Attendance class
     |      :param: student_name (str), date (str), status (str)
     |      :return: None
     |      :exceptions: ValueError for invalid input types or values
     |      :error handling: Catches ValueError and prompts user to enter valid input
     |      :side effects: None
     |      :description: Initializes an Attendance object with the given student name, date, and status.
     |        Validates the input to ensure that student_name and date are strings and status is either "Present" or "Absent".
     |      :example usage:
     |        attendance = Attendance("John Doe", "2024-06-01", "Present")
     |
     |  __str__(self)
     |      Docstring for __str__ method
     |      :param: None
     |      :return: str representation of the Attendance object
     |      :exceptions: None
     |      :error handling: None
     |      :side effects: None
     |      :description: Returns a string representation of the Attendance object in the format "Student Name:
     |          Date: YYYY-MM-DD, Status: Present/Absent".
     |      :example usage:
     |        attendance = Attendance("John Doe", "2024-06-01", "Present")
     |        print(attendance)  # Output: Student Name: John Doe, Date: 2024-06-01, Status: Present
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables
     |
     |  __weakref__
     |      list of weak references to the object
     |      :side effects: None
     |      :description: Returns a string representation of the Attendance object in the format "Student Name:
     |          Date: YYYY-MM-DD, Status: Present/Absent".
     |      :example usage:
     |        attendance = Attendance("John Doe", "2024-06-01", "Present")
     |        print(attendance)  # Output: Student Name: John Doe, Date: 2024-06-01, Status: Present
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables
     |
     |  __weakref__
     |      list of weak references to the object
     |          Date: YYYY-MM-DD, Status: Present/Absent".
     |      :example usage:
     |        attendance = Attendance("John Doe", "2024-06-01", "Present")
     |        print(attendance)  # Output: Student Name: John Doe, Date: 2024-06-01, Status: Present
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables
     |
     |  __weakref__
     |      list of weak references to the object
     |        print(attendance)  # Output: Student Name: John Doe, Date: 2024-06-01, Status: Present
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables
     |
     |  __weakref__
     |      list of weak references to the object
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables
     |
     |  __weakref__
     |      list of weak references to the object
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables
     |
     |  __weakref__
     |      list of weak references to the object
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables
     |
     |  __weakref__
     |      list of weak references to the object
     |  __dict__
     |      dictionary for instance variables
     |
     |  __weakref__
     |      list of weak references to the object
     |
     |  __weakref__
     |      list of weak references to the object
     |
     |      list of weak references to the object

DATA

DATA
    attendance = <attendance.Attendance object>
DATA
    attendance = <attendance.Attendance object>
    attendance = <attendance.Attendance object>
```

Problem 5: File Handling Function

Consider the function:

```
def read_file(filename):
with open(filename, 'r') as f:
return f.read()
```

Task:

1. Write documentation using all three formats.
2. Identify which style best explains exception handling.
3. Justify your recommendation.

Code:

```python
def read_file(filename:str) -> str:
    """
    Reads the contents of a file and returns it as a string.
    :param filename: The name of the file to read
    :return: The contents of the file as a string
    :raises ValueError: If the filename is not a string
    :raises FileNotFoundError: If the file does not exist
        :raises IOError: If there is an error reading the file
    Example usage:
    print(read_file("example.txt"))

    """
    if not isinstance(filename, str):
        raise ValueError("Filename must be a string.")

    try:
        with open(filename, 'r') as file:
            contents = file.read()
            return contents
    except FileNotFoundError:
        raise FileNotFoundError(f"The file '{filename}' does not exist.")
    except IOError as e:
        raise IOError(f"An error occurred while reading the file: {e}")
try:
    print(read_file("example.txt"))
except ValueError as ve:
    print(f"ValueError: {ve}")
except FileNotFoundError as fnfe:
    print(f"FileNotFoundError: {fnfe}")
except IOError as ioe:
    print(f"IOError: {ioe}")

print(read_file("example.txt"))
```

Output:

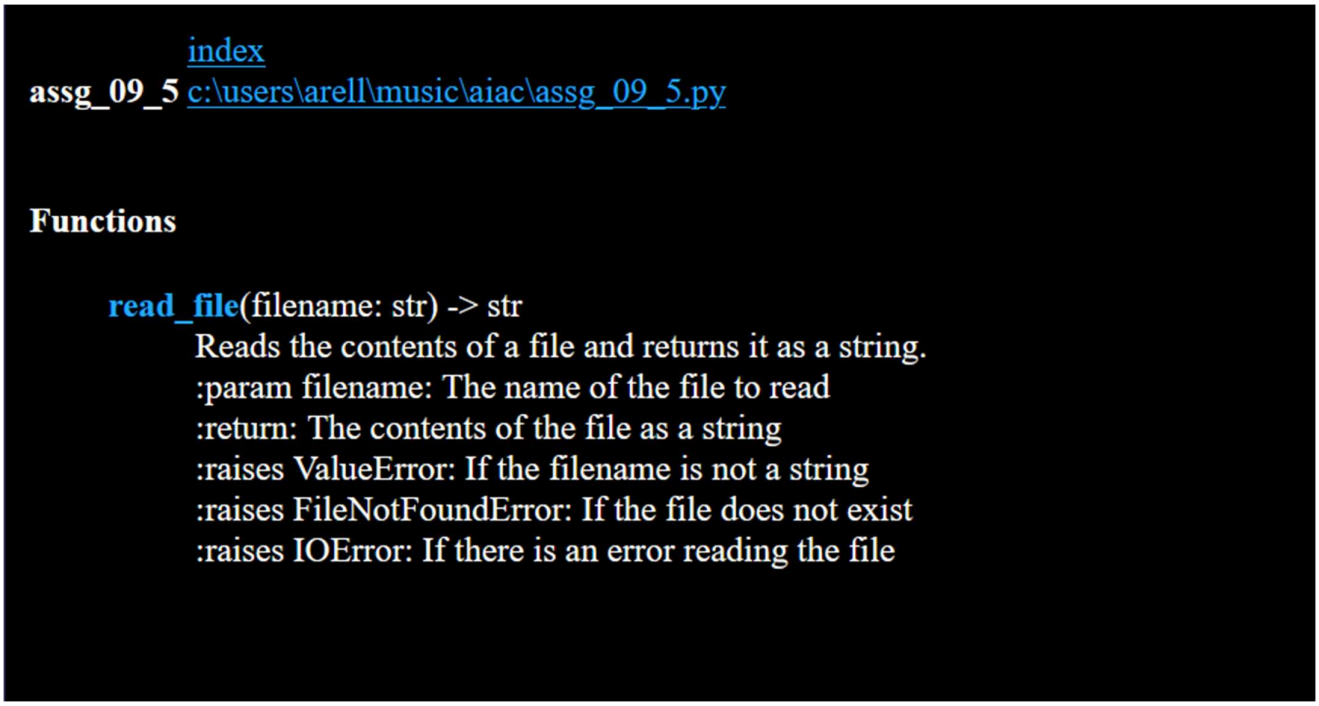
```
PS C:\Users\arell\Music\aiac> python -m pydoc C:\Users\arell\Music\aiac\assg_09_5.py
hello world!.
Help on module assg_09_5:

NAME
    assg_09_5

FUNCTIONS
    read_file(filename: str) -> str
        Reads the contents of a file and returns it as a string.
        :param filename: The name of the file to read
        :return: The contents of the file as a string
        :raises ValueError: If the filename is not a string
        :raises FileNotFoundError: If the file does not exist
        :raises IOError: If there is an error reading the file

FILE
    c:\users\arell\music\aiac\assg_09_5.py


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



Incline comments:

assg_09_5.py > ...

```python
# Define a function that reads a file and returns its contents as a string
def read_file(filename: str) -> str:
    """
    Reads the contents of a file and returns it as a string.
    :param filename: The name of the file to read
    :return: The contents of the file as a string
    :raises ValueError: If the filename is not a string
    :raises FileNotFoundError: If the file does not exist
    :raises IOError: If there is an error reading the file
    """
    # Check if the filename parameter is a string, raise ValueError if not
    if not isinstance(filename, str):
        raise ValueError("Filename must be a string.")

    # Try to open and read the file
    try:
        # Open the file in read mode
        with open(filename, 'r') as file:
            # Read the entire file contents into a string variable
            contents = file.read()
            # Return the file contents
            return contents
    # Catch FileNotFoundError if the file doesn't exist
    except FileNotFoundError:
        raise FileNotFoundError(f"The file '{filename}' does not exist.")
    # Catch IOError for any other file reading errors
    except IOError as e:
        raise IOError(f"An error occurred while reading the file: {e}")


# Attempt to read example.txt and print its contents
try:
    print(read_file("example.txt"))
# Handle ValueError if filename is not a string
except ValueError as ve:
    print(f"ValueError: {ve}")
# Handle FileNotFoundError if the file doesn't exist
except FileNotFoundError as fnfe:
    print(f"FileNotFoundError: {fnfe}")
# Handle IOError for any file reading errors
except IOError as ioe:
    print(f"IOError: {ioe}")

```