

Assignment 9.3(Present number)

2303A51132

Bt:17

Task 1: Basic Docstring Generation

Scenario

You are developing a utility function that processes numerical lists and must be properly documented for future maintenance.

#Code:

```
#You are developing a utility function that processes numerical lists and must be properly
```

```
#documented for future maintenance.
```

```
def calculate_statistics(numbers):
```

Calculate and return basic statistics (mean, median, mode) for a list of numbers.

Parameters:

numbers (list): A list of numerical values.

Returns:

dict: A dictionary containing the mean, median, and mode of the input numbers.

Raises:

```
ValueError: If the input is not a list or if the list is empty
```

```
if not isinstance(numbers, list):
```

```
    raise ValueError("Input must be a list."
```

```
if len(numbers) == 0:
```

```
    raise ValueError("Input list cannot be empty.")
```

```
# Calculate mean
```

```
mean = sum(numbers) / len(numbers)
```

```
# Calculate median
```

```
sorted_numbers = sorted(numbers)
```

```
n = len(sorted_numbers)
```

```
if n % 2 == 0:
```

```
    median = (sorted_numbers[n // 2 - 1] + sorted_numbers[n // 2]) / 2
```

```
else:
```

```
    median = sorted_numbers[n // 2]
```

```
# Calculate mode
```

```
from collections import Counter
```

```
count = Counter(numbers)
```

```
mode_data = count.most_common()
```

```

mode = [num for num, freq in mode_data if freq == mode_data[0][1]]

return {
    "mean": mean,
    "median": median,
    "mode": mode
}

```

Example usage:

```

numbers = [1, 2, 2, 3, 4]

stats = calculate_statistics(numbers)

print(stats)

```

#Output:

```

PS C:\Users\ashri\OneDrive\Desktop\hh> & C:/Users/ashri/OneDrive/Desktop/hh/.venv/Scripts/python.exe c:/Users/ashri/OneDrive/Desktop/hh/18.py
{'mean': 2.4, 'median': 2, 'mode': [2]}
PS C:\Users\ashri\OneDrive\Desktop\hh> & C:/Users/ashri/OneDrive/Desktop/hh/.venv/Scripts/python.exe c:/Users/ashri/OneDrive/Desktop/hh/18.py
{'mean': 2.4, 'median': 2, 'mode': [2]}
PS C:\Users\ashri\OneDrive\Desktop\hh>

```

Task 2: Automatic Inline Comments

Scenario

You are developing a student management module that must be easy to understand for new

Developers

#Code:

#write a python code to developing a student management module that must be easy to understand for new

#developers it must inculd about the stundent class age and hostel state

class Student:

"""

A class to represent a student.

Attributes:

name (str): The name of the student.

age (int): The age of the student.

hostel (bool): Whether the student is staying in a hostel or not.

Methods:

__init__(self, name, age, hostel): Initializes the Student object with name, age, and hostel status.

__str__(self): Returns a string representation of the Student object.

"""

def __init__(self, name, age, hostel):

"""

Initializes the Student object with name, age, and hostel status.

Parameters:

name (str): The name of the student.

age (int): The age of the student.

hostel (bool): Whether the student is staying in a hostel or not.

"""

```
self.name = name
```

```
self.age = age
```

```
self.hostel = hostel
```

```
def __str__(self):
```

"""

Returns a string representation of the Student object.

Returns:

str: A string describing the student.

"""

```
hostel_status = "staying in a hostel" if self.hostel else "not staying in a hostel"
```

```
return f"{self.name} is {self.age} years old and is {hostel_status}."  
  
# Example usage:  
  
student1 = Student("Alice", 20, True)  
  
student2 = Student("Bob", 22, False)  
  
print(student1) # Output: Alice is 20 years old and is staying in a hostel.  
  
print(student2) # Output: Bob is 22 years old and is not staying in a hostel.
```

#output:

```
Alice is 20 years old and is staying in a hostel.  
Bob is 22 years old and is not staying in a hostel.  
PS C:\Users\ashri\OneDrive\Desktop\hh>
```

Task 3: Module-Level and Function-Level Documentation Scenario

You are building a small calculator module that will be shared across multiple projects and requires structured documentation

#Code:

```
#write a python code to building a small calculator module that will be shared across multiple projects and
```

```
#requires structured documentation. it must have basic operations like addition, subtraction, multiplication, and division.
```

```
class Calculator:
```

```
    """
```

```
    A simple calculator class to perform basic arithmetic operations.
```

Methods:

add(a, b): Returns the sum of a and b.

subtract(a, b): Returns the difference of a and b.

multiply(a, b): Returns the product of a and b.

divide(a, b): Returns the quotient of a and b. Raises ValueError if b is zero.

"""

@staticmethod

def add(a, b):

"""

Returns the sum of a and b.

Parameters:

a (float): The first number.

b (float): The second number.

Returns:

float: The sum of a and b.

"""

return a + b

@staticmethod

def subtract(a, b):

"""

Returns the difference of a and b.

Parameters:

a (float): The first number.

b (float): The second number.

Returns:

float: The difference of a and b.

"""

return a - b

@staticmethod

def multiply(a, b):

"""

Returns the product of a and b.

Parameters:

a (float): The first number.

b (float): The second number.

Returns:

float: The product of a and b.

"""

return a * b

@staticmethod

def divide(a, b):

"""

Returns the quotient of a and b. Raises ValueError if b is zero.

Parameters:

a (float): The first number.

b (float): The second number.

Returns:

float: The quotient of a and b.

Raises:

ValueError: If b is zero.

"""

if b == 0:

 raise ValueError("Cannot divide by zero.")

return a / b

Example usage:

```
calc = Calculator()
```

```
print(calc.add(10, 5))
```

```
print(calc.subtract(10, 5))
```

```
print(calc.multiply(10, 5))
```

```
print(calc.divide(10, 5))
```

```
try:
```

```
    print(calc.divide(10, 0))
```

```
except ValueError as e:
```

```
    print(e)
```

#OutPut:

15

5

50

2.0

Cannot divide by zero.