

System Requirements

Specification Index

For

Calculate Cumulative Sum using NumPy

(Topic: Advanced NumPy)

Version 1.0

Cumulative Sum Analysis Console

Project Abstract

The Cumulative Sum Analysis Console is a Python-based application designed to calculate the cumulative sum of sales data. The system handles input data, processes it using both NumPy and Python's built-in list structures, and provides a time measurement of the operation for performance comparison. The primary goal of this system is to showcase the efficiency of NumPy over native Python lists for cumulative sum calculations in large datasets, providing useful insights into the speed benefits of using optimized libraries.

Business Requirements:

- **Input Validation:** Ensure the sales data is valid and convertible into a NumPy array.
 - **Cumulative Sum Calculation:** Compute the cumulative sum of sales data using both NumPy and Python methods.
 - **Performance Comparison:** Measure the time taken for both NumPy and Python methods to calculate the cumulative sum and provide a comparison.
-

Constraints:

Input Requirements:

- **Sales Data:**
 - Must be a list or array of numerical values (integers or floats).
 - Example: `[100.5, 200.0, 50.0, 150.5, 300.0]`

Output Requirements:

1. **Cumulative Sum Calculation:**
 - The system must calculate the cumulative sum using both NumPy and Python list operations.
 - Return the results as an array for NumPy and a list for Python.
2. **Performance Measurement:**
 - The system must measure the time taken for both methods

- (NumPy and Python).
 - Provide results in seconds.
-

Conversion Constraints:

1. Input Conversion:

- The sales data should be processed into a NumPy array using `np.array()` for NumPy-based calculations.
- Ensure that invalid data types are rejected and raise appropriate errors if conversion fails.

2. Cumulative Sum Operations:

- NumPy Cumulative Sum:
 - Use `np.cumsum()` to calculate the cumulative sum efficiently.
- Python Cumulative Sum:
 - Use Python's native list and a loop to manually calculate the cumulative sum.

3. Performance Measurement:

- Time both the NumPy and Python-based cumulative sum calculations using the `time.time()` method.
-

Required Output Format:

1. Cumulative Sum Results:

- Show cumulative sum for both NumPy and Python methods.
- Example output format:
 - NumPy Cumulative Sum: [100.5, 300.5, 350.5, 501.0, 801.0]
 - Python Cumulative Sum: [100.5, 300.5, 350.5, 501.0, 801.0]

2. Performance Output:

- Display the time taken for both operations.
 - Example output:
 - Time for NumPy: 0.0002 seconds
 - Time for Python: 0.0005 seconds
-

Template Code Structure:

1. **Initialization and Input Validation:**
 - Initialize the sales data and ensure it is in a valid format.
 - Raise errors for invalid input (non-numeric data or empty lists).
2. **Cumulative Sum Calculation:**
 - Implement both methods to compute cumulative sums: one using NumPy (`np.cumsum()`) and the other using Python lists.
3. **Performance Measurement:**
 - Measure the time taken for both operations using `time.time()`.
4. **Output Display:**
 - Display the cumulative sums and the time taken for each method in a readable format.

Execution Steps to Follow:

- All actions like build, compile, running application, running test cases will be through Command Terminal.
- To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal
- This editor Auto Saves the code
- If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page)
.
- These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
- To setup environment:
You can run the application without importing any packages
- To launch application:
python3 mainclass.py
To run Test cases:
python3 -m unittest

Screen shot to run the program

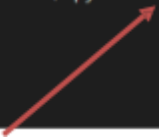
To run the application

A screenshot of a terminal window with a dark background. The prompt is 'coder@dighe20250227t070305r21fj5p3: /home/myproject/dighegmailcom_20250227t070305\$'. The command 'python3 <<scriptname>>.py' is entered. A red arrow points from the text 'python3 mainclass.py python3' below to the '<<scriptname>>.py' part of the command in the terminal.

```
OK
coder@dighe20250227t070305r21fj5p3: /home/myproject/dighegmailcom_20250227t070305$ python3 <<scriptname>>.py
```

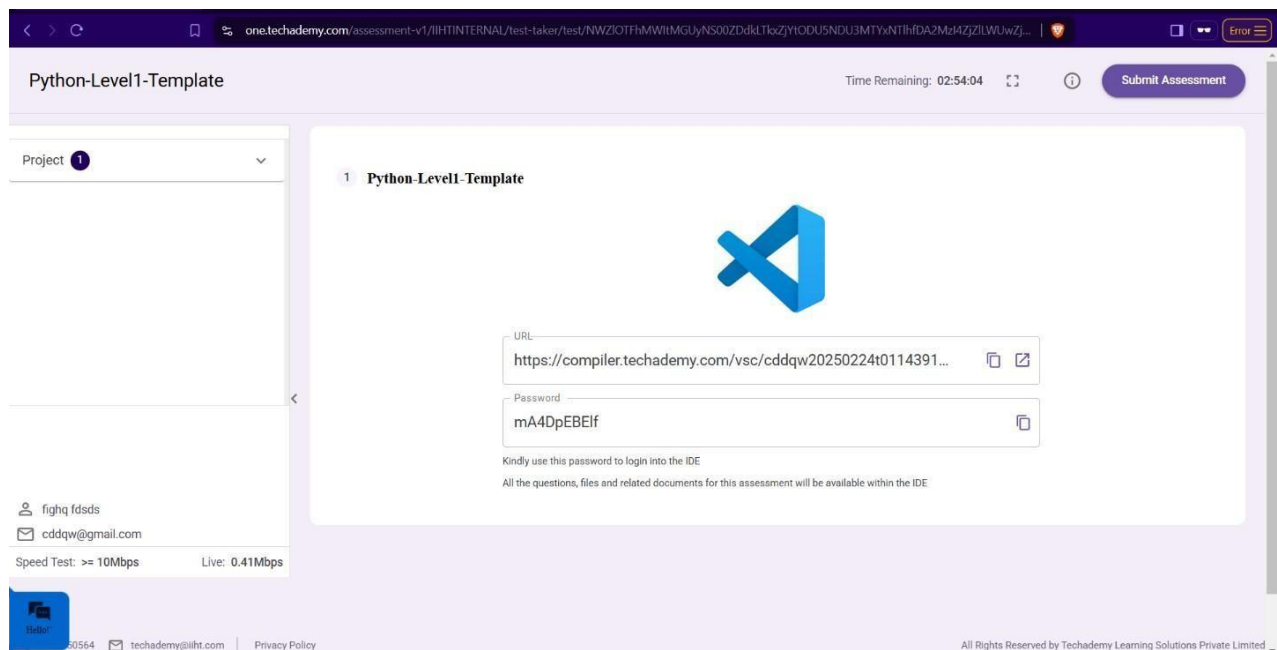
python3 mainclass.py python3

```
● coder@dighe20250227t070305rz1fj5p3:/home/myproject/dighegmailcom_20250227T070305$ python3 -m unittest
TestBoundary = Passed
.TestExceptional = Passed
.TestCalculateTotalDonations = Failed
.TestCalculateTotalStockValue = Failed
.TestCheckFrankWhiteDonated = Failed
```



To run the testcase

python3 -m unittest7



- **Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.**