

# **System Requirements**

## **Specification Index**

**For**

### **Cumulative Sum Calculation for Sales using NumPy**

**(Topic: Advanced NumPy)**

**Version 1.0**

# Cumulative Sales Sum Analysis Console

## Project Abstract

The Cumulative Sales Sum Analysis system is a Python application that provides efficient calculations of cumulative sales figures, allowing businesses to evaluate their sales performance over time. The system offers two approaches for computing cumulative sums: one using NumPy for optimized performance and another using native Python lists for comparison. The system is designed to handle a variety of sales data inputs, ensuring accuracy and efficiency. The core functionality of the system allows for the calculation of cumulative sums, benchmarking performance between NumPy and Python lists, and providing results to help businesses analyze trends in their sales data.

## Business Requirements

- The system must calculate the cumulative sum of sales data.
  - The system should support two methods of calculating the cumulative sum: one using NumPy and another using native Python lists.
  - The system must measure the time taken for both methods to help evaluate the performance difference.
  - The system should handle errors, such as invalid or empty data inputs.
- 

## Input Requirements

1. Sales Data:
    - Must be stored as a list of numerical values (floats).
    - Example: `[10.5, 20.0, 30.0, 40.0]`
    - Data must be provided as a list of numbers. If any data cannot be converted to a float, an error is raised.
  2. Empty Data:
    - The input data cannot be empty.
    - If the data is empty, the system raises an error: `ValueError("data is empty")`.
- 

## Conversion Constraints

1. Data Type Conversion:
  - Sales data must be converted into a NumPy array of type

- `np.float32`.
    - If conversion fails, the system raises an error: `ValueError("could not convert string to float")`.
  - 2. Cumulative Sum Calculation:
    - NumPy Approach: Use NumPy's `np.cumsum()` to calculate the cumulative sum.
    - Python Approach: Calculate the cumulative sum using a loop and Python lists.
    - Both methods should return the cumulative sum of sales data.
- 

## Output Constraints

1. Results Display:
    - Both cumulative sum methods must return a list or array with the cumulative sum for each item in the sales data.
    - Performance Metrics: Measure and compare the execution time of both methods (NumPy and Python).
    - The output will show how long each method took to process the same set of sales data.
- 

## Required Output Format

The output should provide:

1. Cumulative sum results for each method:
    - NumPy Result: `[sum_1, sum_2, sum_3, ...]`
    - Python Result: `[sum_1, sum_2, sum_3, ...]`
  2. Performance Times:
    - Display the time taken for NumPy and Python operations.
- 

## Template Code Structure

1. Data Initialization:
  - Store sales data as a NumPy array.
  - Raise error if data is empty or if conversion fails.
2. Cumulative Sum Calculation Functions:
  - `cumulative_sum_numpy()` — Calculates cumulative sum using NumPy.
  - `cumulative_sum_python()` — Calculates cumulative sum using Python lists.

### 3. Performance Measurement:

- `measure_time()` — Measures the execution time for both methods and returns the time taken by each.

### 4. Execution Logic:

- Initialize sales data.
  - Perform cumulative sum calculation with both methods.
  - Measure the time taken by each method.
- 

## Example Usage

### 1. Input:

- `sales_data = [10.5, 20.0, 30.0, 40.0]`

### 2. Output:

- NumPy Cumulative Sum Result:
  - `[10.5, 30.5, 60.5, 100.5]`
- Python Cumulative Sum Result:
  - `[10.5, 30.5, 60.5, 100.5]`
- Execution Time Comparison:
  - NumPy Time: `0.0005 seconds`
  - Python Time: `0.0025 seconds`

### **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) then you need to use **CTRL+Shift+B** -command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
- 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**
- Before Final Submission also, you need to use **CTRL+Shift+B** - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

### **Screen shot to run the program**

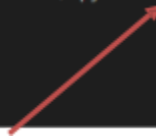
**To run the application**

```
OK
coder@dighe20250227t070305rz1fj5p3:/home/myproject/dighegmailcom_20250227T070305$ python3 <<scriptname>>.py []
```



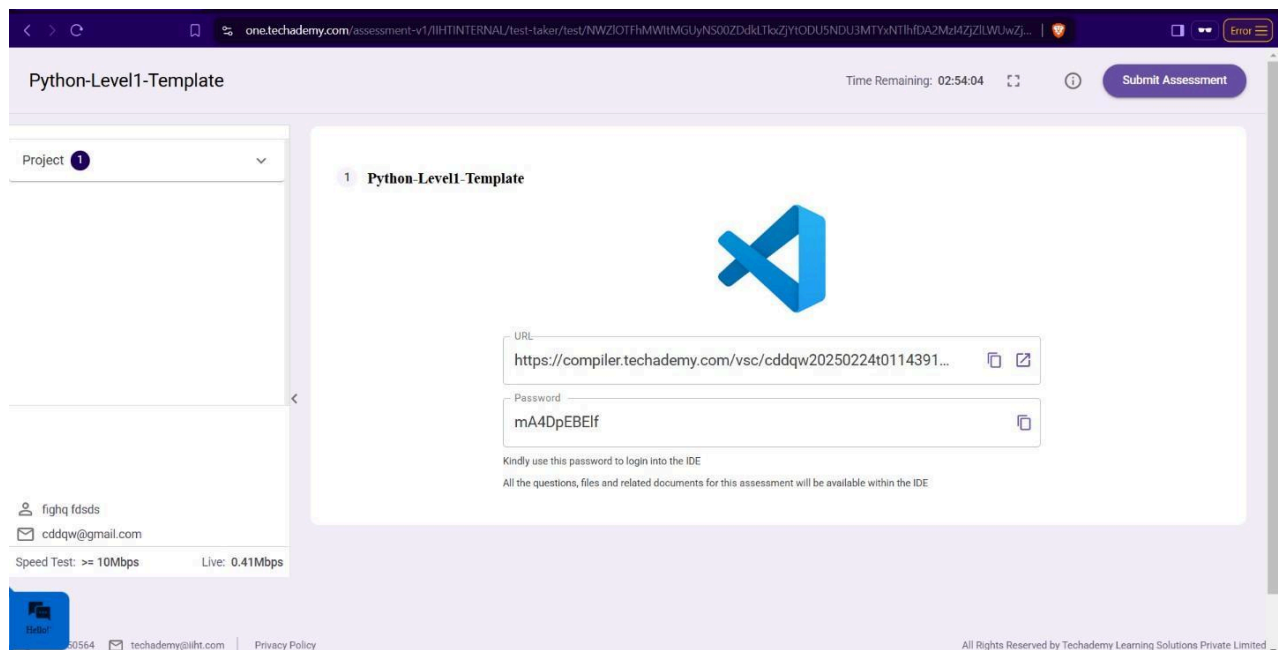
**python3 mainclass.py python3**

```
● 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.