

System Requirements

Specification Index

For

Performance Comparison using NumPy and Python Lists

(Topic: Advanced NumPy)

Version 1.0

Performance Comparison Console

Project Abstract

The Performance Comparison application is a Python-based tool that compares the performance of two data structures: Python lists and NumPy arrays. This tool is designed to perform element-wise summation of two lists and two arrays, measure the time taken for each operation, and compare the performance differences between these two structures. The tool will help users understand the computational efficiency of Python lists versus NumPy arrays when working with large datasets. This performance comparison is particularly important for data scientists and engineers working with numerical data and seeking to optimize their operations.

Business Requirements

- The system should be able to handle large datasets for performance comparison (default size of 1,000,000 elements).
 - The system must compare the performance of Python lists and NumPy arrays for element-wise summation.
 - The system should track the time taken for both list and array summations and output the comparison.
 - The system must handle errors gracefully, including type mismatches in the lists and arrays.
-

Constraints

Input Requirements

1. Python Lists:
 - Two lists must be provided, `list_a` and `list_b`, both containing numeric values.
 - Example: `[0.5, 1.2, 3.4]`
2. NumPy Arrays:
 - Two arrays must be provided, `array_a` and `array_b`, both containing numeric values.
 - Example: `np.array([0.5, 1.2, 3.4])`

Performance Requirements

- The application must handle large datasets, with a default dataset size of 1,000,000 elements.
 - The summation operation must be efficient and optimized for both Python lists and NumPy arrays.
 - Time measurements must be accurate, and the comparison should reflect real performance differences.
-

Conversion Constraints

1. **Python List Summation:**
 - Convert the lists to numerical values if they are not in the correct format (i.e., integers or floats).
 - Perform element-wise summation using a Python list comprehension.
 2. **NumPy Array Summation:**
 - Ensure both NumPy arrays are properly initialized with numeric values.
 - Perform element-wise summation using NumPy's vectorized operations, which are more optimized than Python's list operations.
-

Output Constraints

1. **Display Results:**
 - Show the time taken to sum the Python lists.
 - Show the time taken to sum the NumPy arrays.
 - Provide a clear comparison of the performance of the two data structures.
-

Required Output Format

The output must show the time taken for both operations, and display them in a readable format:

- "Time taken for summing Python lists: {list_time} seconds"
 - "Time taken for summing NumPy arrays: {array_time} seconds"
-

Template Code Structure:

- 1. Initialization:**
 - **Create lists and arrays using random values.**
 - **Initialize the PerformanceComparison class with the desired size.**
- 2. Sum Functions:**
 - **sum_lists(): Perform element-wise summation of two Python lists.**
 - **sum_arrays(): Perform element-wise summation of two NumPy arrays.**
- 3. Performance Measurement:**
 - **measure_time(): Measure the time taken to sum both lists and arrays.**
- 4. Output Section:**
 - **Display the performance results, including the time taken for each operation.**

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@dighe20250227t070305r1fj5p3:/home/myproject/dighegmailcom_20250227T070305$ python3 <<scriptname>>.py
```

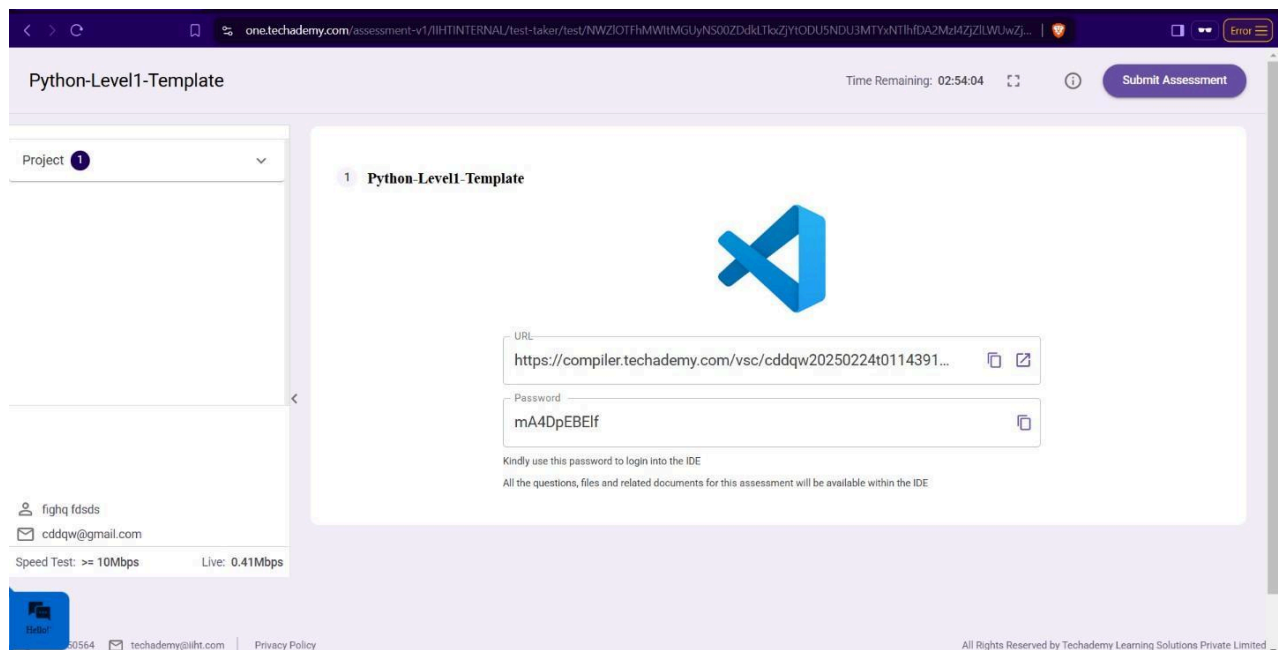
python3 mainclass.py

```
● 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.**