SystemRequirements Specification Index

For

- Python Basics and NumPy, Pandas
- Usecase No 10
  1.0

IIHT Pvt. Ltd.

fullstack@iiht.com

Use Case No 1: Student Fee Management System (StudentFeeManagementSystem.py)

Dataset :

```
student_fees = {
   "S101": {"name": "Alice", "grade": "5th", "fees_paid": 2000, "total_fees": 5000},
   "S102": {"name": "Bob", "grade": "6th", "fees_paid": 3500, "total_fees": 5000},
   "S103": {"name": "Charlie", "grade": "7th", "fees_paid": 5000, "total_fees": 5000},
   "S104": {"name": "David", "grade": "8th", "fees_paid": 1000, "total_fees": 6000},
   "S105": {"name": "Emma", "grade": "9th", "fees_paid": 4000, "total_fees": 7000}
}
```

Implement a Python function to save student fee records to a file. Define: save_to_file()
The function should:

- Write student fee data to fees_data.txt. (You can write in any format)

Write a Python function to retrieve and display student's fee details from the file.
Define: get_fees(student_name)
The function should:
Read data from fees_data.txt.

- Search for student's record and retrieve his/her fee paid details.
- Return the fee paid value from the method as number and without currency symbol.
- If student name not found then return 0;

Write a Python function to find if total fee is paid
Define:is_total_fee_paid()
The function should:
Read data from fees_data.txt.

- Search for student's record and retrieve his/her fee paid details.
- Return TRUE if the fee paid value is equal to total fees, else return FALSE
- If student name not found then return FALSE.

Usecase No 2: Transport Management System (TransportManagementSystem.py)

Dataset:

```
transport_data = {
   "T101": {"route": "New York - Boston", "passengers": 40, "fare_per_passenger": 15},
   "T102": {"route": "Los Angeles - San Francisco", "passengers": 30, "fare_per_passenger": 20},
   "T103": {"route": "Chicago - Detroit", "passengers": 25, "fare_per_passenger": 25},
   "T104": {"route": "Houston - Dallas", "passengers": 50, "fare_per_passenger": 10},
   "T105": {"route": "Miami - Orlando", "passengers": 20, "fare_per_passenger": 30}
}
```

Write a Python function to calculate the total revenue for a given trip.
Define: calculate_trip_revenue(trip_id)
The function should:

- Look up the trip in transport_data using the given trip ID.
- Calculate the total revenue using passengers * fare_per_passenger.
- Return total revenue as number without any currency symbol.
- If trip ID not found then return 0

Write a Python function to validate if a trip meets a minimum passenger requirement.
Define: validate_trip(trip_id, min_passengers)
The function should:

- Retrieve the number of passengers for the given trip ID.
- Compare it with the required minimum passenger count.
- Return a TRUE of passenger count is greater than equal to min_passengers, else return FALSE.
- IF trip_id not found then return FALSE

Write a Python function to calculate the total revenue generated from all trips.
Define: total_transport_revenue()
The function should:

- Iterate through all transport records.
- Compute total revenue by summing up passengers * fare_per_passenger for all trips.
- Return total amount of revenue as number without any currency sign.

Use Case No 3: Warehouse Management System (WarehouseManagementSystem.py)

Dataset:
Structure is as follows:
"<Code>":{"Item Name", <stock>, <price per unit>}

```
inventory = {
   "W101": ("Laptops", 50, 800),
   "W102": ("Smartphones", 100, 500),
   "W103": ("Headphones", 200, 50),
   "W104": ("Keyboards", 150, 30),
   "W105": ("Monitors", 75, 200)
}
```

1. Write a Python function to find the most expensive item in the warehouse.
Define: most_expensive_item(inventory)
The function should:

- Iterate through all inventory records.
- Compare item prices to find the most expensive one.
- Return the most expensive item as a tuple (Item Name, Quantity, Price per Unit).

2.Write a Python function to calculate the total stock in the warehouse.

Define: total_items_in_warehouse(inventory)
The function should:

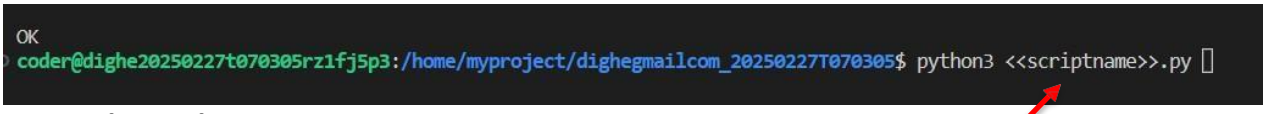
- Sum up all item quantities available in the warehouse.
- Return the total count of items.

Execution Steps to Follow:

- All actions like build, compile, running application, running test cases will be through the 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 TransportManagementSystem.py
  python3 WarehouseManagementSystem.py
  python3 StudentFeeManagementSystem.py
- To run Test cases:
  python3 -m unittest

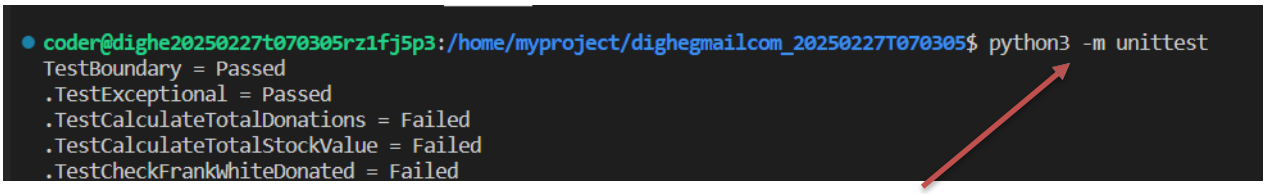
- Before Final Submission also, you need to use **CTRL+Shift+B** - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

Screen shot to run the program



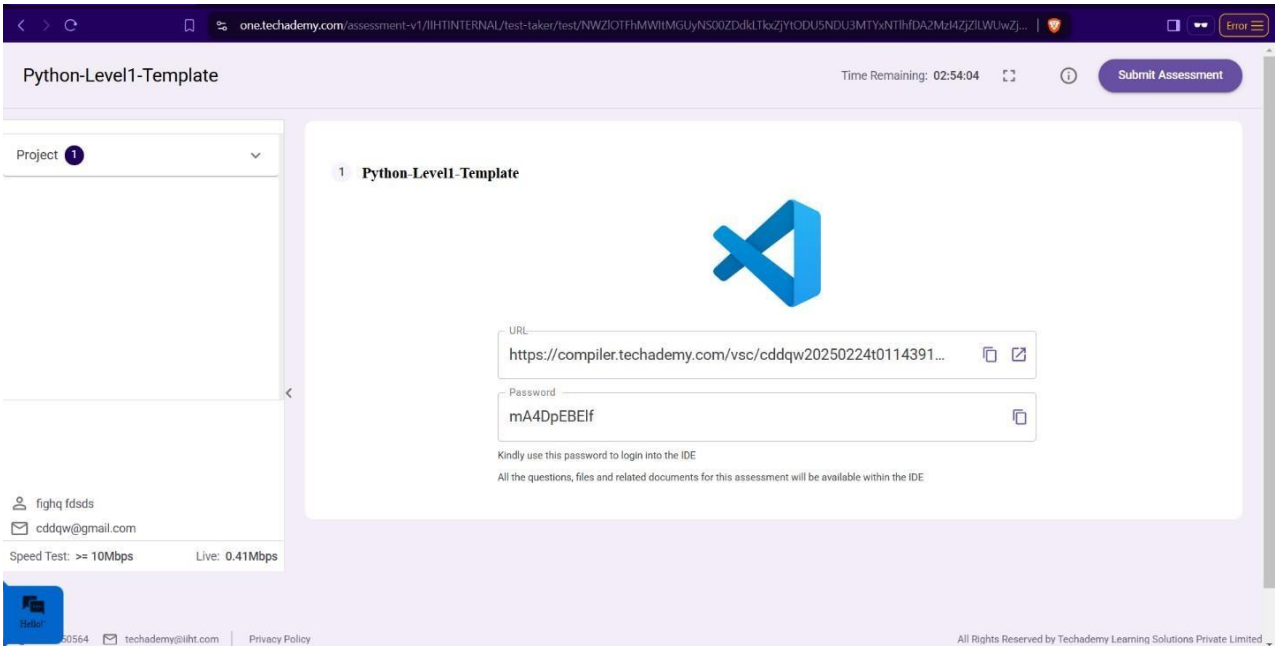
To run the application

python3 TransportManagementSystem.py
python3 WarehouseManagementSystem.py
python3 StudentFeeManagementSystem.py



To run the testcase

python3 -m unittest



- **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.**