System Requirements Specification Index

For

Python Basics and NumPy, Pandas

Usecase 4

1.0

IHT Pvt. Ltd.

Task 1: Blood Bank Management (`blood_bank_management.py`)

Dataset:

```
blood_groups = ["A+", "B+", "O-", "AB-"]
units_available = [10, 8, 4, 3]
```

Function: `add_blood_group(df, group, units)`

Purpose:
Add a new blood group and the corresponding number of units to the DataFrame.

Parameters:

- `df (pd.DataFrame)`: The existing blood bank dataset.
- `group (str)`: The blood group to be added (e.g., "B-").
- `units (int)`: Number of units available for the new blood group.

Returns:

- Updated DataFrame with the new entry.

Instructions:
Append a new row to the DataFrame with the provided group and units, then return the updated DataFrame.

Function: `total_units(df)`

Purpose:
Calculate the total number of blood units available across all blood groups.

Parameters:

- `df (pd.DataFrame)`: The blood bank dataset.

Returns:

- Sum of units (integer).

Instructions:
Sum all values under the "Units Available" column and return the result.

Function: `low_stock_groups(df, threshold=5)`

Purpose:
Retrieve all blood groups where available units are less than the given threshold.

Parameters:

- `df (pd.DataFrame)`: The blood bank dataset.
- `threshold (int)`: The limit to consider a stock as low.

Returns:

- Filtered DataFrame of low stock groups.

Instructions:
Filter the DataFrame to include only rows where units are less than the specified threshold.

Task 2: Employee Leave Management (`employee_leave_management.py`)

Dataset:

```
employee_ids = [101, 102, 103, 104]
employee_names = ["Alice", "Bob", "Charlie", "Diana"]
leaves_taken = [5, 2, 8, 1]

leave_df = pd.DataFrame({
    "Employee ID": employee_ids,
    "Name": employee_names,
    "Leaves Taken": leaves_taken
})
```

Function: `total_leaves_taken(df)`

Purpose:
Calculate the total number of leaves taken by all employees.

Parameters:

- `df (pd.DataFrame)`: The employee leave dataset.

Returns:

- Integer representing total leaves.

Instructions:
Aggregate the "Leaves Taken" column to get the total number of leaves.

Function: `employees_exceeding_leaves(df, limit=5)`

Purpose:
Get employees who have taken more than a specified number of leaves.

Parameters:

- `df (pd.DataFrame)`
- `limit (int)`: Leave limit threshold.

Returns:

- Filtered DataFrame of employees exceeding the limit.

Instructions:
Filter the rows where the "Leaves Taken" value is greater than the given limit.

Function: `average_leaves_taken(df)`

Purpose:
Find the average number of leaves taken per employee.

Parameters:

- `df (pd.DataFrame)`: The employee leave dataset.

Returns:

- Float value representing the average.

Instructions:
Calculate and return the mean of the "Leaves Taken" column.

Task 3: Food Classification System (`food.py`)

Dataset (File: `food_items.txt`):

```
Pizza,Veg
Chicken Wings,Non-Veg
Fries,Veg
Mutton Curry,Non-Veg
Burger,Veg
Fish Fry,Non-Veg
Paneer Tikka,Veg
Egg Roll,Non-Veg
```

Function: `read_food_items(file_path)`

Purpose:
Read food items from a file and convert them into a structured list of tuples.

Parameters:

- `file_path (str)`: Path to the input file.

Returns:

- A list of tuples in the format [(item_name, item_type), …]

Instructions:
Open the file and read each line. Split each line into food name and type, then return the list of valid entries. Handle missing or malformed lines appropriately. Also handle file-not-found errors gracefully.

Function: `classify_food_items(food_items)`

Purpose:
Group food items into Veg and Non-Veg categories.

Parameters:

- `food_items (list)`: List of tuples with food name and type.

Returns:

- Dictionary with keys `'Veg'` and `'Non-Veg'`, each holding a list of item names.

Instructions:
Loop through each item and sort them into the correct category based on their type.

.

Execution Steps to Follow:

1. All actions like build, compile, running application,running test cases will be through Command Terminal.
2. To open the command terminal the test takers need to go to Application menu(Three horizontal lines at left top) -> Terminal -> New Terminal
3. This editor auto-saves the code
4. These are time-bound assessments the timer would stop if you log out and while logging in back in using the same credentials, the timer would resume from the same time it was stopped from the previous logout.
5. To set up the environment:

You can run the application without importing any packages

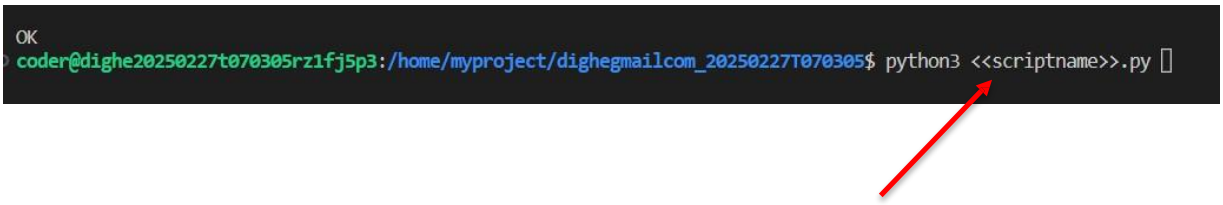
6. To launch the application:

- **python3 blood_bank_management.py**
- **python3 food.py**
- **python3 employee_leave_management.py**

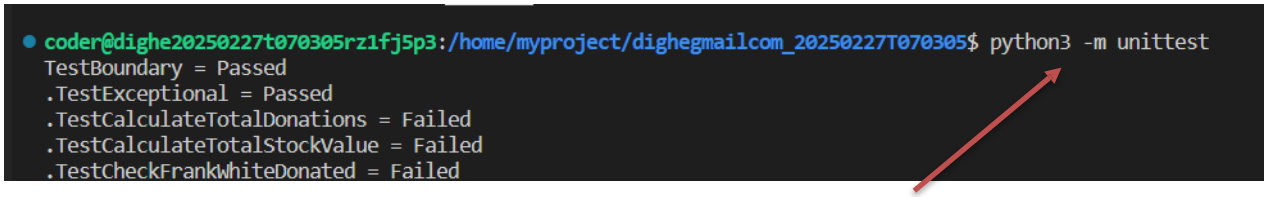
To run Test cases:

- python3 -m unittest

Screen shot to run the program

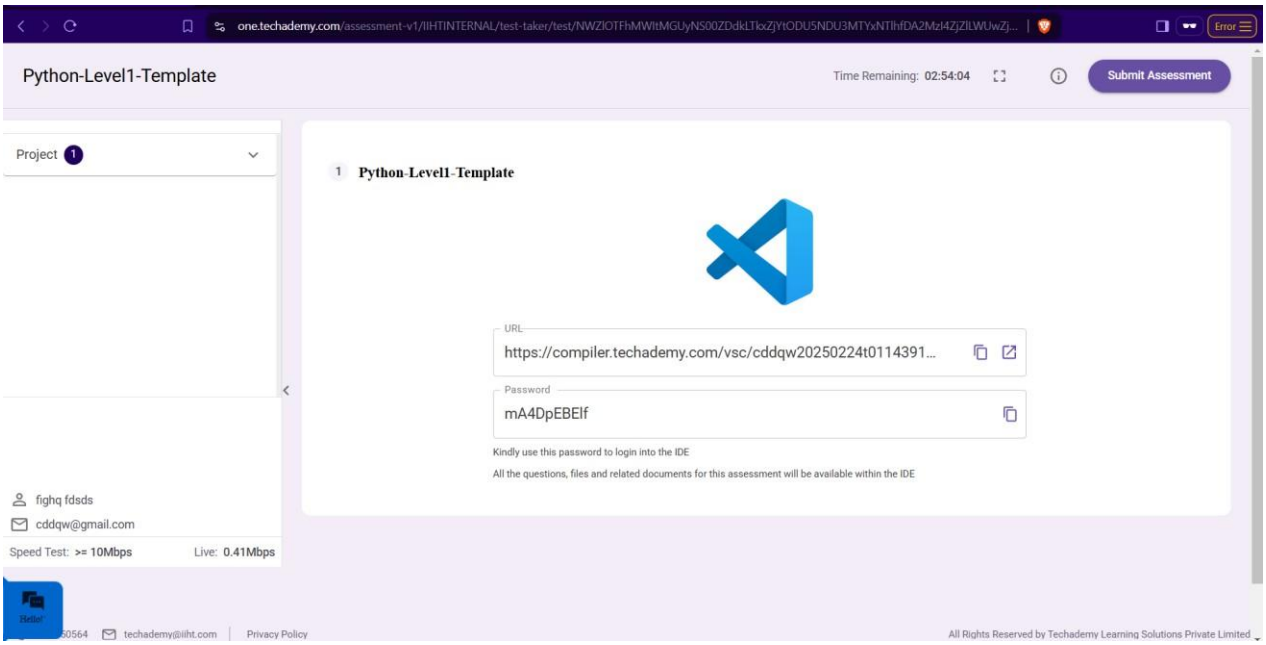


- **python3** `blood_bank_management.py`
- **python3** `employee_leave_management.py`
- **python3 food.py**



To run the testcase

- **python3 -m unittest**



7.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.