
SystemRequirements Specification Index

For

Python Basics and NumPy, Pandas

Usecase 4

1.0

Task 1: Blood Bank Management (BloodBankManagementSystem.py)

In this use case, you are provided with a dataset containing blood group types and the number of units available for each. Your task is to process this information using Python and pandas to manage and monitor blood inventory levels effectively.

The first objective is to **calculate the total number of blood units available across all blood groups**, providing an overall measure of the current stock level.

The second objective is to **identify all blood groups with stock below a given threshold** (default 5 units), allowing the blood bank to prioritize replenishment and ensure critical blood types are always available for emergencies.

Dataset:

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

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:

- return type : int {Sum of units }

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 which is equal to 5 .

Parameters:

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

Returns:

- Return type : dataframe {filter the 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`)

In this use case, you are provided with a dataset containing employee IDs, names, and the number of leaves taken by each employee. Your task is to process and analyze this information using Python and pandas to monitor leave usage and ensure adherence to company policies.

The first objective is to **calculate the total number of leaves taken by all employees**, giving management a clear overview of overall absenteeism.

The second objective is to **identify employees who have exceeded a specified leave limit** (default 5 leaves), enabling targeted follow-up to maintain workforce productivity.

The third objective is to **determine the average number of leaves taken per employee**, providing insight into general leave patterns and helping in resource planning and policy adjustments.

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:

- return type :`int` {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 equal to 5 .

Returns:

- Return type : dataframe { Filtered DataFrame of employees exceeding the limit use the ID to return in the dataframe }.

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:

- return type :Float {return the float value value representing the average with one decimal }.

Instructions:

Calculate and return the average of the "Leaves Taken" column.

Task 3: Food Classification System (food.py)

You are provided with a text file containing a list of food items along with their category (Veg or Non-Veg). Your task is to process this data using Python to organize and categorize food items for better menu management and classification.

The first objective is to **read food items from the file and convert them into a structured list of tuples**, where each tuple contains the food name and its type. This ensures that all items are captured in a consistent, machine-readable format while gracefully handling missing or malformed data as well as file-not-found errors.

The second objective is to **classify food items into Veg and Non-Veg categories**, grouping them into a dictionary with keys 'Veg' and 'Non-Veg'. This helps in quickly segregating dishes for menu creation, dietary filtering, and order processing.

Dataset (File: food_items.txt):

```
Pizza,Veg
Chicken Wings,Non-Veg
Pasta,Veg
Fish Fry,Non-Veg
Noodles,Veg
```

`lamb, 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:

- return type :tuple {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:

- Return type: Dictionary { dict[str, list[str]] , 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

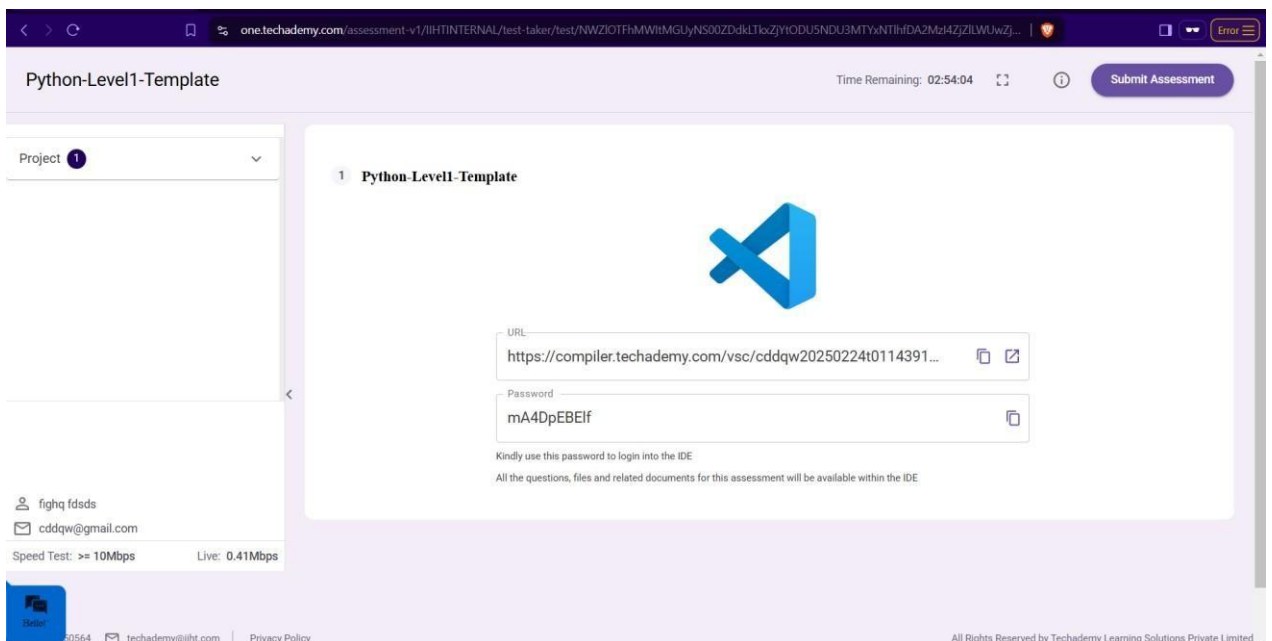
```
OK
coder@dighe20250227t070305rz1fj5p3:/home/myproject/dighegmailcom_20250227T070305$ python3 <<scriptname>>.py []
```

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