Python - Practice on Data Structures and Functions

November 2025

1 Network Packet Duplicate Detector

1.1 Problem Description

In computer networking, a ”packet” is a unit of data sent over a network. When debugging network issues, it’s common to analyze a stream of packets to look for anomalies. One common anomaly is a *duplicate packet*, which can be identified by its unique sequence number (an integer).

You are tasked with building a program for a network monitoring tool. Write a Python function called `find_duplicate_packets` that takes a **list** of packet sequence numbers (integers).

This function should process the list and return a **dictionary** where:

- The **keys** are the sequence numbers that appeared *more than once*.
- The **values** are the total counts of how many times that duplicate sequence number appeared.

If no duplicates are found, the function should return an empty dictionary.

1.2 Sample Input

```
# A list of packet sequence numbers captured from the network
packets = [1001, 1002, 1003, 1002, 1004, 1005, 1001, 1003, 1002]
```

1.3 Sample Output

(The function `find_duplicate_packets(packets)` should return this dictionary)

```
{
    1001: 2,
    1002: 3,
    1003: 2
}
```

2 Access Control List (ACL) Validator

2.1 Problem Description

In cybersecurity, an Access Control List (ACL) defines who is allowed to access a specific system. Security analysts often compare logs of users who *attempted* to login against the list of users who are actually *authorized*.

Write a function called `identify_intruders`. It accepts two arguments:

- `attempts`: A **list** of usernames that tried to log in.
- `authorized`: A **set** of usernames that are allowed access.

The function should return a **set** containing only the usernames from `attempts` that are **NOT** in the `authorized` set.

2.2 Sample Input

```
# List of all login attempts
login_attempts = ["alice", "bob", "eve", "alice", "mallory", "frank", "eve"]
# Set of authorized users
authorized_users = {"alice", "bob", "frank", "grace"}
```

2.3 Sample Output

(The function `identify_intruders(login_attempts, authorized_users)` should return this set)

```
{'eve', 'mallory'}
```

3 Log File First-Error Finder

3.1 Problem Description

In system administration, you often need to parse log files to find the *first occurrence* of a problem. Log entries are often stored or processed as **tuples** because they represent a fixed structure (e.g., timestamp, log level, message) that shouldn't change.

Write a Python function named `find_first_error` that takes a **list** of log entries. Each log entry is a **tuple** with three elements:

- `timestamp` (string)
- `log_level` (string, e.g., 'INFO', 'WARN', 'ERROR')
- `message` (string)

Your function should iterate through the list and return the `timestamp` (a string) of the **very first log entry** where the `log_level` is 'ERROR'. If no 'ERROR' entries are found in the list, the function should return `None`.

3.2 Sample Input

```
# A list of log entry tuples
log_data = [
    ('08:30:01', 'INFO', 'Server started'),
    ('08:30:05', 'WARN', 'Low disk space'),
    ('08:31:10', 'INFO', 'User login'),
    ('08:32:00', 'ERROR', 'Database connection failed'),
    ('08:32:01', 'INFO', 'Retrying connection...'),
    ('08:33:00', 'ERROR', 'Authentication service timeout')
]
```

3.3 Sample Output

(The function `find_first_error(log_data)` should return this string)

```
'08:32:00'
```

4 Screen Resolution Scaler

4.1 Problem Description

In Computer Graphics, a display resolution is often represented as a **tuple** of `(width, height)`. When a user changes their monitor settings or when an image needs to be resized, these dimensions must be scaled up or down.

Write a function called `scale_resolutions`. It accepts:

- A **list of tuples**, where each tuple represents a resolution (e.g., `(1920, 1080)`).
- A scaling factor (a float, e.g., `0.5` for half size).

The function should return a **new list of tuples** where each dimension is multiplied by the scaling factor and converted to an **integer**.

4.2 Sample Input

```
# A list of resolutions (width, height)
resolutions = [
    (1920, 1080),
    (1280, 720),
    (2560, 1440)
]
factor = 0.5
```

4.3 Sample Output

(The function `scale_resolutions(resolutions, factor)` should return)

```
[
    (960, 540),
    (640, 360),
    (1280, 720)
]
```

5 Unique Web Crawler Links

5.1 Problem Description

A web crawler is a program that browses the web, and a key part of its job is to keep track of all the *unique* pages it has visited. A **set** is the perfect data structure for this, as it automatically handles uniqueness.

Write a Python function called `update_visited_links` that simulates this process. The function should take two arguments:

- `visited_links_set`: A **set** containing all the URLs (strings) visited so far.
- `new_links_list`: A **list** of URLs (strings) found on the current page.

Your function should add all the new links to the `visited_links_set`. The function should then return a **tuple** containing two values:

- The *updated* `visited_links_set`.
- The *number* of *truly new* links that were added.

5.2 Sample Input

```
# The set of links already visited
visited = {'http://example.com', 'http://google.com', 'http://test.com'}

# The list of links found on the current page
new_links = ['http://google.com', 'http://python.org', 'http://example.com/
    about', 'http://test.com']
```

5.3 Sample Output

(The function `update_visited_links(visited, new_links)` should return this tuple)

```
(
    {'http://google.com', 'http://example.com', 'http://test.com', 'http://
        python.org', 'http://example.com/about'},
    2
)
```

(Note: The order of elements in the output set may vary)

6 Server Load Balancer

6.1 Problem Description

In distributed computing, a ”load balancer” distributes incoming tasks across a cluster of servers. We can represent the current load (number of active tasks) of a server cluster using a list of integers, where the index represents the Server ID.

Write a function called `distribute_tasks`. It accepts:

- `server_loads`: A **list** of integers (current load on each server).
- `new_tasks`: An integer (how many new tasks to assign).

The function should loop `new_tasks` times. In each iteration, it must find the server with the **minimum** current load and add **1** to it. Return the updated `server_loads` list.

6.2 Sample Input

```
# Current load on Server 0, Server 1, Server 2, Server 3
server_loads = [10, 5, 2, 8]
# We have 5 new tasks to distribute
new_tasks = 5
```

6.3 Sample Output

(The function `distribute_tasks(server_loads, new_tasks)` should return)

```
[10, 6, 6, 8]
```

7 Game Character Movement (List as Array)

7.1 Problem Description

In many simple games, a 1D game world can be represented by a **list** (acting as an **array**). Each index in the list is a position on the map.

Write a Python function `simulate_movement` that calculates a character's final position. The function takes two arguments:

- `board_size`: An integer representing the *length* of the game board (e.g., a `board_size` of 10 means valid indices are 0 through 9).
- `moves_list`: A **list** of integers representing the character's moves (positive is right, negative is left).

The character always starts at index `0`. The character **cannot fall off the board**. If a move would take the character below `0`, they stay at `0`. If a move would take them to `board_size` or higher, they stay at the last valid index (`board_size - 1`).

Return the character's *final index* (an integer).

7.2 Sample Input

```
# Board has 10 spots (indices 0-9)
size = 10
# List of moves to perform
moves = [3, 2, -1, 5, -8, 2]
```

7.3 Sample Output

(The function `simulate_movement(size, moves)` should return this integer)

```
3
```

8 File System Organizer

8.1 Problem Description

Operating systems often group files by their type (extension). You are writing a script to organize a messy folder.

Write a function called `group_by_extension`. It accepts a **list** of file names (strings). It should return a **dictionary** where:

- The **keys** are the file extensions (without the dot, e.g., ”txt”, ”py”).
- The **values** are **lists** containing the full names of the files with that extension.

(Assume every file has exactly one dot).

8.2 Sample Input

```
files = [
    "script.py",
    "notes.txt",
    "data.csv",
    "main.py",
    "image.png",
    "list.txt"
]
```

8.3 Sample Output

(The function `group_by_extension(files)` should return)

```
{
    "py": ["script.py", "main.py"],
    "txt": ["notes.txt", "list.txt"],
    "csv": ["data.csv"],
    "png": ["image.png"]
}
```

9 Student Gradebook Aggregator

9.1 Problem Description

You are building a system to process student grades. You have grade information stored in a **list** of **dictionaries**, but you need to aggregate this data into a single, clean **dictionary** that maps each *student's name* to a **list** of all their scores.

Write a Python function `build_gradebook` that takes one argument:

- `grade_entries`: A **list** where each item is a **dictionary** (with 'name' and 'score' keys).

Your function should process this list and return a **new dictionary** (the "gradebook").

- Each **key** in the gradebook should be a student's name (string).
- Each **value** should be a **list** of all scores (integers) that student has achieved.

9.2 Sample Input

```
# List of grade entries from various tests
entries = [
    {'name': 'Alice', 'score': 85},
    {'name': 'Bob', 'score': 90},
    {'name': 'Alice', 'score': 92},
    {'name': 'Charlie', 'score': 78},
    {'name': 'Bob', 'score': 88},
    {'name': 'Alice', 'score': 81}
]
```

9.3 Sample Output

(The function `build_gradebook(entries)` should return this dictionary)

```
{
    'Alice': [85, 92, 81],
    'Bob': [90, 88],
    'Charlie': [78]
}
```

(Note: The order of scores in the lists may vary)