Assignment 2: Cognitive Computing (UCS420)

1. List Operations

Question

Create a List `L` defined as `[10, 20, 30, 40, 50, 60, 70, 80]` and perform the following operations:
i. Add
`200` and `300` to `L`.

ii. Remove `10` and `30` from `L`.

iii. Sort `L` in ascending order.

iv. Sort `L` in descending order.

Pre-Code Explanation

- **Basics of Lists:**
  - Lists are mutable, allowing elements to be added, removed, or modified.
  - Operations like sorting and appending can be performed directly on lists.
- **Why Use Lists?**
  - Lists are dynamic and ideal for ordered collections of data.
- **Operations to Know:**
  - **Adding Elements:**
    - Use `append()` to add a single element.
    - Use `extend()` to add multiple elements from another list.
  - **Removing Elements:**
    - Use `remove(value)` to delete an element by its value.
    - If the value doesn't exist, `remove()` raises a `ValueError`.

    - **Sorting:**
        - Use `sort()` to modify the list in-place.
        - Use `sort(reverse=True)` for descending order.
- **Things to Keep in Mind:**
    - Ensure elements being added or removed exist or are valid for the operation.
    - The list must contain comparable data types for sorting.

---

2. Tuple Operations

Question

Create a tuple of marks scored as `scores = (45, 89.5, 76, 45.4, 89, 92, 58, 45)` and perform the following operations:

i. Identify the highest score and its index in the tuple.

ii. Find the lowest score and count how many times it appears.

iii. Reverse the tuple and return it as a list.

iv. Check if a specific score `76` (input by the user) is present in the tuple and print its first occurrence index, or a message saying it's not present.

Pre-Code Explanation

- **Basics of Tuples:**
    - Tuples are immutable sequences used to store fixed data.
    - They support operations like indexing, slicing, and functions such as `max()`, `min()`, and `count()`.
- **Why Use Tuples?**
    - Tuples are faster than lists for fixed data.
    - Their immutability ensures data integrity.
- **Operations to Know:**
    - Use `max()` and `min()` to find the highest and lowest values.

- Use `index(value)` to find the first occurrence of a value.
- Reverse tuples using slicing (`[::-1]`) and convert to a list using `list()`.
- Use `in` to check for the presence of an element.

- **Things to Keep in Mind:**
  - Ensure the tuple contains comparable data types for operations like `max()` or `min()`.
  - `index(value)` raises a `ValueError` if the value isn't found.

---

3. Random Numbers and Counting

Question

Write a program to create a list of 100 random numbers between 100 and 900. Count and print:
i. All odd numbers.

ii. All even numbers.

iii. All prime numbers.

Pre-Code Explanation

- **Basics of Random Numbers:**
  - Use the `random` module to generate pseudo-random numbers.
  - `randint(a, b)` generates a random integer between `a` and `b`, inclusive.
- **Number Characteristics:**
  - **Odd Numbers:** Numbers with a remainder of 1 when divided by 2 (`num % 2 == 1`).
  - **Even Numbers:** Numbers divisible by 2 without a remainder (`num % 2 == 0`).
  - **Prime Numbers:** Numbers greater than 1 that are divisible only by 1 and themselves.
- **Why Use Random Numbers?**

- They are essential in simulations, testing algorithms, and generating test data.

- **Things to Keep in Mind:**
  - Use list comprehensions for concise filtering.
  - For efficient prime checking, test divisors up to the square root of the number.
  - Avoid hardcoding the range or count of numbers to improve flexibility.

---

4. Set Operations

Question

Consider the following sets representing scores of two teams:

`A = {34, 56, 78, 90}` and `B = {78, 45, 90, 23}`. Perform the following operations:
i. Find the unique scores achieved by both teams (union).

ii. Identify the scores that are common to both teams (intersection).

iii. Find the scores that are exclusive to each team (symmetric difference).

iv. Check if the scores of team A are a subset of team B, and if team B's scores are a superset of team A.

v. Remove a specific score `X` (input by the user) from set A if it exists. If not, print a message saying it is not present.

Pre-Code Explanation

- **Basics of Sets:**
  - Sets are unordered collections of unique elements.
  - They are ideal for mathematical operations like union, intersection, and difference.
- **Why Use Sets?**
  - Sets allow efficient operations and ensure uniqueness of elements.
- **Operations to Know:**

- **Union (`|` or `union()`):** Combines all unique elements.
- **Intersection (`&` or `intersection()`):** Finds common elements.
- **Symmetric Difference (`^` or `symmetric_difference()`):** Finds elements unique to each set.
- **Subset and Superset Checks:**
  - `issubset()`: Checks if one set is a subset of another.
  - `issuperset()`: Checks if one set is a superset of another.
- **Element Removal:**
  - `remove(value)`: Removes an element; raises `KeyError` if it doesn't exist.
  - Use `discard(value)` to avoid errors when the value doesn't exist.

- **Things to Keep in Mind:**
  - Sets do not maintain order.
  - Ensure inputs for operations are sets or convert them before performing operations.

---

5. Dictionary Operations

Question

Write a program to rename a key `city` to `location` in the following dictionary:

`data = {"city": "New York", "population": 8419600, "area": 468.9}`.

Pre-Code Explanation

- **Basics of Dictionaries:**
  - Dictionaries store data in key-value pairs.
  - Keys must be unique and immutable.
- **Why Use Dictionaries?**
  - They allow fast data retrieval and are well-suited for structured data.

- **Operations to Know:**
  - Use `dict[key]` to access or modify values.
  - Add a new key-value pair by assigning to a new key.
  - Use `pop(key)` to remove a key-value pair.
- **Things to Keep in Mind:**
  - Renaming a key involves adding a new key with the same value and removing the old key.
  - Ensure the key being renamed exists to avoid a `KeyError`.