### **PW SKILLS : DATA SCIENCE WITH GENERATIVE AI**

### **Python: Data Structure Assignment (Assignment No 02)**

### **1. Discuss string slicing and provide examples**

**String slicing** is a method in Python used to extract a part of a string. It uses the syntax string[start:end:step].

* **start**: The starting index (inclusive, default is 0).
* **end**: The ending index (exclusive).
* **step**: The step size (default is 1).

#### **Example**

# Example of string slicing

string = "Hello, Python!"

# Slicing to get "Hello"

print(string[:5]) # Output: Hello

# Slicing to get "Python"

print(string[7:13]) # Output: Python

# Reverse the string

print(string[::-1]) # Output: !nohtyP ,olleH

# Skip characters while slicing

print(string[::2]) # Output: Hlo yhn

#### **Explanation**

1. string[:5] gets characters from index 0 to 4.
2. string[7:13] gets characters from index 7 to 12.
3. string[::-1] reverses the string.
4. string[::2] selects every second character.

### **2 Explain the key features of lists in Python**

Lists are **ordered, mutable** collections in Python used to store multiple items.

#### **Key Features**

1. **Ordered**: Elements maintain their order of insertion.
2. **Mutable**: Elements can be modified after creation.
3. **Heterogeneous**: Can contain different data types (e.g., integers, strings, other lists).
4. **Dynamic**: Can grow or shrink as needed.
5. **Indexed**: Elements can be accessed using indices.

#### **Example**

my\_list = [1, "Python", 3.14, [10, 20]]

# Accessing elements

print(my\_list[1]) # Output: Python

# Modifying an element

my\_list[2] = "Pi"

print(my\_list) # Output: [1, 'Python', 'Pi', [10, 20]]

# Adding an element

my\_list.append("New")

print(my\_list) # Output: [1, 'Python', 'Pi', [10, 20], 'New']

# Removing an element

my\_list.remove("Python")

print(my\_list) # Output: [1, 'Pi', [10, 20], 'New']

### **3. Describe how to access, modify, and delete elements in a list with examples**

#### **Accessing Elements**

lst = [10, 20, 30, 40]

print(lst[0]) # Access first element (Output: 10)

print(lst[-1]) # Access last element (Output: 40)

#### **Modifying Elements**

lst[2] = 300 # Modify the third element

print(lst) # Output: [10, 20, 300, 40]

#### **Deleting Elements**

# Using del

del lst[1] # Remove element at index 1

print(lst) # Output: [10, 300, 40]

# Using remove()

lst.remove(300) # Remove element by value

print(lst) # Output: [10, 40]

# Using pop()

popped = lst.pop(0) # Remove and return the first element

print(popped) # Output: 10

print(lst) # Output: [40]

### **4. Compare and contrast tuples and lists with examples**

| Feature | List | Tuple |
| --- | --- | --- |
| Mutability | Mutable | Immutable |
| Syntax | Square brackets [ ] | Parentheses ( ) |
| Performance | Slower(more overhead) | Faster |
| Use Cases | Dynamic Collections | Fixed/Static Collections |

#### **Example**

# List Example

my\_list = [1, 2, 3]

my\_list[1] = 20 # Mutable

print(my\_list) # Output: [1, 20, 3]

# Tuple Example

my\_tuple = (1, 2, 3)

# my\_tuple[1] = 20 # Error: Tuples are immutable

### **5. Describe the key features of sets and provide examples**

**Sets** are unordered, mutable collections of unique elements.

#### **Key Features**

1. **Unordered**: No specific order.
2. **Unique**: No duplicate elements.
3. **Mutable**: Can add or remove elements.
4. **Efficient**: Optimized for membership testing.

#### **Example**

# Creating a set

my\_set = {1, 2, 3, 3}

print(my\_set) # Output: {1, 2, 3} (duplicates removed)

# Adding an element

my\_set.add(4)

print(my\_set) # Output: {1, 2, 3, 4}

# Removing an element

my\_set.remove(2)

print(my\_set) # Output: {1, 3, 4}

# Membership testing

print(3 in my\_set) # Output: True

### **6. Discuss the use cases of tuples and sets in Python programming**

#### **Use Cases of Tuples**

1. Storing **immutable data** (e.g., coordinates (x, y)).
2. Acting as keys in dictionaries (since they are hashable).
3. Returning multiple values from a function.

#### **Use Cases of Sets**

1. Removing duplicates from a list.
2. Performing mathematical set operations (union, intersection, difference).
3. Efficient membership testing.

### **7. Describe how to add, modify, and delete items in a dictionary with examples**

#### **Adding Items**

# Adding new key-value pair

my\_dict = {"a": 1, "b": 2}

my\_dict["c"] = 3

print(my\_dict) # Output: {'a': 1, 'b': 2, 'c': 3}

#### **Modifying Items**

# Modifying value of an existing key

my\_dict["b"] = 20

print(my\_dict) # Output: {'a': 1, 'b': 20, 'c': 3}

#### **Deleting Items**

# Using del

del my\_dict["a"]

print(my\_dict) # Output: {'b': 20, 'c': 3}

# Using pop()

value = my\_dict.pop("b")

print(value) # Output: 20

print(my\_dict) # Output: {'c': 3}

### **8. Discuss the importance of dictionary keys being immutable and provide examples**

**Dictionary keys must be immutable because**:

1. Keys are hashed to allow fast lookups.
2. Mutable objects like lists can change their hash, leading to inconsistencies.
3. This ensures data integrity in a dictionary.

#### **Example**

# Valid dictionary

valid\_dict = {(1, 2): "Tuple as key"}

print(valid\_dict) # Output: {(1, 2): 'Tuple as key'}

# Invalid dictionary

# invalid\_dict = {[1, 2]: "List as key"} # Error: TypeError

**Explanation**: Tuples are immutable and hashable, but lists are mutable and cannot be used as keys.