1. String Slicing

String slicing allows you to extract a substring from a string by specifying a start and end index.

Syntax: string[start:end]

- start: The index where the slice begins (inclusive).
- end: The index where the slice ends (exclusive).

Examples:

python

```
text = "Hello, World!"
print(text[0:5])  # Output: Hello
print(text[7:])   # Output: World!
print(text[:5])   # Output: Hello
print(text[-6:])  # Output: World!
```

2. Key Features of Lists in Python

- Ordered: Lists maintain the order of elements.
- Mutable: You can change elements, add, or remove items.
- Dynamic Size: Lists can grow or shrink in size.
- Heterogeneous: Lists can hold elements of different types.

Examples:

python

```
my_list = [1, 2, 3, "Python", 4.5]
print(my_list)        # Output: [1, 2, 3, 'Python', 4.5]
```

3. Accessing, Modifying, and Deleting Elements in a List

Accessing:

python

```
my_list = [10, 20, 30]
print(my_list[1])    # Output: 20
```

Modifying:

python

```
my_list = [10, 20, 30]
my_list[1] = 25
print(my_list)       # Output: [10, 25, 30]
```

Deleting:

python

```
my_list = [10, 20, 30]
```

```
del my_list[1]
print(my_list)      # Output: [10, 30]
```

4. Comparing Tuples and Lists

Lists:

Mutable: Lists can be changed after creation.
Syntax: Defined with square brackets [].

Example:

python

```
my_list = [1, 2, 3]
```

Tuples:

Immutable: Tuples cannot be changed after creation.
Syntax: Defined with parentheses ().

Example:

python

```
my_tuple = (1, 2, 3)
```

Comparison:

python

```
my_list = [1, 2, 3]
my_tuple = (1, 2, 3)

print(type(my_list))  # Output: <class 'list'>
print(type(my_tuple)) # Output: <class 'tuple'>

# Lists can be changed
my_list[1] = 10
print(my_list)       # Output: [1, 10, 3]

# Tuples cannot be changed
# my_tuple[1] = 10    # Raises TypeError
```

5. Key Features of Sets

Unordered: Sets do not maintain the order of elements.
Mutable: Sets can be changed (elements can be added or removed).
Unique Elements: Sets do not allow duplicate elements.

Examples:

python

```
my_set = {1, 2, 3, 4}
```

```
print(my_set)        # Output: {1, 2, 3, 4}

my_set.add(5)
print(my_set)        # Output: {1, 2, 3, 4, 5}

my_set.remove(2)
print(my_set)        # Output: {1, 3, 4, 5}
```

6. Use Cases of Tuples and Sets

Tuples:

Used for fixed collections of items.
Often used as keys in dictionaries because they are immutable.

Example:

python

```
def get_coordinates():
    return (10, 20)

coordinates = get_coordinates()
```

Sets:

Useful for operations involving unique items or mathematical set operations (union, intersection).
Used to remove duplicates from a collection.

Example:

python

```
unique_items = set([1, 2, 2, 3, 4, 4, 5])
print(unique_items)  # Output: {1, 2, 3, 4, 5}
```

7. Adding, Modifying, and Deleting Items in a Dictionary

Adding:

python

```
my_dict = {’a’: 1, ’b’: 2}
my_dict[’c’] = 3
print(my_dict)  # Output: {’a’: 1, ’b’: 2, ’c’: 3}
```

Modifying:

python

```
my_dict = {’a’: 1, ’b’: 2}
my_dict[’a’] = 10
print(my_dict)  # Output: {’a’: 10, ’b’: 2}
```

Deleting:

python

```
my_dict = {’a’: 1, ’b’: 2}
del my_dict[’b’]
print(my_dict)  # Output: {’a’: 1}
```

8. Importance of Dictionary Keys Being Immutable

Dictionary keys must be immutable because the key’s hash value is used to determine where the key-value pair is stored in the dictionary. If keys were mutable, their hash values could change, leading to inconsistencies.

Examples of Immutable Keys:

    Strings: my_dict = {"name": "Alice"}
    Tuples: my_dict = {(1, 2): "Point A"}

Examples of Non-Immutable Keys (Not Allowed):

    Lists: my_dict = {["a", "b"]: 1} # Raises TypeError
    Sets: my_dict = {{1, 2}: 2} # Raises TypeError

This ensures that the dictionary can efficiently and correctly manage and retrieve the key-value pairs.