1. Discuss string slicing and provide examples.

- **String slicing** allows you to access a portion (substring) of a string by specifying a range of indices.
- **Syntax**: string[start:stop:step]
    - start: the index to start the slice (inclusive).
    - stop: the index to stop the slice (exclusive).
    - step: the step count between indices (optional).
- **Examples**:

python

Copy code

```
my_string = "Hello, World!"
print(my_string[0:5])    # Output: Hello
print(my_string[7:12])   # Output: World
print(my_string[::2])    # Output: Hlo ol!
print(my_string[::-1])   # Output: !dlroW ,olleH (reverses the string)
```

2. Explain the key features of lists in string.

The key features of lists in Python:

- **Lists** are ordered, mutable, and can hold items of different data types.
- **Key features**:
    - **Mutable**: Elements can be changed after the list is created.
    - **Ordered**: Elements have a defined order and can be accessed by index.
    - **Allows duplicates**: Lists can contain multiple occurrences of the same element.
    - **Dynamic size**: Lists can grow or shrink dynamically.
- **Examples**:

  python

  Copy code

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

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

- **Access**: Use index values to access elements in a list.

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

- **Modify**: Lists are mutable, so you can modify elements by assigning new values to specific indices.

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

- **Delete**: You can delete elements using del, pop(), or remove().

  ```
  python
  Copy code
  del my_list[1]    # Deletes the element at index 1
  my_list.pop()     # Removes the last element
  my_list.remove(10) # Removes the first occurrence of the element 10
  ```

4. Compare and contrast tuples and lists with examples:

- **Lists**:
  - Mutable (can be modified).
  - Slower than tuples.

- Syntax: [].

- **Tuples**:
  - Immutable (cannot be modified after creation).
  - Faster than lists.
  - Syntax: ().
- **Example**:

  python

  Copy code

  my_list = [1, 2, 3] # List

  my_tuple = (1, 2, 3) # Tuple

5. **Describe the key features of sets and provide examples of their use:**

- **Sets** are unordered collections of unique elements.
- **Key features**:
  - **Unordered**: No indexing; elements do not maintain order.
  - **Unique**: No duplicate elements.
  - **Mutable**: You can add or remove elements.
- **Examples**:

  python

  Copy code

  my_set = {1, 2, 3, 2} # Duplicates will be removed

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

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

- **Tuples** are used when:
  - Data should not be modified after creation (e.g., coordinates, fixed configurations).

    - Data needs to be accessed quickly (tuples are faster than lists).
- **Sets** are used when:
    - You need to store unique items (e.g., removing duplicates from a list).
    - You need to perform set operations (e.g., union, intersection).

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

- **Add**: Use assignment to add key-value pairs.

  python

  Copy code

  my_dict = {'a': 1, 'b': 2}

  my_dict['c'] = 3 # Adds a new key-value pair
- **Modify**: Assign a new value to an existing key.

  python

  Copy code

  my_dict['a'] = 10 # Modifies the value of 'a'
- **Delete**: Use del or pop() to remove key-value pairs.

  python

  Copy code

  del my_dict['b'] # Deletes the key 'b'

  my_dict.pop('a') # Removes the key 'a' and its value

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

- **Keys in dictionaries must be immutable** because they need to remain constant for efficient lookups (hashing). Immutable types like strings, numbers, and tuples can be used as keys, but mutable types like lists and sets cannot.
- **Example**:

  python

Copy code

```
my_dict = {1: 'one', 'a': 'apple'}  # Valid keys

# Invalid key:

# my_dict[[1, 2]] = 'invalid'  # This will raise a TypeError because lists are mutable
```