**1.List(Introduction,Operation,Function,Comprehension)**

**2.Tuple(Introduction,Operation,Function,Comprehension)**

**3.Set & Frozenset : Introduction, Operations,**

### **1 .Introduction to Lists:**

In Python, a list is a mutable and ordered sequence of elements. Lists are defined by enclosing the elements in square brackets ([]). Elements within a list can be of different data types, and you can perform various operations on lists.

### List Operations:

1. **Creation:**
   * Create a list by enclosing elements in square brackets.

my\_list = [1, 2, 3, 4, 5]

1. **Accessing Elements:**
   * Access elements using indexing.

print(my\_list[0]) # Output: 1

1. **Slicing:**
   * Extract a portion of the list using slicing.

sublist = my\_list[1:4]

print(sublist) # Output: [2, 3, 4]

1. **Concatenation (**+**):**
   * Concatenate two lists to create a new list.

list1 = [1, 2, 3]

list2 = [4, 5, 6]

combined\_list = list1 + list2

1. **Repetition (**\***):**
   * Repeat a list multiple times.

repeated\_list = my\_list \* 2

1. **Length (**len()**):**
   * Get the length of a list.

length = len(my\_list)

### List Functions:

1. append()**:**
   * Add an element to the end of the list.

my\_list.append(6)

1. insert()**:**
   * Insert an element at a specific position in the list.

my\_list.insert(2, 10) # Insert 10 at index 2

1. remove()**:**
   * Remove the first occurrence of a specified element from the list.

my\_list.remove(3)

1. pop()**:**
   * Remove and return the element at a specified position.

popped\_element = my\_list.pop(1) # Remove and return element at index 1

1. index()**:**
   * Return the index of the first occurrence of a specified element.

index = my\_list.index(4) # Returns the index of 4

### List Comprehension:

List comprehension is a concise way to create lists in Python.

squared\_numbers = [x\*\*2 for x in range(5)]

# Output: [0, 1, 4, 9, 16]

List comprehensions consist of an expression followed by a for clause. They can also include an optional if clause to filter elements.

even\_squares = [x\*\*2 for x in range(10) if x % 2 == 0]

# Output: [0, 4, 16, 36, 64]

### **2. Introduction to Tuples:**

In Python, a tuple is a collection data type that is similar to a list. Tuples are ordered, immutable, and can contain elements of different data types. They are defined using parentheses ().

### Tuple Operations:

1. **Creation:**
   * Create a tuple by enclosing elements in parentheses.

my\_tuple = (1, 2, 3, 'a', 'b')

1. **Accessing Elements:**
   * Access elements using indexing.

print(my\_tuple[0]) # Output: 1

1. **Slicing:**
   * Extract a portion of the tuple using slicing.

subtuple = my\_tuple[1:4]

print(subtuple) # Output: (2, 3, 'a')

1. **Concatenation (**+**):**
   * Concatenate two tuples to create a new tuple.

tuple1 = (1, 2, 3)

tuple2 = ('a', 'b', 'c')

combined\_tuple = tuple1 + tuple2

1. **Repetition (**\***):**
   * Repeat a tuple multiple times.

repeated\_tuple = my\_tuple \* 2

1. **Length (**len()**):**
   * Get the length of a tuple.

length = len(my\_tuple)

### **Immutability of Tuples:**

One key characteristic of tuples is their immutability. Once a tuple is created, you cannot change its elements or size. This immutability provides data integrity and ensures that the tuple remains unchanged throughout its existence.

### **3. Introduction to Sets:**

In Python, a set is an unordered and mutable collection of unique elements. Sets are defined by enclosing elements in curly braces {}. Sets do not allow duplicate elements and are useful for performing mathematical set operations.

### Set Operations:

1. **Creation:**
   * Create a set by enclosing elements in curly braces.

my\_set = {1, 2, 3, 4, 5}

1. **Adding Elements:**
   * Add an element to the set using the add() method.

my\_set.add(6)

1. **Removing Elements:**
   * Remove an element from the set using the remove() or discard() method.

my\_set.remove(3)

1. **Union (**|**):**
   * Combine two sets to create a new set with unique elements.

set1 = {1, 2, 3}

set2 = {3, 4, 5}

union\_set = set1 | set2

1. **Intersection (**&**):**
   * Create a new set with common elements between two sets.

intersection\_set = set1 & set2

1. **Difference (**-**):**
   * Create a new set with elements from the first set that are not in the second set.

difference\_set = set1 - set2

### Introduction to Frozensets:

A frozenset is similar to a set, but it is immutable. Once a frozenset is created, its elements cannot be changed or modified. Frozensets are defined by the frozenset() constructor.

### Frozenset Operations:

1. **Creation:**
   * Create a frozenset using the frozenset() constructor.

my\_frozenset = frozenset([1, 2, 3, 4, 5])

1. **Operations:**
   * Frozensets support operations such as union, intersection, difference, and symmetric difference, similar to sets.

frozenset1 = frozenset([1, 2, 3])

frozenset2 = frozenset([3, 4, 5])

union\_frozenset = frozenset1 | frozenset2

Frozensets are useful when you need an immutable set, such as when using sets as dictionary keys. They lack the mutability of sets but offer certain advantages in specific use cases.

### **4. Introduction to Dictionaries:**

In Python, a dictionary is an unordered and mutable collection of key-value pairs. Each key must be unique within a dictionary, and it is associated with a corresponding value. Dictionaries are defined using curly braces {} and consist of key-value pairs separated by colons :.

### Functions and Operations on Dictionaries:

1. **Creation:**
   * Create a dictionary by specifying key-value pairs inside curly braces.

my\_dict = {'name': 'John', 'age': 25, 'city': 'New York'}

1. **Accessing Values:**
   * Access the value associated with a key.

print(my\_dict['name']) # Output: John

1. **Adding or Modifying Entries:**
   * Add a new key-value pair or modify an existing one.

my\_dict['occupation'] = 'Engineer'

my\_dict['age'] = 26

1. **Removing Entries:**
   * Remove a key-value pair using the pop() method.

removed\_value = my\_dict.pop('city')

1. **Keys, Values, and Items:**
   * Retrieve lists of keys, values, or items (key-value pairs).

keys = my\_dict.keys()

values = my\_dict.values()

items = my\_dict.items()

1. **Checking if a Key Exists:**
   * Check if a key exists in the dictionary.

if 'age' in my\_dict:

print("Age exists in the dictionary.")

### Dictionary Comprehension:

Similar to list comprehensions, dictionary comprehensions provide a concise way to create dictionaries.

**squared\_numbers = {x: x\*\*2 for x in range(5)}**

##You can also use conditions in dictionary comprehensions.##

**even\_squares = {x: x\*\*2 for x in range(10) if x % 2 == 0}**

**Another update Method**

Concatenation in the context of dictionaries typically refers to combining two dictionaries into a single dictionary. In Python, you can concatenate dictionaries using the update() method or the {\*\*d1, \*\*d2} syntax.

### Using update() method:

dict1 = {'a': 1, 'b': 2}

dict2 = {'b': 3, 'c': 4}

dict1.update(dict2)

print(dict1)

### Using {\*\*d1, \*\*d2} syntax:

dict1 = {'a': 1, 'b': 2}

dict2 = {'b': 3, 'c': 4}

concatenated\_dict = {\*\*dict1, \*\*dict2}

print(concatenated\_dict)

### Dictionary Functions:

1. get()**:**
   * Get the value associated with a key. Returns None if the key is not found.

**Eg:**age = my\_dict.get('age')

1. update()**:**
   * Update the dictionary with key-value pairs from another dictionary.

**Eg:**additional\_info = {'gender': 'Male', 'education': 'Bachelor'}

my\_dict.update(additional\_info)

1. clear()**:**
   * Remove all items from the dictionary.

my\_dict.clear()

These are some of the key features, functions, and operations associated with dictionaries in Python. Dictionaries are widely used for storing and manipulating data in various applications.