**1. Lists**

Lists are data structures that can hold multiple values. They are enclosed in square brackets ([]), with elements separated by commas.

Creating Lists Example:

fruits = ["apple", "banana", "cherry"] # String list

numbers = [1,2,3,4,5] # Integer list

fruits = [1, "apple", 3.14, True] # mixed list

**Functions of Lists**

1. Append

The append() method is used to add an element at the end of the list.

fruits = ["apple", "banana", "cherry"]

fruits.append("orange") # Add "orange" to the end of the list

print(fruits) # ["apple", "banana", "cherry", "orange"]

2. Insert

The insert() method is used to add an element at a specific position in the list.

fruits = ["apple", "banana", "cherry"]

fruits.insert(1, "orange") # Insert "orange" at index 1

print(fruits) # ["apple", "orange", "banana", "cherry"]

The first parameter is the index where you want to insert the element.

The second parameter is the element you want to insert.

3. Pop

The pop() method removes and returns the element at a given index. If no index is provided, it removes and returns the last element.

fruits = ["apple", "banana", "cherry"]

popped\_item = fruits.pop() # Remove the last item

print(popped\_item) # "cherry"

print(fruits) # ["apple", "banana"]

You can also specify an index to remove a specific element:

popped\_item = fruits.pop(1) # Remove the element at index 1

print(popped\_item) # "banana"

print(fruits) # ["apple"]

4. Reverse

The reverse() method reverses the order of elements in a list.

fruits = ["apple", "banana", "cherry"]

fruits.reverse() # Reverse the order of the list

print(fruits) # ["cherry", "banana", "apple"]

This operation is performed in-place, meaning the list is modified directly.

5. Subarray (Slicing)

You can extract a subarray (slice) of a list using slicing syntax.

fruits = ["apple", "banana", "cherry", "orange", "grape"]

sub\_fruits = fruits[1:4] # Slice the list from index 1 to 3 (not including index 4)

print(sub\_fruits) # ["banana", "cherry", "orange"]

* The syntax for slicing is list[start:end]. The slice includes the element at the start index, but excludes the element at the end index.
* You can also use the following to extract elements:
  + fruits[:3] → Elements from the start to index 2 (first 3 items).
  + fruits[2:] → Elements from index 2 to the end of the list.

6. Convert List to String

To convert a list to a string, you can use the join() method. This is typically used when you have a list of strings.

fruits = ["apple", "banana", "cherry"]

fruits\_str = ", ".join(fruits) # Join the list elements into a single string with ", " separator

print(fruits\_str) # "apple, banana, cherry"

You can use any separator within the join() method (like space, comma, etc.).

7. Convert String to List

To convert a string to a list, you can use the split() method.

fruits\_str = "apple, banana, cherry"

fruits\_list = fruits\_str.split(", ") # Split the string by ", " separator

print(fruits\_list) # ["apple", "banana", "cherry"]

You can specify any separator in the split() method. By default, split() will split by spaces if no separator is provided.

**2. Tuples**

Tuples are similar to lists, but they are **immutable** (cannot be changed after creation). They are enclosed in **parentheses** () and elements are separated by commas.  
Creating Tuples Example:

fruits = ("apple", "banana", "cherry") # String tuple

numbers = (1, 2, 3, 4, 5) # Integer tuple

mixed = (1, "apple", 3.14, True) # Mixed type tuple

Key Tuple Operations:

• Accessing Elements: Use indices to access elements. Indexing starts at 0.

print(fruits[0]) # "apple"

• Length: Use the len() function to find the length of the tuple.

print(len(fruits)) # 3

• Concatenation: You can concatenate two tuples using the + operator.

new\_fruits = fruits + ("orange",) # ("apple", "banana", "cherry", "orange")

• Repetition: You can repeat a tuple using the \* operator.

repeated\_fruits = fruits \* 2 # ("apple", "banana", "cherry", "apple", "banana", "cherry")

**3. Dictionaries**

Dictionaries are unordered collections of **key-value pairs**. Keys must be unique and are used to access the corresponding value. They are enclosed in **curly braces** {} with keys and values separated by a colon :.  
Creating Dictionaries Example:

person = {"name": "John", "age": 30, "city": "New York"} # String keys and values

numbers = {1: "one", 2: "two", 3: "three"} # Integer keys and string values

Key Dictionary Operations:

• Accessing Values: Use keys to access corresponding values.

print(person["name"]) # "John"

• Adding or Modifying Elements: You can add or modify key-value pairs.

person["age"] = 31 # Update the value of "age"

person["country"] = "USA" # Add a new key-value pair

• Removing Elements: Use the del keyword or the pop() method to remove a key-value pair.

del person["city"] # Removes the "city" key-value pair

person.pop("age") # Removes and returns the value of "age"

• Length: Use the len() function to find the number of key-value pairs.

print(len(person)) # 3

• Keys, Values, and Items: You can get all keys, values, or key-value pairs as a list or iterator.

print(person.keys()) # dict\_keys(['name', 'age', 'country'])

print(person.values()) # dict\_values(['John', 31, 'USA'])

print(person.items()) # dict\_items([('name', 'John'), ('age', 31), ('country', 'USA')])

**Casting**

list(): Converts a string or another iterable to a list.

tuple(): Converts a list or another iterable to a tuple.

dict(): Converts a sequence of key-value pairs into a dictionary.