List	Set	Tuple	Dictionary	String
used to store a collection of items	represents an unordered collection of unique elements.	ordered and immutable collection of elements	used to store collections of data in a key-value pair format.	A string is a sequence of characters
Features: • **Ordered** • **Mutable** • **Heterogeneous** • **Dynamic** • **Iterable** • **Sliceable** • **Nesting** • **Indexing**	**Features:** • **Unordered** • **Mutable** • **Unique Elements** • **Dynamic** • **Iterable** • **Hashable Elements** • **No Duplicates**	**Features:** • **Ordered** • **Immutable** • **Heterogeneous** • **Dynamic** • **Unpackable**	**Features:** • **Key-Value Pairs** • **Unordered** • **Mutable** • **Dynamic Sizing** • **Fast Lookups** • **Keys Must Be Hashable** • **Heterogeneous Values** • **Keys Are Unique**	**Features:** • **Immutable** • **Sequences** • **Indexing** • **Slicing** • **Unicode Support**
Methods: • **append(x):** Adds an element x to the end of the list. • **extend(iterable):** Appends the elements from an iterable to the end of the list. • **insert(i, x):** Inserts an element x at a specified index i. • **remove(x):** Removes the first occurrence of element x from the list. • **pop([i]):** Removes and returns the element at index i (or the last element if i is not specified). • **index(x):** Returns the index of the first occurrence of element x. • **count(x):** Returns the number of times element x appears in the list. • **sort():** Sorts the elements of the list in ascending order. • **reverse():** Reverses the order of elements in the list. • **copy():** Returns a shallow copy of the list. • **clear():** Removes all elements from the list. • **len(list):** Returns the number of elements in the list. • **list1 + list2:** Concatenates two lists.	**Methods**: • **add(element):** Adds the specified element to the set. If the element is already in the set, it won't be duplicated. • **remove(element):** Removes the specified element from the set. If the element is not present, it raises a KeyError. • **discard(element):** Removes the specified element from the set if it is present. It doesn't raise an error if the element is not in the set. • **pop():** Removes and returns an arbitrary element from the set. This is often used when you don't care which element is removed. • **clear():** Removes all elements from the set, leaving an empty set. • **copy():** Returns a shallow copy of the set. This copy is a new set with the same elements but is independent of the original set. • **union(set2):** Returns a new set that is the union of the original set and set2. It contains all the unique elements from both sets. • **intersection(set2):** Returns a new set that is the intersection of the original set and set2. It contains all the elements that are common to both sets. • **difference(set2):** Returns a new set that contains the elements from the original set that are not in set2.	**Methods:** • **count(element):** Returns the number of times the specified element appears in the tuple. • **index(element):** Returns the index of the first occurrence of the specified element in the tuple. • **len(tuple):** Returns the number of elements in the tuple. • **tuple(iterable):** Creates a new tuple from an iterable (e.g., a list or another tuple). • **+(concatenation):** You can use the + operator to concatenate two tuples and create a new tuple. **Note:** Tuples support common operations such as indexing, slicing, iteration, and concatenation, similar to lists.	**Methods:** • **dict[key]:** Retrieve the value associated with a specific key. If the key is not found, it raises a KeyError. You can also use the **get()** method to retrieve a value with a default if the key doesn't exist. • **dict[key] = value:** Add a new key-value pair to the dictionary or update the value associated with an existing key. • **del dict[key]:** Delete the key-value pair with the specified key. • **key in dict:** Check if a key exists in the dictionary. This is often used to avoid KeyError exceptions. • **dict.keys():** Return a view of all the keys in the dictionary. • **dict.values():** Return a view of all the values in the dictionary. • **dict.items():** Return a view of all key-value pairs (tuples) in the dictionary. • **dict.clear():** Remove all key-value pairs from the dictionary. • ****dict.copy() or dict.copy():** Create a shallow copy of the dictionary. • **dict.fromkeys(keys, value=None):** Create a new dictionary with keys from an iterable and values set to a default or specified value. • **dict.setdefault(key, default=None):** Get the value for a	**Methods:** • **str.capitalize():** Returns a copy of the string with the first character capitalized and the rest in lowercase. • **str.upper():** Returns a copy of the string with all characters converted to uppercase. • **str.lower():** Returns a copy of the string with all characters converted to lowercase. • **str.title():** Returns a copy of the string with the first character of each word capitalized and the rest in lowercase. • **str.strip([chars]):** Returns a copy of the string with leading and trailing characters specified in removed. If chars is not provided, it removes leding and trailing whitespace. • **str.split([sep [, maxsplit]]):** Splits the string into a list of substrings based on the specified separator • **str.join(iterable):** Combines the elements of an iterable (e.g., a list) into a single string, using the string as the separator. • **str.startswith(prefix [, start [, end]]):** Checks if the string starts with the specified . • **str.endswith(suffix [, start [, end]]):** Checks if the string ends with the specified . **str.replace(old, new[, count]):**

- **list1 * n:** Repeats the list n times.
- **list.index(x, start, end):** Returns the index of the first occurrence of element x within the specified slice.
- **list.count(x):** Counts the number of occurrences of element x within the list.
- **list.sort(key=None, reverse=False):** Sorts the list with an optional sorting key and in reverse order if specified.
- **list.reverse():** Reverses the elements of the list in place.
- **list.copy():** Returns a shallow copy of the list (equivalent to list[:]).
- **list.clear():** Removes all elements from the list.

- **symmetric_difference(set2):** Returns a new set that contains the elements that are in either the original set or set2, but not in both.
- **issubset(set2):** Returns True if the original set is a subset of set2, meaning that all elements of the original set are also in set2.
- **issuperset(set2):** Returns True if the original set is a superset of set2, meaning that set2 is a subset of the original set.
- **isdisjoint(set2):** Returns True if the original set and set2 have no elements in common.
- **len(set):** Returns the number of elements in the set.
- **update(iterable):** Updates the set with elements from an iterable, such as another set or a list.
- **intersection_update(set2):** Updates the set to contain only the elements that are also in set2.
- **difference_update(set2):** Updates the set to remove elements that are in set2.
- **symmetric_difference_update(set2):** Updates the set to contain elements that are in either the set or set2, but not in both.

given key; if the key doesn't exist, add it with the default value.
- **dict.pop(key, default=None):** Remove the key-value pair for the given key. If the key doesn't exist, return the default value.
- **dict.popitem():** Remove and return an arbitrary key-value pair from the dictionary. This method is available in Python 3.6 and later.
- **dict.update(other_dict):** Merge the contents of another dictionary or an iterable of key-value pairs into the current dictionary.
- **dict.get(key, default=None):** Retrieve the value associated with a key, or return a default value if the key is not found.
- ****dict.items() and dict.values():** These methods return views of the dictionary's key-value pairs and values, respectively, which are iterable and can be used for various operations.

Replaces all occurrences of the substring with the substring.
- **str.find(sub [, start [, end]]):** Searches the string for the first occurrence of the substring and returns the index (or -1 if not found).
- **str.index(sub [, start [, end]]):** search for the first occurrence of a substring sub within the given string. If the substring is found, the method returns the index at which the substring starts. If the substring is not found, it raises a ValueError exception.
- ****'str.count :** Returns the number of non-overlapping occurrences of the substring in the string.
- **str.isalpha():** Checks if all characters in the string are alphabetic.
- **str.isdigit():** Checks if all characters in the string are digits.
- **str.islower():** Checks if all characters in the string are lowercase.
- **str.isupper():** Checks if all characters in the string are uppercase.
- **str.isalnum():** Checks if all characters in the string are alphanumeric.
- **str.isidentifier():** Checks if the string is a valid Python identifier.
- **str.isdecimal():** Checks if all characters in the string are decimal (base 10) digits.

Example :

```
# Creating a list
my_list = [1, 2, 3, 4, 5]

# Appending elements to a list
my_list.append(6)
# Appends element 6 to the end of the list

# Extending a list with another iterable
my_list.extend([7, 8, 9])
```

Example :

```
# Creating a set
my_set = {1, 2, 3, 4, 5}

# Adding elements to a set
my_set.add(6)
# Adds element 6 to the set

# Removing elements from a set
my_set.remove(3)
# Removes element 3 from the set
```

Example :

```
# Creating a tuple
my_tuple = (1, 2, 3, 'hello', 3.14)

# Accessing elements
first_element = my_tuple[0]
# Access the first element (1)
last_element = my_tuple[-1]
# Access the last element (3.14)

# Slicing
subset = my_tuple[1:4]
```

Example :

```
# Creating a dictionary
person = {
    'name': 'Alice',
    'age': 30,
    'city': 'New York'
}

# Accessing values by key
name = person['name']
 # Access the value associated with the 'name' key
```

Example:

```
# String Concatenation
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name
print("Full Name:", full_name)

# String Slicing
message = "Hello, World!"
sub_message = message[0:5]
print("Substring:", sub_message)
```

```
# Extends the list with elements from the iterable

# Inserting an element at a specific index
my_list.insert(2, 10)
# Inserts 10 at index 2

# Removing elements from a list
my_list.remove(3)
# Removes the first occurrence of 3 from the list

# Popping an element by index
popped_element = my_list.pop(4)
 # Removes and returns the element at index 4

# Clearing the list
my_list.clear()
# Removes all elements, resulting in an empty list

# Copying a list
copy_of_list = my_list.copy()
# Creates a shallow copy of the list

# Reversing a list
my_list = [5, 4, 3, 2, 1]
my_list.reverse()
# Reverses the order of elements in the list

# Sorting a list
my_list = [3, 1, 4, 2, 5]
my_list.sort()
 # Sorts the list in ascending order

# Sorting in reverse
my_list.sort(reverse=True)
# Sorts the list in descending order

# Sorting with a custom key function
my_list = ['apple', 'banana',
```

```
# Discarding elements (no error if element not present)
my_set.discard(7)
# If 7 is in the set, it's removed; otherwise, no error

# Pop an arbitrary element
popped_element = my_set.pop()
# Removes and returns an arbitrary element

# Clearing the set
my_set.clear()
# Removes all elements, resulting in an empty set

# Copying a set
copy_of_set = my_set.copy()
# Creates a shallow copy of the set

# Set Operations
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}

# Union of sets
union_set = set1.union(set2)
# Creates a new set containing all unique elements from both sets

# Intersection of sets
intersection_set = set1.intersection(set2)
 # Creates a new set containing elements common to both sets

# Difference between sets
difference_set = set1.difference(set2)  # Creates a new set containing elements in set1 but not in set2

# Symmetric difference between sets
symmetric_difference_set = set1.symmetric_difference(set2)
# Creates a new set containing elements in either set, but not both
```

```
# Creates a new tuple (2, 3, 'hello')

# Unpacking
a, b, c, d, e = my_tuple
 # Unpack the elements into variables

# Finding element count
count_of_3 = my_tuple.count(3)  # Count of the element '3' is 1

# Finding element index
index_of_hello = my_tuple.index('hello')
# Index of 'hello' is 3

# Concatenating tuples
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
combined_tuple = tuple1 + tuple2
# Creates a new tuple (1, 2, 3, 4, 5, 6)

# Length of a tuple
tuple_length = len(my_tuple)
 # Length of my_tuple is 5

# Creating a new tuple from an iterable
my_list = [7, 8, 9]
new_tuple = tuple(my_list)
# Creates a new tuple (7, 8, 9)
```

```
# Adding a new key-value pair
person['country'] = 'USA'

# Modifying an existing key-value pair
person['age'] = 31

# Deleting a key-value pair
del person['city']

# Checking if a key exists
if 'country' in person:
    print(f"Country: {person['country']}")
else:
    print("Country key does not exist")

# Retrieving keys, values, and items
keys = person.keys()
# Returns a view of keys
values = person.values()
# Returns a view of values
items = person.items()
# Returns a view of key-value pairs

# Creating a copy of the dictionary
person_copy = person.copy()

# Removing all key-value pairs
person.clear()

# Using setdefault to add a key-value pair if the key doesn't exist
city = person.setdefault('city', 'Unknown')  # Adds 'city' if it doesn't exist

# Removing a key-value pair with pop
removed_age = person.pop('age')
# Removes and returns the value for 'age'

# Iterating through the dictionary
for key, value in person.items():
    print(f"{key}: {value}")

# Merging dictionaries with update
additional_info = {'occupation':
```

```
# String Length
text = "Python Programming"
length = len(text)
print("Length:", length)

# String Methods
text = "  Python is awesome!  "
trimmed_text = text.strip()
print("Trimmed Text:", trimmed_text)

# String Splitting
sentence = "This is a sample sentence."
words = sentence.split()
print("Words:", words)

# String Case Conversion
text = "Hello, World!"
uppercase = text.upper()
lowercase = text.lower()
print("Uppercase:", uppercase)
print("Lowercase:", lowercase)

# String Searching
text = "Python is a powerful programming language. Python is versatile."
position = text.find("Python")
count = text.count("Python")
print("Position of 'Python':", position)
print("Count of 'Python':", count)

# String Replacement
text = "Python is easy to learn."
modified_text = text.replace("easy", "powerful")
print("Modified Text:", modified_text)

# String Formatting
name = "Alice"
age = 30
formatted_string = f"My name is {name} and I am {age} years old."
print("Formatted String:", formatted_string)
```

'cherry']
my_list.sort(key=len)
# Sorts the list based on string length 	
 	
# Counting occurrences 	
count_of_2 = my_list.count(2)
# Count of the element 2 in the list is 1 	
 	
# List Comprehension 	
squared_numbers = [x ** 2 for x in range(1, 6)]
# Generates a list of squared numbers 	
 	
# Slicing a list 	
subset = my_list[1:4]
# Creates a new list containing elements from index 1 to 3 	
 	
# Concatenating lists 	
list1 = [1, 2, 3]
list2 = [4, 5, 6]
combined_list = list1 + list2
# Creates a new list [1, 2, 3, 4, 5, 6] 	
 	
# Length of a list 	
list_length = len(my_list)
# Returns the number of elements in the list 	
	# Checking for subsets and supersets
	is_subset = set1.issubset(set2)
	# Checks if set1 is a subset of set2
	is_superset = set1.issuperset(set2)
	# Checks if set1 is a superset of set2
	
	# Checking for disjoint sets
	is_disjoint = set1.isdisjoint(set2)
	# Checks if sets have no common elements
	
	# Length of a set
	set_length = len(my_set)
	# Returns the number of elements in the set
			'Engineer', 'country': 'USA'}
			person.update(additional_info)
			
			# Nested dictionaries
			address = {
			'street': '123 Main St',
			'zip_code': '10001'
			}
			person['address'] = address
			
			# Accessing values in a nested dictionary
			street = person['address']['street']
			
			# Dictionary comprehension to create a new dictionary
			squared_numbers = {x: x ** 2 for x in range(1, 6)}
			
			# Length of the dictionary
			dictionary_length = len(person)
			# Returns the number of key-value pairs	

IMP Questions : **1. What is the difference between shallow copy and deep copy of a list?** **Ans :** **Shallow Copy:** A shallow copy creates a new list, but it does not create new copies of the objects within the original list. Instead, it copies references to the objects. If the objects within the original list are mutable (e.g., lists or dictionaries), changes made to those objects in the	**IMP Questions :** **1. What is the difference between remove() and discard() for sets?** **Ans :** **remove(element):** Removes the specified element from the set. If the element is not present, it raises a KeyError. **discard(element):** Removes the specified element from the set if it is present. It doesn't raise an error if the element is not in the set. **2. What are frozen sets in Python?** **Ans:**	**IMP Questions :** 1. **What is tuple packing and unpacking?** **Ans.** Tuple packing refers to the process of creating a tuple by placing multiple values (variables or expressions) separated by commas. The values are automatically packed into a tuple. Example : my_tuple = 1, 'Hello', 3.14 Tuple unpacking is the process of extracting individual values from a tuple and assigning them to variables. It allows you to access the elements	**IMP Questions :** **1. How do dictionaries handle duplicate keys?** **Ans :** dictionaries do not allow duplicate keys. If you attempt to insert a key into a dictionary that already exists, the dictionary will replace the existing key's associated value with the new value. Example: my_dict = {'name': 'Alice', 'age': 25} my_dict['name'] = 'Bob' # Overwrites the value associated with the 'name' key	**IMP Questions :** **1. Explain the difference between the `find()` and `index()` methods for searching in strings.** **Ans.** find() Searches the string for the first occurrence of the substring and returns the index (or -1 if not found). Index() If the substring is found, the method returns the index at which the substring starts. If the substring is not found, it raises a ValueError exception.

shallow copy will affect the original list as well.

Example:

```
import copy
original_list = [1, [2, 3], [4, 5]]
shallow_copy = copy.copy(original_list)
# Modifying a nested list within the shallow copy affects the original list
shallow_copy[1].append(6)
print(original_list)
# Output: [1, [2, 3, 6], [4, 5]]
```

Deep Copy:

A deep copy creates a new list and recursively copies all objects within the original list. This results in a completely independent copy with no shared references to objects. Changes made to objects in a deep copy will not affect the original list.

Example :

```
import copy
original_list = [1, [2, 3], [4, 5]]
deep_copy = copy.deepcopy(original_list)
# Modifying a nested list within the deep copy does not affect the original list
deep_copy[1].append(6)
print(original_list)
# Output: [1, [2, 3], [4, 5]]
```

2. What's the difference between remove() and pop()?

Ans.

remove() is used to remove the first occurrence of a specific value from a list.

pop() is used to remove an element from a list at a specific index.

a frozen set is an immutable version of the built-in set data type. Unlike regular sets, which are mutable and can be modified after creation, frozen sets are hashable and immutable, meaning their elements cannot be changed once the frozen set is created.

```
my_set = {1, 2, 3, 4}
my_frozen_set = frozenset(my_set)
```

3. How can you remove duplicate elements from a list using a set?

Ans.

```
original_list = [1, 2, 2, 3, 4, 4, 5]
unique_list = list(set(original_list))
print(unique_list)
 # Output: [1, 2, 3, 4, 5]
```

of a tuple conveniently

Example:.

```
my_tuple = (1, 'Hello', 3.14)
x, y, z = my_tuple
```

2. What are f-strings (formatted string literals) in Python ?

Ans.

Used to simplify string formatting and make it more concise and readable. F-strings provide a way to embed expressions inside string literals

Example:

```
name = "Alice"
age = 30
formatted_string = f"My name is {name} and I am {age} years old."
print(formatted_string)
```