Python Data Structures Cheat Sheet

List

Package/Method	Description	Code Example

append()	The `append()` method is used to add an element to the end of a list.	**Syntax:**
		```1 list_name.append(element)```
		Example:
		```1 fruits = ["apple", "banana", "orange"]```
		```2 fruits.append("mango") print(fruits)```

copy()	The `copy()` method is used to create a shallow copy of a list.	**Example 1:**
		```1 my_list = [1, 2, 3, 4, 5]```
		```2 new_list = my_list.copy() print(new_list)```
		```3 # Output: [1, 2, 3, 4, 5]```

count()	The `count()` method is used to count the number of occurrences of a specific element in a list in Python.	**Example:**
		```1 my_list = [1, 2, 2, 3, 4, 2, 5, 2]```
		```2 count = my_list.count(2) print(count)```
		```3 # Output: 4```

Creating a list	A list is a built-in data type that represents an ordered and mutable collection of elements. Lists are enclosed in square brackets [] and elements are separated by commas.	**Example:** ```1 fruits = ["apple", "banana", "orange", "mango"]```

del	The `del` statement is used to remove an element from list. `del` statement removes the element at the specified index.	**Example:**
		```1 my_list = [10, 20, 30, 40, 50]```
		```2 del my_list[2] # Removes the element at index 2 print(my_list)```
		```3 # Output: [10, 20, 40, 50]```

extend()	The `extend()` method is used to add multiple elements to a list. It takes an iterable (such as another list, tuple, or string) and appends each element of the iterable to the original list.	**Syntax:**
		```1 list_name.extend(iterable)```
		Example:
		```1 fruits = ["apple", "banana", "orange"]```
		```2 more_fruits = ["mango", "grape"]```
		```3 fruits.extend(more_fruits)```
		```4 print(fruits)```

Indexing	Indexing in a list allows you to access individual elements by their position. In Python, indexing starts from 0 for the first element and goes up to `length_of_list - 1`.	**Example:**
		```1 my_list = [10, 20, 30, 40, 50]```
		```2 print(my_list[0])```
		```3 # Output: 10 (accessing the first element)```
		```4 print(my_list[-1])```
		```5 # Output: 50 (accessing the last element using negative indexing)```

insert()	The `insert()` method is used to insert an element.	**Syntax:**
		```1 list_name.insert(index, element)```
		Example:
		```1 my_list = [1, 2, 3, 4, 5]```
		```2 my_list.insert(2, 6)```
		```3 print(my_list)```

Modifying a list	You can use indexing to modify or assign new values to specific elements in the list.	**Example:**
		```1 my_list = [10, 20, 30, 40, 50]```
		```2 my_list[1] = 25 # Modifying the second element```
		```3 print(my_list)```
		```4 # Output: [10, 25, 30, 40, 50]```

pop()	`pop()` method is another way to remove an element from a list in Python. It removes and returns the element at the specified index. If you don't provide an index to the `pop()` method, it will remove and return the last element of the list by default	**Example 1:**
		```1 my_list = [10, 20, 30, 40, 50]```
		```2 removed_element = my_list.pop(2) # Removes and returns the element at index 2```
		```3 print(removed_element)```
		```4 # Output: 30```
		```5```
		```6 print(my_list)```
		```7 # Output: [10, 20, 40, 50]```
		Example 2:
		```1 my_list = [10, 20, 30, 40, 50]```
		```2 removed_element = my_list.pop() # Removes and returns the last element```
		```3 print(removed_element)```
		```4 # Output: 50```
		```5```
		```6 print(my_list)```
		```7 # Output: [10, 20, 30, 40]```

remove()	To remove an element from a list. The `remove()` method removes the first occurrence of the specified value.	**Example:**
		```1 my_list = [10, 20, 30, 40, 50]```
		```2 my_list.remove(30) # Removes the element 30```
		```3 print(my_list)```
		```4 # Output: [10, 20, 40, 50]```

reverse()	The `reverse()` method is used to reverse the order of elements in a list	**Example 1:**
		```1 my_list = [1, 2, 3, 4, 5]```
		```2 my_list.reverse() print(my_list)```
		```3 # Output: [5, 4, 3, 2, 1]```

Slicing	You can use slicing to access a range of elements from a list.	**Syntax:**
		```1 list_name[start:end:step]```
		Example:
		```1 my_list = [1, 2, 3, 4, 5]```
		```2 print(my_list[1:4])```
		```3 # Output: [2, 3, 4] (elements from index 1 to 3)```
		```4```
		```5 print(my_list[:3])```
		```6 # Output: [1, 2, 3] (elements from the beginning up to index 2)```
		```7```
		```8 print(my_list[2:])```
		```9 # Output: [3, 4, 5] (elements from index 2 to the end)```
		```10```
		```11 print(my_list[::2])```
		```12 # Output: [1, 3, 5] (every second element)```

sort()	The `sort()` method is used to sort the elements of a list in ascending order. If you want to sort the list in descending order, you can pass the `reverse=True` argument to the `sort()` method.	**Example 1:**
		```1 my_list = [5, 2, 8, 1, 9]```
		```2 my_list.sort()```
		```3 print(my_list)```
		```4 # Output: [1, 2, 5, 8, 9]```
		Example 2:
		```1 my_list = [5, 2, 8, 1, 9]```
		```2 my_list.sort(reverse=True)```
		```3 print(my_list)```
		```4 # Output: [9, 8, 5, 2, 1]```

Dictionary

Package/Method	Description	Code Example

Accessing Values	You can access the values in a dictionary using their corresponding `keys`.	Syntax:
		```\nValue = dict_name["key_name"]\n```
		Example:
		```\nname = person["name"]\nage = person["age"]\n```

Add or modify	Inserts a new key-value pair into the dictionary. If the key already exists, the value will be updated; otherwise, a new entry is created.	Syntax:
		```\ndict_name[key] = value\n```
		Example:
		```\nperson["Country"] = "USA" # A new entry will be created.\nperson["city"] = "Chicago" # Update the existing value for the same\n```

clear()	The `clear()` method empties the dictionary, removing all key-value pairs within it. After this operation, the dictionary is still accessible and can be used further.	Syntax:
		```\ndict_name.clear()\n```
		Example:
		```\ngrades.clear()\n```

copy()	Creates a shallow copy of the dictionary. The new dictionary contains the same key-value pairs as the original, but they remain distinct objects in memory.	Syntax:
		```\nnew_dict = dict_name.copy()\n```
		Example:
		```\nnew_person = person.copy()\nnew_person = dict(person) # another way to create a copy of diction\n```

Creating a Dictionary	A dictionary is a built-in data type that represents a collection of key-value pairs. Dictionaries are enclosed in curly braces `{}`.	Example: ```\ndict_name = {} #Creates an empty dictionary\nperson = { "name": "John", "age": 30, "city": "New York"}\n```

del	Removes the specified key-value pair from the dictionary. Raises a `KeyError` if the key does not exist.	Syntax:
		```\ndel dict_name[key]\n```
		Example:
		```\ndel person["Country"]\n```

items()	Retrieves all key-value pairs as tuples and converts them into a list of tuples. Each tuple consists of a key and its corresponding value.	Syntax:
		```\nitems_list = list(dict_name.items())\n```
		Example:
		```\ninfo = list(person.items())\n```

key existence	You can check for the existence of a key in a dictionary using the `in` keyword	Example: ```\nif "name" in person:\n print("Name exists in the dictionary.")\n```

keys()	Retrieves all keys from the dictionary and converts them into a list. Useful for iterating or processing keys using list methods.	Syntax:
		```\nkeys_list = list(dict_name.keys())\n```
		Example:
		```\nperson_keys = list(person.keys())\n```

update()	The `update()` method merges the provided dictionary into the existing dictionary, adding or updating key-value pairs.	Syntax:
		```\ndict_name.update({key: value})\n```
		Example:
		```\nperson.update({"Profession": "Doctor"})\n```

values()	Extracts all values from the dictionary and converts them into a list. This list can be used for further processing or analysis.	Syntax:
		```\nvalues_list = list(dict_name.values())\n```
		Example:
		```\nperson_values = list(person.values())\n```

Sets

Package/Method	Description	Code Example

add()	Elements can be added to a set using the `add()` method. Duplicates are automatically removed, as sets only store unique values.	Syntax:
		```\nset_name.add(element)\n```
		Example:
		```\nfruits.add("mango")\n```

clear()	The `clear()` method removes all elements from the set, resulting in an empty set. It updates the set in-place.	Syntax:
		```\nset_name.clear()\n```
		Example:
		```\nfruits.clear()\n```

copy()	The `copy()` method creates a shallow copy of the set. Any modifications to the copy won't affect the original set.	Syntax:
		```\nnew_set = set_name.copy()\n```
		Example:
		```\nnew_fruits = fruits.copy()\n```

Defining Sets	A set is an unordered collection of unique elements. Sets are enclosed in curly braces `{}`. They are useful for storing distinct values and performing set operations.	Example: ```\nempty_set = set() #Creating an Empty Set\nfruits = {"apple", "banana", "orange"}\n```

discard()	Use the `discard()` method to remove a specific element from the set. Ignores if the element is not found.	Syntax:
		```\nset_name.discard(element)\n```
		Example:
		```\nfruits.discard("apple")\n```

issubset()	The `issubset()` method checks if the current set is a subset of another set. It returns True if all elements of the current set are present in the other set, otherwise False.	Syntax:
		```\nis_subset = set1.issubset(set2)\n```
		Example:
		```\nis_subset = fruits.issubset(colors)\n```

issuperset()	The `issuperset()` method checks if the current set is a superset of another set. It returns True if all elements of the other set are present in the current set, otherwise False.	Syntax:
		```\nis_superset = set1.issuperset(set2)\n```
		Example:
		```\nis_superset = colors.issuperset(fruits)\n```

pop()	The `pop()` method removes and returns an arbitrary element from the set. It raises a `KeyError` if the set is empty. Use this method to remove elements when the order doesn't matter.	Syntax:
		```\nremoved_element = set_name.pop()\n```
		Example:
		```\nremoved_fruit = fruits.pop()\n```

remove()	Use the `remove()` method to remove a specific element from the set. Raises a `KeyError` if the element is not found.	Syntax:
		```\n1 set_name.remove(element)\n```
		Example:
		```\n1 fruits.remove("banana")\n```

Set Operations	Perform various operations on sets: `union`, `intersection`, `difference`, `symmetric difference`.	Syntax:
		```\n1 union_set = set1.union(set2)\n2 intersection_set = set1.intersection(set2)\n3 difference_set = set1.difference(set2)\n4 sym_diff_set = set1.symmetric_difference(s...\n```
		Example:
		```\n1 combined = fruits.union(colors)\n2 common = fruits.intersection(colors)\n3 unique_to_fruits = fruits.difference(colors)\n4 sym_diff = fruits.symmetric_difference(col...\n```

update()	The `update()` method adds elements from another iterable into the set. It maintains the uniqueness of elements.	Syntax:
		```\n1 set_name.update(iterable)\n```
		Example:
		```\n1 fruits.update(["kiwi", "grape"]\n```

Skills Network