Skills Network

### 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:<br>`list_name.append(element)`<br>Copied!<br>Example:<br>`fruits = ["apple", "banana", "orange"]`<br>`fruits.append("mango") print(fruits)`<br>Copied! |
| copy() | The `copy()` method is used to create a shallow copy of a list. | Example 1:<br>`my_list = [1, 2, 3, 4, 5]`<br>`new_list = my_list.copy() print(new_list)`<br>`# Output: [1, 2, 3, 4, 5]`<br>Copied! |
| count() | The `count()` method is used to count the number of occurrences of a specific element in a list in Python. | Example:<br>`my_list = [1, 2, 2, 3, 4, 2, 5, 2]`<br>`count = my_list.count(2) print(count)`<br>`# Output: 4`<br>Copied! |
| 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:<br>`fruits = ["apple", "banana", "orange", "mango"]`<br>Copied! |
| del | The `del` statement is used to remove an element from list. `del` statement removes the element at the specified index. | Example:<br>`my_list = [10, 20, 30, 40, 50]`<br>`del my_list[2] # Removes the element at index 2 print(my_list)`<br>`# Output: [10, 20, 40, 50]`<br>Copied! |
| 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:<br>`list_name.extend(iterable)`<br>Copied!<br>Example:<br>`fruits = ["apple", "banana", "orange"]`<br>`more_fruits = ["mango", "grape"]`<br>`fruits.extend(more_fruits)`<br>`print(fruits)`<br>Copied! |
| 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:<br>`my_list = [10, 20, 30, 40, 50]`<br>`print(my_list[0])`<br>`# Output: 10 (accessing the first element)`<br>`print(my_list[-1])`<br>`# Output: 50 (accessing the last element using negative indexing)`<br>Copied! |
| insert() | The `insert()` method is used to insert an element. | Syntax:<br>`list_name.insert(index, element)`<br>Copied!<br>Example:<br>`my_list = [1, 2, 3, 4, 5]`<br>`my_list.insert(2, 6)`<br>`print(my_list)`<br>Copied! |
| Modifying a list | You can use indexing to modify or assign new values to specific elements in the list. | Example:<br>`my_list = [10, 20, 30, 40, 50]`<br>`my_list[1] = 25 # Modifying the second element`<br>`print(my_list)`<br>`# Output: [10, 25, 30, 40, 50]`<br>Copied! |
| 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:<br>`my_list = [10, 20, 30, 40, 50]`<br>`removed_element = my_list.pop(2) # Removes and returns the element at index 2`<br>`print(removed_element)`<br>`# Output: 30`<br><br>`print(my_list)`<br>`# Output: [10, 20, 40, 50]`<br>Copied!<br>Example 2:<br>`my_list = [10, 20, 30, 40, 50]`<br>`removed_element = my_list.pop() # Removes and returns the last element`<br>`print(removed_element)`<br>`# Output: 50`<br><br>`print(my_list)`<br>`# Output: [10, 20, 30, 40]`<br>Copied! |
| remove() | To remove an element from a list. The `remove()` method removes the first occurrence of the specified value. | Example:<br>`my_list = [10, 20, 30, 40, 50]`<br>`my_list.remove(30) # Removes the element 30`<br>`print(my_list)`<br>`# Output: [10, 20, 40, 50]`<br>Copied! |
| reverse() | The `reverse()` method is used to reverse the order of elements in a list | Example 1:<br>`my_list = [1, 2, 3, 4, 5]`<br>`my_list.reverse() print(my_list)`<br>`# Output: [5, 4, 3, 2, 1]`<br>Copied! |
| Slicing | You can use slicing to access a range of elements from a list. | Syntax:<br>`list_name[start:end:step]`<br>Copied!<br>Example:<br>`my_list = [1, 2, 3, 4, 5]`<br>`print(my_list[1:4])`<br>`# Output: [2, 3, 4] (elements from index 1 to 3)`<br><br>`print(my_list[:3])`<br>`# Output: [1, 2, 3] (elements from the beginning up to index 2)`<br><br>`print(my_list[2:])`<br>`# Output: [3, 4, 5] (elements from index 2 to the end)`<br><br>`print(my_list[::2])`<br>`# Output: [1, 3, 5] (every second element)`<br>Copied! |
| 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:<br>`my_list = [5, 2, 8, 1, 9]`<br>`my_list.sort()`<br>`print(my_list)`<br>`# Output: [1, 2, 5, 8, 9]`<br>Copied!<br>Example 2:<br>`my_list = [5, 2, 8, 1, 9]`<br>`my_list.sort(reverse=True)`<br>`print(my_list)`<br>`# Output: [9, 8, 5, 2, 1]`<br>Copied! |

## Dictionary

| Package/Method | Description | Code Example |
|---|---|---|
| Accessing Values | You can access the values in a dictionary using their corresponding `keys`. | Syntax:<br>`Value = dict_name["key_name"]`<br>Copied!<br>Example:<br>`name = person["name"]`<br>`age = person["age"]`<br>Copied! |
| 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:<br>`dict_name[key] = value`<br>Copied!<br>Example:<br>`person["Country"] = "USA" # A new entry will be created.`<br>`person["city"] = "Chicago" # Update the existing value for the same key`<br>Copied! |
| 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:<br>`dict_name.clear()`<br>Copied!<br>Example:<br>`grades.clear()`<br>Copied! |
| 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:<br>`new_dict = dict_name.copy()` |

| | | |
|---|---|---|
| | | Copied!<br>Example:<br>`new_person = person.copy()`<br>`new_person = dict(person) # another way to create a copy of dictionary`<br>Copied! |
| 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:<br>`dict_name = {} #Creates an empty dictionary`<br>`person = { "name": "John", "age": 30, "city": "New York"}`<br>Copied! |
| del | Removes the specified key-value pair from the dictionary. Raises a `KeyError` if the key does not exist. | Syntax:<br>`del dict_name[key]`<br>Copied!<br>Example:<br>`del person["Country"]`<br>Copied! |
| 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:<br>`items_list = list(dict_name.items())`<br>Copied!<br>Example:<br>`info = list(person.items())`<br>Copied! |
| key existence | You can check for the existence of a key in a dictionary using the `in` keyword | Example:<br>`if "name" in person:`<br>`    print("Name exists in the dictionary.")`<br>Copied! |
| keys() | Retrieves all keys from the dictionary and converts them into a list. Useful for iterating or processing keys using list methods. | Syntax:<br>`keys_list = list(dict_name.keys())`<br>Copied!<br>Example:<br>`person_keys = list(person.keys())`<br>Copied! |
| update() | The `update()` method merges the provided dictionary into the existing dictionary, adding or updating key-value pairs. | Syntax:<br>`dict_name.update({key: value})`<br>Copied!<br>Example:<br>`person.update({"Profession": "Doctor"})`<br>Copied! |
| values() | Extracts all values from the dictionary and converts them into a list. This list can be used for further processing or analysis. | Syntax:<br>`values_list = list(dict_name.values())`<br>Copied!<br>Example:<br>`person_values = list(person.values())`<br>Copied! |

### 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:<br>`set_name.add(element)`<br>Copied!<br>Example:<br>`fruits.add("mango")`<br>Copied! |
| clear() | The `clear()` method removes all elements from the set, resulting in an empty set. It updates the set in-place. | Syntax:<br>`set_name.clear()`<br>Copied!<br>Example:<br>`fruits.clear()`<br>Copied! |
| copy() | The `copy()` method creates a shallow copy of the set. Any modifications to the copy won't affect the original set. | Syntax:<br>`new_set = set_name.copy()`<br>Copied!<br>Example:<br>`new_fruits = fruits.copy()`<br>Copied! |
| 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:<br>`empty_set = set() #Creating an Empty Set`<br>`fruits = {"apple", "banana", "orange"}`<br>Copied! |
| discard() | Use the `discard()` method to remove a specific element from the set. Ignores if the element is not found. | Syntax:<br>`set_name.discard(element)`<br>Copied!<br>Example:<br>`fruits.discard("apple")`<br>Copied! |
| 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:<br>`is_subset = set1.issubset(set2)`<br>Copied!<br>Example:<br>`is_subset = fruits.issubset(colors)`<br>Copied! |
| 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:<br>`is_superset = set1.issuperset(set2)`<br>Copied!<br>Example:<br>`is_superset = colors.issuperset(fruits)`<br>Copied! |
| 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:<br>`removed_element = set_name.pop()`<br>Copied!<br>Example:<br>`removed_fruit = fruits.pop()`<br>Copied! |
| remove() | Use the `remove()` method to remove a specific element from the set. Raises a `KeyError` if the element is not found. | Syntax:<br>`set_name.remove(element)`<br>Copied!<br>Example:<br>`fruits.remove("banana")`<br>Copied! |
| Set Operations | Perform various operations on sets: `union`, `intersection`, `difference`, `symmetric difference`. | Syntax:<br>`union_set = set1.union(set2)`<br>`intersection_set = set1.intersection(set2)`<br>`difference_set = set1.difference(set2)`<br>`sym_diff_set = set1.symmetric_difference(set2)`<br>Copied!<br>Example:<br>`combined = fruits.union(colors)`<br>`common = fruits.intersection(colors)`<br>`unique_to_fruits = fruits.difference(colors)`<br>`sym_diff = fruits.symmetric_difference(colors)`<br>Copied! |
| update() | The `update()` method adds elements from another iterable into the set. It maintains the uniqueness of elements. | Syntax:<br>`set_name.update(iterable)`<br>Copied!<br>Example:<br>`fruits.update(["kiwi", "grape"]`<br>Copied! |

Skills Network