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:
		`list_name.append(element)`
		Example:
		`fruits = ["apple", "banana", "orange"]`
		`fruits.append("mango") print(fruits)`

copy()	The `copy()` method is used to create a shallow copy of a list.	Example 1:
		`my_list = [1, 2, 3, 4, 5]`
		`new_list = my_list.copy() print(new_list)`
		`# 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:
		`my_list = [1, 2, 2, 3, 4, 2, 5, 2]`
		`count = my_list.count(2) print(count)`
		`# 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: `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:
		`my_list = [10, 20, 30, 40, 50]`
		`del my_list[2] # Removes the element at index 2 print(my_list)`
		`# 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:
		`list_name.extend(iterable)`
		Example:
		`fruits = ["apple", "banana", "orange"]`
		`more_fruits = ["mango", "grape"]`
		`fruits.extend(more_fruits)`
		`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:
		`my_list = [10, 20, 30, 40, 50]`
		`print(my_list[0])`
		`# Output: 10 (accessing the first element)`
		`print(my_list[-1])`
		`# Output: 50 (accessing the last element using negative indexing)`

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

Modifying a list	You can use indexing to modify or assign new values to specific elements in the list.	Example:
		`my_list = [10, 20, 30, 40, 50]`
		`my_list[1] = 25 # Modifying the second element`
		`print(my_list)`
		`# 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:
		`my_list = [10, 20, 30, 40, 50]`
		`removed_element = my_list.pop(2) # Removes and returns the element at index 2`
		`print(removed_element)`
		`# Output: 30`
		`print(my_list)`
		`# Output: [10, 20, 40, 50]`
		Example 2:
		`my_list = [10, 20, 30, 40, 50]`
		`removed_element = my_list.pop() # Removes and returns the last element`
		`print(removed_element)`
		`# Output: 50`
		`print(my_list)`
		`# 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:
		`my_list = [10, 20, 30, 40, 50]`
		`my_list.remove(30) # Removes the element 30`
		`print(my_list)`
		`# Output: [10, 20, 40, 50]`

		Example 1:

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

Slicing	You can use slicing to access a range of elements from a list.	Syntax:
		`list_name[start:end:step]`
		Example:
		`my_list = [1, 2, 3, 4, 5]`
		`print(my_list[1:4])`
		`# Output: [2, 3, 4] (elements from index 1 to 3)`
		
		`print(my_list[:3])`
		`# Output: [1, 2, 3] (elements from the beginning up to index 2)`
		
		`print(my_list[2:])`
		`# Output: [3, 4, 5] (elements from index 2 to the end)`
		
		`print(my_list[::2])`
		`# 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:
		`my_list = [5, 2, 8, 1, 9]`
		`my_list.sort()`
		`print(my_list)`
		`# Output: [1, 2, 5, 8, 9]`
		Example 2:
		`my_list = [5, 2, 8, 1, 9]`
		`my_list.sort(reverse=True)`
		`print(my_list)`
		`# Output: [9, 8, 5, 2, 1]`

Tuple

Package/Method	Description	Code Example

count()	The count() method for a tuple is used to count how many times a specified element appears in the tuple.	Syntax:
		`tuple.count(value)`
		Example:
		`fruits = ("apple", "banana", "apple", "orange")`
		`print(fruits.count("apple")) #Counts the number of times apple is found in tuple.`
		`#Output: 2`

index()	The index() method in a tuple is used to find the first occurrence of a specified value and returns its position (index). If the value is not found, it raises a ValueError.	Syntax:
		`tuple.index(value)`
		Example:
		`fruits = ("apple", "banana", "orange")`
		`print(fruits[1]) #Returns the value at which apple is present.`
		`#Output: banana`

sum()	The sum() function in Python can be used to calculate the sum of all elements in a tuple, provided that the elements are numeric (integers or floats).	Syntax:
		`sum(tuple)`
		Example:
		`numbers = (10, 20, 5, 30)`
		`print(sum(numbers))`
		`#Output: 65`

min() and max()	Find the smallest (min()) or largest (max()) element in a tuple.	Example:
		`numbers = (10, 20, 5, 30)`
		`print(min(numbers))`
		`#Output: 5`
		`print(max(numbers))`
		`#Output: 30`

len()	Get the number of elements in the tuple using len().	Syntax:
		`len(tuple)`
		Example:
		`fruits = ("apple", "banana", "orange")`
		`print(len(fruits)) #Returns length of the tuple.`
		`#Output: 3`