Module 2 Cheatsheet: Python Data Structures Part-1

List

Package/Method	Description	Code Example
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"]` Copied!

append()	The `append()` method is used to add an element to the end of a list.	Syntax:
		1
		`list_name.append(element)`
		Copied!
		Example:
		1
		2
		3
		`fruits = ["apple", "banana", "orange"]`
		`fruits.append("mango")`
		`print(fruits)`
		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:
		1
		`list_name.extend(iterable)`
		Copied!
		Example:
		1
		2
		3
		4
		`fruits = ["apple", "banana", "orange"]`
		`more_fruits = ["mango", "grape"]`
		`fruits.extend(more_fruits)`
		`print(fruits)`
		Copied!

insert()	The `insert()` method is used to insert an element.	Syntax: 1 `list_name.insert(index, element)` Copied! Example: 1

```
2. 2
3. 3

1. my_list = [1, 2, 3, 4, 5]
2. my_list.insert(2, 6)
3. print(my_list)
```

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:
		`1. 1`
		`2. 2`
		`3. 3`
		`1. my_list = [10, 20, 30, 40, 50]`
		`2. print(my_list[0])  # Output: 10 (accessing the first element)`
		`3. print(my_list[-1]) # Output: 50 (accessing the last element using negative indexing)`
		Copied!
		Syntax:
		`1. 1`
		`1. list_name[start:end:step]`
		Copied!

Slicing	You can use slicing to access a range of elements from a list.	Example:
		`1. 1`
		`2. 2`
		`3. 3`
		`4. 4`
		`5. 5`
		`1. my_list = [1, 2, 3, 4, 5]`
		`2. print(my_list[1:4])  # Output: [2, 3, 4] (elements from index 1 to 3)`
		`3. print(my_list[:3])  # Output: [1, 2, 3] (elements from the beginning up to index 2)`
		`4. print(my_list[2:])  # Output: [3, 4, 5] (elements from index 2 to the end)`
		`5. print(my_list[::2])  # Output: [1, 3, 5] (every second element)`
		Copied!

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

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

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

pop()	`pop()` method is another way to remove an	Example 1: `1. 1`

	element from a list in Python. It removes and returns the element at the specified index.	2. 2 3. 3 4. 4 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) # Output: 30 4. print(my_list) # Output: [10, 20, 40, 50] Copied!
	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 2: 1. 1 2. 2 3. 3 4. 4 1. my_list = [10, 20, 30, 40, 50] 2. removed_element = my_list.pop() # Removes and returns the last element 3. print(removed_element) # Output: 50 4. print(my_list) # Output: [10, 20, 30, 40] Copied!
count()	The `count()` method is used to count the number of occurrences of a specific element in a list in Python.	Example: 1. 1 2. 2 3. 3 1. my_list = [1, 2, 2, 3, 4, 2, 5, 2] 2. count = my_list.count(2) 3. print(count) # Output: 4 Copied!
sort()	The `sort()` method is used to sort the elements of a list in ascending order.	Example 1: 1. 1 2. 2 3. 3 1. my_list = [5, 2, 8, 1, 9] 2. my_list.sort() 3. print(my_list) # Output: [1, 2, 5, 8, 9] Copied!
	If you want to sort the list in descending order, you can pass the `reverse=True` argument to the `sort()` method.	Example 2: 1. 1 2. 2 3. 3 1. my_list = [5, 2, 8, 1, 9] 2. my_list.sort(reverse=True) 3. print(my_list) # Output: [9, 8, 5, 2, 1] Copied!
reverse()	The `reverse()` method is used to reverse the order of elements in a list	Example 1: 1. 1 2. 2 3. 3 1. my_list = [1, 2, 3, 4, 5] 2. my_list.reverse() 3. print(my_list) # Output: [5, 4, 3, 2, 1] Copied!
copy()	The `copy()` method is used to create a shallow copy of a list.	Example 1: 1. 1 2. 2 3. 3 1. my_list = [1, 2, 3, 4, 5] 2. new_list = my_list.copy() 3. print(new_list) # Output: [1, 2, 3, 4, 5] Copied!

Tuples

Defining Tuples	A tuple is an immutable sequence of elements enclosed in parentheses `()` or without any enclosing brackets. Tuples are similar to lists, but unlike lists, they cannot be modified once created.	Example: 1 `my_tuple = (1, 2, 3, "four", 5.0)` Copied!

Indexing	Access individual elements of a tuple using indexing.	Example:
		1
		2
		3
		`my_tuple = (1, 2, 3, "four", 5.0)`
		`print(my_tuple[0])  # Output: 1`
		`print(my_tuple[3])  # Output: "four"`
		Copied!

index()	`index()` method to find the index of a specific value within a tuple. The `index()` method returns the first occurrence of the value in the tuple.	Example:
		1
		2
		3
		4
		5
		`my_tuple = ("apple", "banana", "orange", "banana", "grape")`
		
		`# Find the index of "banana"`
		`index = my_tuple.index("banana")`
		`print(index)  # Output: 1`
		Copied!

Slicing	Tuple slicing in Python allows you to extract a portion of a tuple by specifying a range of indices.	Syntax:
		1
		`tuple[start:end:step]`
		Copied!
		Example:
		1
		2
		3
		4
		5
		6
		7
		8
		9
		10
		11
		12
		13
		14
		15
		16
		`my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`
		
		`sliced_tuple = my_tuple[2:5] # get elements from index 2 to index 5 (exclusive)`
		`print(sliced_tuple)  # Output: (3, 4, 5)`
		
		`sliced_tuple = my_tuple[0:7:2]   # get elements from index 0 to index 7 (exclusive) with a step of 2`
		`print(sliced_tuple)  # Output: (1, 3, 5, 7)`
		
		`sliced_tuple = my_tuple[5:]   # get elements from index 5 to the end`
		`print(sliced_tuple)  # Output: (6, 7, 8, 9, 10)`
		
		`sliced_tuple = my_tuple[:3]  # get elements from the beginning to index 3`
		`print(sliced_tuple)  # Output: (1, 2, 3)`

```
15. sliced_tuple = my_tuple[-3:]   # get the last 3 elements
16. print(sliced_tuple)  # Output: (8, 9, 10)
```

Copied!

count()	The `count()` method is used to count the number of occurrences of a specified value in a tuple. The `count()` method returns an integer representing the count of occurrences.	Example:
		`1. 1`
		`2. 2`
		`3. 3`
		
		`1. my_tuple = (1, 2, 2, 3, 4, 2, 5)`
		`2. count = my_tuple.count(2)`
		`3. print(count)  # Output: 3`
		
		Copied!

Author(s)

Pooja Patel

Other Contributor(s)

Malika Singla

Changelog

Date	Version	Changed by	Change Description
2023-17-10	0.2	Malika	Updated cheatsheet
2023-17-10	0.1	Pooja Patel	Initial version created