

Python Data Structures Cheat Sheet

List

Package/Method	Description	Code Example
		Syntax: <pre>1. 1 1. list_name.append(element)</pre> <div>Copied!</div>
append()	The `append()` method is used to add an element to the end of a list.	Example: <pre>1. 1 2. 2 1. fruits = ["apple", "banana", "orange"] 2. fruits.append("mango") print(fruits)</pre> <div>Copied!</div>
copy()	The `copy()` method is used to create a shallow copy of a list.	Example 1: <pre>1. 1 2. 2 3. 3 1. my_list = [1, 2, 3, 4, 5] 2. new_list = my_list.copy() print(new_list) 3. # Output: [1, 2, 3, 4, 5]</pre> <div>Copied!</div>
count()	The `count()` method is used to count the number of occurrences of a specific element in a list in Python.	Example: <pre>1. 1 2. 2 3. 3 1. my_list = [1, 2, 2, 3, 4, 2, 5, 2] 2. count = my_list.count(2) print(count) 3. # Output: 4</pre> <div>Copied!</div>
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: <pre>1. 1 1. fruits = ["apple", "banana", "orange", "mango"]</pre> <div>Copied!</div>
del	The `del` statement is used to remove an element from list. `del` statement removes the element at the specified index.	Example: <pre>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 print(my_list) 3. # Output: [10, 20, 40, 50]</pre> <div>Copied!</div>
		Syntax: <pre>1. 1 1. list_name.extend(iterable)</pre> <div>Copied!</div>
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.	Example: <pre>1. 1 2. 2 3. 3 4. 4 1. fruits = ["apple", "banana", "orange"] 2. more_fruits = ["mango", "grape"] 3. fruits.extend(more_fruits) 4. print(fruits)</pre> <div>Copied!</div>
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: <pre>1. 1 2. 2 3. 3 4. 4 5. 5 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])</pre>

```
5. # Output: 50 (accessing the last element using negative indexing)
```

Copied!

Syntax:

```
1. 1
```

```
1. list_name.insert(index, element)
```

Copied!

Example:

```
1. 1
2. 2
3. 3
```

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

Copied!

Example:

```
1. 1
2. 2
3. 3
4. 4
```

```
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]
```

Copied!

Example 1:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
```

```
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]
```

Copied!

Example 2:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
```

```
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]
```

Copied!

Example:

```
1. 1
2. 2
3. 3
4. 4
```

```
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]
```

Copied!

Example 1:

```
1. 1
2. 2
3. 3
```

```
1. my_list = [1, 2, 3, 4, 5]
2. my_list.reverse() print(my_list)
3. # Output: [5, 4, 3, 2, 1]
```

Copied!

Syntax:

insert()

The `insert()` method is used to insert an element.

Modifying a list

You can use indexing to modify or assign new values to specific elements in the list.

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

remove()

To remove an element from a list. The `remove()` method removes the first occurrence of the specified value.

reverse()

The `reverse()` method is used to reverse the order of elements in a list

Slicing

You can use slicing to access a range of elements from a list.

```
1. 1
1. list_name[start:end:step]
```

Copied!

Example:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12

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)
```

Copied!

Example 1:

```
1. 1
2. 2
3. 3
4. 4

1. my_list = [5, 2, 8, 1, 9]
2. my_list.sort()
3. print(my_list)
4. # Output: [1, 2, 5, 8, 9]
```

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.

Copied!

Example 2:

```
1. 1
2. 2
3. 3
4. 4

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]
```

Copied!

Tuple

Package/Method	Description	Code Example
----------------	-------------	--------------

Syntax:

```
1. 1
1. tuple.count(value)
```

Copied!

count()

The count() method for a tuple is used to count how many times a specified element appears in the tuple.

Example:

```
1. 1
2. 2
3. 3

1. fruits = ("apple", "banana", "apple", "orange")
2. print(fruits.count("apple")) #Counts the number of times apple is found in tuple.
3. #Output: 2
```

Copied!

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:

```
1. 1
1. tuple.index(value)
```

Copied!

Example:

```
1. 1
2. 2
3. 3

1. fruits = ("apple", "banana", "orange")
2. print(fruits[1]) #Returns the value at which apple is present.
3. #Output: banana
```

Copied!

Syntax:

```
1. 1
1. sum(tuple)
```

Copied!

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).

Example:

```
1. 1
2. 2
3. 3

1. numbers = (10, 20, 5, 30)
2. print(sum(numbers))
3. #Output: 65
```

Copied!

Example:

```
1. 1
2. 2
3. 3
4. 4
5. 5

1. numbers = (10, 20, 5, 30)
2. print(min(numbers))
3. #Output: 5
4. print(max(numbers))
5. #Output: 30
```

min() and max()

Find the smallest (min()) or largest (max()) element in a tuple.

Copied!

Syntax:

```
1. 1
1. len(tuple)
```

Copied!

len()

Get the number of elements in the tuple using len().

Example:

```
1. 1
2. 2
3. 3

1. fruits = ("apple", "banana", "orange")
2. print(len(fruits)) #Returns length of the tuple.
3. #Output: 3
```

Copied!



Skills Network