

Python Data Structures Cheat Sheet

List

Package/Method	Description	Code Example
		Syntax: <pre>1. 1 1. list_name.append(element)</pre>
append()	The <code>`append()`</code> method is used to add an element to the end of a list.	<div>Copied!</div> <div>Example:<pre>1. 1 2. 2 1. fruits = ["apple", "banana", "orange"] 2. fruits.append("mango") print(fruits)</pre></div> <div>Copied!</div>
copy()	The <code>`copy()`</code> method is used to create a shallow copy of a list.	<div>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> <div>Copied!</div>
count()	The <code>`count()`</code> method is used to count the number of occurrences of a specific element in a list in Python.	<div>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> <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 <code>[]</code> and elements are separated by commas.	<div>Example:<pre>1. 1 1. fruits = ["apple", "banana", "orange", "mango"]</pre></div> <div>Copied!</div>

del	<p>The <code>del</code> statement is used to remove an element from list. <code>del</code> statement removes the element at the specified index.</p>	<p>Example:</p> <ol style="list-style-type: none">123 <pre>1. my_list = [10, 20, 30, 40, 50] 2. del my_list[2] # Removes the element at index 2 3. # Output: [10, 20, 40, 50]</pre> <p>Copied!</p>
extend()	<p>The <code>extend()</code> 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.</p>	<p>Syntax:</p> <ol style="list-style-type: none">11. list_name.extend(iterable) <p>Copied!</p> <p>Example:</p> <ol style="list-style-type: none">1234 <pre>1. fruits = ["apple", "banana", "orange"] 2. more_fruits = ["mango", "grape"] 3. fruits.extend(more_fruits) 4. print(fruits)</pre> <p>Copied!</p>
Indexing	<p>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 <code>length_of_list - 1</code>.</p>	<p>Example:</p> <ol style="list-style-type: none">12345 <pre>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)</pre> <p>Copied!</p> <p>Syntax:</p> <ol style="list-style-type: none">11. list_name.insert(index, element) <p>Copied!</p>
insert()	<p>The <code>insert()</code> method is used to insert an element.</p>	<p>Example:</p> <ol style="list-style-type: none">123 <pre>1. my_list = [1, 2, 3, 4, 5] 2. my_list.insert(2, 6) 3. print(my_list)</pre> <p>Copied!</p>

	Example:
Modifying a list	<div><div>You can use indexing to modify or assign new values to specific elements in the list.</div><div><div>1. 1</div><div>2. 2</div><div>3. 3</div><div>4. 4</div></div><div><div>1. my_list = [10, 20, 30, 40, 50]</div><div>2. my_list[1] = 25 # Modifying the second element</div><div>3. print(my_list)</div><div>4. # Output: [10, 25, 30, 40, 50]</div></div></div> <div>Copied!</div>
	Example 1:
pop()	<div><div><div>`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</div><div><div>1. 1</div><div>2. 2</div><div>3. 3</div><div>4. 4</div><div>5. 5</div><div>6. 6</div><div>7. 7</div></div><div><div>1. my_list = [10, 20, 30, 40, 50]</div><div>2. removed_element = my_list.pop(2) # Removes and returns the element at index 2</div><div>3. print(removed_element)</div><div>4. # Output: 30</div><div>5.</div><div>6. print(my_list)</div><div>7. # Output: [10, 20, 40, 50]</div></div></div><div>Copied!</div></div>
	Example 2:
	<div><div><div><div>1. 1</div><div>2. 2</div><div>3. 3</div><div>4. 4</div><div>5. 5</div><div>6. 6</div><div>7. 7</div></div><div><div>1. my_list = [10, 20, 30, 40, 50]</div><div>2. removed_element = my_list.pop() # Removes and returns the last element</div><div>3. print(removed_element)</div><div>4. # Output: 50</div><div>5.</div><div>6. print(my_list)</div><div>7. # Output: [10, 20, 30, 40]</div></div></div><div>Copied!</div></div>
remove()	<div><div>Example:</div><div><div>To remove an element from a list. The `remove()` method removes the first occurrence of the specified value.</div><div><div>1. 1</div><div>2. 2</div><div>3. 3</div><div>4. 4</div></div><div><div>1. my_list = [10, 20, 30, 40, 50]</div><div>2. my_list.remove(30) # Removes the element 30</div><div>3. print(my_list)</div><div>4. # Output: [10, 20, 40, 50]</div></div></div><div>Copied!</div></div>
reverse()	<div><div>Example 1:</div><div><div>The `reverse()` method is used to</div><div><div>1. 1</div><div>2. 2</div><div>3. 3</div></div></div></div>

reverse the
order of
elements in a
list

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

Copied!

Syntax:

```
1. 1

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

Copied!

Example:

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

Slicing

```
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:

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.

sort()

```
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]
```

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!

Dictionary

Package/Method	Description	Code Example
Accessing Values	You can access the values in a dictionary using their corresponding `keys`.	Syntax: <pre>1. 1 1. Value = dict_name["key_name"]</pre> <div>Copied!</div>
		Example: <pre>1. 1 2. 2 1. name = person["name"] 2. age = person["age"]</pre> <div>Copied!</div>
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: <pre>1. 1 1. dict_name[key] = value</pre> <div>Copied!</div>
		Example: <pre>1. 1 2. 2 1. person["Country"] = "USA" # A new entry will be created. 2. person["city"] = "Chicago" # Update the existing value for the same key</pre> <div>Copied!</div>
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: <pre>1. 1 1. dict_name.clear()</pre> <div>Copied!</div>
		Example: <pre>1. 1 1. grades.clear()</pre> <div>Copied!</div>

	<p>Syntax:</p> <pre>1. 1</pre> <pre>1. new_dict = dict_name.copy()</pre>
copy()	<p>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.</p> <p>Example:</p> <pre>1. 1 2. 2</pre> <pre>1. new_person = person.copy() 2. new_person = dict(person) # another way to create a copy of dictionary</pre> <p>Copied!</p>
Creating a Dictionary	<p>A dictionary is a built-in data type that represents a collection of key-value pairs. Dictionaries are enclosed in curly braces `{}`.</p> <p>Example:</p> <pre>1. 1 2. 2</pre> <pre>1. dict_name = {} #Creates an empty dictionary 2. person = { "name": "John", "age": 30, "city": "New York"}</pre> <p>Copied!</p>
del	<p>Syntax:</p> <pre>1. 1</pre> <pre>1. del dict_name[key]</pre> <p>Removes the specified key-value pair from the dictionary. Raises a `KeyError` if the key does not exist.</p> <p>Example:</p> <pre>1. 1</pre> <pre>1. del person["Country"]</pre> <p>Copied!</p>
items()	<p>Syntax:</p> <pre>1. 1</pre> <pre>1. items_list = list(dict_name.items())</pre> <p>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.</p> <p>Example:</p> <pre>1. 1</pre> <pre>1. info = list(person.items())</pre> <p>Copied!</p>
key existence	<p>Example:</p> <pre>1. 1 2. 2</pre> <pre>1. if "name" in person: 2. print("Name exists in the dictionary.")</pre> <p>Copied!</p>

		Syntax:
		1. 1
keys()	Retrieves all keys from the dictionary and converts them into a list. Useful for iterating or processing keys using list methods.	1. keys_list = list(dict_name.keys()) Copied!
		Example:
		1. 1
		1. person_keys = list(person.keys()) Copied!
		Syntax:
		1. 1
update()	The `update()` method merges the provided dictionary into the existing dictionary, adding or updating key-value pairs.	1. dict_name.update({key: value}) Copied!
		Example:
		1. 1
		1. person.update({"Profession": "Doctor"}) Copied!
		Syntax:
		1. 1
values()	Extracts all values from the dictionary and converts them into a list. This list can be used for further processing or analysis.	1. values_list = list(dict_name.values()) Copied!
		Example:
		1. 1
		1. person_values = list(person.values()) Copied!

Sets

Package/Method	Description	Code Example
		Syntax:
		1. 1
		1. set_name.add(element)
		Copied!
add()	Elements can be added to a set using the `add()` method. Duplicates are automatically removed, as sets only store unique values.	Example:
		1. 1
		1. fruits.add("mango")
		Copied!
clear()	The `clear()` method removes all elements from the set, resulting in an empty set. It updates the set in-place.	Syntax:

copy()

The `copy()` method creates a shallow copy of the set. Any modifications to the copy won't affect the original set.

```
1. 1
1. set_name.clear()
```

Copied!

Example:

```
1. 1
1. fruits.clear()
```

Copied!

Syntax:

```
1. 1
1. new_set = set_name.copy()
```

Copied!

Example:

```
1. 1
1. new_fruits = fruits.copy()
```

Copied!

Example:

```
1. 1
2. 2

1. empty_set = set() #Creating an Empty Set
2. fruits = {"apple", "banana", "orange"}
```

Copied!

Syntax:

```
1. 1
1. set_name.discard(element)
```

Copied!

Example:

```
1. 1
1. fruits.discard("apple")
```

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.

discard()

Use the `discard()` method to remove a specific element from the set. Ignores if the element is not found.

		<p>Syntax:</p> <pre>1. 1 1. is_subset = set1.issubset(set2)</pre> <p>Copied!</p>
issubset()	<p>The <code>issubset()</code> 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.</p>	<p>Example:</p> <pre>1. 1 1. is_subset = fruits.issubset(colors)</pre> <p>Copied!</p>
		<p>Syntax:</p> <pre>1. 1 1. is_superset = set1.issuperset(set2)</pre> <p>Copied!</p>
issuperset()	<p>The <code>issuperset()</code> 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.</p>	<p>Example:</p> <pre>1. 1 1. is_superset = colors.issuperset(fruits)</pre> <p>Copied!</p>
		<p>Syntax:</p> <pre>1. 1 1. removed_element = set_name.pop()</pre> <p>Copied!</p>
pop()	<p>The <code>pop()</code> method removes and returns an arbitrary element from the set. It raises a <code>KeyError</code> if the set is empty. Use this method to remove elements when the order doesn't matter.</p>	<p>Example:</p> <pre>1. 1 1. removed_fruit = fruits.pop()</pre> <p>Copied!</p>
		<p>Syntax:</p> <pre>1. 1 1. set_name.remove(element)</pre> <p>Copied!</p>
remove()	<p>Use the <code>remove()</code> method to remove a specific element from the set. Raises a <code>KeyError</code> if the element is not found.</p>	<p>Example:</p> <pre>1. 1 1. fruits.remove("banana")</pre> <p>Copied!</p>
		<p>Syntax:</p> <pre>1. 1 2. 2 3. 3 4. 4 1. union_set = set1.union(set2)</pre>
Set Operations	<p>Perform various operations on sets: <code>union</code>, <code>intersection</code>, <code>difference</code>, <code>symmetric difference</code>.</p>	

update()

The `update()` method adds elements from another iterable into the set. It maintains the uniqueness of elements.

```
2. intersection_set = set1.intersection(set2)
3. difference_set = set1.difference(set2)
4. sym_diff_set = set1.symmetric_difference(set2)
```

Copied!

Example:

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

```
1. combined = fruits.union(colors)
2. common = fruits.intersection(colors)
3. unique_to_fruits = fruits.difference(colors)
4. sym_diff = fruits.symmetric_difference(colors)
```

Copied!

Syntax:

```
1. 1
```

```
1. set_name.update(iterable)
```

Copied!

Example:

```
1. 1
```

```
1. fruits.update(["kiwi", "grape"])
```

Copied!



Skills Network

© IBM Corporation. All rights reserved.