

Python Data Structures Cheat Sheet

List

Package/Method	Description
<code>append()</code>	The 'append()' method is used to add an element to the end of a list.
<code>copy()</code>	The 'copy()' method is used to create a shallow copy of a list.
<code>count()</code>	The 'count()' method is used to count the number of occurrences of a specific element in a list in Python.
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.
<code>del</code>	The 'del' statement is used to remove an element from list. 'del' statement removes the element at the specified index.
<code>extend()</code>	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.
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'.
<code>insert()</code>	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.
<code>pop()</code>	'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.
<code>remove()</code>	To remove an element from a list, the 'remove()' method removes the first occurrence of the specified value.
<code>reverse()</code>	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.
<code>sort()</code>	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.

Code Example

```
System:
1. 1
1. list_name.append(element)
[Copy]

Example:
1. 1
1. 2
1. 3
1. fruits = ["apple", "banana", "orange"]
1. fruits.append("mango") # Add mango to the list
1. print(fruits)
[Copy]

Example 1:
1. 1
1. 2
1. 3
1. my_list = [1, 2, 3, 4, 5]
1. my_list = my_list.copy() # Create a shallow copy
1. # Output: [1, 2, 3, 4, 5]
[Copy]

Example:
1. 1
1. 2
1. 3
1. my_list = [1, 2, 3, 4, 5, 2]
1. count = my_list.count(2) # Count the number of occurrences of 2
1. # Output: 2
[Copy]

Example:
1. 1
1. fruits = ["apple", "banana", "orange", "mango"]
[Copy]

Example:
1. 1
1. 2
1. 3
1. 4
1. my_list = [10, 20, 30, 40, 50]
1. del my_list[2] # Remove the element at index 2
1. # Output: [10, 20, 40, 50]
[Copy]

System:
1. 1
1. list_name.extend(iterable)
[Copy]

Example:
1. 1
1. 2
1. 3
1. 4
1. fruits = ["apple", "banana", "orange"]
1. more_fruits = ["mango", "grape"]
1. fruits.extend(more_fruits)
1. print(fruits)
[Copy]

Example:
1. 1
1. 2
1. 3
1. 4
1. 5
1. my_list = [10, 20, 30, 40, 50]
1. print(my_list[0])
1. # Output: 10 (Accessing the first element)
1. print(my_list[-1])
1. # Output: 50 (Accessing the last element using negative indexing)
[Copy]

System:
1. 1
1. list_name.insert(index, element)
[Copy]

Example:
1. 1
1. 2
1. 3
1. 4
1. my_list = [1, 2, 3, 4, 5]
1. my_list.insert(2, 0) # Insert 0 at index 2
1. print(my_list)
[Copy]

Example:
1. 1
1. 2
1. 3
1. 4
1. my_list = [10, 20, 30, 40, 50]
1. my_list[1] = 25 # Modify the second element
1. print(my_list)
1. # Output: [10, 25, 30, 40, 50]
[Copy]

Example 1:
1. 1
1. 2
1. 3
1. 4
1. 5
1. 6
1. 7
1. my_list = [10, 20, 30, 40, 50]
1. removed_element = my_list.pop(2) # Removes and returns the element at index 2
1. print(removed_element)
1. # Output: 30
1. # Output: [10, 20, 40, 50]
[Copy]

Example 2:
1. 1
1. 2
1. 3
1. 4
1. 5
1. 6
1. 7
1. 8
1. my_list = [10, 20, 30, 40, 50]
1. removed_element = my_list.pop() # Removes and returns the last element
1. print(removed_element)
1. # Output: 50
1. # Output: [10, 20, 30, 40]
[Copy]

Example:
1. 1
1. 2
1. 3
1. 4
1. 5
1. my_list = [10, 20, 30, 40, 50]
1. my_list.remove(30) # Removes the element 30
1. print(my_list)
1. # Output: [10, 20, 40, 50]
[Copy]

Example 1:
1. 1
1. 2
1. 3
1. 4
1. 5
1. my_list = [1, 2, 3, 4, 5]
1. my_list.reverse() # Reverse the list
1. # Output: [5, 4, 3, 2, 1]
[Copy]

System:
1. 1
1. list_name[start:stop]
[Copy]

Example:
1. 1
1. 2
1. 3
1. 4
1. 5
1. 6
1. 7
1. 8
1. 9
1. 10
1. 11
1. 12
1. my_list = [1, 2, 3, 4, 5]
1. print(my_list[1:4])
1. # Output: [2, 3, 4] (Elements from index 1 to 3)
1. # Output: [1, 2, 3] (Elements from the beginning up to index 2)
1. # Output: [2, 3, 4, 5] (Elements from index 2 to the end)
1. # Output: [1, 3, 5] (Every second element)
[Copy]

Example 1:
1. 1
1. 2
1. 3
1. 4
1. 5
1. my_list = [1, 2, 3, 4, 5]
1. my_list.sort()
1. print(my_list)
1. # Output: [1, 2, 3, 4, 5]
[Copy]

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

Dictionary

Package/Method	Description
Accessing Values	You can access the values in a dictionary using their corresponding 'key'.
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.

```
System:
1. 1
1. value = dict_name["key_name"]
[Copy]

Example:
1. 1
1. 2
1. name = "person"
1. age = 30
1. # Create a dictionary
[Copy]

System:
1. 1
1. dict_name[key] = value
```

Package Method	Description	Code Example
		<pre>System: 1. 1 1. set_name.add(element) (Copy)</pre>
add()	Elements can be added to a set using the 'add()' method. Duplicates are automatically removed, so sets only store unique values.	<pre>Example: 1. 1 1. fruits.add("orange") (Copy)</pre> <pre>System: 1. 1 1. set_name.clear() (Copy)</pre>
clear()	The 'clear()' method removes all elements from the set, resulting in an empty set. It updates the set in place.	<pre>Example: 1. 1 1. fruits.clear() (Copy)</pre> <pre>System: 1. 1 1. new_set = set_name.copy() (Copy)</pre>
copy()	The 'copy()' method creates a shallow copy of the set. Any modifications to the copy won't affect the original set.	<pre>Example: 1. 1 1. new_fruits = fruits.copy() (Copy)</pre> <pre>Example: 1. 1 1. 2 2. 2 (Copy)</pre>
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.	<pre>1. empty_set = set() #Creating an Empty Set 2. fruits = {"apple", "banana", "orange"} (Copy)</pre> <pre>System: 1. 1 1. set_name.discard(element) (Copy)</pre>
discard()	Use the 'discard()' method to remove a specific element from the set. Ignores if the element is not found.	<pre>Example: 1. 1 1. fruits.discard('apple') (Copy)</pre> <pre>System: 1. 1 1. is_subset = set1.issubset(set2) (Copy)</pre>
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.	<pre>Example: 1. 1 1. is_subset = fruits.issubset(colors) (Copy)</pre> <pre>System: 1. 1 1. is_superaset = set1.isuperaset(set2) (Copy)</pre>
isproper()	The 'isproper()' method checks if the current set is a proper superset of another set. It returns True if all elements of the other set are present in the current set, otherwise False.	<pre>Example: 1. 1 1. is_properaset = colors.isproperaset(fruits) (Copy)</pre> <pre>System: 1. 1 1. removed_element = set_name.pop() (Copy)</pre>
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.	<pre>Example: 1. 1 1. removed_fruit = fruits.pop() (Copy)</pre> <pre>System: 1. 1 1. set_name.remove(element) (Copy)</pre>
remove()	Use the 'remove()' method to remove a specific element from the set. Raises a 'KeyError' if the element is not found.	<pre>Example: 1. 1 1. fruits.remove("banana") (Copy)</pre> <pre>System: 1. 1 1. 2 2. 2 2. 2 2. 4 (Copy)</pre>
Set Operations	Perform various operations on sets: 'union()', 'intersection()', 'difference()', 'symmetric difference'.	<pre>1. union_set = set1.union(set2) 2. intersection_set = set1.intersection(set2) 3. difference_set = set1.difference(set2) 4. sym_diff_set = set1.symmetric_difference(set2) (Copy)</pre>

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



© IBM Corporation. All rights reserved.

```
Example:
1. 1
2. 2
3. 2
4. 4

1. combined = fruits.union(colors)
2. common = fruits.intersection(colors)
3. unique_fruits = fruits.difference(colors)
4. sym_diff = fruits.symmetric_difference(colors)
[Copy]
```

```
System:
1. 1

1. set_name.update(iterable)
[Copy]
```

```
Example:
1. 1

1. fruits.update(["banana", "orange"])
[Copy]
```