

Cheat Sheet: Python Data Structures Part-2

Dictionaries

Package/Method	Description	Code Example
Creating a Dictionary	A dictionary in Python is a built-in data structure that stores information in key-value pairs. Each key must be unique, and it is used to access its corresponding value. Dictionaries are enclosed in curly braces {}, and each pair is separated by a comma.	Example: <pre>dict_name = {} #Creates an empty dictionary person = { "name": "John", "age": 30, "city": "New York"}</pre>
Accessing Values	You can access the values in a dictionary using their corresponding keys.	Syntax: <pre>Value = dict_name["key_name"]</pre> Example: <pre>name = person["name"] age = person["age"]</pre>
Add or modify	You can add or modify elements in a dictionary by assigning a value to a key using the assignment operator =. When you assign a value to a key that does not already exist, Python automatically creates a new key-value pair and adds it to the dictionary. However, if the key already exists, the existing value will be replaced with the new one.	Syntax: <pre>dict_name[key] = value</pre> Example: <pre>person["Country"] = "USA" # A new entry will be created. person["city"] = "Chicago" # Update the existing value for the same key</pre>
del	Removes the specified key-value pair from the dictionary. Raises a <code>KeyError</code> if the key does not exist.	Syntax: <pre>del dict_name[key]</pre>

		<p>Example:</p> <pre>del person["Country"]</pre>
key existence	You can check for the existence of a key in a dictionary using the <code>in</code> keyword	<p>Example:</p> <pre>if "name" in person: print("Name exists in the dictionary.")</pre>
keys()	Retrieves all keys from the dictionary and converts them into a list. Useful for iterating or processing keys using list methods.	<p>Syntax:</p> <pre>keys_list = list(dict_name.keys())</pre> <p>Example:</p> <pre>person_keys = list(person.keys())</pre>
values()	Extracts all values from the dictionary and converts them into a list. This list can be used for further processing or analysis.	<p>Syntax:</p> <pre>values_list = list(dict_name.values())</pre> <p>Example:</p> <pre>person_values = list(person.values())</pre>
items()	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>Syntax:</p> <pre>items_list = list(dict_name.items())</pre>

		<p>Example:</p> <pre>info = list(person.items())</pre>
--	--	--

Sets

Package/Method	Description	Code Example
add()	Add elements to a set. You can use the <code>add()</code> method to insert a new element into a set. If the element already exists, it will not be added again, since sets automatically store only unique values.	<p>Syntax:</p> <pre>set_name.add(element)</pre> <p>Example:</p> <pre>fruits.add("mango")</pre>
Defining Sets	<p>A set in Python is an unordered collection that stores unique elements, meaning it automatically removes duplicates. Sets are enclosed in curly braces <code>{}</code> and are commonly used when you need to store a group of distinct values or perform mathematical operations such as union, intersection, and difference.</p> <p>Because sets are unordered, the elements have no fixed position or index, and their order may change each time you print or iterate over them.</p>	<p>Example:</p> <pre>empty_set = set() #Creating an Empty Set fruits = {"apple", "banana", "orange"} colors = ("orange", "red", "green")</pre> <p>Note: These two sets will be used in the examples that follow.</p>
in	Checks if an element exists in a set. Use the <code>in</code> keyword to verify whether a specific element is present in a set. It returns <code>True</code> if the element exists, and <code>False</code> otherwise.	<p>Syntax:</p> <pre>element in set_name</pre> <p>Example:</p> <pre>"banana" in fruits</pre>

issubset()	<p>Check if one set is a subset of another. The <code>issubset</code> method is used to determine whether all elements of one set exist within another set. If every element of the current set is found in the specified set, it returns <code>True</code>; otherwise, it returns <code>False</code>.</p>	<p>Syntax:</p> <pre>is_subset = set1.issubset(set2)</pre> <p>Example:</p> <pre>is_subset = fruits.issubset(colors)</pre>
issuperset()	<p>Check if one set is a superset of another. The <code>issuperset()</code> method is used to verify whether the current set contains all elements of another set. It returns <code>True</code> if every element of the specified set exists within the current set; otherwise, it returns <code>False</code>.</p>	<p>Syntax:</p> <pre>is_superset = set1.issuperset(set2)</pre> <p>Example:</p> <pre>is_superset = colors.issuperset(fruits)</pre>
remove()	<p>Remove a specific element from a set. The <code>remove()</code> method is used to delete a particular element from a set. If the element you try to remove does not exist, Python will raise a <code>KeyError</code>, which means you must ensure the element is present before calling this method.</p> <p>This method is useful when you are certain that the item you want to remove is already in the set.</p>	<p>Syntax:</p> <pre>set_name.remove(element)</pre> <p>Example:</p> <pre>fruits.remove("banana")</pre>
Set Operations	<p>Perform various operations on sets: union, intersection, difference, symmetric difference.</p>	<p>Syntax:</p> <pre>union_set = set1.union(set2) intersection_set = set1.intersection(set2) difference_set = set1.difference(set2) sym_diff_set = set1.symmetric_difference(set2)</pre>

Example:

```
combined = fruits.union(colors)
common = fruits.intersection(colors)
unique_to_fruits = fruits.difference(colors)
sym_diff = fruits.symmetric_difference(colors)
```

