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:
		`dict_name = {} #Creates an empty dictionary`
		`person = { "name": "John",  "age": 30, "city": "New York"}`

Accessing Values	You can access the values in a dictionary using their corresponding `keys`.	Syntax:
		`Value = dict_name["key_name"]`
		Example:
		`name = person["name"]`
		`age = person["age"]`

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:
		`dict_name[key] = value`
		Example:
		`person["Country"] = "USA" # A new entry will be created.`
		`person["city"] = "Chicago"  # Update the existing value for the same key`

del	Removes the specified key-value pair from the dictionary. Raises a `KeyError` if the key does not exist.	Syntax: `del dict_name[key]` Example:

		`del person["Country"]`

key existence	You can check for the existence of a key in a dictionary using the `in` keyword	Example:
		`if "name" in person:`
		`  print("Name exists in the dictionary.")`

keys()	Retrieves all keys from the dictionary and converts them into a list. Useful for iterating or processing keys using list methods.	Syntax:
		`keys_list = list(dict_name.keys())`
		Example:
		`person_keys = list(person.keys())`

values()	Extracts all values from the dictionary and converts them into a list. This list can be used for further processing or analysis.	Syntax:
		`values_list = list(dict_name.values())`
		Example:
		`person_values = list(person.values())`

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.	Syntax: `items_list = list(dict_name.items())` Example:

		`info = list(person.items())`

Sets

Package/Method	Description	Code Example

add()	Add elements to a set. You can use the `add()` 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.	Syntax:
		`set_name.add(element)`
		Example:
		`fruits.add("mango")`

Defining Sets	A set in Python is an unordered collection that stores unique elements, meaning it automatically removes duplicates. Sets are enclosed in curly braces {} and are commonly used when you need to store a group of distinct values or perform mathematical operations such as union, intersection, and difference.
	
	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.
		Example:
		`empty_set = set() #Creating an Empty Set`
		`fruits = {"apple", "banana", "orange"}`
		`colors = ("orange", "red", "green")`
		
		Note: These two sets will be used in the examples that follow.

in	Checks if an element exists in a set. Use the `in` keyword to verify whether a specific element is present in a set. It returns `True` if the element exists, and `False` otherwise.	Syntax:
		`element in set_name`
		Example:
		`"banana" in fruits`

issubset()	Check if one set is a subset of another. The `issubset` 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 `True`; otherwise, it returns `False`.	Syntax: `is_subset = set1.issubset(set2)`

		Example: `is_subset = fruits.issubset(colors)`

issuperset()	Check if one set is a superset of another. The `issuperset()` method is used to verify whether the current set contains all elements of another set. It returns `True` if every element of the specified set exists within the current set; otherwise, it returns `False`.	Syntax:
		`is_superset = set1.issuperset(set2)`
		Example:
		`is_superset = colors.issuperset(fruits)`

remove()	Remove a specific element from a set. The `remove()` 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 `KeyError`, which means you must ensure the element is present before calling this method.
	This method is useful when you are certain that the item you want to remove is already in the set.
		Syntax:
		`set_name.remove(element)`
		Example:
		`fruits.remove("banana")`

Set Operations	Perform various operations on sets: `union`, `intersection`, `difference`, `symmetric difference`.	Syntax:
		`union_set = set1.union(set2)`
		`intersection_set = set1.intersection(set2)`
		`difference_set = set1.difference(set2)`
		`sym_diff_set = set1.symmetric_difference(set2)`
		Example:
		`combined = fruits.union(colors)`
		`common = fruits.intersection(colors)`
		`unique_to_fruits = fruits.difference(colors)`
		`sym_diff = fruits.symmetric_difference(colors)`

Skills Network