Cheat Sheet: Python Data Structures Part-2

Dictionaries

Package/Method	Description	Code Example
| Creating a Dictionary | A dictionary is a built-in data type that represents a collection of key-value pairs. Dictionaries are enclosed in curly braces {}. | Example:<br><br>```python
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:<br><br>```python
Value = dict_name["key_name"]
```<br><br>Example:<br><br>```python
name = person["name"]
age = person["age"]
``` |
| 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:<br><br>```python
dict_name[key] = value
```<br><br>Example:<br><br>```python
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:<br><br>```python
del dict_name[key]
``` |

		Example:
		`del person["Country"]`

update()	The `update()` method merges the provided dictionary into the existing dictionary, adding or updating key-value pairs.	Syntax:
		
		`dict_name.update({key: value})`
		
		Example:
		
		`person.update({"Profession": "Doctor"})`

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:
		
		`dict_name.clear()`
		
		Example:
		
		`grades.clear()`

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.")`

copy()	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.	Syntax:
		
		`new_dict = dict_name.copy()`
		
		Example:
		
		`new_person = person.copy()`
		`new_person = dict(person) # another way to create a copy of 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())`

		Syntax:

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.	`items_list = list(dict_name.items())`
		
		Example:
		
		`info = list(person.items())`

Sets

Package/Method	Description	Code Example

add()	Elements can be added to a set using the `add()` method. Duplicates are automatically removed, as sets only store unique values.	Syntax:
		
		`set_name.add(element)`
		
		Example:
		
		`fruits.add("mango")`

clear()	The `clear()` method removes all elements from the set, resulting in an empty set. It updates the set in-place.	Syntax:
		
		`set_name.clear()`
		
		Example:
		
		`fruits.clear()`

copy()	The `copy()` method creates a shallow copy of the set. Any modifications to the copy won't affect the original set.	Syntax:
		
		```\nnew_set = set_name.copy()\n```
		
		Example:
		
		```\nnew_fruits = fruits.copy()\n```

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.	Example: ```\nempty_set = set() #Creating an Empty Set\nfruits = {"apple", "banana", "orange"}\ncolors = ("orange", "red", "green")\n``` **Note:** These two sets will be used in the examples that follow.

discard()	Use the `discard()` method to remove a specific element from the set. Ignores if the element is not found.	Syntax:
		
		```\nset_name.discard(element)\n```
		
		Example:
		
		```\nfruits.discard("apple")\n```

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.	Syntax: ```\nis_subset = set1.issubset(set2)\n``` Example:

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

issuperset()	The `issuperset()` 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.	Syntax:
		
		`is_superset = set1.issuperset(set2)`
		
		Example:
		
		`is_superset = colors.issuperset(fruits)`

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.	Syntax:
		
		`removed_element = set_name.pop()`
		
		Example:
		
		`removed_fruit = fruits.pop()`

remove()	Use the `remove()` method to remove a specific element from the set. Raises a `KeyError` if the element is not found.	Syntax:
		
		`set_name.remove(element)`
		
		Example:
		
		`fruits.remove("banana")`

| Set Operations | Perform various operations on sets: `union`, `intersection`, `difference`, `symmetric difference`. | Syntax:

```\nunion_set = set1.union(set2)\nintersection_set = set1.intersection(set2)\ndifference_set = set1.difference(set2)\nsym_diff_set = set1.symmetric_difference(set2)\n```

Example:

```\ncombined = fruits.union(colors)\ncommon = fruits.intersection(colors)\nunique_to_fruits = fruits.difference(colors)\nsym_diff = fruits.symmetric_difference(colors)\n``` |
| update() | The `update()` method adds elements from another iterable into the set. It maintains the uniqueness of elements. | Syntax:

```\nset_name.update(iterable)\n```

Example:

```\nfruits.update(["kiwi", "grape"])\n``` |

Skills Network