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:
		
		```\n1. 1\n2. 2\n```
		
		```\n1. dict_name = {} #Creates an empty dictionary\n2. person = { "name": "John", "age": 30, "city": "New York"}\n```
		Copied!

Accessing Values	You can access the values in a dictionary using their corresponding `keys`.	Syntax:
		
		```\n1. 1\n```
		
		```\n1. Value = dict_name["key_name"]\n```
		Copied!
		
		Example:
		
		```\n1. 1\n2. 2\n```
		
		```\n1. name = person["name"]\n2. age = person["age"]\n```
		Copied!

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:
		
		```\n1. 1\n```
		
		```\n1. dict_name[key] = value\n```
		Copied!
		
		Example:
		
		```\n1. 1\n2. 2\n```
		
		```\n1. person["Country"] = "USA" # A new entry will be created.\n2. person["city"] = "Chicago" # Update the existing value for the same key\n```
		Copied!

del	Removes the specified key-value pair from the dictionary. Raises a `KeyError` if the key does not exist.	Syntax:
		
		```\n1. 1\n```
		
		```\n1. del dict_name[key]\n```
		Copied!
		
		Example:
		
		```\n1. 1\n```
		
		```\n1. del person["Country"]\n```

Syntax:

1. 1

```
1. dict_name.update({key: value})
```

| update() | The `update()` method merges the provided dictionary into the existing dictionary, adding or updating key-value pairs. |

Example:

1. 1

```
1. person.update({"Profession": "Doctor"})
```

Syntax:

1. 1

```
1. dict_name.clear()
```

| 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. |

Example:

1. 1

```
1. grades.clear()
```

Example:

1. 1
2. 2

```
1. if "name" in person:
2.     print("Name exists in the dictionary.")
```

| key existence | You can check for the existence of a key in a dictionary using the `in` keyword |

Syntax:

1. 1

```
1. new_dict = dict_name.copy()
```

| 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. |

Example:

1. 1
2. 2

```
1. new_person = person.copy()
2. 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:

1. 1

```
1. keys_list = list(dict_name.keys())
```

Example:

```
1. 1
```
```
1. person_keys = list(person.keys())
```

Copied!

Syntax:

```
1. 1
```
```
1. values_list = list(dict_name.values())
```

Copied!

| values() | Extracts all values from the dictionary and converts them into a list. This list can be used for further processing or analysis. | Example:

```
1. 1
```
```
1. person_values = list(person.values())
```

Copied!

Syntax:

```
1. 1
```
```
1. items_list = list(dict_name.items())
```

Copied! |
|---|---|---|
| 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. | Example:

```
1. 1
```
```
1. info = list(person.items())
```

Copied! |

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:

```
1. 1
```
```
1. set_name.add(element)
```

Copied!

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:

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

Copied!

Example:

```
1. 1
``` |

Syntax:

```
1. 1
```

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

Copied!

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

Example:

```
1. 1
```

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

Copied!

Example:

```
1. 1
2. 2
```

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

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.

Syntax:

```
1. 1
```

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

Copied!

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

Example:

```
1. 1
```

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

Copied!

Syntax:

```
1. 1
```

```
1. is_subset = set1.issubset(set2)
```

Copied!

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.

Example:

```
1. 1
```

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

Copied!

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:

```
1. 1
```

```
1. 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.	

Copied!

Syntax:

```
1. 1
```

```
1. removed_element = set_name.pop()
```

Copied!

Example:

```
1. 1
```

```
1. removed_fruit = fruits.pop()
```

Copied!

Syntax:

```
1. 1
```

```
1. set_name.remove(element)
```

Copied!

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

Example:

```
1. 1
```

```
1. fruits.remove("banana")
```

Copied!

Syntax:

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

```
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)
```

Copied!

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

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!

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

Syntax:

```
1. 1
```

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

Copied!

Example:

1. 1

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

Copied!