Cheatsheet

A concise cheat sheet covering Python dictionaries and tuples, including creation, manipulation, and common operations. Ideal for quick reference and efficient coding.

Dictionaries: Creation and Access

Dictionary Creation

Empty Dictionary

```
my_dict = {}
my_dict = dict()
```

Dictionary with Initial Values

```
my_dict = {'key1': 'value1', 'key2': 'value2'}
```

Dictionary using `dict()` constructor

```
my_dict = dict(key1='value1', key2='value2')
```

From a list of tuples

```
tuples_list = [('key1', 'value1'), ('key2', 'value2')]
my_dict = dict(tuples_list)
```

Accessing Values

Using square brackets `[]`

```
my_dict = {'key1': 'value1', 'key2': 'value2'}
value = my_dict['key1']
# value is 'value1'
```

Using `.get()` method

```
my_dict = {'key1': 'value1', 'key2': 'value2'}
value = my_dict.get('key1')  # value is 'value1'
value = my_dict.get('key3')  # value is None
value = my_dict.get('key3', 'default_value')  # value is 'default_value'
```

Handling missing keys

```
try:
    value = my_dict['key3']
except KeyError:
    print('Key does not exist')
```

Modifying Dictionaries

Adding new key-value pairs

```
my_dict['key3'] = 'value3'
```

Updating existing values

```
my_dict['key1'] = 'new_value1'
```

Deleting key-value pairs

```
del my_dict['key2']
```

Using `.pop()` to remove and return a value

```
value = my_dict.pop('key1')
```

Using `.popitem()` to remove and return the last inserted key-value pair

```
key, value = my_dict.popitem()
```

Dictionary Operations

Common Dictionary Methods

`.keys()`	Returns a view object that displays a list of all the keys in the dictionary. `keys = my_dict.keys()`
`.values()`	Returns a view object that displays a list of all the values in the dictionary. `values = my_dict.values()`
`.items()`	Returns a view object that displays a list of a dictionary's key-value tuple pairs. `items = my_dict.items()`
`.clear()`	Removes all items from the dictionary. `my_dict.clear()`
`.copy()`	Returns a shallow copy of the dictionary. `new_dict = my_dict.copy()`

`.update()`	Updates the dictionary with the elements from another dictionary object or from an iterable of key/value pairs.
	`my_dict.update({'key3': 'value3'})`
	`my_dict.update(key4='value4')`

Dictionary Comprehension

Basic Syntax	`{key: value for item in iterable}`
Example: Creating a dictionary of squares	`squares = {x: x*x for x in range(6)}` `# squares == {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}`
Conditional Dictionary Comprehension	`{key: value for item in iterable if condition}`
Example: Filtering even numbers	`even_squares = {x: x*x for x in range(10) if x % 2 == 0}` `# even_squares == {0: 0, 2: 4, 4: 16, 6: 36, 8: 64}`

Membership Test

Using `in` keyword	Checks if a key is present in the dictionary.
	`my_dict = {'key1': 'value1', 'key2': 'value2'}`
	`'key1' in my_dict  # Returns True`
	`'key3' in my_dict  # Returns False`

Using `not in` keyword	Checks if a key is not present in the dictionary.
	`my_dict = {'key1': 'value1', 'key2': 'value2'}`
	`'key3' not in my_dict  # Returns True`
	`'key1' not in my_dict  # Returns False`

Tuples: Creation and Basic Operations

Tuple Creation

Empty Tuple	`my_tuple = ()` `my_tuple = tuple()`
Tuple with Initial Values	`my_tuple = (1, 2, 3)` `my_tuple = 1, 2, 3  # Parentheses are optional`
Tuple with a single element	`my_tuple = (1,)` `my_tuple = 1,  # Trailing comma is important`
Using `tuple()` constructor	`my_tuple = tuple([1, 2, 3])` `my_tuple = tuple('abc')  # ('a', 'b', 'c')`

Accessing Tuple Elements

Using indexing	`my_tuple = (1, 2, 3)`
	`first_element = my_tuple[0]  # first_element is 1`
	`second_element = my_tuple[1]  # second_element is 2`

Negative indexing	`my_tuple = (1, 2, 3)` `last_element = my_tuple[-1]  # last_element is 3`

Slicing	`my_tuple = (1, 2, 3, 4, 5)`
	`sub_tuple = my_tuple[1:4]`
	`# sub_tuple is (2, 3, 4)`

Tuple Operations

Concatenation	`tuple1 = (1, 2)`
	`tuple2 = (3, 4)`
	`combined_tuple = tuple1 + tuple2  # (1, 2, 3, 4)`

Repetition	`my_tuple = (1, 2)` `repeated_tuple = my_tuple * 3  # (1, 2, 1, 2, 1, 2)`

Length	`my_tuple = (1, 2, 3)`
	`length = len(my_tuple)`
	`# length is 3`

Membership Test	`my_tuple = (1, 2, 3)`
	`1 in my_tuple  # Returns True`
	`4 in my_tuple  # Returns False`

Tuple Use Cases and Methods

Tuple Unpacking

Basic Unpacking	`my_tuple = (1, 2, 3)` `a, b, c = my_tuple  # a=1, b=2, c=3`

Unpacking with `*` (star operator)	`my_tuple = (1, 2, 3, 4, 5)`
	`a, b, *rest = my_tuple  # a=1, b=2, rest=[3, 4, 5]`
	`a, *middle, c = my_tuple  # a=1, middle=[2, 3, 4], c=5`

Ignoring values during unpacking	`my_tuple = (1, 2, 3)` `a, _, c = my_tuple  # a=1, c=3, value 2 is ignored`

Tuple Methods

`.count(value)`	Returns the number of times a specified value occurs in a tuple.
	`my_tuple = (1, 2, 2, 3, 2)`
	`count = my_tuple.count(2)  # count is 3`

`.index(value)`	Returns the index of the first occurrence of a value.
	`my_tuple = (1, 2, 3, 2)`
	`index = my_tuple.index(2)  # index is 1`

When to Use Tuples

- **Data Integrity:** When you want to ensure that data remains constant.
- **Returning Multiple Values:** From a function.
- **Keys in Dictionaries:** Tuples can be used as keys, lists cannot (because they are mutable).
- **Representing Records:** Lightweight data structures.
- **Read-Only Data:** Scenarios where the data should not be modified after creation.