

Dictionaries

Collections in Python

Dictionaries are associative arrays.
They:

- Have an **order**.
- **Allow duplicate** values.
- **Allow** their objects to be **changed**.
- Allow objects of **different types**.

Defining Dictionaries

They are defined using curly brackets `{}`. Each key-value pair is separated by a comma and every key is separated by a colon from the value.



Some **iterables** can be converted into a dictionary by using the `dict` constructor.



The `dict` constructor can also be used to create empty tuples if no argument is given.



```
>>> address = {
...     "name": "Harry Potter",
...     "street": "Private Drive",
...     "number": 4,
...     "city": "Little Whinging",
...     "county": "Surrey",
... }
>>> choice_dict = dict(choices)
>>> print(choice_dict)
{'Mon': 'Monday', 'Tue': 'Tuesday', ...}
>>> empty_dict = dict()
>>> print(empty_dict)
()
```

Defining Dictionaries

Dictionaries can contain values of any type.

Dictionary indices (keys) can also be of any type.

Items in the dictionary can be of mixed types and values can be repeated.

The items themselves can also be dictionaries.

```
>>> values = {"a": 1, "b": "string"}
>>> keys = {1: "one", date: "day"}
>>> repeated = {1: "hi", 2: "hi"}
>>> users = {
...     "john": {
...         "name": "John Doe",
...         "age": 30
...     },
...     "jane": {
...         "name": "Jane Doe",
...         "age": 40
...     }
... }
```

Defining Dictionaries

Dictionaries can also be initialized with the **fromkeys** class method.



This method takes a first argument as a sequence of keys and an optional second argument as the default value to initialize the values of each key.

Dictionaries can also be created by copying other dictionaries, using **copy**.



```
>>> template = dict.fromkeys(
...     ("street", "number", "zip",
...     "city", "country"),
...     "Unknown"
... )
>>> print(template)
{'street': 'Unknown', 'number': 'Unknown',
'zip': 'Unknown', 'city': 'Unknown',
'country': 'Unknown'}
>>> address = template.copy()
>>> address.update({"street": "Hogwarts"})
{'street': 'Hogwarts', 'number':
'Unknown', 'zip': 'Unknown', 'city':
'Unknown', 'country': 'Unknown'}
```


Python Dictionaries: Accessing Values

Printing the dictionary will show its values in the **same order** used when the list was defined.

!! This is only true in versions >3.6 of the Python interpreter. In Python 3.6 dictionaries do not have an order.

Each value in the dictionary can be accessed indexing its **key** or using the **get** method.

The **get** method accepts a second argument that will be used as default value if the given key does not exist.



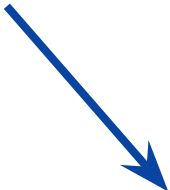

```
>>> print(users)
{'john': {'name': 'John Doe', ...
>>> print(users["john"])
{'name': 'John Doe', 'age': 30}
>>> print(users.get("jane"))
{'name': 'Jane Doe', 'age': 40}
>>> print(users.get(
...     "mario",
...     {"name": "Unknown"}
... ))
{"name": "Unknown"}
```

Python Dictionaries: Accessing Values

The **setdefault** method works similarly to the **get** method. It returns the value if the key exists, and if not it returns the default value given.

The difference is that if the given key does not exist in the dictionary, this key is created with the given default value.

If no default value is passed, it is created with the value **None**.



```
>>> print(users.setdefault(
...     "mario",
...     {"name": "Mario Sanz",
...      "age": 25}
... ))
{"name": "Mario Sanz", "age": 25}
>>> print(users)
{'john': {'name': 'John Doe',
'age': 30},
'jane': {'name': 'Jane Doe',
'age': 40},
'mario': {'name': 'Mario Sanz',
'age': 25}}
```

Python Dictionaries: Changing Values

The values in the dictionary can be changed by using the keys as indexes.

The `get` method can not be used to accomplish this.

```
>>> addr = {  
...     "name": "Harry Potter",  
...     "street": "Private Drive",  
...     "number": 4,  
...     "city": "Little Whinging",  
...     "county": "Surrey",  
... }  
>>> addr["street"] = "Hogwarts End"  
>>> print(addr["street"])  
Hogwarts End
```


Python Dictionaries: Adding Values

New values can be added to the dictionary the same way they are changed.

If the key does not exist, it will be created.

The **update** method can be used to merge a dictionary into another.

Values will be overwritten or created.

```
>>> addr = {
...     "name": "Harry Potter",
...     "street": "Private Street",
...     "number": 4,
...     "city": "Little Whinging"
... }
>>> addr["country"] = "UK"
>>> print(addr["country"])
UK
>>> fix = {"country": "UK", "continent": "Europe"
...       "street": "Hogwarts End"}
>>> addr.update(fix)
>>> print(addr)
{'name': 'Harry Potter', 'street': 'Hogwarts End',
'number': 4, 'city': 'Little Whinging', 'country':
'UK', 'continent': 'Europe'}
```

Python Dictionaries: Removing Values

The **popitem** method removes and returns the last item in the dictionary. It does not accept any argument.

The **pop** method removes the item with the given key and returns its value. The argument is required.

Notice the **popitem** method returns a tuple containing both the key and the value. The **pop** method only returns the value.

```
>>> addr = {  
...     "name": "Harry Potter",  
...     "street": "Private Drive",  
...     "number": 4,  
...     "city": "Little Whinging",  
...     "county": "Surrey",  
... }  
>>> addr.popitem()  
( 'county', 'Surrey' )  
>>> addr.popitem()  
( 'city', 'Little Whinging' )  
>>> addr.pop("street")  
Private Drive
```

Python Dictionaries: Removing Values

The **clear** method removes all the items in the dictionary.

```
>>> addr = {  
...     "name": "Harry Potter",  
...     "street": "Private Drive",  
...     "number": 4,  
...     "city": "Little Whinging",  
...     "county": "Surrey",  
... }  
>>> addr.clear()  
>>> print(addr)  
{}
```

Python Dictionaries: Other Methods

Dictionaries have three additional methods that are used often.

The method **keys** returns an iterable with all the keys (and no values) of the dictionary.

The method **values** returns an iterable with all the values (and no keys) of the dictionary.

The method **items** returns an iterable with all the keys and values of the dictionary.


```
>>> addr = {  
...     "name": "Harry Potter",  
...     "street": "Hogwarts",  
...     "number": 4,  
...     "city": "Little Whinging"  
... }  
>>> addr.keys()  
dict_keys(['name', 'street', 'num...  
>>> addr.values()  
dict_values(['Harry Potter', 'Hog...  
>>> addr.items()  
dict_items([('name', 'Harry Potte...
```

Comparing Python Dictionaries

Comparing two dictionaries will return **True** if the following statements are true for the elements in it:

- They have the same keys
- For each key the value is the same

Notice the order of the keys in the dictionary is not considered.



```
>>> dict1 = {"a": 1, "b": 2}
>>> dict1 = {"a": 1, "b": 2}
>>> dict1 == dict2
True
>>> dict1 is dict2
False

>>> dict1 == {"b": 2, "a": 1}
True

>>> dict1 == {"a": 2, "b": 1}
False
```

Python Dictionaries: Use Case Examples

USER PROFILE

- First name
- Family name
- Date of birth
- City of residence
- Country of residence
- Sex
- Job title
- Company
- Interests

REGISTRATION

- Student
- Course
- Tutor
- Date of registration
- Passed (Yes/No)
- Date of finalization

ADDRESS

- Type of street
- Street name
- Street number
- Door number
- Postal code
- District
- City
- Country

Python Dictionary Methods: Summary

Dictionaries have the following methods:

Add & Create

- Setdefault
- Update
- Copy
- Fromkeys

Remove

- Pop
- Popitem
- Clear

Access

- Get

Other

- Items
- Keys
- Values

```
>>> print(dir(address))  
[..., 'clear', 'copy', 'fromkeys',  
'get', 'items', 'keys', 'pop',  
'popitem', 'setdefault', 'update',  
'values']
```

We learned ...

- That Python's associative arrays are called dictionaries.
- That dictionaries, as opposed to lists and tuples, use keys instead of indices to refer to each of the values inside them.
- That template dictionaries can be created with the **fromkeys** method using a custom default value.
- That dictionaries have specific methods to return different types of iterables: **keys**, **values** and **items**.
- That two dictionaries are considered the same if they have the same keys and values, even if they are in different order.