Two More Data Structures

Today, we will have a look on two more data structures, `set` s and `dict` ionaries. This completes the overview of Python's basic datastructures.

Sets

A set is not the same as a list. A list is an **ordered** sequence: `[1, 2, 3]`

A set is **unordered**, and is written with curly braces: `{ ... }`

In [1]:

```
print({2, 4, 1, 3})
```

```
{1, 2, 3, 4}
```

In [4]:

```
print({'Call', 'me', 'Ishmael'})
```

```
{'Ishmael', 'me', 'Call'}
```

In [5]:

```
[2, 4] == [4, 2]
```

Out[5]:

```
False
```

In [6]:

```
{2, 4} == {4, 2}
```

Out[6]:

```
True
```

In [7]:

```
type({1, 2, 3, 4})
```

Out[7]:

```
set
```

```
In [8]:
```

```
type([1, 2, 3, 4])
```

```
Out[8]:

list
```

Items in sets are *unique*

An item can only appear *once* in a set:

```
In [9]:
```

```
{2, 2}
```

```
Out[9]:

{2}
```

```
In [2]:
```

```
{'Anton', 'Karla', 'Anton', 'Karla'}
```

```
Out[2]:

{'Anton', 'Karla'}
```

```
In [10]:
```

```
{2, 2, 2, 2, 2, 2, 2, 2} == {2}
```

```
Out[10]:

True
```

Exercise

- Create a set containing the names (strings) `'Anton', 'Karla', 'Anton', 'Karla'`
- Create a list called animals with the following data:

  ```
  animals = ['elephant', 4500, 'rat', 0.2, 'bat', 0.057, 'elephant', 45
  00]
  ```

  - Now, what happens when you convert the list to a set with the function `set`?

    ```
    animals_set = set(animals)
    print(animals_set)
    ```

Why sets?

Sets are useful to

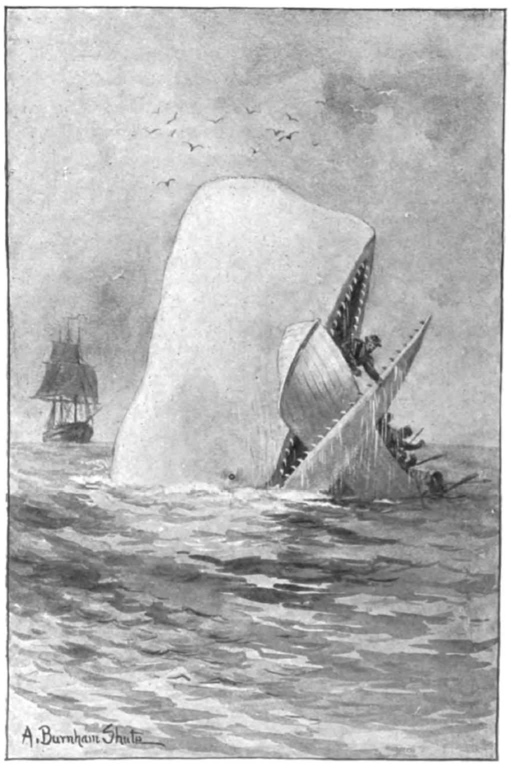
- Check if something exists
    - Names, phone numbers, CPR numbers, etc.
- Check if a set is equal to another set
    - Population sample, bank accounts, etc.

The Dictionary Data Type

The dictionary data type provides a flexible way to access and organize data.

Like a list, a dictionary is a collection of many values. But unlike indexes for lists, indexes for dictionaries can use many different data types, not just integers. Indexes for dictionaries are called keys, and a key with its associated value is called a key-value pair.

In code, a dictionary is typed with braces, `{}` .



" Both jaws, like enormous shears, bit the craft completely in twain."

—*Page 510.*

In [3]:

```python
image = {'color': 'greyscale',
         'size': 289983,
         'type': 'jpg',
         'address': 'https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby

print(image)
```

```
{'color': 'greyscale', 'size': 289983, 'type': 'jpg', 'address': 'ht
tps://upload.wikimedia.org/wikipedia/commons/7/7b/Moby_Dick_p510_ill
ustration.jpg'}
```

This assigns a dictionary to the `image` variable. This dictionary's keys are `'color'` , `'size'` , `'type'` , and `'address'` . The values for these keys are `'greyscale'` , `289983` , `'jpg'` , and `'https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby_Dick_p510_illustration` respectively. You can access these values through their keys.

In [4]:

```python
print(image['color'])
```

```
greyscale
```

In [5]:

```python
print(image['size'])
```

```
289983
```

Note that this is just like a set: you only see each key *once*. If you add the same key, you overwrite it!

In [ ]:

```python
colors_dict = {'color': 'black', 'color': 'red'}
print(colors_dict)
```

Dictionaries can still use integer values as keys, just like lists use integers for indexes, but they do not have to start at `0` and can be any number.

In [7]:

```python
image = {510: 'page in book',
         'color': 'greyscale',
         'size': 289983,
         'type': 'jpg',
         'address': 'https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby
```

In [10]:

```
print(image['type'])
```

```
jpg
```

Dictionaries vs. Lists

Unlike lists, items in dictionaries are unordered. The first item in a list named `values` would be `values[0]`. But there is no "first" item in a dictionary. While the order of items matters for determining whether two lists are the same, it does not matter in what order the key-value pairs are typed in a dictionary.

In [ ]:

```
fst_sentence = ['Call', 'me', 'Ishmael']
fst_sentence_juggled = ['Ishmael', 'me', 'Call']

print(fst_sentence == fst_sentence_juggled=
```

In [11]:

```
fst_sentence = {1: 'Call', 2: 'me', 3: 'Ishmael'}
fst_sentence_juggled = {3: 'Ishmael', 2: 'me', 1: 'Call'}

print(fst_sentence == fst_sentence_juggled)
```

```
True
```

Because dictionaries are not ordered, they cannot be sliced like lists.

In [ ]:

```
fst_sentence = ['Call', 'me', 'Ishmael']
print(fst_sentence[0:2])
```

In [ ]:

```
fst_sentence = {1: 'Call', 2: 'me', 3: 'Ishmael'}
print(fst_sentence[0:2])
```

Accessing Values in a Dictionary

To get the value associated with a key, give the name of the dictionary and then place the key inside a set of square brackets.

Trying to access a key that does not exist in a dictionary will result in a `KeyError` error message, much like a list’s "out-of-range" `IndexError` error message.

In [ ]:

```python
image = {'color': 'greyscale', 'size': 289983, 'type': 'jpg',
         'address': 'https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby

print(image['size'])
```

In [ ]:

```python
print(image['author'])
```

Adding New Key-Value Pairs

Dictionaries are dynamic structures, and you can add new key-value pairs to a dictionary at any time. For example, to add a new key-value pair, you would give the name of the dictionary followed by the new key in square brackets along with the new value.

In [12]:

```python
image = {'color': 'greyscale', 'size': 289983, 'type': 'jpg',
         'address': 'https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby

image['source'] = 'Wikipedia'
print(image)
```

```
{'color': 'greyscale', 'size': 289983, 'type': 'jpg', 'address': 'ht
tps://upload.wikimedia.org/wikipedia/commons/7/7b/Moby_Dick_p510_ill
ustration.jpg', 'source': 'Wikipedia'}
```

Modifying Values in a Dictionary

To modify a value in a dictionary, give the name of the dictionary with the key in square brackets and then the new value you want associated with that key.

In [13]:

```python
image = {'color': 'greyscale', 'size': 289983, 'type': 'jpg',
         'address': 'https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby

image['color'] = 'Black&White'
print(image)
```

```
{'color': 'Black&White', 'size': 289983, 'type': 'jpg', 'address': '
https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby_Dick_p510_i
llustration.jpg'}
```

Removing Key-Value Pairs

When you no longer need a piece of information that's stored in a dictionary, you can use the `del` statement to completely remove a key-value pair. All `del` needs is the name of the dictionary and the key that you want to remove.

In [ ]:

```
del image['color']

print(image)
```

The `keys()`, `values()`, and `items()` Methods

There are three dictionary methods that will return list-like values of the dictionary's keys, values, or both keys and values: `keys()`, `values()`, and `items()`. The values returned by these methods are not true lists: They **cannot be modified and do not have an `append()` method**. But these data types (`dict_keys`, `dict_values`, and `dict_items`, respectively) can be used in for loops.

In [14]:

```
image = {'color': 'greyscale', 'size': 289983, 'type': 'jpg',
         'address': 'https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby

for key in image.keys():
    print(key)
```

```
color
size
type
address
```

In [15]:

```
image = {'color': 'greyscale', 'size': 289983, 'type': 'jpg',
         'address': 'https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby

for value in image.values():
    print(value)
```

```
greyscale
289983
jpg
https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby_Dick_p510_i
llustration.jpg
(https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby_Dick_p510_
illustration.jpg)
```

In [16]:

```
image = {'color': 'greyscale', 'size': 289983, 'type': 'jpg',
         'address': 'https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby

all_my_keys = []
for key, value in image.items():
    all_my_keys.append(key)
    print(key)
    print('\t -' + str(value))
```

```
color
         -greyscale
size
         -289983
type
         -jpg
address
         -https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby_D
ick_p510_illustration.jpg
```

Checking Whether a Key or Value Exists in a Dictionary

Recall from the previous session, that the `in` and `not in` operators can check whether a value exists in a list. You can also use these operators to see whether a certain key or value exists in a dictionary.

In [17]:

```
image = {'color': 'greyscale', 'size': 289983, 'type': 'jpg',
         'address': 'https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby

print('color' in image.keys())
```

```
True
```

In [ ]:

```
print(289983 in image.values())
```

In [ ]:

```
print('compression' not in image.keys())
```

The `get()` Method

It is tedious to check whether a key exists in a dictionary before accessing that key’s value. Fortunately, dictionaries have a `get()` method that takes two arguments: the key of the value to retrieve and a fallback value to return if that key does not exist.

In [18]:

```
image = {'color': 'greyscale', 'size': 289983, 'type': 'jpg',
         'address': 'https://upload.wikimedia.org/wikipedia/commons/7/7b/Moby

color_val = image.get('color', 'unknown')
designer_val = image.get('designer', 'unknown')

print(designer_val)
```

```
unknown
```