OpenTechSchool

Introduction to Data Processing with Python

Data Structures in Python

When working with data, we need ways to store it in variables so we can manipulate it. We will use two new data structures that we didn't cover in Introduction to Programming. These are **lists** and **dictionaries**.

(This chapter contains the same content as the "Data Structures in Python" chapter of the "Websites with Python Flask" workshop.)

If you are already comfortable with lists and dictionaries then you can skip this and move on to the next chapter.

Understanding data structures

Back in the first session we introduced three of the most common data types used in programming: numbers, strings and booleans. We assigned those data types to variables one-by-one, like so:

```
x = 3          # numbers
a = "gorillas" # strings
t = True       # booleans
```

But what if we need something more complicated, like a shopping list? Assigning a variable for every item in the list would makes things very complicated:

```
item_1 = "milk"
item_2 = "cheese"
item_3 = "bread"
```

Lists

Fortunately we don't have to do this. Instead, we have the `list` data type. An empty list is simply `[]`

```
shopping_list = []
```

When you are in the Python interpreter you can see what is inside a list by just typing the name of the list. For example:

```
>>> shopping_list
[]
```

The interpreter shows us that the list is empty.

Now we can add items to `shopping_list`. Try typing the following commands into the Python interpreter.

```
shopping_list.append("milk")
shopping_list.append("cheese")
shopping_list.append("bread")
```

What is in the shopping list? What happens when you append numbers or booleans to the list?

You can also assign a list with some items in it all in a single line, like this:

```
shopping_list = [ "milk", "cheese", "bread" ]
```

To remove an item from the list we use `remove()`:

```
shopping_list.remove("milk")
```

Lists can easily be processed in a `for` loop. Have a look at this example which prints each item of the list in a new row:

```
for item in shopping_list:
    print(item)
```

And that's it! Python also makes it really easy to check if something is in a list or not:

```
if "milk" in shopping_list:
    print("Delicious!")

if "eggs" not in shopping_list:
    print("Well we can't have that!")
    shopping_list.append("eggs")
```

Lists are the most common data structure in programming. There are lots of other things you can do with lists, and all languages have their own subtly different interpretation. But fundamentally they are all very similar.

In summary:

```
shopping_list = []
shopping_list.append("cookies")
shopping_list.remove("cookies")
```

Dictionaries

The other main data type is the dictionary. The dictionary allows you to associate one piece of data (a "key") with another (a "value"). The analogy comes from real-life dictionaries, where we associate a word (the "key") with its meaning. It's a little harder to understand than a list, but Python makes them very easy to deal with.

You can create a dictionary with `{}`

```
foods = {}
```

And you can add items to the dictionary like this:

```
foods["banana"] = "A delicious and tasty treat!"
foods["dirt"]   = "Not delicious. Not tasty. DO NOT EAT!"
```

The keys in this example are "banana" and "dirt", and the values are the things that we assign to them. You can use any data type that won't change as a dictionary key. Check it out by using a number, a boolean value, and a list as keys in a dictionary. What does this say about strings?

As with lists, you can always see what is inside a dictionary:

```
>>> foods
{'banana': 'A delicious and tasty treat!', 'dirt': 'Not delicious. Not tasty. DO NOT EAT!'}
```

You can look up any entry in the dictionary by its key:

```
>>> foods["banana"]
'A delicious and tasty treat!'
```

If the key isn't found in the dictionary, a `KeyError` occurs:

```
>>> foods["cheese"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'cheese'
```

For this reason, you can test whether a key is in the dictionary or not, by using the keyword `in`:

```
if "cheese" in foods:
    print("Cheese is one of the known foods!")
    print(foods["cheese"])
```

`not in` works as well, just like with lists.

You can delete from a dictionary as well. We don't really need to include an entry for dirt:

```
del foods["dirt"]
```

What makes dictionaries so useful is that we can give meaning to the items within them. A list is just a bag of things, but a dictionary is a specific mapping of something to something else. By combining lists and dictionaries you can describe basically any data structure used in computing.

For example, you can easily add a list to a dictionary:

```
ingredients = {}
ingredients["blt sandwich"] = ["bread", "lettuce", "tomato", "bacon"]
```

Or add dictionaries to lists:

```
europe = []
germany = {"name": "Germany", "population": 81000000}
europe.append(germany)
luxembourg = {"name": "Luxembourg", "population": 512000}
europe.append(luxembourg)
```

Outside of Python, dictionaries are often called `hash tables`, `hash maps` or just `maps`.

Next Chapter

Once you're comfortable with lists and dictionaries it's time for the next chapter, Introducing IPython Notebook