03_py_fundementals_more_datatypes

August 12, 2021

1 More Python Data structures

We've seen a couple of python data types, like integers and strings. There are lots and lots of data types.

There are also a lot of structures for how you hold a collection or group of data. We will encounter increasingly powerful data structures throughout the course. These more complext data structure will come with built in methods, attrtibutes and functions that let us do complex data analysis.

Let's start here with two of the core basic data structures in python: `lists` and `dictionaries`.

These two will be used over and over again, so we want to understand them

1.1 Credit:

things here are a mix of the really excellent Software Carpentry tutorial on Python: http://swcarpentry.github.io/python-novice-inflammation/ And Ryan Abernathys open book https://earth-env-data-science.github.io/lectures/core_python/python_fundamentals.html

I've made some slight adaptations here and there, but the credit goes to those organizations. I hope I am using this correctly under the licences:

https://earth-env-data-science.github.io/LICENSE.html https://swcarpentry.github.io/python-novice-inflammation/LICENSE.html

2 `lists`

Similar to a string that can contain many characters, a list is a container that can store many values.

It is fundemental to python and is built into the core. We create a list by putting values inside square brackets and separating the values with commas:

[1]:
```python
odds = [1, 3, 5, 7]
print('odds are:', odds)
```

```
odds are: [1, 3, 5, 7]
```

as we saw with strings, we can access elements of a list using indices – numbered positions of elements in the list. These positions are numbered starting at 0, so the first element has an index of 0.

[2]:
```
print('first element:', odds[0])
print('last element:', odds[3])
print('"-1" element:', odds[-1])
```

```
first element: 1
last element: 7
"-1" element: 7
```

Yes, we can use negative numbers as indices in Python. When we do so, the index `-1` gives us the last element in the list, `-2` the second to last, and so on. Because of this, `odds[3]` and `odds[-1]` point to the same element here.

2.1 lists can hold many types of data

[3]:
```
l = ['dog', 'cat', 'fish']
type(l)
```

[3]:
```
list
```

you can even mix and match data types in the same list:

[4]:
```
mixed = [764, 'cat', 'fish']

print(mixed)
```

```
[764, 'cat', 'fish']
```

`lists` are code ‘objects’, which again have `attributes` and `methods`

these are features of the code object that allow you to do stuff with it

they can be accessed via the syntax `variable.method`

`lists` have lots of methods

[5]:
```
# .sort() is a list method you can use
# in this case the list is sorred alphabetically

l.sort()
l
```

[5]:
```
['cat', 'dog', 'fish']
```

There are many different ways to interact with lists. Exploring them is part of the fun of python.

list.append(x) Add an item to the end of the list. Equivalent to a[len(a):] = [x].

list.extend(L) Extend the list by appending all the items in the given list. Equivalent to a[len(a):] = L.

list.insert(i, x) Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

list.remove(x) Remove the first item from the list whose value is x. It is an error if there is no such item.

list.pop([i]) Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the i in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

list.clear() Remove all items from the list. Equivalent to del a[:].

list.index(x) Return the index in the list of the first item whose value is x. It is an error if there is no such item.

list.count(x) Return the number of times x appears in the list.

list.sort() Sort the items of the list in place.

list.reverse() Reverse the elements of the list in place.

list.copy() Return a shallow copy of the list. Equivalent to a[:].

2.1.1 looping on list elements

lists are collections of stuff, so we can use our `for` loop structure to do something to the elements of a list.

remember the `for` loop is something like this:

```
for element in collection:
    do things using element
```

[6]:
```
for animal in l:
    print('the animal is: ', animal.capitalize())
```

```
the animal is:  Cat
the animal is:  Dog
the animal is:  Fish
```

2.2 Other Data Structures

We are almost there. We have the building blocks we need to do basic programming. But python has some other data structures we need to learn about.

2.3 `Dictionaries`

`Dictionaries` are extremely useful data structures. It maps **`keys`** to **`values`**.

as we go further on in the course it will become clear just why these are so great. But in essence they allow you to hold a collection of data (`values`) which are associated with some labels (`keys`). It turns out this idea is super powerful and O promise will make you very happy at some point!

`dictionaries` are a bit like lists, but with this extra feature that every element in there has a label. We create them with curly brackets,`{}`, and `:`, with the general form:

```
dict = {'key1':'value1', 'key2':'value2', ...}
```

```
[7]: # different ways to create dictionaries
     d = {'name': 'Nick', 'age': 36}
     d
```

```
[7]: {'name': 'Nick', 'age': 36}
```

we access a value from a dictionary by useing square brackets,`[]`, and the `key`:

```
[8]: d['name']
```

```
[8]: 'Nick'
```

2.3.1 note: Square brackets `[...]` are python for "get item" in many different contexts.

```
[9]: # try to access a non-existant key
     d['height']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-9-76eb79130058> in <module>
      1 # try to access a non-existant key
----> 2 d['height']

KeyError: 'height'
```

```
[10]: # add a new key and value:

      d['height'] = 6

      d
```

```
[10]: {'name': 'Nick', 'age': 36, 'height': 6}
```

keys and values can be almost any datatype:

```
[11]: # add a list of grades to the dictionary

      d['grades'] = ['A', 'B', 'C', 'A', 'B+']

      d
```

```
[11]: {'name': 'Nick', 'age': 36, 'height': 6, 'grades': ['A', 'B', 'C', 'A', 'B+']}
```

```
[12]: d['grades']
```

```
[12]: ['A', 'B', 'C', 'A', 'B+']
```

Dictionaries also have lots of methods and attribues. for example you can just list the `keys` with the command `mydict.keys()`:

```
[13]: d.keys()
```

```
[13]: dict_keys(['name', 'age', 'height', 'grades'])
```

The end…

2.4 Breakout/Exercise 03

Prompt: Can you think of some uses of these data structures?

```
[14]: #excercise answer:
      my_list = []
      for char in 'hello':
          my_list.append(char)
      print(my_list)
```

```
['h', 'e', 'l', 'l', 'o']
```