Python for R Users in the Social Sciences: Lesson 3

Aaron Erlich

March 17, 2013

1 Dictionaries

Dictionaries look very different than the storage logic of an R data matrix, but they share something in common. In R if you have a data frame and you want to access a specific variable you can find it by using the `$` sign. Dictionaries is a type of data that store data has a key, value pair. The key is kind of like a variable name and the value is the data value. A value can be a string, a list, another dictionary (nested) !, or a data type we are soon going to meet: the tuple.

We learned that lists are mutable. Dictionaries are also mutable. While lists are defined by Python with the square brackets `[ ]`, dictionaries are defined by the curly braces `{ }`. We can access a dictionary's values by using the square brackets and the name of the key values. We can also access all the keys by using the `keys()` method and the values with the `values()` method.

Let's look at a very small data frame in R.

```
> DOI <- c(1963, 1776)
> pop <- c(4e+07, 3e+08)
> subnat <- c("Kilifi, Nairobi", "WA, ME")
> countryMatrix <- matrix(c(DOI, pop, subnat), nrow = 2, ncol = 3,
+     dimnames = list(c("USA", "KEN"), c("DOI", "pop", "subnat")))
> print(countryMatrix)

    DOI    pop     subnat
USA "1963" "4e+07" "Kilifi, Nairobi"
KEN "1776" "3e+08" "WA, ME"
```

We can see immediately that the typical matrix for data with multiple non-orthogonal dimensions is going to be difficult to deal with in an R matrix. We could add an extra column for each subnational unit. But that would lead to a lot of NAs in the data set. Let's see how the Python dictionary would handle this.

```
>>> KEN = {'DOI': 1963, 'pop': 40000000, "subnat": ['Kilifi', 'Nairobi']}
>>> print(KEN) #order not maintained
{'DOI': 1963, 'subnat': ['Kilifi', 'Nairobi'], 'pop': 40000000}
>>> print(KEN['pop'])
40000000
>>> print(KEN.keys())
```

```
['DOI', 'subnat', 'pop']
>>> print(KEN.values())
[1963, ['Kilifi', 'Nairobi'], 40000000]
```

Like lists we can also add an element into our dictionary or add to nested objects. We do this but using brackets. To reiterate, the brackets reference a key in a dictionary. This use of brackets has no relationship to a list.

```
>>> USA = {'DOI': 1776, 'pop': 300000000, "subnat": ['WA', 'ME']}
>>> USA['flagname'] = "Stars and Strips" #add an element with the key "flagname"
>>> USA['subnat'].append('OR')
>>> print(USA['flagname'], USA['subnat'])
('Stars and Strips', ['WA', 'ME', 'OR'])
```

We could also put our two dicionaries into a list. Then 'subnat' would be a list in a dictionary in a list. There are often quite deeply nested sets of dictionaries and lists in Python. Let's calculate the number of years from independence in our two countries.

```
>>> USA = {'DOI': 1776, 'pop': 300000000, "subnat": ['WA', 'ME']}
>>> KEN = {'DOI': 1963, 'pop': 40000000, "subnat": ['Kilifi', 'Nairobi']}
>>> countries = [USA, KEN]
>>> [2013-country['DOI'] for country in countries]
[237, 50]
```

As you can see this approach is a fundamentally different way of storing data than we would store in R. It has many nice properties, however, for data analysis we will want to return to our more commonly used structures. However, in the process of organizing our munging our data, we will definitely come across and use dictionaries.

2 Tuples

What the heck is a tuple? And how do you pronounce it? Well, there seems to be some variation but pronounce the words "two" and "pull" and you will have the basic idea. The basic idea behind tuples is that they are immutable lists. And how does Python recognize they are tuples? By putting them in normal parentheses—like ( ) Besides potentially encountering tuples when you call some function, you will want tuples if you are going to pass data around for different functions to access and you want to make sure it doesn't get changed int the process

```
>>> countryCodes = ('RUS', 'AFG', 'GER')
>>> type(countryCodes)
<type 'tuple'>
>>> countryCodes.append('GEO')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
```

Here we see that we cannot append anything to our tuple because it is immutable. Tuples probably will not come up immediately, but keep them in the back of your head.