**Dictionaries, Part 1**

Weeeeeelcome back. Now do I have a treat for you. In this video, I'll be talking about a new Python type: the dictionary. Dictionaries are sooooo useful in the data world and to see this, imagine the following: you work for the World Bank and want to keep track of the population in each country.

**List**

You can put the populations in a list. You start with Afghanistan, 30.55 million, Albania, 2.77 million, Algeria, around 40 million, and so on. To keep track about which population belongs to which country, you can create a second list, with the names of the countries in the same order as the populations. Now suppose that want to get the population of Albania. First, you have to figure out where in the list Albania is, so that you can use this position to get the correct population. Remember about the method index(), a list method you learned about in the Intro to Python course? Let's use it to get the index of Albania in countries, like this. Now, we can use this index to subset the pop list, to get the population corresponding to Albania. As expected, we get 2.77, the population of Albania in millions. So we built two lists, and used the index to connect corresponding elements in both lists. It worked, but it's a pretty terrible approach: it's not convenient and not intuitive. Wouldn't it be easier if we had a way to connect each country directly to its population, without using an index? This is where the dictionary comes into play.



**Dictionary**

Let's convert this population data to a dictionary. I included the lists to start from on the top here, so you can see what's going on. To create the dictionary, you need curly brackets. Next, inside the curly brackets, you have a bunch of what are called key:value pairs. In our case, the keys are the country names, and the values are the corresponding populations.

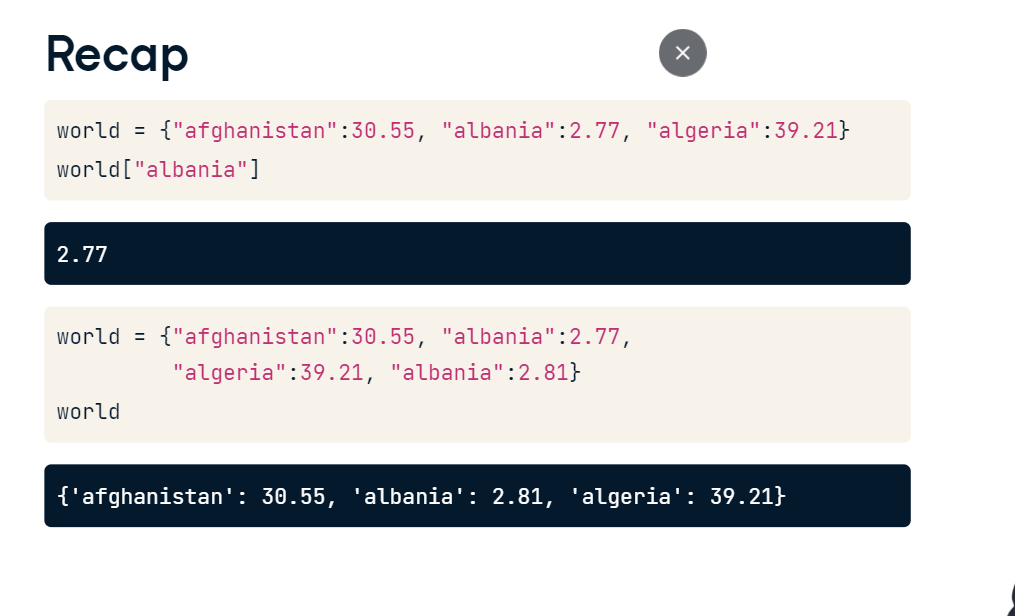
**Dictionary**

The first key is Afghanistan, and its corresponding value is 30.55. Notice the colon that separates the key and value here. Let's do the same thing for the two other key-value pairs, and store the dictionary under the name world.

If you know want to find the population for Albania, you simply type world, and then the string Albania inside square brackets. In other words, you pass the key in square brackets, and you get the corresponding value. The key opens the door to the value: pretty poetic, isn't it? This approach is not only intuitive, it's also very efficient, because Python can make the lookup of these keys very fast, even for huge dictionaries.



**Recap**

we created the dictionary "world", which basically is a set of key value pairs. You could easily access the population of Albania, by passing the key in square brackets, like this. For this lookup to work properly, the keys in a dictionary should be unique. If you try to add another key:value pair to world with the same key, Albania, for example, you'll see that the resulting world dictionary still contains three pairs. The last pair that you specified in the curly brackets was kept in the resulting dictionary. 

Also, these unique keys in a dictionary should be so-called immutable objects. Basically, the content of immutable objects cannot be changed after they're created. Strings, booleans, integers and floats are immutable objects, but the list for example is mutable, because you can change its contents after it's created. That's why this dictionary, that has all immutable objects as keys, is perfectly valid. This one, however, that uses a list as a key, is not valid, so we get an error. So now that you have an idea of how to build a valid dictionary and how to access it using square brackets, let's see how we can add more data to a dictionary that already exists.



**Principality of Sealand**

Say for example that you, an employee of the World Bank, decide to acknowledge the Principality of Sealand. Sealand is an unrecognized micronation, on an offshore platform located in the North Sea. At the moment, it has 27 inhabitants.

1. 1 Source: Wikipedia

**Dictionary**

To add this information, simply write the key sealand in square brackets and assign 27 expressed in millions to it with the equals sign. If you check out "world" again, indeed, sealand is in there. To check this with code, you can also write "sealand in world", which gives you True if the key sealand is in there.



With the same syntax, you can also change values, for example, to update the population of sealand to 28. Because each key in a dictionary is unique, Python knows that you're not trying to create a new pair, but want to update the pair that's already in there. You can see this from the printout here. Suppose now that your boss didn't see the humour of adding Sealand to the list, and asks you to remove it again. You can do this with del, again pointing to sealand inside square brackets. If you print world again, Sealand is no longer in there. Good riddance!



**List vs. Dictionary**

If you remember the discussion of lists from the intro course, you'll notice that using lists

and dictionaries is pretty similar. You can select, update and remove elements with square brackets.

**List vs. Dictionary**

There are some big differences though. The list is a sequence of values

that are indexed by a range of numbers. The dictionary, on the other hand,is indexed by unique keys, that can be any immutable type. When to use which one, I hear you ask? Well, if you have a collection of values where the order matters, and you want to easily select entire subsets of data, you'll want to go with a list. If, on the other hand, you need some sort of look up table, where looking for data should be fast and where you can specify unique keys, a dictionary is the preferred option.

