A (debatably) Brief Introduction to Python

CHAPTER 2: STRINGS

In the previous chapter, I promised that I would dedicate a whole chapter to strings, so here I am, making a chapter about strings!

As previously mentioned, strings are one of python's primitive data types, however they are too complicated to be primitive in other languages (again: for complicated reasons). Since we will be honing in on natural language processing later on in the school year, strings will become increasingly important, so pay great attention. The main reason why I'm dedicating a whole chapter to these beautiful things is because they have so many built-in functions associated with them. Without further ado, here they are (note that `<string>` references any arbitrary string, like `"foo"`, or `"Vedant is a great guy!"`):

- `<string>.split(delim)` This function will take every substring (smaller string) in `<string>` which is separated by `delim` and put them in a list. What is a list? Forget about that for now, we'll go over that in chapter 3. Let's look at `"Foo bar baz!!".split(" ")`. In this case, `<string>` is `"Foo bar baz!!"`, and `delim` is `" "`. This function will return `["Foo", "bar", "baz!!"]`. Notice how the space is gone. Let's take another example. `"thisASDFisASDFaASDFstring".split("ASDF")` will return `["this", "is", "a", "string"]`. Still fuzzy? Ask me personally or ask your fellow club members in the questions slack channel.
- `<string1> + <string2>` This function concatenates (smushes together) two strings. For example, `"abra" + "cadabra"` returns `"abracadabra"`.
- `<string>.format(…)`. Let me explain. This is the easiest way to put non-strings inside strings. For example, `"This is chapter {} of intro to python".format(2)` returns `"This is chapter 2 of intro to python"`. The best part is, you can put literally anything inside a string. Awesome. In order to put more things inside strings, you would do this: `"{} {} {} {}".format(1, 2, 3, 4)`. This would return `"1 2 3 4"`.
- `<string>.replace(sub1, sub2)` This function replaces every instance of `sub1` in `<string>` with `sub2`. For instance, `"preface".replace("ace", "ix")` returns `"prefix"`. This is probably the least complicated of all the string methods.
- `len(<string>)` This returns the length of `<string>`.

On top of all these nifty methods, strings can be **indexed**. What does that mean? Simple: You can isolate certain substrings from a string without actually removing them. Say you wanted to get the very first character of `"hello"`, which is stored inside `str`. In order to do that, you would index hello as such: `str[0]`. Here's a good question that many beginning programmers ask: why does indexing start with 0? Answer: I have absolutely no clue. One more weird thing. Say you wanted to get the very last character in `str`. Let's break this down: there are 5 characters in hello. Could we get the last character using `str[5]`? **ABSOLUTELY NOT**! Indexing ends at the length of the string minus 1. In this case, we would use `str[4]`. How do we get multiple characters from a string using indexing? Here's how:

Say we wanted to get "ell" from "hello". In order to do that, you would have to do `str[1:4]`. Basically, `str[a:b]` gives you `str[a]`, `str[a+1]`, all the way to `str[b]`. Three more things:

- `<string>.index(substring)` gives you the index of the **first occurrence** of the first character of `substring` within `<string>`
- `<string>[a:]` gives you every character in `<string>` from `a` onwards.
- `<string>[:a]` gives you every character in `<string>` up until (but not including) `a`.
- `<substring> in <string>` returns a **boolean value** (`True` or `False`) if `<substring>` is inside `<string>`.

We'l revisit strings much later on whe we discuss sentiment, stopwords, and the `nltk` library.