Strings and manipulating them

October 10, 2019

Strings are lists of characters

In Python, *most* of what you can do with lists, you can also do with strings. That means you can

- Index into a string
- Use the in keyword to find out if a string contains another string

Changing the Case

WE CAN CHANGE THE CASE OF A STRING BY MAKING A CALL TO upper() or lower() METHODS ON THAT STRING

```
>>>'WHY ARE WE YELLING?'.lower()
'why are we yelling?'
```

Why is this useful?

Concatenation

You can "add" strings by putting a plus sign (+) between them. This is called concatenation, and it does NOT add a space between the two strings.

```
>>>cat = "Jules"
>>>print("The cat is " + cat)
The cat is Jules
```

Splicing and dicing

Splicing is a new way of accessing elements of a string or list.

It looks like a normal indexing operation, except you specify two indices, one for the starting point, and one for the ending point.

```
>>>'Blame it on the goose'[3:10]
'me it o'
```

Let's try some stuff out with lists and strings in the interpreter!

Beginning and the end

If you leave one of the numbers out of the splice, Python goes to the beginning or end:

```
>>>"Let's take it from the top"[:10]
"Let's take"

>>>"End of the line, cowboy"[11:]
'line, cowboy'
```

Using Escape Characters

What do we do if we want to have a double quote in a string in our code?

We could use single quotes for the string, but what if we want to have single AND double quotes in the same string?

Escaping to the rescue!

Backslashes

Put a backslash (the one above the enter button) in front of a character to make Python treat it specially. This is called "escaping" because that character is escaping the usual string processing.

```
print("This how we \"quote\" stuff")
This how we "quote" stuff
```

Uh oh. What if I want a slash in a string? Let's try it!

Common escape sequences

Single Quote	`\'`
Double Quote	`\"`
Backslash	`\\`
Tab	`\t`
New line	`\n`

Remember! Python treats escaped characters as a single character!

Another end

The print function usually puts in a newline automatically for you. Sometimes you won't want that. If you don't, you can supply an alternative ending character with the end argument:

```
print("This is", end="")
print("fun")
```

```
This is fun
```

Whitespace

Whitespace is the term for all of the characters that represent some kind of spacing in a string. This includes (but isn't limited to!):

- Space
- New line
- Tabs

Strip() it for parts

The string method .strip() removes ALL the whitespace at the start and end of a string:

```
>>>"  here we go!   \n".strip()
'here we go!'
```

This doesn't immediately seem useful until you're working on a problem and you have a bunch of annoying whitespace at the start and end of a string.

Divide and conquer

.split() lets you get a list of items from a string, where each item was separated by a certain character in the original string. If you give it no arguments, it uses ' '

It makes way more sense when you see it…we'll try examples.

```
>>>'I have misgivings about this method'.split()
['I', 'have', 'misgivings', 'about', 'this', 'method']
```

Join does the opposite

Call .join() on a string that you want to be the "glue" between the items of the list that you sent to join.

```
'\t'.join('You rock, guys'.split())
'You\trock,\tguys'
```

Let's do more examples!