Data Scientist
Codecademy
Python Fundamentals
Strings

Introduction to Strings

A string is a sequence of characters contained within a pair of ‘single quotes’ or “double quotes”. A string can be any length and can contain any letters, numbers, symbols, and spaces.

They’re all Lists!

A string can be thought of as a list of characters.

Each character in a string has an index starting from 0.

Cut Me a Slice of String

We can select substrings/ slice using the following syntax:

```
string_name[first_index : last_index]
```

The character at the last_index is not included in the new string formed.

String slicing works similarly to list slicing.

Concatenating Strings

We can create new string by concatenating them as:

```
new_string = string1 + string2
```

No spaces are added between the strings being added.

How long is that String?

len() function returns the number of characters in a string.

Characters also include whitespaces hence they are counted too.

Negative Indices

Negative Indices count backward from the end of the string.

Strings are Immutable

Strings are immutable. This means that we cannot change a string once it is created. We can use it to create other strings, but we cannot change the string itself.

Escape Characters

By adding a backslash \ in front of the special character we want to escape we can include it in a string.

Iterating through Strings

We can iterate through strings using for or while loops.

Strings and Conditionals

in function

```
letter in word
```

will return True if that letter exists in the word or else False.

This method not only works with letters it works with substrings as well.

Strings Methods

String Methods: 'Hello World'

```
>>> 'Hello world'.upper()
'HELLO WORLD'

>>> 'Hello world'.lower()
'hello world'

>>> 'Hello world'.title()
'Hello World'

>>> 'Hello world'.split()
['Hello', 'world']

>>> ' '.join(['Hello', 'world'])
'Hello world'

>>> 'Hello world'.replace('H', 'J')
'Jello world'

>>> '   Hello world   '.strip()
'Hello world'

>>> "{} {}".format("Hello", "world")
'Hello world'
```

String methods have the same syntax:

```
string_name.string_method(arguments)
```

Formatting Methods

There are three string methods that can change the casing of a string.

- .lower() returns the string with all lowercase characters
- .upper() returns the string with all lowercase characters
- .title() returns the string in title case, which means the first letter of each word is capitalized.

Splitting Strings

.split() is performed on a string, takes one argument, and returns a list of substrings found between the given argument (delimiter).

By default delimiter is set to whitespaces, so the string will split at spaces.

.split returns a list with string splitted at every occurrence of delimiter. The delimiter is not included in the substrings of the list.

If the delimiter is present at the last index of the string then the returned list will contain an empty string at the end of list in addition to the splitted strings.

The delimiter can be a character, string, or an escape sequence.

Joining Strings

.join() is opposite of .split(), it joins a list of strings together with a given delimiter.

```
'delimiter'.join(list_you_want_to_join)
```

Delimiter is generally whitespace but could be any character or string.

.strip()

Stripping a string removes all whitespace characters from the beginning and end.

We can also use .strip using a character argument, which will strip that character from either end of the string.

.replace()

.replace() takes two arguments and replaces all instances of the first argument in a string with the second argument.

```
string_name.replace(substring_being_replaced, new_substring)
```

.find()

.find() takes a string as an argument and searching the string it was run on for that string. It then returns the first index value where that string is located.

.format()

.format() takes variables as an argument and includes them in the string that it is run on. You include {} marks as placeholders for where those variables will be imported.

.format() can be made even more legible for other people reading code by including keywords. The keywords are given inside {} and while passing the arguments you do (keyword = variable/whatever is supposed to replace it)