SUMMER SCHOOL 2021

Introduction to Python Programming

String Indexing and Manipulation

Binjamin Barsch

Summary of commands used in this lecture

Command	Meaning
print()	A Python function that prints text
type()	A Python function that displays the data type of a variable
split()	A Python function that splits a string into list by a separator
startswith()	A Python function that searches for text at the begging of a string
endswith()	A Python function that searches for text at the end of a string

String Indexing & Manipulation

Understanding **strings** and how to manipulate them is a crucial part to understanding Python programming. There are a lot of core principles that lead off from understanding and manipulating **strings**. There are also a lot of similarities between **strings** and **lists**!

Lets first look at a few important concepts. Strings in Python are either surrounded by single quotation marks or double quotation marks. And as we saw before, we can assign it to a variable:

```
>>>
>>> string_one = "Summer School"
>>> string_one
'Summer School'
>>> string_two = '2021'
>>> string_two
```

```
'2021'
>>>
```

In other programming languages single characters like "**`b`**" or "**`4`**" are a unique data type on its own and often referred to as a char or character data type. In Python, it is still considered a **string** just of length 1. And just like we accessed items in a **list** through their index positions, we can also access single or multiple characters of a **string** using their associated index:

```
>>> string_two[0]
'2'
>>> string_one[2]
'm'
>>> string_one[:]
'Summer School'
>>> string_one[1:4]
'umme'
>>> string_one[-1]
'l'
>>>
```

This should all be familiar to you as you have seen it with **lists**. The second last command we have seen in **lists** where we only want a certain range of the string, this is known as **string** slicing. The only new way of accessing strings, which applies to lists too, is the last command where we used "**`[-1]`**". This is a cool feature with Python where "**`[-1]`**" allows you to access the very last value of the **string**. One last feature is that to join or concatenate more than one **strings**. And we do this quite simply:

```
>>> string_three = string_one + string_two
>>> string_three
'Summer School2021'
>>> string_three = string_one + " " + string_two
>>> string_three
'Summer School 2021'
>>> print(len(string_three))
18
>>>
```

We can also use the function "**`len()`**" to get the length of a string as well as seen above. How do we mine or filter certain data. Let us look at an example below:

```
>>> string_three = 'welcome, Summer School 2021'
>>> string_three
'welcome, Summer School 2021'
>>> print(string_three.split(","))
['welcome', ' Summer School 2021']
>>> split_string = string_three.split(",")
>>> split_string
```

```
['welcome', ' Summer School 2021']
>>> type(split_string)
<class 'list'>
>>>
```

We use the "**`split()`**" function and we specify that our criteria to split is based on a comma. Interestingly the split function outputs a list! There is so much more to strings but this will do for now.

There are also things called escape characters that are otherwise impossible to put into a string. Like "**`\t`**" or Tab or "**`\n`**" for newline:

```
>>> print("Hello there!\nWelcome to the Summer School\nHow are you?")
Hello there!
Welcome to the Summer School
How are you?
>>>
```

Lastly, there are useful string features to check certain words or letters that are in a string:

```
>>> 'Hello' in 'Hello, World'
True
>>> 'Hello, world!'.startswith('Hello')
True
>>> 'Hello, world!'.endswith('world!')
True
>>> 'abc123'.startswith('abcdef')
False
>>> 'abc123'.endswith('12')
False
```

Summary

This is the end of lesson 4 where we covered a few concepts about string indexing and manipulation. On to lesson 5!
