Previous Tutorial (/en/String_Formatting)

Next Tutorial (/en/Conditions)

Basic String Operations

Strings are bits of text. They can be defined as anything between quotes:

script.py ()

```
1 astring = "Hello world!"
2 astring2 = 'Hello world!'
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Run

Powered by DataCamp (https://www.datacamp.com)

As you can see, the first thing you learned was printing a simple sentence. This sentence was stored by Python as a string. However, instead of immediately printing strings out, we will explore the various things you can do to them. You can also use single quotes to assing a string. However, you will face problems if the value to be assigned itself contains single quotes.For example to assign the string in these bracket(single quotes are ' ') you need to use double quotes only like this

script.py ()

```
1 astring = "Hello world!"
2 print("single quotes are ' '")
3
4 print(len(astring))
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Run

Powered by DataCamp (https://www.datacamp.com)

That prints out 12, because "Hello world!" is 12 characters long, including punctuation and spaces.

script.py ()

```
1 astring = "Hello world!"
2 print(astring.index("o"))
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Run

Powered by DataCamp (https://www.datacamp.com)

That prints out 4, because the location of the first occurrence of the letter "o" is 4 characters away from the first character. Notice how there are actually two o's in the phrase - this method only recognizes the first.

But why didn't it print out 5? Isn't "o" the fifth character in the string? To make things more simple, Python (and most other programming languages) start things at 0 instead of 1. So the index of "o" is 4

script.py ()

```
1 astring = "Hello world!"
2 print(astring.count("l"))
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Run

Powered by DataCamp (https://www.datacamp.com)

For those of you using silly fonts, that is a lowercase L, not a number one. This counts the number of l's in the string. Therefore, it should print 3.

script.py ()

```
1 astring = "Hello world!"
2 print(astring[3:7])
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Run

Powered by DataCamp (https://www.datacamp.com)

This prints a slice of the string, starting at index 3, and ending at index 6. But why 6 and not 7? Again, most programming languages do this - it makes doing math inside those brackets easier.

If you just have one number in the brackets, it will give you the single character at that index. If you leave out the first number but keep the colon, it will give you a slice from the start to the number you left in. If you leave out the second number, if will give you a slice from the first number to the end.

You can even put negative numbers inside the brackets. They are an easy way of starting at the end of the string instead of the beginning. This way, -3 means "3rd character from the end".

script.py ()

```
1 astring = "Hello world!"
2 print(astring[3:7:2])
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Run

Powered by DataCamp (https://www.datacamp.com)

This prints the characters of string from 3 to 7 skipping one character. This is extended slice syntax. The general form is [start:stop:step].

script.py ()

```
1 astring = "Hello world!"
2 print(astring[3:7])
3 print(astring[3:7:1])
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Run

Powered by DataCamp (https://www.datacamp.com)

Note that both of them produce same output

There is no function like strrev in C to reverse a string. But with the above mentioned type of slice syntax you can easily reverse a string like this

script.py ()

```
1 astring = "Hello world!"
2 print(astring[::-1])
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Run

Powered by DataCamp (https://www.datacamp.com)

This

script.py ()

```
1 astring = "Hello world!"
2 print(astring.upper())
3 print(astring.lower())
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Run

Powered by DataCamp (https://www.datacamp.com)

These make a new string with all letters converted to uppercase and lowercase, respectively.

script.py ()

```
1  astring = "Hello world!"
2  print(astring.startswith("Hello"))
3  print(astring.endswith("asdfasdfasdf"))
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Run

Powered by DataCamp (https://www.datacamp.com)

This is used to determine whether the string starts with something or ends with something, respectively. The first one will print True, as the string starts with "Hello". The second one will print False, as the string certainly does not end with "asdfasdfasdf".

script.py ()

```
1  astring = "Hello world!"
2  afewwords = astring.split(" ")
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Run

Powered by DataCamp (https://www.datacamp.com)

This splits the string into a bunch of strings grouped together in a list. Since this example splits at a space, the first item in the list will be "Hello", and the second will be "world!".

Exercise

Try to fix the code to print out the correct information by changing the string.

script.py ()

```
1   s = "Hey there! what should this string be?"
2   # Length should be 20
3   print("Length of s = %d" % len(s))
4
5   # First occurrence of "a" should be at index 8
6   print("The first occurrence of the letter a = %d" % s
    .index("a"))
7
8   # Number of a's should be 2
9   print("a occurs %d times" % s.count("a"))
10
11  # Slicing the string into bits
12  print("The first five characters are '%s'" % s[:5]) #
    Start to 5
13  print("The next five characters are '%s'" % s[5:10]) # 5
    to 10
14  print("The thirteenth character is '%s'" % s[12]) # Just
    number 12
15  print("The characters with odd index are '%s'" %s[1::2])
    #(0-based indexing)
```

IPython Shell ()

```
In [1]:
```

(https://github.com/datacamp/datacamp-light)

Solution **Run**

Powered by DataCamp (https://www.datacamp.com)

Previous Tutorial (/en/String_Formatting) Next Tutorial (/en/Conditions)