Python More on Strings

We talked about strings when we talked about lists and slicing, and you've seen ways to break up strings through use of the split function, and through direct indexing. Strings in Python can be a bit frustrating too, and I want to share some more details of how strings are handled, so that you're aware of them.

In Python 3 strings are Unicode based. In early computing characters of strings were limited to one of 256 different values. This was enough to get all of the upper or lower case Latin characters, as well as single digit numbers represented. This language was called ASCII and was fairly compact. But the world doesn't just run on Latin characters and there's a need to support non-English languages as well as characters which are not commonly used in words, but are commonly used elsewhere like mathematical operators.

The Unicode Transformation Format, or UTF, is an attempt to solve this. It can be used to represent over a million different characters. This includes not only human languages like you might expect, but symbols like emojis too.

Python 3 uses Unicode by default so there is no problem in dealing with international character sets.

In addition to Unicode, Python uses a special language for formatting the output of strings. One of the challenges with dynamic typing is that it's bit unclear when you have to do type conversion yourself. We saw on the last lecture that if we wanted to print out a name and a number that we can't use concatenation without calling the str function to convert the number to a string first. This creates a lot of nasty looking code where every operator you're looking to concatenate is wrapped in this str function. The Python string formatting mini language allows you to write a string statement indicating placeholders for variables to be evaluated. You then pass these variables in either named or in order arguments, and Python handles the string manipulation for you.

Here's an example. Imagine we have purchase order details and a dictionary, which includes a number of items, a price, and a person's name.

We can write a sales statement string which includes these items using curly brackets.

The Python Programming Language: More on Strings

In [ ]:
```
print('Chris' + 2)
```

In [ ]:
```
print('Chris' + str(2))
```

Python has a built in method for convenient string formatting.

In [ ]:
```
sales_record = {
'price': 3.24,
'num_items': 4,
'person': 'Chris'}

sales_statement = '{} bought {} item(s) at a price of {} each for a total of {}'

print(sales_statement.format(sales_record['person'],
                             sales_record['num_items'],
                             sales_record['price'],
                             sales_record['num_items']*sales_record['price']))
```

We can then call the format method on that string and pass in the values that we want substituted as appropriate.

Now the string formatting language allows you to do much more than this. You can control a number of different things like decimal places, for floating point numbers, or whether you want to prepend the positive numbers with the plus sign, or set the alignment of strings to left or right justified. or even enable to use of scientific notation.

This has been a short lecture, but string manipulation is a big part of data cleaning.