Strings

Strings are a very important data type in all languages. In Python, strings may be quoted several ways:

```
inputfile = "data.txt"
outputfule = 'output.txt'
triplequotes = """woah!
split lines"""
print(triplequotes)
```

This also works:

```
triple_single_quotes = '''I am a string too.
I can span multiple lines!'''
print(triple_single_quotes)
```

Quotes in quotes

In Python, we can quote strings with either single (') or double quotes ("). Sometimes we want to create a string that contains quotes. This is easy to do!

Strings quoted with ' can contain ":

```
'Bob said, "it is hot out there today".'
```

Strings quoted with " can contain ':

```
"Python, it's a wonderful language"
```

Or we can escape the quote with \:

```
'it\'s not Nick\'s birthday today'
```

```
"I don't always quote my strings, but when I do, I prefer \""
```

Strings versus numbers

```
a = 5
b = '5'
a + b
```

```
print("type(a): ", type(a))
print("type(b): ", type(b))
```

It is simple to convert between numbers and strings!

```
# convert int to a string
a = str(55)
print(a)
print(type(a))
```

```
# convert string to a float
a = float("99.45")
print(a)
print(type(a))
```

String slicing

```
quote = """That's all folks!"""
print(quote[2])
print(quote[7:10])
print(quote[:4])
print(quote[7:])
print(quote[:-7])
```

One way to remember how slices work is to think of the indices as pointing between characters, with the left edge of the first character numbered `0`. Then the right edge of the last character of a string of `n` characters has index `n`, for example:

```
word = 'Python'
```

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

```
word[-2:]
```

Strings are immutable

We can access an individual character of a string:

```
a = 'hello'
a[0]
```

We cannot change any part of a string:

```
a[0] = 'k'
```

Concatenation

Concatenate (add together) strings with `+`:

```
b = 'j' + a[1:]
b
```

This creates a new string.

String functions / methods

```
name = 'Leland'
```

```
len(name)
```

```
name.lower()
```

```
name.upper()
```

```
name.find('lan')
```

```
name.find('lan', 1, 4)
```

There are many string methods.

String formatting

It is often important to create strings formatted from a combination of strings, numbers, and other data. In Python 3 this is best handled by the `format` string method. Here is a simple example:

```
name = "Nick"
course = 'CME211'
print("My name is {0}. I am the instructor for {1}.".format(name,course))
```

Format strings contain "replacement fields" surrounded by curly braces `{}`. Anything that is not contained in braces is considered literal text, which is copied unchanged to the output. If you need to include a brace character in the literal text, it can be escaped by doubling: `{{` and `}}`. The number in the braces refers to the order of arguments passed to `format`. Numbers don't need to be specified if the sequence of braces has the same order as arguments:

```
program = 'CME'
number = 211
print("this course is: {}-{}".format(program,number))
```

String formatting is a good way to combine text and numeric data.

String formatting is also how we control the output of floating point numbers:

```
print("    {{:f}}: {:f}".format(42.42))
print("    {{:g}}: {:g}".format(42.42))
print("    {{:e}}: {:e}".format(42.42))
print("  {{:.2e}}: {:.2e}".format(42.42))
print("{{: 8.2e}}: {: 8.2e}".format(42.42))
print("{{: 8.2e}}: {: 8.2e}".format(-1.0))
```

See the Python Format Mini-Language docs and more examples.