Strings

As mentioned in the data lecture earlier, you can make Python strings like `"this"`, `'this'`, `'t'`, or `"a"`. All of these are valid Python strings. Numbers, symbols and special characters also work.

Special characters (Escape Character)

Here is a question: how do I make a string with a quotation mark (the symbols `'` or `"`) inside of it? Won't it end the string? If you use the right one at the wrong time, yes you will. However Python has special character combinations to allow programmers to do this: behold the powerful backwards slash:

`\`

It's pretty cool. Let's use it

```
print "\t \' \"\n \\ \x21"
```

This bit of code would print out the following line:

```
____ '
" \ !
```

Note: The line (_____) isn't there in programming, I have it there to show you what counts as a `\t`.

Let's break down what is going on. * * This tells Python to insert a tab character into the string. Tab characters used for formatting a string (making it look pretty). It's purpose is to add 4 white spaces, however it can add less. Here is an example of `"\t1\t2"` against `"\t\t2"`

```
____1___2
________2
```

Notice that the two's position doesn't move. That's what tabs can do. * ' * This tells Python to insert a single quote mark and to not end the string. * " * This tells Python to insert a double quote mark and to not end the string. * `\n` * This is the newline character. It tells Python to rest everything that follows it on a new line below the current line. * \\ * This tells Python to insert a backslash into the string. This is needed if you are trying to end a string with a backslash, like "this", in Python. You need to use \ or an error will occur. * `\x21` * This is for a hexadecimal value. These are not used very much in beginner Python but it's good to go over. If I type `print "\x21"` into Python, the terminal will print out `!` because 21 is the hexadecimal value, 00100001 in binary and 33 in decimal, of `!` in ASCII.

Each of these symbols (most notably the sing and double quote marks) have special features by default and you must *escape* that feature when you want to use it as a *literal* symbol, so when you use the backslash, you are **escaping** the character and using it as a **literal**

Multi-line strings

A multi-line string can be made by using six double quotes, three at the beginning and three at the end, for saving a string.

```
word = """This is an example
of multi-line
strings"""
```

the variable `word` could contain the following values: `This is an example\nof multi-line\nstrings`. ## Printing

As shown before, you print a string (or a number) by having `print` followed by the item, like

```
print "this"
```

would output `this`. You can add strings together like the following as well

```
print "Hello, " + "world!"
```

which outputs "Hello, world!". This is known as concatenation. The string "hello" It might be bothersome to have to add spaces after words when doing this, so you can use commas (`,`) instead of pluses (`+`).

```
print "Hello,","world!"
```

still gives us "Hello, world!" because a comma is equal to `+' '+`in Python.

Math + Strings = ?

Here is some code:

```
word = "na"
print word * 16, "BATMAN!"
```

prints `nanananananananananananananananana BATMAN!`.

If you have a string and multiple it by a int, it will appear that many times. It can be saved or printed.

Now what would happen if I do the following:

```
word = "what about this?"
print word + 2
```

```
print word + '1'
print word * 4.5
```

Will any of these work?