Letters

Chapter 2

So we've learned all about numbers, but what about letters? words? text?

We refer to groups of letters in a program as strings. (You can think of printed letters being strung together on a banner.) To make it easier to see just what part of the code is in a string, I'll color strings red. Here are some strings:

```
'Hello.'
'Ruby rocks.'
'5 is my favorite number... what is yours?'
'Snoopy says #%^?&*@! when he stubs his toe.'
'     '
''
```

As you can see, strings can have punctuation, digits, symbols, and spaces in them... more than just letters. That last string doesn't have anything in it at all; we would call that an empty string.

We have been using puts to print numbers; let's try it with some strings:

```
puts 'Hello, world!'
puts ''
puts 'Good-bye.'
```

```
Hello, world!

Good-bye.
```

That worked out well. Now try some strings of your own.

String Arithmetic

Just as you can do arithmetic on numbers, you can also do arithmetic on strings! Well, sort of... you can add strings, anyway. Let's try to add two strings and see what puts does with that.

```
puts 'I like' + 'apple pie.'
```

```
I likeapple pie.
```

Whoops! I forgot to put a space between 'I like' and 'apple pie.'. Spaces don't matter usually, but they matter inside strings. (It's true what they say: computers don't do what you want them to do, only what you tell them to do.) Let's try that again:

```
puts 'I like ' + 'apple pie.'
puts 'I like' + ' apple pie.'
```

```
I like apple pie.
I like apple pie.
```

(As you can see, it didn't matter which string I added the space to.)

So you can add strings, but you can also multiply them! (By a number, anyway.) Watch this:

```
puts 'blink ' * 4
```

```
batting her eyes
```

(Just kidding... it really does this:)

```
blink blink blink blink
```

If you think about it, this makes perfect sense. After all, 7*3 really just means 7+7+7, so 'moo'*3 just means 'moo'+'moo'+'moo'.

12 vs '12'

Before we get any further, we should make sure we understand the difference between numbers and digits. 12 is a number, but '12' is a string of two digits.

Let's play around with this for a while:

```
puts  12  +  12
puts '12' + '12'
puts '12  +  12'
```

```
24
1212
12  +  12
```

How about this:

```
puts  2  *  5
puts '2' *  5
puts '2  *  5'
```

```
10
22222
```

```
2  *  5
```

These examples were pretty straightforward. However, if you're not too careful with how you mix your strings and your numbers, you might run into...

Problems

At this point you may have tried out a few things which didn't work. If not, here are a few:

```
puts '12' + 12
```

```
TypeError: no implicit conversion of Fixnum into String
```

```
puts '2' * '5'
```

```
TypeError: no implicit conversion of String into Integer
```

Hmmm... an error message. The problem is that you can't really add a number to a string, or multiply a string by another string. It doesn't make any more sense than does this:

```
puts 'Betty' + 12
puts 'Fred' * 'John'
```

Something else to be aware of: you can write 'pig'*5 in a program, since it just means 5 sets of the string 'pig' all added together. However, you can't write 5*'pig', since that means 'pig' sets of the number 5, which is just silly.

Finally, what if I want a program to print out You're swell!? We can try this:

```
irb(main):001:0> puts 'You're swell!'
irb(main):002:0'  '
SyntaxError: (irb):1: syntax error, unexpected tIDENTIFIER, expecting end-of-input
puts 'You're swell!'
            ^
```

Well, that won't work; I won't even try to run it. The computer thought we were done with the string. (This is why it's nice to have a text editor which does syntax coloring for you.) So how do we let the computer know we want to stay in the string? We have to escape the apostrophe, like this:

```
puts 'You\'re swell!'
```

```
You're swell!
```

The backslash is the escape character. In other words, if you have a backslash and another character, they are sometimes translated into a new character. The only things the backslash escapes, though, are the apostrophe and the backslash itself. (If you think about it, escape characters must always escape themselves.) A few examples are in order here, I think:

```
puts 'You\'re swell!'
puts 'backslash at the end of a string:  \\'
puts 'up\\down'
puts 'up\down'
```

```
You're swell!
backslash at the end of a string:  \
up\down
up\down
```

Since the backslash does not escape a 'd', but does escape itself, those last two strings are identical. They don't look the same in the code, but in your computer they really are the same.

If you have any other questions, just keep reading! I couldn't answer every question on this page, after all.