Playing Around With Strings

Let's dive into Strings and see how can you concatenate them. This lesson will also introduce you to Automatic Type Casting and you'll learn how to parse Strings as Integers and Floats.

A string is basically a series of characters.

Concatenating strings

Let’s perform some operations on strings.

```
console.log('Javascript in ' + 'Practice')
```

The plus operator concatenates strings. Concatenation means that you join the contents of two strings one after the other.

Strings are *immutable* which means that their value cannot be changed. When concatenating two strings, the result is saved in a third string.

Automatic Type Casting

If any of the operands of plus is an integer, the result becomes a string. JavaScript automatically converts the operands of an operator to the same type. This is called *automatic type casting*:

```
console.log("1 + \'2\' becomes",1 + '2');
console.log("\'1\' + 2 becomes",'1' + 2);
```

Rules may become confusing, so don’t abuse automatic type casting.

Parsing Strings as Integers

Just know that you may have to explicitly cast a string to an integer to be able to add it to another integer:

```
console.log("1 + + \"2\"")
console.log(1 + +"2") // +"2" gives a sign to "2", converting it to a number

console.log("1 + Number(\"2\")")
console.log(1 + Number("2"))

console.log("1 + Number.parseInt( \"2\", 10 )")
console.log(1 + Number.parseInt( "2", 10 ))

console.log("1 + Number.parseInt( \"2\" )")
console.log(1 + Number.parseInt( "2" ))
```

All the above conversions work. The first relies on giving a sign to a numeric string which converts it to a number. Then `1+2` becomes `3`. The second type cast is more explicit: you use `Number` to wrap a string and convert it to a number.

I recommend using the third option: `Number.parseInt` with a radix. `parseInt` converts a string into a number. The second argument of `parseInt` is optional: it describes the base in which we represent the number.

```
console.log("Number.parseInt(\"JS in Practice\")");
console.log(Number.parseInt("JS in Practice"));

console.log("Number.parseInt( \"10\", 2 )");
console.log(Number.parseInt( "10", 2 ));

console.log("Number.parseInt( \"a\" )");
console.log(Number.parseInt( "a" ));

console.log("Number.parseInt( \"a\", 16 )");
console.log(Number.parseInt( "a", 16 ));
```

Arbitrary strings are often `NaN`. “2” in base 2 is 10. You can see how easy it is to convert a binary or a hexadecimal (base 16) string into a decimal number. Base 16 digits are `0123456789abcdef`. The last 6 digits may also be upper case.

`Number.parseInt` recognizes the starting characters of a string as integer numbers, and throws away the rest:

```
console.log(Number.parseInt( "1234.567 89" ));
```

Parsing Strings as Floating Numbers

The dot is not a character present in integer numbers, so everything after `1234` is thrown away by `Number.parseInt`.

You can also use `Number.parseFloat` to parse floating point. It parses the floating point number until the terminating space:

```
console.log(Number.parseFloat( "1234.567 89" ));
```