Strings and Numbers

Strings

As we now know, a string is a data type used to represent words, sentences, or writings. To create a string, we can use the `"` and `'` symbols to wrap our string value. For example:

```
const myName = "John Doe"
const yourName = 'Jane Doe'
```

Note that when we use a symbol (`"` `'`) to create a string, we also need to close it with the same symbol we started with.

Dynamic String Values using Variables

We can create dynamic string values by combining/concatenating our string with a variable. For example, let's say we want to create a string with the value of "Hello, my name is John" but we want "John" to be dynamic, meaning that it its value can change. So, that means we have to store "John" inside a variable, making the string look something like this: "Hello my name is `myName`". Now that we have a picture of what we're supposed to do, let's turn it into code.

Plus Operator

First, we need to remember that when we type in the name of a variable inside a string, it will not register as a variable, rather only as a normal string. The way to do it is by using a technique called ***string concatenation***. There are several ways to do so, but let's start with the simplest one, using the plus (+) operator.

```
let myName = "John"
console.log("Hello, my name is " + myName)
```

In this example, we used the plus operator to concatenate (combine) the string with `myName` as opposed to using it as an arithmetic operator. So that's one thing to take note, that when working with strings, the plus operator is used to concatenate values, not sum them up together. Now, if we take a closer look at the example we can see that at the end of the word "is", there's a blank character (space). This is because when concatenating strings, the plus operator will only join the values together without adding any kind of separation, in this case, the blank character. So what we have to do is add a blank character before concatenating our string with a variable, quite simple.

String.concat()

The second way of concatenating strings is by using the `.concat` method. However this method is very rarely used since it has way more error cases compared to using the plus operator. It also works the same way as how the plus operator works, therefore it doesn't make much of a difference. Here's how you can use the method.

```
const introMessage = "Hello, my name is "
const myName = "John"

console.log(introMessage.concat(myName))
```

You can actually add more arguments to the method and it will concatenate those values as well.

```
const introMessage = "Hello, my name is "
const myName = "John "

console.log(introMessage.concat(myName, ", nice to meet you"))
```

Template Literals

The third way and the most used way is by using template literals. To use template literals, all you need to do is wrap your string values with the backtick (`` ` ``) character. Using template literals, we can write multiline strings, insert variables between them, and even write Javascript expressions within the backtick's scope. To write a Javascript expression within the string, we can write a dollar sign symbol followed by curly braces `${}` and use the curly braces to wrap our Javascript expression. Let's take a look at an example.

```
const myName = "John"

console.log(`Hello, my name is ${myName}, nice to meet you`)
```

In this example, we called `myName` inside a string, and as you can see it's much simpler and concise compared to using the plus operator method and the concat method. Remember that we can also write other Javascript expressions within the curly braces, so it's possible to do things like executing functions and writing ternaries (we'll get to these topics in the next sessions) inside them.

Parsing Data Types

toString

Nearly all data types in Javascript can be converted into a string by using the `.toString` method. Here are some examples.

```
// number -> string
const myAge = 29
const ageString = myAge.toString()

// array -> string
const myBooks = ["Harry Potter", "Lord of The Rings", "Sherlock Holmes"]
const bookStrings = myBooks.toString()
```

parseInt

Numbers and strings in Javascript can sometimes be weird, for example take a look at this code snippet.

```
const x = "3"
console.log(10 + x)
```

To most of us, the expected output of the code above should be `13`. Instead, it actually results in `"103"`. The reason is because `x` is actually a string, and because Javascript detects that a string is involved when using the plus operator, it no longer uses the plus operator as an arithmetic operator, rather it uses the plus sign to concatenate. To summarise, Javascript detects that `x` is a string, it somehow converts `10` into a string as well, and concatenates them both resulting in `"103"` with a data type of string.

To fix the code, we need to parse `x` from a string into a number, we can do this by using the `parseInt` function. We only need to call `parseInt` wrap a string with a numeric value using parentheses and it will automatically return an integer (number). Keep in mind that this will convert the string into an integer which means it will only become a number with no decimal or fractional parts. So strings like `"3.14"` will lose its decimals when we use `parseInt` against it. Here are some examples:

```
let myAge = "31"
myAge = parseInt(myAge) // myAge = 31

let pi = "3.14"
pi = parseInt(pi) // pi = 3
```

As we can see `pi` loses its decimal points when it's converted using `parseInt`. Note that this was not because it was rounded down, rather it just lost its decimal points. Meaning, strings with decimal points `.5` and above will still be "rounded down", but not because it was actually rounded down, but because it lost its decimal points.

If the decimal points need to be kept, we can use `parseFloat` instead. `parseFloat` works exactly like `parseInt` but the only difference is that `parseFloat` converts the values into a floating number, which means that decimal points can be kept.