**Substitutions vs Template Literals**

I want to clarify something.

The point of the previous lecture was to show that **template literals are a better approach** to adding variables and expression within a string in most situations (but as you'll see below, sometimes we have to use string substitutions).

I'll warn you - this is a little advanced and there's not a lot of info on the web about this. In fact, many developers don't even know that the console gives us access to string substitutions. **But don't stress.**That's what I'm here for and I've got your back ;) I hope you find this useful.

Lets jump into it.

Where do string substitutions come into play?

Every browser exposes a console that lets you interact with APIs and also gives you an inside look at the code by printing messages that are generated by your JavaScript code running in the page. The **Console API**is a small, but powerful part of what the Google Chrome DevTools gives you.

As your app grows, you’ll likely be producing more output messages in the console that needs to be reviewed. One of the best ways to manage this and maintain your output logs is to understand the formatting options you have at your disposal.

And one of formatting options that Chrome gives you is what they call string substitutions***.***

What are string substitutions?

They are also known as format specifiers, and they work a little like printf in C or PHP.

Format specifiers contain the % symbol, followed by a letter that specifies the kind of formatting that should apply to the value. Using format specifiers, we can render data as strings (%s), integers (%d or %i), floats (%f), JavaScript objects (%O) and even DOM elements (%o).

**IMPORTANT:**I really don’t want you to get confused here. String substitutions are not part of the JavaScript language – they are given to us by the browser (specifically, the Console API).

Why are string substitutions useful?

Firstly, I want to say that string substitutions are very tightly related to console.log(). This basically provides you with the option to use specific expressions in a string, that will later be replaced by provided arguments. It can look somewhat like this (the BLUE text is the output produced):

Graphical user interface, text, application

Description automatically generated

You can even style your logs to the console, like this for example:

Text

Description automatically generated

Why do we use string substitutions?

Well, when doing some console logging from your code, you may feel like you want certain text to stand out. String substitutions allow you to do just that, with ease!

But, what if the formatting is not so much of an issue for you? What if you want to output variables and objects in the console? Then my dear students, there is a better way …

**Use template literals instead**

**Remember, substitutions are only used for logging things to the screen. In other words, we can only use string substitutions when we want to output something (e.g. when we use error, warn, info, log and debug).**

This means we **can’t** use substitutions in our JavaScript code.

Let me give you an example.

In Python and other programming languages, you can substitute a variable’s value in between a string like this:

1. name = "python"
2. x = "My language is %s"%name
3. print x
4. >>>My language is python

But in JavaScript you can’t do this with string substitutions. We have to use ... yep you guessed it, **template literals**.

Another reason we should be using template literals is maintainability. A big problem with string substitutions is that the parameter values have to be in correct order. This means that using string substitutions requires you to REMEMBER what order you need to put things in. In other words, you must explicitly reset each style, object, variable, etc. and remember what %s token, %d token or %o token goes with which console log method parameter so formatters will not bleed over into the next section.

See how confusing it will become?

What is my point?

Using string substitutions can get cumbersome. Because of this, libraries were created to make it easier for us (like *console.style*).

But since template literals have entered the scene, we can mostly avoid string substitutions all together.

In conclusion, try and avoid them. Sometimes you have to use string substitutions, like if you want to specifically style your console.log() text in a beautiful and funky way.

But there is no denying it - template literals are a LOT more intuitive.

The example I used in the lecture was actually quite simple, so it doesn’t quite matter which approach we used. But if your code starts getting more complicated, the recommended approach is to use template literals to get the job done in a more structured way.

Text

Description automatically generated

And the output:

Text

Description automatically generated

I hope this has cleared things up.

See you in the next lecture.