Tip 4: Variables to Readable Strings with Template Literals

In this tip, you will learn how to convert variables into new strings without concatenation.

We'll cover the following

Combining strings

Strings are messy. That's all there is to it. When you're pulling information from strings, you have to deal with the ugliness of natural language: *capitalization, punctuation, misspellings*. It's a headache.

Collecting information into strings is less painful, but it can still get ugly quickly. Combining strings in JavaScript can be particularly rough, especially when you combine strings assigned to variables with strings surrounded by quotes.

Example

Here's a situation that comes up all the time: You need to build a URL. In this case, you're building a link to an image on a cloud service. Your cloud service is pretty great, though. In addition to hosting the asset, you can pass query parameters that will convert the asset in a variety of ways (height, width, and so on).

To keep things relatively simple, you're going to make a function that creates a URL by combining your cloud provider URL with the ID of the image and the width as a query parameter.

To keep things complicated, you're going to combine *regular strings* with strings that are *returned from a function*, strings that are *assigned to variables*, and strings that are converted *right before concatenation*. You're going to use a function (implemented elsewhere) that will return a cloud provider such as

pragprog.com/cloud. Your function will take ID and `width` as parameters, but it will need to parse the `width` to make sure it's an integer.

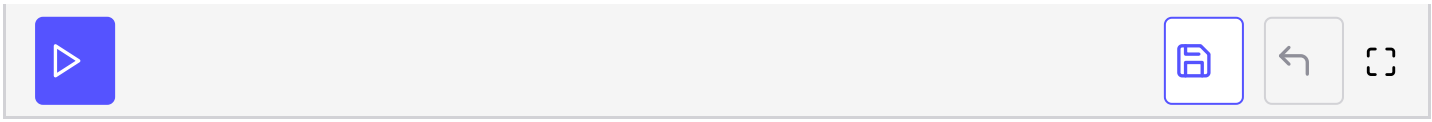
URLs get particularly ugly because you have to add slashes between the parts of a route along with the building blocks of queries such as `?`, `=`, and `&`. Traditionally, you have to combine each piece with a `+` sign.

The final result looks like this:

```
function getProvider() {
    return 'pragprog.com/cloud';
}

function generateLink(image, width) {
    const widthInt = parseInt(width, 10);
    return 'https://' + getProvider() + '/' + image + '?width=' + widthInt;
}

const image = 'foo';
const width = 200.5;
console.log(generateLink(image,width));
```



There's a lot going on there, and the combination of information and `+` signs doesn't help. And this is a particularly simple URL. They can get more complicated fast. What if the route was longer or you needed an additional four parameters? These things get long.

Fortunately, you can cut down the complexity quite a bit using *template literals*.

Template literals

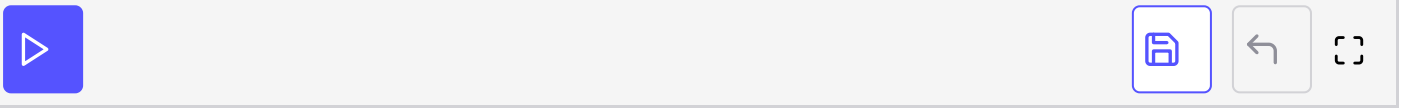
Template literals are a simple syntax that lets you *combine strings along with JavaScript expressions to create a new string*.

There are only two things you need to know:

- First, a template literal is surrounded by backticks (`) instead of single or double-quotes.
- Second, anything that's not a string (including strings assigned to variables) needs to be surrounded by a special designator: a `$` sign with the variables or other JavaScript code in curly braces: `${stuff}`.

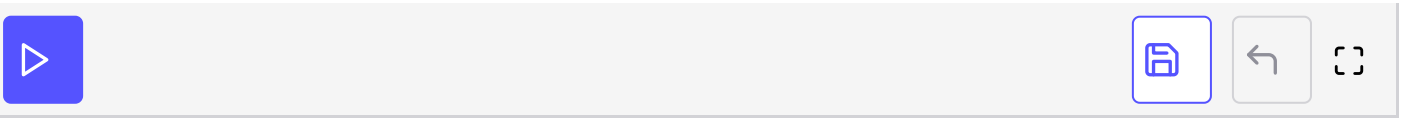
You'll most often use this for combining strings and variables.

```
function greet(name) {
    return `Hi, ${name}`;
}
console.log(greet('Leo'));
```



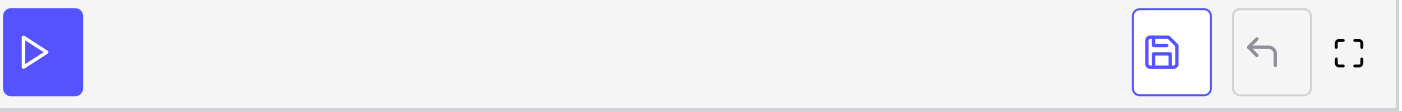
But you can also perform JavaScript actions. For example, you can call a method on a object. In this case, you're converting a string to uppercase:

```
function yell(name) {
    return `HI, ${name.toUpperCase()}!`;
}
console.log(yell('Pankaj'));
```



You can even perform more complex computations, such as combining math calculations. Really, you can perform any action in the curly braces, but it would only make sense to perform actions that return a string or integer.

```
function leapYearConverter(age) {
    return `You'd be ${Math.floor(age / 4)} if born on a leap year.`;
}
console.log(leapYearConverter(34));
```



Try not to do much with the curly braces. It can be more cluttered than it's worth. If you need to do heavy data conversions, perform the action outside the template literal and assign the result to a variable.

You now have all the tools to rewrite your original string concatenation as a single template literal. Take a moment and try it out.

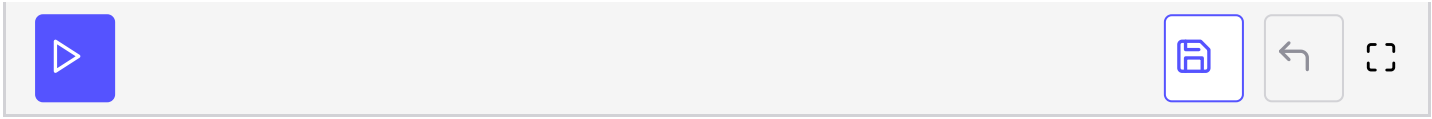
Your solution probably looks something like this:

```
function getProvider() {
    return 'pragprog.com/cloud';
}

function generateLink(image, width) {
```

```
function generateLink(image, width) {
    return `https://${getProvider()}/${image}?width=${parseInt(width, 10)}`;
}

const image = 'foo';
const width = 200.5;
console.log(generateLink(image,width));
```



Doesn’t that look significantly cleaner? Template literals are such an improvement on string concatenation that you should rarely ever combine strings with traditional concatenation. The only time it would be better is if you’re combining two variables with no additional information. Even in that case, you may still use template literals because those backticks are a clue to other developers that you’re returning a string.

In the next chapter, you’re going to learn about how to use collections of data. You’ll be building on many of the ideas in this chapter as you make choices between new and existing collections.