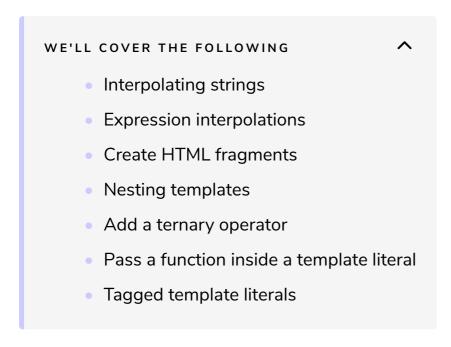
Template Literals

This lesson covers new ways of interpolating strings and more, with template literals.



Template literals were called *template strings prior* to ES6... Let's have a look at what's changed in the way we interpolate strings in ES6.

Interpolating strings

We used to write the following in ES5 in order to interpolate strings:

```
var name = "Alberto";
var greeting = 'Hello my name is ' + name;

console.log(greeting);
// Hello my name is Alberto

\[ \begin{align*}
\text{ \
```

In ES6, we can use backticks to make our lives easier. We also need to wrap our variable names in \${}

```
let name = "Alberto";
const greeting = `Hello my name is ${name}`;

console.log(greeting);
// Hello my name is Alberto
```

Expression interpolations

In ES5 we used to write the following:

```
var a = 1;
var b = 10;
console.log('1 * 10 is ' + ( a * b));
// 1 * 10 is 10
```

In ES6 we can use backticks to reduce our typing:

```
var a = 1;
var b = 10;
console.log(`1 * 10 is ${a * b}`);
// 1 * 10 is 10
```

Create HTML fragments

In ES5 we used to do the following to write multi-line strings:

```
// We have to include a backslash on each line
var text = "hello, \
my name is Alberto \
how are you?\ "
```

In ES6 we simply have to wrap everything inside backticks, so backslashes on each line aren't needed anymore.

```
const content = `hello,
my name is Alberto
how are you?`;
```

Nesting templates

It's very easy to nest a template inside another one, like this:

```
const people = [{
                                                                                   G
       name: 'Alberto',
       age: 27
},{
       name: 'Caroline',
       age: 27
},{
       name: 'Josh',
       age: 31
}];
const markup = `
 ${people.map(person => ` ${person.name}`)}
console.log(markup);
```

Here, we're using the map function to loop over each of our people and display a li tag containing the name of the person.

Add a ternary operator

We can easily add some logic inside our template string by using a ternary

operator.

The syntax for a ternary operator looks like this:

```
const isDiscounted = false

function getPrice(){
      console.log(isDiscounted ? "$10" : "$20");
}
getPrice();
// $20
```

If the condition before the ? can be converted to true then the first value is returned. Otherwise, it's the value after the : that gets returned.

```
// create an artist with name and age
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         6
let artist = {
           name: "Bon Jovi",
            age: 56,
};
// only if the artist object has a song property we then add it to our paragraph, otherwise w
const text = `
            <div>
                         ${artist.name} is ${artist.age} years old ${artist.song ? `and wrote the song ${artist.song } artist.song } artist.song ? `and wrote the song ${artist.song } artist.song } artist.song ? `and wrote the song ${artist.song } artist.song } ar
                        </div>
console.log(text);
 // <div>
//  Bon Jovi is 56 years old
// 
  // </div>
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              []
```

Let's try using an object with the property song:

```
const artist = {
  name: "Trent Reznor",
  age: 53,
  song: 'Hurt'
};

// only if the artist object has a song property we then add it to our paragraph, otherwise w
const text = `
  <div>
```

Pass a function inside a template literal

Similarly to the example above (line 10 of the code), we can pass a function inside a template literal.

```
const groceries = {
                                                                                  G
 meat: "pork chop",
 veggie: "salad",
 fruit: "apple",
 others: ['mushrooms', 'instant noodles', 'instant soup'],
// this function will map each individual value of our groceries
function groceryList(others) {
 return
     ${others.map( other => ` <span> ${other}</span>`).join('\n')}
   }
// display all our groceries in a p tag, the last one will include all the one from the array
const markup = `
 <div>
   ${groceries.meat}
   ${groceries.veggie}
   ${groceries.fruit}
   ${groceryList(groceries.others)}
 <div>
//
   <div>
//
      pork chop
//
      salad
//
      apple
//
      >
//
      >
//
         <span> mushrooms</span>
//
          <span> instant noodles</span>
//
          <span> instant soup</span>
//
```

```
// 
// <div>
```

Inside of the last p tag, we're calling our function groceryList and passing it all the others groceries as an argument. Inside of the function, we're returning a p tag and are using map to loop over each of our items in the grocery list. This includes returning an array of span tags containing each grocery. We're then using .join('\n') to add a new line after each of those spans.

Tagged template literals

By tagging a function to a template literal, we can run the template literal through the function, providing it with everything that's inside of the template.

The way it works is very simple: you take the name of your function and put it in front of the template that you want to run it against.

```
let person = "Alberto";
let age = 25;

function myTag(strings,personName,personAge){
  let str = strings[1];
  let ageStr;

  personAge > 50 ? ageStr = "grandpa" : ageStr = "youngster";

  return personName + str + ageStr;
}

let sentence = myTag`${person} is a ${age}`;
  console.log(sentence);
// Alberto is a youngster
```

We captured the value of the variable age and used a ternary operator to decide what to print. strings will take all the strings of our let sentence, while the other parameters will hold the variables.

In our example the string is divided into 3 pieces: \$\{person\}, is a and \$\{age\}\\$. We use array notation to access the string in the middle like this:

C

To learn more about use cases of template literals check out this article.

In the next lesson, we'll take another quiz and a coding challenge to test the concepts we just covered.