Default Parameters

Often times when writing a function, you need to assign default values for arguments that weren't passed to the function when it was invoked.

For example, let's say we were creating a calculatePayment function. This function has three parameters, price, salesTax, and discount. The purpose of this function, as the name implies, is to calculate the final price of a bill taking into account the initial price as well as any sales tax or discounts that should be applied.

With a function like this, the only parameter that we want to make required is the price. We'll set the default value of salesTax to 0.05 (5%) and the default value of discount to 0 so our function will still work if those values aren't passed in when the function is invoked. This way, the consumer of this function can supply a sales tax as well as a discount if they want, but if they don't, the default values will kick in.

```
calculatePayment(10); // 9.50
calculatePayment(10, 0, 10); // 9.00
```

Historically, one way you could accomplish this is by using the Logical || operator.

```
function calculatePayment(price, salesTax, discount) {
  salesTax = salesTax || 0.05;
  discount = discount || 0;

  // math
}
```

If you're not familiar with ||, you can think of it like you would an if statement checking for falsy values.

```
function calculatePayment(price, salesTax, discount) {
  if (!salesTax) {
    salesTax = 0.05;
  }

  if (!discount) {
    discount = 0;
  }
  // math
}
```

However, this approach has some downsides. Can you spot the issue? What if we wanted to set the salesTax to 0? With our current implementation that would be impossible since 0 is

classified as a falsy value so our if (!salesTax) would always evaluate to true setting the salesTax to our default value of 0.05. To fix this, let's check for undefined rather than falsy.

```
function calculatePayment(price, salesTax, discount) {
  salesTax = typeof salesTax === "undefined" ? 0.05 : salesTax;
  discount = typeof discount === "undefined" ? 0 : discount;

  // math
}
```

Now, both salesTax and discount will only take on their default values if their arguments are undefined.

At this point our code works well, but as you'll see, there's now a better way to do this with ES6's "Default Parameters".

Default Parameters

Default Parameters allow you to set the default values for any parameters that are undefined when a function is invoked. Using Default Parameters, we can now update our calculatePayment function to look like this,

```
function calculatePayment(price, salesTax = 0.05, discount = 0) {
  // math
}
```

Now, just as we had before, if salesTax or discount are undefined when calculatePayment is invoked, they'll be set to their default values of 0.05 and 0.

Required Arguments

One neat trick you can do using Default Parameters is to throw an error if a function is invoked without a required argument. For example, what if we wanted calculatePayment to throw an error if the price wasn't specified when it was invoked?

To do this, first create the function that will throw the error.

```
function isRequired(name) {
  throw new Error(`${name} is required`);
}
```

Next, using Default Parameters, assign the required parameter to the invocation of isRequired

```
function calculatePayment (
  price = isRequired('price'),
  salesTax = 0.05,
  discount = 0
) {

  // math
}
```

Now if calculatePayment is invoked without a price, JavaScript will invoke the isRequired function, throwing the error.