Checkpoint 10

Default parameters

As you've likely noticed, your JavaScript skills are getting more and more sophisticated! In this checkpoint, you'll dive even deeper and learn about setting *defaults* in functions. You will also learn how to combine your destructuring skills with this new technique.

Outline

By the end of this checkpoint, you will be able to do the following:

- Set defaults for parameters in a function

Setting defaults

Take a look at the following function. What do you notice?

```
function calculatePriceInDollars(priceInCents) {
  return priceInCents / 100
}
```

This simple function takes a number—in this case, a specific amount of money—and then divides it by `100`. However, if no argument is entered when the function is called, it will result in `NaN`. This can be seen below.

```
calculatePriceInDollars(250); //> 2.5
calculatePriceInDollars(); //> NaN
```

But imagine you want the value to be `0` instead of `NaN` if no argument is entered. In this case, you can set `0` as the default for the `priceInCents` parameter. Setting a default means you've identified a value to be there even if a specific value is missing from the function call.

```
function calculatePriceInDollars(priceInCents = 0) {
  return priceInCents / 100;
}

calculatePriceInDollars(); //> 0
```

To set a default, place an `=` sign after the parameter name, followed by whatever value you want to use as the default. This value can be anything valid in JavaScript.

If the value of that parameter is `undefined` when the function is called, it will be replaced with the valu

Why use defaults?

At this point, you may be wondering: instead of using default parameters, shouldn't I just make sure that my functions are always called correctly? Indeed, in the above example of the `calculatePriceInDollars()` function, it might seem like it would make more sense just to *always use an argument*. And in general, this is true—it *is* better to simply use your functions as intended rather than set numerous defaults.

However, defaults can be particularly useful in a few circumstances:

- If there is a typical case for your function, but you want to allow for certain options, you might use defaults.
- If one or more of the argument values you're using come from an outside source, you might use defaults because you won't necessarily be able to control the input values.

Outline

Consider the example below.

```
function welcomeMessage(name, isReturnCustomer = fals
  const message = isReturnCustomer ? "Welcome back" :
  return `${message}, ${name}!`;
}
```

You may expect the above function typically only to pass in a single argument. However, if the customer is a returning one, you could easily change the message by adding a sec

function could use defaults to allow for a typical case with some options.

Object defaults

Now, you'll investigate object defaults.

Code setup

You'll use the code below for the following examples, so take a moment to review this function and figure out how it's used.

```
// Function definition
function calculateTotal(products, { discountPercentag
  let result = 0;
  for (let i = 0; i < products.length; i++) {
    const { priceInCents, quantity } = products[i];
    result += priceInCents * quantity;
  }

  if (discountPercentage) result *= 1 - discountPerce
  return result * (1 + salesTax);
}

// Inputs
const products = [
  {
    name: "Black Longline T-Shirt",
    priceInCents: 1500,
    size: "M",
    quantity: 2,
  },
  {
    name: "Light Wash Stretch Skinny Jeans",
    priceInCents: 6000,
    size: 32,
    quantity: 1,
  },
];
const options = { discountP

// Function call
calculateTotal(products, options); //> 8768.25
```

The above function takes in an array of products and calculates the total. If `discountPercentage` is included in the `options` parameter, it will apply a discount. It will then add `salesTax` to the total.

Do this

Run `calculateTotal()`

Copy the function above and paste it into a place or program, such as a code editor, where you can edit it and then run the function. Try changing the quantities and options to see what happens.

Set an object as a default

Outline

Now, in the case of the function above, imagine there are no discounts, and you just want to use a standard sales tax. In this situation, you may want to just call the function without passing anything in. However, this will result in an error.

```
calculateTotal(products);
//> Uncaught TypeError:
//> Cannot destructure property 'discountAmount' of
```

The issue above is that there is no object to destructure, so the code fails to create those variables. But just like you saw above, you can set a default parameter by adding an `=` sig

```
function calculateTotal(products, { discountPer    tag
  // ...
}
```

What is happening in the function signature above? Take a look:

- If there *is an object* passed in as the second argument, the code will destructure that argument and assign the `discountPercentage` and `salesTax` variables.
- If there *is no object* passed in as the second argument, the code will default to an object. *Then* it will attempt to destructure that empty object. The variables will exist, but they will both be set to `undefined`.

Update `calculateTotal()`

Update the `calculateTotal()` function with the above code. Try running it once more, only passing in `products` as an argument. Is the result what you expected? Or is it different?

Outline

Default key values

If you run the function above without a second argument, it will run. But it will give you an undesirable result.

```
calculateTotal(products); //> NaN
```

In this case, the result will be `NaN` because `salesTax` is `undefined`. This problem can be sc

```
function calculateTotal(products, { discountPer   tag
  // ...
}
```

The code above sets a default value for the key in the object. If `salesTax` is `undefined`, `salesTax` will in fact be set to the value of `0.0825`.

Now, if you call this function with `products` as the only argument, it will work correctly.

```
calculateTotal(products); //> 9742.5
```

Do this

Include default values

Outline

Update your `calculateTotal()` function so that it includes a default for the `salesTax` value. Try running the function once again. What is the result?

Setting a default array

With this practice under your belt, you're ready for one final modification. You can update the code so that if `products` happens to be `undefined`, the function will just return `0`. Specifically, you would change the function signature to look like the one below:

```
function calculateTotal(products = [], { discountPerc
  // ...
}
```

The above code sets the default for `products` to be an empty array. This means that the `for` loop will still be able to run.

```
calculateTotal(); //> 0
```

The entire updated function is below.

```
function calculateTotal(products = [], { discountPerc
  let result = 0;
  for (let i = 0; i < products.length; i++) {
    const { priceInCents, quantity } = products[i];
    result += priceInCents * quantity;
  }

  if (discountPercentage) result *= 1 - discountPerce
  return result * (1 + salesTax);
}
```

Outline

Checkpoint

This checkpoint will be autograded. Please click the link below to open your assignment in a new tab. Once you complete the assignment, you will see a button allowing you to submit your answers and move on to the next checkpoint.

Your work

03.29.21 **Approved**

✓ Completed

Next checkpoint

How would you rate this content?

Report a typo or other issue

Go to Overview

Outline

Feeling stuck?
Chat live with an expert now. Beta