Default Function Arguments

In this lesson you will learn what are default function arguments and how to use them.

WE'LL COVER THE FOLLOWING

- Default function arguments

Prior to ES6, setting default values to function arguments was not so easy. Let's look at an example:

```
function getLocation(city,country,continent){
  if(typeof country === 'undefined'){
    country = 'Italy'
  }
  if(typeof continent === 'undefined'){
    continent = 'Europe'
  }
  console.log(continent,country,city)
}

getLocation('Milan')
// Europe Italy Milan

getLocation('Paris','France')
// Europe France Paris
```

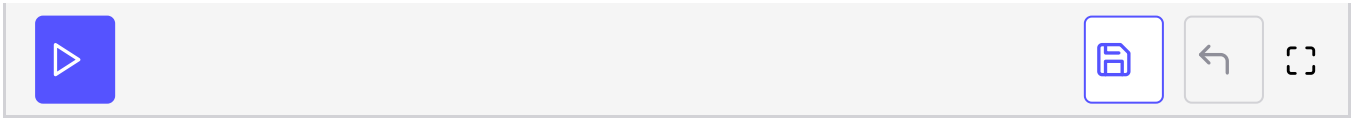
As you can see, our function takes three arguments: a city, a country and a continent. In the function body we are checking if either country or continent are `undefined` and in that case, we are giving them a **default value**.

When calling `getLocation('Milan')` the second and third parameter (country and continent) are undefined and get replaced by the default values of our functions.

But what if we want our default value to be at the beginning and not at the end of our list of arguments?

end of our list of arguments?

```
function getLocation(continent,country,city){
  if(typeof country === 'undefined'){
    country = 'Italy'
  }
  if(typeof continent === 'undefined'){
    continent = 'Europe'
  }
  console.log(continent,country,city)
}

getLocation(undefined, undefined,'Milan')
// Europe Italy Milan

getLocation(undefined,'Paris','France')
// Europe Paris France
```



If we want to replace any of the first arguments with our default we need to pass them as `undefined`, which is not the nicest looking solution. Luckily, ES6 came to the rescue with default function arguments.

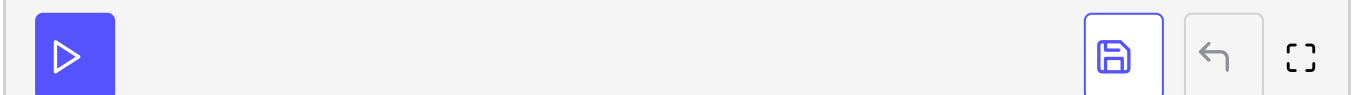
Default function arguments

ES6 makes it very easy to set default function arguments. Let's look at an example:

```
function calculatePrice(total, tax = 0.1, tip = 0.05){
// When no value is given for tax or tip, the default 0.1 and 0.05 will b
e used
return total + (total * tax) + (total * tip);
}
```

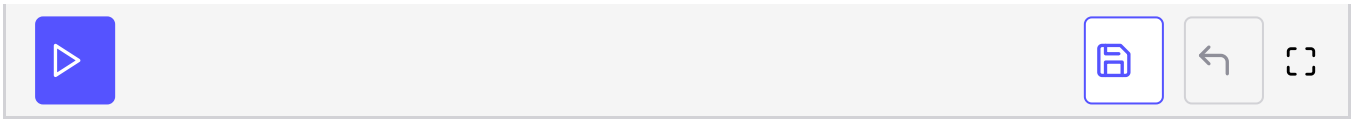
What if we don't want to pass the parameter at all, like this:

```
function calculatePrice(total, tax = 0.1, tip = 0.05){
  console.log(total + (total * tax) + (total * tip));
}
// The 0.15 will be bound to the second argument, tax even if in our intention it was to set
calculatePrice(100, 0.15)
```



We can solve by doing this:

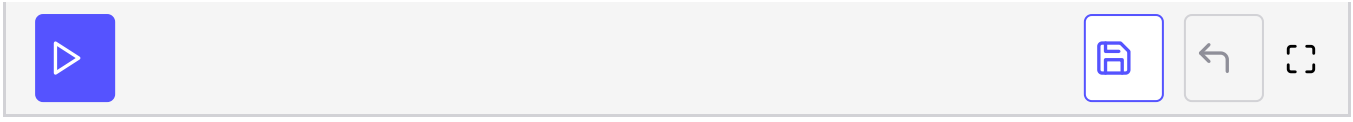
```
function calculatePrice(total, tax = 0.1, tip = 0.05){
  console.log(total + (total * tax) + (total * tip));
}
// In this case 0.15 will be bound to the tip
calculatePrice(100, undefined, 0.15)
```



It works, but it's not very nice, how to improve it?

With **destructuring** we can write this:

```
function calculatePrice({
  total = 0,
  tax = 0.1,
  tip = 0.05} = {} ){
  console.log(total + (total * tax) + (total * tip));
}

const bill = calculatePrice({ tip: 0.15, total:150 });
// 187.5
```



We made the argument of our function an object. When calling the function, we don't even have to worry about the order of the parameters because they are matched based on their key.

In the example above, the default value for *tip* was 0.05, and we overwrote it with 0.15. But we didn't give a value to tax which remained the default 0.1.

Notice this detail:

```
{
  total = 0,
  tax = 0.1,
  tip = 0.05
} = {}
```

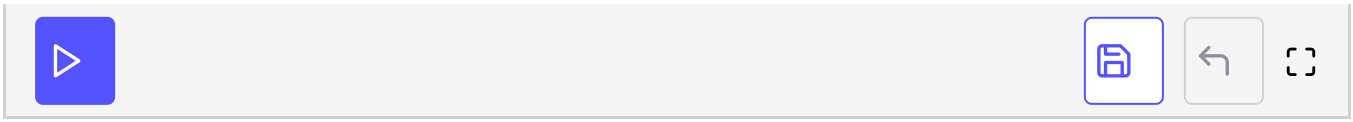
If we don't default our argument object to an empty object, and we were to try

and run `calculatePrice()`, we would get:

```
Cannot destructure property `total` of 'undefined' or 'null'.
```

By writing `= {}` we default our argument to an `Object` and no matter what argument we pass in the function, it will be an `Object`:

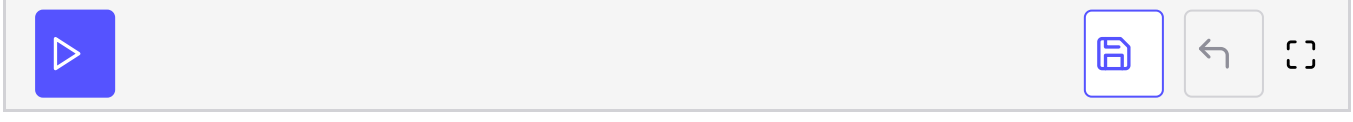
```
function calculatePrice({
  total = 0,
  tax = 0.1,
  tip = 0.05} = {} ){
  console.log(total + (total * tax) + (total * tip));
}

calculatePrice({});
// 0
calculatePrice();
// 0
calculatePrice(undefined)
// 0
```



No matter what we passed, the argument was defaulted to an `Object` which had three default properties of total, tax and tip.

If we hadn't had to do that, look at that what those examples would have returned us:

```
function calculatePrice({
  total = 0,
  tax = 0.1,
  tip = 0.05}){
  return total + (total * tax) + (total * tip);
}
calculatePrice({});
// cannot destructure property `total` of 'undefined' or 'null'.
calculatePrice();
// cannot destructure property `total` of 'undefined' or 'null'.
calculatePrice(undefined)
// cannot destructure property `total` of 'undefined' or 'null'.
```



The code above will stop executing after the first error, to see all of them comment out each function call after looking at the log. You will see that all

three scenarios gave the same error.

Don’t worry about destructuring, since we’ll be covering it in its own lesson.

Coming up next, we’ll test our knowledge and take and a coding challenge to cement the concepts covered in this lesson.