SEP 09 2016

Let's look at a couple more JavaScript arrow function examples.

This is something that comes up often where I'm building an application, and I don't necessary have the data in the right format that I need.

I'm building a website for a bunch of racers, where the winners have been supplied to me in an array of strings in the order that they placed, and then they also gave me a variable, which includes the race that they won.

JavaScript

```
const race = '100m Dash';
const winners = ['Hunter Gath', 'Singa Song', 'Imda Bos'];
```

So Hunter Gath came in first, Singa Song came in second, and Imda Bos came in third.

Ideally for me, I would have an object with `name`, with the name of the racer, `place` with the place that they came in, and we can use the race variable to say what they won. Something like this:

JavaScript

```
{
    name: 'Wes Bos',
    place: 1,
    race: race
}
```

How would we do that? Well, we can actually use `.map()` again for that. By the way, `map` is not the only function that arrow functions work with. Arrow functions work with anything, and they're particularly useful in these callback situations.

JavaScript

```
const win = winners.map((winner, i) => {name: winner, race: race, place: i})
```

We're going to loop over each winner and return an object for each of the winners here. Our function passes two arguments, the `winner` string and `i`, for our index, and we're going to return our `name`, `race`, and `place` that the person came in.

Arrow Function Implicit Return an Object Literal

That should work, where we've got this object here, but it doesn't work, it throws a syntax error: `Unexpected token :`

What's wrong here? Well, earlier we told you that if you take the curly brackets off it's an implicit `return`. How do you implicit `return` curly

brackets when you mean for it to be an **object literal**, not the actual function block?

What we can do is, you just put parentheses around the object literal so that the arrow function knows it's an object and *not* a a function block.

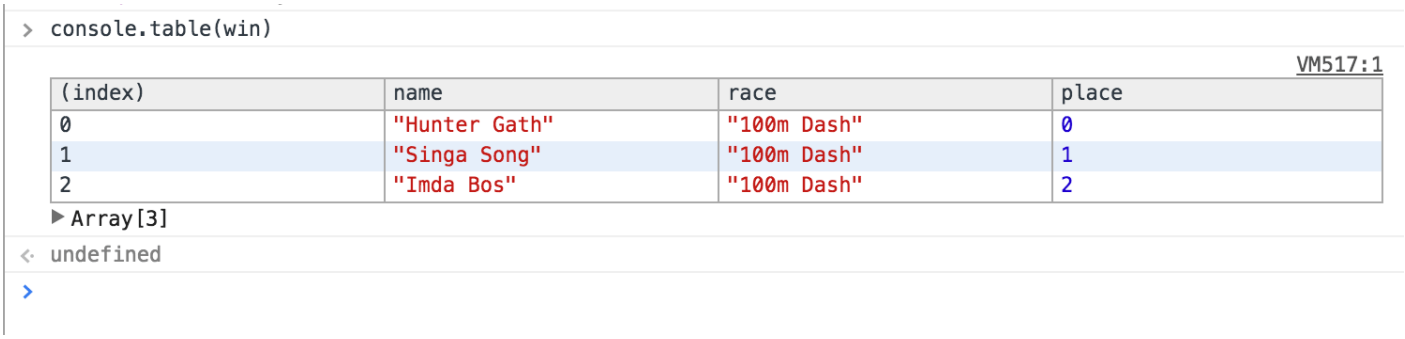
JavaScript

```
const win = winners.map((winner, i) => ({name: winner, race: race, place: i +
```

These parentheses show that you're actually going to return an object literal, and these curly brackets aren't for the actual function block. So `name` is `winner`, `race` is the `race` variable we have, and `place` is going to be set to `i + 1`, our index, plus one so that we aren't zero-based.

Viewing our Data

Now we'll take a look at our data.



A little hot tip, you refresh. You can type `win` in the console to see all your objects, but then you got to do all this work on opening them up. What

you can actually do is use `console.table(win)` and you get a sweet-looking table that will show you all the information.

Then the other thing, we haven't learned about this yet, but `race: race` in our object looks a little bit redundant. A new feature of ES6 is you can just use `race`, which is the same thing as saying, `race: race` Just say the variable `race` and the property name `race` is all named `race`. It's going to do it just for us.

JavaScript

```
const win = winners.map((winner, i) => ({name: winner, race, place: i + 1}))
```

There we go, so that's an example of doing an implicit return with an object literal.

Arrow Function Filtering Example

There's another example here where we poll the people in the room for ages.

JavaScript

```
const ages = [23, 62, 45, 234, 2, 62, 234, 62, 34];
```

We want to filter this list for people who are older than 60 years old. Normally you'd do something like:

JavaScript

```
const old = ages.filter(age => if(age > 60))...
```

Since we can pass it a condition that goes to true or false, and filter will return if it's true and it won't return if it's false, we can just say, "Age is greater or equal to 60."

JavaScript

```
const old = ages.filter(age => age >= 60);
```

That looks like a bit funky because you got an arrow and a greater than, but what that will do is, if the age is greater or equal to 60, it will return `true` and thus be put into the `old` array.

That age is going to be put back into this new old array, and we can use `console.log(old)` to return our filtered array.

This entry was posted in ES6, JavaScript. Bookmark the permalink.

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.

Follow @wesbos 127K followers