# Javascript Concept : React Course

## Let & Const

const. Let and const are different ways of creating variables.

some variables never really change,

They are so-called constants.

Technically though, they are.

They are all just variables because javascript only knows var. With ES6, a version of javascript, two different

keywords were introduced, let and const. Var still works but you're highly encouraged to use let and const

Let is

so to say the new var, you use it for variable values. Let and const are all to changing something behind the

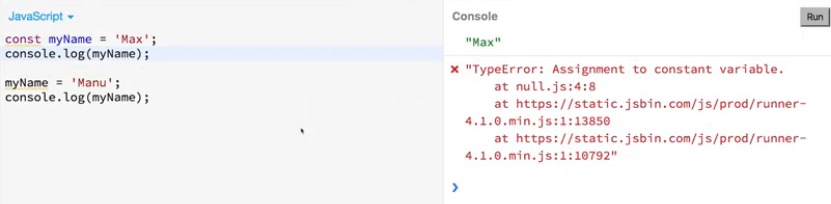
scenes about the scope of variables.

But the most important takeaway here is use let if you want to create a variable that really is variable.

Use const if you plan on creating a constant value, so something which you only assign once and never

change.it is never going to receive a new value.





You can say const should never get a new value and if you then accidentally write some code where you

reassign it you get the error and hence a chance to improve your code.

## Arrow function

That's a different syntax for creating Javascript functions.

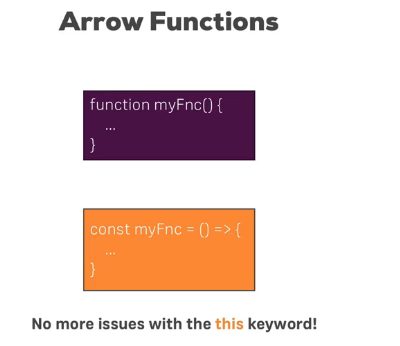
A normal javascript function of course looks like this.

With the function keyword. You might also note a syntax where you have var myFunc equals function.

Now an arrow function looks like this.

Here I'm storing it in a constant and then on the right side of the equals sign.

That's the arrow function syntax.



The arrow function snyntax is a bit shorter than the normal syntax since it omits the function keyword

and it also and that is the great benefit.

**Solves a lot of the issues you often had with the This keyword in javascript.**

**If you've worked with javascript a bit, you probably know that the this keyword doesn't always refer to what you might have expected it to refer to during you writing your code.**

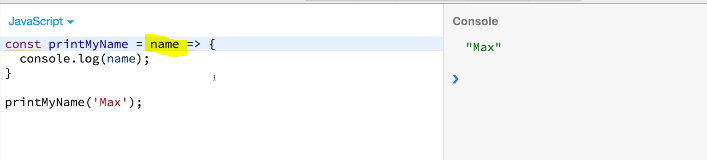
**When you use this inside an arrow function it will always keep its context and not change it surprisingly on runtime.**

**Example -** 

**Note - If you only receive one argument as we do here, you can also use a shortcut of omitting**

**the parentheses around it.**

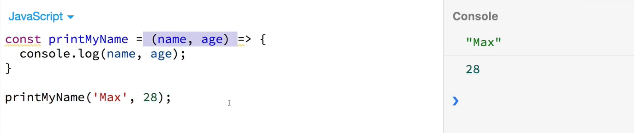
**That's only valid for exactly one argument though, not for more and not for less.**



**Note - If you had a function which receives no arguments.**

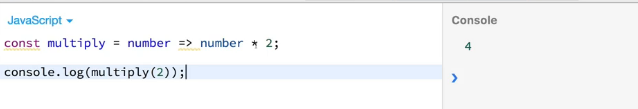
**You need to pass an empty pair of parentheses**

**Note -**  if you have more than one argument. You also need parentheses.



**Note -** If you have this case where all you do in your function body is return and you have no other code in there you can omit the curly braces and write this in one line and then you also have to omit the return keyword.

This is a very short version of writing this function.It gets a bit shorter if we take advantage of the shortcut of removing the parentheses around the single argument.



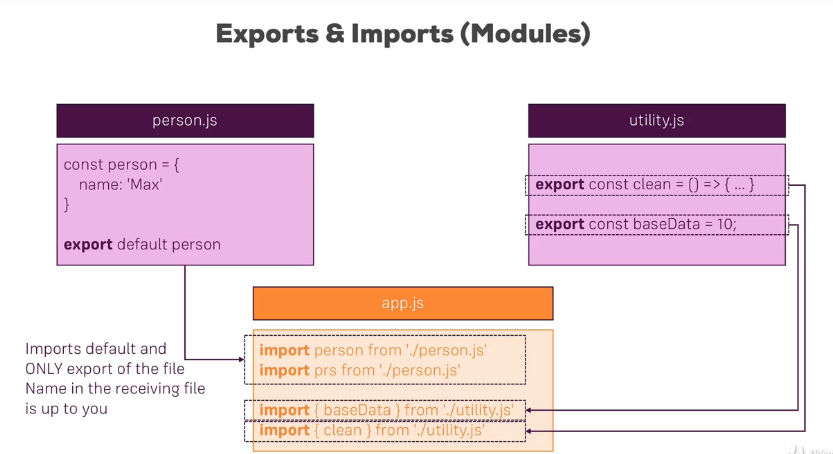
## Exports and Imports (Module)

Another feature which next generation javascript offers is about writing modular code.

so javascript code you split up over multiple files.

**And obviously we already can split our code over multiple files. We just have to import them in the correct order in our html files.**

**The idea behind export and import statements and so-called modules is that inside of a javascript file we can import content from another file so that the javascript files themselves know their dependencies.**



We have one file, person.js and there we have this constant person which is stored as a javascript object and then this is the interesting part.

We export this the **default** keyword this is a special keyword marking this as the default export of this file and we can then import this somewhere else

person.js as you see uses the default keyword, the

default keyword simply means if we just import something from that file it will always be our default export.

So in this case the person constant therefore in import person from person.js we can name person whatever we want, person or prs doesn't matter.

It always refers to the thing you marked as the default with the default keyword.

**For utility.js it's a bit different. There We import from two different constants and therefore the import syntax uses the curly braces to explicitly target specific things from that file.**

These are so-called named exports because we import the stuff by its name.

We import the clean constant by its name and we import baseData by its name because we didn't mark anything as the default.

So for javascript to know what exactly we're pointing to, we need to give it the exact name and the name goes between curly braces.

**By the way, you could also write this as one import statement with baseData comma clean or the other way around.**

with all these next generation javascript features, it won't run like this in all browsers.

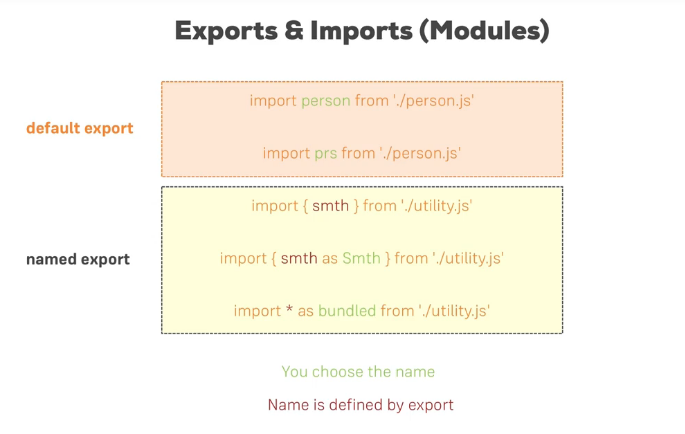
Not even the most modern browsers support all the features

So as of now we need to set up our project which in the

end just compiles all these next generation javascript features to current gen javascript features

so that we as a developer can use the next generation javascript without us shipping code that runs.

You might also see some variations



If you have a named export you actually have to use the exact name defined in the file with the export keyword.

Still what you can do.You can assign an alias which you then again can freely choose in the file you are importing it with the as keyword or if you have multiple named exports in a file.

**You can import everything with this \* special character and then assign an alias and bundled.**

**In this case would be a javascript object which exposes all constants and whatever you export in the other file as properties so that you simply have bundled.baseData, bundled.clean to access the export of things.**

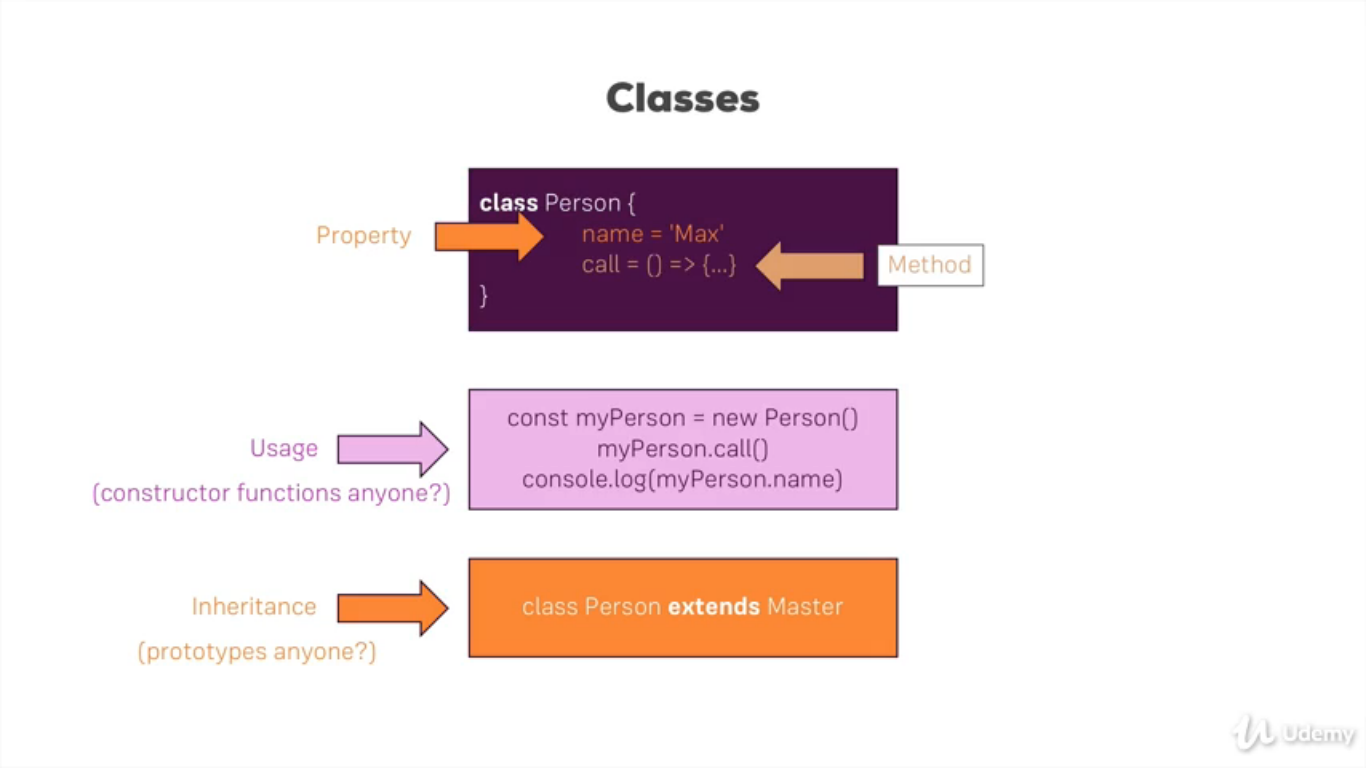
## Classes , Properties and methods

Classes are essentially blueprints for objects.

In our case here for javascript objects. A class is created with the class keyword and a class can have

both properties and methods.

Methods are simply functions attached to classes where properties are variables attached to classes.



a class is instantiated like this with the new keyword.

classes are kind of a more convenient way of

creating constructor functions so you create javascript objects with classes as blueprints.

classes also support inheritance.

Which means you have another class which you inherit from taking all its properties and methods and

potentially adding new properties and methods.



in its simplest form a property is added by adding a

constructor that is a default function method.

You can add to any class which will be executed whenever you instantiate the class. And then there we can now set up properties with the this keyword and we could write this name equals max

the method is created just with the name of the method parentheses and then curly braces.

Now we can use this class to store an instance in a constant with new person and then we can execute person.

classes can also inherit.

Now we inherit this property and this method printGender and we can use both on the person as well

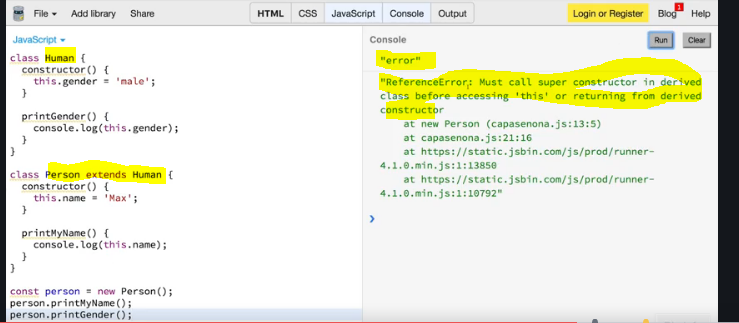
We'll get an error that we must call the super constructor in the derived class and that's important.

If you are extending another class and you are implementing the constructor which you don't have to.

But if you are then you have to add super this special super method in the constructor.

It's a keyword and it simply executes the parent constructor to which you of course have to to correct

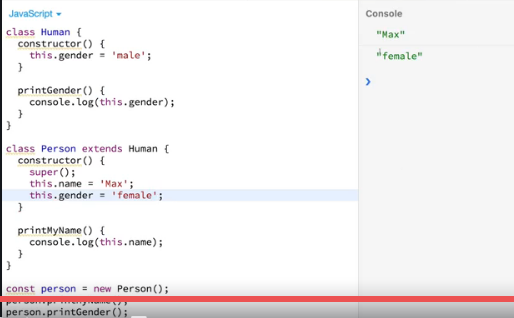
to initialize the parent class.



So now you hit clear and run you'll see Max and male and obviously you could now go into your person

class and still set gender there to female which is not 100 percent correct here but that's just to show case

that this works.



so these are classes and classes are used by React to create its components.

At least this is one of the two ways of creating components.

classes are

just blueprints for javascript objects and are very comparable to constructor functions where inheritance is comparable to prototypes.

## Spread and Rest Operator

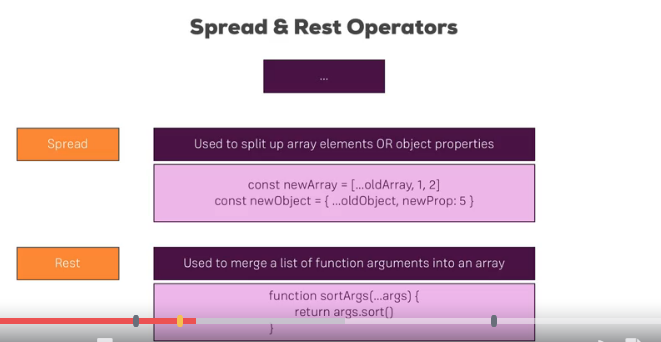
Actually it's only one operator three dots(…)

Now if we call it spread or rest depends on where we use it,

the spread operator is used to split up array elements or object properties.

 the rest operator is the same operator but used differently, here it's used to merge a list of function arguments into an array.

And this shows us where we use it. We use it in a function argument list.



Spread operator example -

For example if we have an old array and we want to add all the elements from that old array to a new

array and additionally add a 1 and a 2 element this first syntax would be what we use, three dots in

front of old array will simply pull out all the elements and add it to the new array which we created

with square brackets.

And of course then we can add more elements to it.

So if we just use the normal syntax with the square brackets to create an array, the same for the object.

We create a new object with curly braces with the new prop but then we also have dot dot dot.

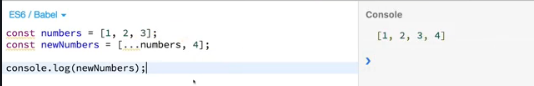
Old object to pull out all the properties of the old object and their values and add them as key value

pairs to the new object as a side note if the old object.

Also had a new property.

It would be overwtitten by a new prop 5 here.

So our own property takes precedence. This is the spread operator.



**If we were to do that without the dots it would be included as one element inside the new element.**



**Spread operator on Objects**



**That’s the spread operator on both arrays and objects to for example conveniently copy arrays or add properties to an object whilst safely copying that old object.**

### **Note :**

**here the new array/object pointing to new memory location not the old arrays/object memory location.**

**So if we use Spread operator then we get new memory allocation**

Rest operator example –

Here is an example.

sortArgs receives an unlimited amount of arguments.

So one argument, two, three or whatever, with dot dot dot we only write one argument args but we may actually

receive more than one and they will all be merged together into an array.

So then we can apply array methods to our argument list or do whatever we want to do with our conveniently

stored arguments.

Example –



we call the built-in filter method which is available on arrays

and keep in mind the dots here are used as a rest operator and that merges the arguments into an array.

So we can use array methods like filter, filter will execute a function on every element in the passed in

array.

So here we would get our element and then we could use the inline arrow function to simply say return true or false.

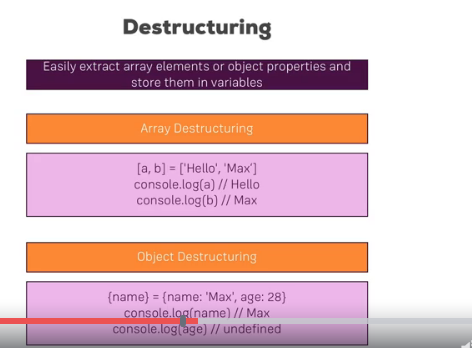
## Destructuring

**Destructuring allows you to easily extract array elements or object properties and store them in variables.**

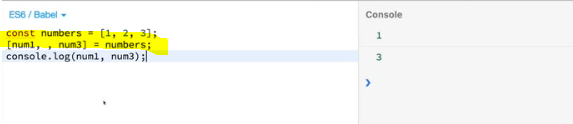
**When you first hear about that it might sound like the exact thing the spread operator does but actually it's a different thing.**

Spread takes out all elements all properties and distributes them in a new array or object or wherever you're using it.

**Destructuring allows you to pull out single elements or properties and store them in variables for arrays and objects.**



Example –



## Reference and Primitive types

They're not next generation javascript but they are features you might have missed or forgotten and

they're super important to keep in mind.

The first feature or concept of javascript I'm talking about is the fact that you have reference and

primitive types.

If I create a number like this, then this is a primitive type.

That means if I create a second number num2 and set it equal to this number then it will actually

create a real copy of number so num2 of course.

Now if I log this, it will also be one.

But it will have copied that value one into num2.

Now numbers, strings, booleans, these are so-called primitive types whenever you reassign or you store

a variable in another variable.

It will copy the value



objects and arrays are reference types.

it will not actually have

copied the person instead.

Person the object is stored in memory and in the constant person we store a pointer to that place in memory.

And if we then assign person to secondPerson that pointer will be copied.

We can see that this is the case if we changed person.name after having it copied.

With that you would expect to print Max here still a person with name Max because we copied person, stored

it in secondPerson and thereafter changed the name.

However if I clear and run you will see name.

Manu here even though I'm printing the secondperson so for secondPerson the name also changed

the reason for it is that it just copied the pointer and points to the exact same object in memory as

person does.

So if we change name on person we automatically change it for secondPerson.

Now that's important.

Keep in mind.



it's the same for arrays.

If you copy in quotation marks.

An array like this.

And you then change an array element.

It will all change in the so-called copied array.

This will become important in React because it can lead to unexpected behaviors.

If you copy objects or arrays like this because you then may manipulate one object in one place in the

app and accidentally manipulate another usage of the same object in another place of the app.

**Therefore we will learn techniques to copy this in an immutable way which means we copy that by really copying the object and not just a pointer for that we can use this spread operator.**

Now we can simply create a new person object here and spread the person properties.

This will pull out the properties and the values of the properties from the object and add it to this

newly created object here and we do create a new one with the curly braces.

Now if I hit clear and run we still print an object with name Max even though we changed the name to

Manu here because now we really created a real copy.



**It's just important to realize and to keep in mind that objects and arrays are reference types.**

**If you reassign them you're copying the pointer not the value.**

**Therefore if you want to do this in a real copy way, you will have to create a new object and just copy the properties and not the entire object.**

## Array functions : Refreshment



Another thing you will see quite a lot in this course are array functions.

We already saw filter a couple of lectures ago.

We also got sort, map and so on.

Let me quickly show you what I mean.

The good old numbers array with 1, 2 and 3.

Now let's say we want to turn this into an array where each number is doubled.

So we have to doubleNumArray, something like that we can use an array function for this.

We can take the numbers array and call map.

Map is a built-in array method.

And there are many of these methods.

I will use quite a lot of them and they're not.

Next generation javascript all these methods work in the same way though they take a function as an

input and this function which is an arrow function here but could be a normal function is then simply

executed on each element in the array here.

So on each element in the numbers array, on 1 and 2 and 3.

So therefore what we get in this arrow function is a number in the end.

But you can name this argument whatever you want here.

We can then simply return something.

And what you have to do in this internal function depends on which area of function you are using.

Check the docs in places like d Mozilla Developer Network docs to learn more about the available array

functions.

So in the map function we have to return the new value we want to turn the old one into so we could

return num \* 2 and since this is executed on every element here.

It will return 2, 4 and 6 and conveniently map all the returns a new array.

So a real new array which is then stored in doubleNumArray.

So now if I output numbers and thereafter doubleNumArray like this and then I'll hit run you'll see

the old one is unchanged but the new one holds double the values and I will explain what these functions

do when we use them in the course.

I just want to bring them to your attention right now.

Explain that they always have this function which gets executed on each element and that they are not

next generation javascript but normal javascript actually, be prepared to meet them.

I will explain what they do when we see them and always feel free to dive into docs like the Mozilla developer network to learn more about them.

### JS Array Functions

JS Array Functions

Not really next-gen JavaScript, but also important: JavaScript array functions like map() , filter() , reduce()  etc.

You'll see me use them quite a bit since a lot of React concepts rely on working with arrays (in immutable ways).

The following page gives a good overview over the various methods you can use on the array prototype - feel free to click through them and refresh your knowledge as required: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array>

Particularly important in this course are:

* map()  => <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map>
* find()  => <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find>
* findIndex()  => <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex>
* filter()  => <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter>
* reduce()  => <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce?v=b>
* concat()  => <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/concat?v=b>
* slice()  => <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice>
* splice()  => <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice>