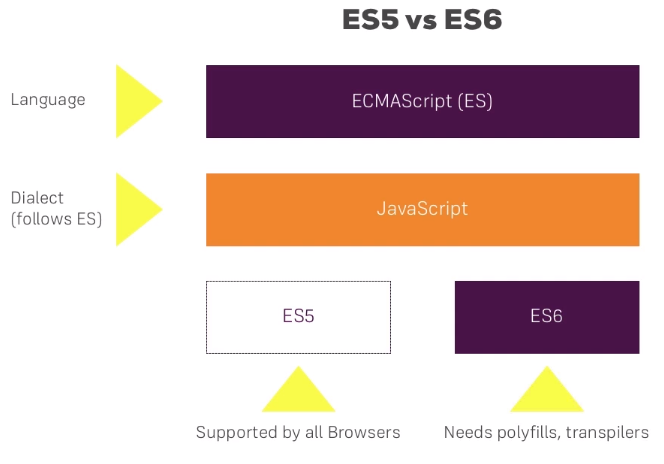
**Section 1: Introduction:**

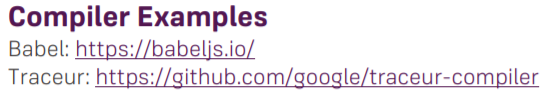
**3. JavaScript Languages - ES6 vs ES5:**

* 
* ES needs polyfills or transpilers to convert it to ES6 because still not all browsers supports all features of ES6.

**5.ES6 Compatibility with Browsers:**

* <https://kangax.github.io/compat-table/es6/>

**6. Using ES6 Today:**

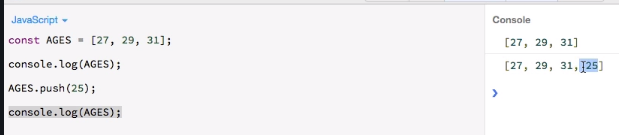
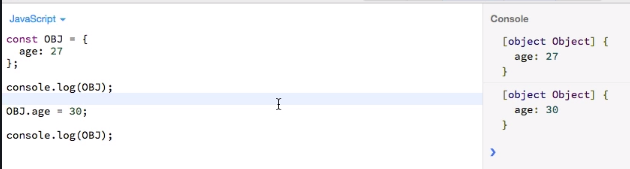
* In order to work with ES6 you basically need three things.
* The first thing is a package or a tool which compiles ES6 code to ES5.
* 
* **The second thing is a Module Loader:**
* Working with modules, so you have a lot of JavaScript files you want to load it dynamically. You need a package to do that because this behavior also isn't supported natively by browsers yet. So, you need the module loader.
* 
* And lastly third thing you need a little server serving your application because even though it is static since you're working with modules those modules need to be served dynamically and therefore a little lightweight server is needed for it.
* 

**Section 2: Syntax Changes & Additions:**

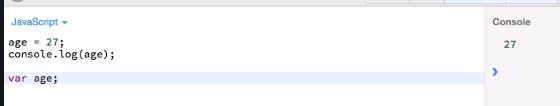
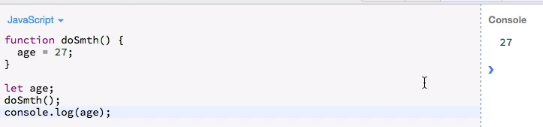
**7. Let & Block Scope:**

* 
* 

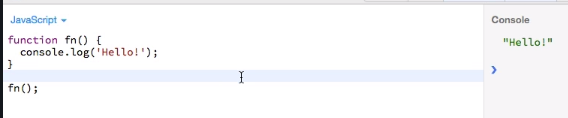
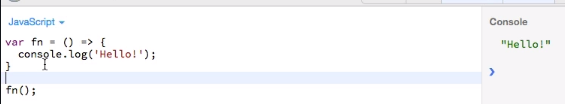
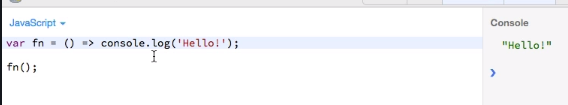
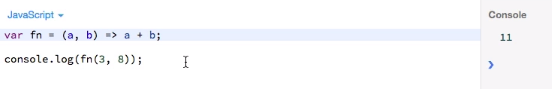
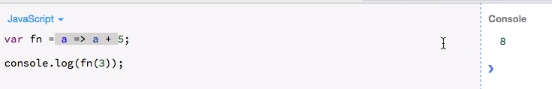
**8. Constants with "const":**

* 
* 

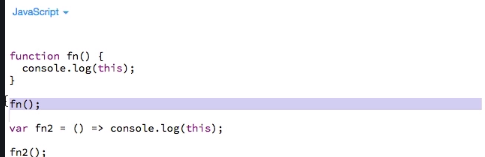
**9. Hoisting in ES6:**

* 
* So, the hoisting of the variables with let or const doesn't work the way it works with var. We have to declare our variable before actually initializing it.
* 
* 

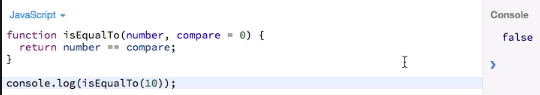
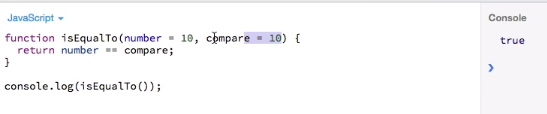
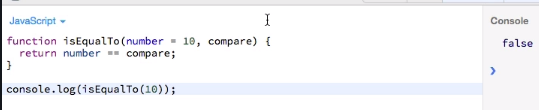
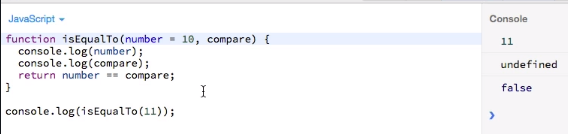
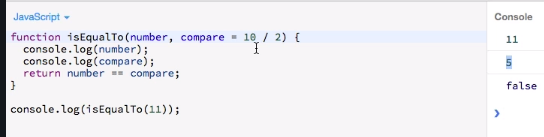
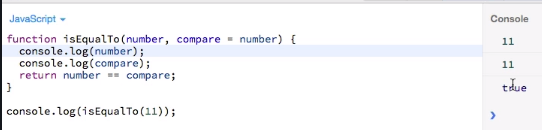
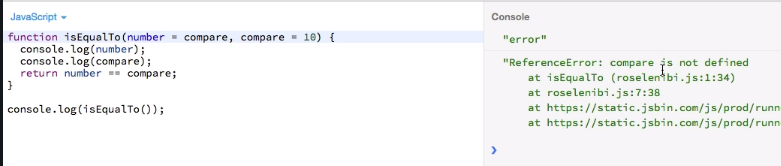
**10. (Fat) Arrow Functions:**

* 
* 
* 
* 
* 
* 
* 
* 

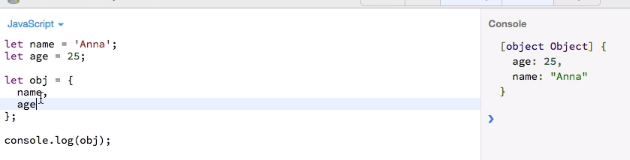
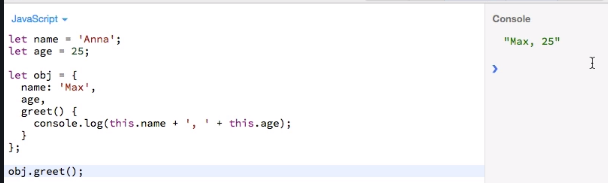
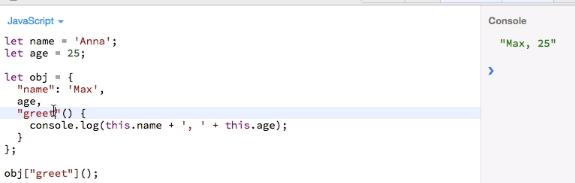
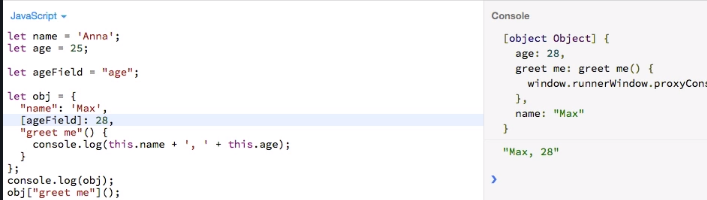
**11. (Fat) Arrow Functions and the "this" Keyword:**

* **Arrow function has lexical scoping.**
* 
* Both this will be referring to global scope(window object).
* **Normal Function call on button click event**: In this case ‘this’ actually refers to what call this function. Here it called on button click, that’s why we got HTMLButtonElemet object.
* 
* **Arrow function call on button click:** here ‘this’ refers to window object.
* 
* You won't need bind or apply or call all those work arounds you use with ES5 to get this to the right context, but it will just keep the context in which arrow function is defined.
* **Do arrow functions have ‘arguments’? will function and prototype objects will be create for the arrow functions.**

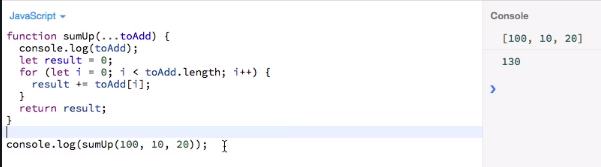
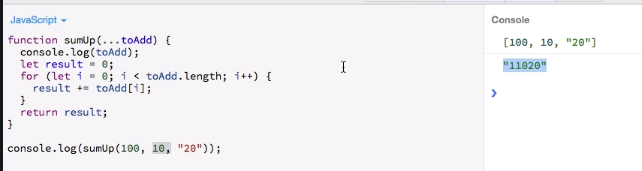
**12. Functions and Default Parameters:**

* 
* 
* 
* 
* 
* 
* 

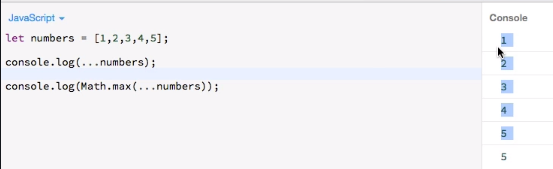
**13. Object Literal Extensions:**

* 
* Well actually the object will take the values from the surrounding variables.
* So, if we're not specifying values here to initialize this object, it will automatically look if it has variables declared before the declaration of this object, which match the property names declared before.
* 
* 
* **Dynamically add property to object literal:**
* 

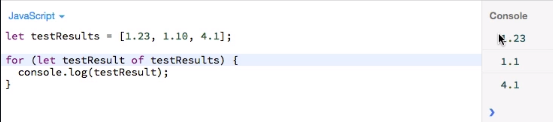
**14. The Rest Operator:**

* 
* 

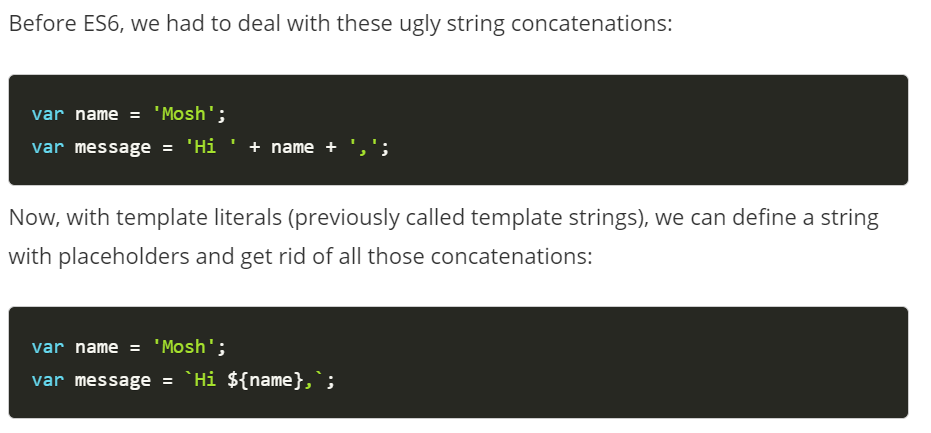
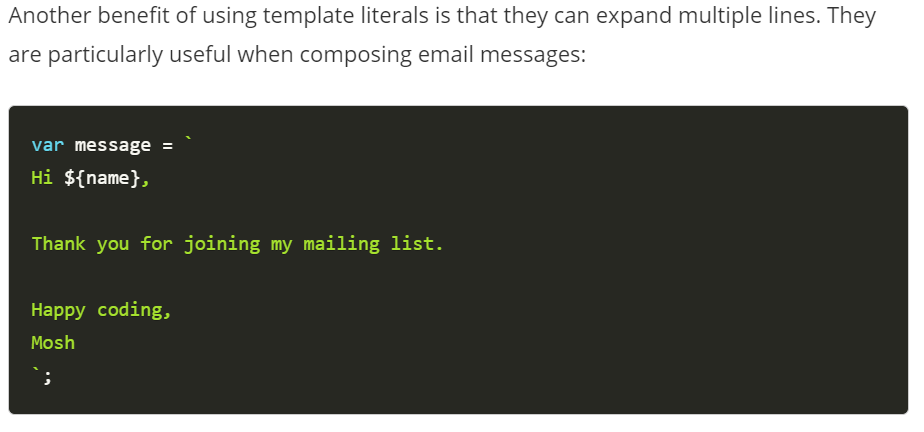
**15. The Spread Operator:**

* 
* Here max method expecting list of arguments not an array.
* 
* Here spread operator spreads an array into list of arguments.
* Spread operator also used for cloning the object and array. It is the alternate of Object.assign().

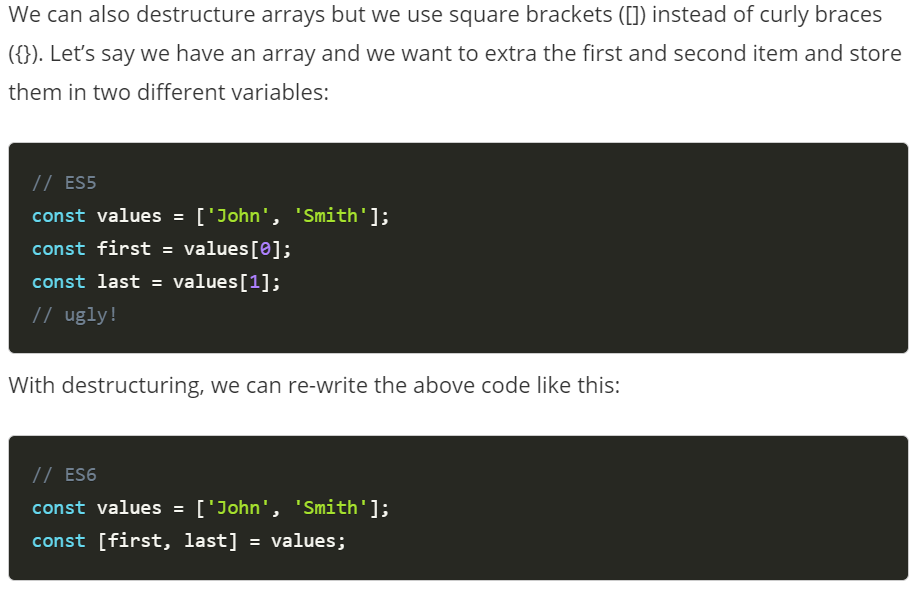
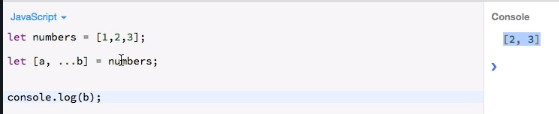
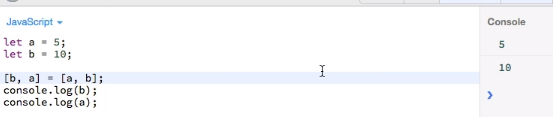
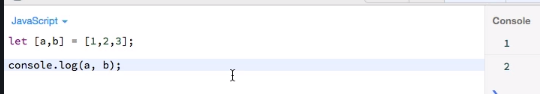
**16. For of loop:**

* 
* This is a shorter syntax and it's perfect for looping through arrays and that is what it's there for you to quickly grab the individual elements off an array and do something with them.

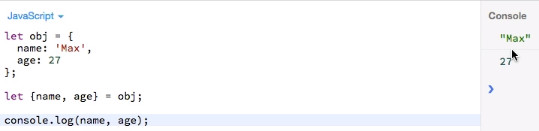
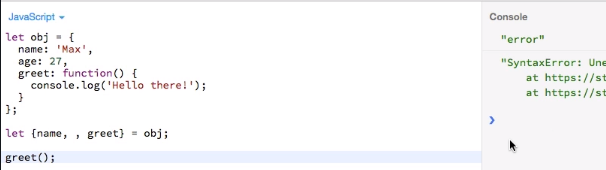
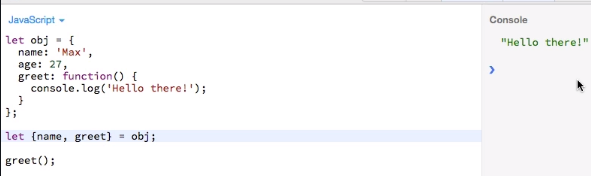
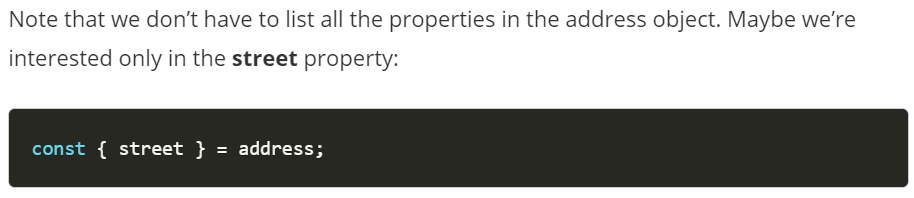
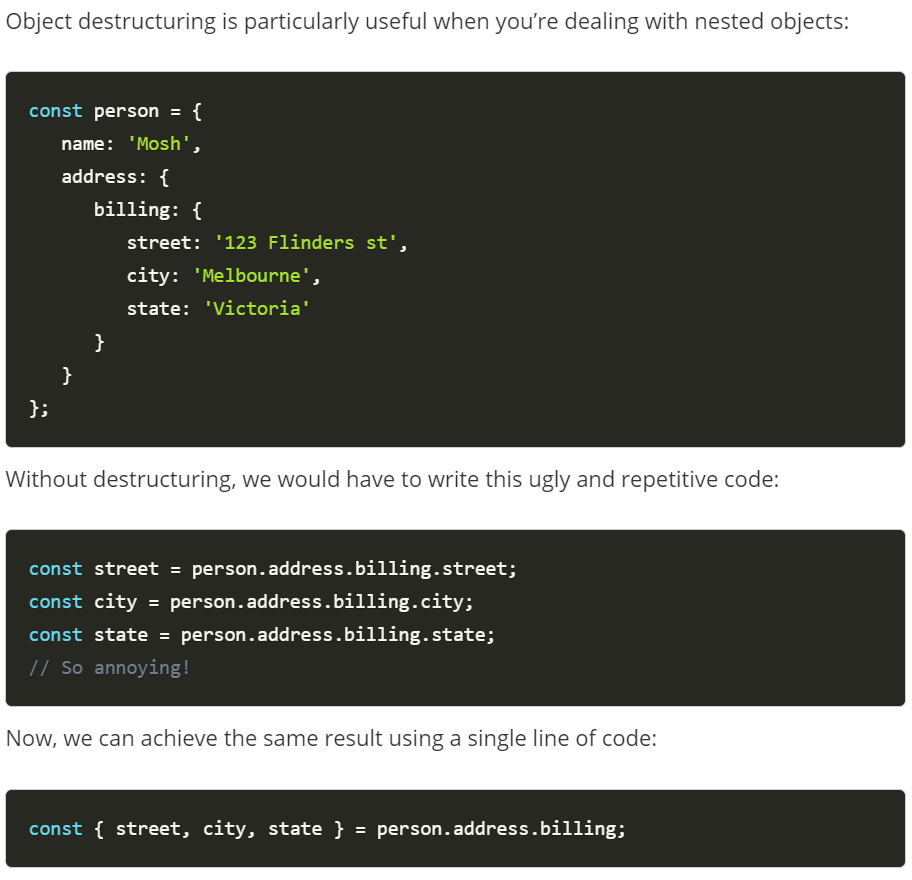
**17. Template Literals:**

* 
* 

**18. Destructuring – Arrays:**

* 
* 
* **With extra element:**
* 
* **With rest parameter:**
* 
* **Using default parameters in Destructuring:**
* 
* 
* **Swapping with Destructuring:**
* 
* **Destructuring specific values not all values:**
* 
* **Immediately while assigning Destructuring the array:**
* 

**19. Destructuring – Objects:**

* 
* 
* we could do this in the array where we have a clear order with an object, we cannot do this. In objects we're Destructuring them by a name not by position like in the array.
* 
* 
* 
* **Object Destructuring:**
* 

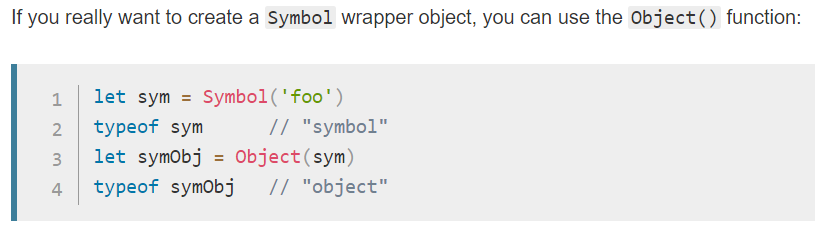
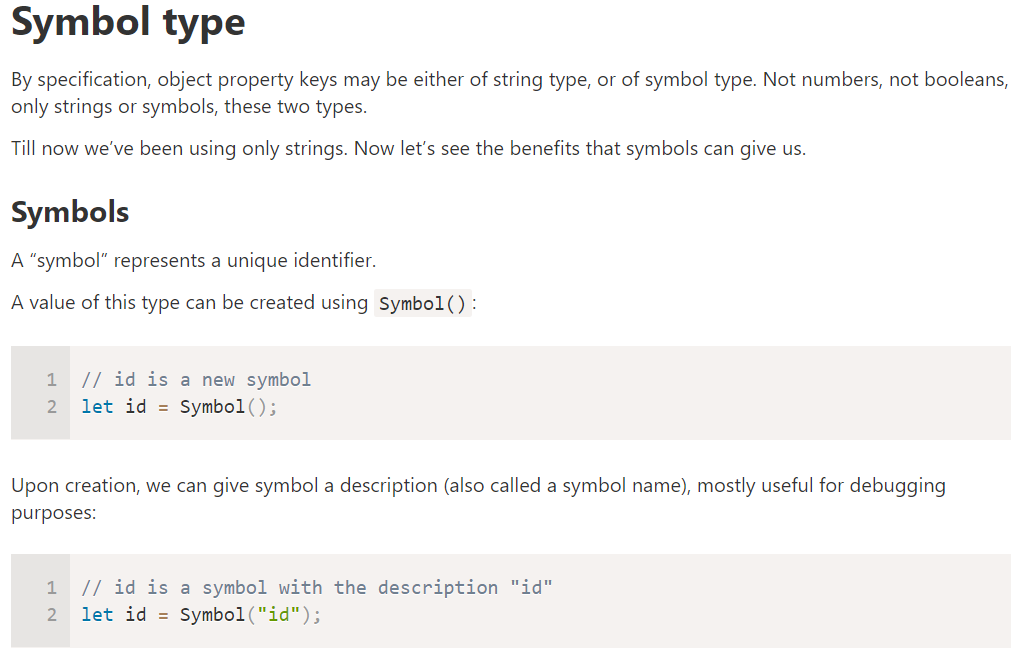
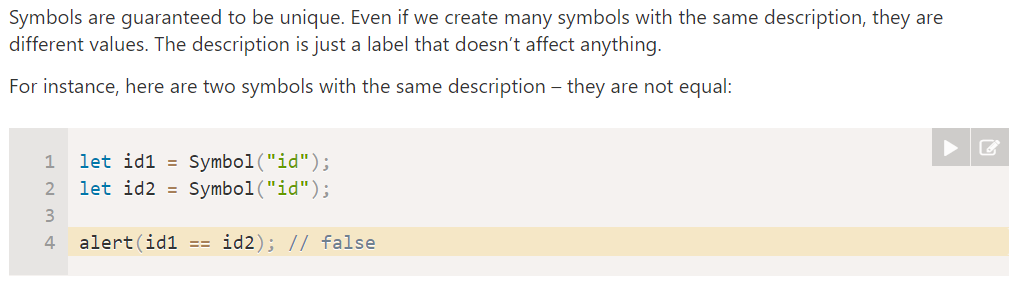
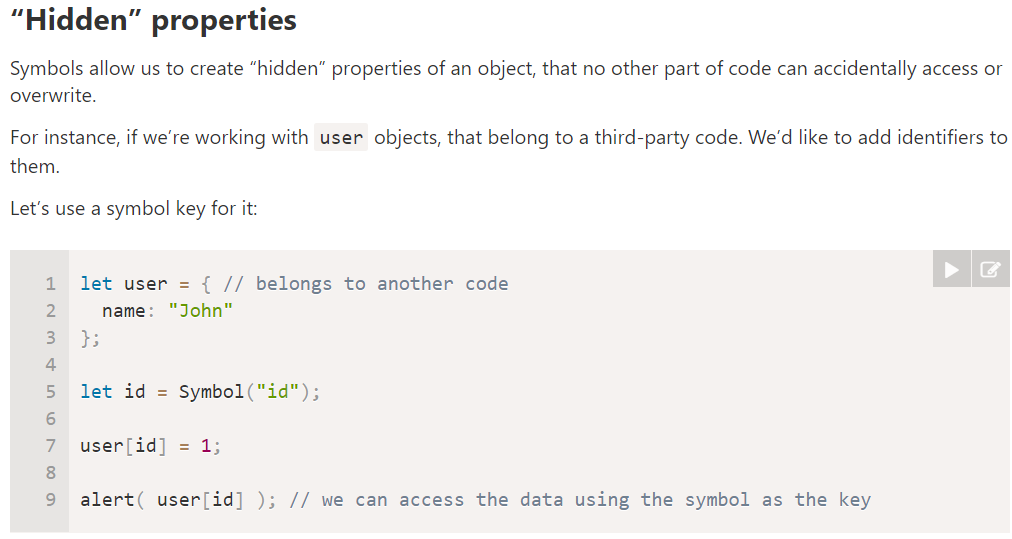
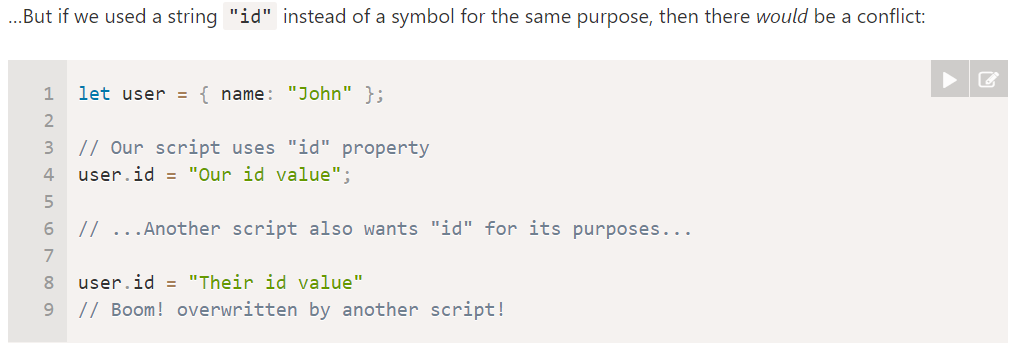
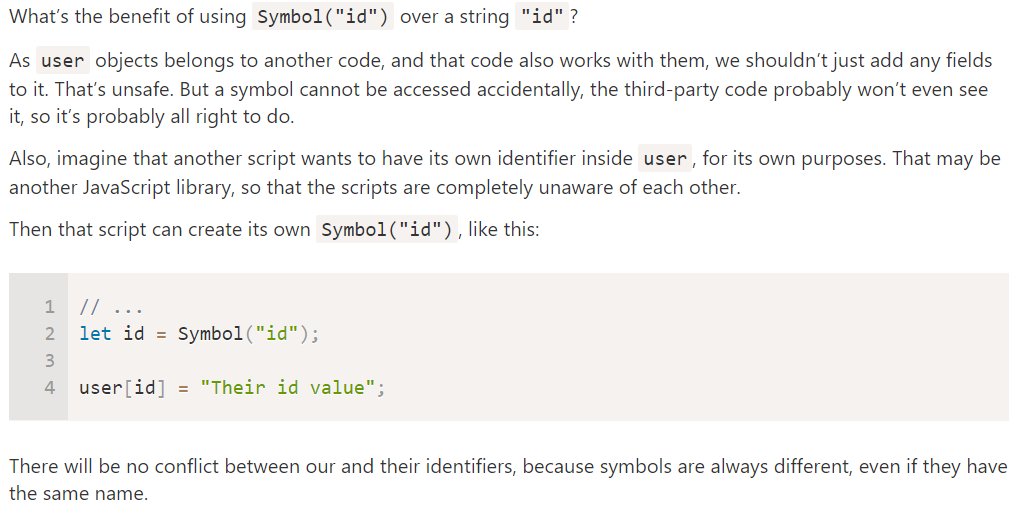
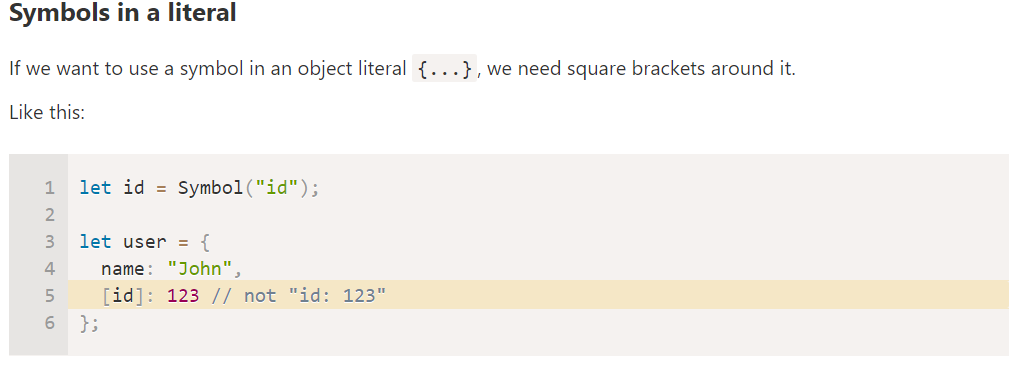
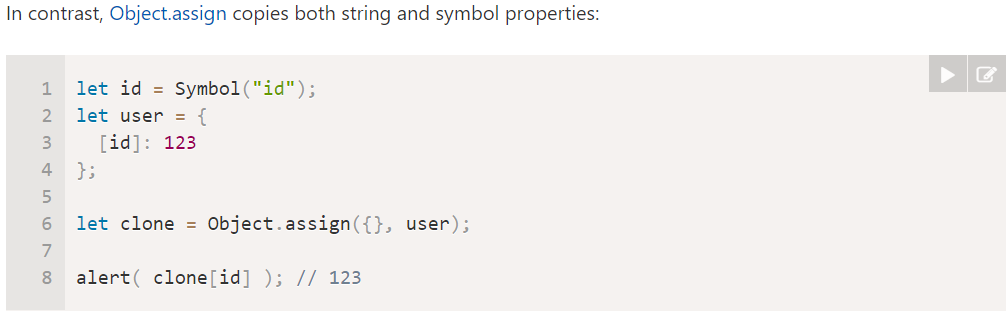
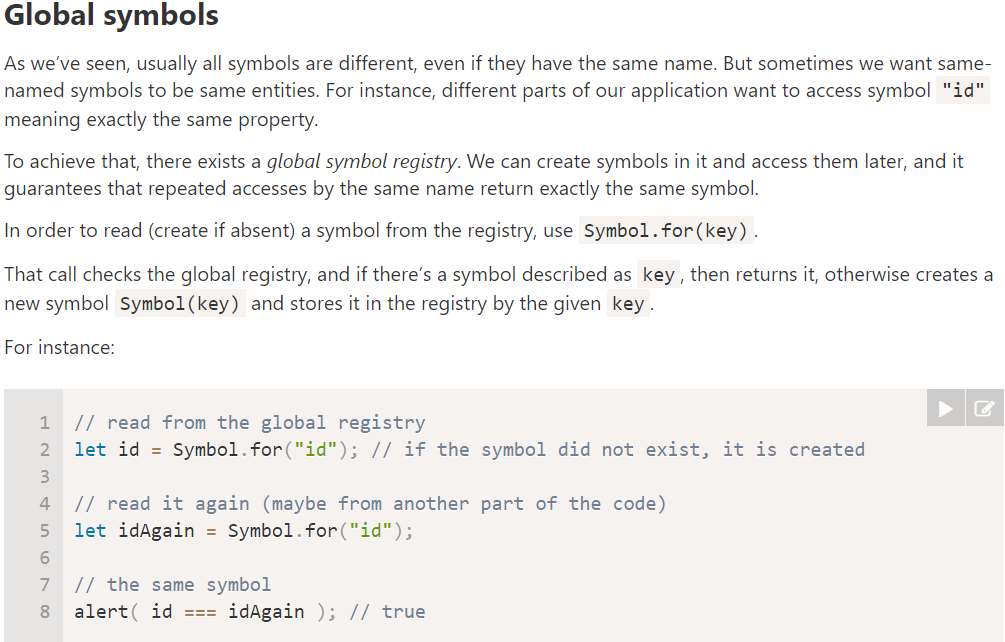
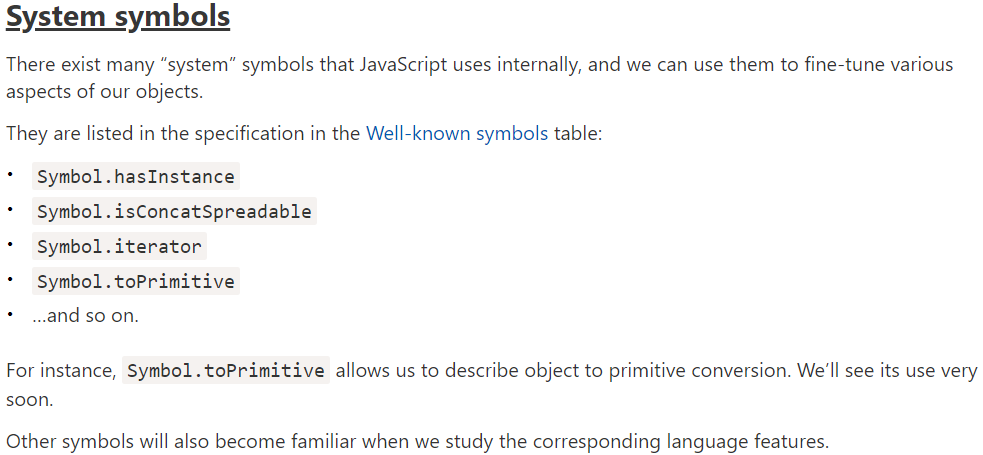
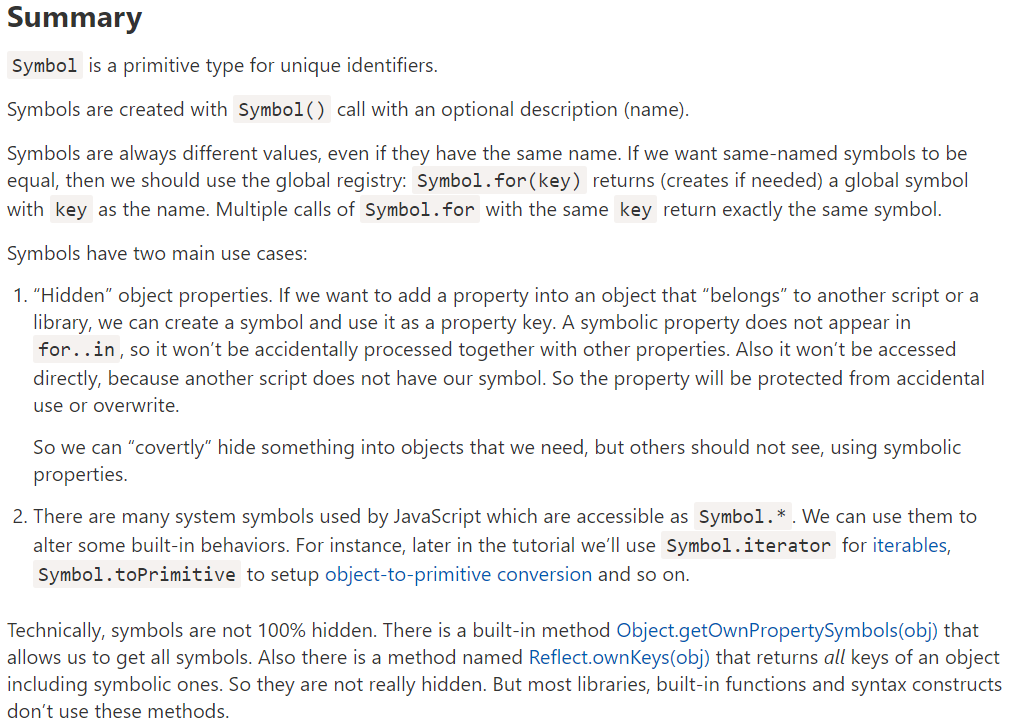
**Section 3: Modules & Classes:**

**Section 4: Symbols:**

**40. Introduction:**

* So, symbols basically are a new primitive type like number, strings or Booleans. And the main thing is they provide a unique identifier.
* Also, symbols are not iterable.

**41. Symbols Basics:**

* It is kind of constructor thing without the new word.
* 
* 
* 
* 
* These symbols do not match because again behind the scenes they stand for unique IDs so are different.
* Now I was already explaining that symbols are great in conjunction with objects.
* To add symbol to our object we need to enclose it into square brackets. Without square brackets it is like adding a property to object not symbol.
* 
* We only see name Max. The symbol is actually not printed out. You also won't see if I use a for loop, but it is still there if I access it explicitly with square brackets notation on the object.
* 
* This is just for meta-programming part. You could store some meta information about this object like created at this time, Add the time stamp if important for your application. And then you could access it if you needed.
* So, this is why symbols are really important and useful tool to add some metaprogramming power to your objects or to your app as a whole.
* Every symbol value returned from Symbol() is unique.  A symbol value may be used as an identifier for object properties.
* 
* From JavaScript.Info:
* 
* 
* 
* 
*  
* 
* 
* 
* 
* 
* 
* 

**42. Shared Symbols:**

**43. Advantages of (unique) IDs / Symbols:**

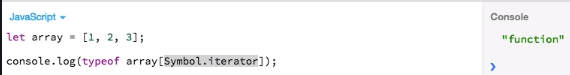
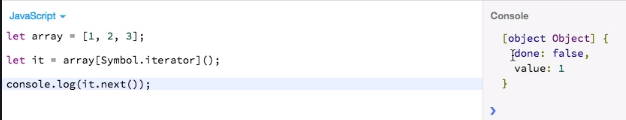
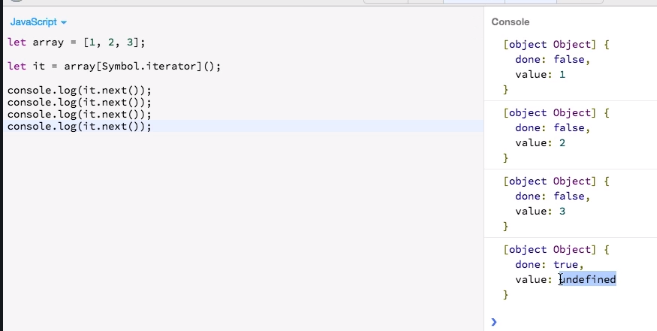
**44. Well-Known Symbols:**

**Section 5: Iterators & Generators:**

**46. Introduction:**

* Iterators are basically all objects that know how a to access values in a collection, one at a time. So, for example an array is such an iterator.
* That means you can loop through it and has a collection of objects and it knows how to output them one after another, means we can loop through.
* Now you can also create your own objects with your own iterator logic.
* A generator is a function which yields different values and especially powerful combinations of course if you use generators and iterators together.

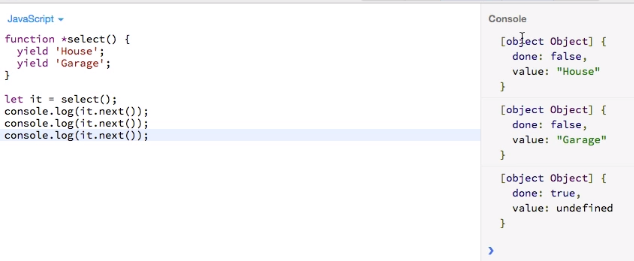
**47. Iterator Basics:**

* So, an array of course is iterable and an object is also iterable if it has a certain well-known symbol built in.
* 
* So, we have an iterator which is a function that is used when we the actual loop through an array.
* Let's call this iterator function from above image:
* 
* It's an object and you can see array iterator here. And actually, it's an object that only has one method, the next method that I can call this method and let's see what we print:
* 
* You see we get a new object which has a ‘done’ property which is set to false and a value of 1. The first value in our array.
* So, let's try again by calling it two times:
* 
* So, you see by calling it multiple times we kind of seem to be stepping through the values of this array.
* 
* So ‘done’ will be only true once it really has exhausted all the values in the collection and therefore, we print three times an object with done false and the individual values.
* And it is only true once we reached the first undefined value here.

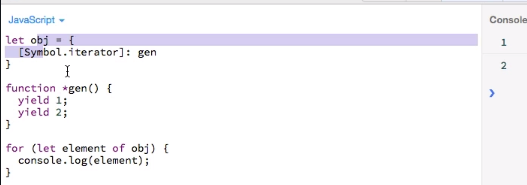
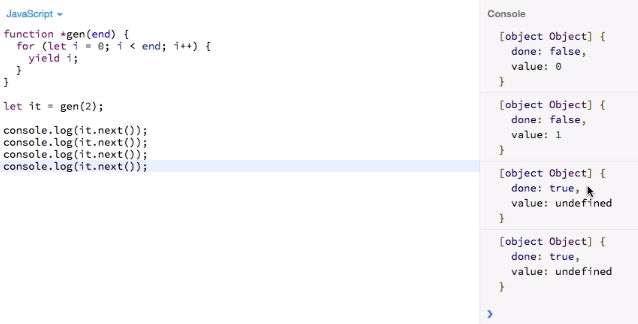
**48. Iterators in Action:**

* We can make any object iterable all we have to do is implement this symbol here. and then we are able to loop through, which will allow us to loop through our own objects.
* **Modified iterator function: looping through next method**
* 
* **Modified iterator function: looping through for of loop**
* 

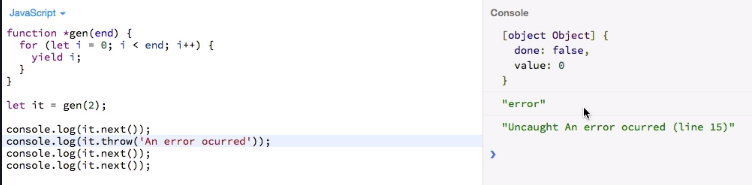
**49. Creating a Custom, Iteratable Object:**

* 
* We just made our personal object Iteratable and it's up to you of course to define what we want to iterate.
* 50. Generators Basics:
* A generator looks like a normal function. The first important thing to make it a generator is to add a star an asterisk. Now you may add this directly in front of the function name. You may add it in between with white spaces around or directly after function.
* Important thing to really make it a generator and to do something is to add the yield keyword.
* **Let's see what actually happens if I run this function:**
* 
* Nothing happened. And here comes the connection to iterators that by running a generator we actually get back an iterator we get back an object for which we can loop.
* **So, I'll assign a to ‘it’ as function call and call it:**
* 
* It allows us to create a logic state and a function to yield different values and that we can use an iterator that will loop through those values.

**51. Generators in Action:**

* <https://javascript.info/generators>
* 
* Now we're using a generator and iterator will return us to loop through our object. It's much easier for us to manage state.
* And also imagine the possibilities it offers us. Because with our generator we might also do some let's say asynchronous task and yield those results step by step like fetching something from a server or anything like that.
* And with our iterator we could then use them and take advantage of that step by step approach instead of having a function which runs from start to finish and then gives result.
* **Passing arguments in generator:**
* 

**52. Controlling Iterators with throw and return:**

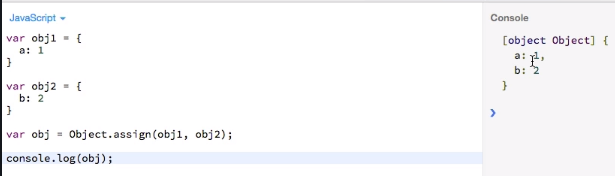
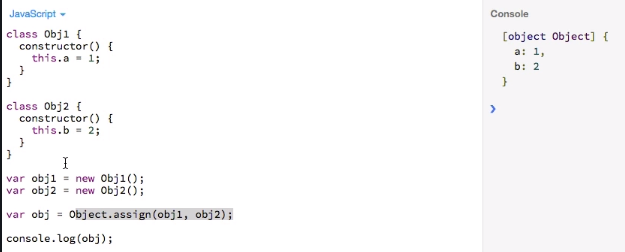
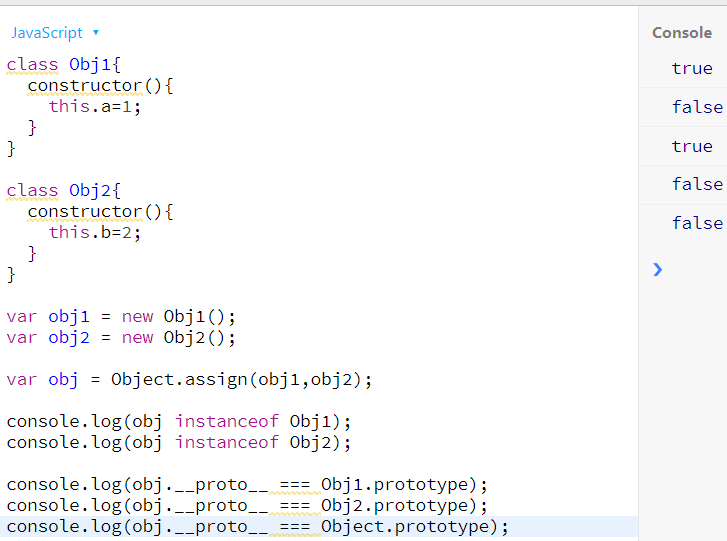
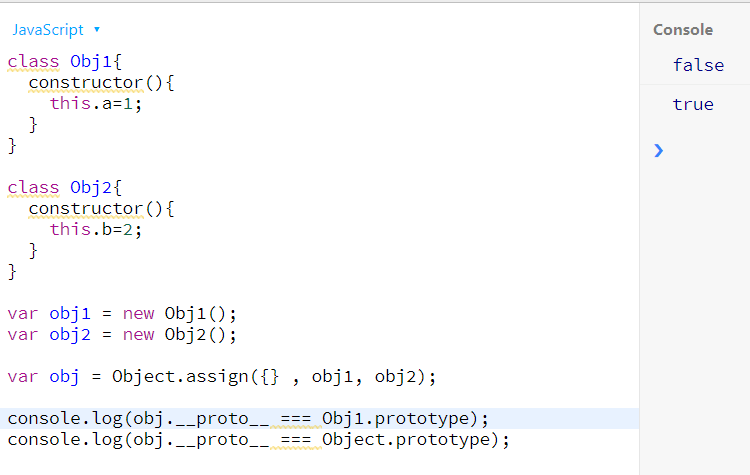
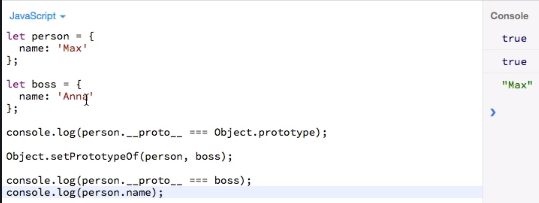
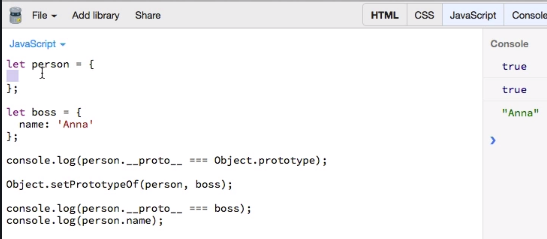
* Throwing an error from iterator:
* 
* **Handling error in iterator:**
* 
* **Return keyword in iterator:**
* 

**Section 6: Promises:**

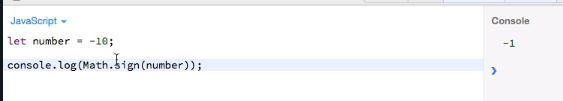
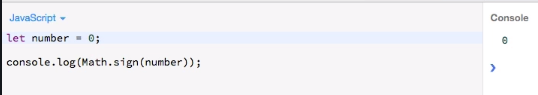
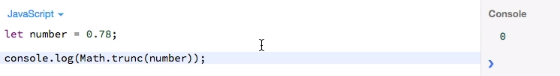
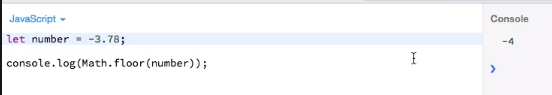
* They are a useful object a helper to work with asynchronous tasks

**Section 7: Extensions of Built-in Objects:**

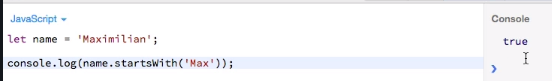
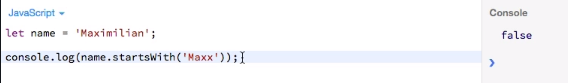
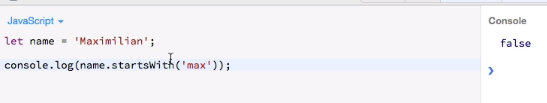
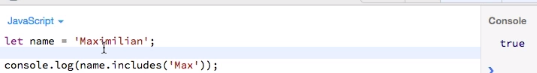
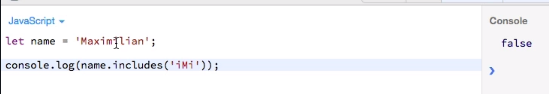
**63. The Object:**

* **Object.assign(): as merging objects.**
* 
* Well we get a new object which is now the combination of the two objects passed to assign.
* **An interesting question is what will happen if we merge two objects which for example have different constructors and they are different prototypes so quickly set up such an example:**
* 
* But of course, the interesting question is what the prototype of that object is:
* Because the prototype of obj1 will be Obj1.prototype and obj2 will have Obj2.prototype. Also, object one is an instance of Obj1 and obj2 is an instance of Obj2.
* 
* So, this assign method merge given objects into the target object means into obj1 in our case, without cloning it. I few change obj then it will impact obj1 also.
* **Merging in empty object:**
* 
* **setPrototypeOf() method:**
* With Object.create() method we would set the prototype at a time we create an object. Now with this new method we can change it after the object is created.
* 
* 

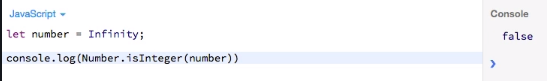
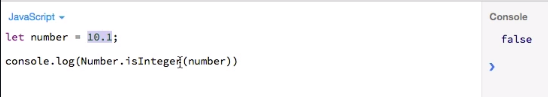
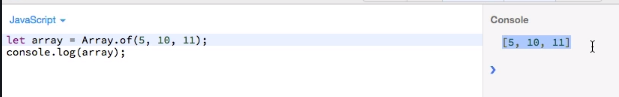
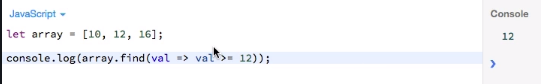
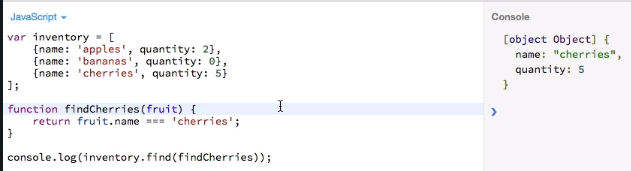
64. The Math Object:

* **sign() method:**
* 
* 
* 
* 
* **trunc() method:**
*  
* **Diff between Math.floor() and Math.trunc():**
* 
* 

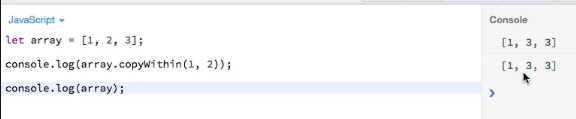
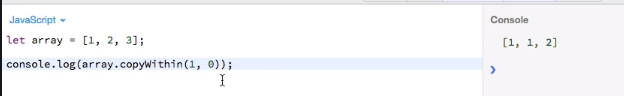
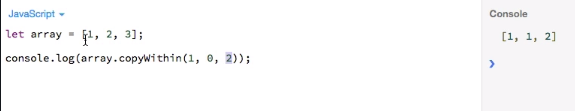
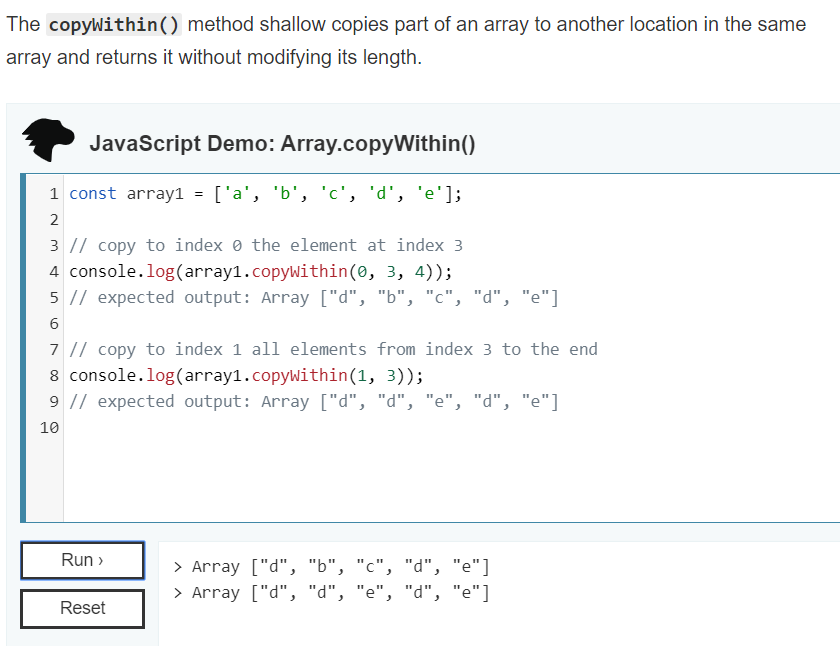
**65. Strings**

* **startsWith(): this is case sensitive.**
* 
* 
* 
* **endsWith(): this is case sensitive.**
* 
* 
* **Includes():**
* 
* 
* 

**66. The Number Object:**

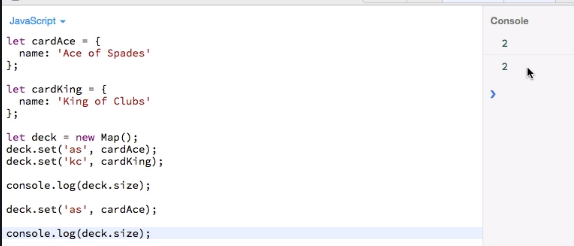
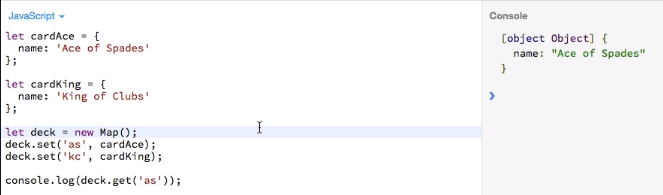
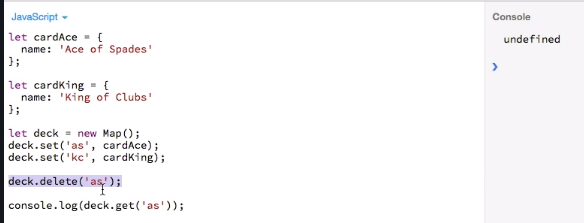
* **isNaN():**
* 
* 
* **isFinite():**
* 
* 
* **isInteger():**
* 
*  
* **67. Arrays (1/2)**
* 
* **Array.of(): Creating a new array by passing a list of arguments.**
* 
* **Array.from(): kind of applying a map function on the passed array in the method and it return new array.**
* 
* **fill():**
* 
* 
* 
* **Find(): Returns first element matching with the passed arrow function.**
* 
* 

**68. Arrays (2/2):**

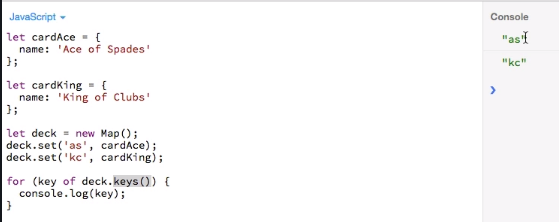
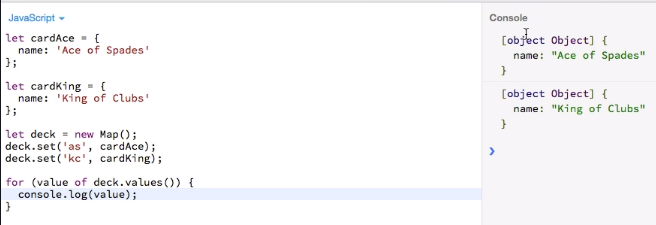
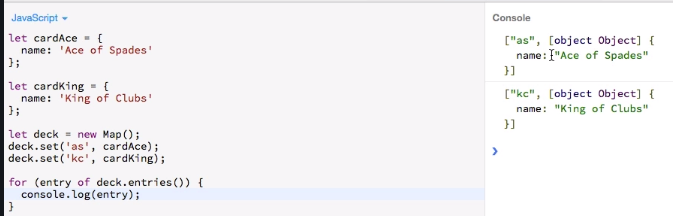
* **copyWithin():**
* 
* 
* 
* 
* 
* array.entries():
* 
* 

**Section 8: Maps & Sets**

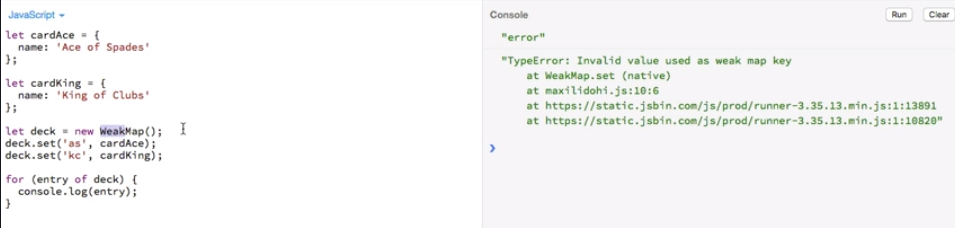
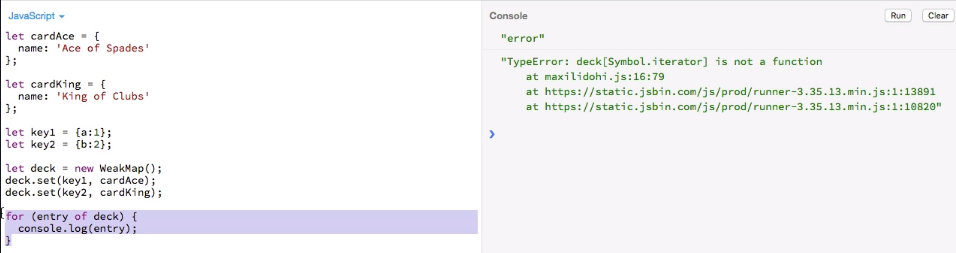
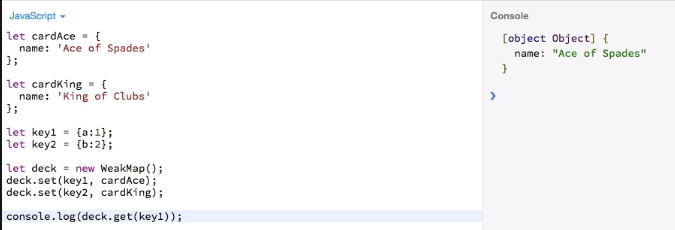
**72. Maps - Creation & Adding Items:**

* 
* 
* 
* 
* **Clear(): will clear whole map.**
* 

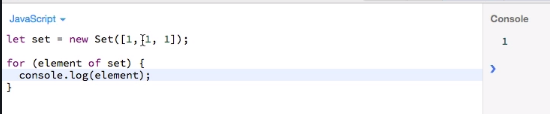
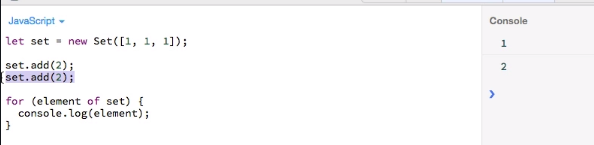
**74. Maps - Looping through Maps:**

* 
* 
* 
* 
* **No need to use entries method it is kind of obsolete.**
* 

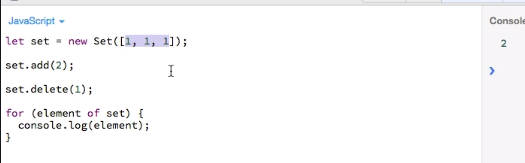
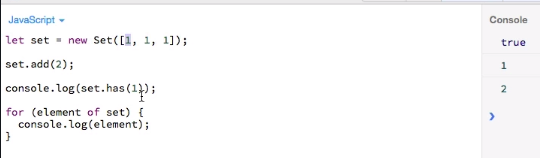
**76. The WeakMap:**

* 
* Now the reason for this error is that in a weak map your key only be a JavaScript object.
* It is called weak because it holds weak references to the entries in the map.
* This in turn means that Ada is able to get rid of those entries if they're not used anymore for your code so they can be garbage collected for dead behavior
* 
* **We can’t to this because it does not have iterator.**
* 

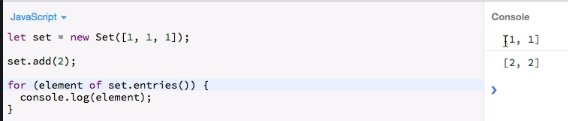
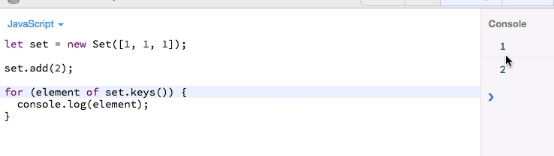
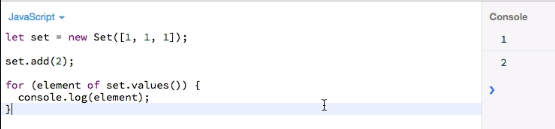
**77. Sets - Creation and Adding Items:**

* 
* 

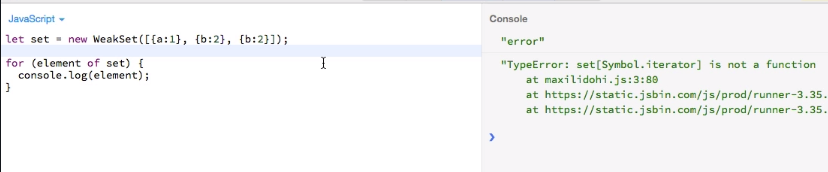
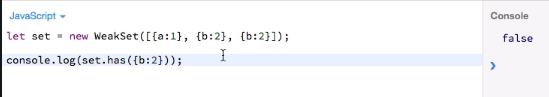
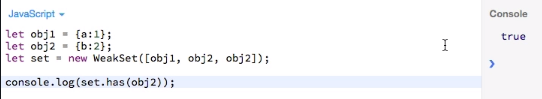
**78. Sets - Managing Items:**

* 
* 
* 

**79. Sets - Looping through Sets:**

* 
* 
* 

**81. The WeakSet:**

* Here also values have to be an object for the same reason as with the map only for objects is JavaScript able to well garbage collect them and to determine if they still in use.
* 
* 
* We still get an error of course because like the weak map the weak set is not iterable.
* 
* So, it is like a new object created in the memory. It is not the same that we have in our weak set.
* 
* 