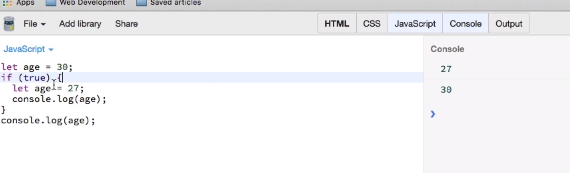
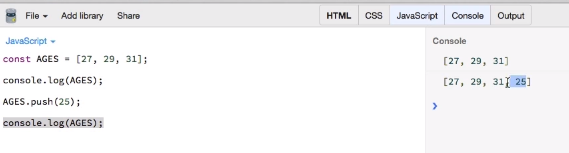
**JavaScript ES6**

**Section 1: By Maxmillan**

1. Syntax changes and Extensions; New Keywords
2. let – let allows us to use block scoping, which means any variable defined inside the for or if or even a block of curly braces will be available only inside the scope of that for or if. However, with var its not possible. In case of var only function scope will work.



1. const: const has also block scope. If we put an array in the const then we push any item in that array, it will get created without giving any error:



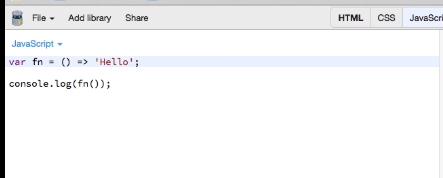
A const or var or let does not contain array they contain reference to the array. We are just changing the value where the pointer points.

If there is a cont object then also we would be able to change the object.

1. Hoisting with let and const:

Variable hoisting will not work with let and const, and also with arrow functions.

1. Fat Arrow Functions

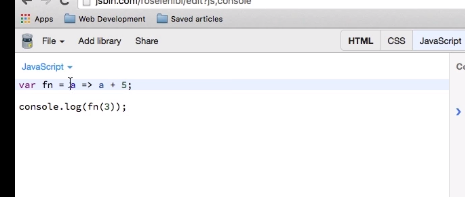


Above is the scenario when the function only returns one word, then we don’t need to write return also.

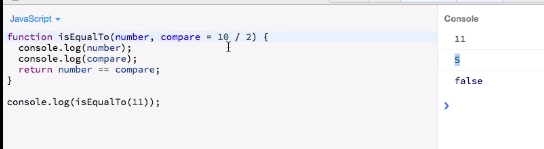


2 Functions in above screenshot are same to each other.

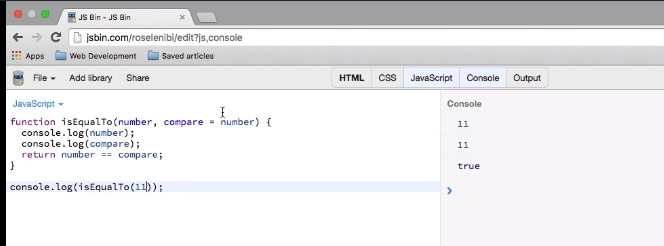
1. If you have 1 parameter passed in function, then we can leave out the parenthesis.

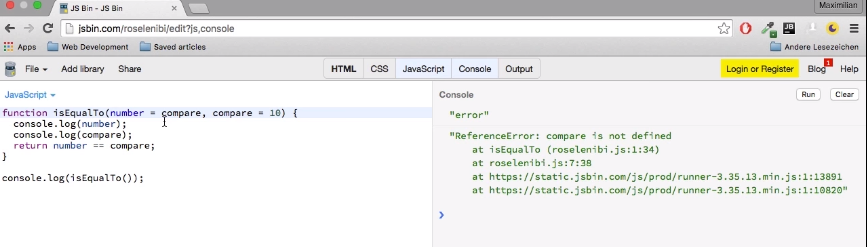


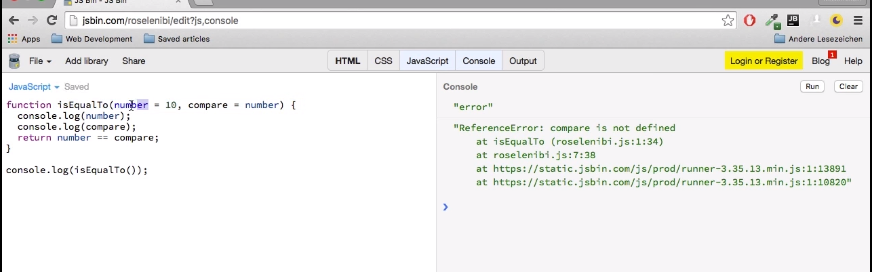
1. Fat Arrow Functions and “this” – For arrow functions its always Global object i.e. this function? Try in callback functions i.e. setTimeout/setInterval - no. If inside the class, this will always will always point to the class.
2. Default Parameters -> we can provide default value to the parameters of a function in ES6.
3. 
4. Below expression will also run fine:



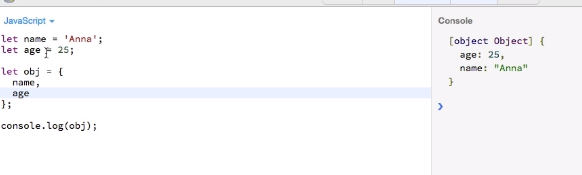
1. Below function arguments are also valid:



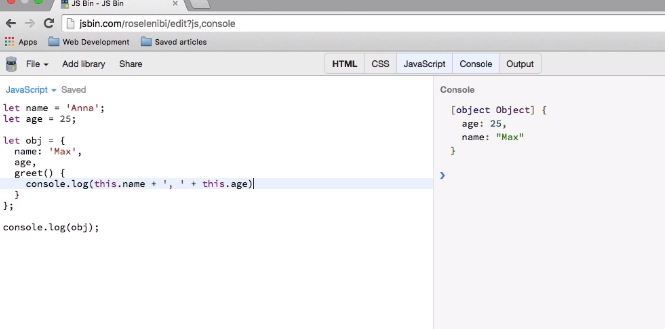
1. Here first assignment of the argument is not defined.
2. 
3. Below one will work:



1. Object Literal Extensions:
2. ES6 object declaration when Property and Value of the object are same for an item:

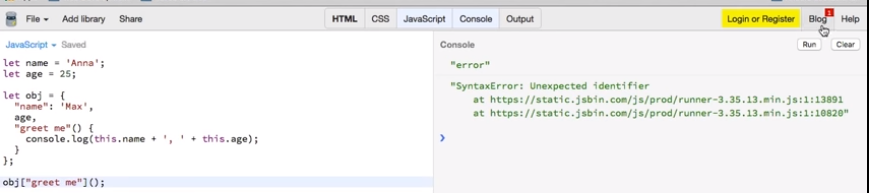


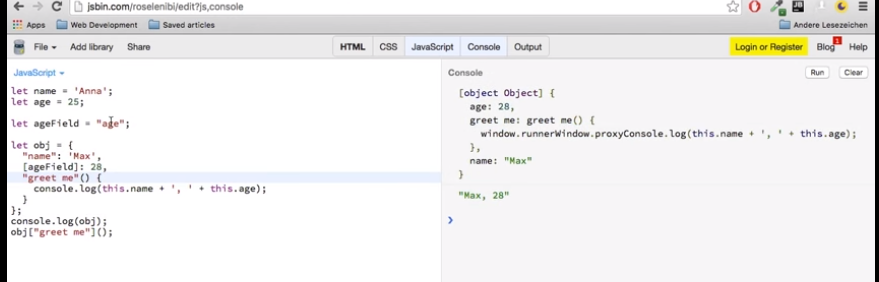
1. 
2. New syntax for the property function of the object:



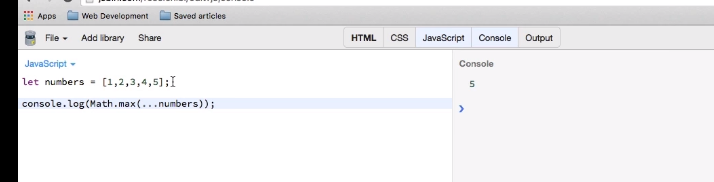
1. We can write property name in the object as string also:



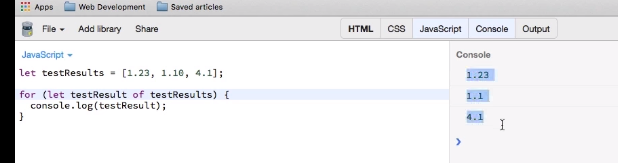
1. 
2. Dynamically add value to the object:



1. The REST operator  
   
2. The Spread operator:

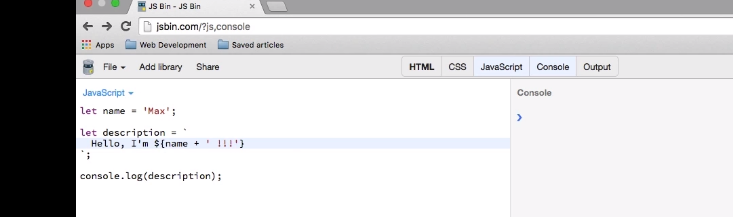
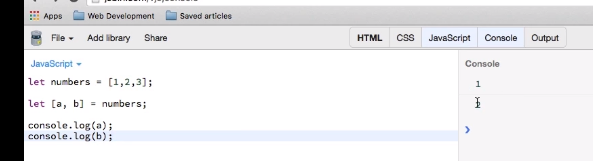


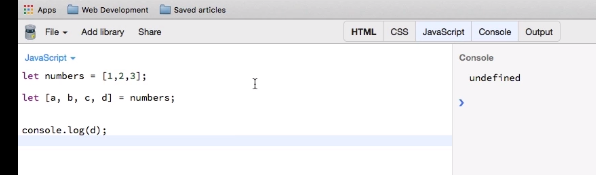
1. The For-of-Loop



1. Template literals – Strings with extra features



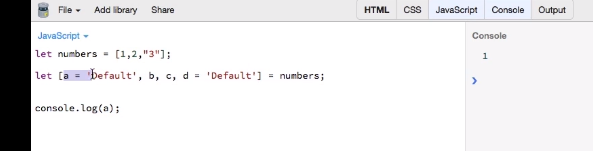
1. 
2. De-structuring - Arrays
3. 
4. If we try to fetch more values than the array has:



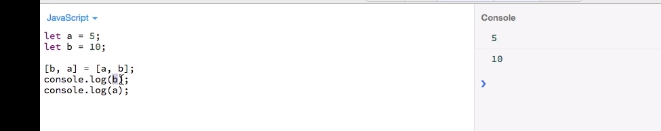
1. Pulling out remaining values using rest operator as a parameter:



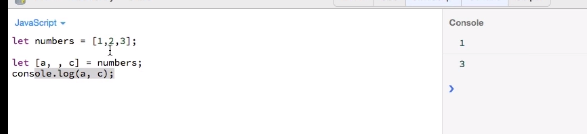
1. Mixture of default values and De-Structuring here:



1. De-structure to Swap variables:

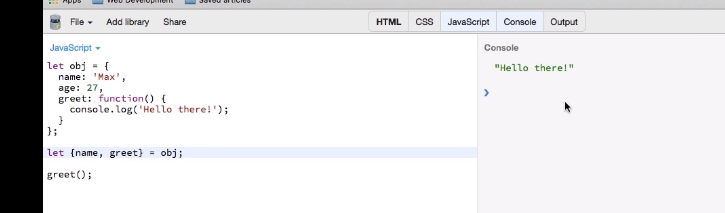
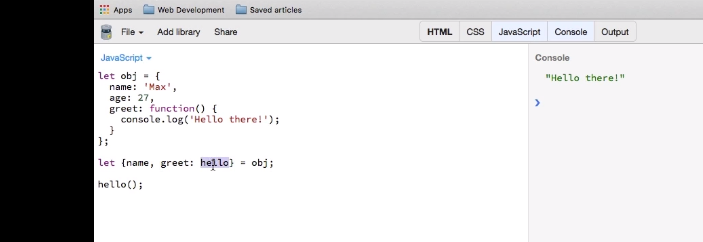


1. Ignore certain values:

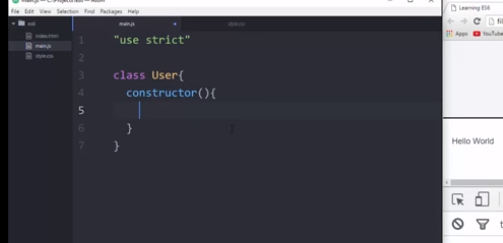
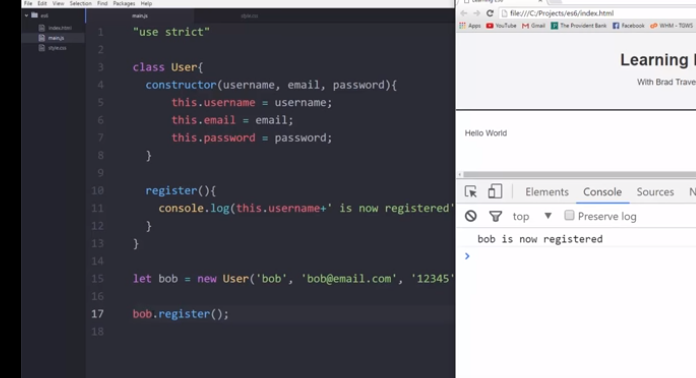
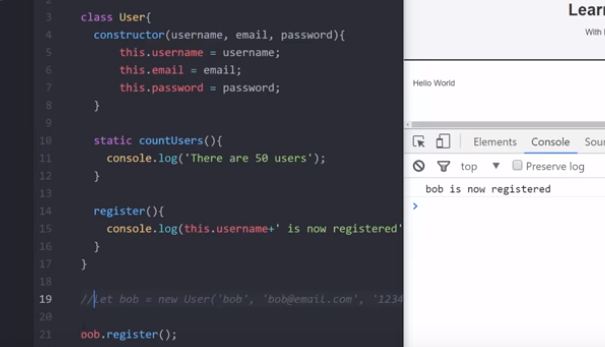
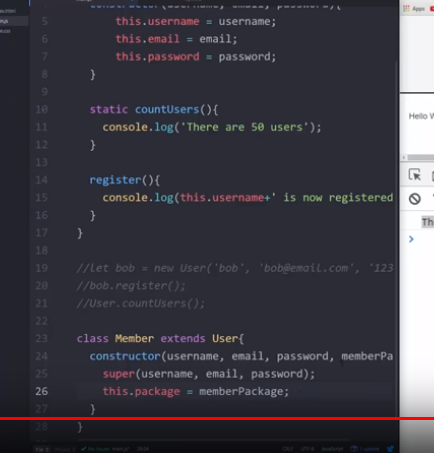
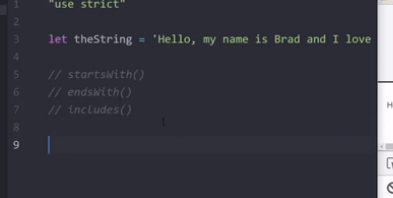
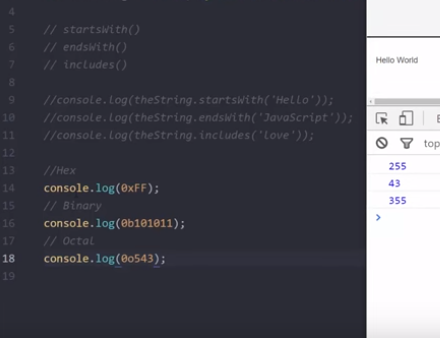
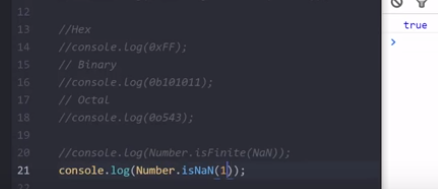
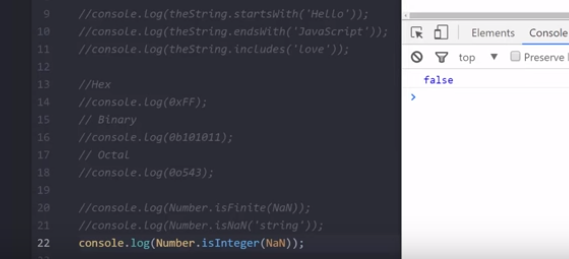
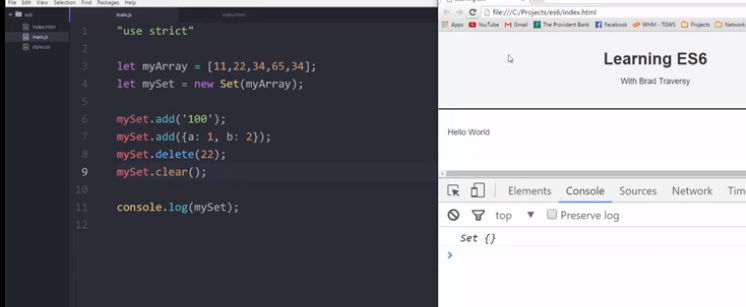
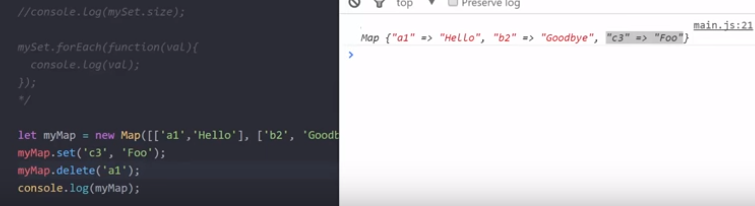


1. Immediate de-structure after creation:



1. de-structuring the objects:
2. 
3. 
4. Using alias for the object property names:
5. 

**Section 2: By Traversy Media**

1. JavaScript is a language based on EcmaScript Standerd.
2. Classes and Inheritance in JavaScript ES6.
3. 
4. Constructor is the method that get initiated when the class gets loaded.
5. This inside the constructor is new empty object.
6. 
7. Static method:
8. The **static** keyword defines a static method for a class. Static methods aren't called on instances of the class. Instead, they're called on the class itself.
9. 
10. Now, we will also extend the class in JavaScript
11. 
12. It also contains the use of super method.
13. New String and Number Methods
14. 
15. 
16. 
17. 
18. 
19. Set, Map, WeakMap, WeakSet functions in JavaScript
20. The **Set** object lets you store unique values of any type, whether [primitive values](https://developer.mozilla.org/en-US/docs/Glossary/Primitive) or object
21. The **map()** method creates a new array with the results of calling a provided function on every element in the calling array.
22. var array1 = [1, 4, 9, 16];
23. // pass a function to map
24. const map1 = array1.map(x => x \* 2);
25. console.log(map1);
26. // expected output: Array [2, 8, 18, 32]
27. 
28. Map are key value pairs as opposed to Set which are individual values
29. 
30. The **WeakSet** object lets you store weakly held objects in a collection.
31. var ws = new WeakSet();
32. var foo = {};
33. var bar = {};
34. ws.add(foo);
35. ws.add(bar);
36. ws.has(foo); // true
37. ws.has(bar); // true
38. ws.delete(foo); // removes foo from the set
39. ws.has(foo); // false, foo has been removed

33. The **WeakMap** object is a collection of key/value pairs in which the keys are weakly referenced.  The keys must be objects and the values can be arbitrary values.

34. var wm1 = new WeakMap(),

wm2 = new WeakMap(),

wm3 = new WeakMap();

var o1 = {},

o2 = function() {},

o3 = window;

wm1.set(o1, 37);

wm1.set(o2, 'azerty');

wm2.set(o1, o2); // a value can be anything, including an object or a function

wm2.set(o3, undefined);

wm2.set(wm1, wm2); // keys and values can be any objects. Even WeakMaps!

wm1.get(o2); // "azerty"

wm2.get(o2); // undefined, because there is no key for o2 on wm2

wm2.get(o3); // undefined, because that is the set value

wm1.has(o2); // true

wm2.has(o2); // false

wm2.has(o3); // true (even if the value itself is 'undefined')

wm3.set(o1, 37);

wm3.get(o1); // 37

wm1.has(o1); // true

wm1.delete(o1);

wm1.has(o1); // false

34. Promises:

The **Promise** object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value.

var promise1 = new Promise(function(resolve, reject) {

setTimeout(function() {

resolve('foo');

}, 300);

});

promise1.then(function(value) {

console.log(value);

// expected output: "foo"

});

console.log(promise1);

// expected output: [object Promise]

## 35. What are callbacks?

A **callback** is a *function* that is *passed into another function* as an *argument* to be *executed later*. (Developers say you “call” a function when you execute a function, which is why callbacks are named callbacks).

36. Generators in ES6

Generators in ES6 are the functions which can be paused and then resumed as many times you want. At each pause it can yield a value back

37. The **function\*** declaration (function keyword followed by an asterisk) defines a generator function, which returns a [Generator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Generator) object.

38. function\* generator(i) {

yield i;

yield i + 10;

}

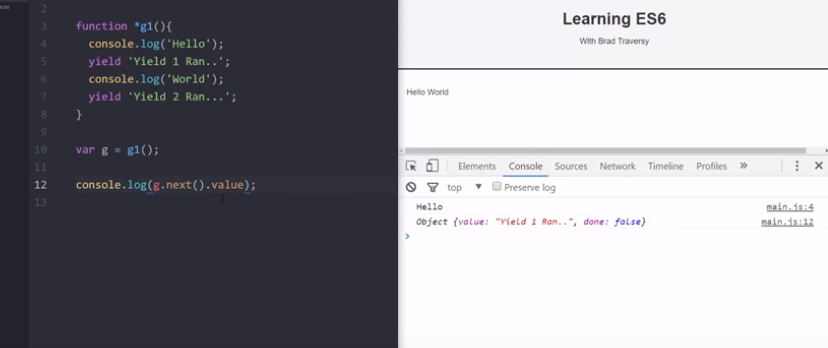
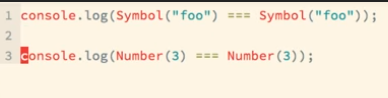
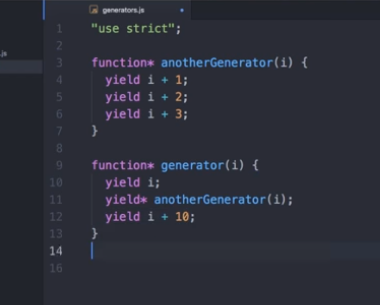
var gen = generator(10);

console.log(gen.next().value);

// expected output: 10

console.log(gen.next().value);

// expected output: 20

1. 
2. Symbols in ES6:
3. 
4. 
5. False; true;
6. Calling generator inside another generator
7. 
8. Proxy in ES6:
9. The **Proxy** object is used to define custom behavior for fundamental operations (e.g. property lookup, assignment, enumeration, function invocation, etc).
10. var handler = {
11. get: function(obj, prop) {
12. return prop in obj ?
13. obj[prop] :
14. 37;
15. }
16. };
17. var p = new Proxy({}, handler);
18. p.a = 1;
19. p.b = undefined;
20. console.log(p.a, p.b); // 1, undefined
21. console.log('c' in p, p.c); // false, 37