"ECMAScript" (also called "ES") — 6_Reference_Guide.md

the international std defining JavaScript. Released 2015, syntax chgs, THE recommended.

Assigning Variables

`let` values are re-assignable, but NOT accessible before they are declared.
Note the error below:

```
function letLogger() {
  console.log(x);
  let x = "hello";
}
letLogger();
```

```
> Uncaught ReferenceError:
x is not defined
  at letLogger
(<anonymous>:2:17)
  at <anonymous>:1:1
```

`const` values may not be reassigned. NOT declared/ initialized until line runs.
VALUES can't be reassigned, but OJECTS / ARRAYS can be manipulated — `.pop()` and `.push()`:

```
const petNames = ["Cleo", "Jax", "Chance", "Buckaroo", "fishy"]
console.log("All of my pets: ", petNames);
// Remove the last value from the array
petNames.pop();
// View the array
console.log("Dogs and cats: ", petNames)
```

```
Dogs and cats: >(4) ["Cleo", "Jax", "Chance", "Buckaroo"]
```

`.forEach` is used to call a function on each item in an array.

```
function printWithIndex(d, i) {
  console.log(i, d)
}
var arr = ["One", "Two", "Three", "Four"];
arr.forEach(printWithIndex);
```

```
0 "One"
1 "Two"
2 "Three"
3 "Four"
```

In the above example, `.forEach` is chained with the variable `arr`, returning both arguments (data and index) of the function.

Template Literals

replace string concatenation; backticks, `${ }`:

```
let firstName = "John";
let lastName = "Doe";
const fullName = `${firstName} ${lastName}`;
console.log(fullName);
> John Doe
```

`.map` — This method creates a new array from an existing array.
First, create function, call the `timesTwo` function on each element in the array:

```
function timesTwo(num) {
    return 2 * num;
}
var doubleItems = [1, 2, 3, 4].map(timesTwo);
console.log(doubleItems);
> (4) [2, 4, 6, 8]
```

Here is another example using `.map`:

```
let students = [{name: "John", grade: 89}, {name: "Jane", grade: 91}];
function getGrades(student) {
    return student.grade;
}
let grades = students.map(getGrades);
console.log(grades)
> (2) [89, 91]
```

When used iwith the `getGrades` function, `.map` creates a new variable containing only the student grades. The original array remains untouched.

Arrow Functions

Arrow functions provide a new syntax for writing functions in JavaScript. Using arrow functions creates code that is more concise and streamlined.
Let's revisit the code from the `.map` example above:

```
// Original function
function getGrades(student) {
    return student.grade;
}
let grades = students.map(getGrades);
// The same function rewritten as an arrow function
students.map(student => student.grade)
```

The same block of code has been condensed into a single line with the use of an arrow function. Note that a "fat arrow" has replaced the word "function". Also, without the curly brackets, the return statement is implied.
To create an arrow function using a single parameter, parentheses and curly brackets are omitted completely:
Parentheses contain two parameters in an arrow function, but curly brackets are still omitted:

```
var square = x => x * x;
var multiply = (a, b) => a * b;
```