JavaScript

Eric Chen
New to programming?

Code Academy tutorial
https://www.codecademy.com/learn/javascript
Recap

**HTML** is the language used to describe the *content and structure* for a web page.

**CSS** is the language used to describe the *presentation* of a web page. Makes stuff pretty!
JavaScript

JavaScript is the programming language used to manipulate the content of the webpage and make it interactive.

Makes things move!
JavaScript and Java

JavaScript is NOT related to Java! They are two different programming languages.
JavaScript and Java

JavaScript is NOT related to Java! They are two different programming languages.

Java : JavaScript : : butter : butterfly
JavaScript
Types

JavaScript has 5 primitive data types:

- Boolean
- Number
- String
Types

JavaScript has 5 primitive data types:

- Boolean
- Number
- String
- Null
- Undefined
Types

1 var myBoolean = true;
2 var myNumber = 5;
3 var mySecondNumber = 5.0;
4 var myString = "peanuts and m&m's";
Types (null and undefined)

1 var myString = "gummy bears";
2 myString = null;
3
4 var unassignedVariable;
5
6 // myString === null
7 // unassignedVariable === undefined
Output

`console.log()` will write to the JavaScript console.

```
1 console.log("JavaScript rules!");
2
```
Arrays

```javascript
var colorsOfTheWind = ['red', 'yellow', 'orange'];

colorsOfTheWind.push('blue'); // red, yellow, orange, blue

colorsOfTheWind.pop(); // red, yellow, orange
```
Arrays

```javascript
var colorsOfTheWind = ['red', 'yellow', 'orange'];

colorsOfTheWind.push('blue'); // red, yellow, orange, blue
colorsOfTheWind.pop(); // red, yellow, orange

// Print the contents of the array.
for (var i = 0; i < colorsOfTheWind.length; ++i) {
  console.log(colorsOfTheWind[i]);
}
```
A JavaScript object is a collection of key-value pairs.

```javascript
var person = {
  firstName : "Eric",
  lastName  : "Chen",
  age       : 22,
  eyeColor  : "brown"
};
```
Objects (accessing fields)

```javascript
8 console.log(person.firstName); // "Eric"
9 console.log(person["age"]); // 22
10
```
Functions

Two ways to declare functions

```javascript
var getGreeting = function(name) {
  return "Hi, " + name + "!";
};

function getSalutation(name) {
  return "Goodbye, " + name + "!";
};

getGreeting("Aaron"); // "Hi, Aaron!"
getSalutation("Yolanda"); // "Goodbye, Yolanda!"
```
Checking Equality

ALWAYS use ===, DO NOT use ==

- === : value equality for primitives, reference equality for objects
- == : coerces the two things to be the same type first

```
2 === 2; // true
2 == 2; // true
2 === "2"; // false
2 == "2"; // true
```
Value checks for primitives, reference checks for objects.

```javascript
var x = 2;
var y = 2;
x === y;

var person1 = {name: 'Monde'};
var person2 = {name: 'Monde'};
person1 === person2;
```
Value checks for primitives, reference checks for objects.

```javascript
1 var x = 2;
2 var y = 2;
3 
4 x == y;  // true
5 
6 var person1 = {name: 'Monde'};
7 var person2 = {name: 'Monde'};
8 
9 person1 == person2;
10 ```
Value checks for primitives, reference checks for objects.

```javascript
var x = 2;
var y = 2;

x == y; // true

var person1 = {name: 'Monde'};
var person2 = {name: 'Monde'};

person1 == person2; // false
```
Regarding Semicolons...

They aren’t strictly necessary, but you should always end each statement with a semicolon.

```javascript
var person = {
  firstName: "Eric",
  lastName: "Chen",
  age: 22,
  eyeColor: "brown"
};
```
Callback Functions
Definition

A callback function is a function that is passed as an argument to another function.
Example

```javascript
var addTwo = function(x) {
    return x + 2;
};

var modifyArray = function(array, callback) {
    for (var i = 0; i < array.length; ++i) {
        array[i] = callback(array[i]);
    }
};

var myArray = [5, 15, 25, 35]; // 5, 15, 25, 35
modifyArray(myArray, addTwo); // 7, 17, 27, 37
```
Example (cont’d)

```javascript
var modifyArray = function(array, callback) {
    for (var i = 0; i < array.length; ++i) {
        array[i] = callback(array[i]);
    }
};

var myArray = [5, 15, 25, 35];
modifyArray(myArray, function(x) {
    return x + 2;
});
```
Example (cont’d)

```javascript
var modifyArray = function(array, callback) {
    for (var i = 0; i < array.length; ++i) {
        array[i] = callback(array[i]);
    }
};

var myArray = [5, 15, 25, 35];
modifyArray(myArray, function(x) {
    return x + 2;
});

Instead of defining the callback function and assigning it to a variable, we are defining it “on-the-fly”.
```
Instead of defining the callback function and assigning it to a variable, we are defining it “on-the-fly” (anonymous function).
Example 2: Iteration

The Array class has the `forEach()` method for iterating through all its elements.
Example 2: Iteration

The Array class has the `forEach()` method for iterating through all its elements.

The `forEach()` method accepts one callback function as its argument, to be run once for each of its elements.
Example 2: Iteration

The Array class has the `forEach()` method for iterating through all its elements.

The `forEach()` method accepts one callback function as its argument, to be run once for each of its elements.

This callback function must have 3 arguments: the value of the element, the index of the element, and the array that is being iterated over.
Example 2: Iteration

```javascript
var colors = ["red", "yellow", "blue"];  

// Using the forEach() callback.
colors.forEach(function(value, index, array) {
    // array[index] === value
    console.log(value);
});

// Using a normal for loop.
for (var i = 0; i < colors.length; ++i) {
    console.log(colors[i]);
}
```
questions?
Scopes
What is scope?

Scope is the set of variables that you have access to.
What is scope?

Scope is the set of variables that you have access to.

```javascript
1 var printHi = function() {
2   var myString = "hi!";
3   console.log(myString);
4 }
5
6 var printHi2 = function() {
7   console.log(myString);
8 }
9
10 printHi2();
11 ```
What is scope?

Scope is the set of variables that you have access to.

```javascript
var printHi = function() {
    var myString = "hi!";
    console.log(myString);
};

var printHi2 = function() {
    console.log(myString);
};

printHi2();
```
Scoping

Each variable (functions, objects, strings, etc) resides within a scope.

Each function has its own local scope.
Scoping

Each variable (functions, objects, strings, etc) resides within a scope.

Each function has its own local scope.

```javascript
1 var printHi = function() {
2   var myString = "hi!";
3   console.log(myString);
4 }
5
6 var printHi2 = function() {
7   console.log(myString);
8 }
9
10 printHi2();
11 ```
Scoping

Each variable (functions, objects, strings, etc) resides within a scope.

Each function has its own local scope.

```javascript
var printHi = function() {
  var myString = "hi!";
  console.log(myString);
};

var printHi2 = function() {
  console.log(myString);
};

printHi2();
```
Scoping

Each variable (functions, objects, strings, etc) resides within a **scope**.

Each function has its own local scope.

```javascript
var printHi = function() {
  var myString = "hi!";
  console.log(myString);
}

var printHi2 = function() {
  console.log(myString);
}

printHi2();
```

printHi’s local scope.

printHi2 doesn’t have access to it!
Scoping

JavaScript has function scope, which means that the scope changes inside and outside of functions.
Local Scope

Each function has its own local scope.

```javascript
1 // code here cannot use myArray
2
3 function myFunction() {
4   var myArray = ['red', 'yellow'];
5
6   // code here can use myArray
7   // ...
8 }
9
10 // code here cannot use myArray
11```
Global Scope

A variable instantiated outside a function is in the global scope and can be accessed anywhere!

```javascript
1 var myArray = ['red', 'yellow'];
2 // code here can use myArray
3 function myFunction() {
4   // code here can use myArray
5   // ...
6 }
7 // code here can use myArray
8 ```
Global Scope

A variable instantiated outside a function is in the global scope and can be accessed anywhere!

```javascript
1 var myCounter = 3;
2
3 function increment() {
4   myCounter += 1;
5 }
6
7 increment(); // counter == 4
8 increment(); // counter == 5
9```
Global Scope

What happens when you do this?

```javascript
var myCounter = 3;

function doesSomething() {
    var myCounter = 20;
}

doesSomething();
```
Global Scope

What happens when you do this?

```
1 var myCounter = 3;
2
3 function doesSomething() {
4   var myCounter = 20;
5 }
6
7 doesSomething();
```

This creates a new variable called ‘myCounter’ in the function’s local scope. The global ‘myCounter’ is unaffected!
Good Practices
Conventions

- Use `===` for equality checking.
- End each statement with a semicolon.
- When creating a new variable, use the `var` keyword.
- camelCase your variables.
Indenting

- Use 2- or 4-space tabs, and be consistent.
- (And convert your tab characters into spaces!)
Spaces

Put spaces around your operators!

1 var x = a * (b + c); // yes!
2 var y=a*(b+c); // yuck!
Spaces (cont’d)

For loops and if-else statements

```javascript
for (var i = 0; i < 10; ++i) {
    // do something
}

var codingRoxMySox = true;
if (codingRoxMySox) {
    console.log('Yeah it does!');
} else {
    console.log('('):');
}
```
Style Guide

See https://github.com/airbnb/javascript for a more comprehensive style guide.
Playing with HTML
Reacting to User Input

- Button clicks
- Keypresses
- Mouse hovers
Getting or Changing HTML

- add/remove divs
- get the contents of a textbox
- add/remove classes
- change CSS
Loading a JavaScript File

```html
<!DOCTYPE HTML>
<html>
<head>
  
  <title>Snow White and the Seven Dwarves</title>
  
  <link rel="stylesheet" type="text/css" href="style.css">
  
  <script src="script.js"></script>

</head>

<body>
  
  <h1>Chapter 1</h1>
  
  <p>Mirror, mirror, on the wall...</p>

</body>

</html>
```
Manipulating the HTML Document

- Each JavaScript file loaded into the HTML document has access to the `window` object.
Manipulating the HTML Document

- Each JavaScript file loaded into the HTML document has access to the `window` object.
- The HTML itself can be accessed through `window.document`.
Manipulating the HTML Document

- Each JavaScript file loaded into the HTML document has access to the `window` object.
- The HTML itself can be accessed through `window.document`.
- But the `window` part is implied, so can just use `document`!
Service Worker termination by a timeout timer was canceled because DevTools is at

```
window

Window {speechSynthesis: SpeechSynthesis, caches: CacheStorage, localStorage: Storage,

window.document

#document

<!DOCTYPE html>

<html itemscope itemtype="http://schema.org/WebPage" lang="en">

<head>...

<body class="hp vasq" onload="try{if(!google.j.b){document.f&&document.f.q.f
nav_logo242.png" id="gsr">...

</body>

</html>
```
Example

```html
<!DOCTYPE HTML>
<html>
  <head>
    <title>Snow White and the Seven Dwarves</title>
    <link rel="stylesheet" type="text/css" href="style.css">
    <script src="script.js"></script>
  </head>
  <body>
    <h1>Chapter 1</h1>
    <p id="quote">Mirror, mirror, on the wall...</p>
  </body>
</html>
```

Let’s make something cool happen with this text.
document.addEventListener('DOMContentLoaded', function() {
    var element = document.getElementById('quote');

    element.addEventListener('click', function(mouseEvent) {
        element.innerHTML += ' Who is the fairest of them all?';
    });
});
Workshop: go.6148.io/js

Hint: Google is your friend!
Fin