Programming: Chapter Two

Mo' Java, Mo' Script

4. Programming Foundations (Continued)

More amazing things JavaScript can do!

Arrays

- Data type that holds an ordered list of values
- You can think of arrays as a group or a list
- Arrays can hold any kind of data (strings, numbers, or a mix)
- Arrays can even hold functions, variables and other arrays!

Creating Arrays

- Arrays are created just like variables, but you put the data inside **brackets**.
- This lets JavaScript know where the array list begins and ends.

```
var primaryColors = ['Red', 'Yellow', 'Blue'];

var billDenominations = [1, 2, 5, 10, 20, 50, 100];
```

Getting values from an array

- You get array values with “bracket notation”
- The number inside the brackets is called an “index”
- **NOTE: this part's a little confusing!**
- Arrays in JavaScript are “zero-indexed”
  - This means we start counting from zero, not one

Returning values in an array

```
var primaryColors = ['Red', 'Yellow', 'Blue'];


console.log(primaryColors[0]); //'Red'

console.log(primaryColors[1]); //'Yellow'

console.log(primaryColors[2]); //'Blue'
```

Returning values in an array

- If it helps to remember zero indexing, think of it like the color slider in a computer program such as Paint or Photoshop: 0 is all the way dark, 255 is all the way light.
- There are **256 possibilities**, but **255** is the max.
- Red 0, Green 0, Blue 0 = Black
- Red 255, Green 255, Blue 255 = White

Returning values in an array

- You can find out or get the number of items in an array easily
- This is done with the `.length` method
- **The array's length is the total number of items it holds**

```
var primaryColors = ['Red', 'Yellow', 'Blue'];


primaryColors.length; //3

//Note that Blue's index value is [2], but the array's
length is 3.
```

Finding the LAST item in an array

```
var primaryColors = ['Red', 'Yellow', 'Blue'];

var colorsLength = primaryColors.length; //3

console.log(primaryColors[colorsLength - 1]);

//This will be the same as:
console.log(primaryColors[2]);
Because 3-1 = 2
```

Updating values in an array

- Use bracket notation to update, or add an item
- The `.push` method is used to add a new item to the end of an array

```
var awesomeAnimals = ['Sloths', 'Dogs', 'Koalas'];
awesomeAnimals[0] = 'Otters'; // replaces Sloths
awesomeAnimals[4] = 'Penguins'; // adds new in fifth
awesomeAnimals.push('Ocelots'); // adds to end
```

Functions

Learn them well!

Functions

- **Function = reusable snippet of code**
- Used to repeat identical or similar actions without writing the same code over and over. Can be short and simple or long and complex
- Kind of like automated commands; they make life easier
- Saves space in code files
- Also extremely important to understand going into PHP…
- …but also has identical syntax, lucky for us!

Functions

```
// Declaring a function
function welcome() {
document.write('Hello there!');
}

// Calling the function (making it run):
welcome();
```

Functions and arguments

- **Every function name must contain parentheses at the end**
- These are used to pass “arguments” (or, modifiers) to the function, to make it run in slightly different ways
- Even if the function doesn’t actually have this capability, it must always be named and called with the parentheses
- This helps JavaScript tell between functions and variables

Functions and arguments

```
//declare the function

function myChunkOfCode(){

       console.log("Hooray! My function works!");

}


//call the function

myChunkOfCode();

Hooray! My function works!
```

Declaring and calling functions

- **Declaring** a function is a little like talking to your friend about somebody they don't know; it's a sort of introduction
- The first time you bring up this person, you need to explain who they are and what they do, like this:
- "There's this guy at work named **Bryan**. He sits across from me at Flywheel, and he does phone and chat support."

Declaring and calling functions

- **Calling** a function is like bringing up the person later; our friend is now familiar with who they are and what they do, so we can just use their name
- “Today **Bryan** said the funniest thing!”

Declaring and calling functions

```
function addNames() {
      var firstName = "Josh";
      var lastName = "Collinsworth";
      var fullName = firstName + ' ' + lastName;
      document.write(fullName);

}


addNames();

//writes “Josh Collinsworth”
```

Arguments

- An **argument** is a piece of data than can be "passed," or given to a function to work with
- If entered, the argument will modify the function or work with it in some way to change the outcome of the function
- **Basically, arguments are variables that are used just for the function**
- Arguments allow functions to be much more flexible
- Arguments allow the same general function to be reused multiple times with different details and results

Arguments: a simple example

```
function welcome(name) {
console.log('Hey there, ' + name + '!');
}


// Calling the function
welcome('Josh');


// The result:
Hey there, Josh!
```

Arguments

You can pass multiple arguments, including variable names

```
function addNumbers(num1, num2) {
        var result = num1 + num2;
        console.log(result);
}

addNumbers(5, 6);            //11
addNumbers(12, 24)           //36
addNumbers(22, -22)          //0
```

Arguments: another example

```
function checkout(items, price) {
        var grandTotal = items * price;
        console.log('Your total is $' + grandTotal + '.');
}

// Calling the function
checkout(5, 8);

// The result:
Your total is $40.
```

Arguments: one last example

```
function greeting(name) {
     document.write("Hello, " + name + "!");

}


greeting("Josh");

//writes “Hello, Josh!”
```

Undefined arguments

- Always beware of `undefined`, `NaN` and `Null`

```
function greeting(name) {
      document.write("Hello, " + name + "!");

}


greeting();

//writes “Hello, undefined!”
```

Avoiding Undefined Arguments with Conditionals

```
function greeting(name) {

    if(name){
         document.write("Hello, " + name + "!");

    } else {

        document.write("Hello, whoever you are!");

    }

}


greeting();

//writes “Hello, whoever you are!”

greeting("Josh");

//writes “Hello, Josh!”
```

Return Values

- Functions can return a value to you to **use**, but not necessarily output right away
- NOTE: no code after "return" will run.

```
function addNumbers(num1, num2) {
        var result = num1 + num2;
        return result;
        // This value is returned to be used later
}
```

Variable Scope

- JavaScript has “function scope”
- If a variable is declared **inside a function**, it is only accessible within that same function. This is a "local" variable.
- It’s created then forgotten every time the function runs
- If a variable is declared **outside a function**, however, it is accessible from anywhere. This is a "global" variable. It is created as soon as the script loads.

Variable Scope

```
function addNumbers(num1, num2) {
          var result = num1 + num2;
          console.log(result);
}

addNumbers(2, 1);       // result == 3
console.log(result);    // result == undefined
```

- The `result` variable only exists inside the `addNumbers` function. It's created when the function runs, then discarded.

New in ES6: Block Scoping

- By default, when you create a variable using the `var` keyword, it's got global scope; it's available anywhere after it's declared
- Using the `var` keyword can also "hoist" variables outside their scope
- Most of the time it won't matter, but it can cause issues
- ES6 gives us better ways to declare variables
- **A "block" is anything in a pair of curly braces**

New in ES6: Block Scoping

- The new let keyword is the same as var, but it keeps the variable inside the current scope
- So if you use let to define a variable inside a function or if statement, for example (AKA, inside a "block"), the variable will not be available outside that function or if statement

```
let apples = 5;
```

New in ES6: Block Scoping

```
let apples = 5;

console.log(apples); //5
```

```
let apples = 5;


function appleCounter(){

    console.log(apples); //not defined

}
```

New in ES6: Block Scoping

- We also have another new keyword in ES6 for creating "immutable" variables
- `var` and `let` create variables with data that can be updated or replaced
- An immutable piece of data **cannot be changed**
- `const` creates an immutable variable

```
const taylorSwift = "Never ever getting back together";
```

New in ES6: Block Scoping

```
const taylorSwift = "Never ever getting back together";


console.log(taylorSwift);
//Never ever getting back together

taylorSwift = "But maybe?";
//TypeError

taylorSwift += "...or are we?";
//TypeError
```

What to use?

- Generally, it's safe to use `var` for declaring any variable
- `var` is the oldest, original way of declaring a variable
- However, to keep variables from being used in the wrong context, `let` is becoming more popular. It “holds” variable inside their current block (curly braces)
- If you have a value you know should never change, then using `const` is probably a good idea

Review: `var`, `let` and `const`

```
var myVariable = 42;

//available inside other blocks and functions
//reassignable


let myVariable = 42;

//only available inside the current block
//reassignable


const myVariable = 42;

//only available inside the current block
//NOT reassignable
```

Loops

- **Used to loop through the same code multiple times, usually with a small change each time**
- Display a countdown
- Going through blog posts and displaying them
- Displaying search results
- Sorting a list of values

`while` Loop

- **Repeats statements while the specified condition is true**
- As long as whatever is inside the parentheses is true, the loop will keep on running over and over.
- This is where operators come in handy!

```
operators == “handy” //true
```

```
while (condition) {
      // keep doing this
}
```

`while` Loop

- Repeats statements while a condition is true

```
var x = 0;

while (x < 5) {
     console.log(x);
     x++;

}
```

`for` Loop

- Works just the same as a while loop, but set up slightly differently.
- Helps prevent endless loops by putting all the details at the top

```
for (initialize; condition; update) {
        // statements to repeat
}

for (var i = 0; i < 5; i++) {
```

for Loop

```
for (initialize; condition; update) {
       // statements to repeat
}
//initialize: How the loop starts
//condition: Run the loop as long as this is true
//update: Each iteration of the loop, do this
for (var i = 0; i < 5; i++) {
       console.log(i);
}
//01234
```

Loops & Arrays

- Use a `for` loop to look at all of the items in an array

```
var rainbowColors = ['Red', 'Orange', 'Yellow',
'Green', 'Blue', 'Indigo', 'Violet'];


for (var i = 0; i < rainbowColors.length; i++) {

      console.log(rainbowColors[i]);

}
//Red Orange Yellow GreenBlue Indigo Violet
```

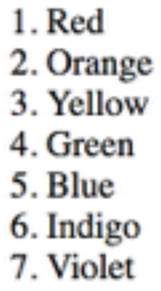
Remember! You can put HTML in JavaScript

```
var rainbowColors = ['Red', 'Orange', 'Yellow', 'Green',
'Blue', 'Indigo', 'Violet'];

document.write("<ol>");

for (var i = 0; i < rainbowColors.length; i++) {
      document.write(`<li> ${rainbowColors[i]} </li>`);
}

document.write("</ol>");
```



```
1. Red
2. Orange
3. Yellow
4. Green
5. Blue
6. Indigo
7. Violet
```

Intro to Objects

- Objects let us store a **collection** of properties and values
- Objects are essentially arrays, but with property/value pairs instead of a list of single items.
- You'll hear the phrase "object-oriented programming." That means working with objects such as these. It's a powerful technique.

```
var myObject = {
        firstName: "Josh",
        lastName: "Collinsworth",
        age: 36,
        bearded: true
};
```

Object Example

```
var charlie = {
        age: 8,
        name: "Charlie Brown",
        likes: ["baseball", "The red-haired girl"],
        pet: "Snoopy",
        bald: true
}
//Notice our object contains a number, string, array AND boolean!
Objects are powerful and portable.
```

Returning Object Values

```
var charlie = {
        age: 8,
        name: "Charlie Brown",
        likes: ["baseball", "The little red-haired girl"],
        pet: "Snoopy",
        bald: true
};
charlie.pet; //Call as dot notation (method)…
charlie['pet']; //…or in bracket notation
```

Changing Object Values

- Use dot or bracket notation to change objects values
- Change existing properties:
  - `charlie.name = "Chuck";`
- Or add new properties:
  - `charlie.gender = "male";`
- You can also delete properties:
  - `delete charlie.gender;`

YO DAWG I HEARD YOU LIKE ARRAYS
SO WE PUT OBJECT ARRAYS INSIDE YOUR ARRAYS
imgflip.com

Arrays can hold objects, which are sort of like arrays,
which can also hold arrays...

Arrays of Objects

- Arrays can hold objects, too
- You can loop through an array of objects

```
var peanuts = [
    {name: "Charlie", pet: "Snoopy"},
    {name: "Linus", pet: "Blue Blanket"}
];


for (var i = 0; i < peanuts.length; i++) {
    var peanut = peanuts[i];
    console.log(peanut.name + ' has a pet named ' + peanut.pet + '.');
}
```

Objects in Functions

- Pass an object into a function as a parameter

```
var peanut = { name: "Charlie Brown", pet: "Snoopy" };

function describeCharacter(character) {
       console.log(character.name + ' has a pet named ' +
character.pet + '.');
}
describeCharacter(peanut);
```

Methods

- Methods are functions that are associated with an object
- They affect or return a value for a specific object
- Used with dot notation

```
document.write("Hello, world!");
```

Adding methods to objects

- Declare method with the object
- Attached using dot notation

```
var charlie = {
      name: "Charlie",
      sayHello: function() {
             document.write("My name is " + charlie.name);
      }
};
charlie.sayHello();
```

“This”

- Inside methods, properties are accessed using the this keyword
- this refers to the "owner" of the property

“This”

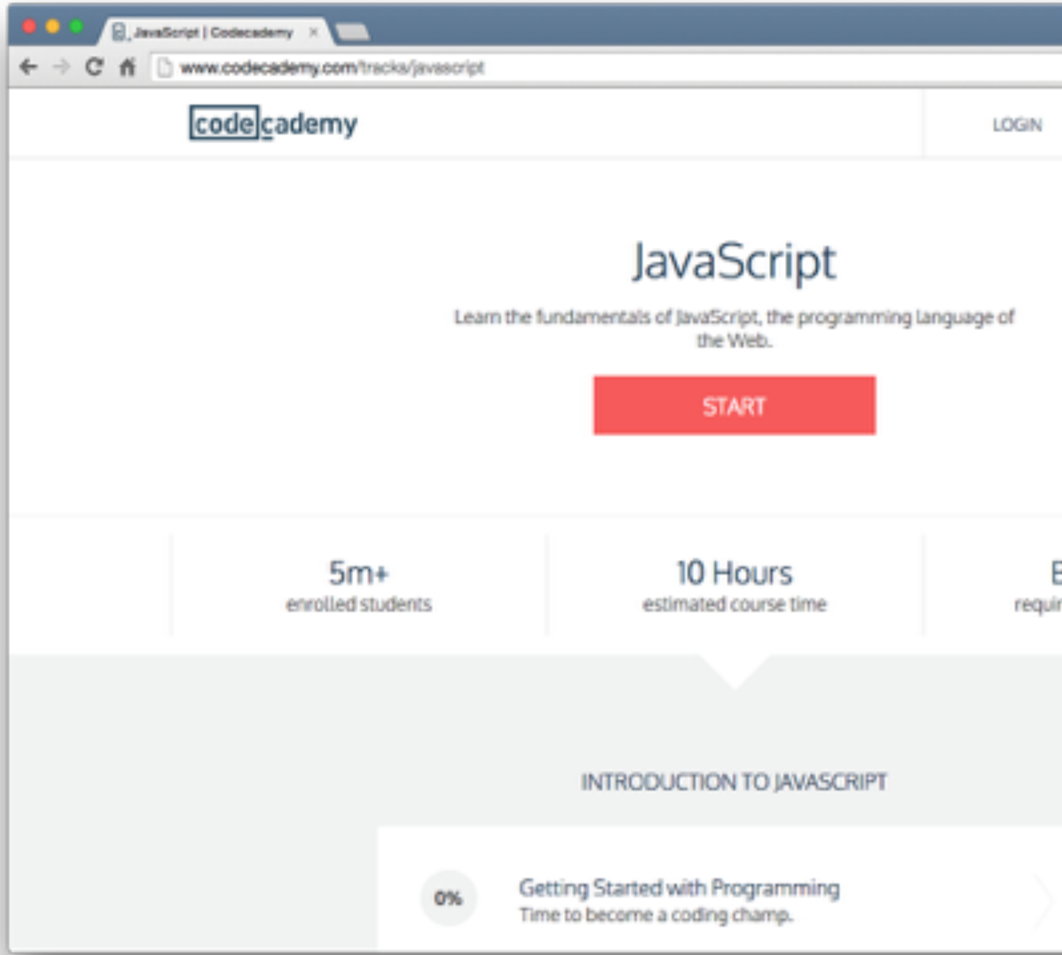
```
var charlie = {
        name: "Charlie",
        sayHello: function () {
                document.write("My name is " + this.name + ".");
        }
};
var lucy = {
        name: "Lucy van Pelt",
        sayHello: function () {
                document.write("My name is " + this.name + ".");
        }
};
charlie.sayHello(); // My name is Charlie.
lucy.sayHello(); // My name is Lucy van Pelt.
```

5.
Tools & Resources

Codecademy

Very thorough JavaScript tutorials (10 hours of material)

Direct code walkthroughs



JavaScript | Codecademy

www.codecademy.com/tracks/javascript

codecademy

LOGIN

JavaScript

Learn the fundamentals of JavaScript, the programming language of the Web.

START

5m+
enrolled students

10 Hours
estimated course time

INTRODUCTION TO JAVASCRIPT

0% Getting Started with Programming
Time to become a coding champ.

JS: The Right Way

Compiled, curated list of good Javascript information

Very reputable source of best practices

JS: The Right Way

jstherightway.org

JavaScript

The Right Way

Hey, you!

This is a guide intended to introduce new developers to JavaScript and help experienced developers learn more about its
practices.

Despite the name, this guide doesn't necessarily mean "the only way" to do JavaScript.

We just gather all the articles, tips, and tricks from top developers and put it here. Since it comes from exceptional folks,
say that it is "the right way", or the best way to do so.

CHOOSE YOUR PATH

Waiting for www.facebook.com...

MDN JavaScript Reference

The definitive guide, per usual

Overview, tutorials, exercises

JavaScript Guide - Jav

Mozilla Foundation [US] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

MDN MOZILLA DEVELOPER NETWORK

ZONES ▾ WEB PLATFORM ▾

MDN › Web technology for developers › JavaScript › JavaScript Guide

JavaScript Guide

▲ HIDE SIDEBAR

SEE ALSO

JavaScript

Tutorials:

▾ JavaScript Guide

- Introduction
- Grammar and types
- Control flow and error handling
- Loops and iteration
- Functions
- Expressions and operators
- Numbers and dates
- Text formatting
- Regular expressions
- Indexed collections

The JavaScript Guide shows you how to use JavaScript and gives an overvie
JavaScript or programming in general, consult the articles in the learning ar
language feature, have a look at the JavaScript reference.

Introduction
- About this guide
- About JavaScript
- JavaScript and Java
- ECMAScript
- Tools
- Hello World

Grammar and types
- Basic syntax & comments
- Declarations
- Variable scope
- Variable hoisting
- Data structures and types
- Literals

Contr
error
- if...e
- switch
- try/ca
- Error ob
- Promise

JavaScript30.com

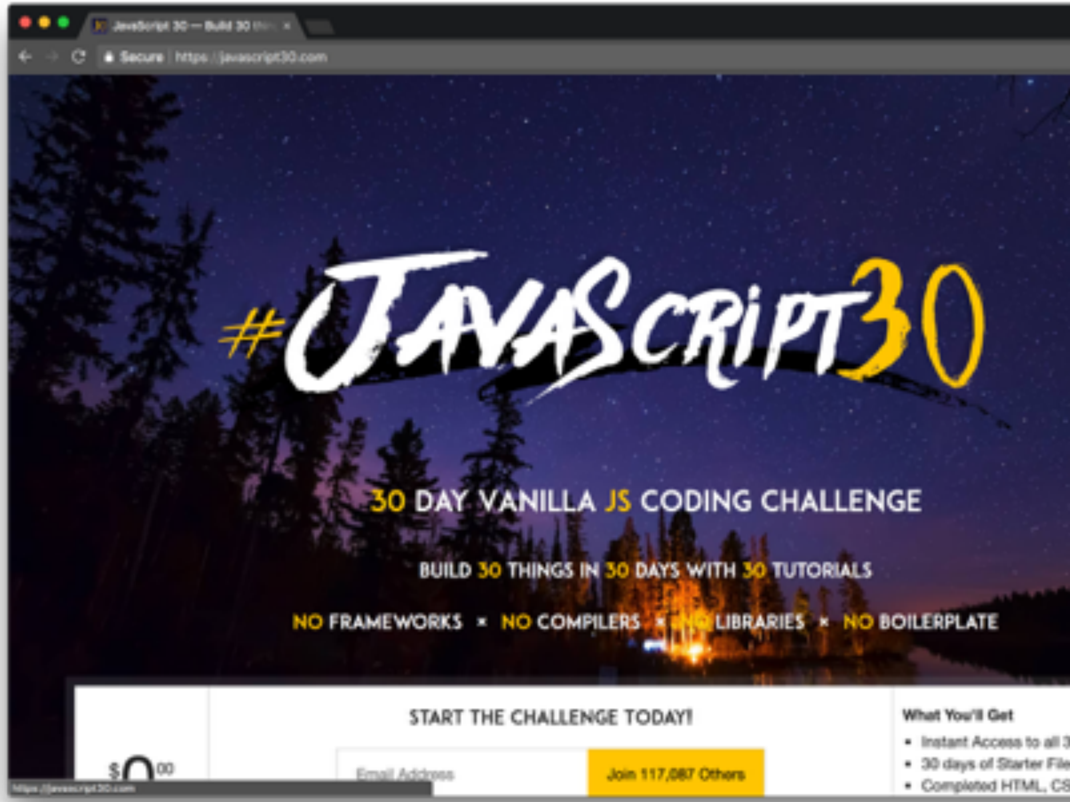
FREE 30-day vanilla JS coding challenge video series

Videos roughly 15–25 minutes

One video per day

Covers LOTS of topics and newer ES6 syntax

Not for learning JS basics



JavaScript 30 — Build 30 thi
Secure https://javascript30.com
#JavaScript30
30 DAY VANILLA JS CODING CHALLENGE
BUILD 30 THINGS IN 30 DAYS WITH 30 TUTORIALS
NO FRAMEWORKS × NO COMPILERS × NO LIBRARIES × NO BOILERPLATE
START THE CHALLENGE TODAY!
$0.00
Email Address
Join 117,087 Others
What You'll Get
Instant Access to all 3
30 days of Starter File
Completed HTML, CS

Eloquent JavaScript

Published physical book, also available online for free

Very thorough

Includes code sandbox and exercises each chapter

Eloquent JavaScript
eloquentjavascript.net/index.html

Eloquent JavaScript
Second Edition

This is a book about JavaScript, programming, and the wonders of the digital.
You can read it online here, or get your own paperback copy of the book.

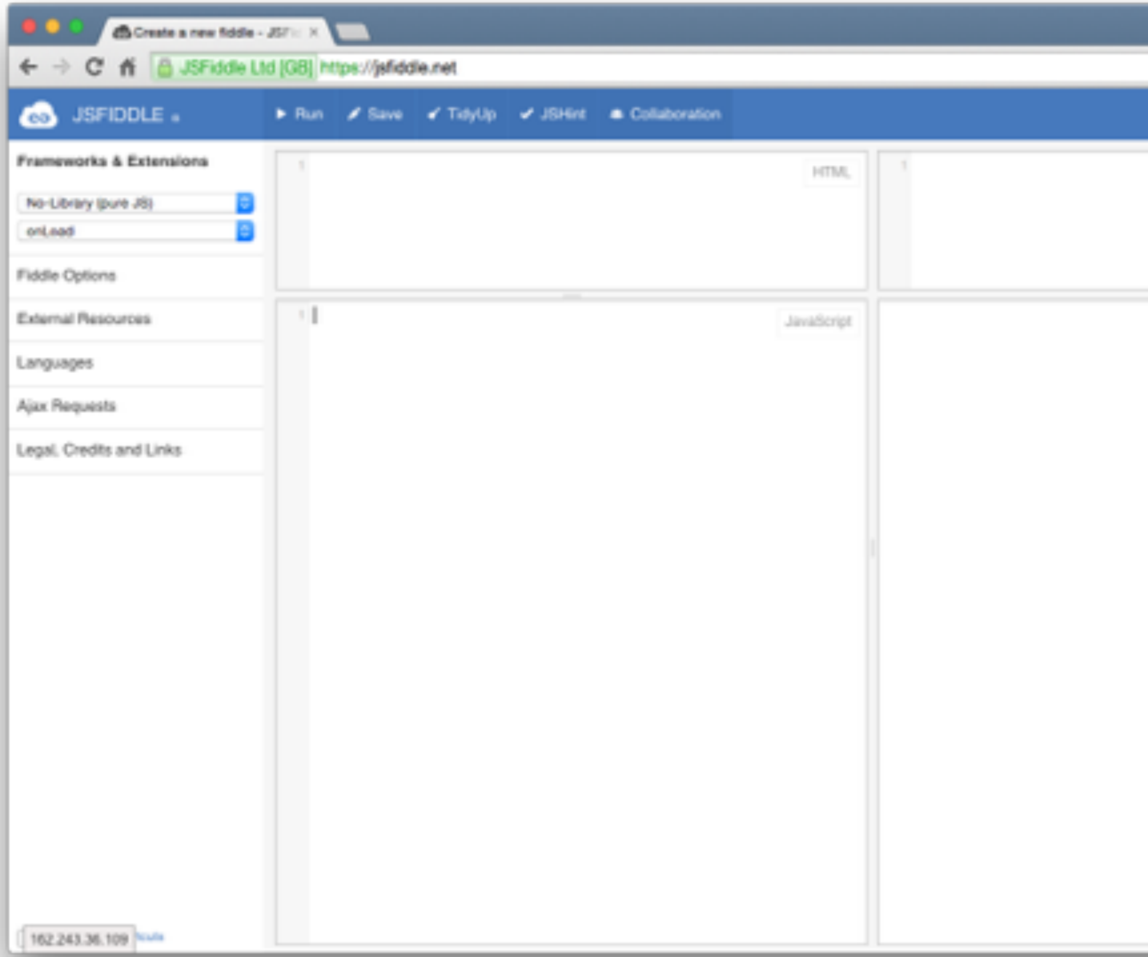
ELOQUENT JAVASCRIPT
SECOND EDITION
A Modern Introduction to Programming
Marijn Haverbeke

JSFiddle

Similar to CodePen

Allows access to many JavaScript Libraries

JSHint built in

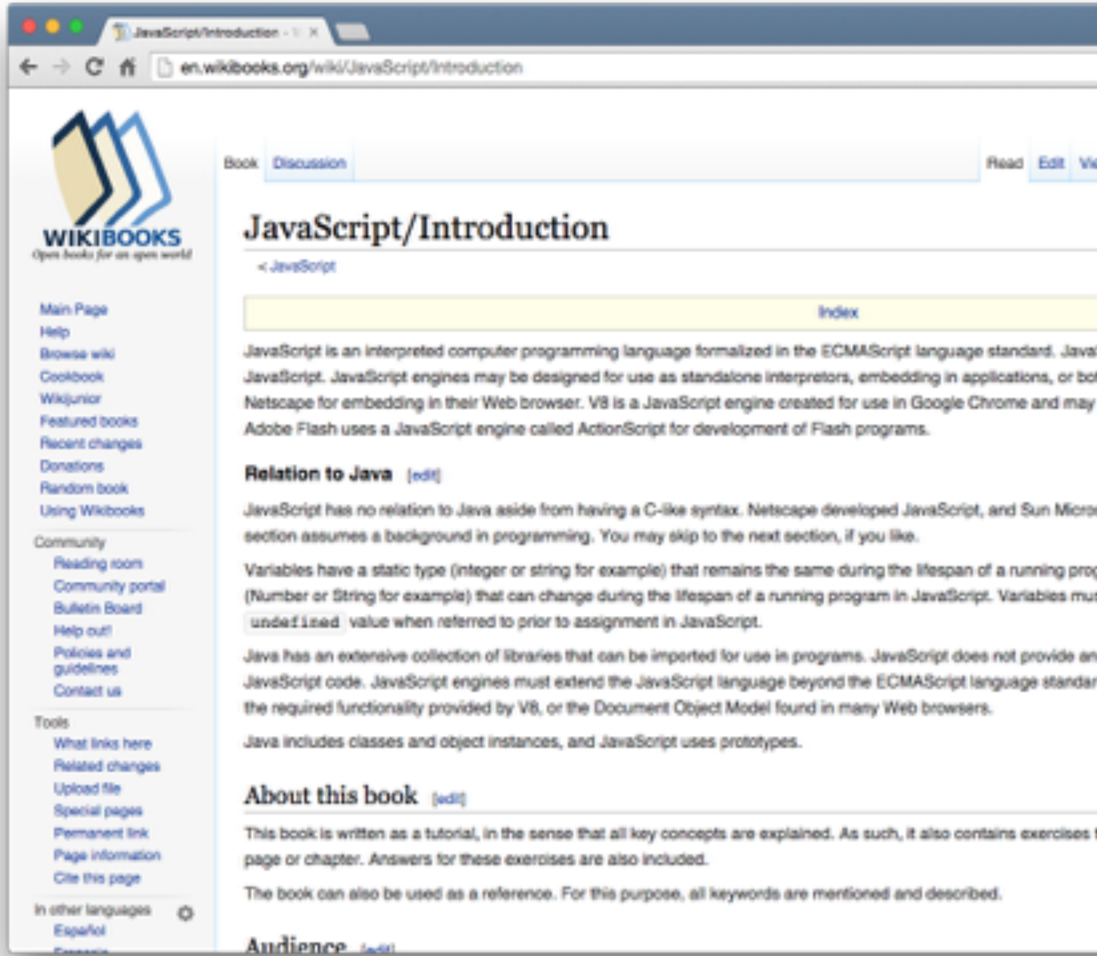


No-Library (pure JS)
onLoad

WikiBooks JavaScript

Lots of great info and links

Created as a ‘walkthrough’ style



JavaScript/Introduction

< JavaScript

Index

JavaScript is an interpreted computer programming language formalized in the ECMAScript language standard. Java
JavaScript. JavaScript engines may be designed for use as standalone interpretors, embedding in applications, or bo
Netscape for embedding in their Web browser. V8 is a JavaScript engine created for use in Google Chrome and may
Adobe Flash uses a JavaScript engine called ActionScript for development of Flash programs.

Relation to Java [edit]

JavaScript has no relation to Java aside from having a C-like syntax. Netscape developed JavaScript, and Sun Micro
section assumes a background in programming. You may skip to the next section, if you like.

Variables have a static type (integer or string for example) that remains the same during the lifespan of a running prog
(Number or String for example) that can change during the lifespan of a running program in JavaScript. Variables mu
`undefined` value when referred to prior to assignment in JavaScript.

Java has an extensive collection of libraries that can be imported for use in programs. JavaScript does not provide an
JavaScript code. JavaScript engines must extend the JavaScript language beyond the ECMAScript language standar
the required functionality provided by V8, or the Document Object Model found in many Web browsers.

Java includes classes and object instances, and JavaScript uses prototypes.

About this book [edit]

This book is written as a tutorial, in the sense that all key concepts are explained. As such, it also contains exercises
page or chapter. Answers for these exercises are also included.

The book can also be used as a reference. For this purpose, all keywords are mentioned and described.

Audience [edit]

Other Resources

- Twitter
  - http://code.tutsplus.com/articles/33-developers-you-must-subscribe-to-as-a-javascript-junkie--net-18151
- Podcasts
  - http://simpleprogrammer.com/2014/03/10/ultimate-list-developer-podcasts

Let’s Review!

Pop quiz, hotshot

On a line of JavaScript where we are declaring a variable (in ES5) the first thing we need to type is:

- var
- $
- document
- console
- new

On a line of JavaScript where we are declaring a variable, the first thing we need to type is:

- **var**
- $
- document
- console
- new

What new keywords for creating variables exist now in ES6?

- now
- let
- give
- const
- do

What new keywords for creating variables exist now in ES6?

- now
- let
- give
- const
- do

What are the four main types of data in JavaScript?

- Text, number, boolean, array
- String, binary, boolean, array
- String, number, boolean, object
- String, number, boolean, array
- Text, number, dual, array

What are the four main types of data in JavaScript?

- Text, number, boolean, array
- String, binary, boolean, array
- String, number, boolean, object
- **String, number, boolean, array**
- Text, number, dual, array

What is a template string? What characters are used to create one?

What is a template string? What characters are used to create one?

- A template string (or template literal) lets us insert variables and line breaks into strings
- The string must be wrapped in backticks
- The variables must be enclosed inside `${}`

What's the difference between = and == ?

What's the difference between = and == ?

- = **assigns** a value:

```
var myVariable = "My variable value";
```

- == **compares** two values:

```
console.log(15 == "banana");
//false
```

What's the difference between == and === ?

What's the difference between == and === ?

- == **does type conversion** and checks only the value:

```
console.log(2 == "2");
//true
```

- === **does no conversion**; it checks both value **and** type:

```
console.log(2 === "2");
//false
```

After the variables below have been declared, which of the statements will evaluate to FALSE?

```
var apples = 5;
var oranges = 8;
var kiwi = 8;


oranges >= kiwi;

kiwi === oranges;

apples < kiwi;

apples !== oranges;

kiwi != oranges;
```

After the variables below have been declared, which of the statements will evaluate to FALSE?

```
var apples = 5;
var oranges = 8;
var kiwi = 8;
```

```
oranges >= kiwi;

kiwi === oranges;

apples < kiwi;

apples !== oranges;
```

`kiwi != oranges;`

What is the proper syntax for an IF statement?

```
if[(apples > oranges) { console.log("true"); }]

if apples > oranges { console.log("true"); }

if(apples > oranges) { console.log("true"); }

if{apples > oranges} { console.log("true"); }

if(apples > oranges { console.log("true"); });
```

What is the proper syntax for an IF statement?

```
if[(apples > oranges) { console.log("true"); }]

if apples > oranges { console.log("true"); }
```

`if(apples > oranges) { console.log("true"); }`

```
if{apples > oranges} { console.log("true"); }

if(apples > oranges { console.log("true"); });
```

Will the following IF statement work?
Or will it evaluate to false?

```
var apples = 6;

var oranges = 8;

var kiwi = 12;



if (oranges > 6 && apples < 8 || kiwi <= 8 &&
oranges !== 12) {

      //Code here. Will it work?

}
```

Will the following IF statement work?
Or will it evaluate to false?

```
var apples = 6;

var oranges = 8;

var kiwi = 12;



if (oranges > 6 && apples < 8 || kiwi <= 8 &&
oranges !== 12) {

     //Code here. Will it work?

} // YES!
```

What is the proper way to create a function that will log the word “test” to the console?

```
function testLog {
        console.log("test");
}

function testLog() {
        console.log(test);
}

function testLog(x) {
        console.log("x");
}

function testLog() {
        console.log("test");
}
```

What is the proper way to create a function that will log the word “test” to the console?

```
function testLog {
        console.log("test");
}

function testLog() {
        console.log(test);
}

function testLog(x) {
        console.log("x");
}
```

`function testLog() {`
`        console.log("test");`
`}`

How do we call our testLog function from the last slide? (Shown at top.)

```
function testLog() {
        console.log("test");
}


function testLog();

testLog;

testLog.write;

document.write(testLog());

testLog();
```

How do we call our testLog function from the last slide? (Shown at top.)

```
function testLog() {
        console.log("test");
}
```

```
function testLog();
```

```
testLog;
```

```
testLog.write;
```

```
document.write(testLog());
```

`testLog();`

Match each column item to its corresponding item in the other column

- A. =
- B. ==
- C. ===

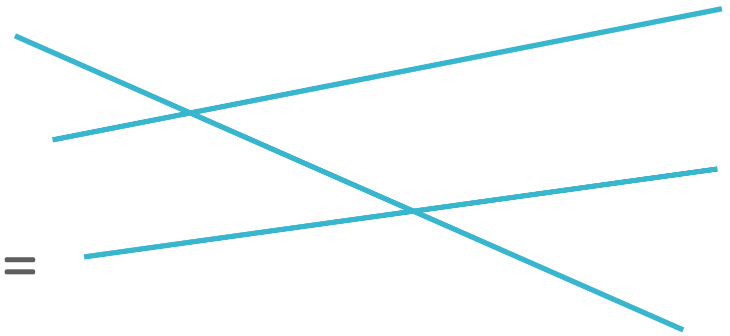
```
1. Matches two values by
performing type conversion

2. Matches both value and
data type

3. The assignment operator;
assigns or reassigns a value
```

Match each column item to its corresponding item in the other column

- A. =
- B. ==
- C. ===



1. Matches two values by performing type conversion
2. Matches both value and data type
3. The assignment operator; assigns or reassigns a value

How do we log only the LAST item in this array to the console?

```
var belchers =
    ['Bob', 'Linda', 'Tina', 'Gene', 'Louise'];
```

How do we log only the LAST item in this array to the console?

```
var belchers =
    ['Bob', 'Linda', 'Tina', 'Gene', 'Louise'];


console.log(belchers[4]);


//OR, as long as we know it's the last item:

console.log(belchers[belchers.length -1]);
```

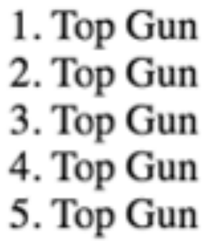
Ok! Let’s do some JavaScript! (1 of 2)

- Open CodePen and create a new pen
- **Create two new variables**, and assign them both **numeric values**
- Create a new function named “success” that logs a success message to the console. (Don’t forget to open the console so you can see it.)
- Write an “IF” statement. The condition should involve:
  - **Both your variables** from above (e.g., comparing their values)
  - At least one “and” logical operator **-OR-** one “or” logical operator
- If your IF statement evaluates as true, it should **call the “success” function**
- Otherwise, add an “ELSE” condition that logs “Too bad!” to the console

Ok! Let's do some JavaScript! (2 of 2)

- Open CodePen and create a new pen
- **Create an array** with the names of your top five favorite movies in it
- Write a loop that will cycle through each item in the array, writing each one to the page and numbering them.

  The loop can be whatever kind of loop you want; just be sure you end up with the whole list showing on the screen.
  The end result should look something like this:



```
1. Top Gun
2. Top Gun
3. Top Gun
4. Top Gun
5. Top Gun
```

Questions ?

Ask away