

Advanced JavaScript

cbansimba@gmail.com

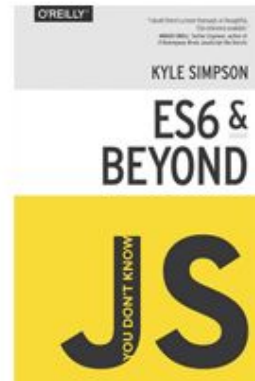
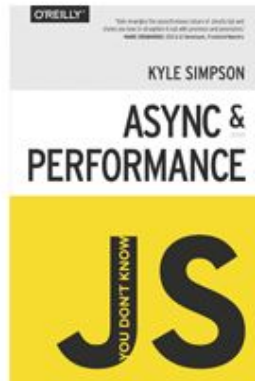
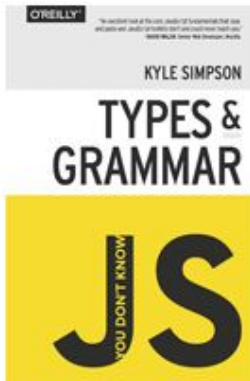
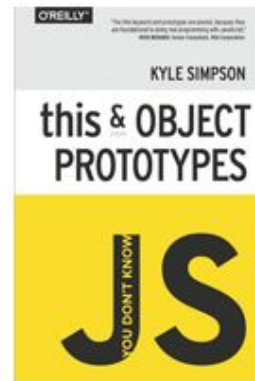
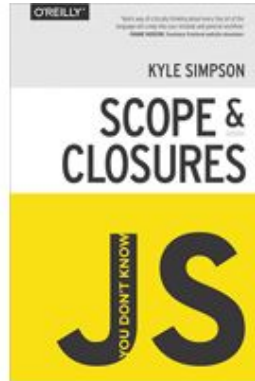
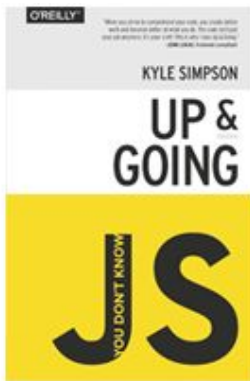
ITM - SDM S9

EPITA

Summary

1. Introduction
2. Values & Types
3. Object
4. Arrays
5. Comparing
6. Variables
7. Conditionals
8. Strict Mode
9. Function As Values
10. this Identifier
11. Prototypes

Bibliography



<https://github.com/getify/You-Dont-Know-JS>

Unearthing the Excellence in JavaScript



JavaScript: The Good Parts

O'REILLY®

YAHOO! PRESS

Douglas Crockford

Introduction

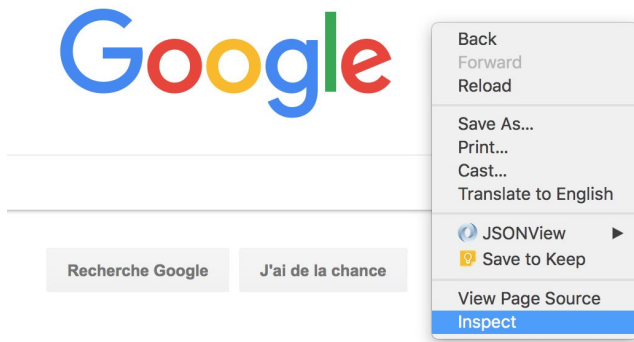
Brendan Eich



Javascript is created in just 10 days
by Brendan Eich while in working on
Netscape

How to execute JavaScript code ?

- In browser console





ogle



Elements Console Sources >> ⋮ ✕

⊘ top ▼ Filter Default levels ▼ ⚙

> console.log('I teach you Javascript')

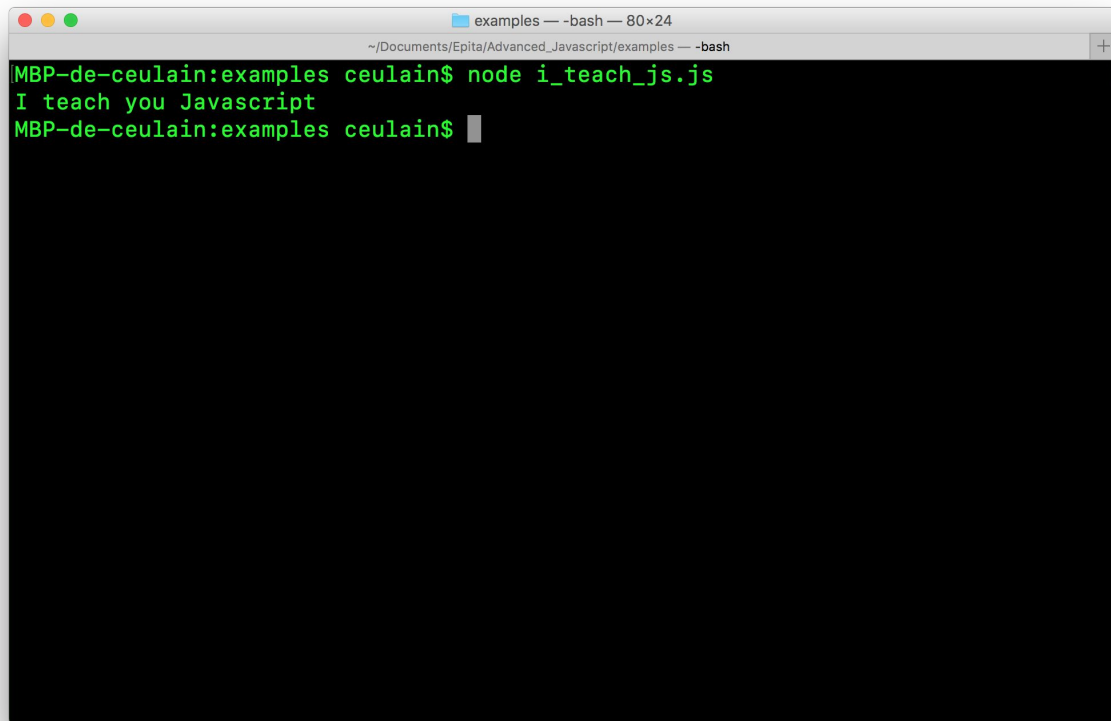
I teach you Javascript VM2425:1

<> undefined

>

- with Node.js

install Node.js <https://nodejs.org/en/>

A screenshot of a macOS terminal window. The title bar shows 'examples — -bash — 80x24'. The path bar shows '~/Documents/Epita/Advanced_Javascript/examples — -bash'. The terminal content shows a command being executed and its output.

```
MBP-de-ceulain:examples ceulain$ node i_teach_js.js
I teach you Javascript
MBP-de-ceulain:examples ceulain$
```

Statements

- In Javascript, statement finish by semicolon

```
a = a * 2;
```

Variables

- Variables are **weakly typed**.
- Declare with the ***var*** statement.
- Variables must be a valid *identifiers*
- An identifier start with **a-z, A-Z, \$, or _**. It can then contain any of those characters plus the numerals **0-9**.

```
var amount = 99.99;  
  
amount = amount * 2  
  
console.log(amount); // 199.98  
  
// convert `amount` to a string, and  
// add "$" on the beginning  
amount = "$" + String(amount);  
  
console.log(amount);
```

Types

JavaScript has typed values, not typed variables.

- string
- number
- boolean
- null and undefined
- object
- symbol

Javascript provides a *typeof* operator that can examine a value and tell you what the type it is.

```
var a;  
typeof a;           // "undefined"  
  
a = "hello world";  
typeof a;           // "string"  
  
a = 42;  
typeof a;           // "number"  
  
a = true;  
typeof a;           // "boolean"  
  
a = null  
typeof a;           // "object" -- weird, bug  
  
a = undefined  
typeof a;           // "undefined"  
  
a = { b: "c" };  
typeof a;           // "object"
```

Operator

Operator	Sign
Assignment	=
addition	+
subtraction	-
multiplication	*
division	/
Compound Assignment	+=, -=, *= and /= (a += 2 same as a = a + 2)
Increment	++
Object Property access	.

Equality operator

Operator	Sign
losse-equals	==
strict-equals	===
!=	loose not-equals
!==	strict not-equals

Comparison operator

Operator	Sign
less than	<
greater than	>
less than or loose-equals	<=
greater than or loose-equals	>=

Logical operator

Operator	Sign
and	&&
or	

Exshautive list of operators:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators

Comparing Values

- There are two main types of value comparisons that will need to make in your JavaScript programs:
 - Equality
 - Inequality

Coercion

- Convert a value to another type called *Coercion*

```
var a = "42"  
var b = Number(a);  
  
console.log(a); // "42"  
console.log(b); // 42
```

Explicit coercion : **Explicit coercion** is simply that you can see obviously from the code that a conversion from one type to another will occur

```
var a = "42";  
  
var b = Number(a);  
  
console.log(a); // "42"  
console.log(b); // 42
```

Implicit coercion : **implicit coercion** is when the type conversion can happen as more of a non-obvious side effect of some other operation

```
var a = "42";  
  
var b = a * 1; // "42" implicitly coerced to 42  
here  
  
console.log(a); // "42"  
console.log(b); // 42 -- the number
```

Truthy & Falsy Value

The specific list of falsy values in Javascript is as follows :

- "" (empty string)
- 0, -0, NaN (invalid number)
- null, undefined
- false

Any value that's not on this "falsy" list is "truthy". Here are some examples of those:

- "hello"
- 42
- true
- [], [1, "2", 3] (arrays)
- {}, { a: 42 } (objects)
- function foo() { ... } (functions)

Equality

There are four equality operators: `==`, `===`, `!=`, `!==`.

What is the difference between `==` and `===` ?

- `==` allows coercion
- `===` doesn't allow coercion

```
var a = "42"  
var b = 42;  
  
a == b;    // true  
a === b;   // false
```

double equal comparison algorithm:

<http://www.ecma-international.org/ecma-262/8.0> in section 7.2.13

To boil down a whole lot of details to a few simple takeaways, and help you know whether to use `==` or `===` in various situations, here are my simple rules:

- If either value (aka side) in a comparison could be the true or false value, avoid `==` and use `===`.
- If either value in a comparison could be of these specific values (0, "", or [] -- empty array), avoid `==` and use `===`.
- In all other cases, you're safe to use `==`. Not only is it safe, but in many cases it simplifies your code in a way that improves readability.

The `!=` non-equality form pairs with `==`, and the `!==` form pairs with `===`. All the rules and observations we just discussed hold symmetrically for these non-equality comparisons.

Special note

If we compare two non-primitive values, like objects, functions and array.

```
var a = [1,2,3];  
var b = [1,2,3];  
var c = "1,2,3";
```

```
a == c; // true  
a == b; // true  
a == b; // false
```

Inequality

The `<`, `>`, `<=`, `>=` operators are used for inequality.

Typically they will be used with ordinally comparable values like numbers. It's easy to understand that `3 < 4`.

But JavaScript string values can also be compared for inequality, using typical alphabetic rules (`"bar" < "foo"`).

```
var a = 41;  
var b = "42";  
var c = "43";
```

```
a < b; // true  
b < c; // true
```

What's happen here ?

Conditionals

IF

If statement takes a condition if it is true, do the following

Example :

```
var bank_balance = 302.13;  
var amount = 99.99;  
  
if (amount < bank_balance) {  
  console.log("I want to buy this phone!");  
}
```

IF - ELSE

If statement takes a condition if it is **true**, do the following block code but it is **false** do the following else block code.

```
var bank_balance = 302.13;
var amount = 99.99;

if (amount < bank_balance) {
  console.log("I want to buy this phone!");
} else {
  console.log("I don't want to buy this phone!");
}
```

IF - ELSE - IF

```
if (a == 2) {
  // do something
} else if (a == 10) {
  // do another thing
} else if (a == 42) {
  // do yet another thing
} else {
  // fallback to here
}
```

SWITCH

```
switch (a) {  
  case 2:  
    // do something  
    break;  
  case 10:  
    // do another thing  
    break;  
  case 42:  
    // do yet another thing  
    break;  
  default:  
    // fallback to here  
}
```

The **break** is important if you want only the statement(s) in one case to run. If you omit break from a case, and that case matches or runs, execution will continue with the next case's statements **regardless of that case matching**.

```
switch (a) {  
  case 2:  
  case 10:  
    // some cool stuff  
    break;  
  case 42:  
    // other stuff  
    break;  
  default:  
    // fallback  
}
```

Loops

While

A loop includes the test condition as well as a block (typically as { .. }). Each time the loop block executes, that's called an iteration.

```
while (numOfCustomers > 0) {  
    console.log( "How may I help you?" );  
    // help the customer...  
    numOfCustomers = numOfCustomers - 1;  
}  
// versus:  
do {  
    console.log( "How may I help you?" );  
    // help the customer...  
    numOfCustomers = numOfCustomers - 1;  
} while (numOfCustomers > 0);
```

For

```
for (statement 1; statement 2; statement 3) {  
    code block to be executed  
}
```

- **Statement 1** is executed before the loop (the code block) starts.
- **Statement 2** defines the condition for running the loop (the code block).
- **Statement 3** is executed each time after the loop (the code block) has been executed.

Break and continue

- The break statement "jumps out" of a loop.
- The continue statement "jumps over" one iteration in the loop.

Functions

How to declare a function ?

```
function myFunction(arguments) {  
  // code that will be executed by function  
}
```

How to call a function ?

```
function printAmount(amt) {  
    console.log( amt.toFixed( 2 ) );  
}
```

```
function formatAmount() {  
    return "$" + amount.toFixed( 2 );  
}
```

```
var amount = 99.99;
```

```
printAmount( amount * 2 );           // "199.98"
```

```
amount = formatAmount();
```

```
console.log( amount );               // "$99.99"
```

[https://github.com/ceulain/Advanced-Jav
ascript](https://github.com/ceulain/Advanced-JavaScript)