TWeb

Foundations of JavaScript

Bertil Chapuis

Overview of Today's Class

- Quiz about last week's lecture
- Correction of last week's assignment
- JavaScript
  - Values, Types, and Operators
  - Program Structure
  - Functions
- Developper Tools
  - Chrome DevTools
  - Visual Studio Code
- Introduction of next week's assignment

Quiz

Speakup

You can answer to the following Quiz on Speakup.

http://www.speakup.info/

Room Number: **XXXXX**

Once connected, answer to the first test question.

❓ Question 1

A quelle couche de la Suite des protocoles Internet (Internet Protocol Suite) le protocol TCP appartient-il?

A) Application

B) Transport

C) Internet

D) Link

❓ Question 2

Quel protocol est utilisé par DNS pour transférer les fichiers de Zone?

A) HTTP

B) TCP

C) IP

D) UDP

❓ Question 3

Qu'est qu'un Root Server dans le Domain Name System (DNS)?

A) Un serveur qui gère les sous domaine d'un nom de domaine racine (www.domaine.com, web.domaine.com, etc.)

B) Un serveur qui gère la racine des noms appartenant à un domaine (domaine1.com, domaine2.com, etc.)

C) Un serveur qui gère la racine des noms de domaine (com, org, net, etc.)

D) Aucune affirmation correcte

Question 4

Dans le domaine `www.example.com`, qu'est-ce que le sous-domaine (subdomain)?

A) www

B) example

C) com

D) Aucune affirmation correcte

❓ Question 5

A quoi sert un DNS lookup?

A) A assigner une adresse IP à un nom de domaine?

B) A obtenir une connexion HTTP à partir d'un nom de domaine?

C) A visualiser la route prise par les packets sur le réseau?

D) Aucune affirmation correcte

❓ Question 6

A quelle partie de l'url correspond le fragment?

`https://tim:1234@example.com:443/index.html?param=value#home`

A) ?param=value

B) /index.html

C) tim:1234

D) #home

E) https://

❓ Question 7

Une requête HTTP contient toujours:

A) une méthode

B) une resource

C) des headers

D) un body

E) un port de destination

F) une code de status (status code)

G) un user agent (User-Agent)

❓ Question 8

Parmis les éléments suivants, identifiez le ou lesquels correspondent à du HTML valide:

A) `<a href="https://www.heig-vd.ch">Heig-vd<a>`

B) `<img src="undefined" />`

C) `<p>Mon paragraphe</p>`

D) `<!DOCTYPE html>`

E) `Aucun element correct`

❓ Question 9

Qu'est ce que le DOM?

A) Un arbre (tree) qui représente le CSS en mémoire

B) La strucutre d'un fichier HTML sauvée sur le disque

C) Un language de programmation permettant de manipuler des documents

D) Aucune réponse correcte

❓ Question 10

Qu'est-ce qu'une déclaration CSS?

A) Une paire propriété et valeur

B) Un bloc contenant une liste de propriétés CSS

C) Un sélecteur CSS associé à une liste de propriétés CSS

D) La valeur d'une propriété CSS

Question 11

Quelles elements HTML de la structure suivante sont sélectionnés par le selecteur `ul li:nth-child(1)`?

```
<ul>
    <li>A</li>
    <li>B</li>
    <li>C
        <ul>
            <li>D</li>
            <li>E
                <ol>
                    <li>F</li>
                    <li>G</li>
                </ol>
            </li>
        </ul>
    </li>
</ul>
```

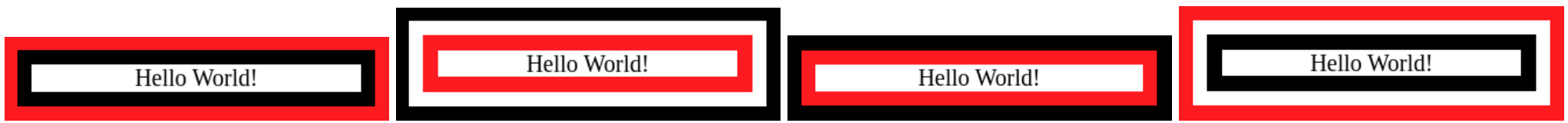
Bonne réponses: A, D, F

Question 12

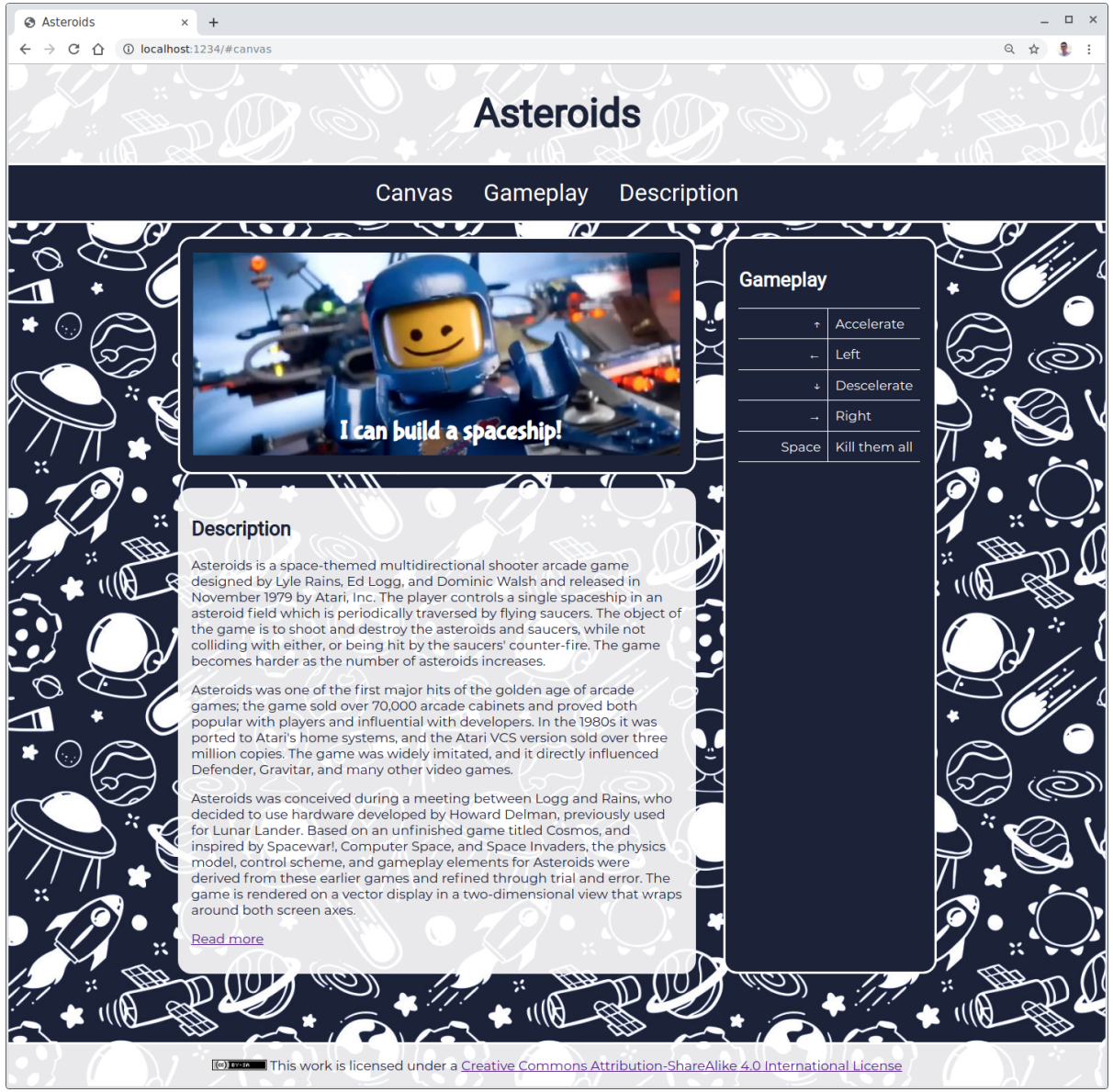
A quel rendu correspond le code HTML/CSS suivant? (Bonne réponse: C)

```html
<div id="a">
  <div id="b">
    Hello World!
  </div>
</div>
```

```css
#a {
  display: block;
  padding: 10px;
  background-color: red;
  border: solid 10px black;
}
#b {
  text-align: center;
  display: block;
  background-color: white;
}
```



Correction


Asteroids
localhost:1234/#canvas
Asteroids
Canvas
Gameplay
Description
I can build a spaceship!
Gameplay
↑ Accelerate
← Left
↓ Descelerate
→ Right
Space Kill them all
Description
Asteroids is a space-themed multidirectional shooter arcade game designed by Lyle Rains, Ed Logg, and Dominic Walsh and released in November 1979 by Atari, Inc. The player controls a single spaceship in an asteroid field which is periodically traversed by flying saucers. The object of the game is to shoot and destroy the asteroids and saucers, while not colliding with either, or being hit by the saucers' counter-fire. The game becomes harder as the number of asteroids increases.
Asteroids was one of the first major hits of the golden age of arcade games; the game sold over 70,000 arcade cabinets and proved both popular with players and influential with developers. In the 1980s it was ported to Atari's home systems, and the Atari VCS version sold over three million copies. The game was widely imitated, and it directly influenced Defender, Gravitar, and many other video games.
Asteroids was conceived during a meeting between Logg and Rains, who decided to use hardware developed by Howard Delman, previously used for Lunar Lander. Based on an unfinished game titled Cosmos, and inspired by Spacewar!, Computer Space, and Space Invaders, the physics model, control scheme, and gameplay elements for Asteroids were derived from these earlier games and refined through trial and error. The game is rendered on a vector display in a two-dimensional view that wraps around both screen axes.
Read more
This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License

Questions ?

JavaScript

JavaScript*

JavaScript is a lightweight, **interpreted**, or **just-in-time** compiled programming language with first-class functions.

Interpreted

The execution does not require the compilation of the programm into machine-code.

Just-in-time (JIT)

The parts of the the program that are hot (called a lot) are compiled into machine code.

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

JavaScript*

JavaScript is a **prototype-based**, multi-paradigm, **dynamic** language, supporting **object-oriented**, **imperative**, and **declarative** (e.g. functional programming) styles.

Dynamic

Performs at runtime what static languages perform at compilation time (e.g. dynamic typing).

Object Oriented and Prototype-based

Behavior reuse (and inheritence) is performed by cloning and extending objects.

Imperative

Mutates the state of the program using statements.

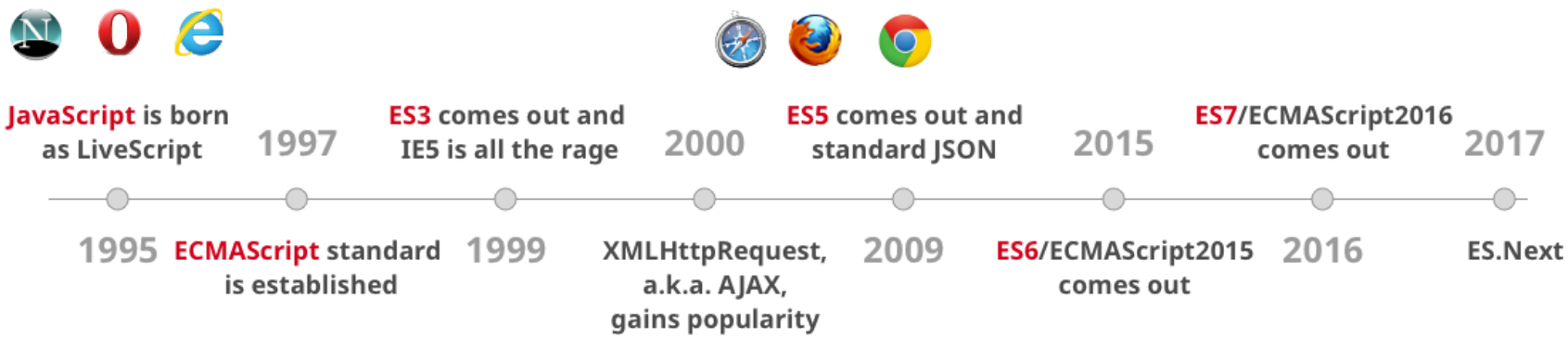
Declarative

A programming style that avoids side effects by describing what a computation should perform.

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

JavaScript and ECMAScript History

ECMAScript (or ES) is a specification created by Ecma International to standardize JavaScript.



From the browser to the server

Server-side JavaScript is not a novel idea (Netscape was already doing it in 1996).

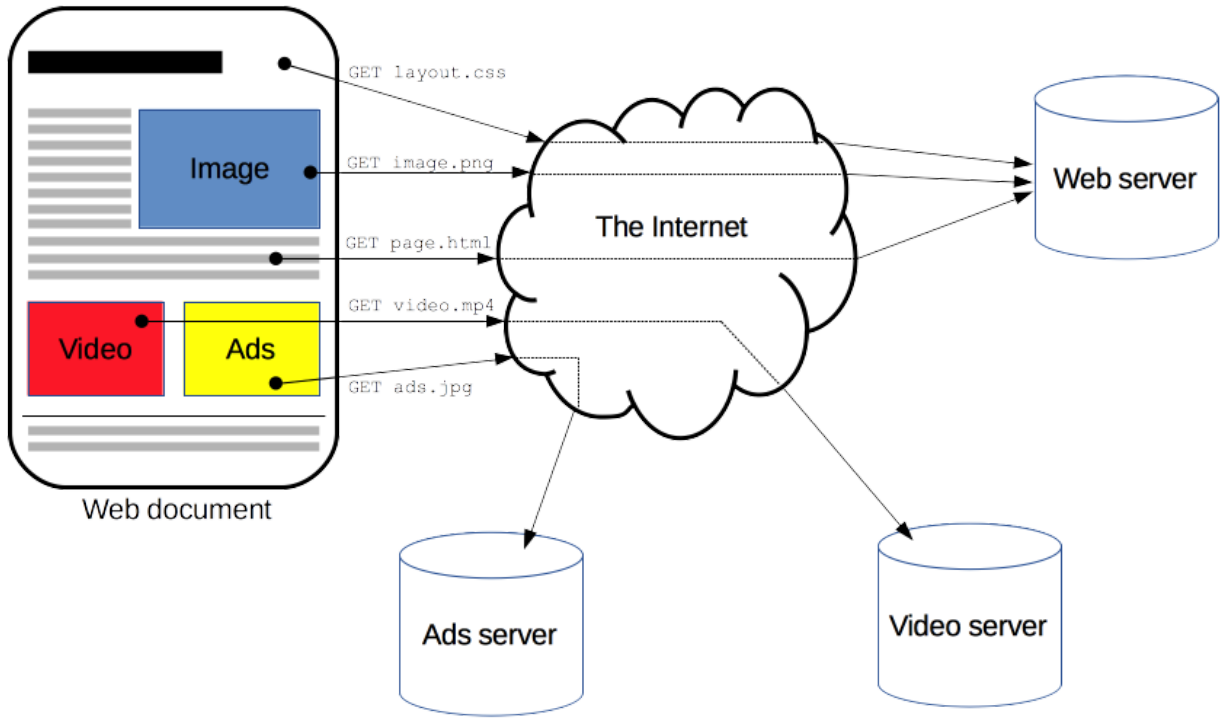
Rhino, a JavaScript engine written in Java by has been released as early as 1997.

In 2008, the first version of Chrome includes V8, an open-source JavaScript engine created by Lars Bak.

In 2009, Ryan Dahl creates node.js, a JavaScript environment based on V8 that runs on the server.

Client-side and Server-side Programming

Today, JavaScript is commonly used in the browser (client-side) and on the server (server-side).

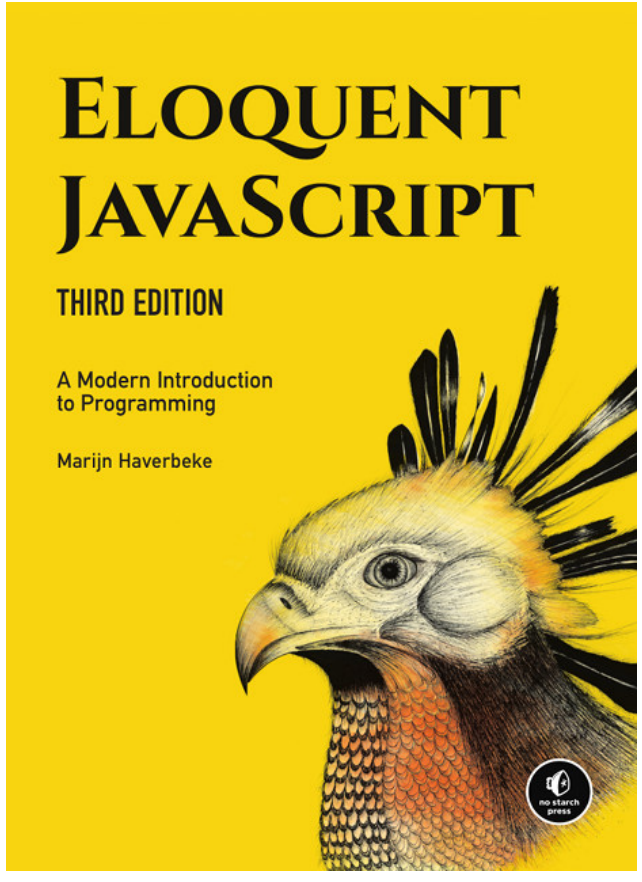


ECMAScript 6 Support*

Feature name	Current browser	Compilers/polyfills: Traceur	Babel 6 + core-js 2	Babel 7 + core-js 2	Babel 7 + core-js 3	Closure 2019.07	TypeScript + core-js 3	es6-shim	Desktop browsers: Kong 4.14[1]	IE 11	Edge 17	Edge 18	FF 60 ESR	FF 68 ESR	FF 69	CH 76, OP 63	CH 77, OP 64	SF 12	SF 12.1	Servers/runtimes: XS6	JXA	Node >=6.5 <7[3]	Node >=8.10 <9[3]	Node >=10.9 <11[3]	Node 11[3]	Node 12.0-12.4[3]	Node 12.5+[3]	DUK 1.8	DUK 2.3	JJS 1.8	JJS 10	GraalVM 19.0.0[4]
	98%	56%	71%	71%	72%	50%	69%	17%	5%	11%	96%	96%	98%	98%	98%	98%	98%	98%	99%	95%	59%	97%	97%	98%	98%	98%	98%	2%	26%	7%	28%	98%
Optimisation																																
proper tail calls (tail call optimisation)	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	0/2	0/2	0/2
Syntax																																
default function parameters	7/7	4/7	4/7	4/7	4/7	5/7	5/7	0/7	0/7	0/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	0/7	7/7	7/7	7/7	7/7	7/7	7/7	0/7	0/7	0/7	4/7	7/7
rest parameters	5/5	4/5	3/5	3/5	3/5	2/5	4/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	0/5	0/5	0/5	5/5
spread syntax for iterable objects	15/15	15/15	13/15	13/15	14/15	11/15	14/15	0/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	11/15	15/15	15/15	15/15	15/15	15/15	15/15	0/15	0/15	0/15	0/15	15/15
object literal extensions	6/6	6/6	6/6	6/6	6/6	5/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	5/6	6/6	6/6	6/6	6/6	6/6	6/6	0/6	4/6	0/6	2/6	6/6
for..of loops	9/9	9/9	9/9	9/9	9/9	6/9	9/9	0/9	0/9	0/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	8/9	9/9	9/9	9/9	9/9	9/9	9/9	0/9	0/9	0/9	4/9	9/9
octal and binary literals	4/4	2/4	4/4	4/4	4/4	2/4	4/4	2/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	4/4	0/4	2/4	4/4
template literals	7/7	6/7	6/7	6/7	6/7	5/7	5/7	0/7	0/7	0/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	6/7	6/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	0/7	0/7	0/7	5/7	7/7
RegExp "y" and "u" flags	6/6	4/6	4/6	4/6	4/6	0/6	0/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	2/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	0/6	0/6	0/6	0/6	6/6
destructuring, declarations	22/22	20/22	21/22	21/22	21/22	20/22	21/22	0/22	0/22	0/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	21/22	19/22	22/22	22/22	22/22	22/22	22/22	22/22	0/22	0/22	0/22	0/22	22/22
destructuring, assignment	24/24	23/24	24/24	24/24	24/24	22/24	24/24	0/24	0/24	0/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	21/24	24/24	24/24	24/24	24/24	24/24	24/24	0/24	0/24	0/24	0/24	24/24
destructuring, parameters	24/24	19/24	21/24	21/24	21/24	20/24	21/24	0/24	0/24	0/24	23/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	23/24	18/24	24/24	24/24	24/24	24/24	24/24	24/24	0/24	0/24	0/24	0/24	24/24
Unicode code point escapes	2/2	1/2	1/2	1/2	1/2	1/2	1/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	2/2	0/2	0/2	2/2
new.target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	1/2	0/2	0/2	2/2
Bindings																																
const	18/18	16/18	16/18	16/18	16/18	16/18	16/18	0/18	2/18	14/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	12/18	18/18	18/18	18/18	18/18	18/18	18/18	1/18	4/18	12/18	16/18	18/18
let	14/14	12/14	12/14	12/14	12/14	12/14	12/14	0/14	0/14	12/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	0/14	14/14	14/14	14/14	14/14	14/14	14/14	0/14	0/14	10/14	12/14	14/14
block-level function declaration[18]	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes
Functions																																
arrow functions	13/13	11/13	9/13	9/13	9/13	10/13	9/13	0/13	0/13	0/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	12/13	0/13	13/13	13/13	13/13	13/13	13/13	13/13	0/13	0/13	0/13	6/13	13/13
class	24/24	17/24	19/24	19/24	19/24	14/24	19/24	0/24	0/24	0/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	18/24	24/24	24/24	24/24	24/24	24/24	24/24	0/24	0/24	0/24	0/24	24/24
super	8/8	7/8	4/8	4/8	4/8	7/8	7/8	0/8	0/8	0/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	7/8	8/8	8/8	8/8	8/8	8/8	8/8	0/8	0/8	0/8	0/8	8/8
generators	27/27	24/27	24/27	24/27	24/27	19/27	22/27	0/27	0/27	0/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	0/27	27/27	27/27	27/27	27/27	27/27	27/27	0/27	0/27	0/27	0/27	27/27
Built-ins																																
typed arrays	46/46	0/46	45/46	45/46	46/46	0/46	46/46	0/46	8/46	16/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	17/46	20/46	19/46	20/46	46/46
Map	19/19	14/19	19/19	19/19	19/19	15/19	19/19	15/19	0/19	8/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	18/19	19/19	19/19	19/19	19/19	19/19	19/19	0/19	0/19	0/19	16/19	19/19
Set	19/19	14/19	19/19	19/19	19/19	15/19	19/19	15/19	0/19	8/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	18/19	19/19	19/19	19/19	19/19	19/19	19/19	0/19	0/19	0/19	16/19	19/19
WeakMap	12/12	6/12	12/12	12/12	12/12	9/12	12/12	0/12	0/12	6/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	11/12	11/12	12/12	12/12	12/12	12/12	12/12	12/12	0/12	0/12	0/12	10/12	12/12
WeakSet	11/11	5/11	11/11	11/11	11/11	8/11	11/11	0/11	0/11	0/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	11/11	10/11	10/11	11/11	11/11	11/11	11/11	11/11	11/11	0/11	0/11	0/11	9/11	11/11
Proxy[23]	34/34	0/34	0/34	0/34	0/34	0/34	0/34	0/34	0/34	0/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	34/34	0/34	34/34	34/34	34/34	34/34	34/34	34/34	1/34	15/34	0/34	0/34	34/34
Reflect[24]	20/20	0/20	15/20	15/20	15/20	13/20	15/20	14/20	0/20	0/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	16/20	0/20	20/20	20/20	20/20	20/20	20/20	20/20	6/20	14/20	0/20	0/20	20/20
Promise	8/8	4/8	8/8	8/8	8/8	6/8	8/8	7/8	0/8	0/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	0/8	8/8	8/8	8/8	8/8	8/8	8/8	0/8	0/8	0/8	0/8	8/8
Symbol	12/12	4/12	9/12	9/12	9/12	3/12	8/12	2/12	0/12	0/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	0/12	12/12	0/12	11/12	12/12
well-known symbols[25]	26/26	1/26	15/26	15/26	22/26	1/26	22/26	0/26	0/26	0/26	17/26	17/26	26/26	26/26	26/26	26/26	26/26	26/26	26/26	25/26	22/26	26/26	26/26	26/26	26/26	26/26	26/26	0/26	6/26	0/26	1/26	26/26

* https://kangax.github.io/compat-table/es6/

Eloquent Javascript*



* https://eloquentjavascript.net/

Client-side Javascript

Include JavaScript stored in the HTML:

```
<script type='text/javascript'>
  console.log('Hello, World!');
  document.writeln('Hello, World!')
</script>
```

Include JavaScript stored in a separate file:

```
<script src="file:///home/bchapuis/Projects/github.com/tweb/slides/script.js"></script>
```

The `async` attribute indicates that the browser should load the script asynchronously and then execute it as soon as it's downloaded.

The `defer` attribute indicates that the browser should execute the script after the document has been parsed but before the DOM content has been loaded.

The `type="module"` attribute causes the code to be treated as a JavaScript module (ECMAScript 6).

Server-side Javascript*

After installing nodejs, a REPL (Read-Eval-Print-Loop) can be obtained by typing the node command:

```
$ node
Welcome to Node.js v12.8.0.
Type ".help" for more information.
> console.log("Hello, World!")
Hello World!
```

* https://nodejs.org/api/repl.html

Get to know the REPL commands*

- `.clear` - Resets the REPL context to an empty object and clears any multi-line expression currently being input.
- `.exit` - Close the I/O stream, causing the REPL to exit.
- `.help` - Show this list of special commands.
- `.save` - Save the current REPL session to a file: > .save ./file/to/save.js
- `.load` - Load a file into the current REPL session. > .load ./file/to/load.js
- `.editor` - Enter editor mode (-D to finish, -C to cancel).

* https://nodejs.org/api/repl.html#repl_commands_and_special_keys

JavaScript's Types

ECMAScript defines 7 **primitive** (Immutable) types for values.

```
3.14; // Number
true; // Boolean
"Heig-vd"; // String
undefined; // Undefined
null; // Null
9007199254740992n; // BigInt
Symbol("Symbol") // Symbol
```

ECMAScript defines a special mutable type called **object** for collections of properties (objects and array).

```
{prop: "value"}; // Object
```

In a dynamic language you don't specify the type when you declare a variable and the type of a variable can change.

* https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#Data_types

JavaScript Operators

Typeof

The `typeof` operator returns a string indicating the type of the unevaluated operand.

Assignment

An assignment operator assigns a value to its left operand based on the value of its right operand.

```
a = 1;

// arithmetic assignments
a += 1; // addition
a -= 1; // subtraction
a *= 1; // multiplication
a /= 1; // division
a %= 1; // modulo
a **= 1; // exponentiation
```

JavaScript Operators

Arithmetic

```
1 + 1; // addition
1 - 1; // subtraction
1 / 1; // division
1 * 1; // multiplication
1 % 1; // modulo
1 ** 1; // exponentiation
```

String

```
"con" + "cat" + "e" + "nate";
`PI = ${Math.PI}`; // template literals
```

Logical

```
!true // false
true && false // false
true || false // true
true ? true : false // true
```

JavaScript Operators

Comparison

```
1 < 2;
2 > 1;
1 == 1;
1 != 2;
```

Automatic Type Conversion

Automatic type conversion is a the root of many issues.

```
"1" == 1 // true
false == 0 // true
8 * null // 0
```

Strict equality

Strict equality compares both the type and the value.

```
"1" === 1 // false
```

JavaScript Statements

The `var` statement declares a variable, optionally initializing it to a value. The **scope** of a variable declared with var is its current execution context, which is either the enclosing function or, for variables declared outside any function, global.

```
var x = 1;
{var x = 2;} // same variable
console.log(x); // 2
```

The `let` statement declares a block scope local variable, optionally initializing it to a value. The **scope** of a variable declared with let is limited to its block or expression.

```
let x = 1;
{let x = 2;} // different variable
console.log(x); // 1
```

Constants are block-scoped, much like variables defined using the let statement. The value of a constant can't be changed through reassignment, and it can't be redeclared.

```
const x = 1;
x = 2; // raises an error
```

A single statement can define multiple variables or constants: `let one = 1, two = 2;`.

JavaScript Conditional Execution

In JavaScript, conditional execution is created with the if keyword.

```
let num = 1;
if (num < 10) {
  console.log("Small!");
} else if (num < 100) {
  console.log("Medium");
} else {
  console.log("Large");
}
```

JavaScript While and Do While

While and do while are used to loop until a condition is met.

```
let num = 0;
while (num < 10) {
  console.log(num);
  num += 1;
}
```

```
let num = 0;
do {
  console.log(num);
  num += 1;
} while (num < 10);
```

JavaScript For Loops

The classic `for` statement is used to loop a given number of times over a block.

```
for (let num = 0; num < 10; num++) {
  console.log(num);
}
```

The `for...in` statement iterates over the enumerable properties of an object.

```
var object1 = {a: 1, b: 2, c: 3};
for (var property1 in object1) {
  string1 += object1[property1];
}
```

The `for...of` statement creates a loop iterating over iterable objects.

```
let nums = [0, 1, 2, 3, 4, 5, 6, 7, 8 , 9];
for (let num of nums) {
  console.log(num);
}
```

JavaScript Break and Continue*

The `break` statement terminates the current loop.

The `continue` statement terminates execution of the current iteration and continues execution of the loop with the next iteration.

`break` and `continue` can also be used with labelled statements.

```
mylabel:
for (let num = 0; num < 5; num++) {
  if (num === 5) {
    continue mylabel;
  }
  console.log(num)
}
```

* https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/label

JavaScript Switch

Sometimes a switch is more direct than an `if...elseif...else` statement.

```
let val = "a";
switch(val) {
  case "a":
    doSomething(val);
    break;
  case "b":
    doSomethingElse(val);
    break;
  default:
    doNothing(val);
    break;
}
```

JavaScript Exceptions

In Javascript, exceptions can be handled using the `try...catch` statement.

```
try {
  variable; // ReferenceError: variable is not defined
} catch (error) {
  // Fails silently
}
```

Exceptions can be triggered using `throw` and `Error`:

```
throw new Error("AAHHARG!!!");
```

JavaScript Functions

A function is created with an expression that starts with the keyword `function` and can be assigned to a regular regular variable. It can have parameters and may `return` a value.

```
var square = function(x) {
  return x * x;
}
```

Declaration Notation

```
function square(x) {
  return x * x;
}
```

Arrow Functions

```
var square = x => x * x
```

```
var square = (x) => {
  return x * x;
}
```

JavaScript Function Parameters

Function parameters can be made optional by specifying default values.

```
var square = function(x = 2) {
  return x * x;
}
square() // 4
```

JavaScript Function Scopes

The scope of a variable, is the part of the program in which it is visible.

The scope of the variables defined with `var` outside of a function is `global`, it is visible everywhere.

The scope of the variables defined with `let` and `const` are `local` to the block that they are declared in.

The scope of function parameters can be referenced only in that function and are `local`.

Local variables added to the **call stack** every time a function is called and freed it returns.

Closure

A closure is the combination of a function and the local scope within which that function was declared.

```
function wrap(value) {
  let v = value;
  return () => v;
}
wrap(1)() // 1
```

JavaScript Recursion

It is fine for a function to call itself.

```
function factorial(n) {
  return n == 1 ? n : n * factorial(n-1);
}
factorial(5) // 5 * 4 * 3 * 2 * 1 = 120
```

.. as long as it does not overflow the call stack.

JavaScript DevTools

- Experiment with the Chrome DevTools
- Experiment with the Visual Studio Code

JavaScript Exercise

- Implement a recursive function that computes the fibonacci sequence.
- Look the call stack with the Chrome DevTools.

Questions ?

Questions about Today's Lecture

- JavaScript
  - Values, Types, and Operators
  - Program Structure
  - Functions
- Developper Tools
  - Chrome DevTools
  - Visual Studio Code

Group Assignment

Group Assignment

- Same groups as last week
- Install Visual Studio Code, Node.js and Docker
- If needed, watch Olivier's webcasts on client-side and server-side debugging.
- Go to the Github Classroom and start exercise 2 (Introduction to JavaScript)
- **Interact** with the assistants if needed… ;)
- The repository will be frozen **next Tuesday at 12am**