# JAVASCRIPT

## LANGUAGE DU FUTUR ?

Par Valentin Demeusy

# OBJECTIFS

# OBJECTIF

- Présentation JavaScript
- Challenges à relever
- Solutions amenées par la communauté et les évolutions du langage

# QU'EST CE QUE JAVASCRIPT ?

# PRÉSENTATION

- Langage interprété à typage faible et dynamique
- Connu comme le langage des pages Web
- Dernière version stable : ECMAScript2016, 17 Juin 2016
- Dernière version : ECMAScript2017

# LES MOTEURS

- Navigateurs :
  - Chrome (V8)
  - Firefox (Spider Monkey)
  - Edge (Chakra)
  - Safari (JavaScriptCore)
- Node.js (V8)

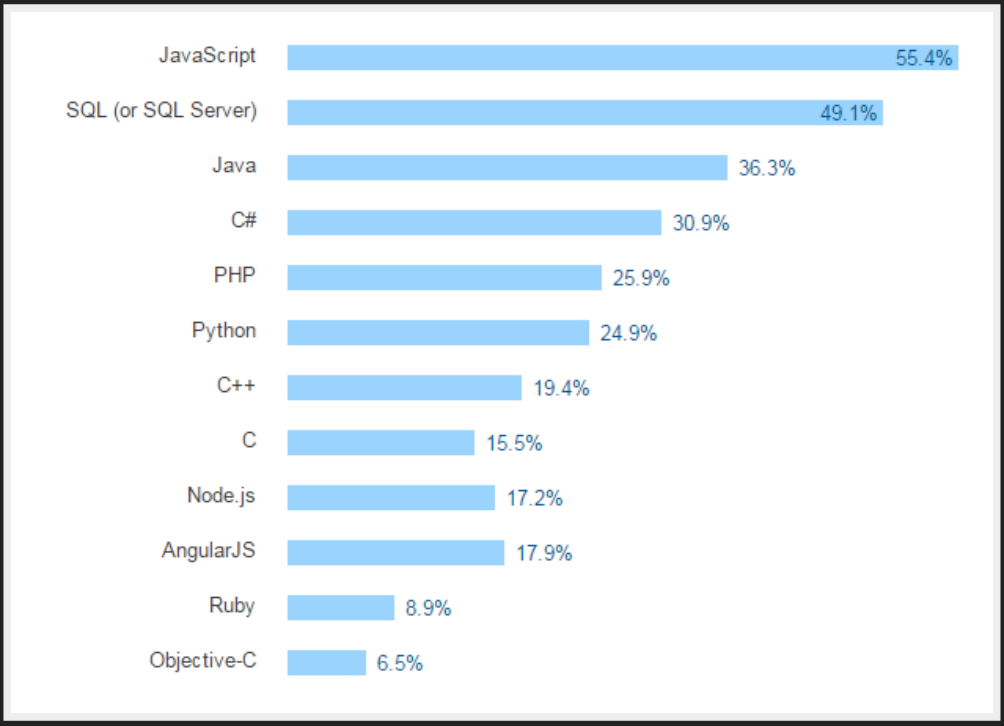
# UN LANGAGE POUR TOUT FAIRE

- Frontend via les navigateurs
- Serveurs via Node.js
- Applications bureau via Electron
- Applications mobiles via Cordova

# DÉVELOPPEMENT ACTIF

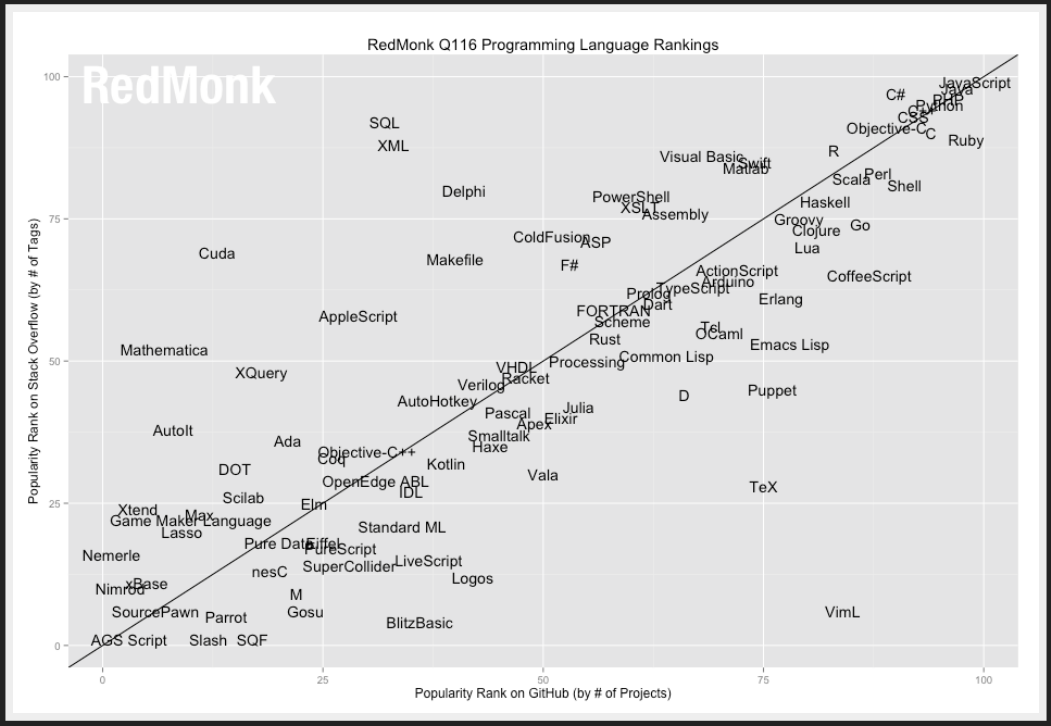
- Utilisé par 94% des sites Web (source)
- Langage n°1 sur GitHub depuis 2013 (source)
- 390 000 packages sur NPM, 525 packages/jour (source)
- Le plus populaire et le plus discuté sur StackOverflow (source)
- 420Ko/page en moyenne, en augmentation (source)

# POPULARITÉ




Pourcentage des développeurs connaissant le langage (Sondage Annuel Stackoverflow)

# POPULARITÉ




Nombre de tags Overflow en fonction du nombre de projets github (Source)

# COMPATIBILITÉ

| Feature name | Current browser | Traceur | Babel + core-js[2] | Closure | Type-Script + core-js | es6-shim | KQ 4.14[3] | IE 11 | Edge 13[4] | Edge 14[4] | Edge 15[4] | FF 45 ESR | FF 51 | FF 52 Beta | FF 53 Aurora | FF 54 Nightly | CH 56, OP 43[1] | CH 57, OP 44[1] | CH 58, OP 45[1] | SF 9 | SF 10 | SF 10.1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 97% | 56% | 71% | 48% | 59% | 18% | 5% | 11% | 83% | 93% | 95% | 86% | 94% | 94% | 97% | 97% | 97% | 97% | 97% | 54% | 100% | 100% |
| | | Compilers/polyfills | | | | | Desktop browsers | | | | | | | | | | | | | | | |
| **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 | 0/2 | 0/2 | 0/2 | 2/2 | 2/2 |
| **Syntax** | | | | | | | | | | | | | | | | | | | | | | |
| default function parameters | 7/7 | 4/7 | 4/7 | 4/7 | 5/7 | 0/7 | 0/7 | 0/7 | 0/7 | 7/7 | 7/7 | 4/7 | 6/7 | 6/7 | 7/7 | 7/7 | 7/7 | 7/7 | 7/7 | 0/7 | 7/7 | 7/7 |
| rest parameters | 5/5 | 4/5 | 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 | 5/5 | 0/5 | 5/5 | 5/5 |
| spread (...) operator | 15/15 | 15/15 | 13/15 | 12/15 | 4/15 | 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 | 15/15 | 9/15 | 15/15 | 15/15 |
| object literal extensions | 6/6 | 6/6 | 6/6 | 4/6 | 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 | 6/6 | 5/6 | 6/6 | 6/6 |
| for..of loops | 9/9 | 9/9 | 9/9 | 6/9 | 3/9 | 0/9 | 0/9 | 0/9 | 7/9 | 7/9 | 9/9 | 7/9 | 7/9 | 7/9 | 9/9 | 9/9 | 9/9 | 9/9 | 9/9 | 8/9 | 9/9 | 9/9 |
| octal and binary literals | 4/4 | 2/4 | 4/4 | 4/4 | 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 |
| template literals | 5/5 | 4/5 | 4/5 | 3/5 | 3/5 | 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 | 5/5 | 5/5 | 5/5 | 5/5 |
| RegExp "y" and "u" flags | 5/5 | 3/5 | 3/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 5/5 | 5/5 | 5/5 | 2/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 0/5 | 5/5 | 5/5 |
| destructuring, declarations | 22/22 | 20/22 | 21/22 | 19/22 | 15/22 | 0/22 | 0/22 | 0/22 | 0/22 | 21/22 | 22/22 | 19/22 | 21/22 | 21/22 | 22/22 | 22/22 | 22/22 | 22/22 | 22/22 | 19/22 | 22/22 | 22/22 |
| destructuring, assignment | 24/24 | 23/24 | 24/24 | 17/24 | 19/24 | 0/24 | 0/24 | 0/24 | 0/24 | 23/24 | 24/24 | 21/24 | 23/24 | 23/24 | 24/24 | 24/24 | 24/24 | 24/24 | 24/24 | 21/24 | 24/24 | 24/24 |
| destructuring, parameters | 23/23 | 19/23 | 20/23 | 18/23 | 15/23 | 0/23 | 0/23 | 0/23 | 0/23 | 22/23 | 23/23 | 18/23 | 20/23 | 20/23 | 23/23 | 23/23 | 23/23 | 23/23 | 23/23 | 18/23 | 23/23 | 23/23 |
| Unicode code point escapes | 2/2 | 1/2 | 1/2 | 1/2 | 1/2 | 0/2 | 0/2 | 0/2 | 2/2 | 2/2 | 2/2 | 1/2 | 1/2 | 1/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 |
| new.target | 2/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 1/2 | 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 |
| **Bindings** | | | | | | | | | | | | | | | | | | | | | | |
| const | 16/16 | 14/16 | 14/16 | 14/16 | 14/16 | 0/16 | 2/16 | 12/16 | 12/16 | 16/16 | 16/16 | 12/16 | 16/16 | 16/16 | 16/16 | 16/16 | 16/16 | 16/16 | 16/16 | 1/16 | 16/16 | 16/16 |
| let | 12/12 | 10/12 | 10/12 | 10/12 | 10/12 | 0/12 | 0/12 | 10/12 | 10/12 | 12/12 | 12/12 | 10/12 | 12/12 | 12/12 | 12/12 | 12/12 | 12/12 | 12/12 | 12/12 | 0/12 | 12/12 | 12/12 |

Table d'implémentation d'ECMAScript dans les différents moteurs (Source)

# POLYFILLS

- Pas de numéro de version
- Polyfills pour les "vieux" moteurs
- Babel pour transpiler les nouvelles fonctionnalités

# LES REPROCHES

# LES REPROCHES

- Pas de typage fort
- Callback Hell
- Mauvaise lisibilité
  - Callback Hell
  - Syntaxe étrange
- Pas de mémoire partagée
- Lenteur

# TYPAGE

# ABSENCE DE TYPAGE FORT

- Rend le débuggage difficile
- Autocomplétion difficile
- Exécution plus lente

# ALTERNATIVES

- Dart
- TypeScript
- asm.js

  Efficaces mais requierent un préprocesseur

# EXEMPLE TYPESCRIPT

```
class Greeter {
    constructor(public greeting: string) { }
    greet() {
        return "<h1>" + this.greeting + "</h1>";
    }
};

var greeter = new Greeter("Hello, world!");

document.body.innerHTML = greeter.greet();
```

# CALLBACK HELL

# CALLBACK HELL

```
function foo(finalCallback) {
  request.get(url1, function(err1, res1) {
    if (err1) { return finalCallback(err1); }
    request.post(url2, function(err2, res2) {
      if (err2) { return finalCallback(err2); }
      request.put(url3, function(err3, res3) {
        if (err3) { return finalCallback(err3); }
        request.del(url4, function(err4, res4) {
          // let's stop here
          if (err4) { return finalCallback(err4); }
          finalCallback(null, "whew all done");
        })
      })
    })
  })
}
```

# PROMISES (ES6)

```
function foo() {
  return request.getAsync(url1)
  .then(function(res1) {
    return request.postAsync(url2);
  }).then(function(res2) {
    return request.putAsync(url3);
  }).then(function(res3) {
     return request.delAsync(url4);
  }).then(function(res4) {
     return "whew all done";
  });
}
```

# ASYNC/WAIT (ES7)

```
async function foo() {
  var res1 = await request.getAsync(url1);
  var res2 = await request.getAsync(url2);
  var res3 = await request.getAsync(url3);
  var res4 = await request.getAsync(url4);
  return "whew all done";
}
```

# SYNTAXE

# CLASSES

```
function MyObjectA () {}
MyObjectA.prototype = {
  myMethod: function () {}
};

var obj = new MyObjectA();
```

# CLASSES

```
class MyObjectC {
  myMethod () {

  }
}
var obj = new MyObjectC();
```

# PORTÉES DES VARIABLES

```
function fn() {
  let foo = "bar";
  var foo2 = "bar";
  if (true) {
    let foo; // pas d'erreur, foo === undefined
    var foo2; // foo2 est en réalité écrasé !
    foo = "qux";
    foo2 = "qux";
    console.log(foo); // "qux"
    console.log(foo2); // "qux"
  }
  console.log(foo);// "bar"
  console.log(foo2); // "qux"
}
```

# MODULES

- Evite la pollution de l'espace principal
- Node.js
- Webpack
- Browserify

# MODULES

```
const myModule = require("./my-module.js")
```

# PARALLÉLISME

# WEB WORKER

- Programme s'éxécutant en parallèle dans un environnement indépendant
- Communication par message entre les workers et le main
- Données envoyées par copies ou "transférées"

# GPU

- Utilisation du GPU via WebGL
- Les shaders traitent les données en parallèle
- Turbo.js

# SHARED ARRAY BUFFER

```
// main.js

const worker = new Worker('worker.js');

// To be shared
const sharedBuffer = new SharedArrayBuffer( // (A)
    10 * Int32Array.BYTES_PER_ELEMENT); // 10 elements

// Share sharedBuffer with the worker
worker.postMessage({sharedBuffer}); // clone

// Local only
const sharedArray = new Int32Array(sharedBuffer); // (B)
```

# ATOMICS

```
// Initialization before sharing the Array
    Atomics.store(sharedArray, 0, 1);

    // main.js
    Atomics.store(sharedArray, 0, 2);

    // worker.js
    while (Atomics.load(sharedArray, 0) === 1) ;
    console.log(Atomics.load(sharedArray, 0)); // 2
```

# PERFORMANCES

D'après l'article ES proposal: Shared memory and atomics, les gains de performances sont linéaires avec le nombre de web worker jusqu'à 4 workers avec l'algorithme testé. Au delà les performances s'améliorent plus modestement.

# OPTIMISATION

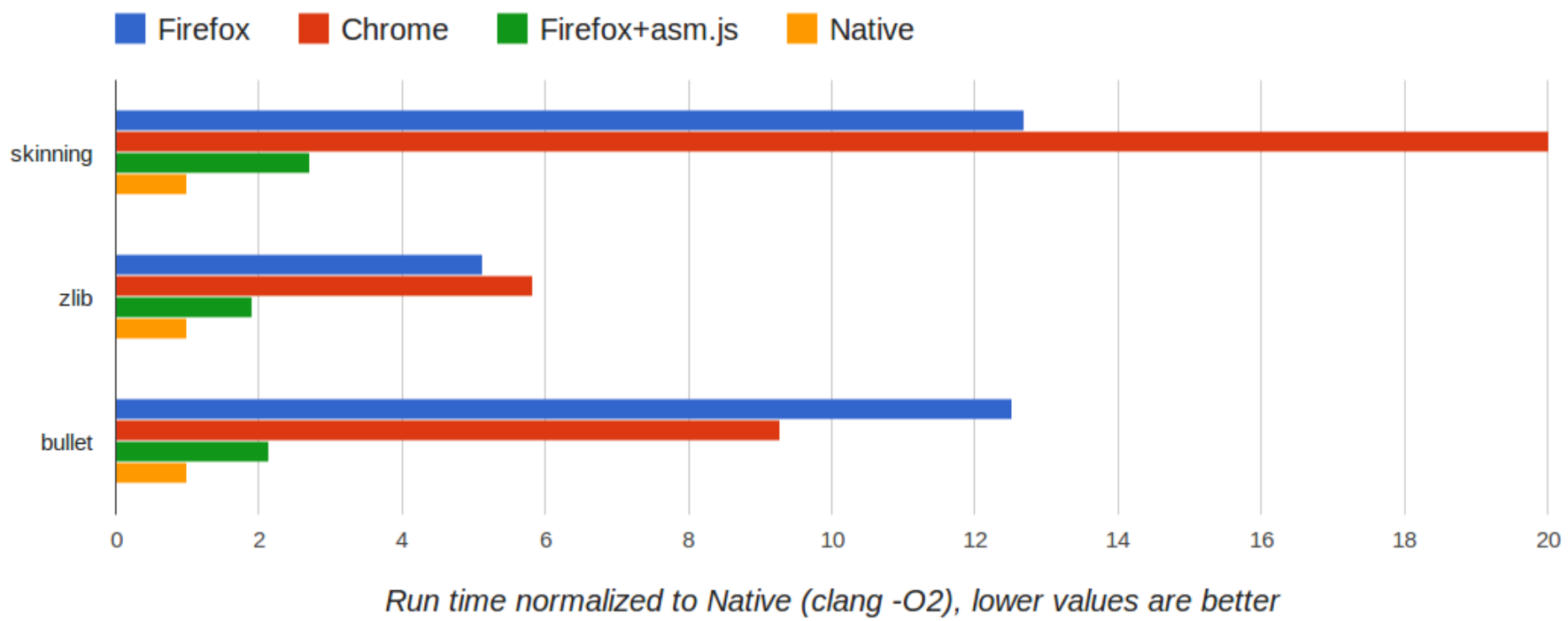
# ASM.JS

- Sous ensemble très restraint de JavaScript, pas d'objets
- Evite le typage dynamique
- Permet de réaliser des optimisations de bas niveau
- Ahead-Of-Time (AOT) VS Just-In-Time (JIT)
- Compilation de C++ en asm.js

# ASM.JS EXEMPLE

```
function compiledCalculation() {
    var x = f()|0;  // x is a 32-bit value
    var y = g()|0;  // so is y
    return (x+y)|0; // 32-bit addition, no type or overflow check
  }
```

# ASM.JS BENCHMARK



# WEBASSEMBLY

- Format binaire d'asm.js
- Moins de temps de chargement
- Moins de temps de parsing
- Même espace sémantique que JavaScript, permet les appels entre JavaScript et WASM

# PERFORMANCE

- Parsing : jusqu'à 20 fois meilleur
- Chargement : 1,5-3 fois meilleur

# CONCLUSION

# CONCLUSION

- Omniprésence
- Evolution rapide

# QUESTIONS ?

Retrouvez cette présentation en ligne sur
stity.github.io/future-javascript