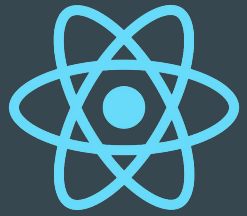


INITIATION À REACT

...

EPSI l'École d'ingénierie informatique

MAGNITUDE LABS

Plan

Jour 1 : JAVASCRIPT

- Présentation
- ES5
  - Découverte
  - Fonctions avancés
  - Jquery
  - Pratique
- ES6
  - Découverte
  - Installation
  - Arrow Fonction
  - Destructeurs
  - Reduce

Plan

Jour 2 : REACT

- Présentation & Installation
- Concept
  - Virtual DOM
  - Component
- Composants React
  - Cycle de vie
  - Props
  - States
  - Pratique
- Projet : Bataille navale

Plan

Jour 3 : REDUX

- Concept & Installation
  - Utilité
  - Architecture
  - Installation
- Intégration de Redux au projet

Plan

Pour aller plus loin

- Application multi-screen : React-navigation ; React Routeur...

- Communication avec un serveur : Firebase
  - Bataille navale multijoueur temps réel

- Création d'application mobile native : React Native

JAVASCRIPT

Présentation

Javascript est un langage de programmation orienté objet

Javascript permet d’écrire des scripts

Langage compilé

Langage pré-compilé

Langage interprété

Langage compilé

Langage pré-compilé

Langage interprété

Langage compilé

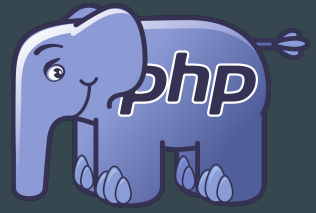
Langage pré-compilé

Langage interprété

Langage compilé

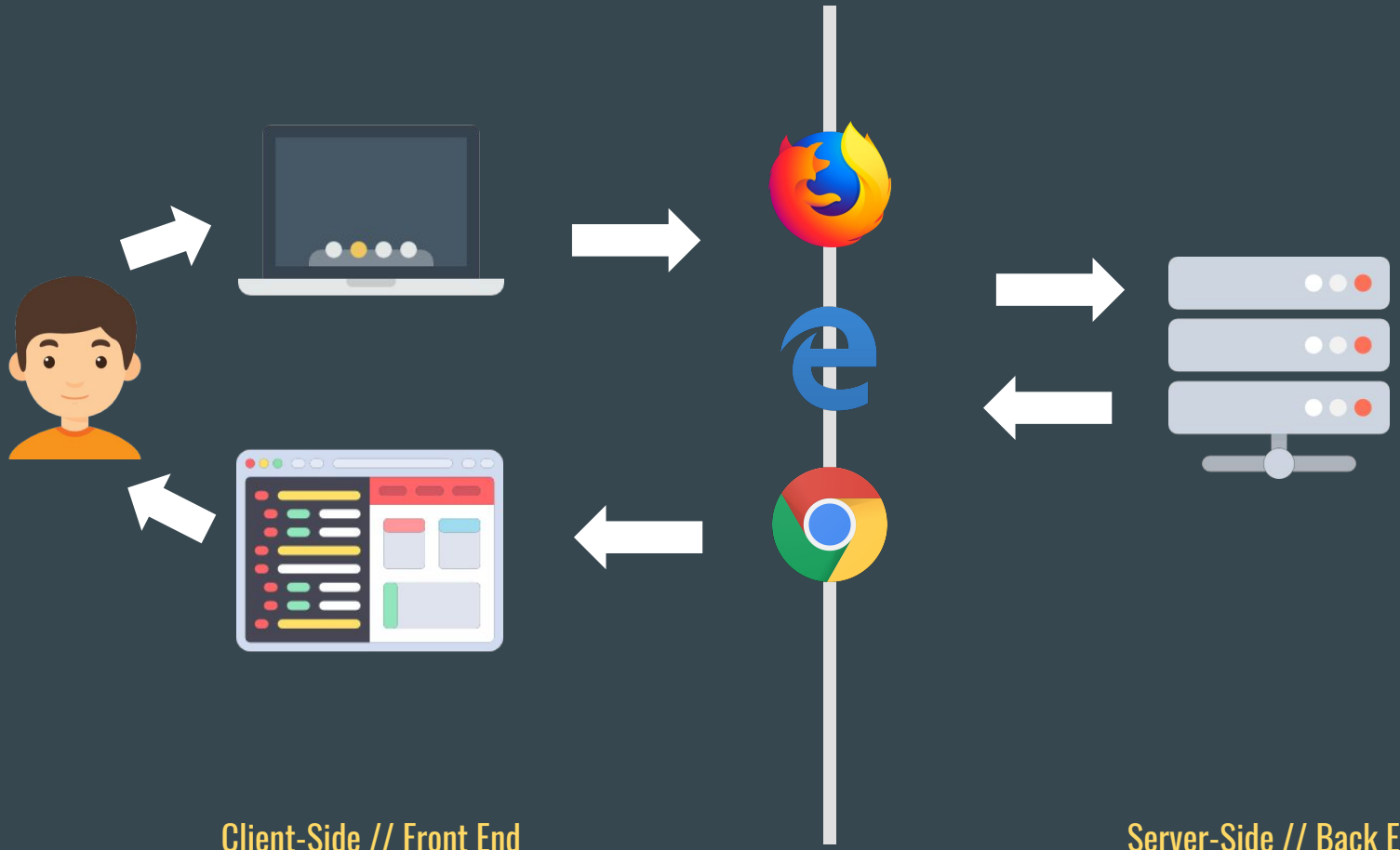
Langage pré-compilé

Langage interprété



Il est donc nécessaire de posséder un interpréteur pour faire fonctionner du code JavaScript

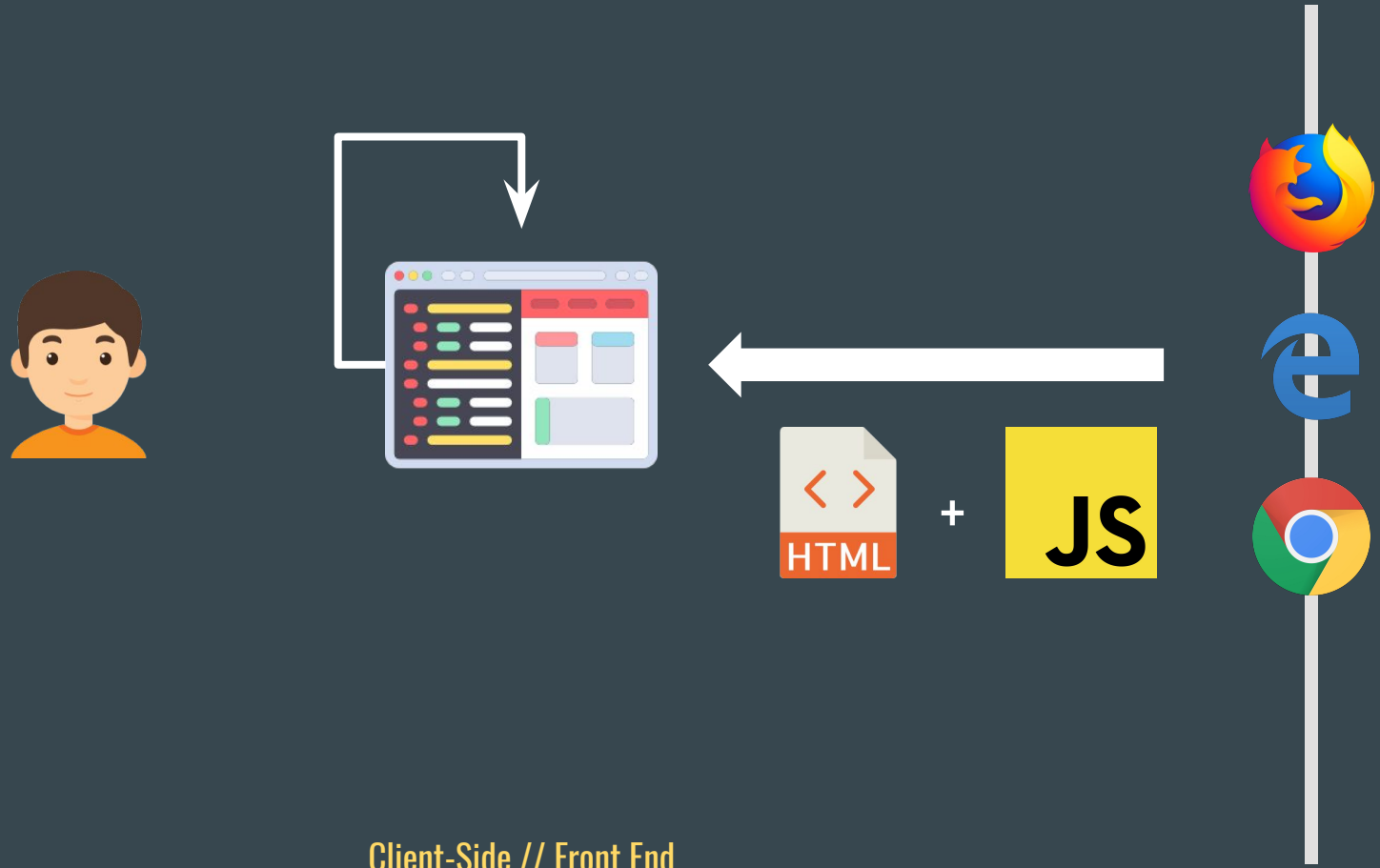
JavaScript est un langage utilisé “Client-side” et “Server-Side”



Client-Side // Front End

Server-Side // Back End

Client-Side // Front End



Client-Side // Front End

Javascript suit la norme ECMAScript ⇔ ES

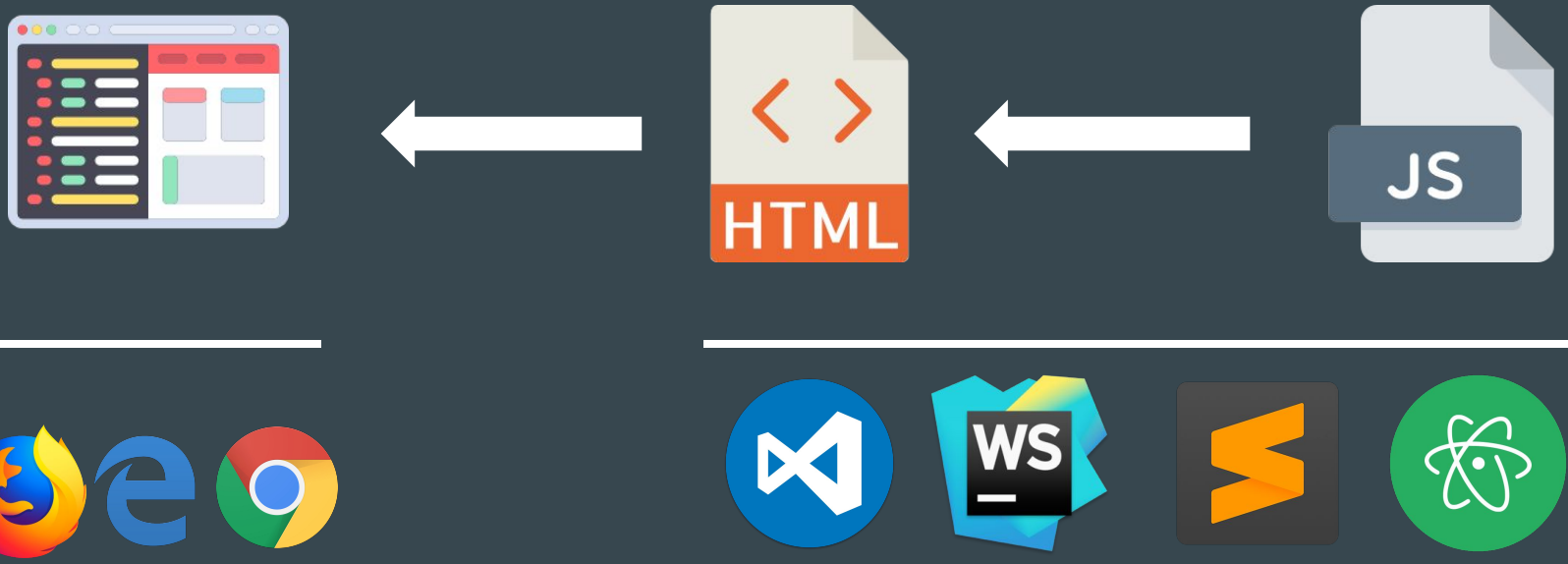
ES 5 : Sortie en 2009
ES 6 : Déploiement depuis 2014

JAVASCRIPT ES 5

SET-UP

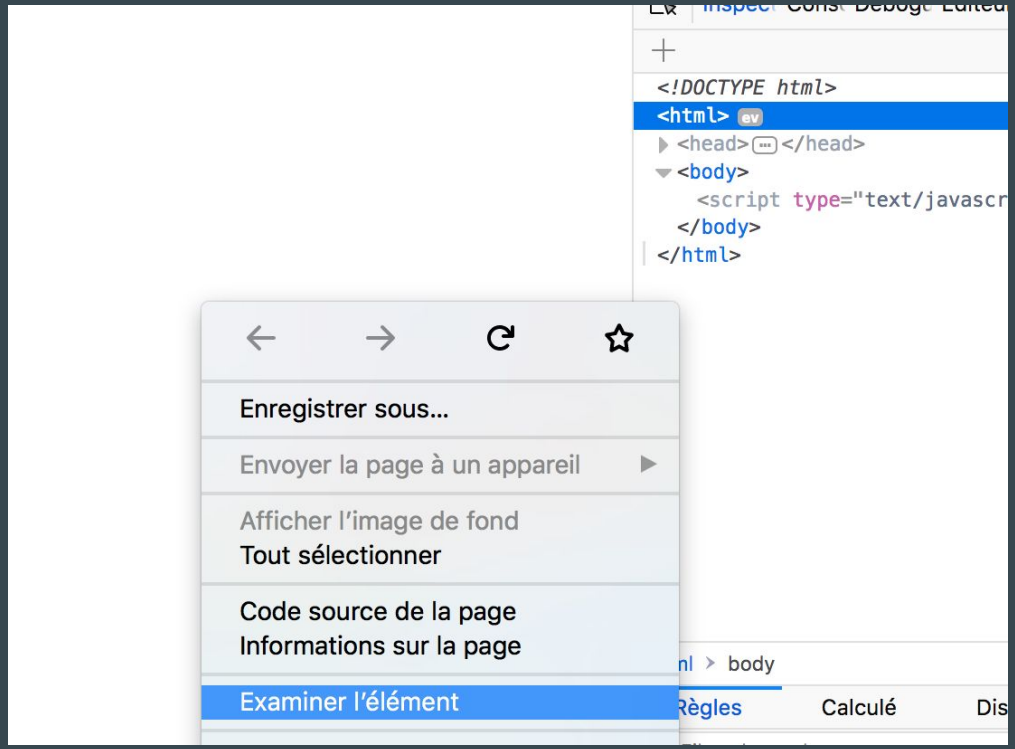
Structure

Structure




```
console.log("coucou");
```


```
<!doctype html>
<html>
 <head>
  <meta charset="utf-8">
  <title>JS ES5</title>
 </head>
 <body>
  <script type="text/javascript" src="script.js"></script>
 </body>
</html>
```


<!DOCTYPE html>
<html>
<head></head>
<body>
<script type="text/javascr
</body>
</html>
Enregistrer sous...
Envoyer la page à un appareil
Afficher l'image de fond
Tout sélectionner
Code source de la page
Informations sur la page
Examiner l'élément
body
Règles
Calculé

Inspecteur Console Débogueur Éditeur de style Performances

Filtrer Conserver les journaux

coucou script.js:1:1

Règles de syntaxe

- Instruction1;Instruction2;Instruction3;...
  - Si il y a un retour à la ligne, le “;” n’est pas obligatoire.

- Pas sensible aux espaces

- Commentaires
  - // Coucou !
  - /* Coucou */

Variables : déclaration

```
var ma_var = 1;
console.log(ma_var); // --> affiche 1
```

Variables

L'ordre est important !

```
console.log(ma_var); // --> affiche NaN
var ma_var = 1;
```

Variables

Types dynamiques

```
var a = 1;
console.log(a); // --> affiche 1
a = "bonjour";
console.log(a); // --> affiche bonjour

console.log(typeof a); // --> affiche le type : string, number, boolean, object ou undefined
```

Variables

Types dynamiques

```
var a = 1;
console.log(a); // --> affiche 1
a = "bonjour";
console.log(a); // --> affiche bonjour

console.log(typeof a); // --> affiche le type : string, number, boolean, object ou undefined
```

Variables

Manipulation

```
var result = "1";
var text = "Ma valeur = ";
console.log(text + result); // --> Ma valeur = 1

var result = 1;
var text = "Ma valeur = ";
console.log(text + result); // --> Ma valeur = 1

var nombre1 = "1";
var nombre2 = "2";
console.log(nombre1 + nombre2); // --> 12
console.log(parseInt(nombre1) + parseInt(nombre2)); // --> 3
```

NULL et UNDEFINED

Undefined

- Une variable pour laquelle aucune valeur n'a été assignée sera de type undefined.
- Une méthode ou instruction renvoie également undefined si la variable à évaluer n'a pas de valeur assignée.
- Une fonction renvoie undefined si aucune valeur n'a été retournée.

Null

- La valeur null est un littéral (et non pas une propriété de l'objet global telle que undefined)
- null est souvent utilisé en valeur de retour lorsqu'un objet est attendu mais qu'aucun objet ne convient

NULL et UNDEFINED

```
typeof null;          // "object"
typeof undefined;     // "undefined"
null === undefined;   // false
null  == undefined;   // true
null === null;        // true
null  == null;        // true
!null;                // true
```

Opérateurs

- Arithmétiques :
  - +
  - -
  - /
  - *
  - %
- Comparaisons :
  - < : inférieur
  - <= : inférieur ou égal
  - > : supérieur
  - >= : supérieur ou égal
  - == : égale en valeur
  - != : différente en valeur
  - === : égale en valeur et en type
  - !== : différente en valeur ou en type
- Logiques :
  - &&
  - ||

Conditions

```
var a,b;
a = prompt("A ?");
b = prompt("B ?");

if(a === b) {
 console.log("égalité de a et b");
}
```

Conditions

```
var a,b;
a = prompt("A ?");
b = prompt("B ?");

if(a === b) {
console.log("égalité forte de a et b");
} else if (a == b) {
 console.log("égalité de valeur de a et b");
} else {
console.log("a et b différents");
}
```

```
var drawer = parseInt(prompt('Choisissez le tiroir à ouvrir (1 à 4) :'));

switch (drawer) {
  case 1:
    alert('Contient divers outils pour dessiner : du papier, des crayons, etc.');
    break;

  case 2:
    alert('Contient du matériel informatique : des câbles, des composants, etc.');
    break;

  case 3:
  case 4:
    alert('Ah ? Ce tiroir est fermé à clé ! Dommage !');
    break;

  default:
    alert("Info du jour : le meuble ne contient que 4 tiroirs et, jusqu'à preuve du contraire, les tiroirs négatifs n'existent pas.");
}
```

Conditions

Ternaires

```
var age = parseInt(prompt("Votre âge ?"));
var category = (age >= 18) ? '18+' : '-18'; // (test) ? si vrai : si faux
alert(category);
```

Boucles

While

```
var i = 0;
while(i < 100) {
 i++;
 if( i === 13) {
   continue; // skip
 } else if(i === 42) {
   break; // quit
 }
 console.log(i);
 if(i % 2 === 0) {
   console.log("nombre paire");
 } else {
   console.log("nombre impaire");
 }
}
```

Boucles

For

```
for (var i = 0; i < 5; i++) {
 console.log('Itération n°' + i);
}

for (var i = 0, nameListe = '', name; true; i++) {
 name = prompt('Entrez un prénom :');
 if (name) {
   nameListe += name + ' ';
 } else {
     break;
 }
}
console.log('Il y a ' + i + ' prénoms :\n\n' + nameListe);
```

Fonctions

```
function sayHi() {
 console.log("coucou");
}

sayHi(); // OK
```

Fonctions

```
function add(a, b) {
 return a + b
}

console.log(add(1,3));
```

Fonctions

L’ordre n’a pas d’importance

```
sayHi(); // OK

function sayHi() {
 console.log("coucou");
}
```

Fonctions

Portée des variables

```
var a = 1;

function add(b) {
 return a + b;
}

console.log(add(2)); // --> affiche 3
```

Fonctions

Portée des variables

```
var a = 1;

function add(b) {
 var a = 0;
 return a + b;
}

console.log(add(2)); // --> affiche 2
```

Fonctions

Portée des variables

```
var message = 'globale !';

function showMsg() {
   var message = 'locale !';
   console.log("Dans la fonction : "+message);
}

showMsg();
console.log("A l'extérieur : "+message);
```

Fonctions

Arguments facultatifs

```
function optional(arg) {
 console.log(arg); // --> undefined
}

optional();
```

Fonctions

Arguments facultatifs

```
function optional(arg) {
 if(typeof arg === 'undefined')
   arg = "ras";
 console.log(arg); // --> ras
}

optional();
```

Fonctions

Fonctions anonymes

```
function() {
 alert('Bonjour !');
};
```

Fonctions

Fonctions anonymes

```
var sayHello = function() {
 alert('Bonjour !');
};

sayHello();
```

Fonctions

Fonctions anonymes, exemple d'usage : isolation du code

```
(function() {
  // Code isolé
})();
```

```
var test = 'noir'; // On crée une variable « test » contenant le mot « noir »

(function() { // Début de la zone isolée
  var test = 'blanc'; // On crée une variable du même nom avec le contenu « blanc » dans la zone isolée
  console.log('Dans la zone isolée, la couleur est : ' + test);
})(); // Fin de la zone isolée. Les variables créées dans cette zone sont détruites.

console.log('Dans la zone non-isolée, la couleur est : ' + test); // Le texte final contient bien le mot « noir »
```

Fonctions

Fonctions anonymes, exemple d'usage

```
var operation = function(type) {
 switch(type){
   case "add":
     return function(param1, param2) {
       return param1 + param2;
     }
     break;
   case "sub":
     return function(param1, param2) {
       return param1 - param2;
     }
     break;
 }
}
var result = operation("add")(1,2);
console.log(result);
```

Objets

JS est un langage objet par prototype

Un objet contient par défaut :

- Constructeur
- Propriétés
- Méthodes

```
var myString = 'Ceci est une chaîne de caractères';
alert(myString.length);
alert(myString.toUpperCase());
```

Objets

Objets natifs déjà rencontrés :

- number
- boolean
- string

A voir :

- Array
- Littéraux

Array

```
var myArray_a = [42, 12, 6, 3];
var myArray_b = [42, 'Sébastien', 12, 'Laurence'];
var myArray_c = new Array('Sébastien', 'Laurence', 'Pauline');

myArray_a[1] = 'Clarisse';
console.log(myArray_a[1]); // Affiche : « Clarisse »

myArray_c.push('Ludovic'); // Ajoute « Ludovic » à la fin du tableau
myArray_c.shift(); // Retire « Sébastien »
myArray_c.pop(); // Retire « Ludovic »

for (var i = 0 ; i < myArray_b.length ; i++) {
console.log(myArray_b[i]);
}
```

Littéraux

```
var family = {
 self: 'Sébastien',
 sister: 'Laurence',
 brother: 'Ludovic',
 cousin_1: 'Pauline',
 cousin_2: 'Guillaume'
};

console.log(family.self);
console.log(family['self']);
family.father = 'Pascal';

for(var id in family) {
 console.log(id+ " -- "+family[id]);
}
```

Objets : Constructeurs

```
function Person(nick, age, sex, parent, work, friends) {
 this.nick = nick;
 this.age = age;
 this.sex = sex;
 this.parent = parent;
 this.work = work;
 this.friends = friends;
 this.addFriend = function(friend) {
    this.friends.push(friend);
 };
}
var seb = new Person('Sébastien', 23, 'm', 'aîné', 'JavaScripteur', []);
var lau = new Person('Laurence', 19, 'f', 'soeur', 'Sous-officier', []);
seb.addFriend(lau);
```

Objets : prototype

```
function Person(nick, age, sex, parent, work, friends) {
 this.nick = nick;
 this.age = age;
 this.sex = sex;
 this.parent = parent;
 this.work = work;
 this.friends = friends;
}
Person.prototype.addFriend = function(friend) {
 this.friends.push(friend);
}
var seb = new Person('Sébastien', 23, 'm', 'aîné', 'JavaScripteur', []);
var lau = new Person('Laurence', 19, 'f', 'soeur', 'Sous-officier', []);

seb.addFriend(lau);
```

Interaction avec HTML

Interaction avec HTML

- DOM = Document Object Model :
  - Interface qui permet la manipulation du HTML depuis l'extérieur (exemple : en JavaScript)
  - HTML est standardisé suivant les spécifications DOM-2

- Objet window :
  - Objet global de JS, dans un navigateur web
  - Utilisation implicite

- Objet document :
  - Fils de window
  - Représente la balise <html> de la page

L’objet window

```
window.alert('Hello world!'); // alert est en réalité une méthode de window !
alert('Hello world!'); // window est implicitement utilisé par le navigateur

parseInt("10"); // parseInt n'a aucun liens avec window

var text = 'Variable globale !';
(function() {
   var text = 'Variable locale !';
   alert(text); // Forcément, la variable locale prend le dessus
   alert(window.text); // Mais il est toujours possible d'accéder à la variable globale grâce à l'objet « window »
})();

(function() {
 text = 'Variable accessible !'; // Cette variable n'a jamais été déclarée et pourtant on lui attribue une valeur
})();
alert(text); // Affiche : « Variable accessible ! »
```

L’objet document

```
var divs = document.getElementsByTagName('div'); // récupère tous les divs de la page
    formDivs = document.getElementsByName('surname'); // récupère les éléments de formulaire avec le nom correspondant
var div = document.getElementById('myDiv'); // récupère un élément par son id

console.log(div); // affiche [object HTMLDivElement]

var query = document.querySelector('#menu .item span'), // récupère le premier élément
    queryAll = document.querySelectorAll('#menu .item span'); // renvoie un tableau d'éléments
```

Modification du DOM

```
var link = document.getElementById('myLink');
var href = link.href;
link.href = 'http://www.google.fr';

document.getElementById('myColoredDiv').className = 'blue';

console.log(document.getElementById('myP').innerHtml());
console.log(document.getElementById('myP').innerText());
```

Les évènements

```
var element = document.getElementById('clickme');
element.addEventListener('click', function() {
  alert("Vous m'avez cliqué !");
});
element.addEventListener('click', function() {
  alert("Vous m'avez cliqué aussi !");
});

// type d'événement : click, mouseover, mouseout, input, focus, blur ...
```

Utile, mais pas pratique...


jQuery
write less, do more.

Jquery

Simplifier l'écriture de code JS pour manipuler le DOM

```
// Version JS
var listItems = document.querySelectorAll('li');
var i;
for (i = 0; i < listItems.length; i++) {
   listItems[i].className = 'starred';
}

// Version jQuery
$("li").addClass("starred");
```

Jquery

- Jquery permet de :
  - Ajouter/supprimer des classes
  - Éditer un/des attribut(s)
  - Modifier le contenu
  - Ajouter/supprimer des éléments du DOM
  - Animer : fade, slide etc.
  - Changer les propriétés css
  - Gérer les évènements
  - ...

Jquery

```
$(".close").on("click", function() {
 var target = $(this).attr("data-target");
 $("#"+target).slideUp();
 $("#"+target+"_trigger").css("z-index", 0);
 mySwiper.allowTouchMove = true;
 mySwiper.allowSlideNext = true;
 mySwiper.allowSlidePrev = true;
});
```

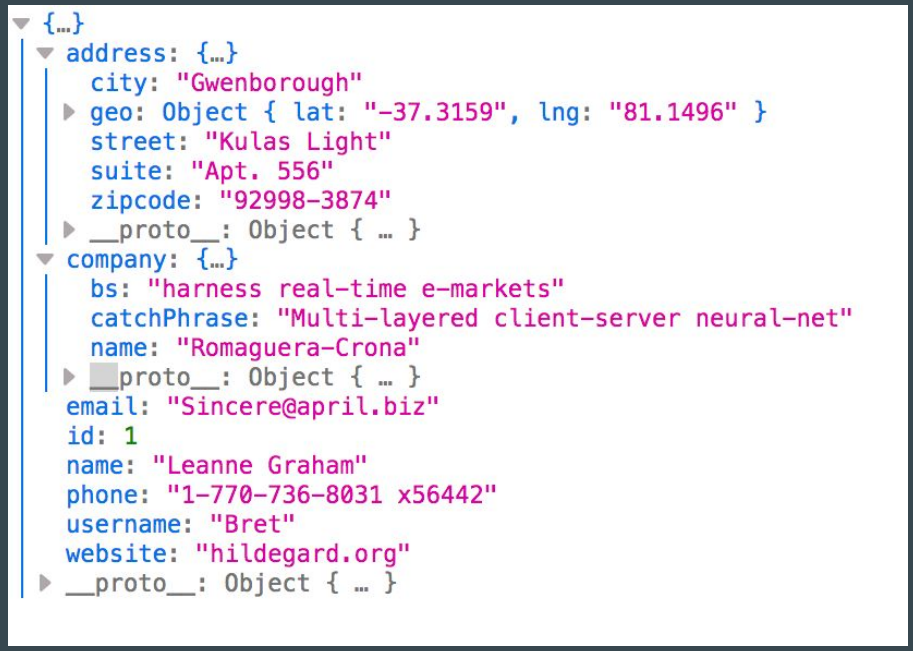
Jquery

Requêtes AJAX

```
$.ajax({
 url: "https://jsonplaceholder.typicode.com/users/1",
 success: function( result ) {
   console.log(result);
 }
});
```

Jquery

Requêtes AJAX



Jquery

Requêtes AJAX

```
$.ajax({
 url: "https://jsonplaceholder.typicode.com/users/1",
 success: function( result ) {
   console.log(result.username + " live in " +result.address.city);
 }
});
```

À vous de jouer !

Réalisez une page utilisant JS/Jquery

- L'objectif de cette page est de pouvoir afficher les profils utilisateur venant de l'API : https://jsonplaceholder.typicode.com/users/
  - exemple :
    - https://jsonplaceholder.typicode.com/users/2
    - https://jsonplaceholder.typicode.com/users/5

- L'utilisateur doit pouvoir :
  - Entrer l'id du profil recherché (entre 1 et 10)
  - Afficher le détails recherché
  - Passer à l'utilisateur précédent/suivant
  - Si l'id est incorrect, l'indiquer

Le tout, sans recharger la page !

JAVASCRIPT ES 6

Mais pourquoi ?

ES 5 : Héritage d'objet

```
function Car(licensePlate, tankSize, trunkSize) {
 // On appelle le constructeur de « Vehicle » par le biais de la méthode
 // call() afin qu'il affecte de nouvelles propriétés à « Car ».
 Vehicle.call(this, licensePlate, tankSize);
 // Une fois le constructeur parent appelé, l'initialisation de notre objet peut continuer.
 this.trunkOpened = false; // Notre coffre est-il ouvert ?
 this.trunkSize = trunkSize; // La taille de notre coffre en mètres cube.
}
// L'objet prototype de « Vehicle » doit être copié au sein du prototype
// de « Car » afin que ce dernier puisse bénéficier des mêmes méthodes.
Car.prototype = Object.create(Vehicle.prototype, {
 // Le prototype copié possède une référence vers son constructeur, actuellement
 // défini à « Vehicle », nous devons changer sa référence pour « Car »
 // tout en conservant sa particularité d'être une propriété non-énumerable.
 constructor: {
     value: Car,
     enumerable: false,
     writable: true,
     configurable: true
 }
});
```

ES 6 : Héritage d’objet

```
class Car extends Vehicle {
 // ...
}
```

C Q F D

SETUP

PROBLÈME

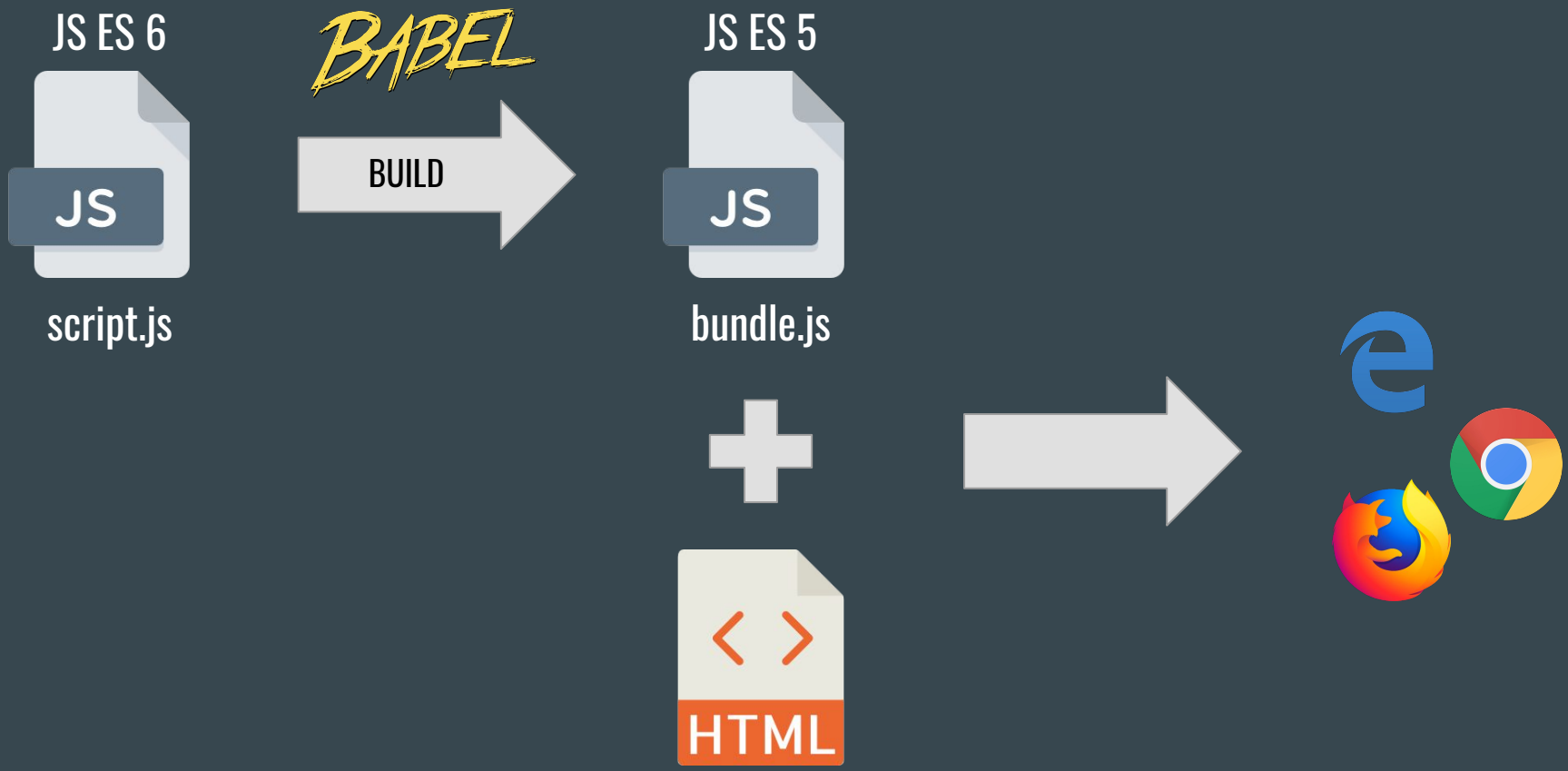
ES6 n’est pas forcément compatible avec les navigateurs

SOLUTION

Compiler le JS ES6 en JS ES5 !

INTRODUCING

BABEL


JS ES 6
JS
script.js
BABEL
BUILD
JS ES 5
JS
bundle.js
HTML

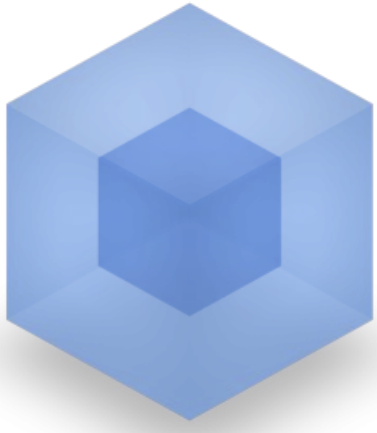
PROBLÈME

A chaque modification de script.js, bundle.js n’est plus à jour

SOLUTION

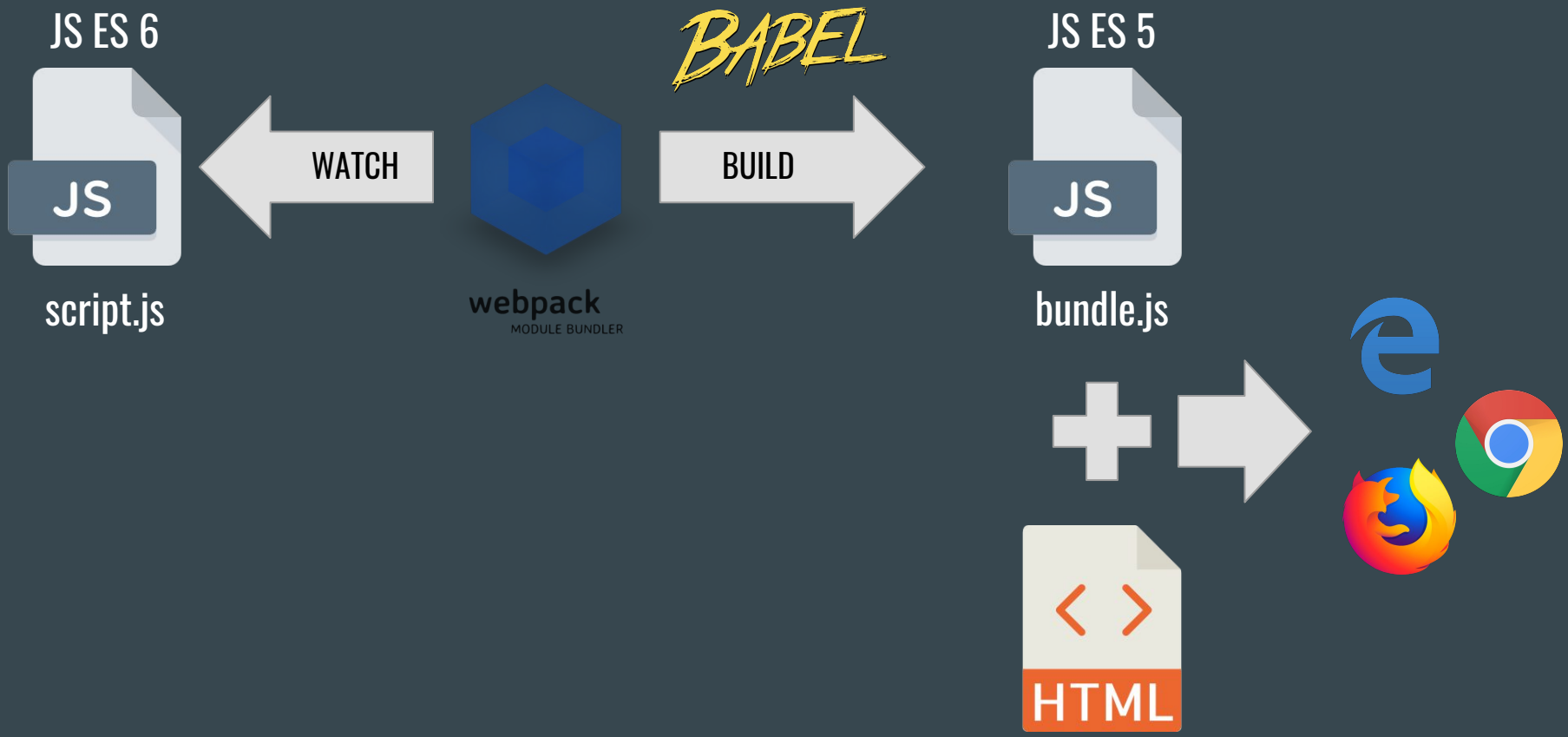
Recompiler à chaque modification

INTRODUCING



webpack

MODULE BUNDLER


JS ES 6
JS
script.js
WATCH
webpack
MODULE BUNDLER
BABEL
BUILD
JS ES 5
JS
bundle.js
HTML

Installation de Babel, Webpack et configuration

- Dans le terminal

```
npm init
npm install --save-dev webpack
npm install --save-dev babel-loader babel-core
```

- Créez un fichier webpack.config.js

```
const path = require('path');
module.exports = {
 entry: "./main/script.js", // string | object | array
  output: {
   path: path.resolve(__dirname, "dist"), // string
   filename: "bundle.js", // string
 },
 watch: true,
 module: {
   rules: [
     {
       test: /\.jsx?$/,
       include: [
         path.resolve(__dirname, "app")
       ],
       exclude: [
         path.resolve(__dirname, "app/demo-files")
       ],
       loader: "babel-loader",
       options: {
         presets: ["es2015"]
       },
     },
```

- Dans le terminal

```
webpack
```

```
Webpack is watching the files…

Hash: 03518bf94fb91f010d5c
Version: webpack 4.0.1
Time: 330ms
Built at: 2018-02-28 13:37:19
    Asset       Size  Chunks             Chunk Names
bundle.js  591 bytes       0  [emitted]  main
Entrypoint main = bundle.js
   [0] ./main/script.js 65 bytes {0} [built]

WARNING in configuration
The 'mode' option has not been set. Set 'mode' option to 'development' or 'production' to enable defaults for this environment.
```

Webpack surveil et compile automatiquement !

ES 6 : Variables

Variables

const

```
const maVar = 3;
maVar = 1; // PAS BON !
```

let

```
let autreVar = 1;
autrvar = 2; // OK
```

ES 6 : Arrow functions

Arrow Functions

Une expression de fonction fléchée (*arrow function* en anglais) permet d'avoir une syntaxe plus courte que les expressions de fonction et ne possède pas ses propres valeurs pour this, arguments, super, ou new.target. Les fonctions fléchées sont souvent anonymes et ne sont pas destinées à être utilisées pour déclarer des méthodes.

```
const add = (a, b) => {
 return a + b;
}
console.log(add(1,2));

const execute = (f) => {
 f();
}
execute(() => {console.log("coucou")});
```

ES 6 : Array helpers

Array Helpers

forEach
Parcours l'objet

```
const arr = [1,2,3,4]
arr.forEach(item => console.log(item))
```

Array Helpers

map

Parcours l'objet en passant valeur et clé ;

```
const peoples = [{name:"James",age:18}, {name:"Alice",age:20}]

peoples.map((people, key) => {
 console.log(people.name);
})

peoples.map((people, key) => {
 return people.name;  // Le résultat renvoyé par map sera un tableau de nom
})
```

Array Helpers

filter

La méthode passée en paramètre est appelée pour chaque objet. Si elle renvoie true, l'objet est conservé ; renvoie un tableau

```
const fruits = [
 {name:"banana",color:"yellow"},
 {name:"watermelon",color:"red"},
 {name:"orange",color:"orange"}
]
const filteredFruits = fruits.filter(fruit => fruit.color === 'yellow');
console.log(filteredFruits)
```

Array Helpers

find

Renvoie le premier élément qui satisfait la méthode passée en paramètre ; renvoie un objet

```
let x = [
 {name:"jay",age:1},
 {name:"jay",age:2},
 {name:"hulu",age:30}
]
console.log(x.find(obj => obj.name === "jay"));
```

Array Helpers

some

Renvoie true si au moins une des évaluation de la méthode passée en paramètre a renvoyé true

```
let x = [
 {name:"jay",age:1},
 {name:"jay",age:2},
 {name:"hulu",age:30}
]
console.log(x.some(obj => obj.age > 2));
```

Array Helpers

every

Renvoie true si toutes les évaluations de la méthode passée en paramètre ont renvoyé true

```
let x = [
 {name:"jay",age:1},
 {name:"jay",age:2},
 {name:"hulu",age:30}
]
console.log(x.every(obj => obj.age > 2));
```

Array Helpers

reduce

Le plus complex / puissant des helpers.
Il prend en premier paramètre la fonction d'itération, en second la valeur initial.
La fonction d'itération prend en paramètre la valeur précédente et l'objet courant

```
function addNumbers(numbers) {
 return numbers.reduce((sum, number) => {
   return sum+number
 }, 0)
}
```

Array Helpers

reduce

Le plus complex / puissant des helpers.
Il prend en premier paramètre la fonction d'itération, en second la valeur initial.
La fonction d'itération prend en paramètre la valeur précédente et l'objet courant

```
var desks = [
 { type: "sitting" },
 { type: "standing" },
 { type: "sitting" },
 { type: "sitting" },
 { type: "standing" }
];
// Compter le nombre de bureaux
counted = desks.reduce((prev, desk) => {
 prev[desk.type]++;
 return prev;
}, { sitting: 0, standing: 0 });
console.log(counted);
```

ES 6 : Enhanced Object Literal

Enhanced Object Literal

Optimise l'écriture du code

```
const user = {
 surname : "jean",
 name : "bon"
}

const profile = {
 user : user,
 getfullname : function() {
   return this.user.surname + " " + this.user.name;
 }
}

console.log(profile.getfullname());
```

Enhanced Object Literal

Optimise l'écriture du code

```
const user = {
 surname : "jean",
 name : "bon"
}

const profile = {
 user,
 getfullname() {
   return this.user.surname + " " + this.user.name;
 }
}

console.log(profile.getfullname());
```

ES 6 : Arguments par défaut

Arguments par défaut

```
function call(url, methode = "GET") {
 const result = "";
 // ... //
 return result;
}

call("http://google.fr", "POST");
call("http://google.fr", "GET");
call("http://google.fr");
```

ES 6 : Rest and Spread operator

Rest and Spread

Rest

En JS ES 5, il est possible qu'un fonction n'ai pas d'arguments fixes

```
function addNumbers() {
 var numbers = Array.prototype.slice.call(arguments, 0);
 return numbers.reduce((sum, number) => {
   return sum+number
 }, 0)
}
console.log(addNumbers(1,2,3,4,5,6));
console.log(addNumbers(1,2,3,4,5,6,7,8,9,10));
```

Rest and Spread

Rest

En JS ES 6, c'est un peu plus pratique.

```
function addNumbers(...numbers) {
 return numbers.reduce((sum, number) => {
   return sum+number
 }, 0)
}

console.log(addNumbers(1,2,3,4,5,6));
console.log(addNumbers(1,2,3,4,5,6,7,8,9,10));
```

Rest and Spread

Spread

```
const colors1 = ["blue", "red"];
const colors2 = ["white", "green"];

const colors = [...colors1, ...colors2];
const colorsOld = [colors1, colors2];
console.log(colors);
console.log(colorsOld);
```

Rest and Spread

Rest & Spread

```
function validateShoping(...list) {
 if(list.indexOf("banane") < 0) {
   return [...list, "banane"];
 }
}

const shopingList = validateShoping("orange","café");

console.log(shopingList);
```

ES 6 : Destructuring

Destructuring

Sur les Object

```
const objetComplex = {
 token : "&1289È!1SIOAJIO;kds",
 url : "http://www.facebook.com",
 update : 3000
}

// AVANT
const token = objetComplex.token;
const url = objetComplex.url;
const update = objetComplex.update;

// APRES
const { token, url, update } = objetComplex;
```

Destructuring

Sur les Array

```
const socials = [
 'Google',
 'Facebook',
 'Twitter',
];

// AVANT
const social1 = socials[0];
const social2 = socials[1];
const social3 = socials[2];

// APRES
const [social1, social2, social3] = socials;
```

ES 6 : Classes

Classes

```
class Car {
 constructor(name, year) { // exécuté automatiquement lors d'un new
   this.name = name;
   this.year = year;
 }

 drive() {
   console.log("vroom");
 }
}

const myCar = new Car("toyota", 1992);
console.log(myCar);
myCar.drive();
```

```
class Car {
 constructor(name, year) { // exécuté automatiquement lors d'un new
   this.name = name;
   this.year = year;
   this.driving = false;
 }

 drive() {
   if(this.driving)
     console.log("vroom");
 }

 start() {
   this.driving = true;
 }
}
```

```
class Toyota extends Car {
 constructor(model, year) {
   super("Toyota", year);
   this.model = model;
 }

 start() {
   super.start();
   console.log("開始");
 }
}

const myToyota = new Toyota("Prius", 2015);
console.log(myToyota);
myToyota.drive();
myToyota.start();
myToyota.drive();
```

À vous de jouer !

Manipuler JS ES 6

- Ecrivez la classe Point qui représente un point en 2 dimensions.
  - Elle comprend 2 attributs : x et y
  - Elle comprend une méthode : plus(otherP) qui permet de l'additionner à un autre Point. Le résultat est un nouveau Point.

```
// TODO
// ...

console.log(new Point(1, 2).plus(new Point(2, 1)))
// → Point{x: 3, y: 3}
```

Manipuler JS ES 6

- En utilisant FILTER et REDUCE, compléter la ligne (calculez la valeur total des “machine”).

```
const inventory = [
 {type:   "machine", value: 5000},
 {type:   "machine", value:  650},
 {type:      "duck", value:   10},
 {type: "furniture", value: 1200},
 {type:   "machine", value:   77}
]

let totalMachineValue = //TODO ;

console.log(totalMachineValue)
```

Manipuler JS ES 6

- Utilisant REDUCE, ecrivez une fonction qui valide les parenthèses d'une chaîne de caractère

```
const test1 = "()()()()" // true
const test2 = "() (()()" // false
const test3 = "((((" // false
const test4 = ")()" // false
const test5 = ")()(" // false

const validate = (str) => {
    // todo
}

console.log(validate(test1));
console.log(validate(test2));
console.log(validate(test3));
console.log(validate(test4));
console.log(validate(test5));
```

Manipuler JS ES 6

- Utilisant REDUCE, ré-ecrivez les fonctions MAP, FILTER, FIND, SOME & EVERY