

JavaScript

Rappel Ecueils Notions avancées

Internet : Javascript

Objectif

JavaScript / Json / Ajax

Sommaire

1. Définition
2. Bases
3. Divers
4. Préparation NodeJS

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1a. Le Javascript c'est quoi ?

JavaScript (souvent abrégé JS) est un langage de programmation de scripts principalement utilisé dans les pages web interactives mais aussi côté serveur.

C'est un langage orienté objet à prototype, c'est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets qui ne sont pas des instances de classes, mais qui sont chacun équipés de constructeurs permettant de créer leurs propriétés, et notamment une propriété de prototypage qui permet d'en créer des objets héritiers personnalisés.

<http://fr.wikipedia.org/wiki/JavaScript>

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b. Les bases du langage

Bob est un homme (= objet)	objet
Bob est né le 28/11/1973	propriétés
Bob peut manger, lire et calculer son âge	méthodes
Bob est une instance de classe "développeur"	classes (OOP classique)
Bob est basé sur un autre objet appelé "développeur"	prototype (OOP prototype)
Bob garde ses informations et les méthodes qui vont avec ses informations	encapsulation
Bob a ses méthodes privées	privé / publique
Bob/dev travaille avec Jill/graph et Jack/PM	aggrégation, composition
Dev/graph/PM basés sur l'objet personne	héritage
Bob:talk – Jill:talk – Jack:talk	Polymorphisme/surcharge

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-i. Les primitives

Par coeur :

- 1 - Number
- 2 - String
- 3 - Boolean
- 4 - undefined
- 5 - null (\neq undefined)
- 6 - NaN
- 7 - Infinity
- 8 - function
- 9 - object

Déclarer une variable

```
var a;
```

Initialiser une variable

```
var a=1;
```

Les variables sont sensibles à la casse

Console : essayer :

```
var aa = 'test';
```

```
var AA = 'autre';
```

```
aa
```

```
AA
```

```
typeof(AA)
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-ii. Les blocs de code

Bloc simple

```
{  
    var a = 1;  
    var b = 3;  
}
```

Blocs dans des blocs

```
{  
    var a = 1;  
    var b = 3;  
    {  
        var c = 1;  
        var d = 3;  
    }  
}
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-ii. La syntaxe

Déclarer une variable

```
var tutu = 12; var t, u, v =15, yy="o";
```

Sensibles à la casse.

```
Tutu != tutu
```

Déclarer une fonction

```
function test() {  
}
```

Fonction avec des paramètres

La portée est par fonction et non pas par...
bloc.

JavaScript / Json / AJAX

1 - JavaScript - Définition + bases

1b-ii. La syntaxe

Erreurs classiques :

```
function test() {  
    return  
        (1 + 2 + 3 + 4 + 5 + 6);  
}  
console.log(test());
```

... ou encore, que donne :

```
3 + "30"  
3 * "30"
```


Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-iii. Les opérateurs de calcul

1 + 2	3
99.99 - 1	98.99
2 * 3	6
6 / 4	1.5
6 % 3	0
255 ^ 128	127
var a=12; a++; a	13
var b=a; b--; b	12

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-iv. Les opérateurs bit à bit

&	Bitwise AND	0b1010 & 0b1100	// 0b1000
	Bitwise OR	0b1010 0b1100	// 0b1110
^	Bitwise XOR	0b1010 ^ 0b1100	// 0b0110
~	Bitwise NOT	~0b1010	// 0b0101
<<	Left shift	0b1010 << 1	// 0b10100
		0b1010 << 2	// 0b101000
>>	Sign-propagating right shift		
>>>	Zero-fill right shift		

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-iv. Les opérateurs bit à bit

```
const FLAG_READ      1           // 0b001
const FLAG_WRITE     2           // 0b010
const FLAG_EXECUTE   4           // 0b100
let p = FLAG_READ | FLAG_WRITE;   // 0b011
let hasWrite = p & FLAG_WRITE;    // 0b010 - truthy
let hasExecute = p & FLAG_EXECUTE; // 0b000 - falsy
p = p ^ FLAG_WRITE;              // 0b001 - inversé (off)
p = p ^ FLAG_WRITE;              // 0b011 - inversé (on)
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-v. Les opérateurs logiques

NOT logique	!xx	
ET logique	&&	
OU logique		
var b = !true; b		false
var b = !!true; b		true
var b = "one"; !b		false
var b = "one"; !!b		true
Chaine vide : var u1=''; !u1		
!null		false
!undefined		false
Nombre !0		false
Nombre !NaN		false
Booléen !false		false

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-vi. Les opérateurs de comparaison

==	Vrai si égalité
===	Vrai si égalité + type
!=	Vrai si non-égalité
!==	Vrai si non-égalité OU types différents
>	Vrai si gauche supérieur à droite
>=	Vrai si gauche supérieur ou égal à droite
<	Vrai si gauche inférieur à droite
<=	Vrai si gauche inférieur ou égal à droite

Tester :

```
1==1
1=== '1'
1!= '2'
1!== '2'
1 > 1
1 <= 1
1 < 1
1 <= 1
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-vii-a. Les structures conditionnelles

Conditions "if"

```
if (a>3) {  
    result = 'ok';  
}
```

Conditions "if-else"

```
if (a>3) {  
    result = 'ok';  
} else {  
    result = 'erreur';  
}
```

Conditions "if-else-if"

```
if (note>15) {  
    result = 'bon';  
} else if (note>10) {  
    result = 'moyen';  
} else {  
    result = 'mauvais';  
}
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-vii-b. Les structures conditionnelles

"if" imbriqués

```
if (a>10) {  
    if (a<15) {  
        result = 'moyen';  
    } else {  
        result = 'bon';  
    }  
} else {  
    result = 'mauvais';  
}
```

Opérateur ternaire

```
var result = (a==1 ? 0 : 1);
```

A utiliser avec parcimonie

```
switch (note) {  
    case 0:  
        result = 'exclu';  
        break;  
    case 1:  
    case 2:  
    case 3:  
        result = 'bidon';  
        break;  
    case 19:  
    case 20:  
        result = 'normal';  
        break;  
    default:  
        result = 'à refaire';  
        break;  
}
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-vii-b. Les structures conditionnelles

Opérateur ternaire

```
var result = (a==1 ? 0 : 1);
```

A utiliser avec parcimonie...

Exemple Php, Wordpress 4.2.3

`wp-admin/includes/class-wp-comments-list-table.php`

```
( ( ( 'approve' == $action || 'unapprove' == $action ) && 2 === $i ) || 1 === $i ) ? $sep = '' : $sep = ' | ';
```

```
(  
    ( 'approve' == $action  
      || 'unapprove' == $action  
    ) && 2 === $i  
  ) || 1 === $i  
) ? $sep = '' : $sep = ' | ';
```


Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-viii. Les boucles

```
var i = 0;
do {
  i++;
} while (i<10);
```

```
var i=0;
while (i<10) {
  i++;
}
```

```
var p='';
for (var i = 0; i<100; i++) {
  p += 'test';
}
```

```
for (var i = 0, p='';
      i<100;
      i++, p += 'test;') {
  /* rien */
}
```

```
var i = 0, p='';
for (;;) {
  if (++i== 100) { break; }
  p += 'test';
}
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-ix.a. Les variables et leurs types

<code>typeof(undefined)</code>	<code>"undefined"</code>
<code>typeof(null)</code>	<code>"object"</code>
<code>typeof({})</code>	<code>"object"</code>
<code>typeof(true)</code>	<code>"boolean"</code>
<code>typeof(1)</code>	<code>"number"</code>
<code>typeof("")</code>	<code>"string"</code>
<code>typeof(Symbol())</code> ← ES6	<code>"symbol"</code>
<code>typeof(function() {})</code>	<code>"function"</code>

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-ix.b. Les variables et leurs portées

La portée des variables n'est **pas par blocs**.
La portée des variables est **par fonction**.

```
var global=1;  
function f() {  
    var local = 2;  
    global++;  
    return global;  
}
```

f(); donnera quoi ?
Encore f(); donnera quoi ?
local donnera quoi ?

```
var a = 123;  
function f() {  
    console.log(a);  
    var a = 1;  
    console.log(a);  
}
```

f(); donnera quoi ?

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1b-ix.b. Les variables et leurs portées

Votre code :

```
if(x !== 3) {  
    console.log(y);  
    var y = 5;  
    if (y === 5) {  
        var x = 3;  
    }  
    console.log(y);  
}  
if (x === 3) {  
    console.log(y);  
}
```

Comment JavaScript l'interprète :

```
var x;  
var y;  
if(x !== 3) {  
    console.log(y);  
    y = 5;  
    if (y === 5) {  
        x = 3;  
    }  
    console.log(y);  
}  
if (x === 3) {  
    console.log(y);  
}
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1c-x. Les chaines - Syntaxe

```
var maChaine="des caracteres";  
var maChaine='des caracteres';
```

\ Caractère d'échappement

\\ Écrire un \

\n Écrire un retour chariot

\t Écrire une tabulation

\u Écrire un caractère unicode

```
var maChaine="\ntest\n123\n456";  
console.log(maChaine);  
donne quoi ?
```

```
var maChaine="\ttest\t123\t456";  
console.log(maChaine);  
donne quoi ?
```

```
var t='I \u2661 JavaScript!';  
console.log(t);  
donne quoi ?
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1c-xi. Les chaines - Conversion

```
var s1 = "un";  
var s2 = "deux";  
var s = s1 + s2;  
s; donnera quoi ?  
typeof s; donnera quoi ?
```

```
var s1 = "un";  
var s2 = "deux";  
var s = s1 * s2;  
s; donnera quoi ?  
typeof s; donnera quoi ?
```

```
var s = "un";  
S = 3 * s  
s; donnera quoi ?  
typeof s; donnera quoi ?
```

```
var s = "3";  
s = s * 3;  
s; donnera quoi ?  
typeof s; donnera quoi ?
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1c-xii. Les tableaux

```
var a=[];                var a=new Array();  
a; donnera quoi ?  
typeof a; donnera quoi ?
```

```
var a = ["12", 1, 2, 3, 5, "un"];  
a; donnera quoi ?  
a[3] = [5];  
a; donnera quoi ?
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1c-xiii. Les tableaux — Opérations / fonctions

```
var a=[1,89,9];
```

Mettre à jour un élément

```
a[1] = 12;
```

a; donnera quoi ?

Ajouter un élément

```
a[10] = 0.25;
```

a; donnera quoi ?

Supprimer un élément

```
delete a[7];
```

a; donnera quoi ?

A connaître par coeur :

```
a.push();  
a.pop();  
a.slice();  
a.splice();  
a.sort();  
a.join();  
a.length
```


Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1c-xiii. Les tableaux - Boucles

Différence entre ces deux boucles ?

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

```
for (let i in a) {  
    console.log(a[i]);  
}
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1c-xv. Les objets - Déclaration

Tapez ce qui suit :

```
var o = {  
  A: 12,  
  B: "test"  
};  
o  
typeof(o);
```

Devinez les réponses.

Tapez ce qui suit :

```
var o = {  
  A: 12,  
  B: "test",  
  a: 3.25,  
  b: "autre"  
};  
o  
typeof(o);
```

Devinez les réponses.

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1C-xvi. Les objets - Syntaxe

Un objet dans un objet :

```
let livre = {  
  titre: "Le chien des Baskerville",  
  publication: 1902,  
  auteur: {  
    civilite: "Sir",  
    nom: "Doyle",  
    prenom: "Arthur Conan"  
  }  
};
```

Donnez quatre façons de sortir avec `console.log()` :
Le chien des Baskerville (Sir Artur Conan Doyle)

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1C-xvi. Les objets - Boucles

```
let o = {  
  A: "test",  
  B: 2.565,  
  a: "autre",  
  b: "valeur"  
};
```

Terminez le code pour afficher

toutes les valeurs des propriétés :

```
for (var i in o) {  
  console.log("/ * Terminer le code ici */");  
}
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1c-xvii. Les objets - Prédéfinis

Object
Array
Number
Boolean
String
Date
RegExp

(!) Les bases JavaScript sont ici...
La documentation
exhaustive est impossible
dans le cadre d'une initiation.

Objet particulier qui n'accepte pas "new" :
Math

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1C-xviii. Les fonctions — Déclarations

```
var fn = function () {  
    console.log('bonjour');  
};  
  
var o={  
    a: fn  
};
```

Comment appeler `fn` qui est dans l'objet `o` ?
Citez les deux possibilités

1C-xix. Les fonctions — Déclarations

Une fonction est une donnée

```
function fn() {  
    console.log('bonjour');  
};  
  
var fn = function () {  
    console.log('bonjour')  
};  
  
var tab=[1, 2, fn, "aa"];
```

Comment appeler fn qui est dans le tableau tab ?

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1d-i. Fonctions - Bases

En général :

```
function f() { return 1; }
```

En JavaScript :

```
var f = function() { return 1; }
```

typeof f donne quoi ?

```
var sum = function(a,b) { return a+b; }
```

```
var add = sum;
```

delete sum donne quoi ?

typeof add donne quoi ?

add(1,2) donne quoi ?

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1d-ii. Fonctions – Anonymes

```
function invoque_et_add(a,b) {  
    return a() + b();  
}  
var un = function () { return 1; }  
var deux = function () { return 2; }  
>> invoque_et_add(un,deux); donne quoi ?  
  
>> invoque_et_add(  
    function () { return 1; },  
    function () { return 2; }  
);  
Est il possible, si oui, donne quoi ?
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

1e. Fonctions – Prédéfinies

<code>parseInt();</code>	<code>=> '123', 'a123', '1a23', '123a'</code>
<code>parseFloat();</code>	<code>=> '1.23', 'a1.23', '1a.23', '1.a23', '1.23a', '123e-2', '1e10'</code>
<code>isNaN();</code>	<code>=> NaN, 123, 1.23</code>
<code>isFinite();</code>	<code>=> NaN, Infinity, 1e308, 1e309</code>
<code>encodeURIComponent();</code>	<code>=> '?test=12&val=80'</code>
<code>decodeURIComponent();</code>	<code>=> 'je%20-test'</code>
<code>eval();</code>	<code>=> 'var a=12;'</code>
<code>alert();</code>	<code>=> 'bonjour'</code>
<code>console.log();</code>	<code>=> eval('var a=12;'); console.log(a);</code>

1f-i. Fonctions – Self-invoking

```
(  
    function() {  
        alert('coucou');  
    }  
) ();  
  
(  
    function(name) {  
        alert(name);  
    }  
) ('ceci est un test');
```

1f-i. Fonctions – Self-invoking

Scoping : quelles seront les sorties consoles ?

```
var test = 123;
if (true) {
    (function () {
        var test = 456;
        console.log(test);
    })();
    console.log(test);
}
console.log(test);
```

1f-ii. Fonctions – Privées

```
var a= function (param) {  
    var b=function (theinput) {  
        return theinput*2;  
    };  
    return 'Résultat ' + b(param);  
};  
  
a(2);  
a(48);  
b(test);
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

2a-i. POO – Bases

```
function Identite(a_nom, a_prenom) {  
    this.nom = a_nom;  
    this.prenom = a_prenom;  
}  
  
var oo = new Identite('pons', 'olivier');  
console.log(oo.nomCompleter());
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

2a-ii. POO – Bases

```
String.prototype.maFonction = function () {  
    return ('Ma longueur est : '+this.length);  
};  
  
var oo = "Test";  
console.log(oo.maFonction());
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

2a-iii. POO – Bases

Expliquez ce qui suit :

```
function Identite(nom, prenom) {  
    this.nom      = nom;  
    this.prenom   = prenom;  
}  
Identite.prototype.nomComplet = function () {  
    return this.nom+' '+this.prenom;  
};  
Identite.prototype.valeur = 'test';  
var oo = new Identite('a', 'c');  
oo.valeur='autre valeur';  
var op = new Identite('b', 'd');  
console.log(oo.valeur);
```


Javascript / Json / AJAX

1 - JavaScript - Définition + bases

2a-iv. POO – Bases

Déterminer un type : `instanceof`

```
function Identite(nom, prenom) {  
    this.nom      = nom;  
    this.prenom   = prenom;  
}
```

```
var h=new Identite('pons', 'olivier');
```

Que donne :

```
h instanceof Identite  
h instanceof Object
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

2a-vi. POO – Bases

Fonction cachée : le constructeur : constructor

```
function Identite(nom, prenom) {  
    this.nom      = nom;  
    this.prenom   = prenom;  
}
```

```
var h=new Identite('pons', 'olivier');
```

Que donne :
h.constructor
typeof(h.constructor)

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

2a-vi. POO – Bases

Créer des objets sans "new" :

```
function sansNew(a, b) {  
    return {  
        nom: a;  
        prenom: b;  
    };  
}  
  
var h=sansNew('pons', 'olivier');
```

Que donne :
h.nom
h.constructor

2b. POO – Objet global - Définition

Dans les navigateurs,

Deux objets globaux à connaître :

- `window`
- `document`

2c. POO – Exceptions

Essayez :

```
try {  
    fonctionQuiNExistePas();  
} catch(e) {  
    console.log(e.name + ' - ' + e.message);  
} finally {  
    console.log('Finally !');  
}
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3a-i. Notions avancées

Object.defineProperty

Firefox 4+, Chrome 5+, Safari 5.1+, Opera 11.6+, IE 9+

```
var obj = {};  
Object.defineProperty(obj, 'maprop', {  
  get: function() {  
    return maprop * 2;  
  },  
  set: function(val) {  
    alert('Nouvelle valeur : '+val);  
    maprop = val;  
  }  
});  
obj.maprop = 20;  
alert(obj.maprop);
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3a-i. Notions avancées

Object.defineProperty

Firefox 4+, Chrome 5+, Safari 5.1+, Opera 11.6+, IE 9+

```
var obj = {};  
obj.laVie = 42;  
Object.defineProperty(obj, 'laVie', {  
    writable: false,      // Lecture seule  
    configurable: false  // Suppression impossible  
});  
obj.laVie = 5;  
alert(obj.laVie);
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3a-ii. Notions avancées

Mot-clé arguments

```
function printf(text) {  
    let i = 0;  
    let args = Array.prototype.slice.call(arguments);  
    text = text.replace(/\%s/g, function(a) {  
        return args[++i];  
    });  
    return text;  
}  
printf(  
    "Bonjour %s! Tu as %s points",  
    "Olivier",  
    1337  
);
```


Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3a-iii. Notions avancées

Méthode call()

```
var brice = { nom: "Brice" };  
var adeline = { nom: "Adeline" };  
function hello() {  
    return 'Hello, je suis ${this.nom} !';  
}  
var a = hello();  
var b = hello.call(brice);  
var c = hello.call(adeline);  
[a, b, c]
```

call sert à spécifier this

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3a-iii. Notions avancées

Méthode call()

```
function update(aaaa, job) {  
    this.annee_naissance = aaaa;  
    this.job = job;  
}  
update.call(brice, 1975, 'danseur');  
update.call(adeline, 1989, 'actrice');  
[brice, adeline]
```

call sert à spécifier this

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3a-iv. Notions avancées

Méthode `apply()`

`apply` est pareil que `call` mais les arguments sont un tableau

```
function update(aaaa, job) {  
    this.annee_naissance = aaaa;  
    this.job = job;  
}  
update.apply(brice, [1975, 'danseur']);  
update.apply(madeline, [1989, 'actrice']);  
[brice, adeline]
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3a-iii. Notions avancées

Méthode `bind()`

`bind()` associe de manière permanente permanente `this`

```
var updateBrice = update.bind(brice);  
updateBrice(1904, "acteur");  
updateBrice.call(adeline, 1174, "reine");  
updateBrice.call(undefined, 10, "roi");
```

Dites ce que contient : `brice`

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3a-iii. Notions avancées

Méthode bind()

```
var TestObj = function() {  
    this.someValue = 100;  
    this.maFc = function() {  
        alert(this.someValue);  
    };  
    setTimeout(this.maFc, 1000);  
    setTimeout(this.maFc.bind(this), 2000);  
}  
new TestObj();
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3a-iii. Notions avancées

ES6 : plus besoin de méthode `bind()`

```
var TestObj = function() {  
    this.location = 100;  
    this.maFc = () => {  
        alert(this.location);  
    };  
    setTimeout(this.maFc, 1000);  
}  
new TestObj();
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3b-i. Classes

```
var Personne = function(nom, prenom){
    this.nom = nom || 'Pons';
    this.prenom = prenom || 'Olivier';
};

Personne.prototype.getNom = function() {
    return 'Mon nom est ' + this.prenom +
        '... ' + this.nom + ' ' + this.prenom + '.';
};

var Employe = function(nom, prenom, job) {
    Personne.call(this, nom, prenom);
    this.job = job || 'sans travail';
};

Employe.prototype = new Personne();
Employe.prototype.constructor = Employe;

Employe.prototype.getNomEtPosition = function() {
    return 'Je suis ' + this.prenom + '... ' +
        this.nom + ' ' + this.prenom + '.' +
        " Mon travail : " + this.job + '.';
};
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3b-ii. Classes

```
function buildPersonne(aNom, aPrenom) {
    var personne = {},
        nom = aNom || 'Olivier',
        prenom = aPrenom || 'Pons';
    personne.getNom = function () {
        return 'Mon nom est ' + prenom +
            '... ' + nom + ' ' + prenom + '.';
    };
    return personne;
}

function buildEmploye(aNom, aPrenom, aJob) {
    var employe = buildPersonne(aNom, aPrenom),
        job = aJob || 'sans emploi';
    employe.getNomEtPosition = function() {
        return 'Je suis ' + aPrenom + '...' + aNom + ' ' +
            aPrenom + '.' + " Mon travail : " + job + '.';
    };
    return employe;
}
```


Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3b-iii. Classes

Premier slide = avec des constructeurs

```
var ePresley = new Employe ('Elvis', 'Presley', 'Star');  
console.log(ePresley.getNom());  
console.log(ePresley.getNomEtPosition());
```

Second cas = avec les fonctions normales

```
var jBond = buildEmploye('James', 'Bond', 'spy');  
console.log(jBond.getNom());  
console.log(jBond.getNomEtPosition());
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3c-i. Subtilités

Deux codes : un seul fonctionne.
Lequel ? Pourquoi ?

```
function Test() {  
    this.t=function() {  
        var self = this,  
            other = -1,  
            self.tutu = 15;  
        console.log(self);  
    }  
}  
var olivier = new Test();
```

```
function Test() {  
    this.t=function() {  
        var self = this,  
            other = -1;  
        self.tutu = 15;  
        console.log(self);  
    }  
}  
var olivier = new Test();
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3c-ii. Subtilités

Tapez ce qui suit :

```
var tab = [];  
tab['a'] = 12;  
tab['b'] = "test";
```

Puis
tab['b']
tab.length

Devinez les réponses.

Tapez ce qui suit :

```
var tab = {};  
tab['a'] = 12;  
tab['b'] = "test";
```

Puis
tab['b']
tab.length

Devinez les réponses.

Il y a deux incohérences. Lesquelles ? Pourquoi ?

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3c-iii. Subtilités

Le code qui suit ne fonctionne pas :

```
var monObj = {  
  nom: 'Olivier',  
  hiRev: function() {  
    function getRevNom() {  
      var rev = '';  
      for(let i=this.nom.length-1; i>=0; i--) {  
        rev += this.nom[i];  
      }  
      return rev;  
    }  
    return getRevNom()+' sius ej, olleH';  
  }  
};  
monObj.hiRev();  
Pourquoi ?
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3c-iii. Subtilités

Solution :

```
var monObj = {  
  nom: 'Olivier',  
  hiRev: function() {  
    var self = this;  
    function getRevNom() {  
      var rev = '';  
      for(let i=self.nom.length-1; i>=0; i--) {  
        rev += self.nom[i];  
      }  
      return rev;  
    }  
    return getRevNom()+' sius ej, olleH';  
  }  
};  
monObj.hiRev();
```



Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3c-iii. Subtilités

A éviter, en Php même problème qu'en JavaScript :

```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

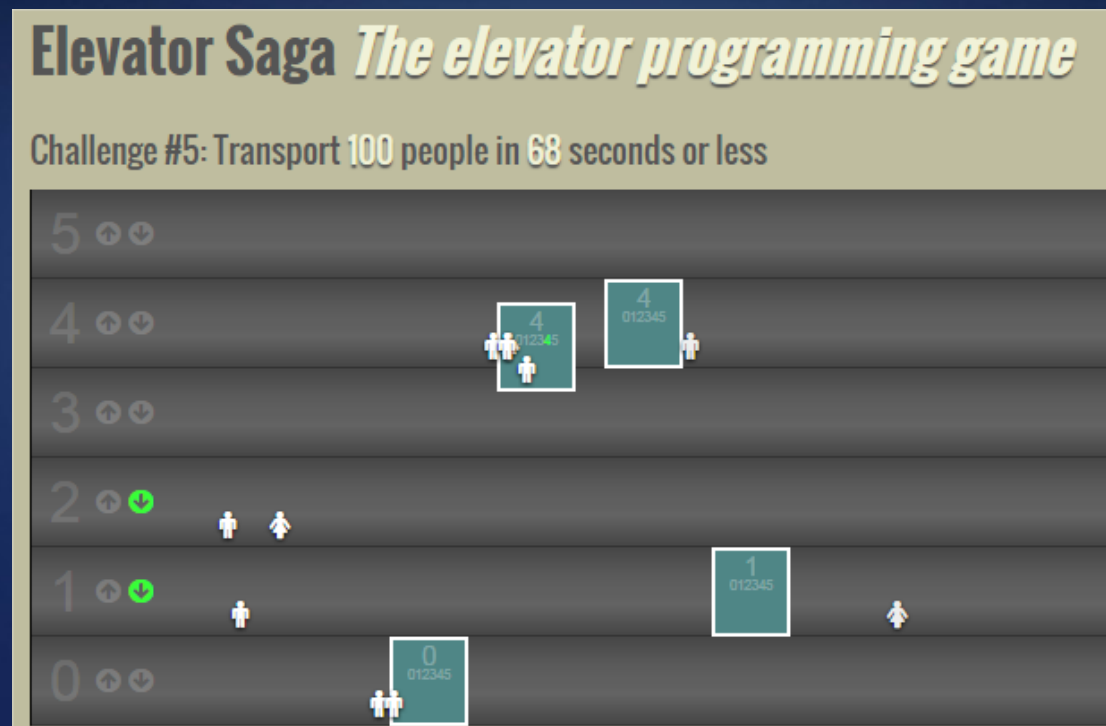


Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-i. Divers

Apprenez en jouant :
<http://play.elevatorsaga.com/#challenge=1>

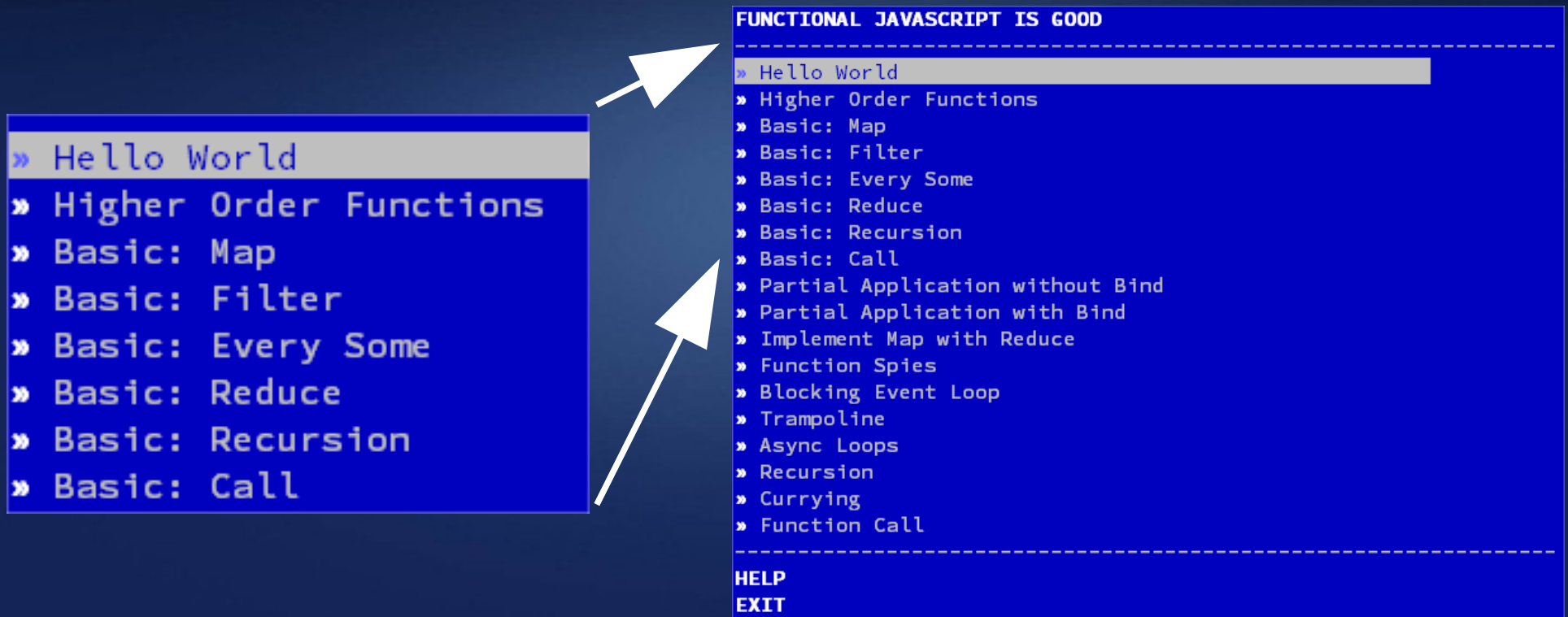


Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-ii. Divers

<http://nodeschool.io/>
`npm install -g functional-javascript-workshop`
`functional-javascript-workshop`



Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-i. ES6 – let – templates

```
let maVar;           // portée par bloc (enfin !)  
const MA_CONST = "T"; // constantes
```

String templates

```
let tempActuelle = 19.5;  
const msg = `La température est ${tempActuelle} \u00b0C`;
```

Symboles = valeurs uniques

```
const ROUGE = Symbol();  
const ORANGE = Symbol("Bonjour !");  
ROUGE === ORANGE // false : un symbol est unique
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-ii. ES6 – set – for .. of

Sets

```
>new Set([1,45,45,45,45,1])
```

```
Set {1, 45}
```

for .. of

```
const verbes = ["ri", "eu peur", "crié"];  
for(let verbe of verbes) {  
    console.log(`J'ai...${verbe}!`);  
}
```

3d-iii. ES6 – Assignations déstructurées

Destructuring assignments

```
const obj = { b: 2, c: 3, d: 4 };  
const {a, b, c} = obj;  
a; // undefined: pas de propriété "a" dans obj  
b; // 2  
c; // 3  
d; // erreur de référence: "d" is not defined
```

Attention, assignation seule = parenthèses obligatoires

```
const obj = { b: 2, c: 3, d: 4 };  
let a, b, c;  
{a, b, c} = obj; // erreur  
({a, b, c} = obj); // pas d'erreur
```

3d-iii. ES6 – Assignations déstructurées

```
let [x, y] = arr;  
x; // 1  
y; // 2  
z; // erreur: z hasn't been defined
```

Opérateur "..."

```
const tab = [1, 2, 3, 4, 5];  
let [x, y, ...reste] = tab;  
x; // 1  
y; // 2  
reste; // [3, 4, 5]
```

3d-iii. ES6 – Assignations déstructurées

Echanger deux valeurs

```
let a = 5, b = 10;
```

```
[a, b] = [b, a];
```

```
a; // 10
```

```
b; // 5
```

3d-iv. ES6 – Arguments déstructurés

Les propriétés doivent être des chaînes qui correspondent

```
function maPhrase({ sujet, verbe, objet }) {  
    return `${sujet} ${verbe} ${objet}`;  
}  
const o = {  
    sujet: "Je",  
    verbe: "préfère",  
    objet: "Python",  
};  
maPhrase(o);
```

JavaScript / Json / AJAX

1 - JavaScript - Définition + bases

3d-iv. ES6 – Arguments déstructurés

Avec des tableaux

```
function maPhrase([ sujet, verbe, objet ]) {  
    return `${sujet} ${verbe} ${objet}`;  
}  
const arr = [ "JavaScript", "copie", "Python" ];  
maPhrase(arr);
```

Avec les "..."

```
function maFn(prefixe, ...mots) {  
    const retour = [];  
    for(let i=0; i<mots.length; i++) {  
        retour[i] = prefixe + mots[i];  
    }  
    return retour;  
}  
maFn("Py", "lône", "thon");
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-v. ES6 – Arguments par défaut

Avec des tableaux

```
function f(a, b = "default", c = 3) {  
    return `${a} - ${b} - ${c}`;  
}  
f(5, 6, 7); // "5 - 6 - 7"  
f(5, 6);    // "5 - 6 - 3"  
f(5);       // "5 - default - 3"  
f();        // "undefined - default - 3"
```


Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-vi. ES6 – Fonctions / propriétés

Avant

```
var o = {  
  nom: 'Olivier',  
  crier: function() { return 'Argh !'; }  
}
```

Maintenant

```
var o = {  
  nom: 'Olivier',  
  crier(): { return 'Argh !'; } // plus court  
}
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-vii. ES6 – Notation fléchée

- Remplace le mot-clé "function"
- Est toujours anonyme
- Si le corps n'est qu'une instruction, "{}" facultatifs
- mot-clé arguments pas utilisable

Identiques :

```
const f1 = function() { return "hello!"; }
```

```
const f1 = () => "hello!";
```

```
const f2 = function(name) { return `Hello, ${name}!`; }
```

```
const f2 = name => `Hello, ${name}!`;
```

```
const f3 = function(a, b) { return a + b; }
```

```
const f3 = (a,b) => a + b;
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-vii. ES6 — Mode strict

Il suffit de mettre "use strict"

sur une ligne seule, avant le code.

Attention : "use strict" prend en compte le scope :

- dans une fonction, que dans la fonction
- dans le scope global, il s'applique à tout y compris aux autres scripts.

Solution, mettre tout son code dans une IIFE

(= Immediately Invoked Function Expression)

(= self invoking function)

```
(function() {  
    "use strict";  
    /* mon code */  
})();
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-viii. ES6 – Compatibilité

<https://kangax.github.io/compat-table/es6/>

The screenshot shows the ES6 compatibility table with various features listed on the left and a grid of compatibility data on the right. An overlay titled "Functions" displays examples of arrow functions and their return values. A hand cursor points to the "return" statement in the example code.

Feature name	V8	SpiderMonkey	JavaScriptCore	Chakra	Carakan	KJS	Other
Optimisation							
proper tail calls (tail call optimisation)							
Syntax							
default function parameters							
rest parameters							
spread (...) operator							
object literal extensions							
for...of loops							
octal and binary literals							
template literals							
RegExp "y" and "u" flags							
destructuring, declarations							
destructuring, assignments							
destructuring, parameters							
Unicode code point escapes							
new.target							
Bindings							
const							
let							
block-level function declaration ^{[1][2]}							
Functions							
arrow functions							
class							
super							
generators							
Built-ins							
typed arrays							

Functions

arrow functions

0 parameters

1 parameter, no brackets

multiple parameters

return (() => 5)() === 5;

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-ix. ES6 – Evolution saine

```
{
  const x = 'blue';
  console.log(x);
}
console.log(typeof x);
{
  const x = 3;
  console.log(x);
}
console.log(typeof x);
```

```
{
  let x = 'blue';
  console.log(x);
  {
    let x = 3;
    console.log(x);
  }
  console.log(x);
}
console.log(typeof x);
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-ix. ES6 – Evolution saine

```
const arr = new Array(5).fill(1); // [1, 1, 1, 1, 1]
arr.fill("a"); // ["a", "a", "a", "a", "a"]
arr.fill("b", 1); // ["a", "b", "b", "b", "b"]
arr.fill("c", 2, 4); // ["a", "b", "c", "c", "b"]
arr.fill(5.5, -4); // ["a", 5.5, 5.5, 5.5, 5.5]
arr.fill(0, -3, -1); // ["a", 5.5, 0, 0, 5.5]
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-ix. ES6 – Evolution saine

```
const arr = [{ id: 5, n: "Inès" },
              { id: 7, n: "Eric" }];
arr.findIndex(o => o.id === 5);           // 0
arr.findIndex(o => o.n === "Eric");       // 1
arr.findIndex(o => o === 3);              // -1
arr.findIndex(o => o.id === 17);          // -1
arr.find(o => o.id === 5);                 // object
                                           // { id: 5, n: "Inès" }
arr.find(o => o.id === 2);                 // null
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-x. ES6 – Exemples concrets

```
$(()) => {  
  const comparator = (a, b, direction = 1) =>  
    a < b ? -direction : (a > b ? direction : 0),  
  const sortDirection = 1;  
  
  $('#t-2').find('tbody > tr').sort((a, b) => comparator(  
    $(a).data('book')[sortKey],  
    $(b).data('book')[sortKey],  
    sortDirection  
  ));  
});
```


Javascript / Json / AJAX

1 - JavaScript - Définition + bases

3d-x. ES6 – Exemples concrets

```
$(() => {  
  const stripe = () => {  
    $('#news')  
      .find('tr.alt')  
      .removeClass('alt')  
      .end()  
      .find('tbody')  
      .each((i, element) => {  
        $(element)  
          .children(':visible')  
          .has('td')  
          .filter(i => (i % 4) < 2)  
          .addClass('alt');  
      });  
  }  
  stripe();  
});
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

4a. Préparation NodeJS

- Installer git en ligne de commandes
 - Installer nodejs
 - > Normalement il installe npm avec et met tout dans le PATH
- Vous devez taper "git" en ligne de commande et obtenir :

```
usage: git [--version] [--help] [-c name=value]
          [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
          [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
          [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
          <command> [<args>]
```

Vous devez taper "node" en ligne de commande et obtenir :

```
C:\Users\Olivier\test>node -v
v5.10.1
C:\Users\Olivier\test>
```

Javascript / Json / AJAX
1 - JavaScript - Définition + bases

4b-i Préparation NodeJS

Build tools : outils de construction. Grunt ou gulp

- Installer gulp

Transpilers / Transcompilers

Babel et traceur transforment (entre autres)

ES6 en code compatible ES5 (100% navigateurs)

- Installer babel

Code propre : linter

- Installer eslint

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

4b-ii Préparation NodeJS

```
sudo apt-get install nodejs npm
sudo npm install -g gulp
sudo npm install -g gulp-babel
sudo npm install -g eslint
mkdir test
cd test
npm init
npm install --save underscore
npm install --save gulp
npm install --save gulp-babel
npm install --save-dev babel-preset-es2015
npm install --save-dev gulp-eslint
```

4b-iii Préparation NodeJS

- dans "test", créer ces dossiers
 - "es6" → côté NodeJS, ES6 script
 - "public"
 - "public/es6" → côté Client Web, ES6 script
 - "public/dist" → code ES6 transpilé ES5 = "distribuable"
- dans "test", lancer "eslint --init", répondre aux questions et choisir de créer un fichier JSON
- dans "test", créer un fichier ".babelrc" et y mettre :
{ "presets": ["es2015"] }

4b-iv Préparation NodeJS

- Créer un fichier ".gitignore" et y mettre :

```
# npm debugging logs
npm-debug.log*
*.log

# project dependencies
node_modules

# OSX folder attributes
.DS_Store

# temporary files
*.tmp
*~
*.bak
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

4b-v Préparation NodeJS

- Créer un fichier "gulpfile.js" et,
pour le faire fonctionner et y mettre :

```
const gulp = require('gulp');
const babel = require('gulp-babel');
const eslint = require('gulp-eslint');
gulp.task('default', function() {
  gulp.src(["es6/**/*.js", "public/es6/**/*.js"])
    .pipe(eslint())
    .pipe(eslint.format());
  gulp.src("es6/**/*.js")
    .pipe(babel())
    .pipe(gulp.dest("dist"));
  gulp.src("public/es6/**/*.js")
    .pipe(babel())
    .pipe(gulp.dest("public/dist"));
});
```

4b-vi Préparation NodeJS

- Modifier le fichier de configuration ".eslintrc.json" afin d'autoriser la virgule sur un dernier élément et autoriser le "console.log();" :

```
"comma-dangle": [  
    "off",  
    "always-multiline"  
],  
"no-console": [  
    "off"  
]
```


4b-vii Préparation NodeJS

- Récapitulatif :
 - npm doit être installé
 - git doit être installé
 - node doit être installé
 - gulp doit être installé
 - babel doit être installé
 - eslint doit être installé
 - un dossier "test" pour écrire en ES6 avec ces outils doit être prêt

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

4b-vii Préparation NodeJS

Créer un fichier es6/test.js :

```
'use strict';
const phrases = [
  { subject: 'JavaScript', verb: 'est', object: 'en version ES6' },
  { subject: 'Le chien', verb: 'est', object: 'joli' },
];
// es6: "object destructuring"
function say({ subject, verb, object }) {
  // es6: chaînes template
  console.log(`${subject} ${verb} ${object}`);
}
// es6: let = portée par bloc
// es6: for..of
for(let s of phrases) {
  say(s);
}
```

Javascript / Json / AJAX

1 - JavaScript - Définition + bases

4b-viii Préparation NodeJS

Le transpiler avec gulp :

```
# gulp
[23:24:50] Using gulpfile ~\test\gulpfile.js
[23:24:50] Starting 'default'...
[23:24:50] Finished 'default' after 190 ms
# l dist/
total 4,0K
-rwxrwx---+ 1 olivier olivier 1,1K  23:24 test.js
```

Copier coller le fichier dans public/es6

Le transpiler avec gulp :
puis vérifier ce qu'il y a dans public/dist

4b-ix Préparation NodeJS

- A partir de là : **boucle infinie** :

1 - développer, faire des modifications

2 - **gulp** pour tester

3 - boucle sur 2 tant qu'il y a des erreurs **eslint**

4 - "**git status**" pour s'assurer que tout est bien pris en compte, ou qu'il faut ignorer des fichiers qu'on a ajouté entretemps (ajout dans "**.gitignore**")

5 - ajouter tous les changements "**git add -A**"

6 - committer ces changements :

git commit -m "Explication claire"