JavaScript

ES6

ES7

Introduction au langage

Franck LAMY – BTS SIO

JAVASCRIPT

JavaScript est un langage de programmation principalement connu pour sa capacité à ajouter des fonctionnalités interactives aux pages web.

JAVASCRIPT

Créé en 1995 (10 jours) par Brendan Eich chez Netscape

ECMAScript

- JavaScript est normalisé ECMAScript (ES)
- ECMAScript : ensemble de normes concernant les langages de programmation de type script dont JavaScript
- **Evolution majeure : ES6 en 2015 (ES2015)**
  "Révolution" pour le JavaScript
- Ensuite différentes évolutions depuis 2015 !

Environnement d'exécution

Au sein du navigateur

script js **"embarqué" dans une page HTML. Interprété par le navigateur**

En dehors du navigateur

script js **exécuté par node.js**

Variables vs Constantes

1 Le mot clé let

```
let a = 10;
console.log(a);
```

```
let a;
a = 10;
console.log(a);
```

```
let a = 10;
a = 20;
console.log(a);
```

2 Le mot clé const

```
const a = 10;
console.log(a);
```

```
const a;
a = 10;
console.log(a);
```


```
const a = 10;
a = 20;
console.log(a);
```

Chaines de caractères

Déclarer une chaine de caractères

```
let promotion = 'BTS SIO2';
console.log(a);
```

```
let promotion = "BTS SIO2";
console.log(a);
```

```
let promotion = `BTS SIO2`;
console.log(a);
```

Template literals

Une chaine peut-être considérée comme un objet (String)

Méthodes

Une chaine est **immutable**

Tableaux

Déclarer un tableau

```
const tab = [10,20,44,12];
console.log(tab);
```

Un tableau est un objet (Array)

Méthodes

Parcours tableaux

```
const tab = [10,20,44,12];
for(let i=0; i<tab.length; i++) {
    console.log(tab[i]);
}
```

Parcours tableaux – ES6

```
const tab = [10,20,44,12];
for(let nombre of tab) {
    console.log(nombre);
}
```

ES6

Tableaux

Ajouter un élément : méthode push

```
const tab = [10,20,44,12];
console.log(tab);
tab.push(66);
console.log(tab);
```

Pourtant le tableau est déclaré en const

Fonctions

Plusieurs manières de créer des fonctions

1. **Fonction 'classique'**
2. **Fonction anonyme**

ES6

3. **Fonction fléchée : arrow function**

Fonction classique

1 **Manière 'classique'**

```
function addition (a,b) {
        return a+b;
}

console.log(addition(2,3));
```

Pas de typage des paramètres et du type de retour

TypeScript

Fonction anonyme

2 **Fonction anonyme**

Fonction anonyme

```
const addition = function (a,b) {
        return a+b;
}
console.log(addition(2,3));
```

Utilisation de la constante comme représentante de la fonction anonyme

Fonction fléchée

ES6

3 **Fonction fléchée : arrow function**

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

```
const addition = (a,b) => {
    return a+b;
}

console.log(addition(2,3));
```

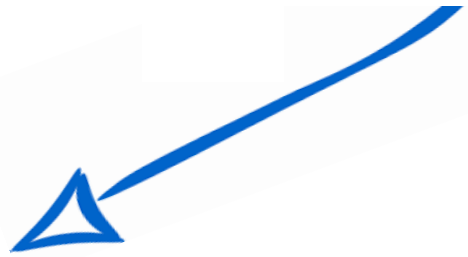
Fonction anonyme

Fonction fléchée

3 Fonction fléchée : arrow function

```
const addition = (a,b) => {
    return a+b;
}
```

Si une seule instruction return ?

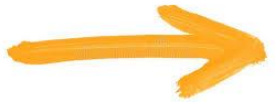


```
const addition = (a,b) => a+b;
```

Fonction fléchée

3 Fonction fléchée : arrow function

```
const addition = (a,b) => {
        const res =  a+b;
        return res;
}
```



Si plusieurs instructions !

Fonction fléchée

3 Fonction fléchée : arrow function

```
const salut = () =>  console.log('Salut');
salut();
```

Si pas de paramètres !

```
const puissance2 = a => a*a;

console.log(puissance2(10));
```

Si un seul paramètre !

Exemples d'application

Exécuter une fonction pour chaque élément d'un tableau

```
let nombres = [2,8,15,17,26,9,42,4];
```

Afficher tous les nombres pairs

```
for (const nombre of nombres) {
    if (nombre % 2 === 0) {
        console.log(nombre);
    }
}
```

Méthode forEach()

MDN MOZILLA DEVELOPER NETWORK

Exemples d'application

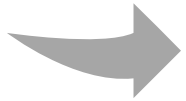
Récupérer tous les nombres > 10

Filtrer un tableau

```
let nombres = [2,8,15,17,26,9,42,4];
```

```
[15,17,26,42]
```

```
let nombresSup10 = [];
for (const nombre of nombres) {
    if (nombre > 10) {
        nombresSup10.push(nombre);
    }
}
```



Méthode filter()

MDN MOZILLA DEVELOPER NETWORK

Exemples d'application

Récupérer tous les nombres à la puissance 2

Mapper un tableau

```
let nombres = [2,8,15,17,26,9,42,4];
```

→ `[4,64,225,289,676,81,1764,16]`

```
let nombresPuissance2 = [];
for (const nombre of nombres) {
  nombresPuissance2.push(nombre*nombre);
}
```

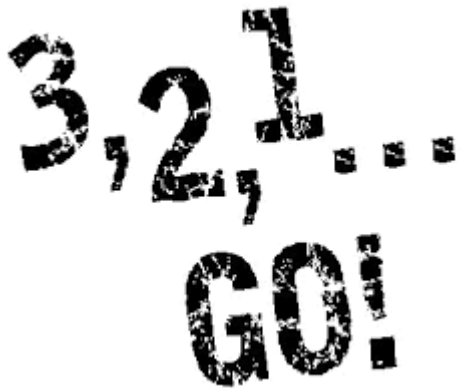
→ **Méthode map()**

Exemples d'application

Récupérer tous les éléments pairs à la puissance 2

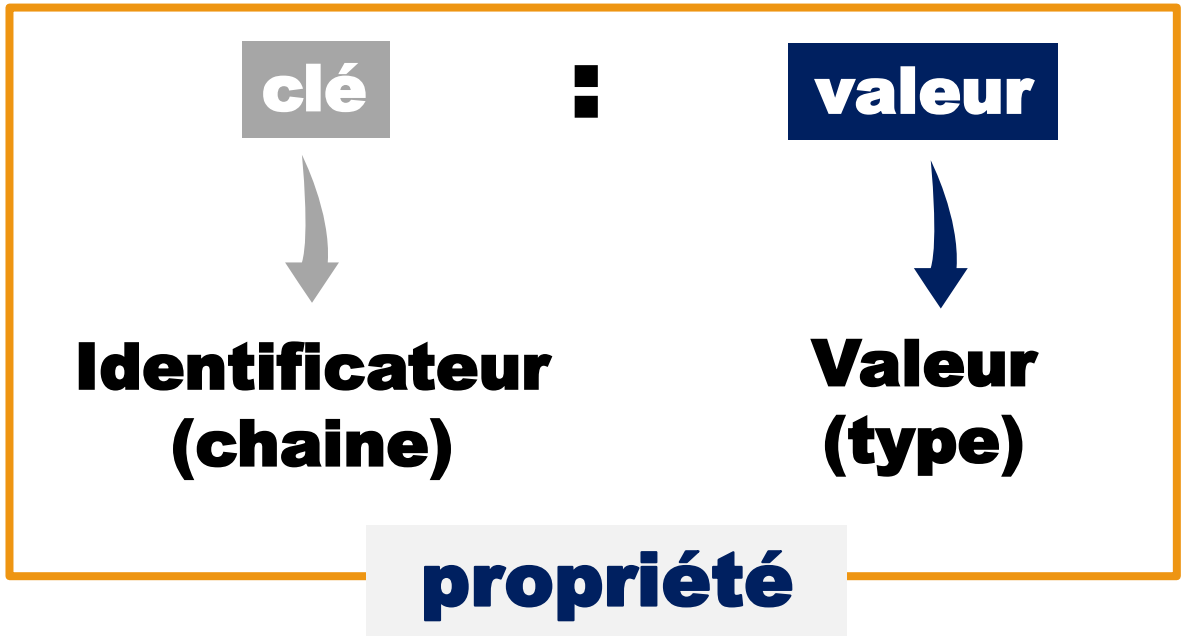
```
let nombres = [2,8,15,17,26,9,42,4];
```

→ **[4, 64, 676, 1764, 16]**



Objets littéraux

- **Regrouper plusieurs valeurs au sein d'une même entité : un objet**
- **Collection de paires**



Objets littéraux

Définir un objet


```
const un_objet = {
        nom : "Dupond",
        prenom : "Jean",
        age : 30
}

console.log(un_objet);

console.log(un_objet.nom);
```

propriétés

un_objet : objet littéral

{ } : délimite l'objet

Tableau associatif

Accès à la propriété nom

Objets littéraux

Définir un objet complexe

Objet imbriqué

```
const un_objet = {
            nom : "Dupond",
            prenom : "Jean",
            age : 30,
            adresse : {
                rue : "Rue de la gare",
                codePostal    : "25000",
                ville : "Besançon"
            }
}

console.log(un_objet);

console.log(un_objet.adresse.ville);
```

adresse est un objet

Objets littéraux

Définir un objet "plus" complexe

```
const un_objet = {
            nom : "Dupond",
            prenoms : ["Jean","Pierre"],
            age : 30,
            adresse : {
                rue : "Rue de la gare",
                codePostal    : "25000",
                ville : "Besançon"
            }
}

console.log(un_objet);

console.log(un_objet.prenoms[0]);
```

prenoms est un tableau

adresse est un objet

Objets littéraux

Accéder aux propriétés

```
const un_objet = {
        nom : "Dupond",
        prenom : "Jean",
        age : 30
}
```

Accès "classique"

```
console.log(un_objet.nom);
```

Accès "tableau"

```
console.log(un_objet['nom']);
```

Destructuring

Définir le destructuring

Le destructuring est une syntaxe introduite en JavaScript avec ES6 (ECMAScript 2015) qui permet de décomposer des valeurs d'un tableau ou des propriétés d'un objet en variables distinctes.

Destructuring

Déstructurer un tableau

```
const nombres = [12,4,5,8];
let a = nombres[0];
let b = nombres[1];
console.log(a);
console.log(b);
```

SANS


```
const nombres = [12,4,5,8];
let [a,b] = nombres;
console.log(a);
console.log(b);
```

AVEC

Destructuring

Déstructurer un objet

SANS

```
const un_objet = {
        nom : "Dupond",
        prenom : "Jean",
        age : 30
}
let nom = un_objet.nom;
let prenom = un_objet.prenom;
let age = un_objet.age;

console.log(nom);
```

AVEC

```
const un_objet = {
        nom : "Dupond",
        prenom : "Jean",
        age : 30
}
let {nom,prenom,age} = un_objet;

console.log(nom);
```

Destructuring

Illustrer avec un exemple concret

```
const un_objet = {
        nom : "Dupond",
        prenom : "Jean",
        age : 30
}
```

```
function getIdentite(objet)
        return objet.nom + ' ' + objet.prenom;
}

console.log(getIdentite(un_objet));
```

```
function getIdentite({prenon,nom})
        return nom + ' ' prenom;
}

console.log(getIdentite(un_objet));
```