





Les bases du langage



ECMASCRIPT

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





EXÉCUTION CODE JAVASCRIPT



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 – ES6



2 manières de créer des variables

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);

TABLEAUX



Déclaration d'un tableau avec const

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



Parcours tableaux

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



Parcours tableaux - ES6

```
const tab = [10,20,44,12];
for(let nombre of tab) {
   console.log(nombre);
                            ES<sub>6</sub>
```



TABLEAUX



Ajout d'un élément : push

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





Pourtant le tableau est déclaré en const!!!



Copie de tableau

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

Copie de tableau - Shallow-copy

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

TABLEAUX



Copie de tableau : slice

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

```
const tab = [10,20,44,12];
const tab2 = tab.slice(1,3);
console.log(tab);
console.log(tab2);
```



Plusieurs manières de créer des fonctions

- Manière historique
- **Utilisation d'une constante**

ES₆

Utilisation d'une arrow function : fonction fléchée





Manière historique

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



2

Utilisation d'une constante

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

Fonction anonyme

Utilisation de la constante comme représentante de la fonction anonyme



ES₆



Utilisation d'une arrow function : fonction fléchée

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

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

Fonction anonyme



ES6



Utilisation d'une arrow function : fonction fléchée

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

```
const addition = (a,b) => a+b;
console.log(addition(2,3));
```





ES₆



Utilisation d'une arrow function : fonction fléchée

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



Si plusieurs instructions?



ES₆



Utilisation d'une arrow function : fonction fléchée

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



Si une seule instruction sans return?



ES₆



Utilisation d'une arrow function : fonction fléchée

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



Si pas de paramètres

```
const puissance2 = a => a*a;
console.log(puissance2(10));
```



Si un seul paramètre



PARAMÈTRES REST



Somme des valeurs d'un tableau

```
const mesNombres = [12,3,7,9,55];
const calculerSomme = nombres => {
 let somme = 0;
 for (let nombre of nombres) {
   somme += nombre;
  return somme;
console.log(calculerSomme(mesNombres));
```

nombres est un paramètre de type tableau



PARAMÈTRES REST



Somme des valeurs d'un tableau

```
const calculerSomme =(...nombres) => {
  let somme = 0;
 for (let nombre of nombres) {
    somme += nombre;
  return somme;
console.log(calculerSomme(12,4,5,55));
console.log(calculerSomme(12,4,5,55,99,8,7));
console.log(calculerSomme(12,4,5,55,99,8,7,44,22));
```

nombres est un paramètre rest



C'est un tableau



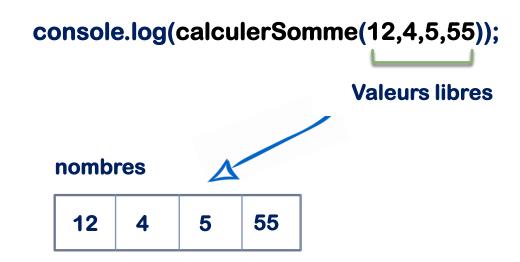
PARAMÈTRES REST



Somme des valeurs d'un tableau

```
const calculerSomme = (...nombres) => {
  let somme = 0;
  for (let nombre of nombres) {
     somme += nombre;
  }
  return somme;
}

console.log(calculerSomme(12,4,5,55));
  console.log(calculerSomme(12,4,5,55,99,8,7));
  console.log(calculerSomme(12,4,5,55,99,8,7,44,22));
```







OPÉRATEUR SPREAD



Opérateur spread : ...

Tableau

```
TRANSFORME Statement
```



OPÉRATEUR SPREAD



Copie de tableau "traditionnelle"

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



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



Copie de tableau avec opérateur spread

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

```
const tabClone = [10,20,44,12]
                  Nouveau tableau
```

OPÉRATEUR SPREAD



Concaténation de tableaux "traditionnelle"

```
const pairs = [0,2,4,6,8];
const impairs = [1,3,5,7,9];
const nombres = pairs.concat(impairs);
console.log(nombres);
```



Concaténation de tableaux avec opérateur spread

```
const pairs = [0,2,4,6,8];
const impairs = [1,3,5,7,9];
const nombres = [...pairs, ...impairs];
console.log(nombres);
```



DESTRUCTURING DE TABLEAUX



Exemple

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



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

Destructuring de tableaux

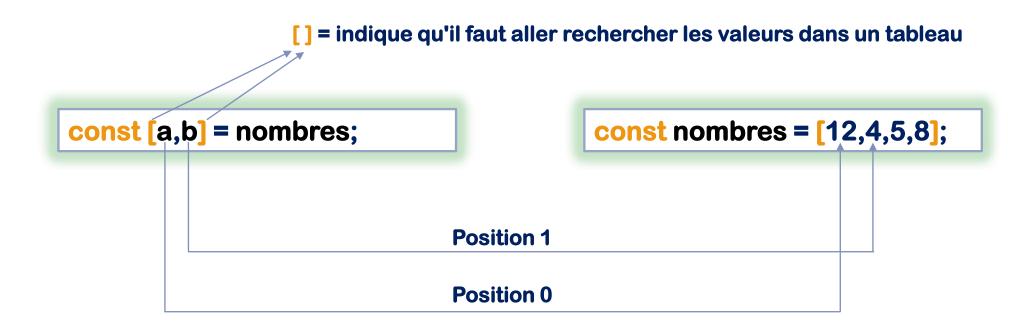


Affecter des variables à partir d'informations stockées dans un tableau



DESTRUCTURING DE TABLEAUX

Principe







Notion centrale en javascript

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

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

Accès l'attribut nom
```





Un objet un peu plus complexe



Objets imbriqués

```
const un_objet = {
        nom: "Dupond",
        prenom: "Jean",
        age: 30,
        adresse: {
          rue: "Rue de la gare",
          codePostal
                       : "39100",
                                                  adresse est un objet
          ville: "Dole"
console.log(un_objet);
console.log(un_objet adresse ville);
```





Un objet un peu plus plus complexe

```
const un_objet = {
        nom: "Dupond",
        prenoms: ["Jean", "Pierre"],
                                                prenoms est un tableau
        age: 30,
        adresse: {
          rue: "Rue de la gare",
          codePostal
                       : "39100",
                                                  adresse est un objet
          ville: "Dole"
console.log(un_objet);
console.log(un_objet • prenoms[0]);
```



Accéder aux attributs d'un objet

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

Accès "classique"

```
console.log(un_objet_nom);
```

Accès "tableau"

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





Copier des objets

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

```
const un_objet_copie = un_objet
un_objet_copie.age = 55;
console.log(un_objet);
console.log(un_objet_copie);
```







Copier des objets : utilisation de l'opérateur spread ...

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

```
const un_objet_copie = {...un_objet}
un_objet_copie.age = 55;
console.log(un_objet);
console.log(un_objet_copie);
```







Copier des objets avec ajout d'attributs

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



Ajout d'un nouvel attribut dans le nouvel objet

```
const un_objet_copie = {...un_objet, sexe : 'M'}
console.log(un_objet);
console.log(un_objet_copie);
```





Copier des objets avec modification d'attributs

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



Modification de l'attribut "age" dans le nouvel objet

```
const un_objet_copie = {...un_objet, age : 55}
console.log(un_objet);
console.log(un_objet_copie);
```





Exemple

```
const un_objet = {
       nom: "Dupond",
       prenom: "Jean",
       age: 30
const nom = un_objet.nom;
const prenom = un_objet.nom;
const age = un_objet;
console.log(nom);
```



```
const un_objet = {
       nom: "Dupond",
       prenom: "Jean",
       age: 30
const {nom,prenom,age} = un_objet;
console.log(nom);
```

Destructuring d'objets

Affecter des variables à partir d'informations stockées dans un objet





{} = indique qu'il faut aller rechercher les valeurs dans un objet

```
const {nom,prenom,age} = un_objet;
```

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



Exemple2: renommage des variables

```
const un_objet = {
       nom: "Dupond",
       prenom: "Jean",
       age: 30
const {nom:n,prenom:p,age:a} = un_objet;
console.log(n);
```



Exemple3: utilisation d'une fonction

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

```
function getIdentite(objet)
       return objet.nom + ' ' + objet.prenom;
console.log(getIdentite(un_objet));
```

Destructuring d'objets



```
function getIdentite2({prenom,nom})
       return nom + ' ' + prenom;
console.log(getIdentite(un_objet));
```





Exemple3: utilisation d'une fonction

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

```
function getIdentite2({prenom,nom})
    return nom + ' ' + prenom;
}
console.log(getIdentite(un_objet));
```

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



Version fonction fléchée! () => {}





Exemple4: objets imbriqués

```
const un_objet = {
    nom : "Dupond",
    prenom :"Jean",
    age : 30,
    adresse : {
       rue : "Rue de la gare",
       codePostal : "39100",
       ville : "Dole"
    }
}
```



Récupérer le nom et la ville dans 2 variables "nom" et "ville"





Déclaration d'un tableau d'objets

```
const personnes = [
    {id:1, nom:'Dupond', prenom : 'Jean', age:55},
    {id:2, nom:'Martin', prenom : 'Martine', age:40},
    {id:3, nom:'Durand', prenom : 'Pierre', age:50},
    {id:4, nom:'Doe', prenom : 'John', age:33},
]
```





Parcours d'un tableau : méthode "traditionnelle"

```
const personnes = [
    {id:1, nom:'Dupond', prenom : 'Jean', age:55},
    {id:2, nom:'Martin', prenom : 'Martine', age:40},
    {id:3, nom:'Durand', prenom : 'Pierre', age:50},
    {id:4, nom:'Doe', prenom : 'John', age:33},
]
```

```
for(let personne of personnes) {
  console.log(personne);
}
```

Traitement à réaliser sur chaque élément du tableau





Parcours d'un tableau : for Each

```
for(let personne of personnes) {
   console.log(personne);
}
```



```
function afficher(personne) {
    console.log(personne);
} Fonction de callback

personnes.forEach(afficher);
```



forEach prend en paramètre une référence sur une fonction



Fonction de callback

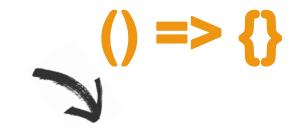
forEach appelle la fonction de callback pour chaque élément du tableau





Parcours d'un tableau : for Each

```
function afficher(personne) {
   console.log(personne);
} Fonction de callback
personnes.forEach(afficher);
```



const afficher = personne => console.log(personne);
personnes.forEach(afficher);

I over

personnes.forEach(personne => console.log(personne));





Création d'un nouveau tableau à partir d'un tableau existant

```
const personnes = [
    {id:1, nom:'Dupond', prenom : 'Jean', age:55},
    {id:2, nom:'Martin', prenom : 'Martine', age:40},
    {id:3, nom:'Durand', prenom : 'Pierre', age:50},
    {id:4, nom:'Doe', prenom : 'John', age:33},
]
```

```
const personnes2 = [];
for(let personne of personnes) {
    personnes2.push(personne.prenom + ' ' + personne.nom.toUpperCase());
}
```







Création un nouveau tableau à partir d'un tableau existant : map

```
const personnes2 = [];
for(let personne of personnes) {
    personnes2.push(personne.prenom + ' ' + personne.nom.toUpperCase());
}
```



```
const personnes2 = personnes.map(
    function (personne) {
        return personne.prenom + ' ' + personne.nom.toUpperCase();
    }
        Fonction de callback
}.
```







Création d'un nouveau tableau à partir d'un tableau existant : map

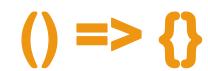
```
const personnes2 = personnes.map(
    function (personne) {
        return personne.prenom + ' ' + personne.nom.toUpperCase();
    }
    Fonction de callback
}:
```







```
const personnes2 = personnes.map( Fonction de callback personne => personne.prenom + ' ' + personne.nom.toUpperCase()
```







Création d'un nouveau tableau à partir d'un tableau existant : map

```
const personnes2 = personnes.map(personne =>
    personne.prenom + ' ' + personne.nom.toUpperCase()
);
```



map s'applique sur le tableau

map prend en paramètre une référence sur une fonction



Fonction de Callback

map appelle la fonction de callback pour chaque élément du tableau et ajoute le résultat dans le tableau qui sera renvoyé au final