



JavaScript

ES6

ES7

# 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

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);
```



# 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++) {  
  console.log(tab[i]);  
}
```

➤ Parcours tableaux – ES6

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

ES6

# 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);
```

# FONCTIONS

 Plusieurs manières de créer des fonctions

1

Manière historique

2

Utilisation d'une constante

3

Utilisation d'une arrow function : fonction fléchée

ES6



# FONCTIONS

1

## Manière historique

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

# FONCTIONS

## 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

# FONCTIONS

ES6

3

## 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

# FONCTIONS

ES6

3

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));
```



Si une seule  
instruction  
**return** ?



# FONCTIONS

ES6

3

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 ?

# FONCTIONS

ES6

3

Utilisation d'une **arrow function** : fonction fléchée

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



Si une seule  
instruction sans  
**return** ?

# FONCTIONS

ES6

3

## 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));
```

`console.log(calculerSomme(12,4,5,55));`

Valeurs libres

nombres



12	4	5	55
----	---	---	----

Valeurs libres → **T R A N S F O R M E** → Tableau

# OPÉRATEUR SPREAD

↪ Opérateur spread : ...

Tableau → **T R A N S F O R M E** → Valeurs libres

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

[10,20,44,12]

10    20    44    12

# 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

**[]** = indique qu'il faut aller rechercher les valeurs dans un tableau

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

```
const nombres = [12,4,5,8];
```

Position 1

Position 0

# OBJETS

➤ Notion centrale en javascript

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

attributs

un\_objet : **objet littéral**

**{ }** : délimite l'objet

➔ Accès l'attribut nom



# OBJETS

➤ 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",  
    ville : "Dole"  
  }  
}  
  
console.log(un_objet);  
  
console.log(un_objet.adresse.ville);
```



adresse est un objet

# OBJETS

➤ Un objet un peu plus complexe

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



**prenoms est un tableau**



**adresse est un objet**

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

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

# OBJETS

## ➤ 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']);
```

# OBJETS

## 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);
```



# OBJETS

➡ 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);
```



# OBJETS

➡ 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);
```

# OBJETS

➡ 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);
```

# DESTRUCTURING D'OBJETS

## Exemple

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



# DESTRUCTURING D'OBJETS

## Principe

**{}** = 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  
}
```

# DESTRUCTURING D'OBJETS

## 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);
```

# DESTRUCTURING D'OBJETS

## 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));
```

# DESTRUCTURING D'OBJETS

## 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 !  
`() => {}`

# DESTRUCTURING D'OBJETS

## 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"**

# TABLEAU D'OBJETS

➡ 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},  
]
```



# TABLEAU D'OBJETS

➡ 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

# TABLEAU D'OBJETS

## Parcours d'un tableau : **forEach**

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



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

**Fonction de callback**



**forEach** s'applique sur le tableau

**forEach** prend en paramètre une référence sur une fonction

➔ **Fonction de callback**

**forEach** appelle la fonction de callback pour chaque élément du tableau



# TABLEAU D'OBJETS



Parcours d'un tableau : **forEach**

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

Fonction de callback

**() => {}**

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

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



ouah

# TABLEAU D'OBJETS

➡ 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());  
}
```



# TABLEAU D'OBJETS



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**



# TABLEAU D'OBJETS



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**

pas mal.



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

**Fonction de callback**

ouah

**() => {}**

# TABLEAU D'OBJETS



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

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



Fonction de **callback**