



JavaScript Perfectionnement





Plan du cours

- ◇ Nouveauté en ES6
- ◇ Orienté objets en JS
- ◇ Import / Export
- ◇ Ajax et les Promesse
- ◇ Web component
- ◇ Complexité algorithmique



A decorative pattern of hexagons in various shades of blue and cyan on the left side of the slide. Some hexagons contain icons: a lightbulb, a thumbs up, a smartphone, a magnifying glass, and a gear.

1

◇ Nouveauté en ES6

Type de variable, Déstructuration, décomposition, Array, Set , Littéraux de gabarits , opérateur conditionnel , this , fonctions fléchés



Type de variable

Var

- permet de définir une variable globale ou locale à une fonction

Exemple:

```
var x = 1;  
if (true) {  
  var x = 2;  
  
  console.log(x); // Prints 2  
}  
  
console.log(x); // Prints 2
```





Type de variable

Let

- permet de déclarer des variables dont la portée est limitée à celle du bloc dans lequel elles sont déclarées

Exemple:

```
let toPrint = 'Bonjour';  
{  
  let toPrint = 'Au revoir';  
  console.log(toPrint); // Prints "Au revoir"  
}  
console.log(toPrint); // Prints 'Hello World'
```

```
{  
  let toPrint="Au revoir";  
}  
console.log(toPrint);  
// error: Uncaught ReferenceError: toPrint is not defined
```



Type de variable

Const

- variable accessible uniquement en lecture
- la valeur d'une constante ne peut pas être modifiée par des réaffectations ultérieures
- Une constante ne peut pas être déclarée à nouveau

Exemple:

```
const numero = 5;

try {
  numero = 10; // 'numero' Is a constant
  console.log(numero);
} catch (err) {
  console.log(err);
  // expected output: TypeError: invalid assignment to const `numero`
}
```



Déstructuration


Object Destructuring

- permet d'extraire des données d'un tableau ou d'un objet grâce à une syntaxe dont la forme ressemble à la structure du tableau ou de l'objet

Exemple:

```
let [a, b] = [10, 20];  
console.log(a); // Prints 10  
console.log(b); // Prints 20
```

```
const personne = {  
  nom: 'Doe',  
  prenom: 'John',  
  age: 25,  
};  
// sans destructuration  
const nom = personne.nom;  
const prenom = personne.prenom;  
const age = personne.age;  
// avec destructuration  
const { nom, prenom, age } = personne;
```





décomposition

Spread operator

- permet d'étendre un itérable en lieu et place de plusieurs arguments ou de plusieurs éléments (pour les littéraux de tableaux)

Exemple:

```
const list1 = [1,2,3,4]
const list2 = [6,7,8,9]
const listFinal = [...list1, 5 ,...list2]
console.log(listFinal)
// Prints List [1,2,3,4,5,6,7,8,9]
```





Littéraux de gabarits


template literals

- Avec eux, on peut utiliser des chaînes de caractères multi-lignes et des fonctionnalités d'interpolation

Exemple:

```
let name="Amir"
let myString =
  "Bonjour, je suis "+name+", je serais \n"+
  "votre formateur durant ce cours."
console.log(myString);
// Bonjour, je suis Amir je serais
// votre formateur durant ce cours.
```

```
let name="Amir"
let myString=
`Bonjour, je suis ${name}, je serais
votre formateur durant ce cours`
console.log(myString);
// Bonjour, je suis Amir je serais
// votre formateur durant ce cours.
```





opérateur conditionnel

conditional operator

- Cet opérateur est fréquemment utilisé comme raccourci pour la déclaration du if...else

Exemple:

```
let moyenne = 9

if(moyenne>=10){
  console.log("passant")
}
else{
  console.log("doublant")
}

// équivalent
moyenne>=10 ? console.log('passant') : console.log("doublant")
```



Tableau

ForEach

- permet d'exécuter une fonction donnée sur chaque élément du tableau sans avoir à se soucier de l'itérateur

Exemple:

```
const alphabets = ['a', 'b', 'c', 'd', 'e'];

alphabets.forEach(function(alphabet) {
  console.log(alphabet);
})

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






Tableau

Map

- crée un nouveau tableau avec les résultats de l'appel d'une fonction fournie sur chaque élément du tableau appelant

Exemple:

```
const personnes = [  
  {id:1,"Moyenne":12, nom:"Julien"},  
  {id:2,"Moyenne":9, nom:"Amir"},  
  {id:3,"Moyenne":14, nom:"Alex"},  
]  
  
const nomPersonnes = personnes.map(function(personne){  
  return personne.nom  
})  
  
console.log(nomPersonnes);  
// Prints ["Julien","Amir","Alex"]
```





Tableau

Filtre

- crée et retourne un nouveau tableau contenant tous les éléments du tableau d'origine qui remplissent une condition déterminée par la fonction callback

Exemple:

```
const personnes = [  
  {id:1,"Moyenne":12, nom:"Julien"},  
  {id:2,"Moyenne":9, nom:"Julien"},  
  {id:3,"Moyenne":14, nom:"Julien"},  
]  
  
// crée un nouveau tableau  
const passantFunc = personnes.filter(function(personne){  
  return personne.Moyenne>=10  
})  
  
console.log(passantFunc);  
  
// Print [{id:1,"Moyenne":12, nom:"Julien"}, {id:3,"Moyenne":14, nom:"Julien"} ]
```



Tableau

Sort

- trie les éléments d'un tableau, dans ce même tableau, et renvoie le tableau
- Par défaut, le tri s'effectue sur les éléments du tableau convertis en chaînes de caractères

Exemple:

```
const film= ["Mission Impossible","Avengers","Coco"]
console.log(film.sort())
// Prints ["Avengers","Coco","Mission Impossible"]
```

```
const chiffres= [10,20,3,40,50]
console.log(chiffres.sort())
// Prints [10,20,3,40,50]
// tri Les chaînes de caractères
const trier = chiffres.sort(function(a,b){
  | return a-b
})
console.log(trier)
// Prints [3,10,20,40,50]
```





Set

Set

- permet de stocker des valeurs uniques, de n'importe quel type, que ce soit des valeurs d'un type primitif ou des objets

Exemple:

```
const chiffres= [1,5,19,5,9,1]
const mySet = new Set(chiffres)
console.log(mySet)
// Prints Set {1,5,19,9}
```





This

This

- Fait référence à l'objet qui exécute la fonction
- si il est appelé à l'intérieur d'une méthode d'un objet, il fera référence à l'objet lui-même
- Si il est appelé à l'intérieur d'une fonction, il fera référence à l'objet global (window)





This

Exemple méthode:

```
let person= {  
  nom: "Amir",  
  parler(){  
    console.log(this)  
  }  
}  
person.parler()
```

CONSOLE ×

```
{  
  nom: "Amir",  
  parler: f parler {...}  
}
```





This

Exemple fonction:

```
function parler(){  
  console.log(this);  
}  
parler()
```

```
▼Window {0: global, window: Window, self: Window, document: document, name: "", location: Location, ...} ⓘ  
  ▶0: global {window: global, self: global, location: {...}, closed: false, frames: global, ...}  
  ▶JSCompiler_renameProperty: f (prop,obj)  
  ▶ShadyCSS: {prepareTemplate: f, prepareTemplateStyles: f, prepareTemplateDom: f, styleSubtree: f, styleElement: f, ...}  
  ▶alert: f alert()  
  ▶atob: f atob()  
  ▶blur: f blur()  
  ▶btoa: f btoa()  
  ▶caches: CacheStorage {}
```



Fonctions fléchées

Arrow function

- une syntaxe plus courte et l'absence de `this` spécifique à la fonction
- une fonction fléchée ne lie pas son propre `this` au sein de la fonction

Exemple:

```
const personnes = [  
  {id:1, "Moyenne":12},  
  {id:2, "Moyenne":9},  
  {id:3, "Moyenne":14},  
]  
  
const passantFunc = personnes.filter(function(personne){  
  return personne.Moyenne>=10  
})  
  
const passantFleche = personnes.filter(personne => personne.Moyenne>=10)  
  
// passantFunc et passantFleche auront le meme résultat
```

A decorative graphic on the left side of the slide. It features a large cyan hexagon with the number '2' inside. Surrounding this central hexagon are several smaller hexagons of varying shades of blue and cyan. Some of these smaller hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and radiating lines.

2

◇ Orienté objet en JS

A decorative graphic on the left side of the slide. It features a large cyan hexagon with the number '3' inside. Surrounding this central hexagon are several smaller hexagons of varying shades of blue and cyan. Some of these smaller hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and radiating lines.

3

◇ Import / Export