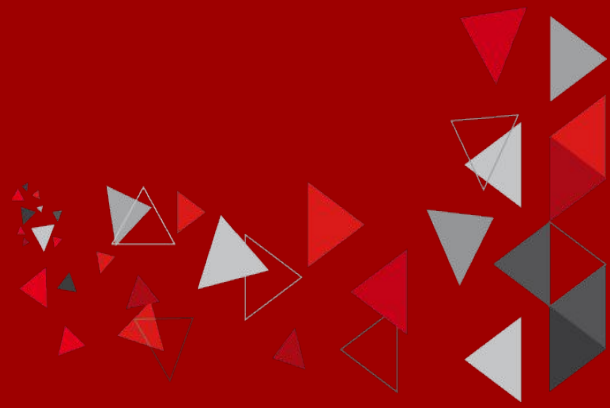
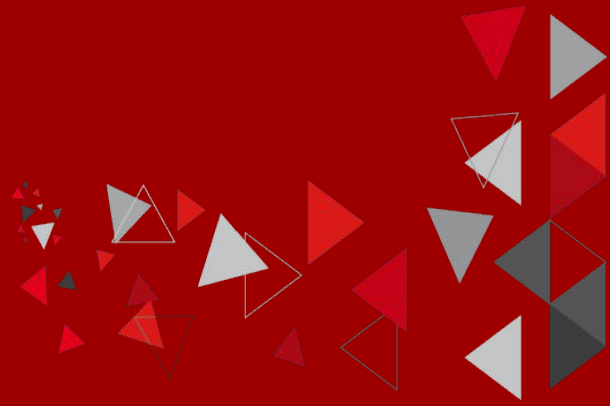
esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

Introduction à ECMAScript 6

United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia
uniTwin
EUR-ACE® Délivrée par la Commission des Titres d'Ingénieur
CONFÉRENCE DES GRANDES ÉCOLES
cdio CONCEIVE DESIGN IMPLEMENT OPERATE



- C’est quoi ECMAScript ?
- Déclaration de variable
- Objets
- Fonctions fléchées
- Tableaux
- Déstructuration
- Itérations
- Spread operator
- Fonctions asynchrones
- Export et Import

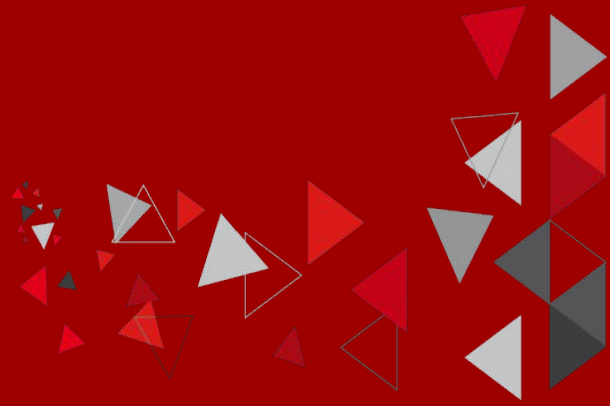


C’est quoi ECMAScript ?

C’est quoi ECMAScript ?

ECMAScript est un ensemble de normes de langage de programmation de type script normalisées par Ecma International et fait partie de la spécification ECMA-262.
Il s'agit donc d'un standard dont la spécification est implémentée dans différents langages de script tels que **JavaScript**.

ECMAScript Edition 6 (ES6) :Nommée **ES2015** et publiée en juin 2015.



Déclaration de variable

Déclaration de variable

Les variables sont utilisées comme des noms symboliques pour les valeurs. Les noms de variables sont appelés "**identificateurs**".

- **var**

Jusqu'à **Ecmascript5** la déclaration de variables se faisait en utilisant l'instruction **var** :

```
var variable ;
var variable2 = 42 ;
var variable1 ="Hello World", variable3 ,variable4 = {id:1};
```

Cette instruction nous permet de déclarer une ou plusieurs variables en lui indiquant son nom avec ou sans valeur initiale.

Déclaration de variable

Portée de la variable

Une variable déclarée à l'aide de **var** a une portée sur **tout le bloc ou elle est déclarée**.

```
function function1(){
  var variable =1 ;
  function function2() {
      console.log(variable);
      // => Resultat : 1
  }
  console.log(variable);
  // => Resultat : 1
  function2();
}
console.log(variable);
// => ERROR : 'variable' is not defined
```

Si une variable est déclarée à l'aide de var en dehors d'une fonction, elle crée un propriété à l'objet global (window dans un navigateur, global ou module dans Node.js, etc)

```
> var greeting="Hello World"
<· undefined
> window.greeting
<· 'Hello World'
```

Déclaration de variable

Principe du “ hoisting”

Le "hoisting" (hissage/remontée) c’est la façon dont Javascript gère les variables au sein du bloc ou elles sont déclarées ; en effet, peu importe la ligne à laquelle se trouve l'instruction var, la variable est déclarée et accessible partout dans le bloc ou elle est déclarée.

```
function hoisting() {
  console.log(a);  // => Resultat :  undefined
  var a = 10;
  console.log(a);  // => Resultat : 10
}

hoisting();
```

Remarque : la variable sera définie sur l'ensemble du bloc ou elle est déclarée, mais n'aura de valeur qu'après sa déclaration.

Déclaration de variable

Avant ES6 (2015), JavaScript n'avait que la portée globale et la portée des fonctions.
ES6 a introduit deux nouveaux mots-clés JavaScript importants : **let et const**.

Portée de la variable

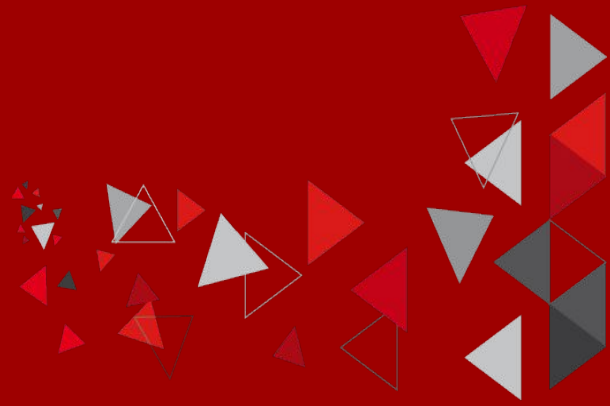
let & **const** fournissent la portée du bloc en JavaScript.
Les variables déclarées à l'intérieur d'un bloc { } ne sont pas accessibles depuis l'extérieur du bloc.

Déclaration de variable

- let & const

```
const num = 90;
console.log(num);  // => Resultat: 90
num = 77;  // Cela provoquera une erreur, car num est une constante.

let x = 10;
x = 100 ;
if (x > 5) {
  let y = 5;
  const z =20 ;
  console.log(y+z);  // => Résultat : 25
  console.log(x + y);  // => Résultat : 105
}
console.log(z); // Cela provoquera une erreur, car z n'est défini que dans le bloc if.
console.log(y);  // Cela provoquera une erreur, car y n'est défini que dans le bloc if.
```



Objets

Objets

En javascript il existe plusieurs façon pour créer des objet la plus simple est l'utilisation de la méthode **Object.create().**

```
const person = {
  nom: "John",
  prenom :"Doe",
  afficherEmail: function() {
    console.log(`Mon nom est ${this.nom} ${this.prenom}`);
  }

};
const objet = Object.create(person);
objet.nom ="Jane"
objet.prenom ="Doeeee"
objet.afficherEmail();
```

Objets

Afin de simplifier la création d'objets simples on peut utiliser **les objets littéraux.**

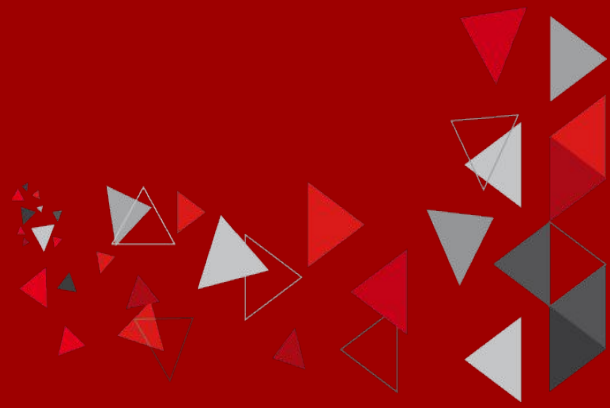
```
const person = {
  nom: "John",
  prenom :"Doe",
  afficherEmail () {
    console.log(`Mon nom est ${this.nom} ${this.prenom}`);
  }

};
console.log(person)
person.afficherEmail()
```

Objets

Fonctions

Méthodes	Rôle
Object.assign()	permet de copier les propriétés d'un ou plusieurs objets dans un objet cible
Object.entries()	retourne un tableau de tous les couples clé-valeur de l'objet en tant qu'objets [key, value]
Object.is()	permet de déterminer si deux valeurs sont strictement égales (identiques)
Object.values()	retourne un tableau qui contient toutes les valeurs de l'objet
Object.defineProperty()	permettent de définir de nouvelles propriétés sur un objet ou de modifier des propriétés existantes avec plus de contrôle et de flexibilité.
Object.defineProperties()	



Fonctions fléchées

Fonctions fléchées

Les fonctions flèches ES6 vous offrent un moyen alternatif d'écrire une syntaxe plus courte par rapport à l'expression de fonction.

```
let somme = function (x, y) {
  return x + y;
};

console.log(somme(1, 2)); // => Resultat :  3
```

=> en utilisant la fonction fléchée

```
let somme = (x, y) => x + y;

console.log(somme(1, 2));  // => Resultat :  3
```

Fonctions fléchées

Sans paramètre

()=> { // instructions }

```
let greeting =() => "hello world "
console.log(greeting())
//  => Resultat :  hello world
```

Avec un seul paramètre

(arg1)=> { // instructions }
 arg1 => { // instructions }

```
let resultat = function facto (n) {
    return n<2 ? 1: n* facto(n-1) ;
}
console.log(resultat(4))
// => Résultat : 24
```

=>

```
let factorielle = n =>  n<2 ? 1: n* factorielle (n-1)
console.log(factorielle(4))
// => Résultat : 24
```

Avec plusieurs paramètres

(arg1, arg2, ...) => {
// instructions
}

Tableaux

Tableaux

Array

JavaScript et ECMAScript fournissent plusieurs méthodes pour travailler avec des tableaux.

```
const arr = [];
const nombre = [1,2,3,4,5];
const arr2 =[ "chaine1","chaine 2"]
```

Tableaux

Array

Voici quelques exemples de méthodes couramment utilisées:

Array.prototype.concat()	Array.prototype.push()	Array.prototype.forEach()
Array.prototype.filter()	Array.prototype.reduce()	Array.prototype.join()
Array.prototype.find()	Array.prototype.reverse()	Array.prototype.slice()
Array.prototype.map()	Array.prototype.shift()	Array.prototype.splice()
Array.prototype.findIndex()	Array.prototype.sort()	Array.prototype.unshift()

Tableaux

Array

Exemple d'utilisation du **Array.prototype.map() :**

```
const users = [
  {name: 'Mohamed', age: 25},
  {name: 'Salma', age: 22},
  {name: 'Kamel', age: 16},
  {name: 'Mariem' ,age : 50 } ];
const TesteAdulte = users.map(person =>
  person.age > 18 ? console.log(`${person.name} est un adulte` )
  : console.log(`${person.name} n'est pas un adulte` ));
```

Tableaux

Collections

En Javascript, les collections sont synonyme de “Array”, Ecmascript ajoute deux nouveaux types “**Map**” et “**Set**” .

Set est une collection de valeurs uniques

```
const set = new Set();
```

Méthodes et Propriétés	Rôle
Set.prototype.size()	retourne la longueur de la Set.
Set.prototype.add(valeur)	ajoute “**valeur**” dans la Set.
Set.prototype.has(valeur)	retourne si “**valeur**” existe dans la Set.
Set.prototype.delete(valeur)	supprime “**valeur**” de la Set si elle est présente.
Set.prototype.clear()	enlève toutes les valeurs de la Set.
Set.prototype.forEach(alert(‘Hello))	execute **alert(‘Hello’)** pour chacune des valeurs de la Set.

Tableaux

Collections

Exemple d'utilisation du **Set:**

```
const set = new Set();
set.add(100);
set.add(90);
set.add(99);
set.add(11);
set.add(15);
set.add(15);
console.log(set);
set.forEach(()=> {
  if (set.size % 2 === 0) {
    console.log("Le nombre d'éléments dans le Set est pair");
} else {
  console.log("Le nombre d'éléments dans le Set est impair");
}
})
```

Tableaux

Collections

Map est une collection de paires clé-valeur, où chaque clé est unique.

Comme le Set, elle peut être itérée et manipulée à l'aide de diverses méthodes intégrées.

```
const map = new Map()
```

Méthodes et Propriétés	Rôle
Map.prototype.size()	retourne le nombre d'éléments de la Map.
Map.prototype.set(valeur)	ajoute un nouvel élément avec une clé et une valeur donnée à une Map.
Map.prototype.get(valeur)	renvoie un élément bien déterminé d'une Map
Map.prototype.has(valeur)	renvoie un booléen permettant de déterminer si l'objet Map en question contient la clé donnée.
Map.prototype.delete(valeur)	permet de retirer un élément donné d'une Map en donnant sa clé en paramètre.
Map.prototype.clear()	supprime tous les éléments d'une Map.

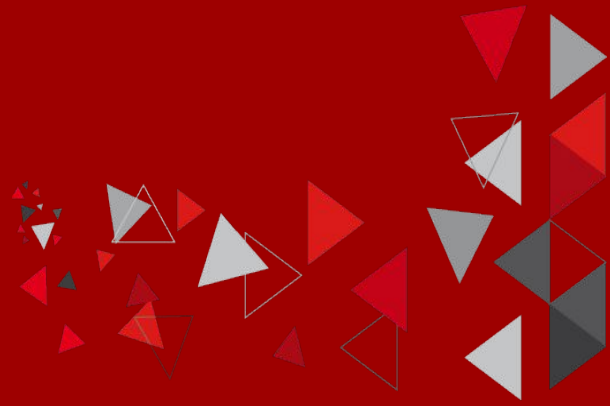
Tableaux

Collections

Exemple d'utilisation du **Map:**

```
const map = new Map([['niveau', '4'], ['classe', 'twin']]);
map.set('Nom_module', 'csa2');
map.set('Nom_chapitre', 'introduction');

console.log(map.get('niveau'));
console.log(map.get('Nom_module'));
console.log(map.get('Nom_chapitre'));
```



Déstructuration

Déstructuration

Déstructuration des tableaux

La déstructuration de tableaux est une technique qui vous permet de décomposer un tableau en variables séparées.
Il est important de respecter l'ordre des éléments lors de la déstructuration, sinon cela va causer des erreurs.

Exemple de déstructuration de tableau :

```
const users = ['Karim', 'Mohamed', 'Mariem'];

const [firstUser, secondUser, thirdUser] = users;

console.log(firstUser); // => Résultat => 'Karim'
console.log(secondUser); // => Résultat => 'Mohamed'
console.log(thirdUser); //=> Résultat => 'Mariem'
```

Déstructuration

Déstructuration des objets

La déstructuration d'objets en JavaScript permet de décomposer un objet en variables séparées. Il est important de respecter les noms des propriétés lors de la déstructuration, sinon cela va causer des erreurs.

Exemple de déstructuration d'objet avec une fonction :

```
function getUser() {
  return {
    username: 'Karim',
    email: "karim@gmail.com"

  };
}

const { username, email } = getUser();

console.log(username); //=> Résultat => 'Karim'
console.log(email); //=> Résultat => 'karim@gmail.com'
```

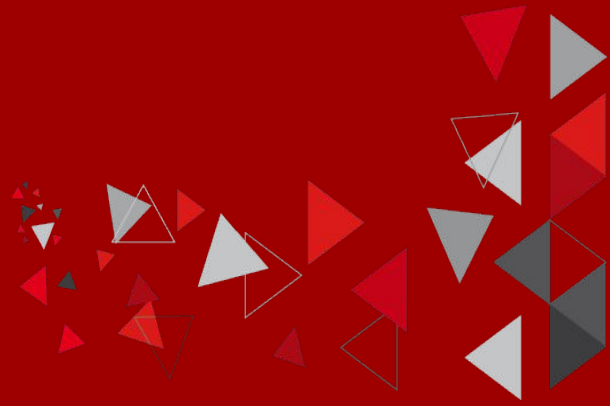
Déstructuration

Remarque:

Lorsque la déstructuration est effectuée lors de la déclaration d'une variable,il est important de noter que les parenthèses sont nécessaires.

Exemple:

```
let x = { a:14, b: 3};
let a, b;
{ a, b } = x;
({ a, b } = x);
```



Itérations

Itérations

L'itération en JavaScript est le processus de parcours et de manipulation de tous les éléments d'un tableau ou d'un objet.
Il y a plusieurs méthodes pour faire cette itération, on peut citer **for** , **for …of** et **for in .**

Exemple avec **for** :

```
const nombres = [1, "deux", "3", 4, "cinq"];

for (let i = 0; i < nombres.length; i++) {
  console.log(nombres[i]);
}
// => Résultat => 1
//deux
//3
//4
//cinq
```

Itérations

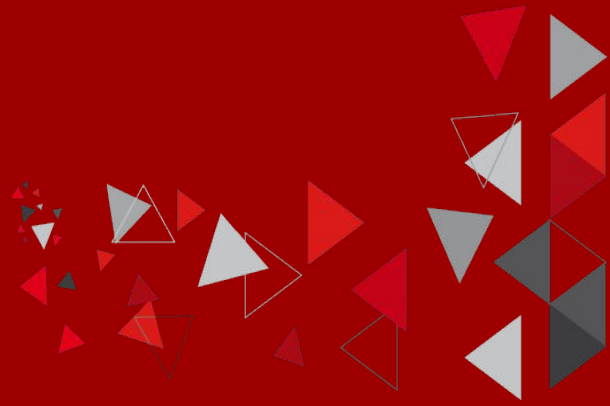
Exemple avec **for …. of** :

```
const nombres = [1, "deux", "3", 4, "cinq"];

for (const n of nombres) {
  console.log(n);
}
// => Résultat => 1
//deux
//3
//4
//cinq
```

Exemple avec **for …. in** :

```
const nombres = {"un": 1, "deux": 2, "trois": 3};
for (let n in nombres) {
  console.log(nombres[n]);
}
```



Spread operator

Spread operator

ES6 fournit un nouvel opérateur appelé opérateur d'étalement qui se compose de trois points (...). L'opérateur d'étalement vous permet d'étaler les éléments d'un objet itérable tel qu'un tableau,un objet,Map ou Set.

Exemple d'utilisation du **spread operator** avec **un objet :**

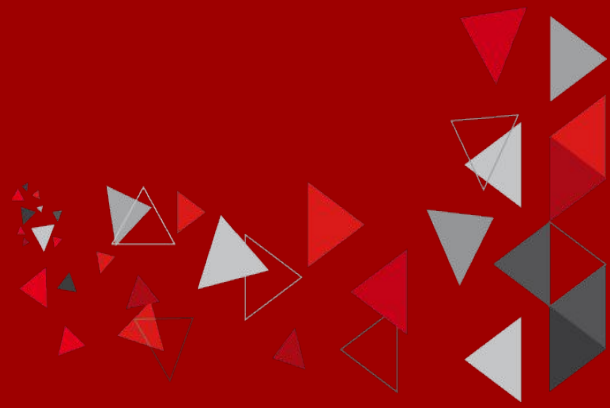
```
const classe = {
  niveau: '4TWIN',
  nombreEtudiant: 32
};
const newClasse = {
  ...classe,
  email: 'esprit2223-4WTIN@esprit.tn'
};

console.log(newClasse);
```

Spread operator

Exemple d'utilisation du **spread operator** avec **un tableau :**

```
const notes = [10, 20, 13,9,0,10];
const newNoteList = [...notes, 4, 5];
const newNoteList2 = [ 4, 5 ,...notes];
console.log(newNoteList);
console.log(newNoteList2);
```



Fonctions asynchrones

Fonctions asynchrones

Les fonctions asynchrones sont des fonctions qui permettent de gérer des opérations qui ne seront pas immédiatement exécutées.

Elles sont souvent utilisées pour effectuer des opérations de réseau, comme des requêtes HTTP, ou pour exécuter des opérations de manière asynchrone sur des données volumineuses, afin de ne pas bloquer l'interface utilisateur.

Fonctions asynchrones

async : déclarer une fonction asynchrone

```
async function getData() {
  const data = await fetch('https://.....................');
  console.log(data);
}
getData();
```

await : attendre la fin de l'exécution avant de poursuivre le code suivant

Remarque : fetch fait partie de l'API Fetch, qui a été introduite dans les navigateurs pour remplacer la méthode XMLHttpRequest utilisée auparavant pour réaliser des requêtes HTTP.

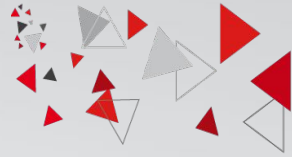
Export et Import

Export et Import

Pour exporter une variable, une fonction ou une classe, il faut la faire précéder du mot-clé “ **export** ”.
Une fois que vous avez défini un module avec des exportations, vous pouvez accéder aux variables, fonctions et classes dans un autre module en utilisant le mot-clé “ **import** ”.

Export et Import

Default Exports



```
// test.js
export default function(){
    console.log("Welcome to the Default Exports")
}
```

```
// index.js
 import fonction from "./test"
 fonction()
 // => Resultat : Welcome to the Default Exports
```

Export et Import

Named Exports

```
// test.js
export function somme(a,b){
    return a+b ;
}
export function moyenne(a,b){
    return somme(a,b)/2 ;
}
```

```
// index.js
import { somme , moyenne} from "./test"
console.log(somme(2,4))
console.log(moyenne(2,4))
// => Resultat :  6
// => Resultat :  3
```

OR

```
// index.js
import * as func  from "./test"
console.log(func.somme(2,4))
console.log(func.moyenne(2,4))
// => Resultat :  6
// => Resultat :  3
```

Export et Import

Mixed named & default exports

```
// test.js
export default function(){
    console.log("Welcome to the third Type of Exports")
}

export function somme(a,b){
    return a+b ;
}
export function moyenne(a,b){
    return somme(a,b)/2 ;
}
```

```
// index.js
import maFonction , { somme, moyenne } from "./test"
maFonction()
console.log(somme(2,4))
console.log(moyenne(2,4))
// => Resultat :  Welcome to the third Type of Exports
// => Resultat :  6
// => Resultat :  3
```

Référence

https://fr.wikipedia.org/wiki/ECMAScript
https://www.w3schools.com/js/js_es6.asp
https://www.javascripttutorial.net
https://developer.mozilla.org/fr/