PV

SOMMAIRE

Fonctions principales ES6....4

LET....4
portée limitée dans le bloc....4
portée limitée dans la condition....4
affecter une nouvelle valeur....4

CONST....5
affectation obligatoire lors de la déclaration de la variable....5

OBJET SIMPLIFIE....5
un objet qui contient un nom d'attribut identique à sa variable....5

AFFECTATION DESTRUCTUREE....5
Il y a maintenant un raccourci pour affecter des variables à partir d'objets ou de tableaux. .5
en utilisant l'objet simplifié on pourrait écrire....5
ça marche aussi avec des objets imbriqués....6
la même écriture est possible avec des tableaux....6

Paramètres optionnels et valeur par défaut....6
déclarer des paramètres optionnels dès la déclaration de la fonction....6
Ce mécanisme de valeur par défaut ne s'applique pas qu'aux paramètres de fonction, mais aussi aux valeurs de variables....6
REST OPERATOR....6
ES6 introduit une nouvelle syntaxe pour déclarer un nombre variable de paramètres dans une fonction....6
on peut extraire le premier paramètre de façon explicite et placer tous les autres dans un tableau....7
peut aussi fonctionner avec des affectations déstructurées....7

SPREAD OPERATOR....7
c'est un opérateur d'étalement....7

CLASSE....8
Il s'agit d'une des fonctionnalités les plus emblématiques dans les applications javascript. 8
une classe peut aussi avoir des attributs et des méthodes statiques....8
une classe peut avoir des accesseurs (getters, setters)....8
l'héritage est possible en ES6....9

PROMISE.....9
Les promises sont plus pratiques que les callbacks parce qu’elles permettent d’écrire du code à plat et le rendent plus simple à lire et à comprendre.....9
comment créer une promise.....10

ARROW FUNCTION.....11
ES6 introduit la nouvelle syntaxe arrow function (fonction flèchée) utilisant l’opérateur fat arrow (grosse flèche ⇒).....11

MAP.....11
L'objet Map représente un dictionnaire c'est à dire une carte de clés/valeurs.....11

SET.....11
l'objet Set (ensemble) permet de stocker des valeurs uniques, de n'importe quel type.....11
on peut aussi itérer sur une collection avec la nouvelle syntaxe for ... of.....12

TEMPLATE DE STRING.....12
les templates de string sont une nouvelle fonctionnalité mineure mais bien pratique.....12
il fournit aussi un moteur de template basique avec support du multi-ligne.....12

MODULE.....13
ES6 a créé une nouvelle syntaxe qui se charge de déclarer ce qu’on exporte depuis des modules, et ce qu’on importe dans d’autres modules.....13
dans un autre fichier on importe ce que l'on a besoin.....13

Fonctions principales ES6

Pour utiliser le code ES6 vous pouvez préparer votre code HTML en important Babel :

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.29/browser.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.29/browser-polyfill.js"></
script>
  <script type="text/babel">

    // le code ES6 ici

  </script>
  <title>ES6 avec babel</title>
</head>
<body>
</body>
</html>
```

LET

portée limitée dans le bloc

```
{
let mavariable = "je teste le mot clé let";
console.log(mavariable);
}
```

portée limitée dans la condition

```
if(true) {
let mavariable = "une variable dans une condition";
console.log(mavariable);
}
```

affecter une nouvelle valeur

let mavariable = "une valeur modifiable";

mavariable = "une autre variable";

CONST

affectation obligatoire lors de la déclaration de la variable

```
const mavariable = "une variable qui ne bouge plus"
```

> **Note:** mavariable = "une autre variable" **est impossible**

OBJET SIMPLIFIE

un objet qui contient un nom d'attribut identique à sa variable

```
let nom = "franck";
const objet = { nom };
```

> **Note:** au lieu d'écrire const objet = { nom: nom }

AFFECTATION DESTRUCTUREE

Il y a maintenant un raccourci pour affecter des variables à partir d’objets ou de tableaux

Objet

```
const httpOptions = { timeout: 2000, isCache: true };
const { timeout: httpTimeout, isCache: httpCache } = httpOptions;

console.log(httpTimeout, httpCache)
```

en utilisant l'objet simplifié on pourrait écrire

```
const httpOptions = { timeout: 2000, isCache: true };
const { timeout, isCache } = httpOptions;

console.log(timeout, isCache)
```

ça marche aussi avec des objets imbriqués

```
const httpOptions = { timeout: 2000, cache: { age: 2 } };
const { cache: { age } } = httpOptions;
```

> **Note:** on a maintenant une variable nommée age avec la valeur 2

Tableau

la même écriture est possible avec des tableaux

```
const timeouts = [1000, 2000, 3000];
const [shortTimeout, mediumTimeout] = timeouts;
```

> **Note:** on a maintenant une variable appelée shortTimeout avec la valeur 1000 et une variable nommée mediumTimeout avec la valeur 2000

Paramètres optionnels et valeur par défaut

déclarer des paramètres optionnels dès la déclaration de la fonction

```
function getBooks(size = 10, page = 1) {
  // ...
  server.get(size, page);
}
```

Ce mécanisme de valeur par défaut ne s’applique pas qu’aux paramètres de fonction, mais aussi aux valeurs de variables

```
const { timeout = 1000 } = httpOptions;
```

> **Note:** la variable timeout vaut 1000 par défaut

REST OPERATOR

ES6 introduit une nouvelle syntaxe pour déclarer un nombre variable de paramètres dans une fonction

```
const library = [];

function addBooks(...books) {
      for (let book of books) {
            library.push(book);
```

```
    }
}

addBooks('livre1', 'livre2', 'livre3', 'livre4');
console.log(library);
```

> **Note:** le paramètre books est désormais un véritable tableau sur lequel on peut itérer

on peut extraire le premier paramètre de façon explicite et placer tous les autres dans un tableau

```
const library = [];

function addBooks(param1, ...books) {
    console.log(param1);

    for (let book of books) {
        library.push(book);
    }
}

addBooks('livre1', 'livre2', 'livre3');

console.log(library);
```

> **Note:** dans ce cas param1 vaut **livre1**

peut aussi fonctionner avec des affectations déstructurées

```
const [winner, ...losers] = raceCars;
```

> **Note:** la variable raceCars est un tableau de voitures ; la variable winner contient la première voiture ; la variable losers est un tableau des autres voitures

SPREAD OPERATOR

c'est un opérateur d'étalement

```
const prices = [12, 3, 4];
const minPrice = Math.min(...prices);
console.log(minPrice);
```

> **Note:** la variable de minPrice vaut 3

CLASSE

Il s’agit d’une des fonctionnalités les plus emblématiques dans les applications javascript

```
class Book {
	constructor(page) {
		this.page = page;
	}

	toString() {
	return `le livre contient ${this.page} pages`;
	}
}
const book = new Book(6);

console.log(book.toString());
```

une classe peut aussi avoir des attributs et des méthodes statiques

```
class Car {
  static defaultSpeed() {
    return 10;
  }
}

const speed = Car.defaultSpeed();

console.log(speed);
```

une classe peut avoir des accesseurs (getters, setters)

```
class Car {
	get color() {
		console.log('get color');
		return this._color;
	}
	set color(newColor) {
    console.log(`set color ${newColor}`);
		this._color = newColor;
	}
```

```
}

const car = new Car();
car.color = 'red';
console.log(car.color);
```

> **Note:** ce code affecte la valeur red à l'attribut **color**

l’héritage est possible en ES6

```
class Animal {
  speed() {
    return 10;
  }
}

class Dog extends Animal {
  speed() {
    return super.speed() + 10;
  }
}

const dog = new Dog();
console.log(dog.speed());
```

> **Note:** la class Dog hérite et surcharge la méthode de son parent Animal grâce au mot clé **super**

PROMISE

Les promises sont plus pratiques que les callbacks parce qu’elles permettent d’écrire du code à plat et le rendent plus simple à lire et à comprendre

```
getUser(login)
.then(function (user) {
      return getRights(user);
})
.then(function (rights) {
      updateMenu(rights);
})
```

> **Note:** Les promises s’exécutent comme elles se lisent :
>
> - je veux récupérer un utilisateur

> - puis ses droits
> - puis mettre à jour le menu

Cette méthode prend deux arguments :

•un callback de succès

•un callback d’erreur

Une promise a trois états :

•pending ("en cours") : tant que la promise n’est pas réalisée, par exemple quand l’appel serveur n’est pas encore terminé

•fulfilled ("réalisée") : quand la promise s’est réalisée avec succès, par exemple quand l’appel HTTP serveur a retourné un status 200-OK

•rejected ("rejetée") : quand la promise a échoué, par exemple si l’appel HTTP serveur a retourné un status 404-NotFound

comment créer une promise

```
const getUser = function (login) {
  return new Promise(function (resolve, reject) {
    const response = 200;
    if (response === 200) {
        resolve(response);
        } else {
        reject('No user');
        }
  });
};

getUser()
.then(function(data) {
  console.log(data)
})
.catch(function(error) {
  console.log(error)
});
```

> **Note:** on peut définir une gestion d’erreur par promise ou globale à toute la chaîne

ARROW FUNCTION

ES6 introduit la nouvelle syntaxe arrow function (fonction flèchée) utilisant l’opérateur fat arrow (grosse flèche ⇒)

```
getUser(login)
.then(user => getRights(user))
.then(rights => updateMenu(rights))
.catch(error => console.log(error));
```

> **Note:** l'écriture est drôlement simplifiée le **return** est implicite s’il n’y a pas de bloc : pas besoin d’écrire **user ⇒ return getRights(user)**

MAP

L'objet Map représente un dictionnaire c'est à dire une carte de clés/valeurs

```
const cedric = {
	id: 1,
	name: 'Cedric'
};

const users = new Map();
users.set(cedric.id, cedric); // ajoute un user
console.log(users.has(cedric.id)); // true
console.log(users.size); // 1
users.delete(cedric.id); // supprimer le user
console.log(users); // []
```

SET

l'objet Set (ensemble) permet de stocker des valeurs uniques, de n'importe quel type

```
const cedric = {
	id: 1,
	name: 'Cedric'
};
```

```
const users = new Set();
users.add(cedric); // ajoute un user
console.log(users.has(cedric)); // true
console.log(users.size); // 1
users.delete(cedric); // supprimer le user
console.log(users); // []
```

on peut aussi itérer sur une collection avec la nouvelle syntaxe for ... of

```
for (let user of users) {
  console.log(user.name);
}
```

TEMPLATE DE STRING

les templates de string sont une nouvelle fonctionnalité mineure mais bien pratique

On doit utiliser des accents graves (**backticks** `) au lieu des habituelles apostrophes (**quote** ') ou apostrophes doubles (**double-quotes** ")

```
const fullname = `Miss ${firstname} ${lastname}`;
```

il fournit aussi un moteur de template basique avec support du multi-ligne

```
const template = `
      <div>
            <h1>Hello</h1>
      </div>`;

console.log(template);
```

MODULE

ES6 a créé une nouvelle syntaxe qui se charge de déclarer ce qu’on exporte depuis des modules, et ce qu’on importe dans d’autres modules

```
export function book(name, title) {
	// ...
}
export function library(ref) {
	// ...
}
```

dans un autre fichier on importe ce que l'on a besoin

```
import {book, library} from './books_service';


book('zola', 'germinal');
library('087DTF536');
```

> **Note:** on peut importer toutes les méthodes avec le joker * avec nommage import * **as** bookService **from**'./book_service';

Si le module n’expose qu’une seule fonction, ou valeur, ou classe, on n’a pas besoin d’utiliser un **named export**

On peut bénéficier de l’export par défaut, avec le mot-clé **default**

```
// book_service.js
export default class Book { }


// book.js
import Book from './book_service';
```

> **Note:** on peut mélanger l’export par défaut et l’export nommé, mais un module ne pourra avoir qu’un seul **export par défaut**