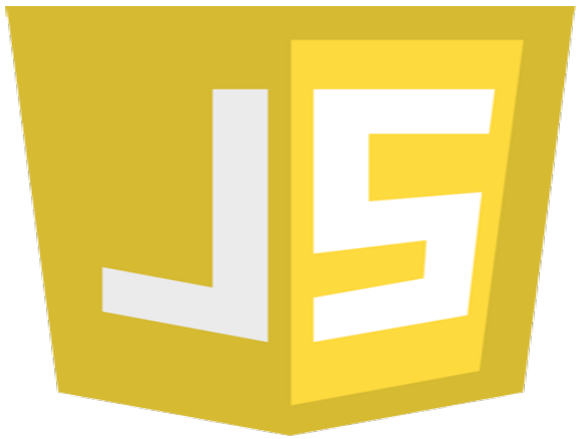
L'algorithmique avec JavaScript

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

JavaScript

ECMAScript

- ensemble de normes sur les langages de programmation de type script (JavaScript, ActionScript...)
- standardisée par Ecma International (European Computer Manufacturers Association) depuis 1994

JavaScript

ECMAScript

- ensemble de normes sur les langages de programmation de type script (JavaScript, ActionScript...)
- standardisée par Ecma International (European Computer Manufacturers Association) depuis 1994

Quelques versions

- ECMAScript version 5 (ES5) ou ES 2009 (compatible avec les navigateurs modernes)
- ECMAScript version 6 (ES6) ou ES 2015 (nécessite, selon le navigateur, un transcompilateur vers ES5)

JavaScript

Le mode strict

- permet d'éviter d'utiliser une variable non-déclarée
- s'utilise comme une chaîne de caractère `"use strict"`

JavaScript

Le mode strict

- permet d'éviter d'utiliser une variable non-déclarée
- s'utilise comme une chaîne de caractère `"use strict"`

Pour utiliser le mode strict

```
"use strict";
```

JavaScript

Le mode strict

- permet d'éviter d'utiliser une variable non-déclarée
- s'utilise comme une chaîne de caractère `"use strict"`

Pour utiliser le mode strict

```
"use strict";
```

Ceci déclenche donc une erreur

```
var x = 2;
y = 3;
```

JavaScript

Le mot-clé `let`

permet de donner une visibilité locale à une variable déclarée dans un bloc.

JavaScript

Le mot-clé `let`

permet de donner une visibilité locale à une variable déclarée dans un bloc.

Ceci génère une erreur car la variable `x` a une visibilité locale limitée au bloc `if`

```
if (5 > 2)
{
  var x = 1;
}
console.log(x);
// affiche ReferenceError: x is not defined
```

JavaScript

Les constantes

- se déclare avec le mot-clé `const`
- permet à une variable de ne pas changer de valeur

JavaScript

Les constantes

- se déclare avec le mot-clé `const`
- permet à une variable de ne pas changer de valeur

Ceci génère une erreur car une constante ne peut changer de valeur

```
const x = 5;
x = "bonjour";
// affiche TypeError: Assignment to constant
   variable.
```

JavaScript

Il est devenu possible d'attribuer une valeur par défaut pour les paramètres d'une fonction

JavaScript

Il est devenu possible d'attribuer une valeur par défaut pour les paramètres d'une fonction

```
function division(x, y = 1)
{
  return x / y;
}

console.log(division(10));
// affiche 10

console.log(division(10,2));
// affiche 5
```

JavaScript

Il est aussi possible de déclarer une fonction en utilisant les expressions Lambda (les fonctions fléchées)

```
var nomFonction = ([les arguments]) => {
  les instructions de la fonction
}
```

JavaScript

Il est aussi possible de déclarer une fonction en utilisant les expressions Lambda (les fonctions fléchées)

```
var nomFonction = ([les arguments]) => {
  les instructions de la fonction
}
```

Exemple

```
var somme = (a,b) => { return a + b; }
```

JavaScript

Il est aussi possible de déclarer une fonction en utilisant les expressions Lambda (les fonctions fléchées)

```
var nomFonction = ([les arguments]) => {
  les instructions de la fonction
}
```

Exemple

```
var somme = (a,b) => { return a + b; }
```

Appeler une fonction fléchée

```
var resultat = somme (1,3);
```

JavaScript

Les classes comme en programmation objet

- Créer des classes en utilisant la fonction constructeur
- Appliquer le principe d'encapsulation
- Faire de l'héritage
- ...

JavaScript

Pour déclarer une classe

```
class Point {
  constructor(abs, ord) {
    this.abs = abs;
    this.ord = ord;
  }
}
```

JavaScript

Pour déclarer une classe

```
class Point {
  constructor(abs, ord) {
    this.abs = abs;
    this.ord = ord;
  }
}
```

Instancier la classe $\equiv$ créer un objet de cette classe

```
var point = new Point(2,4);
console.log(point); // affiche Point { abs: 2, ord: 4 }
```

JavaScript

Pour déclarer une classe

```
class Point {
  constructor(abs, ord) {
    this.abs = abs;
    this.ord = ord;
  }
}
```

Instancier la classe ≡ créer un objet de cette classe

```
var point = new Point(2,4);
console.log(point); // affiche Point { abs: 2, ord: 4 }
```

Pour accéder à un attribut de cet objet

```
point.abs = 3;
console.log(point); // affiche Point { abs: 3, ord: 4 }
```

JavaScript

Pour rendre un attribut prive, on lui ajoute le préfixe # et on le déclare avant

```
class Point {
  #abs;
  #ord;
  constructor(abs, ord) {
    this.#abs = abs;
    this.#ord = ord;
  }
}
```

JavaScript

Pour rendre un attribut prive, on lui ajoute le préfixe # et on le déclare avant

```
class Point {
  #abs;
  #ord;
  constructor(abs, ord) {
    this.#abs = abs;
    this.#ord = ord;
  }
}
```

Ceci n'est plus possible et génère une erreur

```
point.#abs = 3;
console.log(point);
```

Tester en utilisant le navigateur (n'utilisez pas la commande node)

JavaScript

Nous pouvons définir un getter/setter pour chaque attribut qui auront le nom de l'attribut sans le préfixe #

```
class Point {
  #abs;
  #ord;
  constructor(abs, ord) {
    this.#abs = abs;
    this.#ord = ord;
  }
  get abs() {
    return this.#abs;
  }
  set abs(abs) {
    this.#abs = abs;
  }
}
```

JavaScript

Nous pouvons définir un getter/setter pour chaque attribut qui auront le nom de l'attribut sans le préfixe #

```
class Point {
  #abs;
  #ord;
  constructor(abs, ord) {
    this.#abs = abs;
    this.#ord = ord;
  }
  get abs() {
    return this.#abs;
  }
  set abs(abs) {
    this.#abs = abs;
  }
}
```

Ainsi, nous pouvons modifier et récupérer la valeur d'un attribut

```
point.abs = 3;
console.log(point.abs); // affiche 3
```

Pour l'héritage, on utilise le mot-clé `extends`

```
class Personne {
  constructor(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
  }
}

var personne = new Personne ("wick","john");
console.log(personne);
// affiche Personne {nom: "wick", prenom: "john"}

class Etudiant extends Personne {
  constructor(nom, prenom, bourse) {
    super(nom,prenom);
    // pour appeler le constructeur de la classe mère
    this.bourse = bourse;
  }
}

var etudiant = new Etudiant ("wick","john",500);
console.log(etudiant);
// affiche Etudiant {nom: "wick", prenom: "john", bourse: 500}
```

JavaScript

Supposant que l'on a une méthode qui retourne tous les détails de la classe `Personne`

```
class Personne {
  constructor(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
  }

  afficheDetails() {
    return this.prenom + " " + this.nom;
  }
}
var personne = new Personne ("wick","john");
console.log(personne.afficheDetails());
// affiche john wick
```

JavaScript

Cette méthode `afficheDetails()` peut être redéfinie dans les classes filles

```
class Etudiant extends Personne {
  constructor(nom, prenom, bourse) {
    super(nom,prenom);
    this.bourse = bourse;
  }

  afficheDetails() {
    return this.prenom + " " + this.nom + " " + this.
       bourse;
  }
}

var etudiant = new Etudiant ("wick","john",500);
console.log(etudiant.afficheDetails());
// affiche john wick 500
```

La surcharge est aussi possible

```
class Personne {
  constructor(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
  }

  afficheDetails() {
    return this.prenom + " " + this.nom;
  }

  afficheDetails(maj) {
    if (maj == false)
      return this.prenom + " " + this.nom;
    return this.prenom.toUpperCase() + " " + this.nom.toUpperCase();
  }
}

var personne = new Personne ("wick","john");
console.log(personne.afficheDetails(true));
// affiche JOHN WICK

console.log(personne.afficheDetails(false));
// affiche john wick
```

JavaScript

Il est possible de définir des attributs et/ou méthodes statiques (avec `static`)

```
class Personne {
  static nbrPersonnes = 0;
  constructor(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
    Personne.nbrPersonnes++;
  }
}
console.log(Personne.nbrPersonnes);
// affiche 0

var personne = new Personne ("wick","john");
console.log(Personne.nbrPersonnes);
// affiche 1

var personne = new Personne ("abruzzi","john");
console.log(Personne.nbrPersonnes);
// affiche 2
```

JavaScript

Considérons la classe `Personne` définie dans le fichier `personne.js`

```
class Personne {
  constructor(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
  }
};
```

JavaScript

Considérons la classe `Personne` définie dans le fichier `personne.js`

```
class Personne {
  constructor(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
  }
};
```

Pour l'utiliser dans un autre fichier, il faut

- l'exporter là où elle est définie
- l'importer là où on veut l'utiliser

JavaScript

Première méthode d'exportation

```
export class Personne {
  constructor(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
  }
};
```

JavaScript

Première méthode d'exportation

```
export class Personne {
  constructor(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
  }
};
```

Deuxième méthode d'exportation

```
class Personne {
  constructor(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
  }
};
export { Personne };
```

JavaScript

Pour l'importer et l'utiliser dans un fichier `file.js`

```
import {Personne} from "./personne.js";

var personne = new Personne ("wick","john");
console.log (personne);
// affiche Personne {nom: "wick", prenom: "john"}
```

JavaScript

Pour l'importer et l'utiliser dans un fichier `file.js`

```
import {Personne} from "./personne.js";

var personne = new Personne ("wick","john");
console.log (personne);
// affiche Personne {nom: "wick", prenom: "john"}
```

Pour pouvoir utiliser l'importation d'un autre module JS, il faut ajouter l'attribut suivant dans la page HTML

```
<script src="file.js" type="module"></script>
```

JavaScript

On peut renommer l'élément exporté

```
class Personne {
  constructor(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
  }
};
export { Personne as FirstClass };
```

JavaScript

On peut renommer l'élément exporté

```
class Personne {
  constructor(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
  }
};
export { Personne as FirstClass };
```

On peut aussi renommer l'élément importé

```
import { FirstClass as Person } from "./personne.js";

var personne = new Person ("wick","john");
console.log (personne);
// affiche Person {nom: "wick", prenom: "john"}
```

JavaScript

On peut exporter et importer

- une classe
- un objet
- une fonction
- une variable
- une constante
- ...

JavaScript

Pour connaître la compatibilité avec les navigateurs

- ES5 : `https://caniuse.com/#feat=es5`
- ES6 : `https://caniuse.com/#feat=es6`