miam | Web

Janvier–avril 2023

JUNIA
Grande école d'ingénieurs
HEI·ISEN·ISA

I.

Portée des variables

```
var v1 = 20
var myFunction = function() {
    var v2 = 10;
    console.log(v1);
    console.log(v2);
}

myFunction();
    // Affiche
        // 20
        // 10
```

Portée des variables

V1 est globale juste dans la fonction wrapper

```
function wrapper() {
    var v1 = 42;
    var myFunction = function () {
        var v2 = 13;
        console.log(v1);
        console.log(v2);
    };
    myFunction();
}
wrapper();
console.log(v1);
    // Affiche
        // 42
        // 13
        // ReferenceError: v1 is not defined
```

Portée des variables

```
function wrapper() {
    var v1 = 42;
    return function() {
        console.log(v1);
    };
}
var myFunction = wrapper();
myFunction();
console.log(v1);
// Affiche
    // 42
    // ReferenceError: v1 is not defined
```

La fonction anonyme est déclarée dans wrapper. Elle a donc accès a v1.

scope let vs var

```
function varScoping() {
  var x = 1;

  if (true) {
    var x = 2;
    console.log(x); // will print 2
  }

  console.log(x); // will print 2
}

function letScoping() {
  let x = 1;

  if (true) {
    let x = 2;
    console.log(x); // will print 2
  }

  console.log(x); // will print 1
}
```

Les closures

Nous venons d'en faire une!

```
function wrapper() {
    var closure = 42;
    var myFunction = function() {
        console.log(closure);
    };
    return myFunction;
}
var closureFunction = wrapper();
closureFunction();
// Affiche 42
```

Une closure est une fonction qui a toujours accès à son contenu (ce contenu est donc réutilisable) après la fin de son exécution.

Les closures (Exemple du counter)

Cacher une variable

```
function counterFactory() {

    var count = 0;

    return function() {
        return count++;
    }

}

var counter = counterFactory();

console.log(counter()); // 1
console.log(counter()); // 2
console.log(counter()); // 3
```

```
var count = 10;
```

```
function counterFactory() {
    let count = 0;
    return () => ++count;
}
```

Les closures (counter plus complexe)

Encapsuler un module

```
let monModule = (function() {

    let count = 0;

    return {
        inc: () => ++count,
        dec: () => --count,
        add: number => count += number,
        sub: number => count -= number
    }

})();
```

```
console.log(monModule.inc()); // 1
console.log(monModule.inc()); // 2
console.log(monModule.dec()); // 1
console.log(monModule.add(3)); // 4
console.log(monModule.sub(5)); // -1
```

Les modules de code

Les fonction internes ne sont pas exposées à l'exterieur et servent donc aux usages interne des closures

Les fonctions externes, sont accessibles depuis l'extérieur

(Petite analogie, notion de private et public dans les objets en c++)

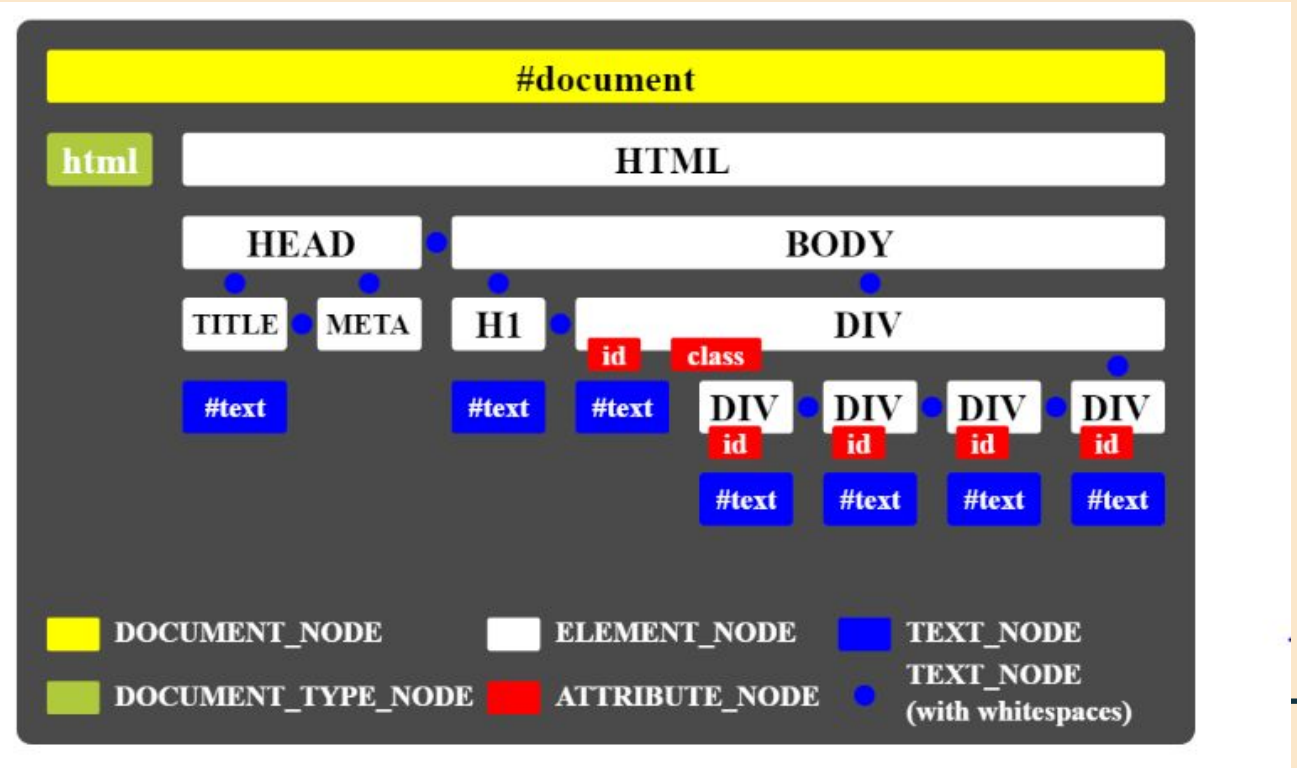
```
let monModule = (function () {
    function getModuleInterne(){
        console.log("fonction interne")
    }

    return {
        getModuleExterne(a){
            getModuleInterne();
            return "Fonction externe : "+a;
        }
    }
}) ();
```

Intérêts

- Création de « bibliothèques de fonctionnalités »
- Découpage du code en fichier ségmentés
- Code plus lisible et compréhensible
- Fonction interne qui se souvient des paramètres qu'elle stocke
- Les variables à l'intérieur sont protégées et modifiées uniquement avec les fonction disponibles de la closure

Structure d'un document HTML



Structure d'un document HTML

Ce qu'il faut retenir :

Le document HTML est un ensemble d'objets de type Node

Organisé sous la forme d'un arbre (Parents, Enfants, Frères)

Un Nœud possède un type, des propriétés, etc

La racine de tout: l'objet Document

Rechercher un Noeud

Références pour les objets Document et Element :

https://developer.mozilla.org/en-US/docs/Web/API/Document
https://developer.mozilla.org/en-US/docs/Web/API/Element

- getElementById(idName)
- getElementsByTagName(tagName)
- getElementsByClassName(className)
- querySelector(selector)
- querySelectorAll(selector)

Rechercher un Noeud

```html
<html>
    <head></head>
    <body>
        <ul id="first" class="list">
            <li>1 One</li>
            <li>1 Two</li>
        </ul>
        <ul id="second" class="list">
            <li>2 One</li>
            <li>2 Two</li>
        </ul>
        <div id="third">Content</div>

    </body>
    <script src="./cours2.js"></script>
</html>
```

```
document.getElementById('second');
document.getElementsTagName('li');
document.getElementsClassName('list');
document.querySelector('ul.list');
document.querySelectorAll('ul.list');
```

```
let ul = document.getElementById('second');
ul.getElementsByTagName('li');
```

Parcourir les Nœuds (lecture seule)

- Node.parentElement
- Element.firstElementChild
- ParentNode.lastElementChild
- ParentNode.children

- NonDocumentTypeChildNode.nextElementSibling
- NonDocumentTypeChildNode,previousElementSibling

Parcourir les Nœuds (lecture seule)

```html
<html>
    <head></head>
    <body>
        <ul id="first" class="list">
            <li>1 One</li>
            <li>1 Two</li>
        </ul>
        <ul id="second" class="list">
            <li>2 One</li>
            <li>2 Two</li>
        </ul>
        <div id="third">Content</div>

    </body>
    <script src="./cours2.js"></script>
</html>
```

```js
let ul = document.getElementById('first');
ul.parentElement;
ul.firstElementChild;
ul.children;
ul.nextElementSibling;
ul.nextElementSibling.nextElementSibling;
```

Manipuler leurs attributs

- Element.id
- Element.classList
  - Add()
  - Remove()
- Element.hasAttribute(attrName)
- Element.getAttributeNames()
- Element.getAttribute(attrName)
- Element.removeAttribute(attrName)
- Element.setAttribute(attrName, value)

Manipuler leurs attributs

```html
<html>
    <head></head>
    <body>
        <ul id="first" class="list">
            <li>1 One</li>
            <li>1 Two</li>
        </ul>
        <ul id="second" class="list">
            <li>2 One</li>
            <li>2 Two</li>
        </ul>
        <div id="third">Content</div>

    </body>
    <script src="./cours2.js"></script>
</html>
```

```js
let ul = document.getElementById('second');
ul.id;                          // second
console.log(ul.classList);                  // DOMTokenList["list"]
ul.getAttributeNames();      // ["id","class"]
ul.getAttribute('class');    // "list"
ul.removeAttribute('class');
ul.setAttribute('class', 'list render');
ul.classList.toggle('render');
ul.classList.toggle('render');
console.log(ul.classList);
```

Ajouter et Supprimer des nœuds

- document.createElement(tagName)
- Element.remove()
- Node.cloneNode()
- Node.appendChild(node)
- Node.insertBefore(node, child)
- Node.removeChild(node)
- Node.replaceChild(node, child)
- Node.contains(node)

Si l'élément ne possède pas d'enfants :
Element.innerHTML

Les bonnes pratiques

- De moins en moins de variables globales
- On favorise le Let par rapport au Var : meilleures performances pour une même utilisation
- On découpe son code en plusieurs fonctions réutilisables
- On commente chaque fonction : à quoi sert-elle, que prend-elle en argument, quels éléments sont retournés
- On essaie de classer les fonctions, et on les mets dans différents fichiers JS appropriés

Le modèle MVC

- Bonne pratique de programmation, sépare votre application en trois parties
  - Modèle (les données)
  - La vue
  - Le contrôleur

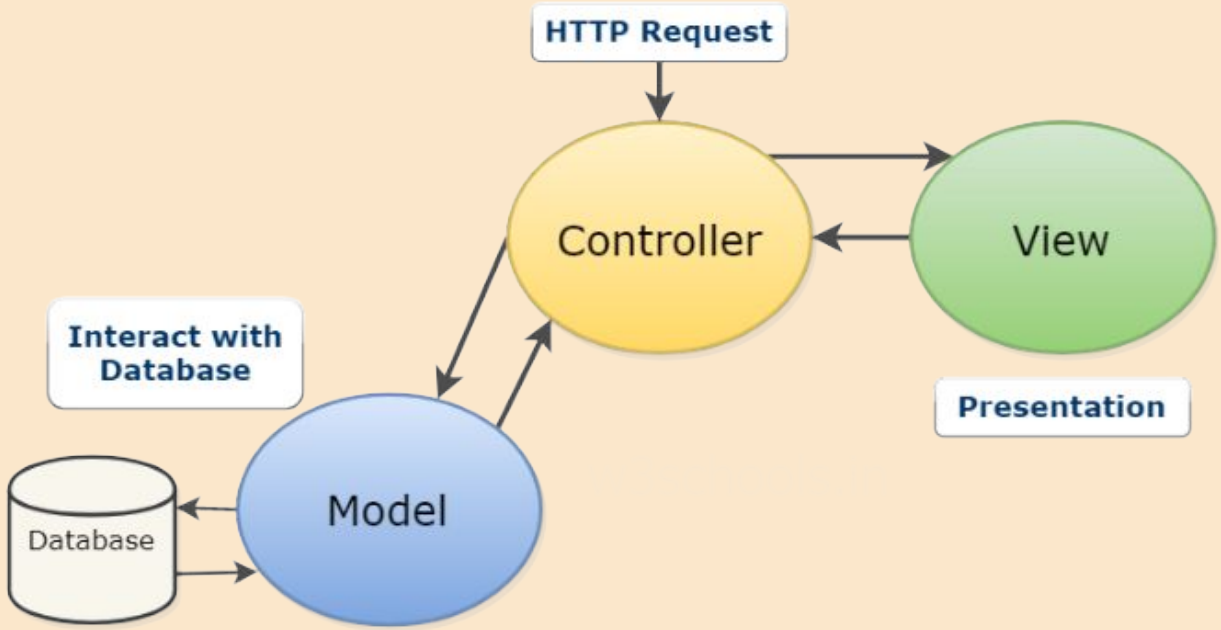
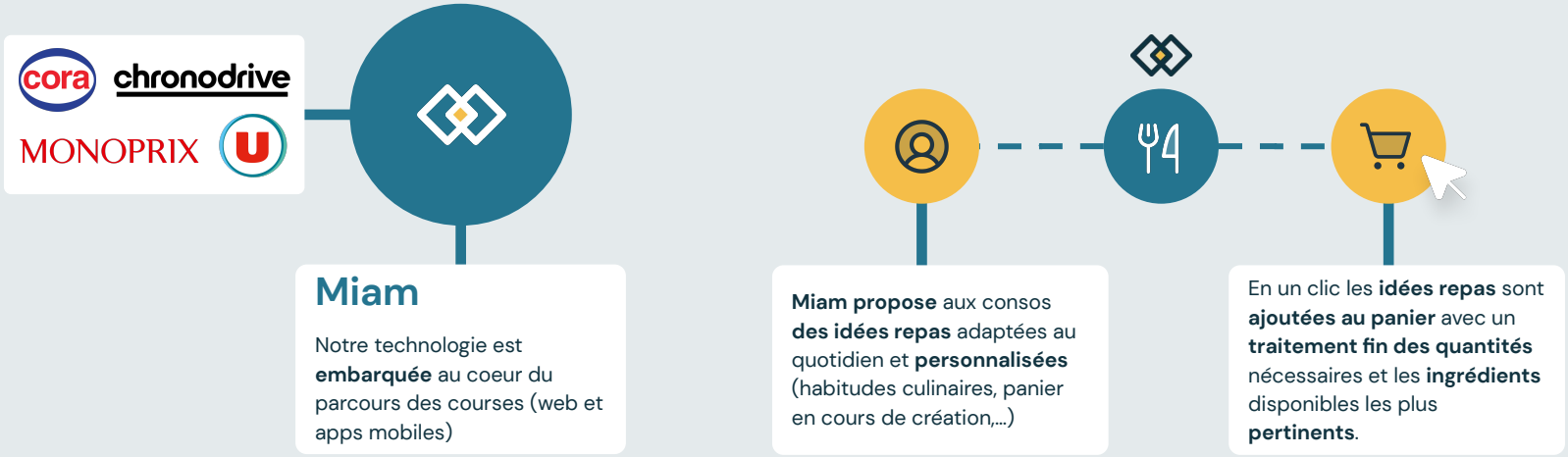


Fig: MVC Architecture

A propos de Miam

Miam réinvente et simplifie les courses alimentaires.



Miam

Notre technologie est **embarquée** au coeur du parcours des courses (web et apps mobiles)

Miam propose aux consos **des idées repas** adaptées au quotidien et **personnalisées** (habitudes culinaires, panier en cours de création,...)

En un clic les **idées repas** sont **ajoutées au panier** avec un **traitement fin des quantités** nécessaires et les **ingrédients** disponibles les plus **pertinents**.

Qui sommes-nous ? Une startup tech lilloise, qui développe depuis 2019 des solutions d'intelligence artificielle à destination de la grande distribution.

Plus de détails sur https://miam.tech