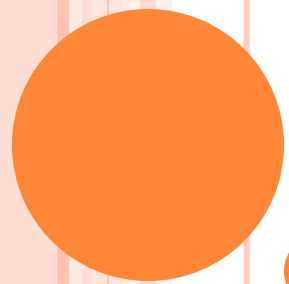


CHAPITRE 4 : LES MODULES, LES CLASSES ET LES WEB COMPONENTS

PLAN DU CHAPITRE

- Partie 1 : Les modules
- Partie 2 : Les classes
- Partie 3 : Les composants Web



PARTIE 1

Les modules

INTRODUCTION - MODULES

- Initialement les applications web développées étaient simples et non complexes
 - Les scripts étaient petits et simples
 - JavaScript n'était pas un langage modulaire puisqu'il n'était donc pas nécessaire.
- Les applications sont devenues de plus en plus complètes et donc complexes
 - Les scripts sont devenus de plus en plus complexes
 - Besoin de scinder un script complexe en plusieurs “modules”
- En 2015, **JavaScript est** devenu un langage **modulaire**

QU'EST-CE QU'UN MODULE ?

- Un module est un fichier.
 - Un script est un module.
- Spécificité : Les modules peuvent échanger des données : des fonctions, des variables, ...
- Utiliser des directives spéciales : **export** et **import**
 - **export** : labelise les données qui doivent être accessibles depuis l'extérieur du module actuel.
 - **import** : permet l'importation des données à partir d'autres modules.

QU'EST-CE QU'UN MODULE ?

- Exemple

calcul.js

```
export const operationName1 = 'addition';  
const operationName2 = 'multiplication';  
  
export function add(nb1,nb2)  
{  
    retrun nb1+nb2;  
}  
  
function multiply(nb1,nb2)  
{  
    retrun nb1*nb2;  
}
```

app.js

```
import {operationName1,add} from './calcul.js ;
```

QU'EST-CE QU'UN MODULE ?

Les modules fonctionnent uniquement via **HTTP(s)**, pas localement à partir des fichiers systèmes



Il faut tester les modules via un **serveur Web**

CARACTÉRISTIQUES D'UN MODULE

- Exécution en mode **strict**
 - Tout ce qu'on déclare dans le module sera par défaut local au module
 - Pour rendre visible une variable ou une fonction, il faut l'exporter
- Extension : .js ou .mjs
 - .js : rien ne spécifie que le fichier contient le code d'un module
 - utiliser l'attribut **type='module'** qui informe le navigateur qu'il est en train de charger le code basé sur la notion de module

CHARGER / DÉCLARER UN MODULE DANS HTML

- Pour charger un module JavaScript dans un fichier HTML en utilise l'attribut `type='module'` dans notre élément `script`.
- Exemple :

```
<html lang="fr">
<head>
  <meta charset="UTF-8">
</head>
<body>
  <script type='module' src='./js/calcul.js'></script>
</body>
</html>
```

CHARGER / DÉCLARER UN MODULE DANS HTML

- Pour déclarer un module JavaScript dans un fichier HTML en utilise l'attribut **type='module'** dans notre élément **script**.
(déconseillé)
- Exemple

```
<html lang="fr">
<head>
  <meta charset="UTF-8">
</head>
<body>
  <script type="module">
    export const operationName1 = 'addition';
    const operationName2 = 'multiplication';

    export function add(nb1,nb2)
    {
      retrun nb1+nb2;
    }

    function multiply(nb1,nb2)
    {
      retrun nb1*nb2;
    }
  </script>
</body>
</html>
```

CHARGER / DÉCLARER UN MODULE DANS HTML

- Les modules associés à un document HTML ne bloquent pas l'analyse du code HTML qui le suit
- Les modules attendent que le document HTML soit complètement chargé pour s'exécuter
- L'ordre des scripts est respecté : le premier module inséré s'exécutera en premier et etc.

EXPORTER / IMPORTER LES MODULES

- **Exportation / Importation groupée** : Il s'agit de regrouper toutes les exportations en fin de fichier

Exemple : fichier moduleExport1.js

```
var nb = 0;
var step = 1;
function setStep (val){
  step = val;
}
function increment (nb){
  nb += step;
  return nb;
}
function decrement (nb){
  nb -= step;
  return nb;
}
export {increment, decrement, nb, setStep};
```

Exportation groupée

EXPORTER / IMPORTER LES MODULES

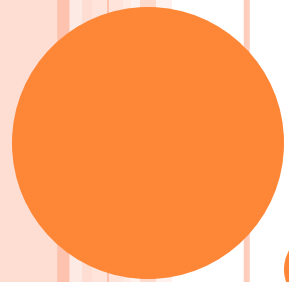
- Exportation / Importation groupée

Exemple : fichier moduleImport.js

```
import { increment, decrement, nb, setStep } from "../moduleExport1.js";  
  
let valInit = nb;  
setStep(2);  
let valDec = decrement(valInit);  
let valInc = increment(valInit);  
  
console.log("La valeur initiale du nombre est : " + nb);  
console.log("La valeur décrétementée du nombre est : " + valDec);  
console.log("La valeur incrémentée du nombre est : " + valInc);
```

Importation





PARTIE 2

Les classes

SYNTAXE

```
class MyClass {
  property1 = value;    // initialized public field
  property2 ;          // public field
  #property3 ;         // private field

  // constructor
  constructor(p1,p2,p3) {

    this.property1 = p1; // pour accéder à un attribut, il faut utiliser this
    this.property2 = p2;
    this.#property3 = p3;
    //instruction
    console.log("Je suis le constructeur de classe");
  }
  method1(...) {}      // method
  get getterMethod(...) {} // getter method
  set setterMethod(...) {} // setter method
}
```

USAGE DES CLASSES

```
//obj instance créée en utilisant le constructeur  
let obj = new MyClass(val1,val2,val3);  
  
obj.property1; // accès à un attribut public  
obj.#property3; // Erreur - accès à un attribut privé  
obj.method1(); // appel de méthode
```


EXEMPLE

Fichier app.js

```
//déclaration de la classe User
class User {

    constructor (vName){
        this.name = vName;
    }
    sayHi () {
        alert("Hello " + this.name);
    }
}

//usage
let user = new User("student");
user.sayHi();
```

Fichier index.html

```
<html lang="fr">
<head/>
<body>
    <script src='./js/app.js'></script>
</body>
</html>
```

Résultat d'exécution



EXEMPLE AVEC MODULE

Fichier module.js exportant une classe


```
export class User {  
  
  constructor (vName){  
    this.name = vName;  
  }  
  sayHi() {  
    alert("Hello " + this.name);  
  }  
}
```

Fichier script principal app.js

```
import {User} from "./module.js"  
  
let user = new User("Student");  
user.sayHi();
```

Fichier index.html

```
<html lang="fr">  
<head/>  
<body>  
  <script src='./js/app.js'></script>  
</body>  
</html>
```



```
<html>  
<head/>  
<body>  
  <script src='./js/app.js' type="module"></script>  
</body>  
</html>
```



Cette page indique

Hello student

OK

CLASSES - EXTENSION

```
class SubClass extends MyClass {  
    //déclaration d'autres propriétés (optionnelle)  
    property4 ;  
    //...  
    //constructeur (optionnel)  
    constructor(v1,v2,v3,v4){  
        // Si le constructeur de la classe fille est présent,  
        // il faut obligatoirement appeler le constructeur de la classe mère avec super  
        super(v1,v2,v3) ;  
        // déclaration d'autres instructions (optionnelle)  
        this.property4 = v4;  
        //...  
    }  
    //déclaration d'autres méthodes (optionnelle)  
    methode2 () {...}  
}
```

EXEMPLE

fichier *module.js* exportant une classe fille

```
class User {  
  
  constructor (vName){  
    this.name = vName;  
  }  
  sayHi() {  
    alert("Hello " + this.name);  
  }  
}  
  
export class Admin extends User {  
  
  constructor(v1,v2){  
    super(v1);  
    this.role = v2;  
  }  
  //déclaration d'autres méthodes (optionnelle)  
  get role(){    //getter  
    return this._role;  
  }  
  set role(v){   //setter  
    this._role=v;  
  }  
}
```

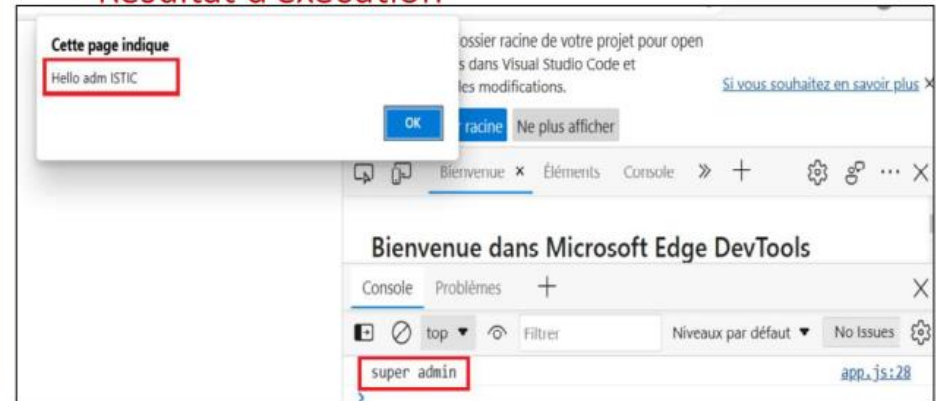
fichier *app.js*

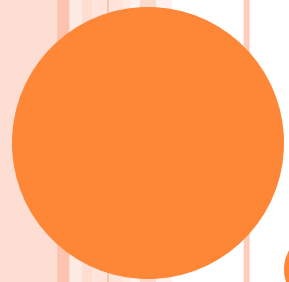
```
import {Admin} from "../module.js"  
  
let user = new Admin("adm ISTIC","super admin");  
console.log(user.role);  
user.sayHi();
```

fichier *index.html*

```
<html>  
<head/>  
<body>  
  <script src="../js/app.js" type="module"></script>  
</body>  
</html>
```

Résultat d'exécution





PARTIE 3

Les composants Web

QU'EST CE QUE WEB COMPONENTS ?

- Les Web Components désignent des **blocs de code** qui englobent la structure interne des éléments HTML incluant du JS et du CSS.
- Ils permettent de créer de nouvelles balises/éléments HTML personnalisés (des widgets, des boutons et diverses fonctions) à l'aide de JavaScript qui peuvent être **réutilisables au sein de différentes pages ou applications Web**.
- Il contient 3 parties :
 - Définition de l'élément (HTML)
 - Un script JavaScript pour définir le comportement de l'élément
 - Le style CSS qui définit l'apparence de l'élément
- La norme des composants Web est élaborée en 2012 par le consortium W3C.

POURQUOI UTILISER LES WEB COMPONENTS?

- Certaines bibliothèques ou frameworks comme Angular ou jQuery font partie des outils principaux des développeurs Web.
 - Les squelettes de code permettant d'économiser beaucoup de travail lors du développement des projets, mais lorsqu'ils sont utilisés entre différents projets, les développeurs doivent souvent réécrire ou modifier le code si un changement de framework est prévu.
 - C'est la raison pour laquelle le W3C a introduit les Web Components → Nouveau cadre pour une réutilisation simple et globale des codes HTML, CSS et JavaScript
- Économiser la charge de travail
- Garantir un gain de temps

BREF HISTORIQUE DES WEB COMPONENTS?

- Le concept de composants web standard a été présenté pour la première fois par Alex Russell en 2011.
- La bibliothèque Polymer de Google est arrivée deux ans plus tard
- Mais les premières implémentations ne sont apparues dans Chrome et Safari qu'en 2016 : les fournisseurs de navigateurs ont mis du temps à négocier les détails
- Les composants web ont été ajoutés à Firefox en 2018 et à Edge en 2020 (lorsque Microsoft a adopté le moteur Chromium)
- Les composants web **fonctionnent dans tous les frameworks JavaScript**

LES ÉLÉMENTS DES WEB COMPONENTS

Le modèle de composants Web prévoit à la base les quatre spécifications suivantes pour la création des composants HTML :

1. **Custom Elements** : ensemble d'API JavaScript pour la définition des éléments définis par l'utilisateur
2. **Shadow DOM** : ensemble d'API JavaScript permettant de créer un DOM interne au composant, inaccessible aux autres
3. **HTML Templates** : des modèles de balisage qui ne sont pas présentés sur la page affichée et peuvent être utilisés pour servir de base aux éléments définis par l'utilisateur
4. **ES Modules** : des modules permettant l'intégration et la réutilisation des documents JavaScript

COMMENT UTILISER LES CUSTOM ELEMENTS ?

- Les Custom Elements (éléments définis par l'utilisateur) sont des balises **HTML** englobant le contenu du HTML y compris les instructions et les scripts CSS.
- Le nom d'un élément doit contenir un tiret pour ne jamais entrer en conflit avec des éléments officiellement pris en charge dans la spécification HTML
 - Exemple : `<hello-world></hello-world>`
- Pour créer un élément personnalisé, il faut donner quelques détails au navigateur : Comment le montrer, que faire lorsque cet élément est chargé dans le DOM, etc.
 - Techniquement : c'est possible en créant une **classe** avec des **méthodes spéciales**.

COMMENT UTILISER LES LES CUSTOM ELEMENTS ?

```
JS
1 // web component
2 class HelloWorld extends HTMLElement {
3
4   // connect component
5   connectedCallback() {
6     this.textContent = 'Hello World!';
7   }
8
9 }
10
11 // register component
12 customElements.define( 'hello-world', HelloWorld );
```

Associer l'élément HTML **hello-world** à la classe **HelloWorld**

La classe doit étendre l'interface **HTMLElement**, qui représente les propriétés et méthodes par défaut de chaque élément HTML.

La classe nécessite une méthode nommée **connectedCallback()** qui est invoquée lorsque l'élément est ajouté à un document

COMMENT UTILISER LES LES CUSTOM ELEMENTS ?

- Lorsque le JavaScript est chargé (`<script type="module" src="./hw_component.js"></script>`), le navigateur associe l'élément `<hello-world>` à la classe **HelloWorld**
- À ce niveau, on a un élément personnalisé
- Ce composant peut être stylé en CSS comme n'importe quel autre élément

```
HTML
1 <p>A new Web Component...</p>
2
3 <hello-world></hello-world>
```

A new Web Component...

Hello World!

```
CSS
1 body {
2   font-family: sans-serif;
3 }
4
5 hello-world {
6   font-weight: bold;
7   color: red;
8 }
```

MÉTHODES DE CYCLE DE VIE DE L'ÉTAT D'UN CUSTOM ELEMENT

```
class MyElement extends HTMLElement {  
  constructor() {  
    super();  
    // créer l'élément  
  }  
  
  connectedCallback() {  
    // le navigateur appelle cette méthode lorsque l'élément est ajouté au document  
    // elle peut-être appelé autant de fois que l'élément est ajouté  
  }  
  
  disconnectedCallback() {  
    // le navigateur appelle cette méthode lorsque l'élément est supprimé du document  
    // elle peut-être appelé autant de fois que l'élément est supprimé  
  }  
  
  static get observedAttributes() {  
    return [/* tableau listant les attributs dont les changements sont à surveiller */];  
  }  
  
  attributeChangedCallback(name, oldValue, newValue) {  
    // appelé lorsque l'un des attributs listé par la méthode observedAttributes() est modifié  
  }  
  
  adoptedCallback() {  
    // méthode appelé lorsque l'élément est envoyé vers un nouveau document  
    // (utilisé très rarement avec document.adoptNode)  
  }  
  
  // vous pouvez ajouter d'autres méthodes ou propriétés  
}
```

SHADOW DOM

Le modèle de composants Web prévoit à la base les quatre spécifications suivantes pour la création des composants HTML :

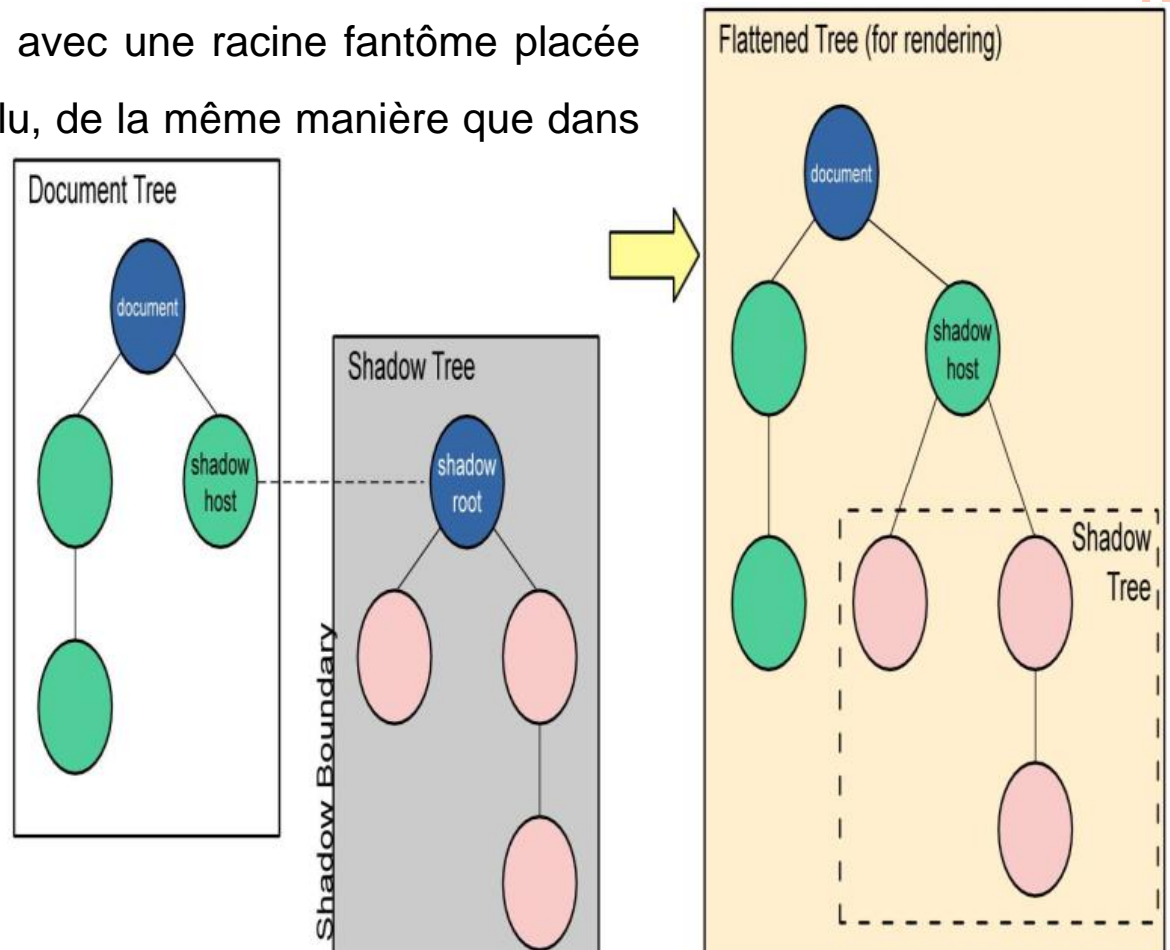
1. **Custom Elements** : ensemble d'API JavaScript pour la définition des éléments définis par l'utilisateur
2. **Shadow DOM** : ensemble d'API JavaScript permettant de créer un DOM interne au composant, inaccessible aux autres
3. **HTML Templates** : des modèles de balisage qui ne sont pas présentés sur la page affichée et peuvent être utilisés pour servir de base aux éléments définis par l'utilisateur
4. **ES Modules** : des modules permettant l'intégration et la réutilisation des documents JavaScript

SHADOW DOM

Le DOM fantôme (Shadow DOM) permet à des arbres DOM cachés d'être associés à des éléments de l'arbre DOM principal

Cet arbre DOM fantôme s'ouvre avec une racine fantôme placée sous n'importe quel élément voulu, de la même manière que dans le DOM normal.

Les nœuds du DOM fantôme sont manipulés exactement de la même manière que les nœuds du DOM principal



SHADOW DOM

Techniquement :

- Pour associer une racine fantôme à un élément, on utilise la méthode `Element.attachShadow()` qui prend en paramètres un objet d'options contenant une option — `mode` — ayant pour valeur `open` (ouvert) ou `closed` (fermé)

```
const shadow = Element.attachShadow({ mode : 'closed' });
```

- Le mode peut être soit :
 - « `open` » – Le JavaScript dans la page externe peut accéder au Shadow DOM (en utilisant `Element.shadowRoot`), ou
 - « `closed` » – le Shadow DOM n'est accessible que dans le composant web (`element.shadowRoot` est toujours `null`).

SHADOW DOM

- Cette technologie optimise le CSS et le DOM et permet d'intégrer des styles scopés dans la plateforme web (définir un style pour un composant dans le shadow DOM).
- **CSS scopé** : la portée du CSS intégré au DOM fantôme est limitée à ce dernier. Les règles de style y restent donc circonscrites tandis que les styles de page ne l'impactent pas.
- DOM isolé : le DOM d'un composant est autonome (par exemple, `document.querySelector()` ne renverra pas les nœuds dans le DOM fantôme des composants).

SHADOW DOM

- Le composant web `<hello-world>` que nous avons déjà construit fonctionne. Mais, il n'est pas à l'abri des interférences extérieures, et CSS ou JavaScript pourraient le modifier.
- Le Shadow DOM résout ce problème d'encapsulation en **attachant un DOM séparé au composant web** avec :

```
const shadow = this.attachShadow({ mode : 'closed' });
```

SHADOW DOM

- Le Shadow DOM peut être manipulé comme n'importe quel autre élément DOM
- Le composant rend maintenant le texte « Hello » à l'intérieur d'un élément <p> et lui donne un style.
- Il ne peut pas être modifié par JavaScript ou CSS en dehors du composant

```
18 // connect component
19 connectedCallback() {
20
21     const shadow = this.attachShadow({ mode:
22         'closed' });
23
24     shadow.innerHTML = `
25         <style>
26             p {
27                 text-align: center;
28                 font-weight: normal;
29                 padding: 1em;
30                 margin: 0 0 2em 0;
31                 background-color: #eee;
32                 border: 1px solid #666;
33             }
34         </style>
35
36         <p>Hello ${ this.name }!</p>`;
37     }
38 }
39 // register component
40 customElements.define( 'hello-world', HelloWorld );
```

A new Web Component...

Hello Ali!

MODÈLES / TEMPLATES HTML

Le modèle de composants Web prévoit à la base les quatre spécifications suivantes pour la création des composants HTML :

1. **Custom Elements** : ensemble d'API JavaScript pour la définition des éléments définis par l'utilisateur
2. **Shadow DOM** : ensemble d'API JavaScript permettant de créer un DOM interne au composant, inaccessible aux autres
3. **HTML Templates** : des modèles de balisage qui ne sont pas présentés sur la page affichée et peuvent être utilisés pour servir de base aux éléments définis par l'utilisateur
4. **ES Modules** : des modules permettant l'intégration et la réutilisation des documents JavaScript

MODÈLES / TEMPLATES HTML

- Un HTML Template est un modèle pour les fichiers HTML.
- Les éléments contenus restent inactifs et non affichés sur la page Web jusqu'à ce qu'ils soient explicitement appelés.
- Ils peuvent être réutilisés de nombreuses fois car ils constituent la **structure de base d'un élément personnalisé**.
- Grâce à cette propriété, ils n'ont aucun effet négatif sur le temps de chargement d'un site Internet.
- La balise <template> permet de définir un modèle HTML. Ex :

```
<template id="mon-element">
```

```
<p>Mon élément</p>
```

```
</template>
```

MODÈLES / TEMPLATES HTML

- On attribut un ID au modèle pour pouvoir le référencer dans la classe du composant.
- Dans cet exemple, trois paragraphes pour afficher le message « Hello »

HTML

```
1 <p>A new Web Component...</p>
2
3 <hello-world name="Ali"></hello-world>
4
5 <template id="hello-world">
6   <style >
7     .hw-text {
8       text-align: center;
9       font-weight: normal;
10      padding: 0.5em;
11      margin: 1px 0;
12      background-color: #eee;
13      border: 1px solid #666;
14    }
15  </style>
16  <p class="hw-text"></p>
17  <p class="hw-text"></p>
18  <p class="hw-text"></p>
19 </template>
```

MODÈLES / TEMPLATES HTML

- Tant que le « template » n'est pas instancié, il ne produira aucun effet, ni ne sera visible.
- Pour utiliser notre template, nous aurons besoin du JavaScript
- La classe de composant web peut accéder à ce modèle, obtenir son contenu et **cloner** les éléments pour s'assurer que nous créons un fragment DOM unique partout où il est utilisé :
- ```
const template = document.getElementById('hello-world').content.
cloneNode(true),
```

# MODÈLES / TEMPLATES HTML

- Modification du Shadow DOM

```
// connect component
connectedCallback() {

 const
 shadow = this.attachShadow({ mode: 'closed' }),
 template = document.getElementById('hello-world').content.cloneNode(true),
 hwMsg = `Hello ${ this.name }`;

 Array.from(template.querySelectorAll('.hw-text'))
 .forEach(n => n.textContent = hwMsg);

 shadow.append(template);
}
```



# ES MODULES

Le modèle de composants Web prévoit à la base les quatre spécifications suivantes pour la création des composants HTML :

1. **Custom Elements** : ensemble d'API JavaScript pour la définition des éléments définis par l'utilisateur
2. **Shadow DOM** : ensemble d'API JavaScript permettant de créer un DOM interne au composant, inaccessible aux autres
3. **HTML Templates** : des modèles de balisage qui ne sont pas présentés sur la page affichée et peuvent être utilisés pour servir de base aux éléments définis par l'utilisateur
4. **ES Modules** : des modules permettant l'intégration et la réutilisation des documents JavaScript

# ES MODULES

- La spécification ES Module définit l'inclusion et la réutilisation de documents JS dans d'autres documents JS.
- Les modules ES permettent de développer des composants Web de manière modulaire, conformément aux autres implémentations acceptées par l'industrie pour le développement d'applications JavaScript.
- On peut définir l'interface d'un élément personnalisé dans un fichier JS qui est ensuite inclus avec un attribut `type="module"`.
- Supposons qu'un élément soit défini dans `my_component.js`, on l'importe à l'aide du script suivant :

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

# WEB COMPONENTS – LES ATTRIBURS

- **Element.setAttribute()** pour définir la valeur d'un attribut
- **Element.getAttribute()** pour récupérer la valeur d'un attribut
- **Element.hasAttribute()** pour vérifier si un attribut existe sur l'élément
- **Element.removeAttribute()** pour supprimer un attribut de l'élément

# ÉTAPES D'IMPLEMENTATION D'UN WEB COMPONENT

- Gréer une **classe JavaScript** qui spécifie la fonctionnalité du composant web
- Déclarer le nouveau Custom Element avec la méthode `CustomElements.define()`.
- Si nécessaire et souhaité, relier un Shadow DOM caché à l'élément personnalisé (à l'aide de la méthode `Element.attachShadow()`)
  - ajouter des éléments enfants, des écouteurs d'événement, etc. au DOM fantôme en utilisant les méthodes DOM courantes...
- Si nécessaire (si la structure de balises se répète sur une page web), définir un modèle HTML avec la balise `<template>`.
  - Utiliser de nouveau les méthodes DOM courantes pour cloner le modèle et le relier à votre DOM fantôme.
- Utiliser le Custom Element généré dans votre site comme un **élément HTML** habituel.

# WEB COMPONENTS – EXEMPLE

The image displays a web development environment with three main components:

- Code Editor (Left):** Shows the `module.js` file with the following code:

```
export class ShowHello extends HTMLElement {
 connectedCallback() {
 this.setAttribute('name', 'ISTIC');
 let shadow = this.attachShadow({mode: 'open'});
 shadow.innerHTML = `<p> Hello, ${this.getAttribute('name')}</p>`;
 shadow.addEventListener("click", function(){alert("I'm a web component")});
 }
}
```
- Code Editor (Top Right):** Shows the `app.js` file with the following code:

```
import {ShowHello} from "./module.js"
customElements.define('show-hello', ShowHello);
```
- Code Editor (Bottom Left):** Shows the `index.html` file with the following code:

```
<html>
<head/>
<body>
 <script src="./js/app.js" type="module"></script>
 <show-hello ></show-hello>
</body>
</html>
```
- Browser Preview (Bottom Right):** Shows the rendered output in a browser at `127.0.0.1:5500/WebComponent/index.html`. The page displays "Hello, ISTIC". A confirmation dialog box is open, showing the message "127.0.0.1:5500 indique I'm a web component" with an "OK" button.