université de BORDEAUX

Bloc 1

JavaScript

université de BORDEAUX

JavaScript

- JavaScript
  - 1995 (début), ECMA standard (1997), ECMAScript (2018)
- Programme le **comportement** des pages web
  - Ajouter, changer et retirer tous les éléments et les attributs HTML.
  - Ajouter, changer et retirer tous les styles CSS.
  - Ajouter, changer, retirer et réagir aux événements HTML.
- **Interprété** par le navigateur web (moteur JavaScript)
  - Pas de compilation
- Utilisé dans d'autres contextes
  - scripts et macros dans des applications de bureau, programmation côté serveur, etc.

Bases du langage (1/3)

- Typage **dynamique**
  - variables typées (entiers, réels, chaines de caractères…)
  - mais changement de type possible à l'exécution

```
var a;
var b;
a = 5; // a est un entier
b = '5'; // b est une chaines de caractères
var c = (a===b); // c est un booléen
b = 10; // b devient un entier
c = a + b; // c devient un entier égal à 15
```

Bases du langage (2/3)

- Langage **fonctionnel**
  - les fonctions structurent le code
  - elles peuvent prendre d'autres fonctions en paramètre, retourner une fonction, etc.
  - mais pas fonctionnel pur :
    les fonctions peuvent modifier des variables externes

```
var a;

function f() {
    a = 5;
}

f();
```

Bases du langage (3/3)

- Langage **objet** (ou plutôt dictionnaire)
  - ensemble de propriétés
    - couples `nom : valeur`
    - accessibles en lecture et en écriture

```
var johnSnow = {
    first : 'John',
    last : 'Snow',
    isAlive : undefined
}

johnSnow.isAlive = false;
```

Bases du langage (3/3)

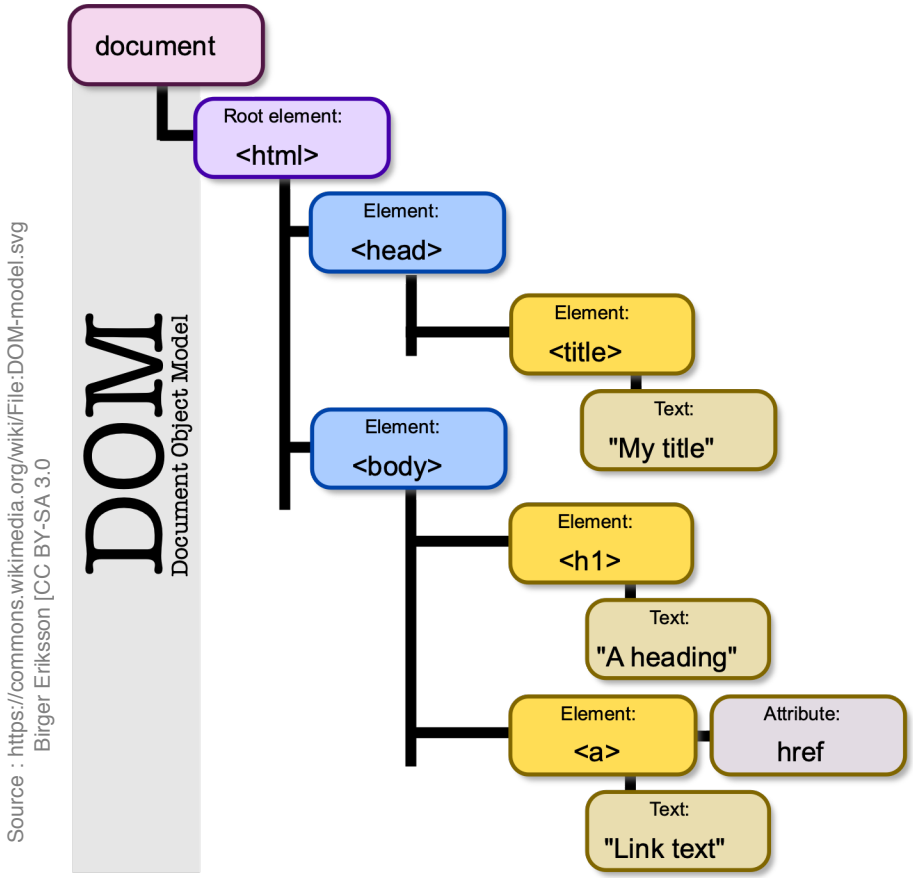
- Langage **objet** (ou plutôt dictionnaire)
  - ensemble de propriétés
  - et de méthodes
    - mot-clé **`this`** pour référencer l'objet courant

```
var johnSnow = {
    first : 'John',
    last : 'Snow',
    isAlive : undefined,
    resurrect : function() {
        this.isAlive = true;
    }
}

johnSnow.isAlive = false;
johnSnow.resurrect();
```

HTML et JavaScript (1/3)

- Lorsqu'une page Web est chargée, le navigateur crée un Document Object Model (DOM)
- Le code JavaScript s'exécute sur le DOM



Source : https://commons.wikimedia.org/wiki/File:DOM-model.svg
Birger Eriksson [CC BY-SA 3.0

HTML et JavaScript (2/3)

➔ Ajouter le code JavaScript dans la page HTML à l'intérieur d'une balise `<script>` :

```
<!DOCTYPE html>
<html>
  <head>
     <meta charset="utf-8" />
  </head>
  <body>
     <script>
       window.addEventListener('load', function () {
       console.log('Cette fonction est exécutée une fois
       quand la page est chargée.'); });
     </script>
  </body>
</html>
```

HTML et JavaScript (3/3)

➔ Mettre le code JavaScript dans un fichier externe (.js)

```
console.log('Cette fonction est exécutée une fois
quand la page est chargée.');
```

et pointer ce fichier depuis le HTML :

```
<!DOCTYPE html>
<html>
  <head>
     <meta charset="utf-8" />
  </head>
  <body>
     <script src="./script.js">
     </script>
  </body>
</html>
```

DOM Element

- Les éléments DOM sont des **objets** JavaScript (API)
- Pour manipuler un élément DOM, il faut :
  1. Le trouver dans le DOM tree par :
     - id : `getElementById`
     - nom de balise : `getElementsByName`
     - nom de classe : `getElementsByClassName`
     - sélecteurs CSS : `querySelector`
  2. Utiliser l'API pour le modifier, créer un nouvel élément, changer son style CSS, etc.

```
var target = document.getElementById("monId");
var img = document.createElement('img');
img.src = './img/02.BMP';
target.appendChild(img);
```

DOM Event (1/3)

- **Événement** (DOM Event) émis lorsqu'un élément DOM subit des interactions
  - Lorsqu'un utilisateur clique sur la souris : `click`
  - Quand une page Web / une image est chargée : `load`
  - Quand la souris passe sur un élément : `mouseover`
  - Lorsqu'un champ de saisie est modifié : `change`
  - Lorsqu'un formulaire HTML est soumis : `submit`

DOM Event (2/3)

- JavaScript permet d'y attacher des **fonctions de traitements** (*callbacks*)

```
function clickAjoutCarte() {
  let img = document.createElement('img');
  img.src = './img/01.BMP';
  document.getElementById("mes-cartes").appendChild(img);
}

document.getElementById("ajout-carte").onclick(clickAjoutCarte);
```

- *Callbacks* exécutées lorsque l'événement associé est émis (exécution asynchrone)

Exemple : Glisser-Déposer (*Drag and Drop*) (1/2)

- Depuis HTML5, tout élément peut devenir **déplaçable** en mettant son attribut **`draggable`** à `true`

```
<img src="img_logo.gif" draggable="true" />
```

- Trois *callbacks* doivent être spécifiés :
  - `ondragstart` : émis lors d'un clique sur l'élément à déplacer
  - `ondragover` : émis lorsque l'élément déplacé survole un autre élément
  - `ondrop` : émis lorsque l'élément déplacé est déposé sur u un autre élément

Exemple : Glisser-Déposer (*Drag and Drop*) (2/2)

JS

```
function allowDrop(ev) {
  ev.preventDefault();
}

function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}

function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
```

HTML

```
<div id="div1" ondrop="drop(event)"
    ondragover="allowDrop(event)"></div>

<img id="drag1" src="img_logo.gif" draggable="true"
    ondragstart="drag(event)">
```

Résumé

- JavaScript est un **langage fonctionnel**, **interprété** et **asynchrone**
- JavaScript s'exécute sur l'**arbre d'éléments** HTML (DOM tree) créé par le navigateur
- Permet de manipuler dynamiquement le comportement des pages web