INF8007 – Languages de scripts
Frontend

Antoine Lefebvre-Brossard

Hiver 2018

Rappels

- Le web fonctionne selon un protocole serveur-client
- Le cours dernier, la partie serveur a été vue
- Le protocole HTTP est utilisé pour communiquer entre les deux en utilisant un système de requêtes et réponses
- Le client utilise typiquement trois languages différents fonctionnant dans tous les fureteurs :
  - HTML : Détermine ce que est affiché
  - CSS : Détermine comment les choses sont affichées
  - JavaScript : Permet de mofifier ce qui est affiché et de communiquer avec des serveurs

HTML

- HTML (HyperText Markup Language)
- Fonctionne par le modèle DOM (Document Object Model)
- Ce DOM détermine la structure d'une page sous la forme d'un arbre
- Ressource principale : *https://www.w3schools.com/tags/default.asp*
- Ce DOM peut être vu dans la plupart des fureteurs (F12 sur Firefox et Chrome) en faisant un click droit et en allant à inspecter

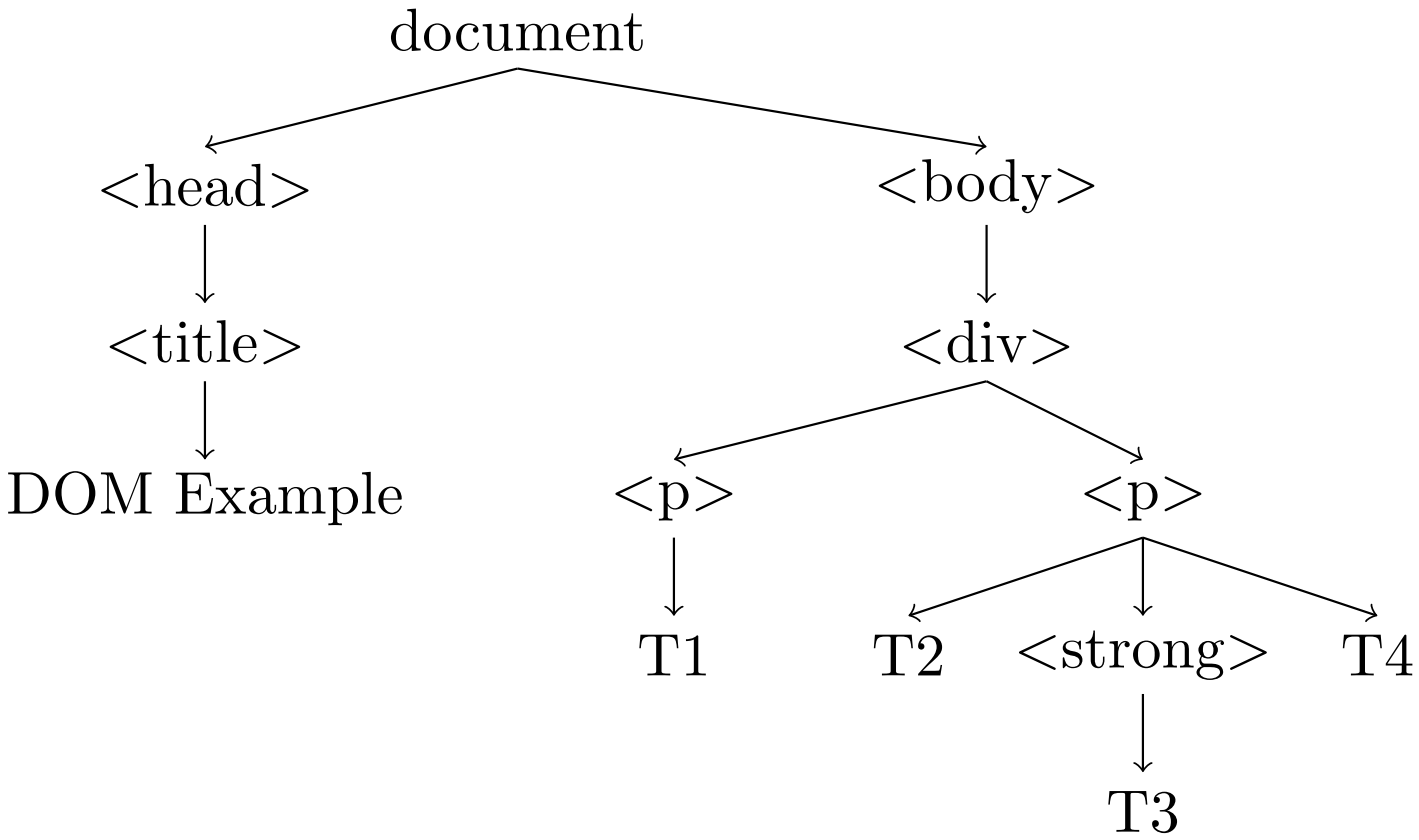
HTML (suite)


```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Example</title>
</head>
<body>
  <div>
    <p>T1</p>
    <p>T2 <strong>T3</strong> T4</p>
  </div>
</body>
</html>
```

T1

T2 **T3** T4

HTML (suite)



CSS

- Alors qu'HTML sert à spécifier l'arbre de contenu, CSS sert à déterminer l'affichage
- CSS (Cascading Style Sheet)
- Plusieurs façons d'accéder à un élément :
  - En suivant l'arbre
  - En donnant une classe à un élément
  - En donnant un id à un élément
- Pour spécifier le chemin dans l'arbre, les éléments sont séparés par des espaces
- La classe est spécifiée en mettant un "." devant le nom
- Le id est spécifié en mettant un "#" devant le nom
- Une classe peut être assignée à plusieurs éléments, mais un id seulement à un seul

CSS (suite)


```html
<!DOCTYPE html>
<html>
<head>
  <title>CSS Example</title>
  <link rel="stylesheet" href="css_example.css">
</head>
<body>
  <div>
    <p>Text 1</p>
    <p class="bold">Text 2</p>
    <p id="green" class="bold">Text 3</p>
  </div>
</body>
</html>
```

```css
body div p {
  font-style: underline;
}

.bold {
  font-weight: bold;
}

#green {
  color: green;
}
```

CSS (suite)

- Ressource principale *https://www.w3schools.com/csSref/default.asp*
- Moyen le plus facile et moderne de placer des éléments sur une page : *https://css-tricks.com/snippets/css/complete-guide-grid/*

JavaScript

- Language de script interprété principalement utilisé pour le côté client, mais de plus en plus populaire du côté des serveurs (node.js) et des applications (electron) du web
- Permet de contrôler le DOM
- Peut être écrit directement dans un fichier HTML entre des tags `script`, mais la meilleure pratique est de le mettre dans un fichier séparé et l'importer dans le HTML par `<script src="file_name.js" type="text/javascript"></script>`

JavaScript (suite)

- Contrairement à Python, une variable est créée en utilisant `var` `` `varName` `` = `` `value` ``
- `let` et `const` sont aussi utilisés, mais il n'est pas nécessaire pour ce cours de connaître leur utilité
- Comme la plupart des languages, JavaScript utilise les "`{}`" pour définir les blocs de code
- Aussi comme beaucoup de languages, les expressions se terminent par un ";"
- Une fonction est définie par `function` et comme Python peut être anonyme

Exemple


```
var x = 10;
var y = [0, 1, 2, 3];
function toThePower(n, e) {
  powers = [];
  for (var i = 0; i < e.length; i++) {
    powers.push(n**e[i]);
  }
  return powers;
}
console.log(toThePower(x, y));
>>> Array [ 1, 10, 100, 1000 ]
```

Fonctions asynchrones

- Quelque chose de très important dans le monde du web (et dans bien d'autres domaines) est le concept de fonctions asynchrones
- L'idée derrière est que plutôt que d'attendre que tous les processus finissent dans l'ordre où ils ont été lancés, on définit plutôt ce qui doit être fait à la fin de chacun d'eux
- Une façon classique de le faire est par le biais d'une fonction de callback, c'est-à-dire une fonction appelée à la fin de l'éxécution
- Dans le contexte du cours, nous ne verrons qu'un exemple : la requête HTTP

Structure de la requête HTTP


```
function getData(url, callback) {
  var xhttp = new XMLHTTPRequest();
  xhttp.onreadystatechanged = function() {
    if (this.readyState == 4 && this.status == 200) {
      let response = JSON.parse(this.responseText);
      callback(response);
    } else if (this.readyState == 4 && this.status == 400) {
      showError(this.responseText);
    }
  };
  xhttp.open('GET', url, true);
  xhttp.send();
}
```

Structure de la requête HTTP (suite)

- La requête est séparée en trois étapes :
  1. La création de l'objet la contenant

     ```
     var xhttp = new XMLHTTPRequest();
     ```
  2. La fonction a éxécuté selon la réponse du serveur

     ```
     xhttp.onreadystatechanged = function() {...}
     ```
  3. L'envoi de la requête au serveur

     ```
     xhttp.open('GET',url, true);
     xhttp.send();
     ```

Structure de la requête HTTP (suite)

- La fonction `onreadystatechanged` est appelée à chaque fois que l'état de la requête change. Le code 4 pour `this.readyState` indique que la requête a été envoyée et reçue. Le reste des codes est disponible à *https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/readyState*
- `this.status` représente le statut HTTP retourné par le serveur (voir diapositives sur les serveurs)
- Le fait d'utiliser `JSON.parse(this.responseText)` assume que le *body* de la réponse est un *string* dans un format JSON. Ceci peut ne pas être approprié si le type de résultat de la requête n'est pas certain. Par exemple, l'erreur retournée est un simple *string* dans l'exemple ci-dessus (nul besoin d'utiliser `JSON.parse`)

Structure de la requête HTTP (suite)

- L'envoi de la requête est fait en 2 étapes :
  1. Débuter l'écriture de la requête (`xhttp.open('GET', url, true)`)
  2. Envoyer celle-ci au serveur (`xhttp.send()`)
- Le troisième paramètre correspond à si la requête devrait être asynchrone ou pas. Par défaut, il est à `true` et devrait presque toujours l'être