

Programmation Web Avancée

Cours 1

Introduction

Généralités et rappels sur le Web

Javascript : survol du langage.



kn@lri.fr

université
PARIS-SACLAY

1 Introduction/Généralités et rappels sur le Web/Javascript : survol du langage

1.1 Introduction

1.2 Généralités et rappels sur le Web

1.3 Javascript : survol du langage

1.4 Javascript : syntaxe

But du cours

- ◆ Vous initier à la programmation Web côté client
- ◆ Vous éviter d'apprendre Javascript par vous même (avec les dégats que cela peut entraîner pour votre santé mentale)
- ◆ Vous permettre d'écrire des programmes rigolos
- ◆ Vous permettre d'avoir des billes pour comprendre ce qui se passe quand du code Javascript plante (c'est à dire à peu près tout le temps)

Format du cours

Chaque semaine :

- ◆ Deux heures de cours (avec moi)
- ◆ Deux heures de TP (avec Diane Gallois-Wong et Florian Faissole)

Le cours présente différents concepts dans un cadre général, les TPs permettent une application directe. Les TPs vous donnent les briques de base pour faire le projet :

Cette année ce sera **un jeu de type bomberman** :

- ◆ C'est un jeu que vous avez tous déjà manipulé
- ◆ Cela touche à plein d'aspects : interaction avec l'utilisateur, parsing (pour la lecture des niveaux), modification dynamique de pages, sauvegarde persistente des données, ...



- ◆ CC (50%) :
 - ◆ Interro (25% du CC) séance 5 ou 6 (QCM, code sur papier et fichier à rendre)
 - ◆ Projet (75% du CC) rendu de code/rapport et démo.
- ◆ Examen (50%) : sur table

Il est plus que recommandé de chercher en TP aussi sur papier pour vous entraîner à l'examen

5 / 57

Le Web en un slide



- ◆ (au commencement) Protocole d'échange de documents (les pages Web)
- ◆ Le format de fichier est **HTML** (on va revenir dessus), spécifié par le W3C
- ◆ Les fichiers sont stockés sur des **serveurs Web**
- ◆ Des clients (**les navigateurs Web**) se connectent au serveur en utilisant le protocole **HTTP** (protocole d'application au dessus de TCP/IP).
- ◆ Les **ressources** sont identifiées par des URLs (des chaînes de caractères au format :
proto://machine:port/chemin/vers/la/ressource
- ◆ Les documents HTML contiennent des liens hypertexte qui permettent de naviguer de pages en pages via des URLs. Les ressources d'une page (images, scripts, feuilles de style, ...) sont aussi dénotées par des URLs.
- ◆ Le navigateur Web récupère les ressources et assure le rendu (souvent graphique) de la page.

7 / 57



- 1 Introduction/Généralités et rappels sur le Web/Javascript : survol du langage
 - 1.1 Introduction ✓
 - 1.2 Généralités et rappels sur le Web
 - 1.3 Javascript : survol du langage
 - 1.4 Javascript : syntaxe

HTML



HyperText Markup Language : langage de mise en forme de documents hypertextes (texte + liens vers d'autres documents). Développé au CERN en 1989.
1991 : premier navigateur en mode texte
1993 : premier navigateur graphique (mosaic) développé au NCSA (National Center for Supercomputing Applications)

8 / 57

- ◆ est un document *semi-structuré*
- ◆ dont la structure est donnée par des *balises*

Exemple	Rendu par défaut
Un texte en gras	Un texte en gras
<code>Un lien</code>	Un lien
<pre> Premièrement Deuxièmement </pre>	<ul style="list-style-type: none"> ◆ Premièrement ◆ Deuxièmement

On dit que **<toto>** est une balise *ouvrante* et **</toto>** une balise *fermante*. On peut écrire **<toto/>** comme raccourci pour **<toto></toto>**.

Convention pour le cours

Afin d'être compatible à la fois XHTML et HTML5, on utilisera dans le cours les conventions suivantes :

- ◆ Les balises suivantes et celle-ci **uniquement** doivent ne pas avoir de contenu :
area, base, br, col, command, embed, hr, img, input, keygen, link, meta, param, source, track, wbr

Exemple : ``.

Toutes les autres balises doivent **obligatoirement** être de la forme:

```
<toto ... > ... </toto>
```

- ◆ Les noms de balises sont toujours en **minuscule**
- ◆ Le *doctype* (balise spéciale indiquant le type de document) est :

```
<!DOCTYPE html>
```

XHTML version « XML » de HTML. Principales différences :

- ◆ Les balises sont *bien parenthésées* (`<a>` `<c>` `</c>` `` est interdit)
- ◆ Les balises sont en minuscules

Les avantages sont les suivants

- ◆ HTML autorise les mélanges majuscule/minuscule, de ne pas fermer certaines balise ... Les navigateurs corrigent ces erreurs de manières *différentes*
- ◆ Le document est *structuré* comme un programme informatique (les balises ouvrantes/fermantes correspondent à { et }). Plus simple à débbugger.

Rôle d'(X)HTML

Séparer la *structure* du document de son *rendu*. La structure donne une *sémantique* au document :

- ◆ ceci est un titre
- ◆ ceci est un paragraphe
- ◆ ceci est un ensemble de caractères importants

Cela permet au navigateur d'assurer un rendu en fonction de la sémantique. Il existe différents types de rendus:

- ◆ graphique interactif (Chrome, Firefox, Internet Explorer, ...)
- ◆ texte interactif (Lynx, navigateur en mode texte)
- ◆ graphique statique (par ex: sur livre électronique)
- ◆ rendu sur papier
- ◆ graphique pour petit écran (terminal mobile)

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
  <head>
    <title>Un titre</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h1>Titre de section</h1>

    <p> premier paragraphe de texte. On met
    un <a href="http://www.lri.fr">lien</a> ici.
    </p>

    <!-- on peut aussi mettre des commentaires -->

  </body>
</html>
```

13 / 57

Titres

Les balises `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, permettent de créer des titres de section, sous-section, sous-sous-section ,...

Titre de niveau 1

Titre de niveau 2

Titre de niveau 3

Titre de niveau 4

Titre de niveau 5

Titre de niveau 6

15 / 57

Pour être *valide* un document XHTML contient **au moins** les balises suivantes :

- ◆ Une balise **html** qui est la **racine** (elle englobe toutes les autres balises). La balise **html** contient deux balises filles: **head** et **body**
- ◆ La balise **head** représente l'en-tête du document. Elle peut contenir diverses informations (feuilles de styles, titre, encodage de caractères, ...). La seule balise **obligatoire** dans **head** est le titre (**title**). C'est le texte qui est affiché dans la barre de fenêtre du navigateur ou dans l'onglet.
- ◆ la balise **body** représente le contenu de la page. On y trouve diverses balises (**div**, **p**, **table**, ...) qui formatent le contenu de la page

14 / 57

Sections

Des sections (groupes de paragraphes, tables, listes, ...) introduits avec les balises `<div>`. Il est courant (on le fera en TP) d'utiliser les **div** comme des « boîtes » rectangulaires dont on ajuste finement la couleur, la position, la taille, ... via leur style CSS.

16 / 57

Paragrapes

Des paragraphes de textes sont introduits avec les balises `<p>`. Par défaut chaque paragraphe implique un retour à la ligne:

```
<p>Lorem ipsum      dolor sit amet, consectetur  
adipiscing elit, sed do eiusmod tempor incididunt ut labore et  
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud  
exercitation ullamc</p>  
<p>Nouveau paragraphe</p>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamc

Nouveau paragraphe

Remarque : par défaut, les espaces, retour à la ligne, ... sont ignorés et le texte est reformaté pour remplir la largeur de la page.

17 / 57

Tableaux

On peut formater des tables en utilisant :

- ◆ La balise `<table>` pour délimiter la table
- ◆ La balise `<tr>` pour délimiter une ligne de la table
- ◆ La balise `<th>` pour délimiter une tête de colonne
- ◆ La balise `<td>` pour délimiter une case
- ◆ L'attribut `colspan` permet de fusionner des colonnes

```
<table>  
<tr> <th>Nom</th> <th>Prénom</th> <th>Note 1</th> <th>Note 2</th></tr>  
<tr> <td>Foo</td> <td>Bar</td> <td> 15</td> <td>12</td> </tr>  
<tr> <td>Doe </td> <td>John</td> <td colspan="2">Absent</td></tr>  
</table>
```

Nom	Prénom	Note 1	Note 2
Foo	Bar	15	12
Doe	Jonh	Absent	

Les espaces et retours à la ligne ne sont là que pour rendre le code lisible !

19 / 57

Mise en forme du texte

Les balises `` (*bold*, gras), `<i>` (*italic*, italique), `<u>` (*underlined*, souligné) `` (*emphasis*, important) et beaucoup d'autres permettent de décorer le texte.

```
<p><b>Lorem ipsum dolor</b> sit amet, consectetur  
adipiscing elit, sed do eiusmod tempor incididunt ut labore et  
dolore magna aliqua. <u>Ut enim ad minim veniam</u>, <em>quis</em> nostrud  
exercitation ullamc</p>  
<p><i>Nouveau</i> paragraphe</p>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, *quis* nostrud exercitation ullamc

Nouveau paragraphe

18 / 57

Listes

On peut créer des listes énumérées (avec ``, *ordered list*) ou non énumérées (avec ``, *unordered list*). Chaque ligne est limitée par une balise `` (*list item*)

```
<ul>  
<li> Un élément </li>  
<li> <ol> <li> Un autre élément </li>  
      <li> <ol> <li> Un sous-élément</li>  
      <li> Un autre sous-élément</li>  
    </ol>  
</li>  
</ol>  
<li>Le dernier</li>  
</ul>
```

- Un élément
- 1. Un autre élément
 2.
 1. Un sous-élément
 2. Un autre sous-élément

20 / 57

On peut faire référence à une autre **ressource** en utilisant un lien hyper-texte (balise `<a/>` et son attribut `href`). La cible du lien peut être absolue (une URL complète avec le protocole, par exemple `https://www.lri.fr`) ou relative (par exemple `foo.html`). Si l'URL est relative, le chemin est substitué à la dernière composante de l'URL de la page courante. Si l'URL commence par un `#` elle référence, l'attribut `id` d'un élément de la page:

```
<a href="https://www.lri.fr">Le LRI</a>
<a href="../../../index.html">Un lien</a>
<a href="#foo">On va vers le titre</a>
...
<h1 id="foo">Le titre</h1>
```

Le LRI Un lien On va vers le titre ...

21 / 57

Cascading Style Sheets (CSS)

CSS : Langage permettant de décrire le *style graphique* d'une page HTML

On peut appliquer un style CSS

- ◆ À un élément en utilisant *l'attribut style*
- ◆ À une page en utilisant l'élément `<style>...</style>` dans l'en-tête du document (dans la balise `<head>...</head>`).
- ◆ À un ensemble de pages en référençant un fichier de style dans chacune des pages

23 / 57

- ◆ On n'a normalement pas le droit de mettre n'importe quel élément n'importe où (i.e. pas de `` tout seul)
- ◆ Il existe une spécification précise de HTML 5 (plusieurs dizaines de pages uniquement pour les balises)
- ◆ Il existe aussi des validateurs, il faut les utiliser le plus possible
- ◆ De manière générale, les espaces sont ignorés, on prendra donc bien soit de les utiliser judicieusement pour rendre le code de la page lisible
- ◆ Tous les éléments ont un style (moche) par défaut. On peut modifier ce style grâce à des propriétés CSS.

22 / 57

L'attribut style

```
<a href="http://www.u-psud.fr" style="color:red">Un lien</a>
```

Aperçu:

Un lien

Inconvénients :

- ◆ il faut copier l'attribut style pour tous les liens de la page
- ◆ modification de tous les éléments difficile

24 / 57

```
<html>
<head>
  <title>...</title>
  <style>
    a { color: red; }
  </style>
</head>
<body>
  <a href="#">Lien 1</a> <a href="#">Lien 2</a>
</body>
</html>
```

Apperçu :

[Lien 1](#) [Lien 2](#)

Inconvénient : local à une page

25 / 57

Syntaxe

Une *propriété* CSS est définie en utilisant la syntaxe:

nom_prop : **val_prop** ;

◆ Si on utilise l'attribut **style** d'un élément:

```
<a href="#" style="color:red;border-style:solid;border:1pt;">Lien 1</a>
```

◆ Si on utilise un fichier **.css** ou une feuille de style:

```
a {
  color : red;
  border-style: solid;
  border: 1pt;
}
h1 { /* Le style des titres de niveau 1 */
  text-decoration: underline;
  color: green;
}
```

27 / 57

Fichier **style.css**:

```
a { color: red; }
```

Fichier **test.html**:

```
<html>
<head>
  ...
  <link href="style.css" type="text/css" rel="stylesheet" />
</head>
  ...
</html>
```

Modifications & déploiement aisés

26 / 57

C'est tout pour le rappel !

Si vous voulez vous rafraîchir la mémoire sur HTML & CSS, voici quelques pointeurs :

- ◆ Spécification du W3C pour HTML : <http://www.w3.org/TR/html5/>
- ◆ Spécification du W3C pour CSS : <http://www.w3.org/Style/CSS/specs.en.html>
- ◆ Le site de tutoriels W3Schools : <http://www.w3schools.com/>
- ◆ Internet (avec entre autres, mon cours de L2 : https://www.lri.fr/~kn/upw_en.html)

Pour le TP, il n'est nécessaire de modifier que les propriétés *top*, *left*, *width*, et *height*.

28 / 57

1 Introduction/Généralités et rappels sur le Web/Javascript : survol du langage

1.1 Introduction ✓

1.2 Généralités et rappels sur le Web ✓

1.3 Javascript : survol du langage

1.4 Javascript : syntaxe

Le modèle du Web présenté précédemment est **statique**. Les documents sont stockés sous forme de fichiers physiques, sur le disque dur d'un serveur.

Très tôt on a souhaité générer **dynamiquement** le contenu d'une page.

1993 : invention des scripts CGI qui permettent au serveur de récupérer les paramètres d'une requête HTTP et de générer du HTML en réponse.

La programmation Web **côté serveur** évolue ensuite (apparition de PHP en 1994, puis possibilité ensuite de programmer le côté serveur dans des langages plus robustes, comme Java, ...)

Un problème subsiste : le manque d'interactivité. En effet, on est contraint par le modèle :

formulaire HTML → envoi au serveur → calcul de la réponse → retour au client → rechargement de page. Problème d'interactivité (latence réseau, rendu graphique d'une nouvelle page, ...).

30 / 57

Web Dynamique côté client

Avec l'arrivée de Java (1995) la notion d'Applet fait son apparition. Ils sont (pour l'époque) une manière **portable** d'exécuter du code côté client.

Problème : Java est trop lourd à l'époque (c'est un vrai langage, il fait peur aux créateurs de site, les performances sont médiocres, ...).

1995 : Brendan Eich (Netscape) crée Javascript en 10 jours. Il emprunte de la syntaxe à Java/C, et Netscape Navigator 2.0 embarque un interpréteur Javascript en standard

Le langage est rapidement adopté, mais chaque navigateur implémente sa propre variante. Le langage lui-même est **standardisé** en 1996 (ECMAScript, standardisé par l'ECMA).

2009 : Standardisation ISO de ECMAScript 5 🇺🇸 (2011 pour la version 5.1)

2015 : Standardisation ISO de ECMAScript 6 🇺🇸

2016 : Standardisation ISO de ECMAScript 7 🇺🇸

2017 : Standardisation ISO de ECMAScript 8 🇺🇸

2018 : Standardisation ISO de ECMAScript 9 🇺🇸

31 / 57

Comment exécute-t-on du Javascript ?

◆ Côté client : le code javascript est exécuté par le navigateur Web. Il doit donc être référencé dans une page HTML :

- ◆ Soit en utilisant l'attribut **src** d'une balise **script**
- ◆ Soit en mettant le code directement dans une balise **script**

```
<html>
...
<script type="text/javascript" src="toto.js"></script>
...
<script type="text/javascript">
  alert("Hello, World!");
</script>
</html>
```

◆ Côté serveur : on peut maintenant utiliser Javascript comme un langage généraliste grâce à l'interpréteur **Node.js**

32 / 57



Javascript est un langage :

Here be dragons

- ◆ **Dynamique** (tout est fait à l'exécution)
- ◆ En particulier **dynamiquement typé** (donc pas typé)
- ◆ **Impératif** (effets de bords, boucles **for**, notion d'instruction, ...)
- ◆ **Fonctionnel** (les fonctions sont des objets de première classe que l'on va manipuler à haute dose)
- ◆ **Objet** (mais sans notion de classe, ce qui rend la chose ~~merdique~~ amusante)
- ◆ **Interprété**, avec une compilation **à la volée** (JIT). Les navigateurs Web modernes ont des performances incroyables (possibilité de faire des jeux 3D par exemple)

33 / 57

Environnement de développement



- ◆ Pour les premiers cours, on utilisera les navigateur ChromeTM ou FirefoxTM.
- ◆ Il est recommandé d'utiliser le même navigateur pour s'abstraire dans un premier temps des problèmes de compatibilité
- ◆ On peut utiliser un éditeur de texte simple (Eclipse est à proscrire, le support Javascript est notoirement mauvais). Visual Studio Code (Microsoft) est un bon éditeur de code Javascript.
- ◆ On utilisera la console de débogage Javascript du navigateur (Ctrl-Shift-J)






35 / 57

34 / 57

Plan



- 1 Introduction/Généralités et rappels sur le Web/Javascript : survol du langage
 - 1.1 Introduction ✓
 - 1.2 Généralités et rappels sur le Web ✓
 - 1.3 Javascript : survol du langage ✓
 - 1.4 Javascript : syntaxe

- ◆ Le standard 5  a mis de l'ordre et rendu le langage utilisable 
- ◆ Le standard 6  (ou plus) est supporté dans la plupart des navigateurs modernes, ainsi que dans les solutions externes (Node, ...) 
- ◆ Certains navigateur Webs (IE, certains navigateurs mobiles) ne supportent que la version 5 ou moins ! 

Le standard 5 introduit le mode *strict*, qui permet plus de verifications et impose une version raisonnable de la portée des variables. On l'utilise en mettant `'use strict';` dans le bloc qu'on souhaite rendre strict

```
'use strict'; //enclenche mode strict pour JS >= 5, sans
              //effet sinon. Appliqué à tout le fichier.
function f(x) {
  'use strict'; //Le corps de la fonction est en mode strict'
  ...
}
```

On utilisera tout le temps le mode strict en TP.

37 / 57

Booléens (boolean)

true/false :

vrai/faux

Opérateurs logiques :

! : négation (unaire)
&&, || : « et » logique, « ou » logique

39 / 57

Il n'y a pas de type entier, uniquement des **numbers** qui sont flottants IEEE-754 double précision (64 bits : 53 bits pour la mantisse, 11 bits pour l'exposant, 1 bit de signe).

Entiers base 10 : 10, 3444, -25, 42, ...
Flottants : 1.3, 0.99, 00.34e102, -2313.2313E-23, ...
Entiers base 8 : 0755, -01234567, ...
Entiers base 16 : 0x12b, -0xb00b5, 0xd34db33f, ...



Le standard garantit que tous les entiers 32bits sont représentables exactement (sans arrondi). On peut écrire des entiers plus grands que $2^{31}-1$ mais au bout d'un moment on a une perte de précision.

Opérateurs arithmétiques :

- : « Moins » unaire
+, -, *, % : addition, soustraction, produit, modulo
/ : Division (flottante)

38 / 57

Variables, affectations

- ◆ Un nom de variable commence toujours par une lettre (majuscule ou minuscule), **\$** ou **_** et se poursuit par un de ces caractères ou un chiffre.
- ◆ Les variables sont définies au moyen de **const** , ou **let** 

Exemples :

```
const $foo = 123;
let bar = 1323e99;
let _toto = bar;
bar = 44; //mise à jour
$foo = 32; //erreur, $foo est const
```

Attention on peut définir une variable sans l'avoir déclarée, et ça « marche » mais ça ne fait pas ce que l'on pense.

Attention le mot clé **var** existe, mais il est à proscrire (explication plus tard).

40 / 57

Encodées en UTF-16 (ou UCS-2), délimitées par des « ' » ou « " »

Opérations sur les chaînes :

- foo[10] :** accès au 11^{ème} caractère, renvoyé sous la forme d'un chaîne contenant ce caractère
- pas de mise à jour :** les chaînes sont immuables
- +** : concaténation
- s.length :** longueur
- s.concat("23") :** concaténation (bis)

Un grand nombre de méthodes sont disponibles, on les verra prochainement (expressions régulières, recherche, remplacement, ...)

41 / 57

Comparaisons

Opérateurs de comparaisons

Opérateur	Description
a == b	Égal, après conversion de type
a != b	Différent, après conversion de type
a === b	Égal et de même type
a !== b	Différent ou de type différent
a < b	Strictement plus petit, après conversion de type
a > b	Strictement plus grand, après conversion de type
a <= b	Plus petit, après conversion de type
a >= b	Plus grand, après conversion de type

43 / 57

null est une *constante* spéciale, de type *object*. Elle permet d'initialiser les variables comme en Java.

undefined est une *constante* spéciale, de type *undefined*. Elle correspond à la valeur d'une variable non initialisée ou d'une propriété non existante pour un objet.

42 / 57

Objets

La structure de données de base est l'objet

```
{ } //Un objet vide
{ x : 1, y : 2 } //Un objet avec deux champs x et y.
o.x //Accès à un champ
o['x'] //Syntaxe alternative
o.z = 3; //rajoute le champ z à l'objet o !
```

En javascript, tout est objet

```
"123".concat("456") //renvoie la chaîne "123456"
3.14.toString() //renvoie la chaîne "3.14"
```

44 / 57

Instructions

Comme en C/C++/Java ... les expressions sont aussi des instructions

```
x = 34;
25;           //la valeur est jetée.
f(1999);
```

Javascript essaye d'insérer automatiquement des « ; ». Pour ce cours on ne lui fait pas confiance et on termine toutes les instructions, sauf les blocs par un « ; »

Structures de contrôle : conditionnelle

```
if (c) {
    // cas then
} else {
    // cas else
}
```

Les *parenthèses* autour de la condition *c* sont obligatoires. La branche *else { ... }* est optionnelle. Les accolades sont optionnelles pour les blocs d'une seule instruction

45 / 57

switch/case

```
switch ( e ) {
  case c1:
    bloc1

  case c2:
    bloc2

  ...
  default:
    blocdefault
}
```

- ◆ l'expression *e* est évaluée et sa valeur comparée tour à tour aux *constantes c_i*
- ◆ s'il y a égalité, le *bloc* correspondant est évalué. En fin de bloc, on exécute **break** pour sortir du **switch**, sinon le *bloc suivant est exécuté*.
- ◆ si aucune égalité et présence d'un label **default**, *bloc_{default}* est exécuté, sinon on sort du **switch**
- ◆ les constantes peuvent être des entiers, booléens ou chaînes de caractères.

47 / 57


46 / 57

Boucles

```
while ( c ) {
    //corps de la boucle while
}

do {
    //corps de la boucle do
} while ( c );

for(init ; test ; incr) {
    //corps de la boucle for
}
```

Il existe des constructions plus modernes (itérateurs de tableaux, boucles « foreach » ) qui seront détaillées au fur et à mesure du cours.

48 / 57

break et continue

break : sort de la boucle immédiatement
continue : reprend à l'itération suivante

49 / 57

Fonctions

On peut définir des fonctions globales :

```
function f(x1, ..., xn) {  
    // instructions  
};
```

On utilise le mot clé **return** pour renvoyer un résultat (ou quitter la fonction sans rien renvoyer)

On peut aussi créer des fonctions « inline » :

```
let z = 1 + (function (x, y) { return x * y; })(2,3);  
// x contient 7
```

On dispose donc de la syntaxe alternative pour la déclaration de fonction :

```
let f = function (z) { return x+1; };
```

51 / 57

Exceptions

Syntaxe similaire à Java/C++ :

```
try{  
    ...  
} catch (ex) { /* faire qqchose avec ex */ }
```

On peut lever une exception avec **throw (e)**, où **e** peut être n'importe quelle valeur.
À l'inverse de Java, il n'y a pas de multi-catch, car il n'y a pas de type pour différencier les objets.

50 / 57

Fonctions anonymes ^{ES6}

Les fonctions anonymes étant souvent utilisées, ES6 introduit une notation allégée :

```
let f = (x, y) => x + y; // le corps consiste en une seule  
                        // expression dont la valeur est renvoyée  
  
let g = (x, y, z) => { // fonction dont le corps est un bloc  
    let u = x + y + z;  
    let v = x - y - z;  
    return u * v;  
};
```

52 / 57

En première approximation, « les méthodes » ne sont que des fonctions stockées dans les champs d'un objet :

```
let obj = { x : 1, y : 1 }; // objet
obj.move = function (i, j) {
  obj.x += i;
  obj.y += j;
};

obj.move(2, 3);
```

On verra que c'est beaucoup plus ~~crade~~ subtil que ça.

53 / 57

Éléments HTML

Les éléments HTML (**document** ou les objets renvoyés par **getElementById**) implémentent l'interface DOM du W3C (Document Object Model, une spécification pour faire correspondre des concepts HTML sur des langages Objets). Les méthodes qui vont nous intéresser pour le TP :

foo.addEventListener("event", f) : Exécute la fonction **f** quand l'évènement "**event**" se produit sur l'élément **foo** (ou un de ses descendants).

foo.innerHTML = "Yo !" : Remplace tout le contenu de l'élément **foo** par le fragment de document contenu dans la chaîne de caractère.

foo.value : Permet de modifier ou récupérer la valeur de certains éléments (en particulier les zones de saisies de texte).

foo.style : Accès au style CSS de l'objet, représenté comme un objet Javascript

55 / 57

L'objet global **document** représente le document HTML. Il implémente l'interface DOM et on peut donc le parcourir comme un arbre (propriétés **firstChild**, **parent**, **nextSibling** ...).

La méthode **document.getElementById("foo")** permet de récupérer un objet représentant l'élément HTML de la page ayant l'attribut **id** valant "**foo**" (**null** si cet élément n'existe pas)

54 / 57

Évènements

Les navigateurs Web se programment de manière événementielle : on attache des fonctions (**event handlers**) à des éléments. Quand un certain évènement se produit, la fonction est appelée :

```
//récupération de l'élément
let divToto = document.getElementById("toto");
//On suppose qu'il existe et on ne teste pas null

toto.addEventListener("click", (e) => {
  if (divToto.style.background != "") {
    //la feuille de style reprend le dessus
    divToto.style.background = "";
  } else {
    // on modifie le style par défaut pour l'élément toto
    divToto.style.background = "pink";
  }
});
```

Le paramètre **e** que prend la fonction permet de récupérer des informations sur l'évènement (coordonnées du pointeur, touche pressée au clavier, ...)

56 / 57

Débuggage : objet console



L'objet **console** possède une méthode **log** permettant d'afficher un objet dans la console du navigateur. C'est un affichage **riche** qui permet d'explorer un objet affiché (démon).