

DIU Enseigner l'Informatique au Lycée

Formulaires HTML, Javascript

kn@lri.fr

<http://www.lri.fr/~kn>



Comprendre le monde,
construire l'avenir

université
PARIS-SACLAY



1 HTTP, HTML, CSS ✓

2 Formulaires HTML, Javascript

2.1 Formulaires HTML

2.2 Javascript : survol du langage

2.3 Javascript : syntaxe

Formulaires



Un *formulaire* est une balise `<form>` contenant des éléments de saisies (champs de texte, boutons, cases à cocher, menus, ...).

La balise `<form>` contient un attribut `action="url"` où `url` est une URL cible.

Si le formulaire contient un *bouton de soumission*, alors l'ensemble des valeurs saisies dans les éléments du formulaire est envoyé à l'URL cible, qui est chargée comme une nouvelle page.

L'utilisation de formulaire pour envoyer des données à un serveur n'est pas au programme du bloc 1

Par contre on va utiliser les éléments de saisies (en particulier les boutons) pour illustrer la notion d'évènement Javascript.

Champs de texte



On peut placer un champ de texte dans un formulaire au moyen de la balise `<input type="text" />`.

```
<input size="10" id="montexte" type="text" value="abc" />
```

Apperçu:

Attributs:

- ◆ size : donne la largeur en caractères
- ◆ id : identifiant de l'élément (pour y accéder depuis Javascript)
- ◆ value : la valeur initiale du champ (si on veut le pré-remplir)

Champs de texte long



Pour des textes multilignes, on utilise la balise `<textarea />`.

```
<textarea id="montexte" rows="3" cols="10">du texte!</textarea>
```

Apperçu: Attributs:

- ◆ rows: donne le nombre de lignes
- ◆ cols: donne le nombre de colonnes (largeur en nombre de caractères)
- ◆ id : identifiant de l'élément (pour y accéder depuis Javascript)

On peut pré-remplir la zone en mettant du texte entre la balise ouvrante et fermante.

Listes déroulantes



On peut définir une liste déroulante avec les balises `<select>` et `<option>`

```
<select id="maliste">
  <option value="bleu" >Bleu</option>
  <option value="vert" >Vert</option>
  <option value="rouge" >Rouge</option>
  <option value="jaune" selected="selected" >Jaune</option>
</select>
```

Apperçu: Attributs:

- ◆ id (dans select): identifiant de l'élément (pour y accéder depuis Javascript)
- ◆ value (dans option): donne la valeur du choix
- ◆ checked (dans option): pré-sélectionne cette option

Le contenu de chaque élément option est celui affiché dans la liste.

Boutons



On peut placer bouton dans un formulaire au moyen de la balise `<button>`.

```
<button id="monbouton"> Clique Moi!</button>
```

Apperçu: 

Attributs:

- ◆ id : identifiant de l'élément (pour y accéder depuis Javascript)

Et les autres ?



Il existe bien d'autres éléments de saisie, mais ceux là sont suffisants pour s'initier à la programmation côté client.

Comme tous les autres éléments HTML, le style de ces éléments est modifiable en CSS.

Plan



1 HTTP, HTML, CSS ✓

2 Formulaires HTML, Javascript

2.1 Formulaires HTML ✓

2.2 Javascript : survol du langage

2.3 Javascript : syntaxe

Web Dynamique ?



Le modèle du Web présenté dans le cours 1 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

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, ...).

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 

Comment exécute-t'on du Javascript ?



- ◆ Plusieurs manières de mettre du code Javascript dans une page HTML. Pour l'enseignement en première, je suggère :

```
<html>
  <head>
    <script type="text/javascript" src="toto.js" defer="defer"></script>
    ...
  </head>
  <body>
    ...
  </body>
</html>
```

- ◆ On expliquera en fin de cours les autres manières de faire et pourquoi elles sont moins indiquées.

Description du langage



Javascript est un langage :

- ◆ **Dynamique** (tout est fait à l'exécution)
- ◆ En particulier **dynamiquement typé** (comme Python)
- ◆ **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)

Environnement de développement



- ◆ On suggère Firefox ou Chromium (la version libre de Google Chrome).
- ◆ 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 (et il est libre).
- ◆ On utilisera la console de débogage Javascript du navigateur (Ctrl-Shift-I)



1 HTTP, HTML, CSS ✓

2 Formulaires HTML, Javascript



2.1 Formulaires HTML ✓

2.2 Javascript : survol du langage ✓

2.3 Javascript : syntaxe

Quel Javascript ?



- ◆ 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 ●
- ◆ Certains navigateur Webs (IE, certains navigateurs mobiles) ne supportent que la version 5 ou moins ! ●

Choix pédagogique : montrer *peu* de Javascript, et montrer du Javascript *propre*.

Avis personnel : le programme se fourvoie quand il dit doctement «*Il s'agit d'examiner le code HTML d'une page comprenant des composants graphiques et de distinguer ce qui relève de la description des composants graphiques en HTML de leur comportement (réaction aux événements) programmé par exemple en JavaScript.*»

On va essentiellement apprendre aux élèves à lire du HTML vilainement écrit, considéré de manière générale comme une mauvaise pratique de programmation (plus de justification en fin de cours)

Nombres (number)



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)

Booléens (boolean)



true/false :

vrai/faux

Opérateurs logiques :

! :


négation (unaire)

&&, || :

« et » logique, « ou » logique

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 du mot clé `let` 

Exemples :

```
let bar = 1323e99;  
let _toto = bar;  
bar = 44;           //mise à jour
```

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 obsolète, à proscrire et signe général de guinolade.

Chaînes de caractères (string)



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

Un grand nombre de méthodes sont disponibles, mais on utilisera juste concaténation et comparaisons.

null et undefined



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.

Comparaisons



Opérateurs de comparaisons

<i>Opérateur</i>	<i>Description</i>
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

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 !
```

On peut considérer que les objets Javascript fonctionnent comme des Dict de Python.

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


Boucles



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

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

```
//exemple :  
for (let i = 0; i < 42; i = i + 1) {  
    // faire qqchose avec i  
}
```

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

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).

La gestion des évènements implique une ***excellente compréhension de la notion de fonction.***

Objet global document



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)

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

Évènements



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

```
//On suppose qu'on a un événement <div id="toto" >
//dans notre HTML
let divToto = document.getElementById("toto");

function toggle_pink() {
  if (divToto.style.background == "") {
    divToto.style.background = "pink";
  } else {
    //s'il est déjà en rose, on l'efface et le
    //style global reprend le dessus
    divToto.style.background = "";
  }
}

toto.addEventListener("click", toggle_pink);
```

ATTENTION



Dans le slide précédant, il est bien écrit :

```
toto.addEventListener("click", toggle_pink);
```

et non pas

```
toto.addEventListener("click", toggle_pink()); //ARCHI FAUX
```

addEventListener attend deux arguments : le nom de l'évènement et une *fonction*.

On appelle cette fonction un *gestionnaire d'évènements*.

Quelques noms d'évènements



`click` : on clique sur l'élément (marche sur tous les éléments, pas seulement les boutons)

`mouseenter/mouseleave` : on survole un élément graphique ou on le quitte avec la souris

`input` : sur les champs de texte, déclenché quand on saisit du texte

`keydown/keyup` : on enfonce ou relâche une touche du clavier

Le gestionnaire d'évènement reçoit un argument `e` en paramètre qui est un objet décrivant l'évènement :

```
let div = document.getElementById("mondiv");
function clavier (e) {
    console.log("On a pressé la touche " + e.key);
}
div.addEventListener("keydown", clavier);
```


Modèle d'exécution



En Javascript, le code est exécuté par le navigateur Web lors de la phase de rendu de la page Web

Si une fonction Javascript prend du temps à s'exécuter, *elle bloque la page*

Si du code Javascript bloque la page, *le navigateur propose à l'utilisateur d'interrompre le script*

Le code Javascript a donc une structure très particulières :

- ◆ Des définitions de fonctions
- ◆ Du code d'initialisation qui associe ces fonctions à des évènements
- ◆ et puis plus rien... mais le programme n'est pas terminé ...
- ◆ ... lorsqu'un évènement se produit, l'une des fonction est appelée. Elle peut donc faire un petit bout de calcul ou modifier la page

Répétitions



Il peut être utile de répéter du code à intervalle régulier (par exemple pour effectuer une animation).

(et évidemment, on ne fait JAMAIS d'attente active, i.e. une boucle qui ne fait « rien » et qui exécute du code tous les X tours)

Javascript propose la fonction `setInterval` :

```
//on réutilise la fonction toggle_pink de tout à l'heure  
  
let timer_id = setInterval(toggle_pink, 3000);  
  
//la fonction est appelée toutes les 3000 ms
```

L'entier renvoyé par `setInterval` est un identifiant du « timer » et permet d'arrêter la répétition:

```
clearInterval(timer_id);  
//la fonction toggle_pink arrête d'être appelée
```

ATTENTION



Dans le slide précédant, il est bien écrit :

```
let timer_id = setInterval(toggle_pink, 3000);
```

et non pas

```
let timer_id = setInterval(toggle_pink (), 3000); //TOUJOURS ARCHI FAUX
```

Modèle d'exécution (suite)



Le modèle d'exécution de Javascript est *séquentiel* :

Deux fonctions Javascript ne peuvent ***jamais*** s'exécuter en même temps, même si les évènements associés se produisent exactement au même moment

Le moteur d'exécution Javascript place tous les gestionnaires d'évènements dans une file d'attente et les exécute l'un après l'autre.

Débuggage : objet console



On peut utiliser `console.log(e)` n'importe où dans du code pour afficher la valeur de l'expression `e` dans la console de débogage Javascript.

Retour sur le programme



Dans l'ancien temps (2000 ?), les gens faisaient des choses comme ceci :

```
<html>
  <head>...</head>
  <body>...

    <button id="b1" onclick="f()">Cliquez Moi</button>

    <input id="t2" type="text" oninput="x = 42; g(y);" />
    ...
```

Cela fonctionne toujours, mais cela va à l'encontre de tous les principes de programmation et de génie logiciel

- ◆ Mélange de HTML (donnée) et de Javascript (code)
- ◆ Maintenance très difficile
- ◆ Lecture très difficile

Les usages actuels veulent que tout le code javascript soit isolé dans des fichiers .js