Présentation de JavaScript

Option Web Multimédia - PeiP 2e année

Claude Barras (claude.barras@u-psud.fr)

Polytech Paris-Sud

septembre 2018

Plan

Evolution

Des débuts chaotiques...

- Lancé en 1995 par Netscape pour son navigateur
- Confusion de nom avec Java (similitudes syntaxiques)
- Des implémentations divergentes entre navigateurs
- Des failles de sécurité (=> JS souvent désactivé)
- Peu performant
- De la concurrence (Flash ActionScript, Microsoft VB...)

...parti pour durer

- Normalisation (ECMAScript), uniformisation, meilleure sécurité
- Course à la performance entre navigateurs
- Au coeur d'AJAX et du Web 2.0 (Google Drive, publicités...)

Généralités

Propriétés

- langage de script interprété
- orienté objet, faiblement typé
- généralement exécuté dans un navigateur

Limitations

- pas d'entrées/sorties de fichiers (pour la sécurité)
- développement difficile (erreurs silencieuses)
- évolution entre les versions (pas de fonctions graphiques avant HTML5)

Syntaxe

Instructions

- syntaxe similaire à C/C++/Java/C#
- différences majuscules/minuscules
- noms de variables ou fonctions (identifiants):
  lettres, '_','$' ou chiffre (sauf en début)
  les mots-clés du langage sont réservés
- instructions séparées par ';' (optionnel, mais à mettre!)
- commentaires de fin de ligne `//` ou multiligne `/* ... */`

Variables et types

Variables

- déclaration et initialisation: `var n=10;`
- non typé (en fait typage automatique et dynamique)
- 2 niveaux: global (déclaration optionnelle) ou local à la fonction

Types de base et valeurs littérales

- **nombre**: décimal (`6.02e+23`) ou entier (32, `0x1e`)
- **chaîne de caractères**: entre `"guillements"` ou `'apostrophes'`
- **booléen**: `true` et `false`
- **null**: aucune valeur (différent de undefined!)

Opérateurs

Opérateurs

- arithmétiques: `+ - * / %`
- concaténation: `+`
- affectation: `= += *= ...`
- égalité: `== !=` (y compris chaînes de caractères)
- identité: `=== !===` (égalité + type identique)
- ordre: `< <= >= >` (nombre ou chaîne)
- logique: `&&` (et) `||` (ou) `!` (non)
- binaire: `&` (et) `|` (ou) `^` (ou exclusif) `~` (non)

Conditions

alternative

```
if (choix == "oui") {
   ...
} else {
   ...
}
```

branchement multiple

```
switch (nom) {
   case "Andreas":
     rep="Guten tag"; break;
   case "John":
     rep="Hello"; break;
   default:
     rep="Bonjour";
}
```

Boucles

répétition avec incrément

```
for (i=0; i<10; i++) {
  j += i;
}
```

tant que ...

```
while (i<10) { ... }
```

faire ... tant que

```
do { ... } while (a == "");
```

ruptures de séquence

- `break` et `continue` comme en C

Fonctions

Déclaration par la mot-clef `function`

```
function ajoute(x,y) {
  return x+y;
}
```

- renvoi du résultat par `return`
- de manière équivalente, on peut créer une fonction comme la valeur d'une variable

```
var ajoute = function(x, y) {
  return x+y;
}
```

Appel de fonction

```
var s = ajoute(1,2);
```

- toujours déclarer les variables locales à la fonction avec 'var' (sinon on modifie des variables globales!)

Structure de données

Déclaration et création d'un objet vide

```
var o = {};
```

- on peut l'utiliser comme un `struct` en langage C
- contient des données de n'importe quel type, sans déclaration préalable

Propriétés (= nom et valeur d'un champ de la structure)

- accessibles par l'opérateur '.'

```
o.x = 1; o.y = 2;
```

Déclaration et initialisation d'un objet

- valeurs des propriétés définies avec ':'

```
var o = {x:1, y:2};
```

Les objets JavaScript permettent de faire beaucoup plus de choses!

- une **méthode** est une fonction qui est la propriété d'un objet

Tableaux

Déclaration et initialisation

- tableau vide ou avec initialisation des valeurs

```
var a = [];
var b = [1, "contenu", false];
```

- un tableau peut mélanger des valeurs de types différents

Elements du tableau

- index avec l'opérateur '[]'

```
a[0] = 1; a[1] = 2;
```

Tableau dynamique

- taille du tableau par la propriété `length`

```
a.length
```

- le tableau s'aggrandit avec la création de nouveaux éléments

```
a[2] = 4; // le tableau contient maintenant 3 éléments
```

- ne pas laisser de cases vides (valeur indéfinie)

Tableaux (suite)

Méthodes

- `join(separateur)`: concatène les éléments
- `slice(debut,fin)`: rend un sous-tableau
- `sort(fonctionCmp)`: trie le tableau
- `push(valeur...)`, `pop()`: gestion en pile
- `shift()`,`unshift(valeur...)`: gestion en file
- `...`

Exemple

```
var a = [3,2,6];
a.push(11);
// rajoute 11 comme 4e élément
a.sort();
// le contenu du tableau est trié
```

Opérateurs spéciaux

Opérateurs

- `new`: création d'un objet
- `delete`: supprime une propriété d'un objet
- `typeof`: renvoie le type de l'opérande ("number", "string", "boolean", "object", "function", "undefined")
- `instanceof`: vrai si l'objet a été créé avec le constructeur de la classe
- `in`: vrai si la propriété appartient à l'objet
- `void`: renvoi `undefined`

Exemple

```
if (typeof a == "object" && test in a) {
  delete a.test
}
```

Number, Boolean et String

Conversion

- `Number(valeur)`: convertit la valeur (ou chaîne) en nombre
- `Boolean(valeur)`: toutes les valeurs converties à `true` à part `0, NaN, null, undefined, ""`
- `String(valeur)`: convertit la valeur en chaîne de caractères `s=String(3);`

Attention!!! Que renvoient ces expressions

```
5 + 3
'5' + '3'
5 + '3'
```

Constantes

- `Number.MAX_VALUE, Number.MIN_VALUE, Number.NaN...`

String

Propriétés

- `length`: longueur de la chaîne

Méthodes

- `charAt(n)`: extrait un caractère (en partant de l'indice zéro)
- `slice/substring(debut,fin)`: sous-chaîne
- `split(délimiteur,limite)`: découpe en tableau de chaînes
- `toUpperCase(),toLowerCase()`: conversion de casse
- `indexOf(sous-chaîne,début)`: recherche de chaîne
- `search(expr),replace(expr,new)`: expressions régulières
- ...

Exemple

```
var s = "azerty"; lastChar = s.charAt(s.length - 1);
```

Math

Constantes

- `Math.PI` : π
- `Math.E` : base des logarithmes e
- ...

Fonctions

- `Math.abs(x)` : valeur absolue
- `Math.sin(x),cos(x),tan(x)...` fonctions trigonométriques
- `Math.ceil(x),floor(x),round(x)`
  arrondi au-dessus, au-dessous, au plus proche
- `Math.sqrt(x),log(x),exp(x),pow(x,y)`
  racine, logarithme et puissance
- `Math.min(...),max(...)` plus petite ou plus grande valeur
- `Math.random()` nombre aléatoire entre 0.0 et 1.0

Plan

Intégration au HTML

balise 'script'

- référence à un fichier JavaScript externe (préférable)

```
<script type="text/javascript" src="code.js"> </script>
```

- dans l'entête pour un pré-chargement du JS
- dans <body> pour une génération dynamique de code HTML

dans un gestionnaire d'événement

```
<input type="button" value="Oui" onclick="zoom()">
```

comme pseudo-URL

```
<a href="javascript: void test()">
```

Exemple de JavaScript intégré

```
<html>
  <head>
<script>
  function fait() {
    document.getElementById("ici").innerHTML = "Merci";
  }
</script>
  </head>
  <body>
<button id="ici" onclick="fait()">Cliquer ici</button>
  </body>
</html>
```

Accès aux balises

Recherche par identifiant

- HTML:

```
<h1 id="titre1">La lune</h1>
```

- JavaScript:

```
document.getElementById("titre1")
```

- retourne un **objet** dont on peut lire ou modifier les propriétés
- Fonction raccourci de la librairie jQuery
  `$("#titre1")`

Recherche par type de balise

- retourne un **tableau** de noeuds

```
document.getElementsByTagName("h1")
```

Retarder l'exécution du code

Ordre d'exécution du code

- le script est exécuté au moment de la lecture de la ligne
- lors de la lecture de l'entête, la page web n'est pas encore créée
- il faut donc retarder l'exécution du code

Trois solutions

- mettre le code JavaScript en fin de fichier
- script externe à exécution retardée dans le HTML

```
<script type="text/javascript" src="code.js" defer> </scrip
```

- lancement programmée à la fin du chargement de la page

```
window.onload = function() {
...
}
```

Modifier le document HTML

Génération dynamique de code

- dans le corps du document:

```
<script>
  document.write("<h1>"+titre+"</h1>");
</script>
```

- exécuté au cours du chargement de la page

Modification du texte

- HTML:

```
<div id="msg">Message</div>
```

- JavaScript:

```
document.getElementById("msg").innerHTML= "OK";
```

- Modification plus complexe: utiliser le DOM

Autres interactions

Afficher un message dans une fenêtre

```
alert('Tout se passe bien.');
```

Saisir une chaîne de caractères dans une fenêtre

```
var nom = prompt('Quel est votre nom?', 'Personne');
if (nom == null) return;
```

Récupérer le contenu d'un champ dans un formulaire

- HTML

```
<input type="text" id="moninput">
```

- JavaScript

```
var m = document.getElementById("moninput").value;
// la valeur obtenue est une chaîne de caractères
```

Plan

TP1

Affichage des diviseurs d'un nombre

- créez fichier .html suivant qui contient uniquement un script

```
<script>
    var n = prompt("rentrer un nombre");
    for (var i = 1; i<= n; i++) {
        if (n%i == 0) {
            alert(i + " est un diviseur de " + n);
        }
    }
</script>
```

TP1

Travail à réaliser

- modifier le code précédent pour décomposer un nombre en facteurs premiers
- définir une **fonction** qui reçoive un nombre en entrée et renvoie un **tableau** contenant la liste des facteurs
- prévoir les cas particuliers (valeur négative ou nulle ou non entière)
- remplacer les appels aux fenêtres de dialogue (`prompt` et `alert`) par l'utilisation d'un formulaire et d'un bouton HTML
- séparer le code Javascript et HTML dans deux fichiers différents
- contrôler le déroulement de l'exécution par des points d'arrêt