Javascript : les bases du langage

Technologies du Web 1

Jean-Christophe Routier
Licence 1 SESI
Université Lille 1

Université Lille1 Sciences et Technologies

UFR IEEA
Formations en Informatique de Lille 1

Javascript

- présentation partielle, et parfois partiale

Javascript

un langage fonctionnel à objet à base de prototypes

- un langage de scripts, interprété
  - les scripts peuvent être placés dans les documents html
  - le navigateur possède un interprète javascript
- le code javascript permet
  - d'agir sur les propriétés des éléments
  - de manipuler l'arbre DOM
  - $\Rightarrow$ dynamicité du document affiché

Intégration de javascript dans la page html

2 cas de figure possibles :

- code javascript directement placé dans le corps du fichier html :

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

- code javascript dans un fichier séparé chargé à l'aide de la balise `script` et de son attribut `src` :

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

NB : flux de chargement du fichier html

Un style de programmation impératif

Programmation impérative

Capitaliser ce qui a été vu en InitProg et API1.

- variables
- structures de contrôles : conditionnelles et itératives

- une nouvelle syntaxe
- quelques différences dans les règles de fonctionnement
- de nouvelles fonctions à apprendre

Types primitifs

- **boolean**
  - 2 constantes `true`, `false`
  - opérateurs : négation `!`, et logique `&&`, ou logique ||
- **number**
  - pas de séparation nette entre entiers et flottants
  - opérateurs : `+`, `-`, `*`, `/` (division flottante), `%` (reste de la division)
  - `-Infinity`, `Infinity`
- **string**
  - pas de type *caractère* séparé de `string`,
    il faut considérer des chaînes de longueur 1
  - les chaînes se notent entre " ou ' : *"exemple"*, *'un autre'*
  - opérateur de concaténation : `+`
  - + objet `String` $\Longrightarrow$ nombreuses méthodes

Variables

Déclaration

Il faut déclarer les variables à l'aide du mot-clef `var`.
Une variable doit être déclarée avant d'être utilisée.

Affectation

L'opérateur d'affectation se note `=`.
Une variable non initialisée a pour valeur `null` ou `undefined`

```
var x;
x = 12;
var texte = "timoleon";
console.log(x);              // 12
var y;
console.log(y);              // undefined
console.log(z);              // ReferenceError: z is not defined
```

Quelques prédicats

```
var x  = 1;
x == 1;                 // vaut true
x != 1;                 // vaut false

var s = "timoleon";
s == "timoleon";        // vaut true
s != "javascript";      // vaut true

var y = 12;
x > y;                  // vaut false
x <= y;                 // vaut true

s < "abracadabra";      // vaut false
"un" >= "deux";         // vaut true

var z;
z == null;              // vaut true
z == undefined;         // vaut true
z = 1;
z == null;              // vaut false
```

Fonctions

Valeur de type fonction

- le mot-clef **function** permet de définir une donnée de type fonction,
- on précise entre parenthèses les **paramètre formels** séparés par des virgules,
- le corps de la fonction est noté entre accolades,
  la valeur de retour d'une fonction est précisé par **return**
  return n'est pas obligatoire (dans ce cas valeur retour = **undefined**)

```
function(x, y) {
  var valeur = x+y;
  return 2*valeur + 1 ;
}

function(x) {
  alert("la valeur est "+x);
}
```

Fonctions

Définition et appel

- définir une fonction c'est définir une variable dont la valeur est de type fonction
- on appelle une fonction en précisant les **paramètres effectifs** entre parenthèses

```
var exemple = function(x, y) {
  var valeur = x+y;
  return 2*valeur + 1 ;
}

exemple(2,3);                    // vaut : 11

exemple;                         // vaut : function(x,y) { ...
```

Autre syntaxe

```
                                    // éqv à :
function exemple (x, y) {           // var exemple = function(x,y) {
  var valeur = x+y;
  return 2*valeur + 1 ;
}

exemple(2,3);                       // vaut 11
```

Règle de portée

Locale et globale

- toute définition de variable dans une fonction est **locale** à la fonction.
- une variable locale masque une variable globale de même nom.

```
var timo = 12;                          // déf. globale, timo vaut 12
var leon = -5;                          // déf. globale, timo vaut -5
var exemple = function() {              // déf. globale de exemple
  var timo = 2;                         // déf. locale, timo vaut 2
  var valeur = 10 ;                     // déf. locale, valeur vaut 10
  return 2*valeur + timo + leon;        // valeur : 2*10 + 2 - 5 = 17
}

exemple();                          // vaut 17
timo;                               // vaut 12
leon;                               // vaut -5
valeur;                             // Erreur : valeur non définie
```

Séquence et bloc

Séquence

Les instructions se terminent par un ; .

Bloc d'instructions

Un bloc d'instructions en séquence se note entre accolades.
Un bloc d'instructions est une instruction.

Attention

Contrairement à de nombreux langages, un bloc ne définit pas de règle de portée.
La seule règle de portée se situe au niveau des fonctions.

Structure conditionnelle

```
if (condition) {
   séquence d'instructions si true
}
else {
   séquence d'instructions si false
}
```

```
var collatz = function(i) {
   if (i % 2 == 0) {
      return i/2;
   }
   else {
      return 3*i+1;
   }
}
```

- la partie `else` n'est pas obligatoire
- `false`, *0*, `""`, `NaN`, `null`, `undefined` valent **false**, tout le reste vaut **true**

Structures itératives : *pour*

```
for (var i =  inf;  i < max ; i=i+1) {     // i=i+1 s'écrit aussi i++
   séquence d'instructions
}
```

```
var sommeEntiers = function(borneMax) {
  var somme = 0;
  for(var i = 0 ; i < borneMax ; i=i+1) {
    somme = somme + i;
  }
  return somme;
}

sommeEntiers(100);                          // somme vaut 4950
```

Structures itératives : *tant que*

```
while ( condition ) {
   séquence d'instructions
}
```

```
var sommeChiffres = function(n) {
  var result = 0;
  while (n > 0) {
    result = result + (n % 10);
    n = Math.floor(n/10);
  }
  return result;
}
sommeChiffres(12345);                    // vaut 15
```

```
do {
   séquence d'instructions
} while (condition)
```

tant que et *pour*

Une boucle *pour* peut toujours s'écrire sous la forme d'une boucle *tant que*.

```
pour i variant de borne_inf à borne_sup répéter
  corps_de_boucle
```

≡

```
i ← borne_inf
tant que i ≤ borne_sup répéter
  debutBloc
    corps_de_boucle
    i ← i + 1
  finBloc
```

En javascript les boucles **`for`** sont des **`while`** déguisées :

```
for ( init ; condition ; increment ) {
  séquence d'instructions
}
```

≡

```
init;
while (condition ) {
  séquence d'instructions;
  increment;
}
```

On peut donc aussi écrire :

```
var sommeChiffres = function(n) {
  for(var result=0; n > 0; n = Math.floor(n/10)) {
    result = result + (n % 10);
  }
  return result;
}
sommeChiffres(12345);                    // vaut 15
```

mais cela ne veut pas dire que l'on doit le faire...

Conversions

- javascript est (très) souple sur la notion de typage.
- javascrit applique « automatiquement » certaines conversions de type sur les valeurs lorsque le contexte le nécessite :
  - vers le type `boolean` (cf. remarque précédente)
  - vers le type `string`
  - vers le type `number`
- a une incidence sur la notion d'égalité

Conversion en booléen et en chaîne

```
var valeurBooleenne = function(val) {
  if (val) {            // dans ce contexte valeur booléenne attendue
    return "'"+val+"' est converti en true";     // chaine attendue
  }
  else {
    return "'"+val+"' est converti en false";
  }
}

valeurBooleenne("abcd");  // -> 'abcd' est converti en true
valeurBooleenne("");      // -> '' est converti en false

var x;
valeurBooleenne(x);       // -> 'undefined' est converti en false
x = 0;
valeurBooleenne(x);       // -> '0' est converti en false
x = 1;
valeurBooleenne(x);       // -> '1' est converti en true
```

Conversion en nombre

- une chaîne dont les caractères représente un nombre est convertie en ce nombre
  NB : dans un expression avec l'opérateur `+` c'est la conversion vers chaîne qui l'emporte
- `NaN` : *Not a Number*
  valeur de conversion pour toute expression qui ne peut être convertie en un nombre
  peut se tester avec fonction `isNaN`.

```
"12.5"*3;           // -> 37.5
"99"-5;             // -> 94

"99"+5              // -> "995"     /!\

"deux"*3;           // -> NaN
isNaN("deux"*3);    // -> true
```

parseInt et parseFloat

- convertissent une chaîne en nombre (entier ou flottant)
- seul le premier nombre dans la chaîne est retourné, les autres caractères (y compris correspondant à des nombres) sont ignorés
- si le premier caractère ne peut être converti en un nombre, le résultat sera `NaN`
- les espaces en tête sont ignorés

```
parseFloat("1.24");                    // ->   1.24
parseInt("42");                        // ->  42
parseInt("42 est la reponse");         // ->  42
parseInt("   42 est la reponse");      // ->  42
parseInt("42estlareponse");            // ->  42
parseInt("42 43 44");                  // ->  42
parseInt("reponse = 42");              // -> NaN
```

Egalités étranges

Attention

Du fait de la conversion, dans certains cas des valeurs de types différents peuvent être considérées égales.

```
1 == "1"          // -> true
10 != "10"        // -> false
1 == "un"         // -> false
0 == false        // -> true
"0" == false      // -> true /!\ alors que "0" se convertit en false
```

L'opérateur `===` teste à la fois le type et la valeur (négation `!==`).

```
1 === "1"          // -> false
0 === false        // -> false
10 === 9+1;        // -> true

1 !== "1";         // -> true
```

Objets

- les objets possèdent des méthodes (= fonctions)
- une méthode s'invoque sur un objet
- on utilise la « *notation pointée* »

Ex : avec l'objet `String`

```
var s = new String("timoleon");     // création d'un objet String
var sub = s.substring(2,6);         // sub vaut "mole"
s.charAt(4);                        // vaut "l"
s.length;                           // vaut 8

                // conversion des valeurs string vers objet String
"abracadabra".charAt(2);            // vaut "r"
"abracadabra".substring(4,8);       // vaut "cada"
```

Premiers liens avec document html

En attendant mieux...

- `window.alert` affiche une « popup » d'information
- `window.prompt`
  - affiche une boîte de dialogue avec une zone de saisie de texte
  - a pour résultat le texte saisi

  Attention : le résultat est de type `string`, prévoir des conversions avec `parseInt` ou `parseFloat` si nécessaire.
- `document.write` et `document.writeln`
  - écrit du texte dans le flux html
  - le texte écrit est interprété par le navigateur

  Attention : efface le contenu du document si le flux doit être réouvert
  $\Rightarrow$ **ne pas utiliser** dans une fonction !