JS : Fondamentaux du langage

L2 MPCIE - UE Développement Web

David Lesaint

david.lesaint@univ-angers.fr

Janvier 2019

Débogage

Débogage

- Affichage dans la console avec `console.log(string);`
- Point d'arrêt pour débogage avec `debugger;`

Utilisez les outils de développement des navigateurs pour analyser et déboguer vos pages HTML/CSS et scripts JS.

Les instructions

Un script JS se compose d'instructions

- Déclaration de variables et constantes.
- Affectation de variables.
- Stuctures de contrôle.

Le `;` en fin d'instruction est optionnel : déconseillé !

bases-instructions.js

```
1 var a = 2;
2 if (undefined === a)
3   console.log('La variable a n\'existe pas.')
4 else
5   console.log('La variable a existe et vaut ' + a);
```

Les blocs d'instructions

Ensembles d'instructions encadrés avec { . . . }

bases-blocs.js

```
1 {
2   var a = 2;
3   if (undefined === a) {
4     console.log('La variable a n\'existe pas.');
5   } else {
6     console.log('La variable a existe et vaut ' + a);
7   }
8 }
```

Les expressions

Instructions dont l'évaluation retourne une valeur

- Opérations et relations sur booléens, nombres, chaînes, tableaux, objets, ...
- Affectation de variable.
- Appel de fonction.
- Déclaration de fonction (les fonctions sont des objets . . .).

JS est un langage à typage dynamique

- Une expression peut combiner des arguments de types différents.
- JS applique alors des règles de transtypage.

Les expressions

bases-expressions.js

```
1  var a;
2  if (a = 3)
3    console.log('a');
4  if (false || true)
5    console.log('true');
6  if (1 + 1 < 3)
7    console.log('1+1>3');
8  if (a = 'a' + 'b')
9    console.log(a);
10 if ('prototype' in Object)
11   console.log('Prototype est dans l\'objet');
12 if (function(a) {
13   console.log(a);
14 })
15   console.log(a);
```

Les commentaires

```
// sur une seule ligne
/* ou
plusieurs */
/**
* @param number input any number
* avec marqueurs pour génération automatique
* de documentation (documentation.js, ...)
*/
```

bases-commentaires.js

```
1 // var a;
2 if (a = 3)
3   console.log('a');
4 /*
5  * if (false || true)
6  * console.log('true');
7  */
8 if ('prototype' in Object) /* comment */ console.log('Prototype est dans l\'objet');
```

Les variables

Les variables : nommage

Un nom de variable

- Peut contenir la plupart des caractères Unicode.
- Doit démarrer par une lettre, `$` ou _.
- Est sensible à la casse.

Eviter le `$` en début de nom de variable (utilisé par de nombreuses librairies JS dont jQuery).

Les variables : nommage

bases-variables-nommage.js

```
1  // Règles de nommage des variables
2  console.log(a);
3  var a = 'a';
4  console.log(a);
5  var A1a$_a = 'A1a$_a';
6  console.log(A1a$_a);
7  var _avélassan = '_avélassan';
8  console.log(_avélassan);
9  // var 1a = '1a'; // ERREUR
10 var $ = '$';
11 console.log($);
```

Les variables : déclaration et portée

Trois formes

`var x=2;`

- En dehors d'une fonction : déclaration d'une variable globale.
- Dans une fonction : déclaration d'une variable locale à la fonction.

`let x=2;`

- Dans un bloc : déclaration d'une variable à portée limitée au bloc.

`x=2;`

- Déclaration d'une variable globale (déconseillé).

Une variable peut être re-déclarée.

Les variables : déclaration et portée

bases-variables-portee.js

```
1  // Portée des variables
2  {
3     let locB = 1; // variable locale
4     var globB = 1; // variable globale
5  }
6  // console.log(locB); //déclenche une erreur
7  console.log(globB);
8
9  function f() {
10    var locF = 1;
11 }
12 // console.log(locF); //déclenche une erreur
13
14 // !! 'var' optionnel !!
15 b = 2;
16 console.log(b);
17 // !! "redéclaration" autorisée !!
18 var b = 3;
19 console.log(b);
```

Les variables : évaluation

Valeur `undefined` pour toute variable sans valeur

`undefined`

- Utilisable pour tester par comparaison si une variable a une valeur.
- Converti à `false` en contexte booléen.
- Converti à `NaN` en contexte numérique.

Une variable valant `null`

- Converti à `false` en contexte booléen.
- Converti à `0` en contexte numérique.

Les variables : évaluation

bases-variables-evaluation.js

```
1  // Evaluation des variables
2  var a;
3  console.log(a); //undefined
4  // console.log("b); //déclenche une exception ReferenceError
5  var b;
6
7  var input;
8  if (input === undefined) {
9    console.log(input);//undefined
10 }
11
12 var monTableau = new Array();
13 if (!monTableau[0]) {
14   console.log(monTableau);//Array []
15 }
16
17 var a;
18 console.log(a + 2);// NaN
19
20 var n = null;
21 console.log(n * 32);//0
```

Les variables : remontée (hoisting)

Faire référence à une variable déclarée ultérieurement

- Mais la valeur sera toujours `undefined`.

bases-variables-hoisting.js

```
1  // Remontée de variables
2  // var x; // sans effet
3  console.log(x === undefined);//true
4  var x = 0;
5
6  var y = 0;
7  (function() {
8    // var y; // sans effet
9    console.log(y);// undefined
10   var y = 1;
11 })();//undefined
```

Remontée de fonctions (hoisting)

Seules les déclarations de fonctions sont remontées

- Les expressions de fonctions ne le sont pas.

bases-fonctions-hoisting.js

```
1  // Remontée de fonctions
2  // Déclaration de fonction
3  toto();// tutu
4  function toto() {
5    console.log("tutu");
6  }
7  // Expression de fonction
8  // machin();// erreur TypeError : machin n'est pas une fonction
9  var machin = function() {
10   console.log("titi");
11 }
```

Les variables globales

Sont des propriétés de l'objet global

- `window` est l'objet global dans les pages web.
- Accès en lecture/écriture aux variables globales avec `window.mavar`

Les constantes

Création avec mot-clé `const`

- Non réinitialisables (sauf leurs propriétés dans le cas de constantes objet).
- Règles de nommage identiques à celles des variables.
- Règles de portée identiques à celles des variables.

bases-constantes.js

```
1 // Constantes
2 const MA_CONST = 1;
3 console.log(MA_CONST);
4 // MA_CONST = 2; //déclenche une erreur
5 const MA_CONST_OBJET = {a:1};
6 MA_CONST_OBJET.a=2;
7 console.log(MA_CONST_OBJET);//Object { a: 2 }
```

Les types

JS est un langage dynamiquement typé

7 types de données

6 types primitifs :

- `boolean` : booléens.
- `number` : entiers et nombres flottants.
- `string` : chaînes de caractères.
- `Undefined` : type d'une variable sans valeur.
- `Null` : type de ce qui est sans type ni valeur.
- `symbol` : symboles utilisables comme clés de propriétés anonymes d'objets (ES6).

1 type complexe :

- `object` : objets.

Conversion de types

Dans les expressions combinant nombres et chaînes avec opérateur

- `+` : conversion des nombres en chaînes.
- `-`, `*`, `/` ou `%` : conversion des chaînes en nombres.

bases-conversion.js

```
1  // Conversions
2  var p = "10p" + 10;
3  console.log(p);//10p10
4  var m = "30" - 10;
5  console.log(m);//20
6  var d = "30" / 10;
7  console.log(d);//3
8  var r = "30" % 10;
9  console.log(r);//0
10
11 var m = "30moins" - 10;
12 console.log(m);//NaN
```

Conversion de types

`parseInt(s,b)`

- Convertit en entier une chaîne `s` le représentant en base `b`.

`parseFloat(s)`

- Convertit en nombre flottant une chaîne `s` le représentant.

bases-parseInt.js

```
1  // parseInt()
2  parseInt(" 0xF", 16);// 15
3  parseInt("17", 8);// 15
4  parseInt("015", 10);// 15
5  parseInt(15.99, 10);// 15
6  parseInt("1111", 2);// 15
7  parseInt("15e2", 10);// 15
8  parseInt("Coucou", 8);// NaN
9  parseInt("546", 2);// NaN
10 parseInt("-15e1", 10);// -15
```

Les littéraux

Sont utilisés pour représenter des valeurs en JS

- Littéraux booléens.
- Littéraux de nombres entiers.
- Littéraux de nombres flottants.
- Littéraux de chaînes de caractères.
- Littéraux d'expressions rationnelles.
- Littéraux de `Null` et `Undefined`.
- Littéraux de tableaux.
- Littéraux objets.

Les valeurs booléennes

Valeurs primitives de `boolean`

Littéraux `true` et `false`.

bases-valeurs-booleennes.js

```
1 // Valeurs booléennes
2 var v = true;
3 console.log(v);// true
4 var f = false;
5 console.log(f);// false
```

Les valeurs numériques

Valeurs primitives de `number`

Les nombres flottants à précision double représentés sur 64 bits au format IEEE 754 $(-(2^{53}-1), 2^{53}-1)$.
Et trois valeurs symboliques :

- `NaN` : *not a number*.
- `+Infinity` : propriété globale représentant $+\infty$.
- `-Infinity` : propriété globale représentant $-\infty$.

Les valeurs numériques

bases-valeurs-numeriques-speciales.js

```
1  // Valeurs numériques prédéfinies
2  var plusGrandNombre = Number.MAX_VALUE;
3  console.log(plusGrandNombre);
4  var plusPetitNombre = Number.MIN_VALUE;
5  console.log(plusPetitNombre);
6  var infini = Number.POSITIVE_INFINITY;
7  console.log(infini);
8  var infiniNégatif = Number.NEGATIVE_INFINITY;
9  console.log(infiniNégatif);
10 var pasUnNombre = Number.NaN;
11 console.log(pasUnNombre);// NaN
12
13 console.log(plusGrandNombre * 2);// Infinity
14 console.log(infini + 0);// Infinity
15 console.log(1E309 + 0);// Infinity
16 console.log(NaN + 0);// NaN
```

Les littéraux numériques

Littéraux de nombres entiers

Les entiers peuvent être exprimés en notation :

- Décimale : suite de chiffres ne commençant pas par 0.
- Octale : préfixe `0o` ou `0O`.
- Hexadécimale : préfixe `0x` ou `0X`.
- Binaire : préfixe `0b` ou `0B`.

Littéraux de nombres décimaux

Un entier, optionnellement signé avec + ou -, suivi

- D'un point comme séparateur décimal puis de la partie décimale,
- Ou d'un exposant séparé avec `e` ou `E` (puissance de 10).

Les valeurs numériques

bases-valeurs-numeriques.js

```
1  // Valeurs numériques
2  var e = 1;
3  console.log(e);// 1
4  var d = 1.2;
5  console.log(d);// 1.2
6  var s = 1.2E34;
7  console.log(s); // 1.2e+34
8  // représentation binaire
9  var b = 0b101;
10 console.log(b);// 5
11 // représentation octale
12 var o = 0o567;
13 console.log(o);// 375
14 // représentation hexadécimale
15 var h = 0xAB;
16 console.log(h);// 171
17 var f = -3.1E-12;
18 console.log(f);//-3.1e-12
```

Les chaînes de caractères

Valeurs primitives de `string`

Ensemble d'éléments de valeurs entières non signées représentées sur 16 bits et indicés à partir de 0.

Littéraux

Zéro ou plusieurs caractères encadrés entre guillemets simples ou doubles.

- `\` est le caractère d'échappement.
- Peut contenir des caractères spéciaux : `\n`, `\t`, `\uXXXX` (Unicode) ...

Interpolation avec *template literals* (ES6)

Une chaîne peut être utilisée comme gabarit (template) :

- Encadrement par backticks.
- Interpolation de la variable `maVar` avec `${maVar}`.

Les chaînes de caractères

bases-chaines.js

```
1  // Chaînes de caractères
2  var s = 'entre guillemets simples';
3  console.log(s);
4  var d = "entre guillemets doubles";
5  console.log(d);
6  var sed = 'chaîne entre \'\' contenant " et échappements';
7  console.log(sed);
8  var des = "chaîne entre \"\" contenant ' et échappements";
9  console.log(des);
10 var m1 = "chaîne sur \
11 deux lignes en utilisant \\";
12 console.log(m1);
13 // chaîne avec caractères spéciaux
14 var m2 = "chaîne \n avec un \\n";
15 console.log(m2);
16 var utf8 = "\u2660 \u2663 \u2665 \u2666";
17 console.log(utf8);
18
19 // chaîne "gabarit" avec backticks
20 var m3= `gabarit avec variable sed : ${sed}`;
21 console.log(m3);
```

Les expressions rationnelles

Littéraux

Motifs encadrés par deux barres obliques.

bases-regex.js

```
1 // Regex
2 var email = /^[a-z0-9]+\.[a-z0-9]+@[a-z0-9\-]+\.[a-z]{2,3}$/;
3 console.log(email);
4 var s = "etudiant.12345@univ-angers.fr";
5 console.log(s.match(email));
6 console.log(s.replace(/(\.)/g, " dot "));
```

Valeur primitive de `Null`

`null`

Littéral (et non pas propriété de l'objet global).

- Utilisable comme valeur-retour de fonction.

bases-valeurs-null.js

```
1 // Valeur de Null
2 var sansTypeNiValeur = null;
3 console.log(sansTypeNiValeur);// null
4
5 console.log(typeof null);// "object" (pas Null pour des raisons historiques)
6 console.log(!null);// true
7 console.log(isNaN(1 + null));// false
```

Valeur primitive de `Undefined`

`undefined`

Propriété de l'objet global.

- Retournée par une méthode ou instruction si la variable à évaluer n'a pas de valeur assignée.
- Retournée par une fonction qui ne renvoie aucune valeur.

bases-valeurs-undefined.js

```
1  // Valeur de Undefined
2  console.log(undefined);// undefined
3  var u;
4  console.log(u);// undefined
5  if (typeof x === 'undefined') { /* donne true */}
6  // if (x === undefined) { } // ReferenceError
7  console.log(typeof undefined);// undefined
8  console.log(!undefined);// true
9  console.log(isNaN(1 + undefined));// true
10 console.log(null === undefined);// false
11 console.log(null == undefined);// true
```

Les valeurs primitives

Les valeurs primitives sont immuables

Excepté dans le cas des valeurs `null` ou `undefined`, il existe un objet équivalent (wrapper) pour chaque valeur primitive qui la contient :

- Objet `Boolean` pour les `boolean`.
- Objet `Number` pour les `number`.
- Objet `String` pour les `string`.
- Objet `Symbol` pour les `symbol`.

La méthode `valueOf()` des wrappers retourne la valeur primitive encapsulée correspondante.

Les tableaux

Littéraux

Eléments séparés par des virgules et entre crochets.

- Les éléments sont indicés à partir de 0.
- Les éléments d'un tableau peuvent être de différents types.
- Les tableaux sont des objets `Array`.

Les tableaux JS ne sont pas associatifs : utilisez des objets.

Les tableaux

bases-tableaux.js

```
1  // Tableaux
2  var v = [];
3  console.log(v);//Array []
4  var r = [ 'A', 'B', 'C' ];
5  console.log(r);//Array [ "A", "B", "C" ]
6  var m = [ r, r, r ];
7  console.log(m);//Array [ [...], [...], [...] ]
8  var t = [ 1, {
9    a : 1,
10   b : '2'
11 }, "c" ];
12 console.log(t);//Array [ 1, {...}, "c" ]
```

Les objets

Littéraux

Sous forme d'une liste de 0 ou plusieurs propriétés entre accolades et définies chacune par une paire `clé:valeur`.

- Une valeur de propriété peut être primitive ou non (autre objet dont tableau et/ou fonction).
- Les valeurs de propriétés peuvent être de différents types.

Les objets

bases-objets.js

```
1  // Objets
2  var v = {};
3  console.log(v);//Object {  }
4  var a = {
5    p : "a",
6    q : 2
7  };
8  console.log(a);//Object { p: "a", q: 2 }
9  var o1 = {
10   mn : {
11     m : 1,
12     n : 2
13   },
14   p : 'p'
15 };
16 console.log(o1);//Object { mn: {...}, p: "p" }
17 var o2 = {
18   q : o1['mn'],
19   r : o1['mn']
20 };
21 console.log(o2);//Object { q: {...}, r: {...} }
```

Les objets

bases-objets-1.js

```
1  var soldes = "Toyota";
2  function carTypes(nom) {
3    return (nom === "Honda") ? nom : "Pas de " + nom + ".";
4  }
5  var voiture = {
6    maVoiture : "Saturn",
7    getVoiture : carTypes("Honda"),
8    spécial : soldes
9  };
10 console.log(voiture.maVoiture); // Saturn
11 console.log(voiture.getVoiture); // Honda
12 console.log(voiture.spécial); // Toyota
13 var voiture = {
14   plusieursVoitures : {
15     a : "Saab",
16     "b" : "Jeep"
17   },
18   7 : "Mazda"
19 };
20 console.log(voiture.plusieursVoitures.b); // Jeep
21 console.log(voiture[7]); // Mazda
```