COLLÈGE DE BOIS-DE-BOULOGNE

420-1WE-BB
Web dynamique

JavaScript – 2

Tirées des notes de Jean-François Brodeur

Objectif pédagogique

- Utiliser le mode strict de JS
- Nommer correctement les variables JS
- Comprendre les déclarations de variables
- Énumérer les caractéristiques des types de variables
- Utiliser la fonction typeof

Soyons strict

- De nouvelles fonctionnalités ont été ajoutées au langage, mais les anciennes fonctionnalités n'ont pas été modifiées.
- Compatibilité parfaite mais les mauvaises décisions restent toujours présentes et ce jusqu'en 2009.
- Lorsque ECMAScript 5 (ES5) est apparu. Il a ajouté de nouvelles fonctionnalités au langage et modifié certaines des fonctionnalités existantes.

Soyons très strict

- **"use strict"** est une directive à l'interpréteur JS et est ignore par les anciennes versions du moteur JS
- **Doit** être déclaré comme la toute première ligne de code, seuls les commentaires sont permis avant.

```
"use strict";
x = 3.14;       // causera une erreur pcq x n'est pas déclaré
```

- Était permis et ne le sont plus avec le mode strict:
  - Assigner une valeur à une variable non déclarée
  - Affecter des valeurs à des propriétés inexistantes ou en mode lecture seul, d'un objet

Les noms de variables

- Les noms de variable en JavaScript doivent débuter par:
  - Une lettre (A-Z ou a-z)
  - Un signe de dollar ($)
  - Un soustiret (_)
- Peuvent contenir des nombres, mais pas au début
- Sont sensible à la casse
- Ne peuvent pas être des mots réservés (let, var, const, return, function…)

La déclaration de variable

- Auparavant n'était pas obligatoire, toujours permis si pas en mode strict, soit le mode défaut
- Ancien mot clé est **var**
  - Possède une portée de fonction – scope en anglais, durée de vie
  - Seulement dans du code plus vieux
- Le mot clé **let** - fortement recommandé
  - Possède une portée de bloc {}
- Le mot clé **const**
  - Déclare une constante

Constantes

- Il existe une pratique répandue d'utiliser des constantes comme alias pour des valeurs difficiles à mémoriser, qui sont connues avant leur exécution.
- Ces constantes sont nommées en utilisant des majuscules et des soustiret « underscores ».

Constantes en majuscules

- Par exemple, créons des constantes pour les couleurs au format dit “web” (hexadécimal) :

```
const COULEUR_ROUGE = "#F00";
const COULEUR_VERT = "#0F0";
// ... quand il faut choisir une couleur
let couleur = COULEUR_ROUGE ;
alert(couleur); // #F00
```

- En d’autres termes, les constantes nommées en majuscules ne sont utilisées que comme alias pour les valeurs “codées en dur”.

Meilleurs pratiques pour les noms de variables

- Utilisez des noms lisibles par des humains comme nomUtilisateur ou shoppingCart.
- Restez à l'écart des abréviations ou des noms courts tels que a, b, c.
- Le nom doit être le plus descriptif et concis possible.
  - Des exemples de noms incorrects sont data et valeur.
  - Meilleur: estUtilisateurCourant,
- S'accorder avec son équipe sur les termes utilisés.
  - Si un visiteur du site est appelé un "utilisateur", nous devrions nommer les variables connexes comme utilisateurCourant ou nouvelUtilisateur, mais pas visiteurCourant ou encore nouvellePersonneEnVille.

Exercices

1. Déclarez deux variables : admin and nom.
2. Assignez la valeur ”Jean" à nom.
3. Copiez la valeur de nom à admin. (affectation)
4. Afficher la valeur de admin en utilisant alert (devrait afficher “Jean”).

Solution

1. let admin, nom; // on peut déclarer deux variables à la fois
2. nom = "Jean";
3. admin = nom;
4. alert(admin); // "Jean"

Discussion

1. const dateNaissance = '18.04.1982’;
2. const age = calcul_age(dateNaissance);

- Devrait-on utiliser des majuscules pour le nom des constantes, pourquoi? DATE_NAISSANCE;

Recommandation

- Généralement les majuscules sont utilisées pour les constantes “codées en dur”. (lorsque la valeur est connue avant exécution et directement écrite dans le code).
- Dans ce code, DATE_NAISSANCE correspond exactement à cela. Nous pourrions donc utiliser les majuscules pour cela.
- En revanche, age est évalué à l’exécution. C’est constant dans le sens où cela ne change pas avec l’exécution du code. Mais c’est un peu “moins constant” que dateNaissance : elle est calculée, nous devrions donc garder les minuscules pour cela.

Les types de données

- Comme vous l'avez sans doute remarqué, la déclaration ne contient pas de type de données, ce qui ne veut pas dire qu'il n'y en a pas.
- Il existe huit types de données de base en JavaScript.
- Une variable en JavaScript peut contenir n'importe quelle donnée. elle peut à un moment être une chaîne de caractères et recevoir plus tard une valeur numérique :

```
// pas d'erreur

let message = "hello";

message = 123456;
```

Les nombres

```
let n = 123;

n = 12.345;
```

- Le type *number* sert à la fois à des nombres entiers et à des nombres à virgule flottante.
- Il existe de nombreuses opérations pour les nombres, par ex. multiplication *, division /, addition +, soustraction - et ainsi de suite.
- Outre les nombres réguliers, il existe des “valeurs numériques spéciales” qui appartiennent également à ce type: Infinity, -Infinity et NaN.

NaN

- Not a Number - NaN
- NaN représente une erreur de calcul. C'est le résultat d'une opération mathématique incorrecte ou non définie, par exemple :
- alert("pas un nombre" / 2); // NaN, une telle division est erronée
- NaN est contagieux. Toute autre opération sur NaN donnerait un NaN :
- alert("pas un nombre" / 2 + 5); // NaN
- Donc, s'il y a NaN quelque part dans une expression mathématique, elle se propage à l'ensemble du résultat.
- Est-ce que ça vous fait penser à quelque chose en SQL?

Opérations mathématiques

- Les opérations mathématiques sont sûres
- Faire des maths est sans danger en JavaScript. Nous pouvons faire n'importe quoi :
  - diviser par zéro, traiter les chaînes non numériques comme des nombres, etc.
- Le script ne s'arrêtera jamais avec une erreur fatale ("crash"). Au pire, nous aurons NaN comme résultat.

BigInt

- En JavaScript, le type “number” ne peut pas représenter des valeurs entières supérieures à 2^{53} (ou moins de -2^{53} pour les négatifs).
- Si il y a un besoin de très gros chiffres, par exemple pour les horodatages de cryptographie ou de précision à la microseconde.
- BigInt a récemment été ajouté au langage pour représenter des entiers de longueur arbitraire.
- Un BigInt est créé en ajoutant n à la fin d’un entier littéral :

```
// le "n" à la fin signifie que c'est un BigInt const bigInt =
12345678901234567890123456789012345678901234567890n;
```

Chaines de caractères

- Une chaîne de caractères en JavaScript doit être entre guillemets.

```
let str = "Hello";
let str2 = ‘Les apostrophes fonctionnent aussi’;
let phrase = `can embed another ${str}`;
```

- En JavaScript, il existe 3 types de guillemets.
  - Double quotes: "Hello".
  - Single quotes: 'Hello'.
  - Backticks: `Hello`.
- Les guillemets simples et doubles sont des guillemets “simples”. Il n’y a pratiquement pas de différence entre eux en JavaScript.

Les guillements obliques – backticks

- Backticks: `Hello`.
- Les backticks sont des guillemets “à fonctionnalité étendue”. Ils nous permettent d’intégrer des variables et des expressions dans une chaîne en les encapsulant dans ${…}, par exemple :

```
let nom = "Jean";


// une variable encapsulée
alert( `Bonjour, ${nom}!` ); // Bonjour, Jean!


// une expression encapulée
alert( `le résultat est:  ${1 + 2}` ); // le résultat est 3
```

- L’expression à l’intérieur de ${…} est évaluée et le résultat devient une partie de la chaîne.

Les booléens – type logique

- Le type booléen n’a que deux valeurs: true et false.
- Ex.: let estPlusGrand = 4 > 1;

```
alert(estPlusGrand ); // true (le résultat de la comparaison est "oui")
```

Truthy and Falsy

- Tous les autres types possèdent une valeur booléenne généralement connu sous truthy ou falsy. C'est un concept unique à JS et vous devez le comprendre afin de vous aider à débugger.
- Les valeurs suivantes **sont toujours falsy**:
  - false
  - 0 (zero)
  - " or "" (empty string).
  - null
  - undefined
  - NaN
- Tout le reste est truthy.

Truthy and Falsy

Falsy	Truthy
false	'0’ (un chaîne de caractère contenant un zéro)
0 (zéro)	'false' (un chaîne de caractère contenant “false”)
" or "" (chaine vide)	[] (un tableau vide)
null	{} (un objet vide)
undefined	function(){} (une function vide)
NaN	

La valeur null

- La valeur spéciale null n'appartient à aucun type de ceux décrits ci-dessus.
- Il forme un type bien distinct qui ne contient que la valeur null :

  ```
  let age = null;
  ```

- En JavaScript, null n'est pas une "référence à un objet non existant" ou un "pointeur null" comme dans d'autres langages.
- C'est juste une valeur spéciale qui a le sens de "rien", "vide" ou "valeur inconnue".
- Le code ci-dessus indique que l'age est inconnu ou vide pour une raison quelconque.

La valeur “undefined”

- La valeur spéciale undefined se distingue des autres. C’est un type à part entière, comme null.
- La signification de undefined est “la valeur n’est pas attribuée”.
- Si une variable est déclarée mais non affectée, alors sa valeur est exactement undefined :

```
let x;
alert(x); // affiche "undefined"
```

- Techniquement, il est possible d’affecter undefined à n’importe quelle variable :

```
let x = 123;
 x = undefined;
alert(x); // "undefined"
```

  - Mais il n’est pas recommandé de le faire. Normalement, null est utilise pour écrire une valeur “vide” ou “inconnue” dans la variable.
  - undefined n’est utilisé que pour les vérifications, pour voir si la variable est affectée ou similaire.

Objects et Symbols

- Le type object est spécial.
- Tous les autres types sont appelés “primitifs”, car leurs valeurs ne peuvent contenir qu’une seule chose (que ce soit une chaîne de caractères, un nombre ou autre).
- Les objets servent à stocker des collections de données et des entités plus complexes.
- Le type symbol est utilisé pour créer des identificateurs uniques pour les objets.

L’opérateur typeof

- L’opérateur typeof renvoie le type de l’argument. Il est utile lorsqu’on souhaite traiter différemment les valeurs de différents types ou de faire une vérification rapide.
- Il supporte deux formes de syntaxe :
  - Sous forme d’opérateur : typeof x.
  - En style de fonction : typeof(x).
- En d’autres mots, cela fonctionne à la fois avec ou sans parenthèses. Le résultat est le même.

typeof

```
1  typeof  undefined  // "undefined"
2
3  typeof  0  // "number"
4
5  typeof  10n  // "bigint"
6
7  typeof  true  // "boolean"
8
9  typeof  "foo"  // "string"
10
11 typeof  Symbol ( "id" )  // "symbol"
12
13 typeof Math // "object"  (1)
14
15 typeof null // "object"  (2)
16
17 typeof alert // "function"  (3)
```

typeof

- Math est un objet interne au langage qui fournit des opérations mathématiques. Il sert uniquement comme exemple d'un objet.
- Le résultat de typeof null est "object".
  - Malgré que ce soit faux. C'est une erreur officiellement reconnue dans typeof, conservée pour compatibilité.
- Le résultat de typeof alert est "function", car alert est une fonction du langage.
- Nous verrons qu'il n'y a pas de type "fonction". Les fonctions appartiennent au type object.
  - typeof les traite différemment, en retournant "fonction". Ce n'est pas tout à fait correct, mais très pratique à l'usage.

Résumé des types

- Il existe 8 types de données de base en JavaScript. (primitif)
  - number: entier ou virgule flottante, les nombres entiers sont limités à $\pm 2^{53}$.
  - bigint: pour des nombres entiers de longueur arbitraire.
  - string: pour les strings. Une chaîne de caractères peut avoir un ou plusieurs caractères, il n'existe pas de type à caractère unique distinct.
  - boolean: pour true/false.
  - null: pour les valeurs inconnues – un type autonome qui a une seule valeur null.
  - undefined: pour les valeurs non attribuées – un type autonome avec une valeur unique undefined.
  - object: pour des structures de données plus complexes.
  - symbol: pour les identifiants uniques.
- L'opérateur typeof nous permet de voir quel type est stocké dans la variable.
  - Deux formes : typeof x ou typeof(x).
  - Renvoie une chaîne de caractères avec le nom du type, comme "string".
  - Pour null il renvoit "object" – C'est une erreur dans le langage, ce n'est pas un objet en fait.

Exercices

```
let nom = "Gilles";
alert( `hello ${1}` ); // ? hello 1
alert( `hello ${"nom"}` ); // ? hello nom
alert( `hello ${nom}` ); // ? hello Gilles
```

Solutions

- Backticks incorpore l’expression à l’intérieur de ${...} dans la chaîne de caractères.

```
let nom = " Gilles ";
// l'expression est un numéro 1
alert( `hello ${1}` ); // hello 1

// l'expression est une chaîne de caractères "nom"
alert( `hello ${"nom"}` ); // hello nom

// l'expression est une variable, il intègre son contenu
alert( `hello ${nom}` ); // hello Gilles
```

Les conversions de types

- La plupart du temps, les opérateurs et les fonctions convertissent automatiquement les valeurs qui leur sont attribuées dans le bon type.
- Par exemple, alert() convertit automatiquement toute valeur en chaîne de caractères pour l'afficher. Les opérations mathématiques convertissent les valeurs en nombres.
- Il y a aussi des cas où nous devons convertir explicitement une valeur pour corriger les choses.

Conversion de chaînes de caractères

- La conversion String se produit lorsque nous avons besoin de la forme chaîne de caractères d'une valeur.
- Par exemple, alert(valeur) le fait pour afficher la valeur.
- Nous pouvons également utiliser un appel de fonction String(valeur) pour ça :

```
let valeur = true;
alert(typeof valeur); // boolean
value = String(valeur); // maintenant la valeur est une chaîne de caractères "true"
alert(typeof valeur); // string
```

- La conversion String est assez évidente. Un false devient "false", null devient "null" etc.

Le signe « = »

- =
- ==
- ===

- https://www.w3schools.com/js/js_comparisons.asp