Labo 2 – Bases du langage Javascript

Exercice 1 : types de valeurs

Une fois Firefox lancé, vous pouvez ouvrir la console grâce au raccourci Ctrl-Shift-K.

Pour tester l'affichage d'une valeur, vous la placerez dans la variable x et utiliserez les lignes suivantes dans la console Firefox.

```
x;                                      // Valeur de x
console.log("Type = " + typeof x);      // Type de x
alert("x = " + x);                      // Écriture de x
console.log("x = " + x);                // Écriture dans la console
```

Dans un premier temps, n'exécutez que la déclaration

```
var x;
```

avant d'entrer les quatre lignes reprises ci-dessus. Cela permettra de repérer comment Javascript fonctionne avec la valeur « undefined ».

Ensuite, faites de même avec les affectations suivantes (considérées une par une).

```
x = 3;
x = 7.18;
x = 3E-46;
x = "Hello";
x = true;
x = false;
```

Cela se passe de la même manière lorsque vous définissez (anonymement ou non) des éléments fonctionnels. Observez ce qu'il se passe si vous renvoyez les quatre lignes indiquées ci-dessus après la définition

```
var x = function (z) { return z + 4 }
```

Ici, « x » désigne une fonction (c'est plutôt un mauvais choix de lettre, mais ça vous permettra de reprendre les 4 lignes présentées plus haut). Vous pouvez le vérifier en demandant (par exemple via la console) à évaluer x(3) ou x(7). Observez ensuite le résultat des quatre lignes données ci-dessus !

Exercice 2 : satanées conversions !

Comme Javascript est un langage non typé, il doit souvent effectuer des conversions « à la volée ». Ces conversions sont régies selon un ensemble de règles dont certaines ont été présentées lors du cours théorique. Le but ici n'est pas de vous faire connaître par cœur ces diverses règles de conversion mais bien de s'assurer que vous comprenez comment elles sont utilisées et quand elles doivent être appliquées.

Ouvrez une nouvelle fenêtre Firefox puis considérez les lignes suivantes.

```
var x;
x += " signifie " + 1 + "défini";
x;
```

Si vous les entrez dans la console, que devraient-elles afficher ? Tentez tout d'abord de répondre par vous-mêmes puis vérifiez votre réponse en utilisant la console.

Une bonne partie des exercices de cette séance portent sur des expressions à évaluer ou des définitions de fonctions dont on vous demande de déterminer les effets. Pour tirer au mieux parti de l'énoncé,

1. *tentez tout d'abord de trouver la réponse en utilisant les slides du cours théoriques et en réfléchissant ;*
2. *vérifiez ensuite votre réponse en entrant le code dans la console Javascript ;*
3. *que votre réponse initiale soit correcte ou non, assurez-vous de comprendre pourquoi Javascript a fourni la réponse donnée !*

Par exemple, il est important de savoir répondre aux questions du style

- *quelles conversions explicites ont été réalisées ?*
- *quelles conversions implicites ont été réalisées ?*
- *quelles règles d'évaluation ont été utilisées ?*

Dans les tests (interrogations et examens) relatifs à ce cours, vous aurez accès à vos notes de cours et à une machine pour déterminer les réponses à ce genre de questions mais il vous sera peut-être demandé de justifier la réponse.

Étape 1

Pour tester les conversions (explicites/forcées) sur la console, vous pouvez utiliser les trois fonctions prédéfinies Number, Boolean et String (qui convertissent respectivement leur argument en un nombre, un booléen ou une chaîne de caractères). Par exemple, pour obtenir la conversion du nombre 1 en booléen, il suffit d'entrer la ligne suivante.

```
Boolean(1);
```

En vous basant sur les notes de cours, tentez de déterminer le résultat des conversions explicites suivantes. Vérifiez ensuite vos prédictions à l'aide de la console !

- À convertir en chaînes de caractères :
  - `undefined`
  - `null`
  - `37.5`
  - `true`
- À convertir en nombres :
  - `undefined`
  - `null`
  - `true`
  - `false`
  - `"\t\t\n"`
  - `"three"`
  - `"42 pommes"`
  - `""`

- À convertir en booléens :
  - `undefined`
  - `null`
  - `"true"`
  - `"false"`
  - `"\t\n\t"`
  - `""`
  - `"0"`

Étape 2

Voici une série d'expressions Javascript mettant en évidence certaines des règles de conversion spécifiques à ce langage. Pour chacune d'entre elles, suivez les étapes suivantes. Considérez chacune des expressions une par une !

1. En vous basant sur les notes du cours, tentez de déterminer le résultat de l'expression.
2. Vérifiez la manière dont Javascript évalue l'expression en utilisant la console.
3. Comparez le résultat au vôtre et, le cas échéant, réexaminez votre réflexion.
4. Dans tous les cas, assurez-vous de bien comprendre quelle(s) règle(s) de conversion a (ont) été appliquée(s), de manière à pouvoir justifier comment on aboutit au résultat.

```
false * "35"
"254" / ""
42 + "1"
"42" + true
"42" + (true + 0)
42 + (true + "0")
42 + true
42 + true + "1"
(true + (2 * true)) * ("35" / 7)
(true + 2 * "5")
1 == true
"0" != false
421 == ("42" + 1)
(25 == "25") + 9
"1" == (0 + true)
(("14" * 2) == ("20" + 8))
true * "0" === false + "1"
420 < ("42" + true * "1")
"43" < "421"
"0" < ("0" * 421 == false)
true || x == y
var nom = "Cécile" ; nom || "John Doe";
var nom = "" ; nom || "John Doe";
var num = 5, denom = 0; denom != 0 && num / denom
var num = 5, denom = 3; denom != 0 && num / denom
```

Exercice 3 : fonctions comme objets de premier ordre

L'idée de considérer les fonctions comme des objets de premier ordre (qu'on peut placer dans une variable, passer en arguments et recevoir comme résultat) sert de base à toute

une série de langages de programmation, des langages dits « fonctionnels ». Bien que Javascript ne soit pas un langage fonctionnel à proprement parler, il fournit de nombreuses situations où utiliser les fonctions comme objets de premier ordre simplifie grandement un code.

Étape 1

Définissez les fonctions suivantes dans un script Javascript directement dans la console ou au sein d'une page HTML vide, selon votre préférence :

- une fonction plus2 qui reçoit une valeur et renvoie cette valeur augmentée de 2 ;
- une fonction fois3 qui reçoit une valeur et renvoie cette valeur multipliée par 3 ;
- une fonction cube qui reçoit une valeur et renvoie cette valeur au cube.

Tant qu'à faire, si vous le désirez, utilisez des méthodes de définition de fonctions différentes pour chacune d'elles. Par exemple : `function…` pour plus2, `var plus3 = function…` pour plus3 et `var cube = new Function…` pour la dernière.

Testez ces fonctions directement dans la console.

Étape 2

Considérez le code suivant et tentez de comprendre à quoi correspond la fonction app2 ainsi définie.

```
function app2 (f,x) { return f(f(x)); }
```

Vérifiez si vous avez bien compris cette définition en déterminant (d'abord mentalement puis à l'aide de la console) la valeur des expressions suivantes.

```
app2(plus2,3)
app2(fois3,2)
app2(cube,2)
```

Étape 3

Définissez une fonction `afficheNFois(s,n)` qui écrit n fois la chaîne de caractères s dans la console (via `console.log`). On supposera que n est un entier positif.

Testez votre code sur deux ou trois exemples. N'oubliez pas de tester le cas où le second argument est 0 !

Note. Il y a au moins 2 manières différentes d'implémenter cette fonction (et les suivantes) : en utilisant une boucle ou en tant que fonction récursive. Vous êtes censés savoir comment coder les deux approches !

Modifiez votre code pour qu'on puisse également appeler la fonction en ne lui donnant qu'un seul argument (la chaîne de caractères). Dans ce cas-là, l'affichage devra être effectué 3 fois. Note : la modification à apporter à la fonction tient en une seule ligne à ajouter !

Testez votre nouveau code sur deux ou trois exemples. Ici encore, n'oubliez pas de tester le cas où on spécifie le second argument et que celui-ci est 0 !

Étape 4

Basez-vous sur le code de la fonction précédente pour écrire une fonction `executeNFois(f,n)` qui exécute `n` fois la fonction/procédure `f` (on supposera que `f` ne prend aucun argument et qu'elle ne renvoie aucun résultat).

Entrez la définition de fonction

```
function message() { console.log("Ceci est un message !"); }
```

puis testez votre code en réalisant l'appel `executeNFois(message,5)`.

Testez aussi votre code en utilisant une fonction anonyme ; par exemple :

```
executeNFois(function () { alert("Bouh !"); } , 3)
```

Question : pourquoi, dans cet appel, n'écrit-on pas `message()` mais juste `message` ?

Modifiez votre code pour qu'on puisse appeler `executeNFois` en omettant le second paramètre, la valeur par défaut étant cette fois-ci 4.

Étape 5

Définissez une fonction `appn` qui prend comme arguments une fonction `f`, un nombre `n` et une valeur `x` et renvoie le résultat obtenu en appliquant `n` fois ($n \geq 0$) la fonction `f` à partir de la valeur `x`.

Vérifiez votre code en effectuant quelques tests. Par exemple : `appn(plus2, 4, 3)` devrait valoir 11 car, si on applique 4 fois la fonction plus2 à 3, on obtient

$$\Big(\big((3+2)+2\big)+2\Big)+2 = 11$$

et `appn(fois3, 3, 1)` devrait valoir 27.

Exercice 4 : Hisse et oh ! Hissez haut !

Les conversions implicites ne constituent pas la seule caractéristique étonnante de Javascript. La gestion des variables locales et globales et le hissage à l'intérieur des définitions de fonctions en sont deux autres. Examinez les bouts de code suivants et, pour chacun d'eux, tentez de déterminer le résultat produit. Vérifiez ensuite vos prédictions en les entrant dans la console Javascript.

1er bout de code

```
function f() {
  alert(x);
  var x = 10;
}
f();
```

2e bout de code

```
function go() {
```

```
  x = 10;
  alert(x);
}
go();
alert(x);
```

3e bout de code

```
function go() {
  alert(x);
  x = 10;
}
go();
alert(x);
```

4e bout de code

```
function boucle() {
  for (var i = 0 ; i < 3 ; i++)
    alert(i);
  alert(i);
}
boucle();
```