Labo 2C : variables et valeurs

Objectifs

- Appliquer les notions de scopes et comprendre les différences entre les diverses manières de déclarer des variables en Javascript via plusieurs exemples pratiques.
- Construire un bout de code qui permet de valider une entrée numérique (application pratique des conversions)
- Être capable de mettre en évidence les notions théoriques qui permettent d'expliquer et de corriger certains problèmes liés aux conversions automatiques
- Être capable d'utiliser les opérateurs booléens pour implémenter judicieusement des arguments optionnels

Exercice 1 : Les opérations || et &&

En utilisant ce que vous savez sur les conversions implicites et le fonctionnement des opérateurs || et &&, prédisez le résultat de chacun des bouts de code suivants. Vérifiez ensuite vos prédictions en utilisant la console Firefox.

```
true || x == y   true

let nom = "Cécile";
nom || "John Doe"; Cécile

let nom = "";
nom || "John Doe"; John Doe

let num = 5, denom = 0;
denom != 0 && num / denom   false

let num = 5, denom = 3;
denom != 0 && num / denom   num / denom
```

Exercice 2 : Hisse et oh ! Hissez haut !

Examinez les cinq 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.

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

```
function go() {
  alert(x);
  x = 10;  — > JS sors dès qu'il y a une erreur
}
go();   erreur
alert(x); erreur
```

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

```
function boucle() {
  for (var i = 0 ; i < 3 ; i++)
    alert(i); 0,1,2
  alert(i); 3
}
boucle();
```

```
function boucle() {
  for (let i = 0 ; i < 3 ; i++)
    alert(i); 0,1,2
  alert(i);  erreur
}
boucle();
```

Exercice 3 : Déclarations diverses

On considère le code Javascript suivant, où ___1___ et ___2___ dénotent des bouts de code qui seront déterminés plus bas. Que va-t-il afficher dans chacun des huit cas suivants ? À nouveau, prévoyez le résultat avant de vérifier la réponse via la console.

```
var x = 3;
function changeX (action) {
  if (action) ___1___ x = Math.random();
  return x;
}
console.log(changeX(___2___));
x;
```

__1__	___2___ devient false	___2___ devient true

(rien)	`var x = 3;`
	`function changeX (action) {`
	`  if (action) {`
	`  x = Math.random();`
	`  }`
	`  return x;`
	`}`
	`console.log(changeX(false));` 3
	`x;` 3
		`var x = 3;`
		`function changeX (action) {`
		`  if (action) {`
		`  x = Math.random();`
		`  }`
		`  return x;`
		`}`
		`console.log(changeX(true));` /random/
		`x;` /random/

var	`var x = 3;`
	`function changeX (action) {`
	`  if (action) {`
	`  var x = Math.random();`
	`  }`
	`  return x;`
	`}`
	`console.log(changeX(false));`
	`x;` 3 undefined
		`var x = 3;`
		`function changeX (action) {`
		`  if (action) {`
		`  var x = Math.random();`
		`  }`
		`  return x;`
		`}`
		`console.log(changeX(true));` /random/
		`x;` 3

let	`var x = 3;`
	`function changeX (action) {`
	`  if (action) {`
	`  let x = Math.random();`
	`  }`
	`  return x;`
	`}`
	`console.log(changeX(false));` 3
	`x;` 3
		`var x = 3;`
		`function changeX (action) {`
		`  if (action) {`
		`  let x = Math.random();`
		`  }`
		`  return x;`
		`}`
		`console.log(changeX(true));` 3
		`x;` 3

const	`var x = 3;`
	`function changeX (action) {`
	`  if (action) {`
	`  const x = Math.random();`
	`  }`
	`  return x;`
	`}`
	`console.log(changeX(false));` 3
	`x;` 3
		`var x = 3;`
		`function changeX (action) {`
		`  if (action) {`
		`  const x = Math.random();`
		`  }`
		`  return x;`
		`}`
		`console.log(changeX(true));` 3
		`x;` 3

Exercice 4 : Vérification de type

Définissez une fonction Javascript `demandeNombre()` qui demande un nombre à l'utilisateur (via la fonction prompt) puis l'affiche dans la console. La fonction reformulera la demande jusqu'à ce qu'elle ait bien reçu un nombre. Notez que « prompt » renvoie toujours une chaîne de caractères correspondant à ce que l'utilisateur a tapé !

Note. Pour les perfectionnistes qui voudraient également interdire les réponses ne contenant que des blancs (espaces, tabulations, …), si `str` est une chaîne de caractères, `str.trim()` est la chaîne de caractères obtenue en supprimant tous les blancs situés au début et à la fin de `str`.

Modifiez ensuite la fonction en `demandeNombre(min,max)` qui, en plus, impose que le nombre entré par l'utilisateur se trouve entre `min` et `max`.

```
function demandeNombre(min, max){
    do{
        var nombreObtenu = parseInt(prompt(`Donnez un nombre entre ${min} et ${max}: `));
    }while(!isFinite(nombreObtenu) || nombreObtenu < min || nombreObtenu > max);

    console.log(nombreObtenu);
}
```

Exercice 5 : Question d'examen [janvier 2015]

1. Un professeur de mathématiques décide d'utiliser Javascript pour tester ses étudiants. Dans un premier temps, voici le code qu'il produit.

```
var reponse = prompt("Que vaut 36 * 0 ?");
if (reponse == 0)
  alert("Bravo !");
else
  alert("T'es nul !");
```

Il teste son programme : il entre tout d'abord la bonne réponse (0) et constate que ça fonctionne ; puis il entre une réponse fausse (36) et remarque que l'affichage attendu se produit. Fier de lui, il demande à son épouse de tester le programme à son tour.

Celle-ci accepte mais, voyant qu'il s'agit d'une question de mathématique, refuse d'aller plus loin. Elle clique sur « Ok » sans entrer de réponse et… obtient « Bravo ! » comme affichage. Pourquoi ?

Car javascript ici fait de la conversion implicite. Un prompte renvoie tjr une chaine de caractère. Or il cherche à la comparer avec un nombre. Lors d'une comparaison de type « == », si javascript détecte une différence de type, il fait une conversion implicite. Or, une Nombre(« ») donnera 0. Ce qui donne du coup la bonne réponse. Il aurait fallut faire parseInt

2. Pour tenter de corriger son erreur, le professeur de mathématiques revoit son code et modifie la condition en ce qui suit :

```
if (reponse === 0)
```

Et là, c'est le drame : dès le premier test, ça buggue. En effet, lorsqu'il entre la réponse correcte (0), le programme affiche « T'es nul ! ». Pourquoi ?

Car dans une comparaison de type « === », javascript vérifie d'abord et avant tout la compatibilité des types. Si dès le début les deux variables comparées ne sont pas de même type, la comparaison renvoie faux sans même tenter une conversion. Pour rappel, le prompte renvoie une chaine de caractère. Donc même si on rentre 0, et bien « 0 » === 0 ? sera donc égal à faux.

3. Comment corriger le code afin qu'il produise l'effet attendu dans tous les cas ?

```
var reponse = parseInt(prompt("Que vaut 36 * 0 ? »));
if (reponse == 0)
  alert("Bravo !");
else
  alert("T'es nul !");
```

Exercice 6 : Question d'examen 2 [janvier 2016]

1. [Vrai/faux] En Javascript, les deux variables suivantes contiennent des valeurs de même type.

```
var pi = 3.14;
var vendredi = 13;
```

Vrai. Le type number

2. Considérez le bout de code suivant, où on suppose que l'utilisateur rentre deux valeurs entières comprises (au sens large) entre 1 et 100 et qui est censé indiquer si la première est plus grande ou plus petite que la seconde.

```
var nb1 = prompt("Entrez le premier nombre.");
var nb2 = prompt("Entrez le second nombre.");
console.log ("Le " + (nb1 >= nb2 ? "1er" : "2e")
             + " est plus grand (ou égal).");
```

On suppose que le premier nombre entré est 47. Ce bout de code affichera-t-il toujours la bonne réponse ? Si oui, indiquer « Oui ». Si non, donner un exemple de second nombre à entrer pour avoir un résultat incorrect.

Non, car quand on compare 2 chaines de caractère, c'est l'ordre lexicographique qui compte. Dans ce cadre là, « 100 » est plus petit que « 47 ».
–> Rentrer 100 en deuxième réponse donnera un résultat érroné

3. Pour calculer le prix d'un article potentiellement soldé, on définit la fonction suivante en Javascript, dont le second argument est prévu pour être optionnel.

```
function prixReduit (prix, reductionPourCents) {
  reductionPourCents = reductionPourCents || 10;
  rabais = prix * reductionPourCents / 100;
  return (prix – rabais);
}
```

Critiquez ce code en mettant en évidence ses (deux) problèmes. Puis proposez une version corrigée.

1 - l'opérateur || ne prends pas en compte les falsy value, dont le 0. Pour palier ce problème, on utiliser le Nullish coalescing operator (??).

2 - Ne jamais déclarer sans rien une variable, sinon elle sera considéré globale et hissé en dehors de la fonction. Il faut donc déclarer rabais avec un LET.

3 (bonus) - On peut donner une valeur par défaut dans les arguments d'une fonction au cas où elle est undefined.

```
function prixReduit (prix, reductionPourCents = 10) {
    //reductionPourCents = reductionPourCents ?? 10 ;
    let rabais = prix * reductionPourCents / 100;
    return (prix - rabais);
}
```