Henallux – DA/IA2 – Technologies web

Laboratoire 2

Variables et valeurs

Sommaire

[1. Les opérations || et && 2](#_Toc178005249)

[2. Hisse et oh ! Hissez haut ! 3](#_Toc178005250)

[3. Déclarations de variables 5](#_Toc178005251)

[4. Vérification de type 6](#_Toc178005252)

[5. Question d’examen [janvier 2015] 7](#_Toc178005253)

[6. Question d’examen 2 [janvier 2016] 8](#_Toc178005254)

[7. Question d’examen 3 [janvier 2016] 9](#_Toc178005255)

# Les opérations || et &&

Objectifs

* Se familiariser avec les conversions entre types pour les opérations  
  booléennes

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

let nom = "Cécile";

nom || "John Doe";

let nom = "";

nom || "John Doe";

let num = 5, denom = 0;

denom != 0 && num / denom

let num = 5, denom = 3;

denom != 0 && num / denom

# Hisse et oh ! Hissez haut !

Objectifs

* Se familiariser avec le hoisting / hissage
* Se familiariser avec les différents de déclarations de variables

Examinez les dix 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.

## Code 1

function f() {

alert(x);

var x = 10;

}

f();

## Codes 2 et 3

|  |  |
| --- | --- |
| function go() {  alert(x);  x = 10;  }  go();  alert(x); | function go() {  x = 10;  alert(x);  }  go();  alert(x); |

## Codes 4 et 5

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

## Codes 6 et 7

|  |  |
| --- | --- |
| var x = 7;  function f () {  var x = 2;  x++;  console.log(x);  }  console.log(x);  f();  console.log(x); | var x = 7;  function f () {  x = 2;  x++;  console.log(x);  }  console.log(x);  f();  console.log(x); |

## Codes 8 et 9

|  |  |
| --- | --- |
| function f () {  var x = 5;  if (x == 5) {  var x = 7;  console.log (x);  }  console.log (x);  }  f(); | function f () {  var x = 11;  function g () {  console.log (x);  var x = 13;  }  g();  console.log (x);  }  f(); |

## Code 10

var x = 17;

function f (z) {

if (z)

var x = 4;

console.log (x);

}

f(true);

f(false);

# Déclarations de variables

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

# 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 !

|  |  |
| --- | --- |
| Entrées que la fonction devrait accepter | Entrées que la fonction devrait refuser |
| 17  -54 | hello  31-4  7nains  (chaîne vide)  □□□ (trois espaces)  □□□17 (espaces avant un nombre) |

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.

# 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 ?

Parce qu’elle obtient «vide » est que «vide» en nombre se traduit par 0 => Conversion implicite.

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 ?

Parce que quand l’utilisateur entre 0 avec le prompt le programme reçoit une chaine de caractère mais avec le === on fait une comparaison stricte du type et de la valeur, mais « 0 » n’est pas égal à 0

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

# 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 en JavaScript les deux variables sont de 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 si on rentre 5 le résultat sera incorrect, car le prompt renvoie deux chaines de caractères et ces chaines sont comparer dans l’ordre lexicographique. Pour faire une comparaison correct, il faut transformer les deux entrée en Number.

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.

# Question d’examen 3 [janvier 2016]

1. On considère le code ci-dessous. Qu’affichera la console ? (Vous pouvez tout écrire sur une seule ligne).

// Définition 1

function f(x) { console.log(x+7); }

// Appel 1

f(2);  
  
// 5

// Définition 2

function f(x) { console.log(x+3); }

// Appel 2

f(3);  
  
// 6

2. Si, dans la question précédente, on écrit les deux définitions de fonctions sous la forme

var f = function (x) { ... }; //9 et //6

qu’affichera la console ?

3. Les réponses aux deux questions précédentes sont différentes. C’est dû à un principe qui, en JavaScript, s’applique un peu différemment aux définitions de fonctions et aux déclarations/initialisations de variables. Quel est le nom de ce principe ?

Le Hoisting.