

LIFAP5 – Programmation fonctionnelle pour le WEB

Contrôle Continu Intermédiaire – durée 40'

20

Licence informatique UCBL – Printemps 2016–2017

Nom : THION

Prénom : Romuald

Groupe TD : 7

Aucun document autorisé. Le barème est indicatif. Les réponses doivent être données sur la feuille.
Les réponses aux questions ouvertes sont courtes.

19

Exercice 1 : Que fait ce programme ?

```
1 function () {
2     function foo(m) {
3         var f = (x) => (y) => (Math.pow(y, x));
4         //Math.pow(a,b) élève 'a' à la puissance 'b'
5         let arr = [];
6
7         for (let i = 0; i < m; ++i)
8             arr[i] = f(i);
9
10        return arr;
11    }
12
13    var arr = foo(8);
14    console.log(arr.map((f) => f(2)));
15 }();
```

Indiquer comment s'appelle la technique utilisant la fonction anonyme de la ligne 1

1 Immediate-Invoked Function Expression - IIFE

Donner le nombre de paramètres de la fonction f de la ligne 3

1 seul : x

Donner la portée (donner l'intervalle des lignes) de la variable arr de la ligne 5

l₂ à l₁₁ : la fonction foo

Donner la portée (donner l'intervalle des lignes) de la variable i de la ligne 7

l₇ à l₁₈ : le bloc du for se réduit à une seule instruction

Donner la portée (donner l'intervalle des lignes) de la variable arr de la ligne 13

l₁ à l₁₅, mais avec masquage entre l₂ et l₁₁

1/2

Réécrire la fonction `f` de la ligne 3 en utilisant la syntaxe C, sans utiliser la notation `=>` de ES6

```
var f = function f(x) {
    return function(g) {
        return Math.pow(g, x);
    }
}
```

curryfié!

Donner le résultat de l'exécution du programme

1/2

$[2^0=1, 2, 4, 8, 16, 32, 64, 128]$

(16)

Exercice 2 : Variations sur la factorielle

On rappelle la définition inductive de la factorielle d'un nombre entier positif n notée $n!$: $0! = 1$ et $n! = n \times (n - 1)!$

1. Écrire une fonction *récursive* `fact_rec(n)` qui calcule $n!$, sans utiliser de boucle `for`

1/2

```
function fact_rec(n) {
    if (n < 2)
        return 1;
    else
        return n * fact_rec(n-1);
}
```

2. Écrire une fonction *non récursive* `fact_for(n)` qui calcule $n!$, en utilisant une boucle `for`

1/2

```
function fact_for(n) {
    var r = 1;
    for (let i = 1; i <= n; i++) {
        r = r * i;
    }
    return r;
}
```

3. Soit un tableau `arr = [1, 2, ..., n]` contenant les entiers de 1 à n . Calculer $n!$ en utilisant la méthode `reduce` sur ce tableau.

1/2

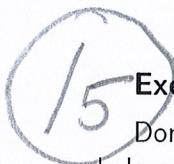
```
var "n!" = arr.reduce((acc, x) => (acc * x), 1);
```

7.2.13 Abstract Equality Comparison

The comparison $x == y$, where x and y are values, produces true or false. Such a comparison is performed as follows:

1. If $\text{Type}(x)$ is the same as $\text{Type}(y)$, then
 - a. Return the result of performing Strict Equality Comparison $x === y$.
2. If x is null and y is undefined, return true.
3. If x is undefined and y is null, return true.
4. If $\text{Type}(x)$ is Number and $\text{Type}(y)$ is String, return the result of the comparison $x == \text{ToNumber}(y)$.
5. If $\text{Type}(x)$ is String and $\text{Type}(y)$ is Number, return the result of the comparison $\text{ToNumber}(x) == y$.
6. If $\text{Type}(x)$ is Boolean, return the result of the comparison $\text{ToNumber}(x) == y$.
7. If $\text{Type}(y)$ is Boolean, return the result of the comparison $x == \text{ToNumber}(y)$.
8. If $\text{Type}(x)$ is either String, Number, or Symbol and $\text{Type}(y)$ is Object, return the result of the comparison $x == \text{ToPrimitive}(y)$.
9. If $\text{Type}(x)$ is Object and $\text{Type}(y)$ is either String, Number, or Symbol, return the result of the comparison $\text{ToPrimitive}(x) == y$.
10. Return false.

FIGURE 1 – Extrait du standard javascript ecma-262/7.0, section 7.2.13



Exercice 3 : == et !=

Donner la valeur à laquelle s'évaluent les tests suivants, justifier la réponse en précisant les cas de la section 7.2.13 du standard javascript¹ utilisés :

1. $\text{null} == \text{undefined}$
2. $2 == \text{true}$
3. $\text{true} == "1"$
4. $("0" + "1") == \text{true}$

- /1 1. true : cas 2
- /1 2. false : cas 7, puis cas 1
- /1 3. true : cas 6, puis cas 4, puis cas 1
- /1 4. true : d'abord " 0 " + " 1 " vaut " 01 ", donc on compare " 01 " == true : cas 7, puis cas 5, puis cas 1

Dans le guide de style javascript de Airbnb, section *Comparison Operators & Equality*², vous lisez la recommandation suivante : « 15.1 Use `==` and `!=` over `==` and `!=`. eslint: eqeqeq ».

Motiver la règle 15.1 du guide de style

/1 La procédure est complexe, décalable et sujette à erreurs
⇒ il vaut mieux ne pas laisser l'interpréteur convertir en utilisation `==` dont la sémantique est claire.

1. <http://www.ecma-international.org/ecma-262/7.0/index.html#sec-abstract-equality-comparison>

2. <https://github.com/airbnb/javascript#comparison-operators--equality>

