LIFAP5 – Programmation fonctionnelle pour le WEB
Contrôle Continu Intermédiaire – durée 40'

Licence informatique UCBL – Printemps 2016–2017

Nom :

Prénom :

Groupe TD :

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

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

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

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

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

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

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

Donner le résultat de l'exécution du programme

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

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 Type(x) is the same as 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 Type(x) is Number and Type(y) is String, return the result of the comparison x == ToNumber(y).
5. If Type(x) is String and Type(y) is Number, return the result of the comparison ToNumber(x) == y.
6. If Type(x) is Boolean, return the result of the comparison ToNumber(x) == y.
7. If Type(y) is Boolean, return the result of the comparison x == ToNumber(y).
8. If Type(x) is either String, Number, or Symbol and Type(y) is Object, return the result of the comparison x == ToPrimitive(y).
9. If Type(x) is Object and Type(y) is either String, Number, or Symbol, return the result of the comparison 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 `javascript`[1] utilisés :

1. `null == undefined`
2. `2 == true`
3. `true == "1"`
4. `("0" + "1") == true`

Dans le guide de 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. http://www.ecma-international.org/ecma-262/7.0/index.html#sec-abstract-equality-comparison
2. https://github.com/airbnb/javascript#comparison-operators--equality