

LIFAP5 – Programmation fonctionnelle pour le WEB

Examen – durée 60'

Licence informatique UCBL – Printemps 2016–2017

Seule une feuille A4 recto-verso manuscrite est autorisée comme document. Le barème est indicatif. Les réponses aux questions ouvertes sont courtes.

Exercice 1 : Réécriture avec `Array.reduce` (/6)

On rappelle que la méthode `Array.reduce(fn, z)` qui prend en paramètre une fonction binaire `fn` et un élément `z` calcule `fn(...(fn(fn(z, x0), x1), ...), xn)` pour un tableau `[x0, x1, ..., xn]`. On rappelle que la méthode `Array.concat(arr)` permet de concaténer un tableau à celui où est appelé `concat`, par exemple, `[0,1,2].concat([3])` produit `[0,1,2,3]`.

1. La méthode `Array.filter(f)` produit un nouveau tableau ne gardant que les éléments qui satisfont le prédicat `f`, par exemple `[0,1,2].filter(x => x > 0)`; produit le tableau `[1,2]`. Soit `arr` un tableau et `f` une fonction donnés, réécrire `arr.filter(f)` en utilisant `Array.reduce`. (/2)
2. La méthode `Array.join(str)` transforme un tableau en chaîne de caractère en séparant les éléments du tableau par le paramètre `str`, par exemple `[0,1,2].join(',')` produit la chaîne `"0,1,2"`. Soit `arr` un tableau et `str` une chaîne donnés, réécrire `arr.join(str)` en utilisant `Array.reduce`. (/2)
3. Écrire une fonction `flatten` qui transforme un tableau de tableaux en les concaténant tous les uns à la suite des autres en utilisant `Array.reduce`. Par exemple, `flatten([[0,1,2], [], [3,4]])` doit produire `[0,1,2,3,4]`. (/2)

Exercice 2 : Du λ -calcul au JavaScript et vice-versa (/8)

1. Réécrire l'expression du λ -calcul $\lambda f.\lambda x.f(fx)$ en JavaScript en utilisant la notation `=>`. (/1)
2. Réécrire la fonction JavaScript `let fn = f => g => x => f(g(x))` en λ -calcul. (/1)
3. Expliquez brièvement en français ce que calcule la fonction `fn`. (/1)
4. Soient les fonctions JavaScript suivantes `let f0 = x => 2 + x` et `let f1 = x => 2 * x`. Calculer `fn(f0)(f1)(3)`. (/1)
5. Réécrire en JavaScript une version β -réduite (plus simple) de `fn(f0)(f1)` avec les fonctions `f0` et `f1` de la question précédente. (/1)
6. Considérons la fonction JavaScript `let w = x => x(x)`. L'exécution de `w(x => x + 1)` produit le résultat `"x => x + 11"`. Expliquez. (/1)
7. En utilisant `Array.reduce` et `fn` définies précédemment, écrire une fonction `comp` qui prend en entrée un tableau de fonctions unaires et les compose toutes. Par exemple, `comp([fn0, fn1, fn2])(x)` calcule `fn0(fn1(fn2(x)))`. Quand le tableau est vide, c'est la fonction identité $(\lambda x.x)$ qui est renvoyée. (/2)

Exercice 3 : Programmation asynchrone (/6)

Dans cet exercice, on souhaite modifier une page HTML en y insérant un contenu généré à partir d'un fichier json contenant des billets téléchargé de façon asynchrone comme dans le TP2.

Le contenu du fichier json est un tableau d'objets contenant trois champs, comme dans l'exemple suivant :

```
[ { "titre": "Faits marquants 2016 du laboratoire LIRIS",
  "date": "2017-02-10",
  "contenu": "Le livret des faits marquants 2016..."
},
...
{ "titre": "Portrait d'un étudiant du master...",
  "date": "2016-10-25",
  "contenu": "Levent Acar, diplômé 2011 du master..."
}]
```

L'extrait pertinent de la page HTML qui charge votre fichier JavaScript est donné ci-dessous :

```
1 <h2 id="exo">Exercice : programmation asynchrone</h2>
2 <input type="button" id="exo-btn" value="Charger"/></p>
3 <div id="exo-output"></div>
```

Votre fichier JavaScript de départ est donné ci-dessous :

```
1 function charge() {
2   return new Promise(function(resolve, reject) {
3     let request = new XMLHttpRequest();
4     request.open("GET", './billets.json');
5     request.overrideMimeType("text/json");
6     request.onload = function() {
7       if (request.status === 200)
8         resolve(JSON.parse(request.response));
9       else
10        reject(Error("URL [./billets.json]; error code:" + request.
11          statusText));
12    };
13    request.onerror = () => reject(Error("Network error on [./billets.
14      json]."));
15    request.send();
16  });
17 }
18 document.addEventListener('DOMContentLoaded', function(){
19   document.getElementById("exo-btn").onclick = function() {
20     afficheBilletsJSON();
21   };
22 }, false);
23 function renduBillet(listeBillets){ /* TODO */ }
24 function afficheBilletsJSON(){ /* TODO */ }
```

1. Expliquer brièvement en français ce que fait la fonction `charge()` (/1)
2. Expliquer brièvement en français ce que fait l'appel de la ligne 17. (/1)
3. Définir la fonction `renduBillet` qui prend une liste de billets au format json (comme dans l'exemple ci-dessus) et génère le code HTML permettant d'afficher les billets *dont le contenu est non vide*. On générera une liste non-ordonnée (`...` en HTML) contenant un billet par item (`...`). La mise en forme du billet n'est pas importante tant que son intégralité est affichée. (/3)
4. Définir la fonction `afficheBilletsJSON` qui appelle `charge` et affiche dans le bloc `exo-output` la liste des billets en utilisant `renduBillet` pour générer le HTML. (/1)