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 contenants 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.
               statusText));
11     };
12     request.onerror = () => reject(Error("Network error on [./billets.
           json]."));
13     request.send();
14   });
15 }
16
17 document.addEventListener('DOMContentLoaded', function(){
18   document.getElementById("exo-btn").onclick = function() {
19     afficheBilletsJSON();
20   };
21 }, false);
22
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 génèrera une liste non-ordonnée (`<ul>...</ul>` en HTML) contenant un billet par item (`<li>...</li>`). 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)

Corrections

Solution de l'exercice 1

1. `arr.reduce( (acc,x)=> acc.concat( ( f(x)? [x] : [] )), [] )`
2. `arr.reduce((acc,x)=> !acc ? x : acc + ','+ x, "")`
3. `var flatten = arr => arr.reduce((acc,x)=> acc.concat(x),[])`

Solution de l'exercice 2

1. `let ex = f => x => f(f(x));`
2. $\lambda f.\lambda g.\lambda x.f(gx)$
3. C'est la composition fonctionnelle notée usuellement $(f \circ g)$
4. Le résultat après réduction est $2 + (2 * 3)$ soit 8
5. On obtient `x => 2 + (2 * x)` d'après les réductions suivantes :

$$
\begin{aligned}
&(\lambda f.\lambda g.\lambda x.f(gx))(\lambda y.2+y)(\lambda z.2*z)\\
\rightarrow&(\lambda g.\lambda x.(\lambda y.2+y)(gx))(\lambda z.2*z)\\
\rightarrow&(\lambda x.(\lambda y.2+y)((\lambda z.2*z)x))\\
\rightarrow&(\lambda x.(\lambda y.2+y)(2*x))\\
\rightarrow&(\lambda x.2+(2*x))
\end{aligned}
$$

6. `w(x => x +1)` se réduit en `(x => x +1)(x => x +1)` puis en `(x => x +1)+1`. Le + à l'extérieur ne peut pas être une addition car le membre gauche est la fonction `x => x +1`. Le terme `x => x +1` est alors *implicitement converti en chaîne de caractère* `"x => x +1"` tout comme le membre droit qui devient `"1"` et l'opération + devient la concaténation de `"x => x +1"` et `"1"` pour le résultat obtenu.
7. `let comp = fs => fs.reduce((acc,x)=> fn(acc)(x), x => x)`. On remarque que `let comp = fs => fs.reduce(fn, x => x)` ne fonctionne pas car le premier argument attendu par `reduce` n'est *pas* curryfié.

Solution de l'exercice 3

1. Elle fait un appel ajax asynchrone pour charger le fichier `./billets.json` et renvoyer une promesse de son contenu parsé.
2. Il attend que le HTML soit complètement chargé pour associer la fonction `afficheBilletsJSON` à un clic sur le bouton `exo-btn` de la page.
3.

```
function renduBillet(listeBillets){
  let str = "<ul>";
  str += listeBillets
    .filter(x => !!x.contenu)
    .map(x => `${x.titre} (${x.date}) : ${x.contenu}`)
    .map(x => `<li>${x}</li>`)
    .join("\n") ;
  str += "</ul>";

  return str;
}
```

4.

```
function afficheBilletsJSON(){
  charge()
  .then(json => {
    document.getElementById("exo-output").innerHTML = renduBillet(
        json);
  })
}
```