LIFAP5 – Programmation fonctionnelle pour le WEB
TD3/4 – programmation asynchrone en `javascript`

Licence informatique UCBL – Printemps 2018–2019

Exercice 1 : Décorateurs

Un décorateur est une fonction qui "modifie le comportement d'une autre fonction". Plus précisément, c'est une fonction d qui prend une fonction f en argument, telle que $d(f)$ est la fonction f dont le comportement est modifié. Donner *le type et le code* pour les décorateurs suivants :

`maybe(f,v)` appelle `f` et si `f` renvoie `undefined`, alors renvoie `v` à la place.

`once(f)` au premier appel, renvoie le résultat renvoyé par `f` et renvoie ce même résultat, quels que soient les arguments passés en paramètres, pour les appels suivants (sans rappeller `f`).

`memoize(f)` si `f` a déjà été appelée avec le même argument, renvoie directement [1] la valeur retournée précédemment par `f` pour cet argument. On supposera que `f` ne prend qu'un seul argument qui peut être utilisé come clé d'un dictionnaire `javascript`. On rappelle que la méthode `hasOwnProperty` permet de tester si un objet possède un champ correspondant à une certaine clé.

Exercice 2 : Calcul asynchrone avec des promesses

```
1  function make_promise(str, time, ok=true){
2    if(ok)
3      return new Promise((resolve, reject) => {
4        setTimeout(() => resolve(str), time);
5      });
6    else
7      return new Promise((resolve, reject) => {
8        setTimeout(() => reject(str), time);
9      });
10   }
11 let p1 = make_promise("P1", 1000, true);
12 let p2 = make_promise("P2",  500, true);
13 let p3 = make_promise("P3",  750, false);
14 // p1.then(console.log); p2.then(console.log); p3.catch(console.log);
```

1. Que fait la fonction `make_promise` ?
2. Qu'affiche le programme sur la console ?
3. Qu'affiche le programme sur la console si on lui ajoute à la fin `p1.then(console.log); p2.then(console.log); p3.catch(console.log);`
4. Une seconde après l'exécution précédente, on exécute `p1.then(console.log); p2.then(console.log); p3.catch(console.log);`. Que s'affiche-t-il ?
5. Qu'affiche `make_promise("PA", 100).then((x)=> make_promise("PB", 200)).then(console.log);`

1. *i.e.* sans rappeler `f`

Exercice 3 : Appel asynchrone et rendu

On souhaite créer une fonction utilitaire `chargeInsere` pour faciliter l'insertion de contenu obtenu via un échange réseau `javascript` avec le serveur. La fonction `chargeInsere` prendra une fonction `rendu` en argument et renverra une fonction. La fonction retournée par `chargeInsere(rendu)` prendra deux arguments : une URL (`url`) et l'identifiant (`id`) d'un élément HTML. La fonction `rendu` passée en paramètre est une fonction de rendu qui attend un objet JSON et renvoit une chaîne de caractère contenant du code HTML pour présenter les données de l'objet JSON.

La fonction `chargeInsere(rendu)` aura le comportement suivant :

1. récupérer le contenu JSON (de type `string`) situé à l'adresse `url` avec `chargeDonnees` ou `fetch` données en exemple ;
2. appeler `rendu` avec l'objet JSON (de type `object`) ;
3. insérer le texte obtenu dans l'élément identifié par `id`.

On rappelle l'existence des fonctions / champs suivants :

— `fetch(url)` télécharge de manière asynchrone le contenu disponible à l'adresse `url` et renvoie une *promesse* de la réponse HTTP ;
— `document.getElementById(id)` retourne l'élément HTML identifié par `id` ;
— Si `elt` est un élément, alors `elt.innerHTML` est un champ qui représente son contenu HTML sous forme d'une chaîne de caractères.

On rappelle également le code suivant du TP3 qui donne un exemple d'utilisation de `fetch` :

```
1 function chargeDonnees(url, callback) {
2   fetch(url)
3     .then(response => response.json())
4     .then(data => callback(data));
5 }
```