Flux de données et programmation — partiel

1 Déconnecté en salle d'examen (1h30, 14 points)

1.1 Reactjs 4 points

Question A. Décrire React.js. En dix à vingt phrases, décrire ce qu'est Reactjs à quelqu'un qui connaît les principaux langages de programmation et le développement web. Décrire son principe de fonctionnement, ses composants et parler du surcoût en temps de calcul d'un tel fonctionnement.

1.2 Scala 5 points

Question B. Appel en Scala. Dire précisément ce qu'affichera le code suivant et pourquoi :

```
def x = {println("Bonjour Toto");1}
val y = {println("Hello");2}
lazy val z = {println("hi");3}
x + x
y + y
z + z
```

Question C. For en Scala. On veut définir une fonction `foo` qui prend en entrée deux listes d'entiers `xs` et `ys` et un entier `cible` et trouve toutes les paires `x`, `y` où `x` est un élément de `xs` et `y` est un élément de `ys` telles que `x + y = cible`. Définir `foo` en effectuant tout le traitement dans un `for`. Puis redéfinir `foo` une deuxième fois (l'appeler `foo2`), en effectuant tout le traitement à l'aide de `map`, `flatMap`, `filter`.

Exemple :

```
scala>  val xs = List(3, 4, 6)
xs: List[Int] = List(3, 4, 6)

scala> val ys = List(4, 6, 7)
ys: List[Int] = List(4, 6, 7)

scala> foo(xs, ys, 10)
res4: List[(Int, Int)] = List((3,7), (4,6), (6,4))
```

Question D. For en Scala (encore). On veut définir une fonction `bar` qui prend en entrée un *stream* `xss` de liste d'entiers et retourne le stream des entiers plus grands que 1 qui appartiennent à des listes de `xss` ayant au moins 3 éléments. Définir `bar` en effectuant tout le traitement dans un `for`. Puis redéfinir `bar` une deuxième fois (l'appeler `bar2`), en effectuant tout le traitement à l'aide de `map`, `flatMap`, `filter`.

Exemple :

```
scala> val xss = Stream(List(1,2,3),List(4,5,1), List(6,7), List(), List(8, 0, 9))
xss: scala.collection.immutable.Stream[List[Int]] = Stream(List(1, 2, 3), ?)

scala> xss.toList
res0: List[List[Int]] = List(List(1, 2, 3), List(4, 5, 1), List(6, 7), List(), List(8, 0, 9))

scala> bar(xss)
res1: scala.collection.immutable.Stream[Int] = Stream(2, ?)

scala> bar(xss).toList
res2: List[Int] = List(2, 3, 4, 5, 8, 9)
```

1.3 Try 5 points

Le code OCaml suivant définit un constructeur de type `mtry` et deux fonctions `return` et `bind`.

```
exception My_error of string

type 'a mtry = | Some of 'a
               | Error of string

let return a = Some a

let bind ma f = match ma with
  | Error msg -> Error msg
  | Some a -> try f a
    with My_error s -> Error s
```

Si cela vous facilite la tâche, vous pouvez utiliser les deux opérateurs infixes dérivés de bind `>>=` et `>=>` définis de la façon suivante :

```
let (>>=) ma f = bind ma f
let (>=>) f g = function a -> (f a) >>= g
```

Question E. Exemple 1. Que retourne `return 42` ?

Question F. Exemple 2. En supposant que `example_f` est définie comme suit, que retourne `bind (return 42) example_f` ? Et `bind (return 0) example_f` ?

```
let example_f a = match a with
  | 0 -> Error "division par zero"
  | n -> Some (42 / n)
```

Question G. Exemple 3. En supposant que `example_g` est définie comme suit, que retourne `bind (return 42) example_g` ? Et `bind (return 0) example_g` ?

```
let example_g a = match a with
  | 0 -> raise (My_error "division par zero")
  | n -> Some (42 / n)
```

Question H. Monade. Que faut-il vérifier pour que `mtry`, `return`, `bind` forment une monade ? Selon vous, est ce que `mtry`, `return`, `bind` forment une monade ?

2 Connecté (rendez-vous en salle G212, 1h00, 6 points)

2.1 Examen papier testé sur machine

Vous pouvez tenter d'améliorer votre note sur la partie déconnectée de l'examen en refaisant certains exercices sur machine et en donnant vos nouvelles réponses sur papier.

2.2 Exercice mystère, 6 points

Reste du sujet communiqué en salle TP.