Flux de données et programmation — partiel

1 Indications

Durée : 2h. Aucun document autorisé. Les questions sont numérotées par des lettres majuscules, vous pouvez répondre dans le désordre. Vous pouvez choisir trois questions auxquelles vous ne voulez pas répondre. Indiquer lesquelles sur la première page de votre copie.

2 Questions de cours

2.1 Javascript

Question A. JSON. Qu'est-ce que le format json ?

Question B. JSONP. Qu'est-ce qu'on appelle jsonp et quel problème cela résout ?

Question C. Riotjs. En quelques phrases, décrire ce qu'est Riotjs (ou Reactjs dont Riotjs reprend le principe) 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

2.2 Scala

Question D. Stratégie appel. En Scala, quelles sont les différentes stratégies d'évaluation (ou stratégies d'appel) directement disponibles lorsqu'on écrit un programme ? En particulier, quelle syntaxe emploie quelle stratégie lorsqu'on déclare une variable ou lorsqu'on déclare une méthode ou une fonction ?

Question E. Stream. Expliquer la différence entre un stream et une liste.

Question F. Monade. Qu'est-ce qu'une monade ? Donner un exemple.

2.3 Akka

Question G. Acteur. Expliquer ce qu'est un acteur. Quand est-ce qu'il exécute du code, et quel type d'action peut effectuer un acteur ?

Question H. Shared mutable states. Beaucoup de problèmes en programmation parallèle viennent des états modifiables partagés entre threads (*shared mutable states*). La programmation fonctionnelle pure règle ce problème en ne manipulant que des données immuables (*immutable*). Quelle(s) solution(s) apportent les systèmes d'acteurs ?

3 Programmation

3.1 Scala

Question I. 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))
```

3.2 Akka

Question J. An actor enters into a bar. Voici le code d'un acteur. Expliquer ce qu'il fait, dire quelles sont les autres types d'acteurs avec lesquels il est censé communiquer et quels sont les types de messages employés. Décrire les messages avec des *case class* comme ceci : `case class TypeDeMessage(texte: String, x: Double)`.

```
class Server extends Actor {
  var customer: Option[ActorRef] = None
  var customerBeverage = ""
  lazy val barman = context.actorOf(Props[Barman])

  def slang(beverage: String): String = {
      "bro! %s subito" format beverage
  }

  def receive = {
    case Order(beverage) => {
      customer = Some(sender)
      customerBeverage = beverage
      val barmanBeverage = slang(beverage)
      barman ! Order(barmanBeverage)
    }
    case Conso(barmanBeverage, amount) => customer map
      { _ forward Delivery(
        customerBeverage,
        "Voici votre commande, ça fait %1.2f" format amount,
        amount)
      }
      customer = None
  }
}
```