Langages et environnements évolués : Scala

Environnement

Installer sbt en suivant les instructions du site scala-sbt.org.

Utilisation d'Ammonite

Le script d'installation fourni par l'auteur d'Ammonite fait l'installation sous la forme d'un fichier amm contenant essentiellement du bytecode dans `/usr/local/bin/`. Vous préférerez sans doute faire l'installation localement dans le répertoire `bin` de votre *home directory* `~/` (qui devrait être dans votre `PATH`).

```
mkdir -p ~/bin
echo '#!/usr/bin/env sh' > ~/bin/amm
curl -L https://github.com/lihaoyi/Ammonite/releases/download/1.4.2/2.12-1.4.2 \
>> ~/bin/amm
chmod u+x ~/bin/amm
```

Assurez-vous d'avoir votre `~/bin` dans votre `PATH`, vous pourrez alors lancer ammonite en tapant le nom de la commande `amm`. Autrement vous pourrez toujours utiliser `~/bin/amm`.

Création de projets avec IntelliJ IDEA

Il vous faudra installer le Plugins Scala dans IntelliJ IDEA (Préférences → Plugins → "scala", installer, relancer l'IDE).

Pour lancer un nouveau projet, choisir Scala → sbt. Vous ajouterez ensuite vos fichiers sources `*.scala`

Création de projets sbt avec un terminal et un éditeur de texte

pour créer le projet `monprojet` on crée un répertoire monprojet `mkdir monprojet && cd $_`. Dans le répertoire on crée la hiérarchie de répertoires et le build.sbt :

```
mkdir -p src/{main,test}/{java,resources,scala}
mkdir lib project target

# Create an initial build.sbt file
cat << EOF >> build.sbt
name := "HelloScala"

version := "0.1"

scalaVersion := "2.12.7"
```

Exercices REPL (amm, ou scala ou autre)

Composition

Programmer la composition de deux fonctions à un argument, de types quelconques.

Utilisation de structures de données et de for

J'ai 4 cales d'épaisseurs différentes exprimées en millimètres (valeur entière). Quand je les associe deux par deux je peux faire toutes les épaisseurs parmi 6, 8, 10, 12, 14, 16 millimètres. Quelles sont les épaisseurs de mes 4 cales ?

Sucre syntaxique

Expliquer et supprimer le sucre syntaxique (réecrire avec map, flatMap, filter) :

```
for {
  x <- List(1, 2, 3, 4)
  if x > 1
  y <- x to 5
  if x + y < 7
} yield (x, y)
```

Implicites

Expliquer les différents résultats de l'évaluation de `f("Bonjour")`.

```
def f(x: String)(implicit y: List[String]) {y.foreach(e => println(s"$x $e"))}
f("Bonjour")
implicit val lespersonnespresentes = List("Alix", "Alma", "Aloïse")
f("Bonjour")
```

```
implicit val lespersonnesdanslasalle = List("Sakari", "Sandeep", "Sascha")
f("Bonjour")
```

Classes, traits, types

Voici un code Scala "réel" qui permet de définir un objet Csvdata permettant de faciliter la lecture et l'écriture de fichiers CSV. Ce code utilise différentes fonctionnalités de Scala. Pouvez-vous reconnaître du typage structurel ? Des arguments optionnels avec valeur par défaut ? de l'héritage multiple ? Où ?

```
object Control {
  def using[A <: { def close(): Unit }, B](resource: A)(f: A => B): B =
    try {
      f(resource)
    } finally {
      resource.close()
    }
}

trait Csvreader {
  import scala.io.Source
  import java.io._

  def readFromFile(filename: String, separator: Char  = ';'):
  Option[Vector[Vector[String]]] = {
    try {
      Control.using(Source.fromFile(filename, "utf-8")) {
        source => Some(source.getLines.map(_.split(separator).toVector).toVector)
      }
    } catch {
      case e: FileNotFoundException => println("Couldn't find that file."); None
      case e: IOException => println("Got an IOException!"); None
      case e: Exception => print("Got an unlisted exception"); None
    }
  }
}

trait Csvwriter {
  import java.io._

  def writeInFile(data: Vector[Vector[String]],
                  filename: String,
```

```
                separator: String  = ";") {
    val bw = new BufferedWriter(new FileWriter(filename))
    data.foreach(v => bw.write(v.mkString(separator)))
    bw.close
  }
}

object Csvdata extends Csvwriter with Csvreader
```

—

— utilisez Csvdata pour écrire un fichier csv contenant deux lignes, trois colonnes et des valeurs quelconques.

— utilisez Csvdata pour relire votre csv

— écrire une fonction qui effectue la jointure naturelle de deux fichiers csv (on supposera que la clé apparaît dans la première colonne.

Exercices sbt

Premiers pas avec sbt

créer un projet `mon-premier-sbt`, ajouter un ficher source `src/main/scala/main.scala` contenant :

```
package fr.mipn.hello

object Main extends App {
  val nom = "MIPN"
  println(s"Bonjour $nom !")
}
```

— lancer sbt en mode interactif (commande `sbt`, ou utilisez sbt shell dans IntelliJ, en bas à gauche).

— attendre, compiler `compile`, (corriger et recompiler), exécuter `run`.

— vous pouvez tester en mode non interactif `sbt run` ou `sbt mon-premier-sbt/run`.

Effectuer des tests avec utest

Nous allons utiliser μtest pour écrire nos tests (les alternatives comme scalatest ou spec2 sont un peu plus complexes).

— ajouter les lignes suivantes à build.sbt :

```
libraryDependencies += "com.lihaoyi" %% "utest" % "0.6.5" % "test"

testFrameworks += new TestFramework("utest.runner.Framework")
```

— si sbt est resté ouvert en mode interactif, rechargez le projet dans sbt avec la commande `reload`.

— créez un fichier `exemple-utest.scala` contenant :

```
package fr.mipn.hello

import utest._

object HelloTests extends TestSuite{
  val tests = Tests{
    'test1 - {
      throw new Exception("test1")
    }
    'test2 - {
      1
    }
  }
}
```

— lancez la commande test dans sbt

— définir une fonction quelconque dans l'objet Main et écrivez des tests qui vérifient son comportement sur plusieurs arguments typiques à la place des tests de HelloTests. Exemple :

```
"f(1) égale 2" - {
  assert(Main.f(1) == 2)
}
```

Jointure

Créez un programme qui effectue la jointure naturelle de deux fichiers csv, tout d'abord en supposant que la clé (unique) est dans la première colonne, puis en considérant que la première ligne de chaque fichier contient l'intitulé de chaque colonne, effectuez la jointure sur les colonnes communes.

Scalajs

Cloner le projet https://github.com/pierreboudes/exemple-scalajs.

— Compiler sous sbt avec fastOptJS et ouvrez le index-dev.html dans un navigateur.

— Que fait le bouton "Rien" ?

— Ajouter un bouton.

Compléments

Scalajs

— Comment ajouter une fonction javascript et l'appeler dans le code Scala ?

Calculer des temps de réponse avec curl

Ce message sur Stackoverflow vous donne une méthode pour évaluer le temps mis par une requête http(s) avec curl. Vérifiez les métriques vues en cours sur quelques serveurs perdu.com, mipn.fr, uq.edu.au (faites plusieurs fois vos requêtes ou soustrayez le temps de résolution du nom). Utiliser ammonite ou un autre REPL Scala pour calculer ce que cela représenterait en jours, mois, années si on appliquait un facteur 10^9 (une nano-seconde égale une seconde).

```
cat <<EOF >> curl-format.txt
    time_namelookup:  %{time_namelookup}\n
       time_connect:  %{time_connect}\n
    time_appconnect:  %{time_appconnect}\n
   time_pretransfer:  %{time_pretransfer}\n
      time_redirect:  %{time_redirect}\n
 time_starttransfer:  %{time_starttransfer}\n
                    ----------\n
         time_total:  %{time_total}\n
EOF

curl -w "@curl-format.txt" -o /dev/null -s "http://wordpress.com/"
```

Réponse (janvier 2018)

— CPU ordinateur : 2-18

— CPU serveur : 2-60 par processeur

— iPhone : 2-6

— carte graphique : 100-3584

— Les meilleurs super-calculateurs : 10 000 000