Scala

Labyrinthe

Vérifiez les conventions à suivre pour l'écriture du code (cf Moodle).

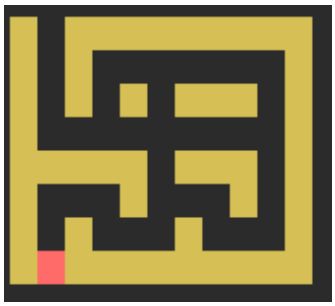
1. **Vous allez réaliser ce TD en Scala d'une manière purement fonctionnelle.**

Le TD peut tenir en un seul fichier : vous pouvez définir plusieurs classes et objets dans un même fichier scala. Faites un jar lançable par scala –jar (ne mettez pas les librairies scala dans un jar Java) / ou un jar pour Java pour les programmes Java.

Vous indiquerez EXPLICITEMENT les types de retours des méthodes et des fonctions dans les déclarations systématiquement.

En vous inspirant des solutions fonctionnelles des TD sur le sudoku et les anagrammes, vous allez réaliser un programme qui recherche la sortie de labyrinthes.

Par exemple, pour le labyrinthe suivant :

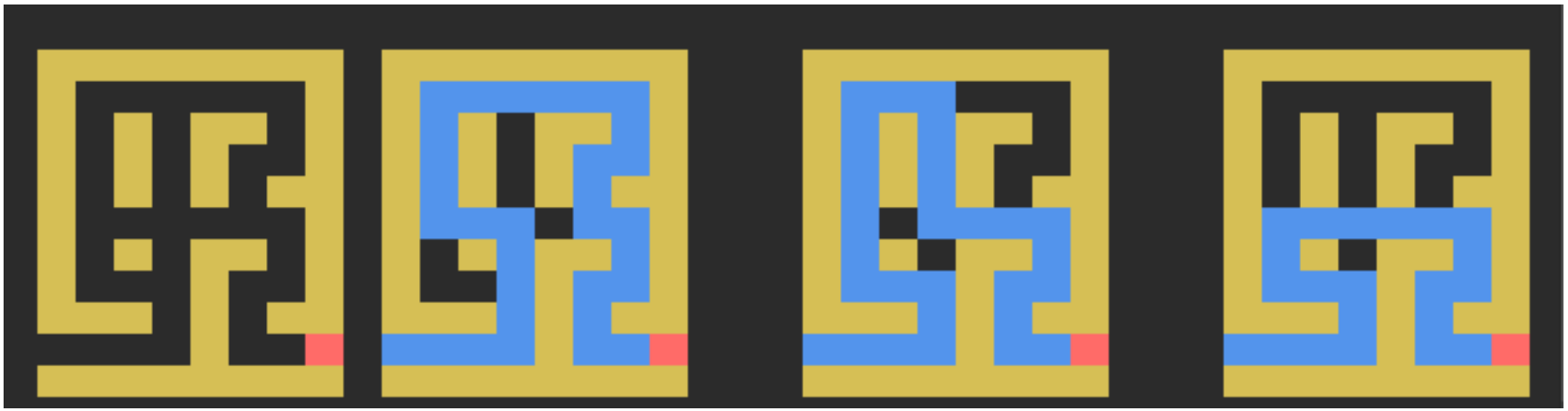


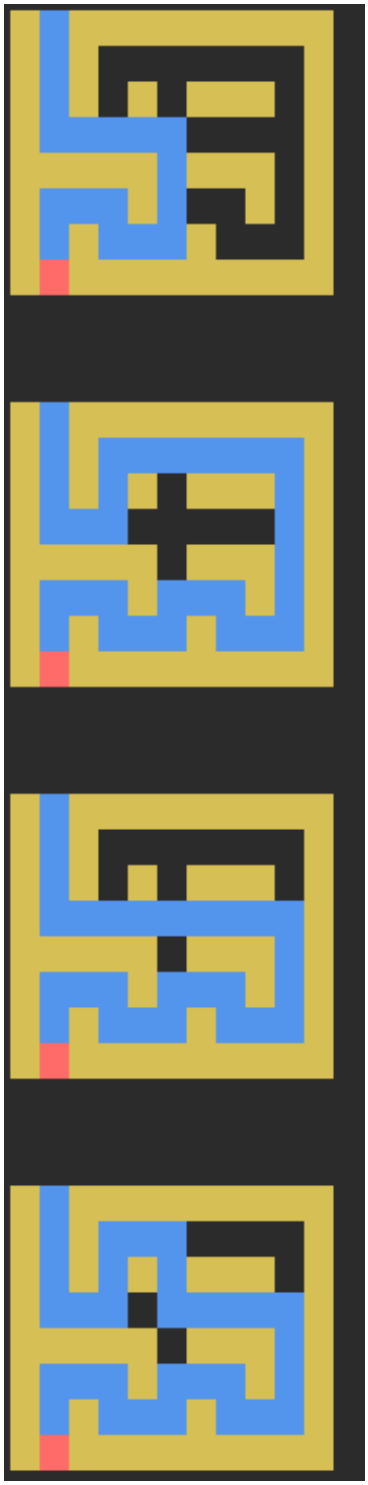
Le départ se trouve toujours à la première ligne, seconde colonne (0,1). La sortie est représentée par le carré rouge en bas (qui peut être n'importe où).

Le programme recherche et affiche les solutions trouvées de la manière suivante :

```
val laby = new Labyrinthe($( // 1 = mur, 0 = vide et 9 = sortie
  $(1, 0, 1, 1, 1, 1, 1, 1, 1, 1),
  $(1, 0, 1, 0, 0, 0, 0, 0, 0, 1),
  $(1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1),
  $(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1),
  $(1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1),
  $(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1),
  $(1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1),
  $(1, 9, 1, 1, 1, 1, 1, 1, 1, 1, 1)
) )

print(laby)
laby.cheminSortieAPartirDe((0,1)).foreach(  s => print((new Labyrinthe(s))+"\n\n") )
```





Le programme pourra gérer des labyrinthes de n'importe quelle taille (pas forcément carrés)

Vous allez fabriquer le code suivant (utilisez les codes ANSI d'un TD précédent) :

```
class Labyrinthe(init_ : Array[Array[Int]]) {
// notez qu'il n'y a pas de variables d'instances supplémentaires : seulement la constante init_
  override def toString = {

      ??? // utilisez seulement les opérateurs map, match et mkString (plusieurs fois si nécessaire)
  } // toString

  def cheminSortieAPartirDe(pos_ : Tuple2[Int, Int], lab_ : Array[Array[Int]] = init_) :
Set[Array[Array[Int]]]= {
    pos_ match {

        ?????
    }
  } // cheminSortieAPartirDe

} // class Labyrinthe
```

N'utilisez QUE des collections immuables (ou de manière immuable, c'est-à-dire, ne modifiez aucune collection directement).

Indices (regardez bien la solution du Sudoku) :

Si la case à partie de laquelle vous commencez votre chemin est un 9 : c'est terminé, vous avez trouvé. Vous pouvez donc rendre la grille courante.

Si c'est un espace vide (0), il faut envisager les recherches à partir des voisins dans la grille. Chaque chemin qui part d'un voisin va donner son ensemble de solutions : le résultat sera donc la fusion de ces ensembles.

Si c'est un mur (1) ou un endroit déjà visité (2), il n'y a rien par là, le résultat est un ensemble vide de solutions.

Vous trouverez sur l'espace Moodle, plusieurs labyrinthes de tailles différentes pour tester votre code (ne réalisez pas de lecture par programme de ces labyrinthes, copiez/collez directement le texte du tableau du labyrinthe).

En théorie, avec ces indications, vous devriez réussir à proposer une solution très rapidement (et très courte).

2. **Maintenant, réalisez ce programme en Java en *fonctionnel* ou *pas* : carte blanche – mais faites VOTRE solution à vous.**

En plus des consignes du Moodle, rendez une archive avec les deux sources : Java et Scala + des copies d'écran .jpg des résultats pour les labyrinthes (mettez le nom du labyrinthe à coté)