INFOM451 Conception d'applications mobiles

Projet « Gastro »
Partie 1 : compositeur de menus simplistes

Gonzague Yernaux - gonzague.yernaux@unamur.be - Bureau 224

Année 2020-2021

1 Introduction

À l'ère où chaque aliment ingurgité doit être remis en question et, surtout, en rayon si sa boîte manque de labels de qualité ; où la conscience de la planète et de la santé s'accroît proportionnellement au nombre de produits chimiques, nocifs et toxiques qui atterrissent dans les supermarchés ; et à l'ère où même les fruits ne sont plus sûrs à cause des pesticides [1], les humains du quotidien ne savent même plus comment faire pour composer un menu de qualité, démunis qu'ils se sentent devant la gamme de produits proposés. Bref, il leur manque un bon outil pour pouvoir trancher [2].

Vous êtes en charge du projet « Gastro ». Le but du projet est de développer une application Scala capable de générer des menus composés de produits qui, dans l'ensemble, s'ils sont consommés en un repas, ne seront « pas trop mauvais »pour la santé. Mais attention, les potentiels clients de l'application ne veulent pas manger du riz blanc à chaque repas. Si l'application sélectionne un peu de gras et de sucre *dans les limites du raisonnable*, ce sera certainement mieux apprécié de vos clients !

2 Objectifs d'apprentissage

Ce premier mini-projet a pour objectif de vous aider à prendre en main les bases à la fois fonctionnelles et orientées-objets de Scala :

- définir des classes
- utiliser l'héritage
- user et abuser des *for comprehension*
- définir des *case classes*
- utiliser le pattern matching sur des *case classes*
- utiliser des fonctions anonymes ou de l'application partielle
- utiliser des méthodes de haut niveau sur les listes que vous connaissez bien [3] (map, filter, foldLeft)
- découvrir quelques structures de données non-mutables en scala (List, Set, Map)

Pour chacun de ces objectifs, vous indiquerez un commentaire à **un** endroit de votre code où vous l'utilisez. Par exemple :

1. Quoique, certains fruits resteront toujours sûrs.
2. Il faut ici comprendre « trancher »au sens figuré.
3. Merci PFL !

```
/* Fonctions de haut niveau (map, foldLeft)
   Fonction anonyme (_ + _) */
(nutrients.map(getNutrientQuality)).foldLeft(0.0)(_ + _)/nutrients.length
```

(Ne faites pas ça sur les parties de code déjà fournies évidemment !)

3 Consignes

Pour cette première partie du projet, vous recevez un code Scala simplet, à partir duquel vous avez deux missions. La première est d'interpréter[4] à votre façon certains algorithmes du programme et de **les rendre plus intéressants**. La seconde est de vous assurer que le programme soit **fonctionnel**[5]. Le tout fera l'objet d'un court rapport (max. 3 pages A4) à remettre à l'assistant. Voyons tout de suite plus en détails chacune de vos missions.

3.1 Description du programme fourni

`gastro.scala` est un programme qui effectue les actions suivantes :

1. Dans une petite base de données nommée `products.csv` (format CSV[6]) se trouve un listing de produits vendus en supermarchés (américains). Le programme va ouvrir ce fichier et lire chaque ligne, dont les informations intéressantes serviront à la création d'un nouvel objet de la classe `Product`. Tous les produits sont stockés dans une `List[Product]` qui sera utilisée par le programme par la suite. Ces actions sont effectuées par l'objet singleton `GastroExtractor`.
2. En l'occurrence, dans le programme fourni un produit a pour attributs sont identifiant, son nom, sa teneur de énergie (kcal) et sa teneur en protéines.
3. Il ne reste plus qu'à demander à un objet de la classe `MenuComposer` de faire son boulot, c'est-à-dire renvoyer (sous forme de `String`) une composition de menu faite à partir des produits qu'il a en sa possession. Pour ce faire, trois produits aléatoires sont tirés au sort parmi la liste de tous les produits ; puis l'appel à une méthode auxiliaire `quality_indicator` de la classe `Product` permet d'avoir une indication de la qualité de chaque produit : plus la valeur est basse, meilleur est le produit. Dans le programme simplet fourni, on considère qu'un bon menu est composé de 3 produits dont la « qualité »totale est inférieure à 1. Le programme affiche donc un message, soit positif (le menu composé aléatoirement est délicieux et sain) soit négatif (aïe, il y a trop de gras/de sucré/de salé...).

3.2 Votre mission : revisiter l'algorithme de composition de menus

Pour l'heure, le programme sélectionne au hasard trois produits et tente de les assembler, sur base d'une mesure de la qualité des produits. Et si on pimentait[7] tout cela ? Vous êtes invités à proposer un algorithme différent de composition de menu et, si applicable, de calcul de la qualité d'un produit. Par exemple, plutôt que de choisir trois produits aléatoirement, vous pourriez en sélectionner un, puis chercher, sur base des nutriments dont il manque, deux autres produits qui le complèteraient au mieux. Cette mission est très libre : vous voyez le nombre de colonnes dans `products.csv` ? il y a au moins autant de façons d'imaginer une composition de menus sains !

Soyez imaginatifs et créatifs[8]. Vous êtes libres de jouer à votre guise avec les données qui vous sont fournies. La seule consigne à respecter crucialement ici est de ne pas perdre de vue la *mission transversale* lorsque vous codez.

4. Dans le sens humain, non-machine du terme.
5. Dans tous les sens du terme.
6. Format simple à manipuler : les données sont structurées, chaque colonne est séparée des autres par un point-virgule
7. Au sens figuré... ?
8. Ce serait mieux. Mais à défaut, implémentez simplement un algorithme différent qui fera usage des objectifs d'apprentissage.

3.3 Mission transversale : les concepts Scala

Il vous est demandé d'écrire **dans un style fonctionnel** propre au Scala tel que vous l'avez vu au cours. Evitez au maximum les boucles `while`, les variables mutables (`var`), et tout le reste qui ne fait pas partie du paradigme véhiculé par Scala. **L'évaluation sera en majeure partie centrée sur cet aspect** : il vaut largement mieux que votre algorithme soit peu original ou simpliste, mais que tout le reste soit codé dans un style fonctionnel propre au Scala, plutôt que de réaliser une implémentation énorme, mais en empruntant un style impératif. L'objectif est que vous appreniez un paradigme nouveau (mélange des paradigmes fonctionnel et orienté objet) et l'addition sera salée s'il s'avère que vous utilisez des paradigmes autres (pur orienté objet, pur fonctionnel, impératif). Comme indiqué à la section 2 sur les objectifs d'apprentissage, il vous est en particulier demandé d'indiquer en commentaire chaque objectif d'apprentissage qui apparait dans votre code.

3.4 Rapport

Un rapport de trois pages maximum vous est demandé à l'issue de ce projet. Vous pouvez y indiquer les choix que vous avez faits pour chacune des missions en les justifiant. En pratique, le rapport vous permettra principalement d'expliquer à votre aise les algorithmes que vous avez implémentés ou adaptés, ainsi que d'autres informations sur lesquelles vous voudriez attirer l'attention de l'assistant (par exemple : « Pour remplir mes deux missions, j'ai drastiquement modifié le `GastroExtractor` pour l'adapter à ma vision des choses car... »). Dans le rapport, vous pouvez mentionner les difficultés rencontrées. Le but du rapport est de mettre de la lumière sur des choix qui ne seraient pas compris limpidement au premier coup d'oeil sur votre code. Mais pour les aspects purement techniques, privilégiez les commentaires dans le code (exemple : pour expliquer ce que fait une ligne de code ou une méthode). Prêtez une attention particulière à la forme du rapport, qui est également cotée.

3.5 Délivrable attendu

Le délivrable attendu est un .zip à soumettre sur WebCampus comprenant le code (votre version de `gastro.scala`), le rapport au format PDF et, si vous y avez apporté des modifications, votre version de `products.csv` (ou tout autre fichier, .csv ou autre, dont votre code fait usage).

N'oubliez pas, voici une *checklist* des choses essentielles que doit vérifier votre délivrable.

- Le code compile, sans *warning* et s'exécute.
- Les consignes de la mission de la section 3.2 sont respectées. L'algorithme que vous avez implémenté est personnel et permet d'utiliser de nombreux concepts propres à Scala.
- Le code fait ce qui est spécifié dans sa documentation. Si un changement de fonctionnement devait survenir par rapport à la description de la section 3.1 (par suite d'une amélioration de l'algorithme de génération de menus par exemple), ce changement est documenté et justifié.
- Le code respecte le paradigme véhiculé Scala, comme expliqué dans la section 3.3.
- Le rapport est correctement écrit (soit en français, soit en anglais) et mis en forme ; il remplit le rôle décrit dans la section 3.4.
- Le code est clair, lisible, optimisé. Il s'exécute dans un temps raisonnable.

4 En pratique

4.1 Scala

Il vous faut bien sûr installer et utiliser Scala. La meilleure référence est bien entendu le site officiel : https ://www.scala-lang.org/

Pour exécuter votre code :

```
scala gastro.scala
```

4.2 Ressources

Les fichiers `gastro.scala` et `products.csv` en leur version initiale sont téléchargeables sur WebCampus. Notez bien qu'il y a de nombreuses lignes dans le fichier de produits ; autrement dit, le fichier est très volumineux et son temps de parcours peut être important. Vous ne serez pas pénalisés à cause de ce temps de parcours, mais pouvez supprimer une partie des lignes des fichiers si vous estimez que c'est cohérent, en justifiant votre démarche dans le rapport.

Les fichiers fournis sont des échantillons des jeux de données disponibles à l'adresse

https ://www.ars.usda.gov/northeast-area/beltsville-md-bhnrc/beltsville-human-nutrition-research-center/food-surveys-research-group/docs/fndds-download-databases/

Si vous le souhaitez, vous pouvez y télécharger les jeux de données complets et jouer avec, ou encore sélectionner un autre échantillon que celui choisi par l'assistant. Bien sûr, dans ce cas, documentez vos choix dans le rapport.

4.3 Dernières consignes importantes

— Envoyez vos fichiers en UTF8 sans BOM

— Mettez des chemins relatifs pour les fichiers, et considérez que les jeux de données sont dans le même dossier que le fichier de code. Ça facilitera la correction.

— Essayez de nettoyer votre code pour ne pas avoir de warning (évitez les « val »dans les for comprehension, les opératuers postfixes (`a toList -> a.toList`)). Cela dit, si vous implémentez `filter` et non `filterWith`, vous aurez un warning. Celui-là n'est pas grave.

— N'indiquez pas de package. Ça facilitera la correction.