INFO0054
2023-2024

Feedback Interrogation 2

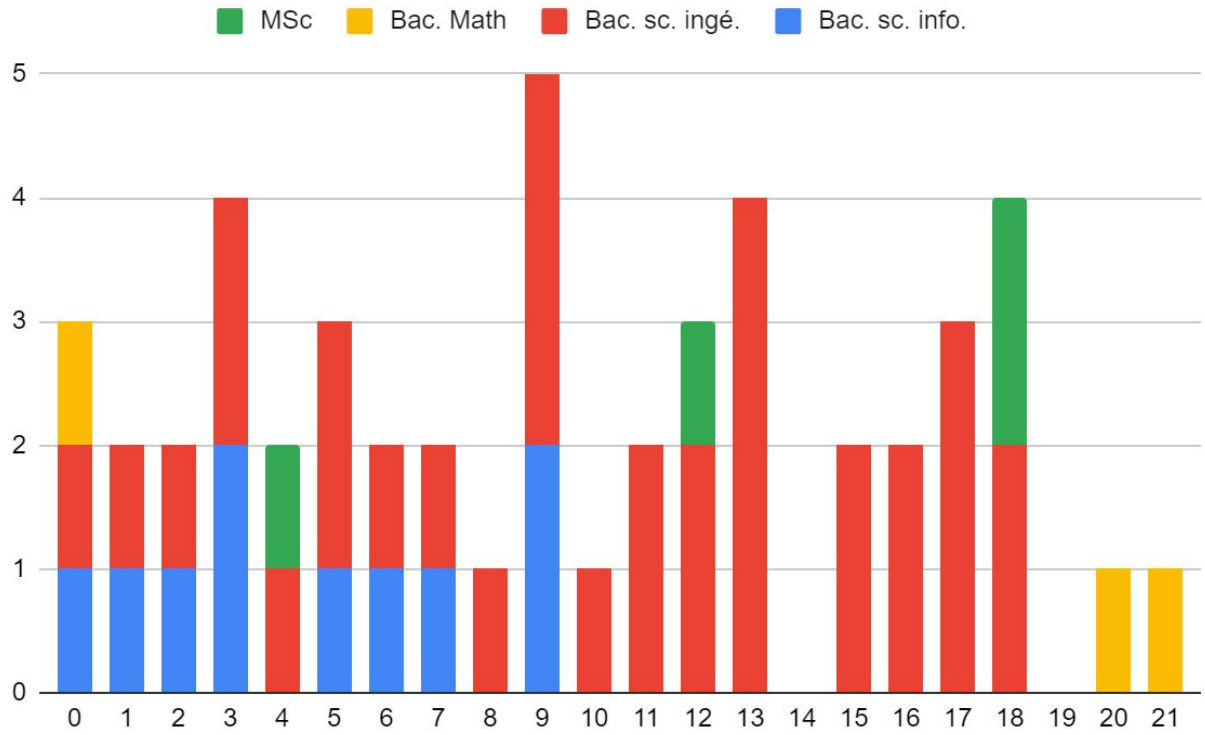
Quelques chiffres

- 55 étudiants inscrits et 49 étudiants ont participé

	Bac. sc. info.	Bac. sc. ingé.	Bac. Math	MSc	All
Nombre	10	32	3	4	49
Échec	80.00%	31.25%	33.33%	25.00%	40.82%
Échec léger	20.00%	12.50%	0.00%	0.00%	12.24%
Réussite	0.00%	56.25%	66.67%	75.00%	46.94%

→ 57% des étudiants n'ont pas réussi. Différences entre options…

Statistiques I



Statistiques II

NAME	Q1 4+2	Q2 8	Q3 2	Q4 4	Q5 2	TOTAL 20
average	2.8	3.1	1.0	1.9	0.8	9.5
st.dev.	1.9	2.0	0.9	1.5	0.9	6.1
max	6.0	7.0	2.0	4.0	2.0	21.0
mode	4.0	2.0	0.0	1.0	0.0	9.0

Statistiques III (ceux qui ont essayé: >= 4/20)

	Bac. sc. info.	Bac. sc. ingé.	Bac. Math	MSc	All
Nombre	10	32	3	4	49
Échec	80.00%	31.25%	33.33%	25.00%	40.82%
Échec léger	20.00%	12.50%	0.00%	0.00%	12.24%
Réussite	0.00%	56.25%	66.67%	75.00%	46.94%

	Bac. sc. info.	Bac. sc. ingé.	Bac. Math	MSc	All
Nombre	5	27	2	4	38
Échec	60.00%	18.52%	0.00%	25.00%	23.68%
Échec léger	40.00%	14.81%	0.00%	0.00%	15.79%
Réussite	0.00%	66.67%	100.00%	75.00%	60.53%

Statistiques III (ceux qui ont essayé: >= 4/20)

NAME	Q1 4+2	Q2 8	Q3 2	Q4 4	Q5 2	TOTAL 20
average	3.4	3.8	1.2	2.4	1.0	11.8
st.dev.	1.7	1.6	0.8	1.3	0.9	4.9
max	6.0	7.0	2.0	4.0	2.0	21.0
mode	4.0	4.0	2.0	4.0	0.0	9.0

Question 1

L'union de Set[Int] et le max entre deux entiers peuvent faire partie d'un monoïde.

- Certains étudiants ont oublié de fournir l'implémentation dans Scala. Certains ont oublié l'identifiant lors de l'implémentation.
- Certains ont oublié de mentionner l'élément neutre des opérations binaires.
- **Associatif != commutatif** ☹️

Les fonctions foo et bar étaient bien associatives, mais n'avais pas un élément neutre / d'identité.

- Beaucoup ont décrit foo et bar comme n'étant pas associatifs en démontrant qu'ils n'étaient pas commutatifs.
  - Quid?! 😭

Question 1 - Bonus

J'apprécie que beaucoup essaient de fournir une fonction.

- 1 point pour la spécification et 1 point pour l'implémentation (si la première était correcte).
- Beaucoup ont oublié que foldMap prend en entrée une fonction **f : A=>B** ! Certains ont implémenté une sorte de **combine** étrange (sourvent avec des listes imbriquées).

Question 2

Demandé :

```
scala> compute(List(1,2,3,4), _ * _)
val res0: List[Int] = List(2, 6, 12)

scala> compute(List(1,2,3,4), _ + "|" + _)
val res1: List[String] = List(1|2, 2|3, 3|4)
```

Mais certains d'entre vous ont interprété :

```
scala> compute(List(1,2,3,4), _ * _)
val res0: List[Int] = List(2, 12)

scala> compute(List(1,2,3,4), _ + "|" + _)
val res1: List[String] = List(1|2, 3|4)
```

J'ai considéré les deux comme corrects à condition que les deux versions fassent la même chose.

Un nombre surprenant (et suspect ?) d'étudiants ont implémenté deux versions différentes. J'ai retiré un point dans ce cas car les comportements des deux fonctions n'étaient pas le même.

Question 2

Plus gros problème : la définition de la récursion structurelle (complète). L'approche la plus évidente reposait sur une récursion structurelle complète.

Beaucoup ont mentionné la récursion structurelle. L'explication ne reposait pas sur cette définition, ou était absurde. "C'est un exemple de récursion structurelle complète car nous parcourons toute la liste." → 😐😵

Question 2

Près de dix étudiants ont fourni une itération (sur des listes) en tant que récursion et une itération sur des index (de liste) comme itération. Avez-vous étudié le cours ?

Quelques étudiants se sont appuyés sur des boucles **for** et **while** avec des effets secondaires. 😡

Question 2

- Mélanger le syntaxe de **match case** et d'instructions **if**
- Les résultats de **f** étant renvoyés à la place des listes
- La fonction **f** étant passée dans la fonction auxiliaire
- Oublier d'inverser l'accumulateur
- Oublier d'appeler la fonction auxiliaire
- **acc :: f(h1,h2)** -> pas une liste valide
- ...

Bref, tout le monde a commis au moins une (bête) erreur.

Question 3

- 1 point par fonction.
- 1/2 si les spécifications étaient correctes

- La fonction **tail** a renvoyé des objets de type **MyList[A]** (et non **A**)
- **List(1).tail() → Some(List())** (et non **None**)
  - Même chose pour Either
- Quelques étudiants ont lancé des exceptions… 😟

Question 4

- 1 point pour chaque spécification
- 1 point pour chaque implémentation

- J'étais content de voir que beaucoup avaient les bonnes spécifications
  - Quelques fautes connes, quand même, e.g., f: List[A] => List[B], f: A => List[A], …
- L'utilisation de foldRight, en revanche
  - Utilisation foldLeft dans le foldRight; (acc, a) au lieu de (a, acc)
  - Usage de cons au lieu de append
  - …
- L'implémentation de map était aussi simple que flatMap((a) => List(f(a))
  - J'en ai parlé en classe !

Question 5

Solution ? Simplement transformer l'argument acc en argument *call-by-name* => **acc**

- Certains d'entre vous mélangent les concepts *call by name* et *call by need*. Ça fait mal.
- Je demandais également le nom de la technique que vous avez utilisée. Une "évaluation non stricte" ne suffisait pas.
- Certains d'entre vous ont utilisé des **lazy vals** où ils n'avaient absolument aucune valeur ajoutée.
- Quelques étudiants ont mentionné LazyList ou Flux, mais aucune opération n'a été enchaînée. Cela n'a de sens que si vous enchaînez des opérations ou si les expressions passées doivent être évaluées de manière paresseuse. **Les arguments avancés n'étaient pas convaincants**.
- Même si beaucoup d'entre vous avaient => acc correct, un nombre important d'étudiants ont donné des explications absurdes (des **println** dans la liste passés en argument, par exemple).

Conclusion

Je suis honnêtement déçu.

- Q2 est une question "classique" et vous devriez être capable de la répondre correctement. Soyez certain qu'une telle question apparaîtra à l'examen.
- La question sur les monoïdes n'aurait pas dû être une surprise.
- Assurez-vous de connaître les fonctions importantes (fold,foldRight,foldMap, flatMap, map, ...).

INFO0054 est un cours sur la programmation fonctionnelle utilisant Scala, ce n'est pas un cours sur Scala.