Premiers pas Scala

On part de l'hypothèse que vous utilisez Scala 2. Pas de récursivité, pas d'itérateurs (for, while etc) dans ce TD. Rendez un document texte avec les réponses aux questions (voir les numéros). Utilisez Scaladoc https://www.scala-lang.org/api/2.13.12/

1. Réalisez un programme qui contient un main (tout simple). Vérifiez que vous arrivez à le compiler et à l'exécuter (faites le afficher qq chose).

2. On fabrique une liste de la manière suivante :
   val l = List[Int](1,2,3,4,5) // remarquez le [Int]
   ou de la manière suivante :
   val l = 1 ::2 ::3 ::4 ::5 ::Nil // ici, :: est une méthode et Nil est une liste vide
   Fabriquez une liste de chaines de caractères et affichez son contenu

3. On accède au ième élément d'une liste de la manière suivante : l(i)
   Attention, c'est comme la syntaxe d'une fonction.
   Affichez le 3 ième élément de la liste que vous avez créé. Notez que Scala considère les accès tableaux comme un appel de fonction – ce qui est assez logique en fait.

4. **map** permet de convertir un élément d'une collection. Une liste est une collection. La conversion d'un élément se fait en passant à **map** une fonction.
   Par exemple, essayez de **calculer** l.map((elt :String) => "("+elt+")") puis affichez l
   Est-ce que l est modifiée ? pourquoi ?
   Affichez maintenant l.map((elt :String) => "("+elt+")")
   Pourquoi est-ce différent ?

5. Lancez un navigateur avec scaladoc : https://www.scala-lang.org/api/2.13.12/
   Faites une recherche sur **mkString**

6. Utilisez **mkString** sur une liste pour faire une chaîne avec séparateur. Par exemple, on souhaite transformer une liste qui contient 1, 2, 5 en chaine : "maliste= 1 puis 2 puis 5 et voilà"
   Trouvez une solution en une ligne (sans itérateur et sans boucle, en utilisant **mkString**)

7. Soit une liste de nombres entiers, on souhaite calculer la liste qui ne contient que les nombres pair. Utilisez **filter** (regardez dans Scaladoc)

8. Le couple de valeurs en Scala est une instance de **Tuple2**, par exemple, on peut écrire
   val c : Tuple2[Int][Int] = Tuple2[Int, Int](4,8) // Tuple2, 2 car 2 valeurs, on peut faire Tuple6 par exemple
   println(c)
   println(c._1) // _1 est le premier élément du couple (à gauche)
   println(c._2) // _2 est le seconde élément (à droite)

9. On peut aussi laisser le compilateur travailler et écrire : **val c = (4,8)**. Essayez.

10. En utilisant la méthode de collection **partition** (recherchez dans scaladoc), écrivez une ligne qui sépare les nombres pairs des nombres impairs d'une liste de nombres entiers.

11. Utilisez **permutations** (voir scaladoc) pour sur une liste d'entier. Quel type est obtenu ?

12. Utilisez **mkString** sur le résultat pour en faire une chaîne et pour afficher le résultat sous la forme (pour la liste 1,2,3, 8, 9) :
    List(1, 2, 3, 8, 9)
    List(1, 2, 3, 9, 8)
    List(1, 2, 8, 3, 9)
    List(1, 2, 8, 9, 3)
    List(1, 2, 9, 3, 8)
    List(1, 2, 9, 8, 3)
    List(1, 3, 2, 8, 9)
    List(1, 3, 2, 9, 8)
    List(1, 3, 8, 2, 9)
    List(1, 3, 8, 9, 2)
    List(1, 3, 9, 2, 8)
    Etc.. (un résultat par ligne)

13. Toujours en une ligne, rajoutez ce qu'il est nécessaire de rajouter pour que le résultat soit, cette fois :
    1-2-3-8-9
    1-2-3-9-8
    1-2-8-3-9
    1-2-8-9-3
    1-2-9-3-8
    1-2-9-8-3
    1-3-2-8-9
    Etc..

14. Une chaîne de caractère est une forme de collection. Vérifiez que vous pouvez simplement, calculer les anagrammes d'une chaîne de caractères.

15. Faites en sorte d'afficher les anagrammes sans répétition. Le plus simple pour éliminer les répétitions est de convertir la collection des anagrammes en ensemble par la méthode **toSet**. Toujours en une ligne, faites une expression qui affiche les anagrammes sans répétition d'une chaîne de caractères.

16. Notez bien l'utilisation de **mkString** pour convertir une collection en chaîne de caractères. Vous n'avez pas besoin de **for** ou de **while** pour afficher le contenu d'une collection.

17. Utilisez **filter** pour sélectionner les nombres pairs d'une liste et **mkString** pour les afficher.

18. De manière générale, et par rapport à ce que vous avez déjà fait, notez que l'utilisation de ces mécanisme simplifie les algorithmes.

19. **If (condition) resultat1 else resultat2** permet de choisir une valeur parmi les deux résultats. En utilisant **map**, **mkString** et une conditionnelle (et en une ligne) et **contains** (voir scaladoc), filtrez une collection pour en extraire les 'gros mots'. Par exemple, si on suppose que les gros mots sont représentés par la liste ("merde","con","boudin"), alors le filtrage de "merde","avion","voiture","eau","con","train" donne "***","avion","voiture","eau","***","train"

20. Modifiez le code précédent (toujours en une ligne) pour ne pas conserver les "***"

21. Essayez **reverse** sur une string. Ca fait quoi ?

22. Essayez **reverse** sur une collection. Ca fait quoi ?

23. Vérifiez qu'une chaîne inversée ou pas donne les même anagrammes (toujours en une ligne)

24. On appelle ***palindrome***, un mot qui peut être lu dans les deux sens. Par exemple, **radar** est un palindrome car **radar** à l'envers donne **radar**. **Hannah**, **rotor**, **kayak**, **ressasser**, **bob** sont des palindromes. Réalisez une expression qui permet, à partir d'une liste de mots, d'en extraire la liste des mots palindromes d'un côté, et de l'autre côté, la liste des mots non palindromes. En une ligne.

25. Est-ce que « Tu l'as trop écrasé, César, ce Port-Salut » (Victor Hugo) est un palindrome ?

26. On suppose que oui. Réalisez un test qui permet de vérifier que cette phrase est une phrase palindrome (le moins de lignes possibles)