Introduction à Scala

Concepts de la programmation avec Scala

2022-2023

Sommaire

1. Scala : C'est quoi– Pourquoi?
2. Règles de base
3. Les variables
4. Manipulation des String
5. Les Opérateurs
6. Les Collections
7. Les Structures de contrôle
8. Les Fonctions

Scala : C'est quoi– Pourquoi?

- Un langage de programmation
- Développé en 2001 par **Martin Odersky** à l'École Polytechnique Fédérale de Lausanne (EPFL) en suisse en 2001
  - Première version publique sortie fin 2003
- Scalabe (d'où son nom Scala)
- Tiré du Java avec une Syntaxe assez simple
- Scala combine
  - **la programmation fonctionnelle**

(basée sur des fonctions. Ex : val a = List (1,2).
En réalité List() est un fonction qui derrière, combine des Iterable, Seq…etc Mais l'utilisateur ne voit que la fonction simple)

  - **et la programmation Orienté Objet**
- Scala est Dynamiquement typé (Ex: val a = 2 , renvoie un Int)
- …etc.

Créer une liste avec Java :

```
1  List<String>list = new ArrayList<String>();
2  list.add("1");
3  list.add("2");
4  list.add("3");
```

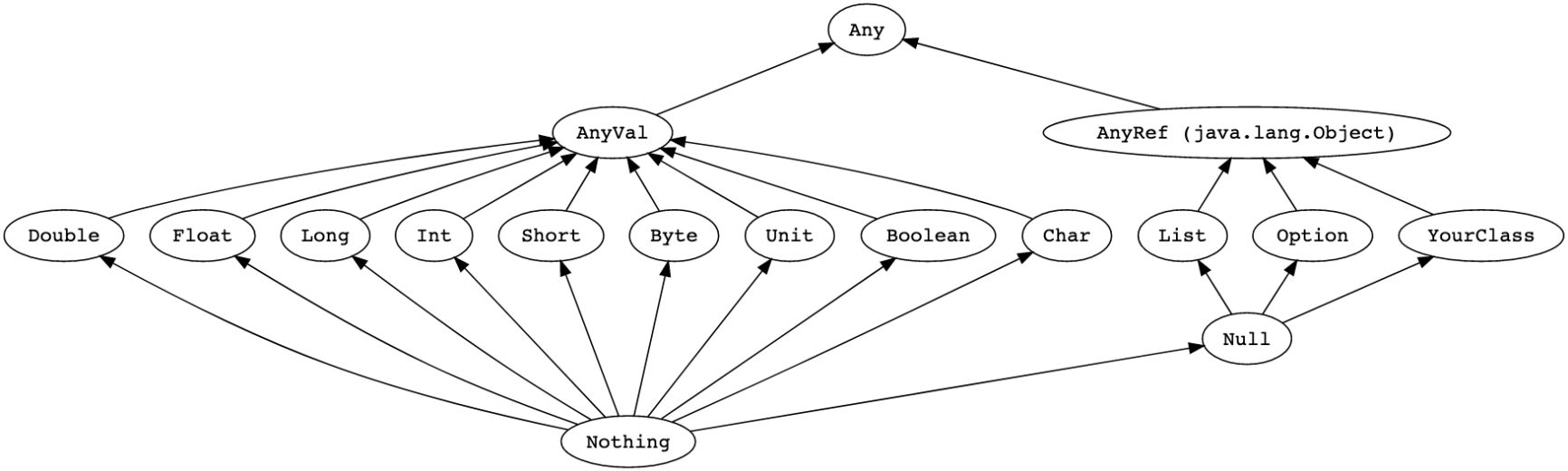
Créer une liste avec Scala :

```
1  val list = List("1", "2", "3")
```

Règles de base

- Le caractère ';' en fin d'instruction est optionnel sauf si plusieurs instructions sur une ligne
- Sensible à la casse
- Le nom des classes débute par une majuscule
- Les methodes commencent par un minuscule
- Le nom du fichier du programme doit correspondre au nom de l'objet en question
- La fonction def main(args: Array[String]) est obligatoire et représente le point d'entré du programme
- Les noms de variables, d'objets, de classes, de fonctions débutent par une lettre ou un underscore. Ex: nom, _nom, _1_nom …
- Commentaires:
  - `//` => une ligne
  - `/* … */` => multiple lignes
- Import de modules:
  - `import org.me._` => importe toutes les méthodes dans me
  - `import org.me.porterHabit` => n'importe que la méthode porterHabit
  - `import org.me.{seLaver, porterHabit}` => n'importe que les méthodes porterHabit et seLaver

Les DataTypes



Les variables

- Syntaxe Générale

Keyword nomVaribale [: DataType] = valeur

- Les valeurs de keywords peuvent être:
  - val : Pour designer une variable immuable (dont le contenu est non modifiable après assignation)
  - var : Pour désigner une variable mutable (dont le contenu est modifiable après assignation)
  - lazy val: Pour designer une variable qui n'est évaluée que lorsqu'elle est appelée

Exemple : var maProfession = ''data engineer''

- On peut déclarer plusieurs Variables en même temps:

Exemple 1 : val (a,b,c) = 1,2,3 **# a, b, c valent 1,2,3**

Exemple 2 : val a,b = 100 **# a et b valent 100**

Exercices : Les variables

- Créer une variable mutable nommée « profession » et renseigner la valeur : Data Engineer

- Vous avez été muté et vous êtes maintenant « scrum master ». Changer votre profession.

- Créer en une seule commande les variable nom, prenom et age. Pour des raisons de fraudes sur l'identité de la personnes, personne ne doit pouvoir modifier les valeurs de ces variables.

- Afficher les variables à l'aide de la fonction println(" ")

```
println(s" Mon age est $age")
println(" Et mon nom complet est %s  %s".format(nom, prenom))
```

Manipulation des String

val proverb: String = "apprendre à positiver ses emotions pour etre en harmonie avec soi-meme et avec les autres«

- proverb(0) ➔ L'élément à l'indice 0
- proverb.length ➔ renvoie la taille de la chaine
- proverb.capitalize ➔ renvoie la première lettre en Majuscule
- proverb. toUpperCase ➔ renvoie la chaine en Majuscule
- proverb.substring(0,9) ➔ renvoie les 9 premiers caractères
- proverb.contains('e') ➔ renvoie True si la chaine contient 'e'
- proverb.count(lettre => lettre =='e') ➔ renvoie le nombre d'occurrence de 'e'
- proverb.replace("e", "a") ➔ remplace tous les 'e' par 'a'
- proverb.concat("Une autre phrase") ➔ rajoute "Une autre phrase« à proverb
- proverb.filter(lettre => lettre !='e') ➔ Supprime tous les 'e' de proverbe

Les opérateurs

- Opérateurs arithmétiques
  - +, -, *, /, % correspondant respectivement à l'addition , la soustraction, la multiplication, la division et le modulo
- Opérateurs de comparaison
  - == ➔ égalité
  - != ➔ différent
  - >, >= ➔ strictement supérieur et supérieur ou égal
  - <, <= ➔ strictement inférieur et inférieur ou égal
- Opérateurs logiques
  - && ➔ ET logique
  - || ➔ OU logique
  - ! ➔ NOT
- Opérateurs d'affectation
  - =, +=, -=, *=, /=, %=

Exercice : Les opérateurs

1. Combien de temps (en Jours et en Heure) il faut à un marcheur pour parcourir une distance de 750km à une vitesse de 4.8km/h

   - **NB** : Vitesse = Distance/Temps et la fonction println() permet d'afficher un résultat

2. Un magicien dit que

   - Quand on choisit un nombre premier différent de 2 et 3
   - On l'èleve au carré
   - On lui ajoute 17
   - On divise par 12
   - Alors le reste de la division vaut 6

3. Le gouvernement a décidé d'offrir une prime de 300€ à certains fonctionnaires en fonction de leur salaire et de leur ancienneté. Comme toutes les autres mesures prises par le gouvernement, il est difficile de comprendre à qui cette mesure s'applique.

De ce que vous avez compris, une personne peut toucher à la prime si :

Critère 1 : Elle a **moins** de **5 ans** d'ancienneté **et** son salaire est **strictement inférieur** à **1500** euros.

Critère 2 : Elle a **entre 5 et 10 ans** d'ancienneté **et** son salaire est compris **entre 1500 et 2300** euros.

Critère 3 : Elle a **plus** de **10 ans** d'ancienneté **et** son salaire est strictement **inférieur** à **1500** euros **ou supérieur** à **2300** euros. C'est à dire qu'une personne ayant plus de 10 ans d'ancienneté et un salaire entre 1500 et 2300 euros ne peut pas toucher à cette prime.

Bernadette a **12** ans d'ancienneté et un salaire de **2400** euros.

Marc a **6** ans d'ancienneté et un salaire de **1490** euros.

Les Collections

- **Les Tableaux (ou Array)**
  - **Syntaxe :**

    - En initialisant la taille du tableau

    val tab = new Array[DataType](longueur)

    - En précisant les valeurs par défaut

    val tab2 [:Array[T]] = Array(value1, value2…., valueN)

Exemple :

```
val tab = new Array[Int](3)
val tab2 : Array[Int] = Array(2,2,3)
```

NB : Les array sont mutables

Les Collections

- **Les Tableaux (ou Array)**
  - **Ajout de valeurs (avant – après) :**

```
val v1 = Array(4,5,6)
```

- `val v2 = v1 :+ 7` // Array(4, 5, 6, 7)
- `val v3 = v2 ++ Array(8,9)` // Array(4, 5, 6, 7, 8, 9)
- `val v4 = 3 +: v3` // Array(3, 4, 5, 6, 7, 8, 9)
- `val v5 = Array(1,2) ++: v4` // Array(1, 2, 3, 4, 5, 6, 7, 8, 9)

Les Collections

- **Les Tableaux (ou Array)**
  - **Quelques fonctions natives :**

```
val nums = Array(1,2,3)  // oubien (1 to 3).toArray
```

- `nums.map(_ * 2)` // Array(2, 4, 6)
- `nums.filter(element => element < 3)` // Array(2)
- `nums.indexOf(2)` // 1
- `nums.size` // 3

Exercice : Les Tableaux

1. Créer un tableau nommé **Fruits**
   - **Initialiser leu avec les Valeurs « Pomme », « Banane », « Poissons », « Mangue »**
2. On s'est rendu compte tardivement que les « Poissons » ne sont pas des fruits: Supprimer le du tableau.
3. Rajouter à la liste de fruits suivante à '**Fruits**' : ['Orange', 'Papaye']
4. Quelle est la nouvelle taille de '**Fruits**'
5. A quelle index se trouve la 'Mangue' ?

Les Collections

- **Les Listes**
  - **Syntaxe :**
    - val maListe: [List[T]] = List(element1, element2,… elementN)
    - Val maListe = element1 :: element2 :: ….. :: ElementN :: Nil
    - val maListe = List()

Exemple :

```
val maListe : List[String] = List("a", "b", "c", "d", "a")
```

NB : Les sont immutables

Les Collections

- **Les Listes**
  - **Ajout de valeurs (avant – après) :**

```
val x = List(1,2,3)
```

- `val y = 0 :: x` // List(0, 1, 2, 3)
- `val z = y ::: List(4,5)` // List(0, 1, 2, 3, 4, 5)
- `val l = List.concat(y, List(4,5))` // List(0, 1, 2, 3, 4, 5)
- `val l2 = y :+ 5` // List(0, 1, 2, 3, 4, 5)

Les Collections

- **Les Listes**
  - **Quelques fonctions natives :**

```
val maListe: List[String] = List("a", "b", "c", "d", "a")
```

- maList.slice(0, 2) ➔ List("a", "b")
- maList.takeRight(1) ➔ z
- maList.filter(x => x !=b) ➔ List("a", "c", "d", "a")
- "q" +: maList :+ "l" ➔ List("q","a", "b", "c", "d", "a", "z", "l")
- maList.contains("b") ➔ true
- maList.length ➔ 6
- maList.count(x => x =="a") ➔ 2
- maList.reverse ➔ List(z, a, d, c, b, a)

Exercice : Les Listes

1. Soit la liste suivante : ["pomme", "banane", "goyave", "banane", "kaki", "pomplemousse"]
    1. Créer la liste et stockée la dans une variable « Fruits »
2. Afficher le 1er et le 3e élément de la liste
3. Ajouter les fruits « ananas » et « pastèque » respectivement au début et à la fin de la liste Fruits et stocker la nouvelle liste dans une liste « Fruits_1 » en une ligne de commande
4. Quelle est la nouvelle taille de « Fruits_1 »
5. Vérifier que « goyave » fait partit de « Fruits_1 »
6. Compter le nombre de fois que « banane » apparait dans « Fruits_1 »
7. Dans une variable « Fruits_sorted », trier les fruits de « Fruits_1 »
8. Supprimer tous les fruits de « Fruits_sorted » dont la taille dépasse 5 caractères et stocker dans une variable « Fruits_small »
9. Afficher « Fruits_small »

Les Collections

- **Les Set**
  - **Syntaxe :**

    - val maSet: [Set[T]] = Set(element1, element2,… elementN)

    Exemple :

    ```
    val maSet : Set[Int] = Set(1, 8, 4, 9)
    ```

NB :

- **Les Set ne contiennent pas de doublons**

Les Collections

- **Les Set**
  - **Ajout et suppression:**

```
val maSet = Set(1,2,3)
```

- `val y = maSet + 0` // Set(1, 2, 3, 0)
- `val z = y ++ Set(4,5)` // Set(0, 1, 2, 3, 4, 5)
- `val l = z - 2` // Set(0, 1, 3, 4, 5)
- `val l2 = l -- Set(1, 2, 3, 0)` // List(4, 5)

Les Collections

- **Les Set**
  - **Quelques fonctions natives :**

```
val maSet1 : Set[Int] = Set(1, 8, 4, 9)
val maSet2 : Set[Int] = Set(1, 8, 5, 0, 12)
```

- maSet1.mkString ("-") ➔ "1-8-4-9"
- maSet1.take(1) ➔ Set(1)
- maSet1.contains(4) ➔ true
- maSet1.apply(10) ➔ false
- maSet1.intersect(maSet2) ➔ Set(1, 8)
- maSet1.diff(maSet2) ➔ Set(4, 9)
- maSet1.drop(1) ➔ Set(8, 4, 9)
- maSet1.filter(_ > 4) ➔ Set(8, 9)

Exercice : Les Set

- Créer un Set contenant les éléments "I","am","trained","to be", "a", "data", "engineer"
- Récupérer les éléments contenant la lettre "a"
- En utilisant Map, transforme le Set en une collection contenant la taille des éléments de l'ensemble.
- filtrer les valeurs paires
- Calculer la somme total de cet ensemble d'entiers

Les Collections

- **Les Tuples**
  - **Syntaxe :**
    - val maTup : TupleX[T1, T2, …] = TupleX(el1, el2, …)

Exemple :

```
val maTup : Tuple3[String,String,Int]=Tuple3("dia", "mor", 28)
```

Les Collections

- **Les Tuples**
  - **Quelques fonctions natives :**

val maTup : Tuple3[String,String,Int]=Tuple3("dia", "mor", 28)

- maTup._1 ➔ "dia"
- maTup._3 ➔ 28
- maTup. productElement(2) ➔ "mor"
- maTup. productArity ➔ 3

Exercice : Les Tuples

- Créer un Tuple "moi" contenant votre prenom, nom, taille, sexe
- Accéder à la votre taille
- Renvoyer la phase suivante: Je suis prenom nom, j'ai une taille de taille et de sexe sexe.

Les Collections

- **Les Map**
  - **Syntaxe :**
    - val maMap: Map[K, V] = Map(k1 -> v1, k2 -> v2, ...)

Exemple :

```
val maMap : Map[String, String] = Map("nom"-> "dia", "prenom"->"mor", "age"->"28")
```

Les Collections

- **Les Listes**
  - **Quelques fonctions natives :**

val maMap: Map[String, String] = Map("nom"-> "dia", "prenom"->"mor", "age"->"28")

- maMap.keys ➔ Set(nom, prenom, age)
- maMap.values ➔ MapLike(dia, mor, 28)
- maMap.getOrElse("sex", "Key unknown !") ➔ " Key unknown !") "
- maMap +("sex" -> "M") ➔ Map("nom"-> "dia", "prenom"->"mor", "age"->"28")
- maMap.filter(_._1 == "age") ➔ Map(age -> 28)
- maList.size ➔ 4

Exercice : Les Map

- Créer le dictionnaire weekDay conntenant les valeurs suivantes : 1=>lundi, 2=>mardi, 3=>mercredi, 4=>jeudi, 5=>vendredi, 6=>samedi, 7=>dimanche
- Recupérer les clefs de weekDay dans une variable weekDayKeys
- Convertir weekDayKeys en liste dans une variable weekDayKeys1
- Filtrer les éléments de weekDayKeys1 qui sont pairs dans une variable weekDayKeys2
- Pour chaque élément de dans une variable weekDayKeys2 afficher la phrase 'La clef … a pour valeur …'
- Récupérer les valeurrs de weekDay dans une variable weekDayVals
- Afficher avec printl 'Les jours de la semaine sont Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche.'

Les structures de contrôle

- **IF – ELSE**

  - Syntaxe générale :

```
If (condition) {
    // code
} else if (condition) {
        // code
}else {
    // code
}
```

Exemple :

```
if (1==2){
            print("true")
}else {
            print("False")
}
```

Les structures de contrôle

- **La boucle For**

  - Syntaxe générale :

```
for( itterator <- collection) {
                    //code
}
```

Exemple :

```
for( i <- 1 to 10) {
        println(i)
}
```

Les structures de contrôle

- **La boucle While**

  - Syntaxe générale :

```
while( expression booléenne) {
                    // code
}
```

Exemple :

```
var i = 1
while( i <=10) {
        println(i)
        i+=1
}
```

Les structures de contrôle

- **La boucle Do - While**

  - Syntaxe générale :

```
do {
                    // code
} while( expression booléenne)
```

Exemple :

```
var i = 1
do {
        println(i)
        i+=1
} while( i <=10)
```

Pattern Matching

- **Match - Case**
  - **Syntaxe générale** :

```
nomVar match {
        case exp1     => instruction1
        case exp2     => instruction2
                      …
        case _        => instruction par défaut
}
```

Exemple :

```
var age = 18
age match {
        case 18                    => println("Vous avez 18 ans.")
        case x if (x < 18)         => println("Vous avez moins de 18 ans.")
        case 19 | 20   => println("Vous avez 19 ou 20 ans.")
        case _                     => println("Vous avais plus de 20 ans")
}
```

Exercices : Les structures de controles

1. Copier la fonction suivante dans votre ide, au niveau de la fonction Main

```
def  randomAge() : Int = {
val r = scala.util.Random
r.nextInt(100)                    //génère un nombre aléatoire compris entre 0 et 100
}
```

   - Déclarer une variable « mon_age » qui est égale randomAge()
   - Vérifier si vous êtes un enfant, un adolescent un adulte ou un papi. En sachant que :
     - 0 < enfant < 9
     - 10 <= adolescent<= 18
     - 18 < adulte <= 60
     - Papi > 60

2. Vérifier les parmi les années 1900 jusqu'à 2030, lesquels sont Bissextiles
   - Tips :
     - Une année est bissextile si elle est divisible par 4 et non par 100 à moins que l'année soit divisible par 400

3. La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent.
   - Pour calculer les termes de la suite de Fibonacci, on fixe les deux premiers termes de la suite :
     - u0=0 u1=1
     - Pour i≥2 on calcule les termes ui à l'aide de la formule : ui=ui−1+ui−2

Les fonctions

- **Syntaxe générale de création d'une fonction**

  - def fonction ([args: DataType]): DataTypeElementDeRetour = {
    //… code
    result
    }

  Ou bien

  - def fonction ([args: DataType]) = result

Exemple:

- def sum(x: Int, y: Int) : Int = {
  val result = x + y
  result
  }

- def sum(x: Int, y: Int) : Int = x + y

- Appel de la fonction : **sum(1,3)** => renvoie 4

Exercices : Les fonctions

- Créer une fonction qui calcule le nombre d'éléments paires dans une liste, tester avec List(3, 12, 16, 32, 54, 5, 23, 87, 98, 52, 99, 24).

- Ecrire une fonction qui prend en entrée un nombre quelconque n et renvoie une liste des nombres d'Amstrong qui sont entre 1 et n.
  - Un nombre est Amstrong si le nombre à la puissance sa longueur est egale à lui même
    - **Ex1**: 100 => $1^3 + 0^3 + 0^3$!= 100 donc 100 n'est pas un nombre Amstrong.
    - **Ex2**: 20 => $2^2 + 0^2$!= 20 donc 200 n'est pas un nombre Amstrong.
    - **Ex3**: 371 => $3^3 + 7^3 + 1^3 = 371$ d'ou 371 est un nombre amstrong

- Créer une fonction qui vérifie si 2 mots sont anagrammes
  - Deux mots S1 et S2 sont anagrammes si toutes les lettres se trouvant dans S1 sont dans S2 et leurs tailles sont égales
  - Tester le code sur les mots suivants :
    - Admise / Samedi
    - Balise / Blaise
    - Centre / Récent
    - Direct / Crédit
    - Équipe / Piquée
    - Entite / Teinte / Tetine
    - Granit / Gratin
    - Ménage / manège
    - Avenir / Navire
    - Cancer / Cancre
    - Cigare / Cirage
    - Dragee / Gardee / Gradee
    - Égerie / Érigée
    - Gerard / Regard / Garder
    - Limace / Malice
    - Orange / Organe

Programmation Orientée Objet