# Programmation Fonctionnelle (Scala)

SOUFIA BENNAI

2022/2023

# Plan du cours

- **Introduction au langage de programmation Scala**
- **Concepts de bases (type, Structures de contrôle,…)**
- Définition de la programmation fonctionnelle
- Immutabilité
- Fonctions pures
- Fonctions d'ordre supérieur

# Introduction

- C’est un langage de programmation de haut niveau
- Syntaxe concise et lisible
- Un langage typé
- Un langage de programmation fonctionnelle
- Un langage orienté objets
- Fusion du paradigme orienté objet de programmation fonctionnelle
- Le code Scala s’exécute sur JVM

# Mon premier code Scala

Il existe deux méthodes pour créer un point de lancement pour une application scala:

- Créer un objet (**object**) qui hérite (**extends**)de ***App***
- Créer un objet dans lequel définir une methode ***main***

```
object c1 extends App {

  println("Bonjour le monde")

}
```

```
object main {

  println("Bonjour le monde")

}
```

# Les variable

- En scala il existe deux modes de definition de variables:

  - Définition **valeur**, on parle de variable immutable (non modifiable).

  Pour declarer une variable immuable, on utilise le mot clé “**Val**”:

  *exemple*:

  ```
  val x = 10 // on ne peut pas modifier le contenu de cette variable après
  x= 12
  ```

  - Définition de **variable** modifiable, avec le mot clés “**var**”: *

  ```
  var y = 13
  y = 10
  ```

# Les variables

- Lors de declaration d'une variable (valeur), il est possible de specifier le type de la variable, comme il est possible de l'omettre.
- Dans le cas ou le type n'est pas spéficié, il faut obligatoirement initialiser la variable. Le type est ensuite déduit à partir de la valeur affectée à cette dernière.
  - Exemple:

```
var x: Int = 0
x = 12
var y = 10.4
var fl = 2.3f
var nom = "Nathalie"
var z : Int =10
```

# Types de variables

- Comme tout autre langage de programmation, en langage scala, il est possible de manipuler plusieurs types de données:
  - 1- Numériques:
    - Short : entiers signés (sur 2octets)
    - Int  : entiers signés (sur 4 octets)
    - Long : entiers signés (sur 8 octets)

      Ex:  var x = 20l
    - Float: flotant à simple precision

      Ex: var y = 10.4f
    - Double: flotant à double precision

      Ex: var d = 10.3e10

# Les variables

- 2- **cacateres**:
  - *Char*: un seul caractère sur 1 octet

    Ex: var ch : char = 'A'
  - *String*: chaine de caractères:

    ex: var str : String = "Bonjour tout le monde"
- **Boolean**: prend deux valeurs True et False
  - ex: var fl = True
- **Autres types**:
  - *Null*: type de pointeur null
  - *Empty*: aucun type
  - *AnyVal*: la variable peut prendre n'importe quelle valeur des types de base.

# Structures de controle

- If/else:

  Syntaxe:

  **if** condition **then**

  {

  bloc instruction1

  }

  **Else**

  { bloc d'instructions 2}

```
val a = 20;
val b = 25

if(a > b)
{
  println("a ="+a)
}
else
{
  println("b = "+b)
}
```

```
b = 25
```

# Input/output

- Pour saisir des valeurs au clavier, on utilise des fonctions de `scala.io.StdIn`

```scala
import scala.io.StdIn._
object c1 extends App {

  println("entrer un nombre entier")
  val val1 = readInt()
  println("val1 = "+val1)

  println("saisir un short: ")
  val val2 = readShort()
  println("val2 = "+val2)

  println("saisir un LONG: ")
  val val4 = readLong()
  println("val4 = "+val4)

  println("saisir un double: ")
  val val3 = readDouble()
  println("val3 = "+val3)

  println("saisir un float: ")
  val val5 = readFloat()
  println("val5 = "+val5)

  println("saisir un caractere: ")
  val val6 = readChar()
  println("val6 = "+val6)

  println("saisir une chaine de caractères: ")
  val val7 = readLine()
  println("val7 = "+val7)

}
```

# Structures de controle

- If/else: contrairement aux autres langages de programmation, en scala, il est possible de retourner le résultat dans une variable:
  - Ex:

```
val a = 20;
val b = 25

val res = if(a > b)
{
  a
}
else
{
  b
}

println("res = "+res)
```

```
b = 25
```

# Structures de controle

- While:
  - Syntaxe:

```
while (condition)
{
        bloc d'instructions;
}
```

```
  var mr :Int = 0
while( mr < 12)
{
  println(mr)
  mr = mr + 2
}
```

```
0
2
4
6
8
10
```

# Structures de controle

- For loops:

  Syntaxe 1:

  **for** (cmp **<-** val_int **to** val_fin)

  {

  bloc d'instructions

  }

  Avec le mot clé "**to**" la boucle s'arrete à cmp =val_fin

```
for (cmp <- 1 to 6)
{
  println("cmp = "+cmp)
}
```

```
cmp = 1
cmp = 2
cmp = 3
cmp = 4
cmp = 5
cmp = 6
```

# Structures de controle

- For loops: (avec le mot clé **until**)

  Syntaxe 2:

  **for** (cmp **<-** val_int **until** val_fin)

  {

  bloc d'instructions

  }

Avec le mot clé "**until**", la boucle s'arrete à cmp = val_fin -1

```
for (cmp <- 1 until 6)
{
  println("cmp = "+cmp)
}
```

```
cmp = 1
cmp = 2
cmp = 3
cmp = 4
cmp = 5
```

# Structures de controle

- For loops:

  Syntaxe 3:

  **for** (p **<-** v1 **to** vf1; p2 **<-** v2 **to** vf2)

  {

  bloc d'instructions

  }

  Cette écriture est l'equivalent de deux boucles imbriquées dans d'autres langages comme le C, Java, …

```
for (cmp <- 1 to 6; cmp2 <- 1 to 10)
 {
   println("cmp = "+cmp+"; cmp2 = "+cmp2)
 }
```

```
cmp = 1; cmp2 = 1
cmp = 1; cmp2 = 2
cmp = 1; cmp2 = 3
cmp = 1; cmp2 = 4
cmp = 1; cmp2 = 5
cmp = 1; cmp2 = 6
cmp = 1; cmp2 = 7
cmp = 1; cmp2 = 8
cmp = 1; cmp2 = 9
cmp = 1; cmp2 = 10
cmp = 2; cmp2 = 1
cmp = 2; cmp2 = 2
cmp = 2; cmp2 = 3
cmp = 2; cmp2 = 4
cmp = 2; cmp2 = 5
cmp = 2; cmp2 = 6
cmp = 2; cmp2 = 7
cmp = 2; cmp2 = 8
cmp = 2; cmp2 = 9
cmp = 2; cmp2 = 10
cmp = 3; cmp2 = 1
cmp = 3; cmp2 = 2
cmp = 3; cmp2 = 3
cmp = 3; cmp2 = 4
cmp = 3; cmp2 = 5
cmp = 3; cmp2 = 6
cmp = 3; cmp2 = 7
cmp = 3; cmp2 = 8
cmp = 3; cmp2 = 9
cmp = 3; cmp2 = 10
```

# Structures de controle

- For loops: permet aussi de parcourir les elements d'une structure de données sans utiliser un compteur:

ex:

```
val liste_ = List(1, 4, 6, 0)
for (elet <- liste_)
  {
    println(elet)
  }
```

```
1
4
6
0
```

# Structures de controle

- For loops: il est possible d'utiliser des conditions à l'interieur d'une boucle for:

  ex:

```
val liste_ = List(1, 4, 6, 0)
for (elet <- liste_; if elet> 3)
  {
    println(elet)
  }
```

```
4
6
```

# Structures de controle

- For loops: il est possible de retourner dans une variable le résultat d'une boucle for. Cela en utilisant le mot clés "**yield**"

  ex:

```
val liste1 = List(1, 4, 6, 0)
val res = for (elet <- liste1) yield {elet * elet}
println("Résultat de la boucle: "+res)
```

```
Résultat de la boucle: List(1, 16, 36, 0)
```

# Structure de controle

- Match/case: l'expression match est l'equivalent de switch/case en algorithmique.
  - Syntaxe:

    valeur **match** :

    **case** v1 **=>** instructionsA

    **case** v2 **=>** instructionsB

    **case** vn **=>** instructionsZ

    **case** _ **=>** instructions dans le cas par defaut

# Structure de controle

- Match/case: Il est possible de retourner le resultat de Match dans une variable:
  - Exemple:

```
print("entrer un entier: ")
val x = readInt()
val res = x match
{
  case 10 => x
  case 20 => x
  case 30 => x
  case _    => "ERREUR"
}
println("res:  "+res)
```

```
entrer un entier: 2
res:  ERREUR
```

# Structure de controle

- Match/case: Il est possible de rassembler plusieurs case dans un seul
  - Exemple:

```
val x = readInt()
val res = x match
{
    case  10|20|30|40 |50 => "multiples de 10"
    case 3 | 9 |15 => "multiples de 3"
    case _     => "ERREUR"
}

println("res = "+res)
```

```
9
res = multiples de 3
```