

# Syntaxe de Base

Jean-Luc Falcone

Septembre 2017

# Variables

Java

```
int i = 3;  
String s = "Hello !";
```

# Variables

## Java

```
int i = 3;  
String s = "Hello !";
```

## Scala

```
var i: Int = 3;  
var s: String = "Hello !";
```

# Variables

## Java

```
int i = 3;  
String s = "Hello !";
```

## Scala

```
var i: Int = 3;  
var s: String = "Hello !";
```

Grâce à l'inférence des types et des points virgules:

```
var i = 3  
var s = "Hello !"
```

# Valeurs

Java

```
final int i = 3;  
final String s = "Hello !";
```

# Valeurs

## Java

```
final int i = 3;  
final String s = "Hello !";
```

## Scala

```
val i = 3  
val s = "Hello !"
```

# Blocs de code

## Scala

```
val a = 1
val b = 2
val c = {
  println( "Computing" )
  val d = 10 - a
  d*d + b
}
```

# Méthodes

Java

```
double pythagore( double a, double b ) {  
    final double c2 = a*a + b*b;  
    return Math.sqrt(c2);  
}
```



# Méthodes

## Java

```
double pythagore( double a, double b ) {  
    final double c2 = a*a + b*b;  
    return Math.sqrt(c2);  
}
```

## Scala

```
def pythagore( a: Double, b: Double ): Double = {  
    val c2 = a*a + b*b  
    return math.sqrt(c2)  
}
```

# Méthodes

## Java

```
double pythagore( double a, double b ) {  
    final double c2 = a*a + b*b;  
    return Math.sqrt(c2);  
}
```

## Scala

```
def pythagore( a: Double, b: Double ): Double = {  
    val c2 = a*a + b*b  
    return math.sqrt(c2)  
}  
  
def pythagore( a: Double, b: Double ) = {  
    val c2 = a*a + b*b  
    math.sqrt(c2)  
}
```

# Remarques

- Inférence du type de retour, pratique sauf:
  - recursion
  - return
  - *méthodes publiques*
  - *méthodes non-triviales*
- Eviter le return, plus lents, effets de bords

## Scala: Méthodes simples

```
def add( i: Int, j: Int ) = i+j
```

# Invocation de méthode

```
val sum = add( 2, 51 )  
val hypotenuse = pythagore( 4, 1.0 )
```

# Multi-parameter lists

```
def add( i: Int )( j: Int ) = i+j
```

```
add(2)(5)
```

# Méthodes Génériques


```
def isSizeEven[A]( as: List[A] ) =  
  as.size % 2 == 0
```

# Procédures (java)

```
void saveAndQuit( State state ) {  
    System.out.println( "Saving state..." );  
    save( state );  
    System.out.println( "Bye" );  
    quit();  
}
```

# Procédures (scala)

```
def saveAndQuit( state: State ): Unit = {  
  println( "Saving state..." )  
  save( state )  
  println( "Bye" )  
  quit()  
}
```





# Procédures (scala)

## Deprecated

```
def saveAndQuit( state: State ) {  
  println( "Saving state..." )  
  save( state )  
  println( "Bye" )  
  quit()  
}
```

# Paramètres par défaut (scala)

```
def brew( coffee: String = "Espresso" ): Unit =  
  println( "Brewing: " + coffee )
```

```
brew()
```

```
brew( "Capuccino" )
```

# Paramètres nommés (scala)

```
def fullname( surname: String, firstname: String ) =  
    firstname + " " + surname
```

```
fullname( "Knuth", "Donald" )
```

```
fullname( firstname="Donald", surname="Knuth" )
```

# While (java)

```
double average( int[] ary ) {  
    final int n = ary.length;  
    int i = 0;  
    int sum = 0;  
    while( i < n ) {  
        sum += ary[i];  
        i++;  
    }  
    return 1.0*sum / n;  
}
```

# While (scala)

```
def average( ary: Array[Int] ): Double = {  
  val n = ary.size  
  var i = 0  
  var sum = 0  
  while( i < n ) {  
    sum += ary(i)  
    i += 1  
  }  
  sum.toDouble / n  
}
```

```
def average2( ary: Array[Int] ) =  
  ary.sum.toDouble / ary.size
```

# If-Else (java)

```
double x = f(1.0);  
int a = 0;  
if( x < 2.5 ) {  
    a = 1;  
} else {  
    a = 2;  
}
```

# If-Else (scala)

```
val x = f(1.0)
val a = if( x < 2.5 ) 1 else 2

if( x < 0 ) {
  println( "x is negative..." )
}

val b = if( a == x ) {
  println("Idem")
  a
} else {
  val y = -x
  a * y
}
```

# Switch/case (java)

```
int day = getDay();
String dayString;
switch (day) {
    case 1:  dayString = "Lundi"; break;
    case 2:  dayString = "Mardi"; break;
    case 3:  dayString = "Mercredi"; break;
    case 4:  dayString = "Jeudi"; break;
    case 5:  dayString = "Vendredi"; break;
    case 6:  dayString = "Samedi"; break;
    case 7:  dayString = "Dimanche"; break;
}
System.out.println(dayString);
```



# Match/Case (scala)

```
val day = getDay()  
val dayString = day match {  
  case 1 => "Lundi"  
  case 2 => "Mardi"  
  case 3 => "Mercredi"  
  case 4 => "Jeudi"  
  case 5 => "Vendredi"  
  case 6 => "Samedi"  
  case 7 => "Dimanche"  
}  
println( dayString )
```

# Match/Case (scala)

```
val number: Int = f()
val kind = number match {
  case 0          => "Zero"
  case 1|2|3      => "Small"
  case i if i > 0 => "Positive: " + i
  case _         => "Negative"
}
```

# Match/Case (scala)

```
def asInt( a: Any ) = a match {  
  case s: String => s.toInt  
  case i: Int    => i  
  case b: Boolean => if(b) 1 else 0  
  case _ =>  
    throw new Exception("Conversion error")  
}
```

# Try-catch (java)

```
try {  
    //...  
} catch (FileNotFoundException e) {  
    System.err.println(  
        "FileNotFoundException: " + e.getMessage()  
    );  
    throw new SampleException(e);  
  
} catch (IOException e) {  
    System.err.println(  
        "Caught IOException: " + e.getMessage()  
    );  
}
```

# Try-catch (scala)

```
try {  
  //...  
} catch {  
  case e: FileNotFoundException => {  
    Console.err.println(  
      "FileNotFoundException: " + e.getMessage()  
    )  
    throw new SampleException(e)  
  }  
  case e: IOException => Console.err.println(  
    "Caught IOException: " + e.getMessage()  
  )  
}
```

# Try-catch (scala)

```
def probability( s: String ) = try {  
  val x = s.toDouble  
  if( x <= 1 && x >= 0 ) x else -1.0  
} catch {  
  case _: NumberFormatException => -1.0  
}
```