Table des matières

[**I.** **Les types numériques dans Scala** 2](#_Toc117939066)

[**II.** **Quelques fonctions du type Integer** 2](#_Toc117939067)

[**III.** **Les opérateurs logiques** 4](#_Toc117939068)

[**A.** **Le Type booléen** 5](#_Toc117939069)

[**B.** **Le type String** 6](#_Toc117939070)

[**a.** **Opérations sur les String** 7](#_Toc117939071)

[**C.** **Travailler avec le type Null** 9](#_Toc117939072)

# **Les types numériques dans Scala**

Les types numériques dans Scala traite de tous les types stockés sous forme numérique. Les types de données utilisés est décimal (float et Double) et entier (Int, Short, Long).

Chacun de ces types possède ces spécificités, découvrons-les :

* **int**: Le **type de données int** stocke des variables entières qui occupent un emplacement mémoire de **4** octets. La valeur du **type de données int** est comprise entre -2147483648 et 2147483647.
* **short**: Le **type de données short** stocke la valeur entière des variables qui occupe un emplacement mémoire de **2** octets. La valeur du **type de données court** est comprise entre -32768 et 32767.
* **long**: Le **type de données long** stocke une valeur entière dans des variables qui prennent un emplacement mémoire de **8** octets. La valeur du **type de données long** est comprise entre -9223372036854775808 et 9223372036854775807.
* **float**: Le **type de données flottant** stocke des valeurs décimales dans ses variables qui prennent un emplacement mémoire de **4** octets. La valeur du **type de données float** est comprise entre -3,4E+38 et +3,4E+38, c'est-à-dire en simple précision.
* **double**: Le **type de données double** stocke des valeurs décimales dans ses variables qui prennent un emplacement mémoire de **8** octets. La valeur du **type de données flottant** est comprise entre le flotteur double précision IEEE 754.

**Exemples**:

var tour : Int = 22

var tour : Short = 2245

var tour : Long = 2245

var tour : Float = 22.45

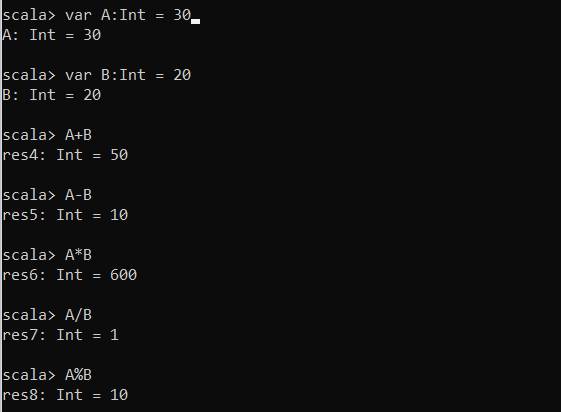
var tour : Double = 46763.8763

# **Quelques fonctions du type Integer**

Scala prend en charge les opérateurs arithmétiques ci-dessous

On suppose deux variable de types int **A =30** et **B = 20**

Dans le REPL on aura



|  |  |  |
| --- | --- | --- |
| **Opérateur** | **La description** | **Exemple** |
| + | Ajoute deux opérandes | A + B donnera 50 |
| - | Soustrait le deuxième opérande du premier | A - B donnera 10 |
| \* | Multiplie les deux opérandes | A \* B donnera 600 |
| / | Divise le numérateur par le dénominateur | B/A donnera 0,666 |
| % | L'opérateur de modulo trouve le reste après la division d'un nombre par un autre | B % A donnera 20 |

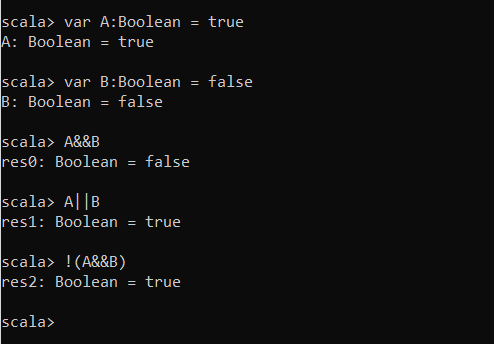
Si on se réfère au site [https://www.alphacodingskills.com](https://www.alphacodingskills.com/scala/notes/scala-operators-precedence.php) la priorité opératoire est fait

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Associativity** |
| ()  [] | Postfix | Left to Right |
| !  ~ | Unary operators - Logical AND, Bitwise NOT | Right to Left |
| \*  /  % | Multiplication, Division, Remainder | Left to Right |
| +  - | Addition, Subtraction |
| <<  >>  >>> | Bitwise left shift, right shift and unsigned right shift |
| <  <=  >  >= | Less than, Less than or equal, Greater than, and Greater than or equal |
| ==  != | Equality and Inequality |
| & | Bitwise AND |
| ^ | Bitwise XOR |
| | | Bitwise OR |
| && | Logical AND |
| || | Logical OR |
| = | Direct assignment | Right to Left |
| +=  -=  \*=  /=  %= | Compound assignment by sum, difference, product, quotient and remainder |
| <<=  >>= | Compound assignment by Bitwise left shift and right shift |
| &=  ^=  |= | Compound assignment by Bitwise AND, XOR and OR |
| , | Comma (separate expressions) | Left to Right |

# **Les opérateurs logiques**

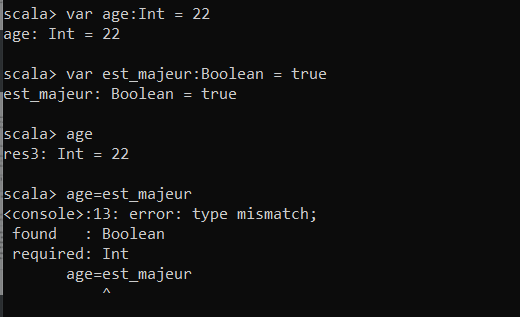
On suppose l’existence de deux variables **A** et **B** telles que : A=1 et B=0

Dans le REPL on a :

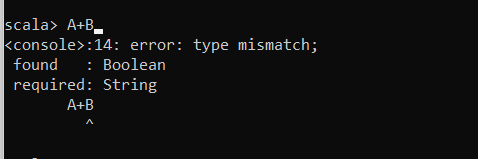
**Explications**

|  |  |  |
| --- | --- | --- |
| **Opérateurs** | **Description** | **Exemple** |
| && | C'est ce qu'on appelle l'opérateur ET logique. Si les deux opérandes sont non nuls, la condition devient vraie. | (A && B) = false. |
| || | C'est ce qu'on appelle l'opérateur OU logique. Si l'un des deux opérandes est différent de zéro, la condition devient vraie. | (A || B) = true. |
| ! | C'est ce qu'on appelle l'opérateur NON logique. Utilisez pour inverser l'état logique de son opérande. Si une condition est vraie, l'opérateur NON logique deviendra faux. | !(A && B) = true. |

## **Le Type booléen**

En essayant d'affecter une variable booléenne à une variable entière on obtient le résultat ci-après

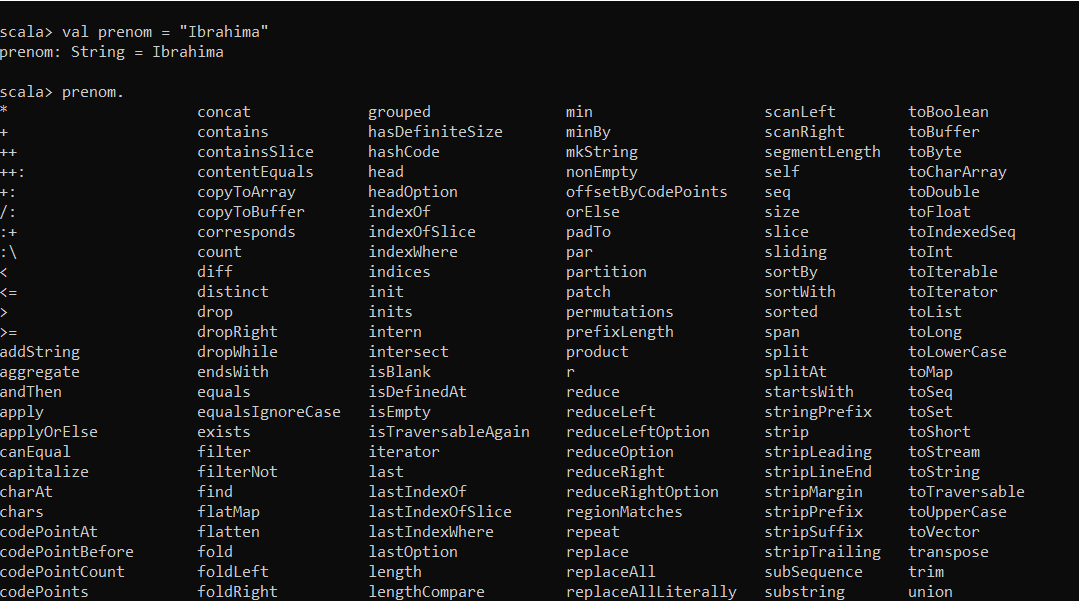
**Addition de 2 booléen**

****

En additionnant ces deux booléen, on obtient une erreur

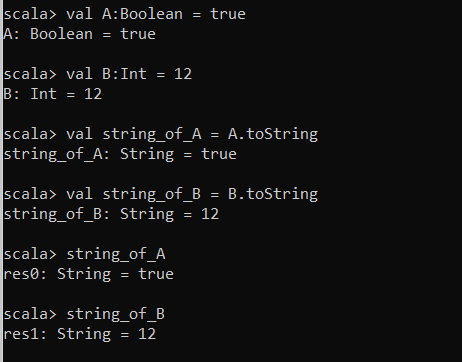
## **Le type String**

Voyons une liste des méthodes disponible pour une variable de type String



Essayons de convertir une variable **Booléen** en **String**. En considérant les variables **A = true** et **B=12,** pour convertir en String on utilise la méthode **toString** de ces variables.

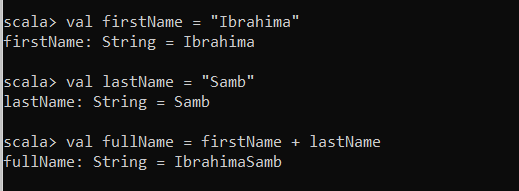
**Ex :**

****

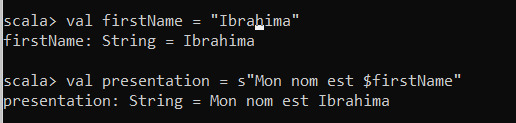
### **Opérations sur les String**

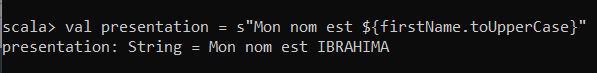
Nous allons découvrir ici quelques opérations les plus utilisées :

* **Concaténation**: la concaténation est le fait de lier deux chaines de caractères

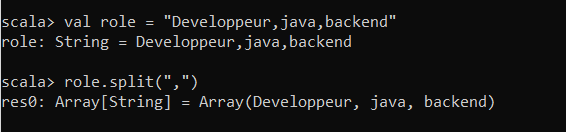


* **Interpolation**: l’interpolation est l’action de remplacer une variable par sa valeur dans une chaine lorsque celle si débute par un **s**



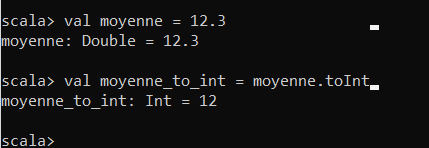
Ainsi, nous pourrons effectuer des transformations sur la variable **firstName** dans **presentation** comme ci-dessous

* **Séparation des chaines :** c’est l’opération qui consiste à séparer une chaine grâce à un délimiteur bien déterminé.



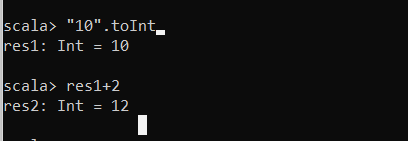
**Exercices :**

* Convertir **double** en **int** :



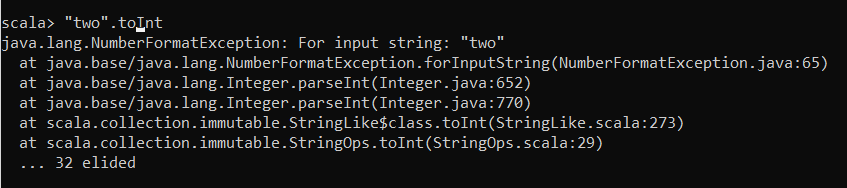
On constate une suppression de la partie décimale

* Convertir **string** en **int**:



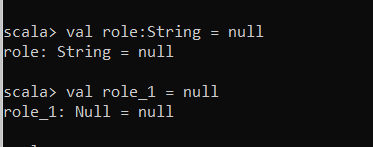
Le contenu de la variable est transformé en **int,** nous pouvons effectuer des opérations de calculs dessus.

**NB :** cette opération n’est possible que si le contenu de la variable est un numérique. Dans le cas contraire, une erreur se produira comme dans cette image



## **Travailler avec le type Null**

Tout comme les autres types, on peut créer une variable de type null en procédant de deux façons : en utilisant le type **Null** directement, soit affectant **null** à notre variable et laisser scala déterminer lui-même le type.



Dans certains cas, nous aurions besoin de rendre optionnel la valeur d’une variable qui peut être nulle. Pour ce faire, scala nous fournis deux possibilité : **Some** et **None** qui hérite tous les deux de la classe **Option**.

