Types de données dans Scala

Par: Ibrahima Khalilou Lahi Samb

Table des matières

I. Les types numériques dans Scala 2

II. Quelques fonctions du type Integer 2

III. Les opérateurs logiques 4

A. Le Type booléen 5

B. Le type String 6

a. Opérations sur les String 7

C. Travailler avec le type Null 9

Types de données dans Scala

Par: Ibrahima Khalilou Lahi Samb

I. 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
```

II. 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

Types de données dans Scala

Par: Ibrahima Khalilou Lahi Samb

```
scala> var A:Int = 30
A: Int = 30

scala> var B:Int = 20
B: Int = 20

scala> A+B
res4: Int = 50

scala> A-B
res5: Int = 10

scala> A*B
res6: Int = 600

scala> A/B
res7: Int = 1

scala> A%B
res8: Int = 10
```

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 la priorité opératoire est fait

Types de données dans Scala

Par: Ibrahima Khalilou Lahi Samb

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

III. 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 :

Types de données dans Scala

Par: Ibrahima Khalilou Lahi Samb

```
scala> var A:Boolean = true
A: Boolean = true

scala> var B:Boolean = false
B: Boolean = false

scala> A&&B
res0: Boolean = false

scala> A||B
res1: Boolean = true

scala> !(A&&B)
res2: Boolean = true

scala>
```

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.

A. Le Type booléen

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

```
scala> var age:Int = 22
age: Int = 22

scala> var est_majeur:Boolean = true
est_majeur: Boolean = true

scala> age
res3: Int = 22

scala> age=est_majeur
<console>:13: error: type mismatch;
 found   : Boolean
 required: Int
       age=est_majeur
           ^
```

ci-après

Types de données dans Scala

Par: Ibrahima Khalilou Lahi Samb

Addition de 2 booléen

```
scala> A+B
<console>:14: error: type mismatch;
 found   : Boolean
 required: String
       A+B
         ^
```

En additionnant ces deux booléen, on obtient une erreur

B. Le type String

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

```
scala> val prenom = "Ibrahima"
prenom: String = Ibrahima

scala> prenom.
*                 concat            grouped             min                 scanLeft          toBoolean
+                 contains          hasDefiniteSize     minBy               scanRight         toBuffer
++                containsSlice     hashCode            mkString            segmentLength     toByte
++:               contentEquals     head                nonEmpty            self              toCharArray
+:                copyToArray       headOption          offsetByCodePoints  seq               toDouble
/:                copyToBuffer      indexOf             orElse              size              toFloat
:+                corresponds       indexOfSlice        padTo               slice             toIndexedSeq
:\                count             indexWhere          par                 sliding           toInt
<                 diff              indices             partition           sortBy            toIterable
<=                distinct          init                patch               sortWith          toIterator
>                 drop              inits               permutations        sorted            toList
>=                dropRight         intern              prefixLength        span              toLong
addString         dropWhile         intersect           product             split             toLowerCase
aggregate         endsWith          isBlank             r                   splitAt           toMap
andThen           equals            isDefinedAt         reduce              startsWith        toSeq
apply             equalsIgnoreCase  isEmpty             reduceLeft          stringPrefix      toSet
applyOrElse       exists            isTraversableAgain  reduceLeftOption    strip             toShort
canEqual          filter            iterator            reduceOption        stripLeading      toStream
capitalize        filterNot         last                reduceRight         stripLineEnd      toString
charAt            find              lastIndexOf         reduceRightOption   stripMargin       toTraversable
chars             flatMap           lastIndexOfSlice    regionMatches       stripPrefix       toUpperCase
codePointAt       flatten           lastIndexWhere      repeat              stripSuffix       toVector
codePointBefore   fold              lastOption          replace             stripTrailing     transpose
codePointCount    foldLeft          length              replaceAll          subSequence       trim
codePoints        foldRight         lengthCompare       replaceAllLiterally substring         union
```

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 :

Types de données dans Scala

Par: Ibrahima Khalilou Lahi Samb

```
scala> val A:Boolean = true
A: Boolean = true

scala> val B:Int = 12
B: Int = 12

scala> val string_of_A = A.toString
string_of_A: String = true

scala> val string_of_B = B.toString
string_of_B: String = 12

scala> string_of_A
res0: String = true

scala> string_of_B
res1: String = 12
```

a. 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

```
scala> val firstName = "Ibrahima"
firstName: String = Ibrahima

scala> val lastName = "Samb"
lastName: String = Samb

scala> val fullName = firstName + lastName
fullName: String = IbrahimaSamb
```

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

```
scala> val firstName = "Ibrahima"
firstName: String = Ibrahima

scala> val presentation = s"Mon nom est $firstName"
presentation: String = Mon nom est Ibrahima
```

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

```
scala> val presentation = s"Mon nom est ${firstName.toUpperCase}"
presentation: String = Mon nom est IBRAHIMA
```

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

```
scala> val role = "Developpeur,java,backend"
role: String = Developpeur,java,backend

scala> role.split(",")
res0: Array[String] = Array(Developpeur, java, backend)
```

Exercices :

- Convertir **double** en **int** :

```
scala> val moyenne = 12.3
moyenne: Double = 12.3

scala> val moyenne_to_int = moyenne.toInt
moyenne_to_int: Int = 12

scala>
```

On constate une suppression de la partie décimale

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

```
scala> "10".toInt
res1: Int = 10

scala> res1+2
res2: Int = 12
```

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

```
scala> "two".toInt
java.lang.NumberFormatException: For input string: "two"
  at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
  at java.base/java.lang.Integer.parseInt(Integer.java:652)
  at java.base/java.lang.Integer.parseInt(Integer.java:770)
  at scala.collection.immutable.StringLike$class.toInt(StringLike.scala:273)
  at scala.collection.immutable.StringOps.toInt(StringOps.scala:29)
  ... 32 elided
```

C. 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.

```
scala> val role:String = null
role: String = null

scala> val role_1 = null
role_1: Null = null
```

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**.

```
scala> val phrase_1 = "Bonjour, moi c'est Ibrahima"
phrase_1: String = Bonjour, moi c'est Ibrahima

scala> val phrase_2 = null
phrase_2: Null = null

scala> val opt_phrase_1 = Option(phrase_1)
opt_phrase_1: Option[String] = Some(Bonjour, moi c'est Ibrahima)

scala> val opt_phrase_2 = Option(phrase_2)
opt_phrase_2: Option[Null] = None
```