

- 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.
- 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 Short va de -32768 à 32767.
- Le type de données Long stocke des valeurs entières dans des variables qui occupent un emplacement mémoire de 8 octets. La valeur du type de données longues va de -9223372036854775808 à 9223372036854775807.
- Le type de données Float stocke des valeurs décimales dans ses variables qui occupent 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.
- Le type de données Double stocke des valeurs décimales dans ses variables qui occupent un emplacement mémoire de 8 octets.

Ainsi le choix du type de données dépend de la marge des valeurs à utiliser dans notre programme. Par exemple on ne peut choisir Int à la place de Long si on doit stocker de très grande valeur entière dans nos variables. Comme aussi il est préférable de choisir Float à la place de Double si les décimales à traiter ne sont pas grande afin d'économiser de la mémoire.

- Il existe plusieurs fonctions disponibles pour le type Int comme vous pouvez le voir avec l'image ci-dessous et nous en utilisons quelques-unes en exemple.

```

scala> val x = 10
x: Int = 10

scala> x.
!= + << >= abs compareTo getClass isNaN isValidChar isWhole round to toDegrees toInt toShort underlying
% - <= >> byteValue doubleValue intValue isNegInfinity isValidInt longValue max self shortValue to toBinaryString toDouble toLong toOctalString unary_+ until
& / == >>> ceil floatValue isInfinte isPosInfinity isValidLong min signum to toByte toFloat toHexString toRadians unary_- |
* < > ^ compare floor isInfinity isValidByte isValidShort

scala> val y = 21
y: Int = 21

scala> x + y
res3: Int = 31

scala> x - y
res4: Int = -11

scala> x < y
res5: Boolean = true

scala> x > y
res6: Boolean = false

scala> x / y
res7: Int = 0

scala> y / x
res8: Int = 2

scala>

```

- Dans une expression numérique on résout d'abord les opérations à l'intérieur des parenthèses ou des crochets. Ensuite tous les exposants s'ils y'en a, puis toutes les multiplications et divisions de gauche à droite et enfin toutes les additions et soustractions de gauche à droite.

BOOLEAN TYPE

- En Scala, nous pouvons trouver ces trois opérateurs logiques: &&(AND), ||(OR) et !(NOT).

A screenshot of a Scala REPL terminal window. The window has a dark background with a sidebar on the left containing icons for various applications. The terminal text shows several Scala commands and their results: 'scala> val a = true' followed by 'a: Boolean = true'; 'scala> val b = false' followed by 'b: Boolean = false'; 'scala> a && b' followed by 'res9: Boolean = false'; 'scala> a || b' followed by 'res10: Boolean = true'; 'scala> !a' followed by 'res11: Boolean = false'; 'scala> !b' followed by 'res12: Boolean = true'; and finally 'scala>' with a cursor. The top right corner of the window shows the username 'momo@momo: ~'.

- Quand on essaye d'assigner une variable de type Int à une variable de type Boolean on obtient une erreur en Scala ce qui n'est pas le cas avec le langage python par exemple qui n'est pas un langage fortement typé.
- On ne peut pas utiliser les opérations arithmétique sur des variables de type Boolean en Scala

STRING TYPES

- Nous avons une liste de fonctions que nous pouvons utiliser sur les chaînes de caractères comme le montre l'image ci-dessous. Par exemple nous pouvons utiliser la fonction `charAt` avec un index pour récupérer un caractère à une position donnée. De même nous la fonction `toUpperCase` pour la conversion d'une chaîne en majuscule...

```
momo@momo: ~  
scala> val myString = "Ceci est une chaine de caracteres"  
myString: String = Ceci est une chaine de caracteres  
  
scala> myString.  
*      capitalize      contentEquals      flatten      indexOfSlice      lengthCompare      patch      reverse      split      takeWhile      toSet  
+      charAt          copyToArray       fold         indexWhere       lift              permutations  reverseIterator  splitAt         to      toShort  
++     chars          copyToBuffer     foldLeft     indices          lines            prefixLength    reverseMap       startsWith     toArray    toStream  
++:    codePointAt     corresponds      foldRight   init            linesIterator   product        runWith         stringPrefix  toBoolean  toString  
+:     codePointBefore count           forall      inits           linesWithSeparators r              sameElements    strip         toBuffer  toTraversable  
/=:   codePointCount  diff            foreach     intern          map             reduce          scan           stripLeading   toByte    toUpperCase  
:+    codePoints     distinct        format      intersect       matches         reduceLeft     scanLeft       stripLineEnd  toCharArray toVector  
:\    collect        drop            formatLocal isBlank         max            reduceLeftOption scanRight      stripMargin   toDouble  transpose  
<    collectFirst  dropRight       genericBuilder isEmptyAt        maxBy          reduceOption   segmentLength  stripPrefix  toFloat   trim  
<=   combinations  dropWhile      getBytes    isDefinedAt     min           reduceRight    self          stripSuffix  toIndexedSeq union  
>    companion     endsWith       getChars    isTraversableAgain minBy          reduceRightOption seq           stripTrailing toInt      unzip  
>=   compare        equals         groupBy     iterator        mkString       regionMatches  size          subsequence  toIterable unzip3  
addString compareTo     equalsIgnoreCase exists        grouped        last           nonEmpty      repeat        slice        substring  toIterator updated  
aggregate compareToIgnoreCase exists        hasDefiniteSize lastIndexOf    lastIndexOfSlice orElse        replace        sliding       sum         toList     view  
andThen  compose      filter        hashCode       lastIndexOfWhere lastIndexOfSlice orElse        replaceAll     sort        tail       toLong    withFilter  
apply    concat     filterNot     head          lastOption     lastOption     padTo         replaceAllLiterally sortBy      sortWith   tails     toLowerCase zip  
applyOrElse contains      flatMap       headOption    lastOption     lastOption     par           replaceFirst  sorted     sorted    take      toMap     zipAll  
canEqual containsSlice flatMap       indexOf       length          length          partition      repr          span        take       takeRight toSeq     zipWithIndex  
  
scala> myString.
```

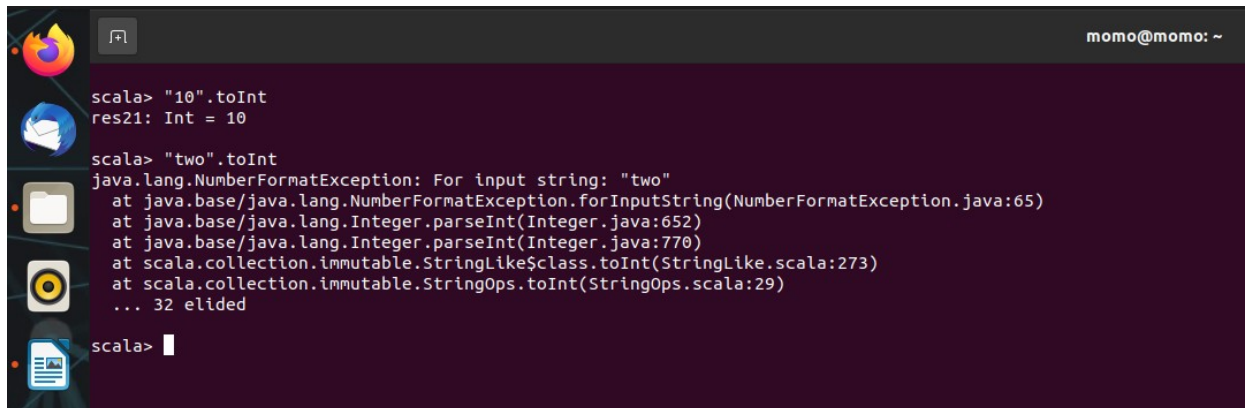
- Avec la fonction `toString()` nous pouvons convertir une variable de type `Int` en `String` de même qu'une variable de type `Boolean` en `String`.

```
scala> val x = 54  
x: Int = 54  
  
scala> x.toString()  
res19: String = 54  
  
scala> val y = false  
y: Boolean = false  
  
scala> y.toString()  
res20: String = false  
  
scala> 
```

TYPE CASTING

- Lorsqu'on essaye de convertir une variable de type `Double` en `Int` on aura juste la partie entière du `Double`.

- Comme nous pouvons le voir avec l'image ci-dessous on ne peut pas caster toutes les valeurs des variables.



```
momo@momo: ~  
scala> "10".toInt  
res21: Int = 10  
  
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  
  
scala> 
```

- En Scala, l'utilisation de null pour représenter des valeurs nullable ou manquantes n'est pas conseillé on utilise plutôt le type Option. Le type Option permet de traiter à la fois la présence et l'absence d'un élément.