Version du logiciel : 1.3

Version du support : 1.0.1

Date de création : 2 novembre 2018
Date de dernière mise à jour : 1 mars 2020

IDE utilisées :

- Notepad++ pour les « applications » (.kt).
- IntelliJ de JetBrains pour les projets Kotlin avec des scripts (.kts).

Table des matières

1.1 - Objectif de ce document

Introduire à Kotlin.

1.1.1 - Définition

https://kotlinlang.org/

Kotlin est un langage de programmation – **objet et fonctionnel** - pour JVM, Android, Browser, Native(C, …), JavaScript, …
Il est principalement développé par une équipe de programmeurs de chez JetBrains basée à Saint-Pétersbourg (son nom vient de l'île de Kotline, proche de St. Pétersbourg).
En 2017 Google annonce pendant la conférence Google I/O que Kotlin devient le second langage de programmation officiellement pris en charge par Android après Java.
En 2019 lors de la conférence Google I/O, Kotlin devient officiellement le langage de programmation voulu et recommandé par Google pour le développement d'applications Android.

Kotlin convient parfaitement au développement d'applications côté serveur. Il permet d'écrire un code expressif et concis tout en maintenant une compatibilité totale avec les piles de technologies Java existantes et une courbe d'apprentissage fluide.

Expressivité: les fonctionnalités langagières innovantes de Kotlin, telles que la prise en charge des générateurs respectueux du type et des propriétés déléguées, permettent de créer des abstractions puissantes et faciles à utiliser.
Évolutivité: la prise en charge par Kotlin des co-routines permet de créer des applications côté serveur pouvant s'adapter à un nombre considérable de clients avec une configuration matérielle modeste.
Interopérabilité: Kotlin est entièrement compatible avec tous les frameworks basés sur Java, ce qui vous permet de rester sur votre pile technologique habituelle tout en profitant des avantages d'un langage plus moderne.
Migration: Kotlin prend en charge la migration progressive, étape par étape, de bases de code volumineuses de Java à Kotlin. Vous pouvez commencer à écrire du nouveau code dans Kotlin tout en conservant les anciennes parties de votre système en Java.
Outils: outre le support IDE général en général, Kotlin propose des outils spécifiques au framework (par exemple, pour Spring) dans le plug-in d'IntelliJ IDEA Ultimate.
Learning Curve: pour un développeur Java, démarrer avec Kotlin est très facile. Le convertisseur automatisé Java vers Kotlin inclus dans le plugin Kotlin facilite les premières étapes. Kotlin Koans propose un guide des principales caractéristiques du langage avec une série d'exercices interactifs.

1.1.2 - Bibliographie et Webographie

Bibliographie

https://kotlinlang.org/docs/books.html

En français :

Un seul livre (ENI éditions) ; ce livre est pour les développeurs Android mais comporte des chapitres sur kotlin « pur et dur ».

« Kotlin, les fondamentaux du développement d'applications Android »

Éditeur(s)	Eni
Auteur(s)	Anthony Cosson
Collection	Epsilon
Parution	10/10/2018
Nb. de pages	466
EAN13	9782409015861
ISBN13	978-2-409-01586-1
Prix	54 €

Webographie

Tutoriel officiel :
https://kotlinlang.org/docs/reference/basic-syntax.html

Référence officielle :
https://kotlinlang.org/docs/reference/

Wikipedia :
https://fr.wikipedia.org/wiki/Kotlin_(langage)

1.2 - Téléchargement et installation

1.2.1 - IntelliJ on Windows

https://www.jetbrains.com/idea/download/download-thanks.html?platform=windows&code=IIC

Projets : C:\pascal___supports\kotlin\projets_kotlin

1.2.2 - Le compilateur Kotlin

Pour travailler directement à la console !

https://kotlinlang.org/docs/tutorials/command-line.html

Avec Windows :

Downloading the compiler.
Every release ships with a standalone version of the compiler.
We can download the latest version (kotlin-compiler-1.3.61.zip) from GitHub Releases.
https://github.com/JetBrains/kotlin/releases/tag/v1.3.61

Manual Install :
Unzip the standalone compiler into a directory and optionally add the bin directory to the system path. The bin directory contains the scripts needed to **compile and run Kotlin** on Windows, OS X and Linux.

Compilateur : kotlinc
Runner : la JVM donc java -jar … (cf plus bas).

Plate-forme DELL

Compilateur :
\pascal___supports\kotlin\kotlin-compiler-1.3.61\bin\kotlinc

Scripts :
cd \pascal___supports\kotlin\scripts\

Plateforme M2i

Idem.

1.2.3 - Travailler à la console

Creating and running a first application.
Create a simple application in Kotlin that displays Hello, World!. Using our favorite editor, we create a new file called **hello.kt** with the following :

```
fun main(args: Array<String>) {
    println("Hello, World!")
}
```

Target platform : JVMRunning on kotlin v. 1.3.61.
Compile the application using the Kotlin compiler.

```
$ kotlinc hello.kt -include-runtime -d hello.jar
```

The -d option indicates the output path for generated class files which may be either a directory or a .jar file. The -include-runtime option makes the resulting .jar file self-contained and runnable by including the Kotlin runtime library in it.

Cf page suivante compilation et exécution.

Note :

Les fichiers .kt et les fichiers .kts !

Un fichier .kt est un fichier de code hors d'un projet.
Un fichier .kt doit contenir une méthode « main ».

```
// hello.kt
fun main(args: Array<String>) {
    println("Hello, World!")
}
```

Un fichier .kts est un fichier de script.
Un fichier .kts ne contient pas de méthode « main ».

```
// hello.kts
println("Hello, World!")
```

1.2.4 - Compilation et exécution à la console

Pour les fichiers .kt.

c.bat

```
/pascal/___supports/kotlin/kotlin-compiler-1.3.61/bin/kotlinc %1.kt -include-
runtime -d %1.jar
```

r.bat

```
java -jar %1.jar
```

1.3 - Quelques remarques générales sur le langage

Kotlin est « case sensitive ».
Chaque instruction n'est pas terminée par un point-virgule (;).
Mais si vous voulez coder 2 instructions sur la même ligne de code vous devez les séparer par un point virgule.
La déclaration d'un package est facultative.

1.4 - Premier code

https://kotlinlang.org/docs/reference/basic-syntax.html

Avec IntelliJ IDEA Community de JetBrains.

Kotlin est sensible à la casse.

```
println ("string")
```

Un script .kts (kotlin script dans un projet)

hello.kts

```
//package un
println ("hello")
```

Commentaires :

// Commentaire de ligne

/*
Commentaires de bloc
*/

Note : package est facultatif.

1.5 - Les types

https://kotlinlang.org/docs/reference/basic-types.html

Char	'a'
String	"Index"
Byte	8 bits
Short	16 bits
Int	32 bits
Long	64 bits
Float	32 bits
Double	64 bits
Boolean	
Array	

1.6 - Déclarations des variables, ...

Variable immuable (constante) : val

```
val variable: Type = valeur
```

Variable mutable et globale au script : var

```
var variable = valeur
```

```
// var.kt
fun main(args: Array<String>) {
     val a: Int = 1 // immediate assignment
     val b = 2 // `Int` type is inferred
     val c: Int // Type required when no initializer is provided
     c = 3 // deferred assignment
     println("a : " + a + ", b : " + b + ", c : " + c)

     // a++ // IMPOSSIBLE car c'est une constante
     //println("a : " + a + ", b : " + b)

     var x = 5 // `Int` type is inferred
     x += 1
     println("x : " + x)

     val PI = 3.14
     var y = 0

     // Fonction dans une fonction !!!
     fun incrementX() {
           y += 1
     }

     println("PI : " + PI)
     incrementX()
     println("y : " + y)
     println("y : ${y}")
}
```

L'interpolation :

println("y : ${y}")

1.7 - Première fonction : syntaxe classique

fun monDeFonction([paramètre : Type[, paramètre : Type]]) : Type

Function having two Int parameters with Int return type :

```
/*
fun1.kts
 */
package un

fun sum(a: Int, b: Int): Int {
    return a + b
}

print("sum of 3 and 5 is ")
println(sum(3, 5))
```

ou

```
//fun1.kt
fun sum(a: Int, b: Int): Int {
    return a + b
}

fun main(args: Array<String>) {
    print("3 + 5 = " + sum(3, 5))
}
```

Affichage :

sum of 3 and 5 is 8

ou

3 + 5 = 8

1.8 - Deuxième fonction : inférence !

Function with an expression body and inferred return type (Fonction avec un corps d'expression et un type de retour inféré) :

```
fun nomDeFonction([paramètre : Type[ , paramètre : Type]]) = expression
```

```
// fun2.kts

fun sum(a: Int, b: Int) = a + b

fun main() {
    println("sum of 3 and 5 is ${sum(3, 5)}")
}

main()
```

ou

```
// fun2.kt
fun sum(a: Int, b: Int) = a + b

fun main(args: Array<String>) {
    println("3 + 5 = ${sum(3, 5)}")
}
```

Notes :
les paramètres sont obligatoirement typés.
notez l' « interpolation » ou String template avec ${} à l'intérieur de double-quote droite.

Comparaison avec la précédente :

« Standard »	« Inférrente »

`fun sum(a: Int, b: Int): Int {`
`  return a + b`
`}`
	`fun sum(a: Int, b: Int) = a + b`

1.9 - Procédure

Function returning no meaningful value

```
fun nomDeProcedure([paramètre : Type[ , paramètre : Type]]): Unit {
```

```
// proc.kts

fun printSum(a: Int, b: Int): Unit {
    println("sum of $a and $b is ${a + b}")
}

printSum(3, 5)
```

ou

```
// proc.kt
fun printSum(a: Int, b: Int): Unit {
    println("$a + $b = ${a + b}")
}

fun main(args: Array<String>) {
    println(printSum(3, 5))
}
```

Note : le type du retour, Unit, est facultatif.

1.10 - Les structures de la programmation

1.10.1 - Conditionnelles

1.10.1.1 - if

```
if (condition) {
   // ...
} else {
   // ...
}
```

```
// if.kt
fun minimum(a: Int, b: Int): String {
    if (a < b) {
        return "a < b"
    } else {
        return "a > b"
    }
}

fun main(args: Array<String>) {
    val string = minimum(3, 5)
    println(string)
}
```

if with range

```
if (x in debut..fin) {
```

```
// if_range.kt
fun main(args: Array<String>) {
     val x = 2
     if (x in 1..5) {
           print("x is in range from 1 to 5")
     }
     println()

     if (x !in 6..10) {
           print("x is not in range from 6 to 10")
     }
}
```

1.10.1.2 - when

```
when (variable) {
   "valeur" -> action
   //
}
```

```
// switch.kts
// Avec when
fun auCasOu(periode: String): String {
    var traduction: String = ""
    when (periode) {
        "matin" -> traduction = "morning"
        "midi" -> traduction = "midday"
        "après-midi" -> traduction = "afternoon"
        "soir" -> traduction = "evening"
        "nuit" -> traduction = "night"
    }
    return traduction
}

val traduction = auCasOu("midi")
println(traduction)
```

ou

```
// switch.kt
fun auCasOu(periode: String): String {
    var traduction: String = ""
    when (periode) {
        "matin" -> traduction = "morning"
        "midi" -> traduction = "midday"
        "après-midi" -> traduction = "afternoon"
        "soir" -> traduction = "evening"
        "nuit" -> traduction = "night"
    }
    return traduction
}

fun main(args: Array<String>) {
    val traduction = auCasOu("midi")
    println(traduction)
}
```

1.10.2 - Itératives

1.10.2.1 - for

« foreach »

No comments !

```
for (item in items) { … }
```

```
// foreach.kt
fun main(args: Array<String>) {
     // listOf est une liste statique
     val items = listOf("Java", "Kotlin", "Scala")
     for (item in items) {
           println(item)
     }
}
```

Affiche :

JavaScript
Kotlin
Scala

« fori »

No comments !

for (x in début..fin [step pas]) { … }

for (x in début downto fin [step pas]) { … }

```
// fori.kt
fun main(args: Array<String>) {
     for (x in 1..5) {
           print(x)
     }
     println()

     for (x in 1..10 step 2) {
           print(x)
     }
     println()

     for (x in 9 downTo 0 step 3) {
           print(x)
     }
}
```

Affiche :

12345
13579
9630

1.10.2.2 - forEach

```
collection.forEach { expression}
```

```
items.forEach { item -> println(item) }
```

```
// foreach.kt
fun main(args: Array<String>) {
     val items = listOf("Java", "Kotlin", "Scala", "JavaScript")
     items.forEach { item -> println(item) }
}
```

1.10.2.3 - while

No comments !

```
var index = 0
while (index < items.size) {
   // Code
   index++
}
```

```
// while.kt
fun main(args: Array<String>) {
     val items = listOf("Java", "Kotlin", "Scala")
     var index = 0
     while (index < items.size) {
           println("item at $index is ${items[index]}")
           index++
     }
}
```

Affiche :

item at 0 is Java
item at 1 is Kotlin
item at 2 is Scala

1.11 - Principaux opérateurs

Comparaison

Opérateur	Description
==	Égalité
===	Identité
!=	Différent de
>	Supérieur
>=	Supérieur ou égal
<	Inférieur
<=	Inférieur ou égal

Logiques

Opérateur	Description
&&	Et
||	Ou
!	Non

1.12 - Les collections

https://kotlinlang.org/docs/reference/collections-overview.html

Collection<T> is the root of the collection hierarchy. This interface represents the common behavior of a read-only collection: retrieving size, checking item membership, and so on. Collection inherits from the Iterable<T> interface that defines the operations for iterating elements. You can use Collection as a parameter of a function that applies to different collection types. For more specific cases, use the Collection's inheritors: List and Set.

List is an ordered collection with access to elements by indices – integer numbers that reflect their position. Elements can occur more than once in a list.

Set is a collection of unique elements.

Map (or dictionary) is a set of key-value pairs. Keys are unique, and each of them maps to exactly one value. The values can be duplicates.

The Kotlin Standard Library provides implementations for basic collection types: sets, lists, and maps. A pair of interfaces represent each collection type:

A read-only interface that provides operations for accessing collection elements.
A mutable interface that extends the corresponding read-only interface with write operations: adding, removing, and updating its elements.

1.12.1 - Array

Tableau statique.

```
arrayOf(value1, value2, ...)
```

```
// array.kt
fun main(args: Array<String>) {
     val items = arrayOf("Java", "Kotlin", "Scala")
     for (item in items) {
           println(item)
     }

     println(items.size)
     println(items[0])
     println(items.get(0))

     println(items.count())

     for (x in 0..items.size - 1) {
           println(items[x])
     }
}
```

1.12.2 - List

Déjà vu ! Liste statique.

listOf(value1, value2, ...)

```
// list.kt
fun main(args: Array<String>) {
     val items = listOf("Java", "Kotlin", "Scala")
     for (item in items) {
           println(item)
     }

     println(items.size)
     println(items[0])
     println(items.get(0))

     items.add("PHP") // list not mutable → error
}
```

Note :

Une « List » de type listOf() est immuable.

1.12.3 - MutableList

Liste dynamique.

```
mutableListOf(value1, value2, ...)
```

```
// listmutable.kt
fun main(args: Array<String>) {

    val nombres = mutableListOf("un", "deux", "trois", "quatre")

    println("Nombre d'éléments : ${nombres.size}")
    println("1e : ${nombres.get(0)}")
    println("4e : ${nombres[3]}")
    println("Index \"deux\" ${nombres.indexOf("deux")}")

    nombres.add("cinq")
    println("Nombre d'éléments : ${nombres.size}")

    nombres.remove("cinq")
    println("Nombre d'éléments : ${nombres.size}")

    nombres.removeAt(0)
    println("Nombre d'éléments : ${nombres.size}")
}
```

Affiche :

Nombre d'éléments : 4
1e : un
4e : quatre
Index "deux" 1
Nombre d'éléments : 5
Nombre d'éléments : 4
Nombre d'éléments : 3

1.12.4 - Set

Liste statique à valeurs uniques.

setOf(value1, value2, ...)

```
// set.kt
fun main(args: Array<String>) {
     // Le dernier ne sera pas pris en compte
     val items = setOf("Java", "Kotlin", "Scala", "Scala")
     for (item in items) {
           println(item)
     }

     println(items.size)
     //println(items[0]) // KO
     //println(items.get(0)) // KO
}
```

1.12.5 - MutableSet

Liste dynamique à valeurs uniques.

mutableSetOf(value1, value2, ...)

```
// setmutable.kt
fun main(args: Array<String>) {
     // Le dernier ne sera pas pris en compte
     val items = mutableSetOf("Java", "Kotlin", "Scala", "Scala")
     for (item in items) {
           println(item)
     }

     println(items.size)

     items.add("Groovy")
     println(items.size)

     items.remove("Groovy")
     println(items.size)
}
```

1.12.6 - Map

Map = table de hachage, dictionnaire, ...
Ensemble de paires clé/valeur.
Les clés sont uniques.
Une clé renvoie une seule valeur.
Les map sont en lecture seule.

mapOf(key1 to value1, key2 to value2, ...)

```
// map.kt
fun main(args: Array<String>) {

	val ascii = mapOf(97 to "a", 98 to "b", 120 to "x")
	println(ascii)

	val user = mapOf("user" to "Pascal", "pwd" to "Mdp123")
	println(user)

	for ((k, v) in user) {
		println("$k = $v")
	}

	println(user["user"])

	println(user.count())

	println(user.size)

	//user["pwd"] = "mdp567" // KO : no set method providing array access
}
```

Affiche :

```
{97=a, 98=b, 120=x}
{user=Pascal, pwd=Mdp123}
user = Pascal
pwd = Mdp123
Pascal
2
2
```

1.12.7 - HashMap

Map = table de hachage, dictionnaire, …, dynamique.

```
map = HashMap<Type, Type>()
```

```
// map.kt
fun main(args: Array<String>) {

     val items = HashMap<String, Int>()

     items["matin"] = 5
     items["midi"] = 13
     items["soir"] = 9

     for ((k, v) in items) {
           println("$k = $v")
     }

     println(items["midi"])

     println(items.size)

     items["nuit"] = 7
     println(items["nuit"])

     println(items.size)

     items["midi"] = 12
     println(items["midi"])

     items.remove("nuit")
     println(items.size)
}
```

Affiche :

soir = 9
midi = 13
matin = 5
13
3
7
4
12
3

1.12.8 - MutableMap

mutableMapOf(key1 to value1, key2 to value2, ...)

```
// mapmutable.kt
fun main(args: Array<String>) {

     val items = mutableMapOf("Paris" to 75, "Lyon" to 69, "Marseille" to 13)

     for ((k, v) in items) {
           println("$k = $v")
     }

     println("Lyon : " + items["Lyon"])

     println("Size : " + items.size)

     items["Lille"] = 59
     println("Lille : " + items["Lille"])

     items.remove("Lille")
     println("Size : " + items.size)
     println("Lille : " + items["Lille"])
}
```

1.12.9 - MutableList of MutableMap

1.13 - Bibliothèque standard

1.13.1 - let

https://play.kotlinlang.org/byExample/05_stdlib/01_letFunction

1.13.2 - with

1.13.3 - any, all, none

Renvoie un booléen si la condition est satisfaite. Dans le cas de any il faut qu'au moins une valeur corresponde à la condition, dans le cas de none aucune et dans le cas de all toutes.

```
List.any { expression }
```

```
// any_all_none.kt
fun main(args: Array<String>) {
     val isEven: (Int) -> Boolean = { it % 2 == 0 }
     val zeroToTen = 0..10
     val evens = listOf(2,4,6,8)
     val odds = listOf(1,3,5,7,9)
     println(zeroToTen.any { isEven(it) } )
     println(zeroToTen.all { isEven(it) } )
     println(zeroToTen.none { isEven(it) } )

     println(evens.all { isEven(it) } )
     println(odds.none { isEven(it) } )
}
```

Affiche :

true
false
false
true
true

1.13.4 - find, first, last, ...

first() et last() sont des méthodes sans argument.
find() est une méthode qui accepte un prédicat.
String? :
Impossible dans les usages de find() de typer en String ou de ne pas typer (le type serait inféré).

```
// find.kt
fun main(args: Array<String>) {
     val items = listOf("Java", "Kotlin", "JavaScript", "Scala")
     var element : String?
     element = items.first()
     println(element)
     element = items.last()
     println(element)
     element = items.find { it.startsWith("Java") }
     println(element)
     element = items.findLast { it.startsWith("Java") }
     println(element)
}
```

Affiche :

Java
Scala
Java
JavaScript

1.13.5 - count

https://play.kotlinlang.org/byExample/05_stdlib/08_count

The extension function count returns either the total number of elements in collection or the number of elements matching the given predicate.

```
Int = collection.count()
```


```
Int = collection.count()
```


```
// count.kt
fun main(args: Array<String>) {
     val items = arrayOf("Java", "Kotlin", "Scala")
     print(items.count())
}
```

Note :
alias de size !

Avec un prédicat :

```
// count.kt
fun main(args: Array<String>) {
     val items = listOf("Java", "Kotlin", "Scala")
     println(items.count())
     var count = items.filter { it.startsWith('J') }.size
     println(count)
     // but better to not filter, but count with a predicate
     count = items.count { it.startsWith('J') }
     println(count)
}
```

1.13.6 - filter

Cf précédent.

1.13.7 - partition

Partitionne une liste en fonction d'une expression.

```
val (liste1, liste2) = liste.partition { … }
```

```
// partition.kt
fun main(args: Array<String>) {
     val numbers = listOf(1, 3, -4, 2, -11)
     val (positive, negative) = numbers.partition { it > 0 }
     println(positive)
     println(negative)
}
```

Affiche :

[1, 3, 2]
[-4, -11]

1.13.8 - associateBy, groupBy

associateBy

```
```

```
//
```

Affiche :

groupBy

Regroupe en fonction d'une expression (un prédicat).

```
List.groupBy { expression }
```

```
// group_by.kt
fun main(args: Array<String>) {
     val words = listOf("b", "bon", "bo", "jour", "soir")
     val byLength = words.groupBy { it.length }

     println("Keys : " + byLength.keys)
     println("Values : " + byLength.values)
}
```

Affiche :

Keys : [1, 3, 2, 4]
Values : [[b], [bon], [bo], [jour, soir]]

1.13.9 - zip

Renvoie une liste de valeurs qui est le résultat de la concaténation de tous les éléments de chaque liste. La taille de la liste correspond à celle de la plus petite liste.

```
List1.zip(List2) { … }
```

```
// zip.kt
fun main(args: Array<String>) {
     val items1 = listOf("Java", "Kotlin", "Scala", "JavaScript")
     val items2 = listOf(1, 2, 3, 4, 5)
     val z = items1.zip(items2) { a, b -> "$a$b" }
     println(z)
}
```

Affiche :

[Java1, Kotlin2, Scala3, JavaScript4]

1.13.10 - flatMap

Returns a single list of all elements yielded from results of transform function being invoked on each element of original array.

```
List.flatMap { … }
```

```
// flat_map.kt
fun main(args: Array<String>) {
     val list = listOf("123", "45")
     println(list.flatMap { it.toList() })
}
```

Affiche :

[1, 2, 3, 4, 5]

1.13.11 - min, max

Retourne la valeur min ou max d'un couple de valeurs.

```
// min_max.kt
fun main(args: Array<String>) {
     val mini = kotlin.math.min(3,2)
     val maxi = kotlin.math.max(3,2)
     println(mini)
     println(maxi)
}
```

1.13.12 - sorted

Trier une collection (arrayOf, listOf, setOf).

```
// tri.kt
fun main(args: Array<String>) {
     val numbers = listOf("un", "deux", "trois", "quatre", "cinq")

     println("Tri ascendant : ${numbers.sorted()}")
     println("Tri descendant : ${numbers.sortedDescending()}")
}
```

Affiche :

Tri ascendant : [cinq, deux, quatre, trois, un]
Tri descendant : [un, trois, quatre, deux, cinq]

1.13.13 - getOrElse

Renvoie la valeur correspondant à une clé ou une valeur par défaut.

```
Map.getOrElse("clé") { valeur par défaut }
```

```
// get_or_else.kt
fun main(args: Array<String>) {
     val map = mutableMapOf<String, Int?>()
     map["x"] = 10
     map["y"] = 20
     map["z"] = 30

     println(map.getOrElse("x") { 1 })
     println(map.getOrElse("y") { 1 })
     println(map.getOrElse("a") { 1 })
}
```

Affiche :

```
10
20
1
```