



Kotlin

Programmation Fonctionnelle

 Fonction Infix

 Fonction Anonyme

 Fonction à Retour Local

 Fonction Récursive Tail Vs Classique

Classe, Propriété, Fonction

Fonction Infix

- Principe : Pas de parenthèse.

```
map(  
  1 to "one",  
  2 to "two",  
  3 to "three"  
)
```

i Pair<A, B>

Classe, Propriété, Fonction

Fonction Infix

true or false

i Boolean : and() xor()

Classe, Propriété, Fonction

Fonction Infix

```
class Assertion<T>(private val target: T) {  
    infix fun isEqualTo(other: T) {  
        Assert.assertEquals(other, target)  
    }  
  
    infix fun isDifferentFrom(other: T) {  
        Assert.assertNotEquals(other, target)  
    }  
}  
  
val result = Assertion(5)  
  
result isEqualTo 5 // This passes  
result isEqualTo 6 // This fails the assertion  
result isDifferentFrom 5 // This also fails the assertion
```

Classe, Propriété, Fonction

Fonction Anonyme

```
// anonymous function with body as an expression  
val anonymous1 = fun(x: Int, y: Int): Int = x + y
```

```
// anonymous function with body as a block  
val anonymous2 = fun(a: Int, b: Int): Int {  
    val mul = a * b  
    return mul  
}
```

```
val sum = anonymous1(3,5)  
val mul = anonymous2(3,5)
```

Classe, Propriété, Fonction

Fonction à Retour Local

```
fun localReturn(list: List<String>) {  
    var counter = 1  
    run {  
        list.forEach {  
            println(it)  
            if (counter == 3) {  
                println("counter == 3")  
                return@run  
            }  
            counter++  
        }  
    }  
    println("End of loop")  
}
```

Return and Local Return

Référence [x] : <https://kotlinlang.org/docs/returns.html>

Référence [x] : [kotlinlang.org: local-return-clarification](https://kotlinlang.org/docs/local-return-clarification.html)

Classe, Propriété, Fonction

Fonction Récursive Tail Vs Classique

Récursivité classique

1. appel récursif
2. retour de la valeur recursive avec calcul du résultat

 `StackOverflowError`

Récursivité tail

1. calcul
2. appel récursif, passant le résultat de l'étape actuel au prochain appel récursif

 `Stack memory`

Programmation Fonctionnelle

📎 Fonction d'extension

📎 Extension de propriété

📎 Expression *Lambda*

📎 Fonctions Inline utiles

📎 *Operator Overloading*

📎 *Closure*

Programmation Fonctionnelle

Fonction d'Extension

- Définition : fonction que l'on définit pour une classe, en dehors de cette classe, cela dit elle va pouvoir être appelée comme étant une méthode de cette classe.
- Mécanisme : possibilité d'étendre une classe existante avec d'autres méthodes.
- Principe : les fonctions d'extension sont résolues à l'exécution.

Programmation Fonctionnelle

Fonction d'Extension : exemple

```
fun IntRange.random() = Random().nextInt((endInclusive + 1) - start) + start
```

Programmation Fonctionnelle

Fonctions d'Extension utiles

```
s.max()  
s.filter { it.equals("cool") }  
s.count()
```

 Pour les collections

Programmation Fonctionnelle

Extension de propriété

```
StringBuilder.firstLetter: Char  
  get() = get(0)  
  set(value) = this.setCharAt(0, value)
```

 Récupérer / modifier
stocker

Programmation Fonctionnelle

Expression *Lambda* : exemple

```
btn.setOnClickListener { toast("hello") }
```

Programmation Fonctionnelle

Expression *Lambda* : signature de `setOnClickListener`

```
fun setOnClickListener(listener: (view: View) -> Unit){}
```

Programmation Fonctionnelle


Expression *Lambda* Vs Classe anonyme

```
btn.setOnClickListener(object : OnClickListener {  
    override fun onClick(v: View) {  
        toast("hello")  
    }  
})
```


Programmation Fonctionnelle

Expression *Lambda* : Règles de syntaxe

Paramètre(s) Corps



```
{ v: View -> toast("hello") }
```

i Toujours dans
des accolades

Programmation Fonctionnelle

Expression *Lambda* : Règles de syntaxe

```
btn.setOnClickListener({ v: View -> toast("hello") })
```

i Syntaxe complète

Programmation Fonctionnelle

Expression *Lambda* : Règles de syntaxe

```
btn.setOnClickListener({ v -> toast("hello") })
```

i Type **View** évident
d'après le contexte

Programmation Fonctionnelle

Expression *Lambda* : Règles de syntaxe

```
btn.setOnClickListener() { v -> toast("hello") }
```

i Lambda est le dernier argument

Programmation Fonctionnelle

Expression *Lambda* : Règles de syntaxe

```
btn.setOnClickListener { v -> toast("hello") }
```

i Parenthèses vides
peuvent être enlevé

Programmation Fonctionnelle

Expression *Lambda* : Règles de syntaxe

```
btn.setOnClickListener { toast("hello") }
```

i Un seul paramètre = **it**

Programmation Fonctionnelle

Fonction `inline` : `with`

```
with(magicCircle) {  
    canvas?.drawCircle(cx, cy, rad, paint)  
    move(dX, dY)  
}
```

i `this` est un receveur
implicite de lambda

Programmation Fonctionnelle

Fonctions `inline` utiles

`with`

`run`

`let`

`also`

`apply`

i nuance `in/out`

Programmation Fonctionnelle

Fonction `inline` : `run`

```
webView.settings?.run {  
    javaScriptEnabled = true  
    databaseEnabled = true  
}
```

i idem `with` en fonction d'extension

Programmation Fonctionnelle

Fonction `inline` : `let`

```
stringVariable?.let {  
    println("The length of this String is ${it.length}")  
}
```

i idem `run`, `it` est un
receveur implicite
de lambda

Programmation Fonctionnelle

Fonction `inline` : `also`

```
fun makeDir(path: String) = path.let{ File(it) }.also{ it.mkdirs() }
```

i idem `let`,
retourne `it`

Programmation Fonctionnelle

Fonction `inline` : `apply`

```
fun createInstance(args: Bundle) = MyFragment().apply { arguments = args }
```

i idem `run`,
retourne `this`

Programmation Fonctionnelle

Fonctions `inline`

	<code>{..this..}</code>	<code>{..it..}</code>
<code>-> {..}</code>	<code>run/with</code>	<code>let</code>
<code>-> this/it</code>	<code>apply</code>	<code>also</code>

Programmation Fonctionnelle

Operator Overloading

- Principe : fournir des implémentations différentes aux opérateurs existants

Programmation Fonctionnelle

Operator Overloading

```
interface IndexedContainer {  
    operator fun get(index: Int)  
}  
  
class OrdersList: IndexedContainer {  
    override fun get(index: Int) { /*...*/ }  
}
```

Programmation Fonctionnelle

Closure

```
var containsNegative = false

val ints = listOf(0, 1, 2, 3, 4, 5)
  ints.forEach {
    if (it < 0)
      containsNegative = true
  }
```

i Modifier une var
dans lambda

Programmation Fonctionnelle

📎 Fonction Infix

📎 Fonction Anonyme

📎 Fonction à Retour Local

📎 Fonction Récursive Tail Vs Classique

📎 Fonction et propriété d'extension

📎 *Lambda*

📎 *Operator Overloading*

📎 *Closure*

THE EXPERT
AT ANYTHING
WAS ONCE
A BEGINNER.

_ Helen Hayes

Références

Kotlin for Android

- [TRY Kotlin](#)
- [Kotlin Workshop on Github: Slides and Questions](#)
- <https://antonioleiva.com/free-kotlin-android-course/>
- [ChillCoding.com : Introduction à Kotlin](#)
- [ChillCoding.com : Configurer Kotlin dans un projet Android Studio](#)

Library

- [ChillCoding.com : Utiliser des bibliothèques graphiques Kotlin dans un projet Android](#)

Fonction d'extension

- [Odelia Technologies : Les fonctions d'extension de Kotlin](#)

Kotlin in videos

- [Jake Wharton and Kotlin \(DEC 2015\)](#)
- [Tue Dao & Christina Lee on The Road to Kotlin town \(KotlinConf 2017\)](#)
- [Introduction to Kotlin Google I/O '17](#)

Références

Why Kotlin?

- [Swift is like Kotlin](#)
- [Langage Java](#)
- [API Java : Google a enfreint les brevets d'Oracle, selon la Cour Suprême](#)
- [Antoniroleiva: 12 reasons to strat Kotlin for Android](#)

Kotlin in brief

- [Kotlin: pourquoi ce nouveau langage est une bonne nouvelle](#)

Android and Kotlin

- [Kotlin - Official Site](#)
- [developer.android: Get Started with Kotlin on Android](#)

Type Kotlin

- [https://code.tutsplus.com/tutorials/kotlin-from-scratch-variables-basic-types-arrays-type-inference-and-comments--cms-29328](#)
- [https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-array/index.html](#)
- [http://kotlinlang.org/docs/reference/basic-types.html#arrays](#)

-

try.kotlinlang.org

The screenshot shows a web browser window with the URL `try.kotlinlang.org`. The page features the Kotlin logo and navigation links for `LEARN`, `COMMUNITY`, and `TRY ONLINE`. Below the navigation bar, there are social media icons for Facebook, Google+, GitHub, and JetBrains. A toolbar contains buttons for `Shortcuts`, `Convert from Java`, and `Fullscreen`.

The main content area is titled `Kotlin Koans > Introduction > Hello, world! > Task.kt`. On the left, a sidebar lists the koans, with `Task.kt` selected. The main panel displays the task instructions for `Simple Functions`:

Take a look at [function syntax](#) and make the function `start` return the string `"OK"`.

In the tasks the function `TODO()` is used that throws an exception. Your job during the koans will be to replace this function invocation with a meaningful code according to the problem.

Below the instructions are buttons for `Check`, `Revert`, and `Show answer`. The code editor shows the following code:

```
1 fun start() = "OK"
```