



*Kotlin*

*per TD: 7937*

<b>1. Història</b>	<b>2</b>
<b>2. Propòsit i aplicacions</b>	<b>3</b>
<b>3. Paradigmes i Exemples</b>	<b>3</b>
<b>4. Sistema de tipus i gestió de NULL</b>	<b>8</b>
<b>5. Avantatges i Inconvenients</b>	<b>10</b>
<b>6. Comparacions</b>	<b>11</b>
6.1 Kotlin vs Java	12
6.2 Kotlin vs Python	12
<b>7. Opinió Personal</b>	<b>13</b>
<b>8. Bibliografia</b>	<b>14</b>
<b>9. Descripció de les fonts</b>	<b>16</b>
<b>10. Avaluació de les fonts</b>	<b>16</b>

# 1. Història

La història d'aquest llenguatge és molt breu, ja que gairebé acaba de néixer. Fou desenvolupat des del 2011 fins a 2016 per la famosa empresa *JetBrains*. Ara ja està disponible des d'agost d'aquest any la versió 1.7.20.

El nom Kotlin prové de l'illa Kotlin (situada a prop de sant Petersburg, Rússia), imitant burlescament a Java, ja que aquest també el seu nom té origen en una illa d'Indonèsia (Java).

L'any 2017, en la conferència Google I/O, Google va anunciar que donaria suport complet de Kotlin amb Android, cosa que va despertar l'interès de molts desenvolupadors de programari.

## 2. Propòsit i aplicacions

Kotlin va estar concebut exactament per tractar de substituir Java, que és molt usat avui en dia en la indústria del software, per una alternativa millorada i ensucrada sintàcticament per tal de simplificar i millorar aquell llenguatge útil però antiquat. Llavors van decidir que per tal d'accelerar la transició de Java a Kotlin es respectés una màxima interoperabilitat entre llenguatges, és a dir, tot codi en Kotlin es pot convertir en Java i viceversa. Totes les llibreries de Java són disponibles a Kotlin.

Un dels seus punts forts és que és multiplataforma, ja que es pot executar en qualsevol màquina virtual de Java (JVM) o qualsevol dispositiu amb intèrpret de JavaScript.

S'usa principalment per desenvolupament de software (tant front-end amb l'extensió K-JavaScript com back-end possible gràcies a les crides asíncrones), aplicacions i també, per la seva utilitat, simplicitat i quantitat de llibreries útils, en ciències de dades.

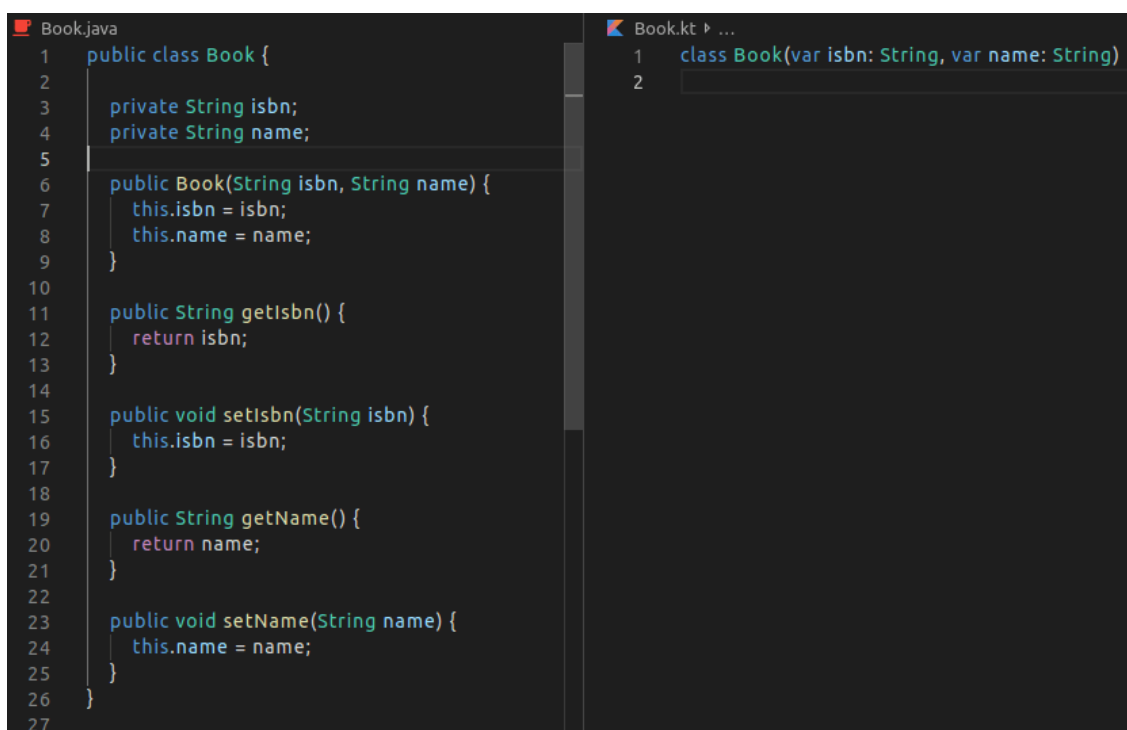
## 3. Paradigmes i Exemples

Tal com Java, es focalitza principalment en el Paradigma de programació orientada a objectes (OOP). Inclou millores notòries pel que fa a les classes,

perquè s'ha d'escriure molt menys. Tots els àmbits de visibilitat també hi són a Kotlin (S'afegeix l'àmbit Internal, que és com públic per tots els fitxers d'un mateix mòdul Kotlin, o sigui, un mateix conjunt de fitxers que es compilen de forma conjunta, un projecte), però per defecte les funcions, atributs i classes seran públics. En aquest llenguatge diferenciem d'Atributs i Propietats (És preferible usar propietats): Una propietat és un atribut + el seu getter (+Setter si és mutable (Var), o no si és immutable (Val)) . Aquests setters i getters són implícits i només cal usar l'operador punt com amb els Structs de C per invocar-los. En l'exemple de la figura 1, per obtenir ISBN caldria simplement fer “X.ISBN” on X és una instància de la classe llibre.

Si realment volem un atribut privat caldria fer servir el keyword “private” i llavors si haurem de crear funcions per accedir.

No només tenim classes, sinó que també s'afegeixen les Data Classes, que són Classes pensades per ser contenidors d'informació. Són molt útils, ja que per defecte creen els mètodes equals(), hashCode(), toString(), copy()... Que Java no fa automàticament.

La imatge mostra una comparació de codi entre Java i Kotlin. A l'esquerra, el codi Java (Book.java) defineix una classe pública Book amb dos atributs privats: isbn i name. Té un constructor públic que inicialitza aquests atributs, i mètodes getters i setters públics per a cada atribut. A la dreta, el codi Kotlin (Book.kt) defineix una classe Book amb dos paràmetres de var: isbn i name. Aquesta és una Data Class, ja que el nom de la classe està envoltat de parèntesis i els paràmetres de var són utilitzats directament com atributs dins de la classe sense necessitat de getters i setters explícits.

```
Book.java
1 public class Book {
2
3     private String isbn;
4     private String name;
5
6     public Book(String isbn, String name) {
7         this.isbn = isbn;
8         this.name = name;
9     }
10
11     public String getIsbn() {
12         return isbn;
13     }
14
15     public void setIsbn(String isbn) {
16         this.isbn = isbn;
17     }
18
19     public String getName() {
20         return name;
21     }
22
23     public void setName(String name) {
24         this.name = name;
25     }
26 }
27

Book.kt
1 class Book(var isbn: String, var name: String)
2
```

Figura 1: Comparació equivalent d'una classe qualsevol en Java i Kotlin  
La creadora ve implícita en el parèntesi. Els mateixos VAR's de la creadora seran atributs públics, si es vol que no ho siguin cal treure VAR.

Brinda noves funcionalitats de les quals no disposa el seu antecessor (funcions d'extensió, asincronia, control d'errors...).

Aquestes funcions d'extensió permeten afegir funcionalitats noves a classes que no podríem modificar i hauríem de crear una nova classe que l'expandeixi (Per exemple si volem afegir Fibonacci o potenciació en els INT, hauríem de crear un INT2 extends INT per poder-ho fer en Java). Un altre cop, guanyem simplicitat gràcies a no crear documents nous redundants.

```
//Estil funcional , descomposició per casos com a haskell
fun Int.factorial(): Int {
    return when {
        this <= 1 -> this
        else -> this * factorial(this-1)
    }
}

//Estil imperatiu
fun Int.potencia(n: Int): Int {
    if (n < 0) throw Exception("red flag amiga")
    else if(n == 0) return this
    else{
        if(n%2 == 0){
            var r = potencia(this,n/2)
            return r * r
        }
        else{
            var r = potencia(this, (n-1)/2)
            return this * r * r
        }
    }
}

var x = 2.potencia(30)
```

Figura 2: Exemple de funcions d'extensió. Afegeix a la Classe Int, la qual no tenim accés per modificar, 2 operacions.

L'asincronia permet facilitar que no es quedin penjades les aplicacions sense necessitat d'usar costosos threads. Les Corutines que porten el keyword "suspend" tenen la capacitat de suspendre's temporalment i després reprendre la seva execució. Són semblants callbacks però fent servir autòmats finits deterministes (DFA's). De fet, el compilador igual que et generà els getters i funcions de classes automàticament, generà els callback's (Recordatori: aquest mecanisme d'asincronia tracta de passar les subrutines com a paràmetres per poder executar-les quan la principal hagi finalitzat).

```

suspend fun loginUser(userId: String, password: String): User {
    val user = userRemoteDataSource.logUserIn(userId, password)
    val userDb = userLocalDataSource.logUserIn(user)
    return userDb
}

// UserRemoteDataSource.kt
suspend fun logUserIn(userId: String, password: String): User

// UserLocalDataSource.kt
suspend fun logUserIn(userId: String): UserDb

```

Figura 3: Exemple d'una corutina asíncrona amb corutines asíncrones

```

fun loginUser(userId: String, password: String, completion:
Continuation<Any?>) {
    when(label) {
        0 -> { // Label 0 -> first execution
            userRemoteDataSource.logUserIn(userId, password)
        }
        1 -> { // Label 1 -> resumes from userRemoteDataSource
            userLocalDataSource.logUserIn(user)
        }
        2 -> { // Label 2 -> resumes from userLocalDataSource
            completion.resume(userDb)
        }
        else -> throw IllegalStateException(...)
    }
}

```

Figura 4: Codi generat pel compilador usant un automàt finit determinista per establir l'ordre de l'execució de funcions amb Call Back.

Però realment és multiparadigma, oferint funcionalitats imperatives i funcionals (Funcions lambda, Funcions d'ordre superior, Guardes, Descomposició d'Estructures de Dades...).

```

fun main(args: Array<String>) {
    listOf(1, 2, 3); mutableListOf("a", "b", "c")

    setOf(1, 2, 3); mutableSetOf("a", "b", "c")

    mapOf(1 to "a", 2 to "b", 3 to "c")
    mutableMapOf("a" to 1, "b" to 2, "c" to 3)

    val aList = listOf(1, 2, 4)

    println(aList.map { elem ->
        elem + 1
    })

    println(aList)

    println(aList.filter { it != 1 })

    fun sum(a: Int, b: Int) = a + b
    println(aList.reduce(::sum))
}

```

Figura 5: Exemple de codi amb funcions d'ordre superior com filter, reduce o map.

```

val map = mapOf(1 to "one", 2 to "two")
for ((k, v) in map) { // Traverse a map or a list of pairs
    println("$k -> $v")
}

fun obtainKnowledge() = Pair("The Answer", 42) // Single-expression functions

val (description, answer) = obtainKnowledge() // Destructure into a pair of two variables
println("$description: $answer")

```

Figura 6: Codi d'exemple de manipulació de maps i pairs. Igual que altres llenguatges funcionals conté destructuració per patrons.

```

val x = //value
val xResult = when (x) {
    0, 11 -> "0 or 11"
    in 1..10 -> "from 1 to 10"
    !in 12..14 -> "not true from 12 to 14"
    else -> if (isOdd(x)) { "is odd" } else { "otherwise" }
}

val y = //value
val yResult = when {
    isNegative(y) -> "is negative"
    isZero(y) -> "is zero"
    isOdd(y) -> "is odd"
    else -> "otherwise"
}

```

Figura 7: Ús de guardes (When) en comptes de condicionals if-elseif-else

## 4.Sistema de tipus i gestió de NULL

Les variables en Kotlin es declaren amb VAL (valor només de lectura, similar al const de C) i amb VAR (variable de lectura i també escriptura). Si es vol inferir tipus per assegurar-se, es pot fer una declaració com `var x :: Int`. A pesar que sembla que el tipatge és dinàmic és purament estàtic, se saben en temps de compilació i no poden canviar. A més a més, el tipatge és fort, ja que no es permeten fer operacions entre tipus diferents sense fer casting o smart-casting.

Per saber els tipus evidentment Kotlin compta amb inferència de tipus. El punt fort en la meua opinió d'aquest sistema de tipus és en la poquíssima quantitat de tipus de dades bàsiques que hi ha, en comparació amb altres llenguatges. Hi ha els tipus numèrics INT i LONG, STRING, BOOLEAN, UNIT (void), classes pròpies i els tipus especials ANY i NOTHING. ANY és la superclasse abstracta de tots els tipus a Kotlin i Nothing és la subclasse inferior que no pot tenir instàncies i té un ús idèntic al d'altres llenguatges com Haskell.

També cal mencionar que he parlat dels tipus no anul·lables, o sigui, que no poden contenir valor NULL. Aquests són subclasses del seu tipus anul·lable, per exemple, `Int?` és extensió d'INT. Totes les classes amb interrogació són aquelles que poden contenir el valor NULL.

Evidentment, no només tenim accés a tipus primitius, sinó als que nosaltres creem com a Classes o Data Classes i a tots els de les llibreries de Java i Kotlin: HashSet, HashMap, ArrayList...



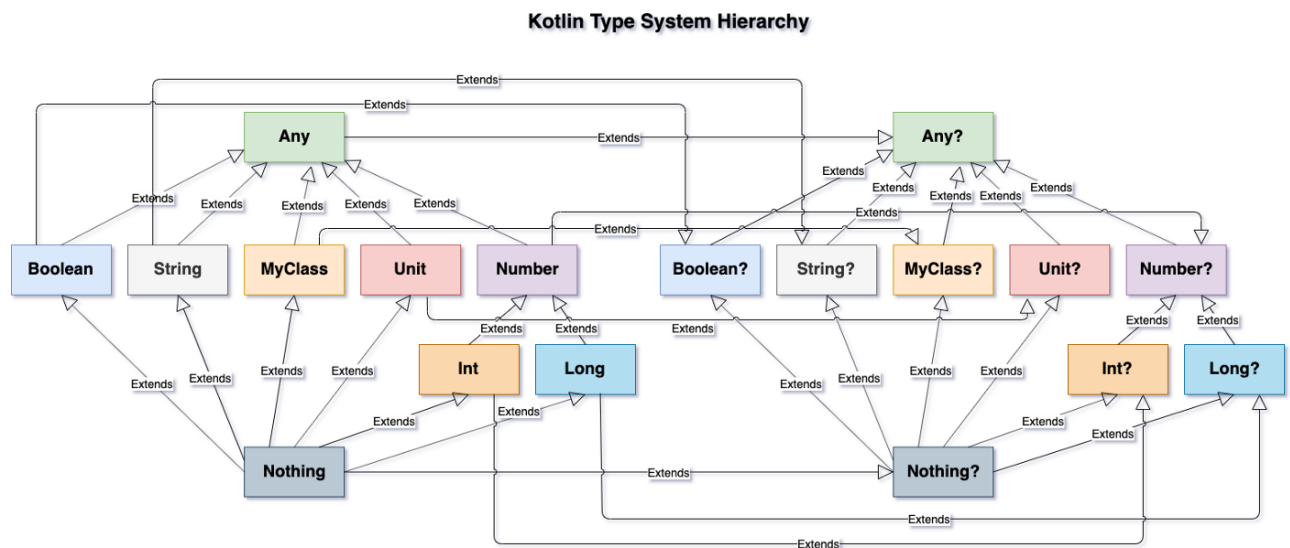


Figura 8: Jerarquia de tipus bàsics de Kotlin. La jerarquia està duplicada per permetre els tipus no anul·lables.

Un cop coneixem els tipus bàsics i les seves versions no anul·lables podem introduir els nous conceptes de seguretat de NULL que conté Kotlin. Per començar el simple fet de poder especificar si es tracta d'un `Int?` o un `Int` ja provocarà menys errors, ja que dona més control al programador. Però hi ha 3 funcionalitats més:

-Safe Null Assignment (`?.`): Aquest operador ens permetrà que les crides a funcions ens retornin valors nuls. Per exemple, si executéssim `graf.toString()` no es permet que retorni NULL perquè `toString()` torna `String`, no pas `String?`, llavors fent la crida d'aquesta manera: `graf?.toString()` es permetrà que es retornin valors nuls i en general que no rebenti el programa per culpa dels nuls. En resum, o bé retorna un valor del tipus que toca, o bé si rep com a paràmetre un nul, retorna nul sense executar la funció de manera instantània. Això és útil perquè el programador tingui més control del flux d'execució seu codi.

-Redefine Value if Null (`?:`): També conegut com l'operador Elvis, serveix per, en cas que el contingut d'una variable o return-value sigui nul, se li redefineix un nou valor. Per exemple en els Enters molts de cops associem el nul amb 0 o amb -1.

```
val l = b?.length ?: -1  
val l: Int = if (b != null) b.length else -1
```

Les dues expressions anteriors són equivalents, si la llargada de b és nul·la, en comptes de propagar el nul, que és el que faria l'operador (?.) se li donaria el valor -1.

-Not Null Assertion(!!): Es tracta simplement d'assertions per provocar i indicar on haurien d'haver-hi els Null Pointer Exception, potser sembla que no és útil, però hem de tenir en compte que és necessari per complementar les funcionalitats anteriors. Un codi ben fet en Kotlin hauria de tenir poc marge a NP Exception i ho hauria de tenir escrit textualment en el codi.

```
val l = b!!.length
```

L'expressió anterior fa que la crida a length retorni o bé el valor desitjat o bé llenci excepció NPE. Funciona exactament igual que l'operador safe-null, però en comptes de retornar nul llença una excepció, donant-nos la flexibilitat de substituir moltes NPE per (?.) i només fer-les servir quan siguin realment necessàries.

## 5. Avantatges i Inconvenients

Els avantatges que aporta Kotlin en enfront del seu predecessor són enormes, ja que podem considerar Java com un subconjunt reduït d'aquest. El codi és molt més senzill, llegible i compta amb les noves funcionalitats que ens ajudaran a reduir codi, com les funcions d'extensió (Ens evitaran crear classes i documents innecessaris) i les Data Classes. Ens facilitarà la vida com a programadors amb el sistema de nuls un cop s'hagi après a fer servir. Resulta fàcil d'aprendre si ja coneixes Java.

També és multiplataforma, interoperable amb el seu predecessor i compta amb diversos paradigmes de programació fent-lo més flexible

Per l'altra banda no es pot negar que costa aprendre determinats aspectes del llenguatge perquè encara la comunitat és relativament petita i la

documentació sincerament a vegades no ajuda, li falta contingut, cosa que he trobat frustrant personalment. La compilació és similar en velocitat a Java, en definitiva, lenta. A part que costa inicialment entendre com fer servir totes les funcionalitats del sistema de nuls segurs, jo no crec que sigui el millor llenguatge per principiants, ja que proporciona maneres d'estalviar codi que potser no entendrien, a part del sistema de nuls segurs.

Avantatges	Inconvenients
Codi minimalista	Comunitat petita i Documentació escassa
Fàcil d'aprendre (Si coneixes bé Java)	Compilació lenta
Multiplataforma	Difícil per nous programadors
Interoperable amb Java i retrocompatible	Costa inicialment acostumar-se al sistema de no anul·labilitat i els operadors...
Dona eines per evitar errors i NPE	
Flexibilitat de paradigmes de programació	
Noves funcionalitats (asincronia, funcions d'extensió, Data Classes)	

Figura 9: Taula comparativa d'avantatges i inconvenients resumida

## 6.Comparacions

En aquest apartat vull fer una petita discussió de com de millor o pitjor és respecte a altres llenguatges de programació que tenen rols i aplicacions similars, per exemple no faig comparacions amb C/C++, ja que són eines que competeixen en 2 àmbits diferents.

## 6.1 Kotlin vs Java

És trivial, per tot el que he redactat fins ara que Java serà inferior o semblant a poder que Kotlin, ja que tot el que està en Java està en l'altre de manera ensucrada o directa, igual que les llibreries. A més que Kotlin té moltes noves funcionalitats i sense afegir gens de càrrega a la compilació. Kotlin en molts aspectes és superior, però ambdós llenguatges són molt semblants.

## 6.2 Kotlin vs Python

En senzillesa i facilitat d'ús pels principiants seria millor Python, aquest és de tipatge dinàmic mentre que Kotlin estàtic. Els dos llenguatges tenen una gran quantitat de llibreries i són multiplataforma, però quant a seguretat del flux d'execució Kotlin és molt més segur gràcies al seu tipatge i al control de nuls. També Kotlin és millor quant a desenvolupament web i de software, ja que es pot implementar front-end amb aquest, sense necessitat d'altres eines i Python no pot pas. També Kotlin és més ràpid en compilar i executar que no pas el seu competidor.

## 7. Opinió Personal

En lo personal m'ha encantat el llenguatge, tant que en els meus projectes personals estic començant a fer-lo servir. És breu, concís i útil per desenvolupar software o aplicacions i tot apunta al fet que serà el rei indiscutible d'aquests àmbits en el futur, a part que es força útil en ciències de dades. Cada cop hi ha més gent que està fent la transició de Java a Kotlin.

M'agrada que no em restringeixi a un paradigma en concret com altres llenguatges i pugui fer codis més senzills que en C++, el meu llenguatge habitual.

Per l'altra banda m'ha frustrat força en el procés d'aprenentatge lo petita que és la comunitat actualment (No hi ha masses posts a StackOverFlow) i lo curta i escassa documentació oficial que hi ha, gràcies a deu que hi ha desenvolupadors de software que es dediquen a fer sèries en YouTube ensenyant el llenguatge, perquè aprendre amb la documentació oficial no ho recomano en absolut. També, com a tothom, m'ha costat acostumar-me al sistema de nuls segurs de Kotlin, però quan el saps fer servir es força útil (Igualment no s'ha d'ignorar la complexitat addicional que afegeix pel programador). I com deia a apartats anteriors jo no ho recomanaria a absoluts principiants com el seu primer llenguatge, ja que encara la comunitat no és lo força gran i pot arribar a ser molt frustrant i confús si et manquen conceptes essencials. Jo el recomanaria però com segon llenguatge.

En definitiva a mi m'agrada i el recomanaria, perquè és potent i pel fet que a mi em sembla que en el futur podria arribar a ser molt rellevant, però ara mateix que tot just ha sortit encara té aspectes a polir. Li poso un 8/10.

## 8. Bibliografía

-Salas, I. (s. f.). *Kotlin*. programandointentandolo.

<https://programandointentandolo.com/kotlin>

-González, R. (5d. C., marzo 21). *Qué es Kotlin?* Crehana.

<https://www.crehana.com/blog/transformacion-digital/que-es-kotlin/>

-colaboradores de Wikipedia. (2022, 2 agosto). *Kotlin (lenguaje de programación)*. Wikipedia, la enciclopedia libre.

[https://es.wikipedia.org/wiki/Kotlin\\_\(lenguaje\\_de\\_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/Kotlin_(lenguaje_de_programaci%C3%B3n))

-*Kotlin Programming Language*. (s. f.). Kotlin.

<https://kotlinlang.org/>

-Team, D. (2021, 13 agosto). *Kotlin vs Java: Which one's better for android app development in 2021?* Devathon.

<https://devathon.com/blog/kotlin-vs-java-android-app-development/>

-Akhin, M. (s. f.). *Kotlin language specification*.

<https://kotlinlang.org/spec/introduction.html>

Engineerhoon. (2020, 14 junio). *Tutorial #13 | How does Suspend function work internally? | Engineer*. YouTube.

<https://www.youtube.com/watch?v=4YyVxNLFaBQ>

*MoureDev by Brais Moure.* (s. f.). YouTube.

<https://www.youtube.com/@mouredev>

*Concurrency and Coroutines.* (2022, 22 septiembre).

<https://kotlinlang.org/docs/multiplatform-mobile-concurrency-and-coroutines.html>.

Wikipedia contributors. (2022, 11 noviembre). *Callback (computer programming)*. Wikipedia.

[https://en.wikipedia.org/wiki/Callback\\_\(computer\\_programming\)](https://en.wikipedia.org/wiki/Callback_(computer_programming))

Vivo, M. (2021, 13 diciembre). *The suspend modifier — under the hood - Android Developers*. Medium.

<https://medium.com/androiddevelopers/the-suspend-modifier-under-the-hood-b7ce46af624f>

## 9. Descripció de les fonts

Per posar-me en context he usat Wikipedia i la pàgina oficial de Kotlin. A més per posar-me en context he fet servir altres fòrums de programació per entendre l'asincronia principalment.

Per entendre les noves funcionalitats he fet servir molt la pàgina de programandoointentandolo, que explica molt bé funcionalitats com funcions d'extensió. Finalment, he requerit molt el canal de Youtube d'un desenvolupador, el BraisMoure, que m'ha sigut molt útil per lluitar contra la manca d'informació en la documentació i fòrums.

## 10. Avaluació de les fonts

Normalment, no em fiaria de primeres de Wikipedia o de fòrums d'internet, si es tractés de temes polítics o de societat, però tractant-se de desenvolupament de software, on és fàcil comprovar la veracitat del que es diu, simplement compilant els programes. Surten perdent si escampen notícies falses, ja que perden credibilitat i visites, en el cas dels fòrums. A més la gran majoria del que llegia coincidia amb la documentació oficial.

Exactament, el mateix passa tant pels programes que vaig llegir en programandoointentandolo com pel que vaig veure en els canals de YouTube. Vaig compilar els codis i provar-los en el meu editor i funcionaven a la perfecció, no hi ha gaire marge de desinformació en el món de la informàtica, no pas com altres disciplines socials on hi ha interès per desinformar intencionadament.

Finalment, el canal de BraisMoure em fio en especial, ja que és un desenvolupador de software conegut aquí a Espanya amb cert renom.