# Kotlin là gì?

Kotlin là một ngôn ngữ ngữ dụng kiểu tĩnh dành cho Java Virtual Machine đã chính thức phát hành phiên bản 1.0. Nó cũng sẽ chạy trên JVM (máy ảo java) và sử dụng các công cụ và thư viện hiện có của Java. Và ngược lại Java cũng có thể sử dụng các item được xây dựng trong Kotlin.

**Nguyên lý cốt lõi của Kotlin**

* Nhỏ gọn để giảm số lượng mã boilerplate cần viết.
* Mã code dễ đọc và dễ hiểu hơn.
* An toàn, tránh toàn bộ các lớp lỗi như các null pointer exceptions. Linh hoạt cho việc xây dựng ứng dụng phía máy chủ, ứng dụng Android hoặc mã lối vào đang chạy trong trình duyệt.
* Khả năng tương tác để tận dụng các khuôn khổ và thư viện hiện có của JVM với khả năng tương tác Java 100%.

Mã boilerplate đó là mã mà đã chứng tỏ hiệu quả và có thể được mở rộng đến nhiều ứng dụng. Mã để tạo ra các hoạt động toán học tiêu chuẩn, chương trình mẫu, và đáng chú ý nhất, mã nguồn mở tất cả có thể mã boilerplate xem xét.

**Ưu điểm của Kotlin**

* code Java và Kotlin có thể tồn tại song song trong cùng dự án, Vì Kotlin là hoàn toàn tương thích với Java, nên cũng có thể sử dụng phần lớn các thư viện Java và các framework trong dự án Kotlin của bạn—thậm chí nâng cao các framework dựa vào chú thích xử lý. => **Có thể thay thế Java**
* **Code ngắn gọn hơn**
* **Kết hợp những gì tốt nhất của lập trình hàm và thủ tục**

# Kiểu dữ liệu và khai báo biến trong Kotlin

* **Immutable variable: val keyword**

**Val**: là biến bất biến , không thể thay đổi được khi khai báo nhưng nó không phải là 1 giá trị cố định sẵn trong code (hard code), tức là có thể định nghĩa trong Runtime.

* **Const:**

Cũng đc biết đến như một biến bất biến nhưng khác với val là nó là giá trị cố định sẵn trong code. Trình compiler sẽ đưa giá trị này vào inline code trong quá trình biên dịch.

* **Mutable variable: var keyword**

Không giống như biến bất biến (Immutable), chúng ta có thể thay đổi giá trị của biến có thể thay đổi. Trong kotlin, chúng ta sử dụng từ khóa var để khai báo một biến mutable.

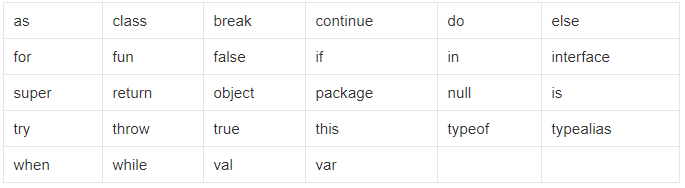
**var** dogName="Hachiko"

Nếu muốn khai báo rõ về loại biến muốn sử dụng thì “: tên kiểu dữ liệu”.

**var** dogName: String="Hachiko"

# Keyword và toán tử trong Kotlin

## **Hard Keywords**

Những từ khóa này không thể được sử dụng làm định danh. ví dụ****

## **Soft Keywords**

Soft Keywords là những từ khóa chỉ được sử dụng trong một ngữ cảnh nhất định, điều đó có nghĩa là chúng ta có thể sử dụng chúng làm định danh. Ngoài danh sách từ khóa trên, còn có những từ khóa khác có thể được sử dụng làm định danh. Ví dụ, "By" là một Soft keywords nó dùng để implement một đối tượng khác. Chúng ta cũng có thể sử dụng keywords "By" như một identifier (định danh).

//valid code

**fun** **main**(args: Array) {

**val** **by**=10

println(**by**+10)

}

Có một số soft keywords khác có sẵn trong Kotlin như Catch, Get, Finally, Field etc.

## **Kotlin Identifiers**

Tên mà chúng ta đặt cho một variable, class, function vv được gọi là identifier (định danh). Ví dụ:

**var** num = 100

# Câu lệnh rẽ nhánh và vòng lặp trong Kotlin

## **Câu lệnh When**

Đây là lệnh có công dụng giống với lệnh switch - case ở các ngôn ngữ lập trình khác như JAVA, PHP, C, C++.

**package** beginnersbook

**fun** **main** (args: Array<String>) {

**var** num = 78

**when**(num) {

**1** -> println("$num is a single digit number")

**in** 10..99 -> println("$num is a two digit number")

**in** 100..999 -> println("$num is a three digit number")

**else** -> println("$num has more than three digits")

}

}

## **Vòng lặp for**

* **Loại for thứ 1 – Duyệt tuần tự hết giá trị trong danh sách (closed range)**

**for** (i in a..b) { **X**ử **l**ý **bi**ế**n** **i** }

**for** (i **in** a **until** b) { Xử lý biến i } //[)

Với cú pháp ở trên thì biến i thực ra là biến bước nhảy, nó tự động tăng dần từ a cho tới b. Ta có thể thay tên biến i thành tên biến bất kỳ

Ví dụ 1: Viết chương trình tính giai thừa của một số nguyên dương n:

**fun** **main** (args: Array<String>) {

**var** gt:Int=1

**val** n:Int=5

**for** (i **in** 1..n)

    {

        gt \*= i

    }

    println("$n!=$gt")

}

* **Loại for thứ 2 – Điều hướng bước nhảy step**

**for** (i in a.. b step x) { **X**ử **l**ý **bi**ế**n** **i** }

Với cú pháp ở trên thì biến i thực ra là biến bước nhảy, nó tự động tăng dần từ a cho tới b, nhưng mỗi lần duyệt nó tăng theo x đơn vị. Ta có thể thay tên biến i thành tên biến bất kỳ

Ví dụ 2: Viết chương trình tính tổng các số chẵn nhỏ hơn hoặc bằng số nguyên dương n:

**fun** **main**(args: Array<String>) {

**var** sum:Int=0

**var** n:Int=10

**for** (i **in** 2 .. n step 2)

        sum+=i

    println("Tổng chẵn=$sum")

}

* **Loại for thứ 3 – Điều hướng bước nhảy downTo**

**for** (i in b downTo a) { **X**ử **l**ý **bi**ế**n** **i** }

Với cú pháp ở trên thì biến i thực ra là biến bước nhảy, nó tự động giảm dần từ b cho tới a, nhưng mỗi lần duyệt nó giảm 1 đơn vị. Ta có thể thay tên biến i thành tên biến bất kỳ

**for** (i in b downTo a step x) { **X**ử **l**ý **bi**ế**n** **i** }

Với cú pháp ở trên thì biến i thực ra là biến bước nhảy, nó tự động giảm dần từ b cho tới a, nhưng mỗi lần duyệt nó giảm x đơn vị. Ta có thể thay tên biến i thành tên biến bất kỳ

Ví dụ 3: Viết chương trình tính Ước số chung lớn nhất của 2 số bất kỳ

**fun** **main**(args: Array<String>) {

**var** a:Int=9

**var** b:Int=6

**var** ucscln=1

**var** min=**if** (a>b) b **else** a

**for** (i **in** min downTo 1)

    {

**if**(a%i==0 && b%i==0)

        {

            ucscln=i

**break**

        }

    }

    println("USCL của $a và $b = $ucscln")

}

Được thực hiện, với kết quả: USCL của 9 và 6 = 3

* **Loại for thứ 4 – Lặp tập đối tượng**

**for** (item in collection)

{

**println**(item)

}

Cấu trúc for trên sẽ duyệt từng đối tượng trong một tập đối tượng

Ví dụ 4: Duyệt danh sách tên sản phẩm

**fun** **main**(args: Array<String>) {

**var** dsSanPham= arrayOf("kotlin","java","c#","python","R")

**for** (item **in** dsSanPham)

        println(item)

}

Ngoài ra ta cũng có thể duyệt theo vị trí, từ vị trí này ta có thể xuất ra giá trị như sau:

**fun** **main**(args: Array<String>) {

**var** dsSanPham= arrayOf("kotlin","java","c#","python","R")

**for** (i **in** dsSanPham.indices)

        println("Sản phâm thứ $i có tên "+dsSanPham[i])

}

Cuối cùng kotlin cũng hỗ trợ vừa lấy vị trí vừa lấy giá trị theo cách sau:

**fun** **main**(args: Array<String>) {

**var** dsSanPham= arrayOf("kotlin","java","c#","python","R")

**for** ((index,value) **in** dsSanPham.withIndex())

    {

        println("Sản phẩm thứ $index có tên $value")

    }

}

## **Break, continue, return**

* **Continue:**  bỏ qua các câu lệnh tiếp theo trong vòng lặp để tiếp tục sang giá trị tiếp theo
* **Break:**

+ Trường hợp nếu dùng break trong 1 vòng lặp thì nó thoát luôn khỏi vòng lặp

+ Trường hợp dùng các **label@** để kiểm soát vị trí break

fun main (args: Array<String>) {  
 var gt:Int=1  
 val n:Int=5  
 label1@for (i in 1..n)  
 {  
 label2@for (j in 1..n){  
 gt \*= i  
 *print*("a = $i,b = $j")  
 break@label1; // thoát vòng lặp được đánh dấu là label1 => thoát cả 2 vòng lặp  
 break@label2; // thoát vòng lặp được đánh dấu là label2 => thoát vòng lặp j  
 }  
  
 }  
 *println*("$n!=$gt")  
}

* S

# Xử lý ngoại lệ trong Kotlin

Xử lý ngoại lệ trong Kotlin cũng tượng như trong Java. Và chúng ta đều biết, ngoại lệ là những lỗi có thể phát sinh lúc thực thi. Chẳng hạn như đọc một file không còn tại. Hoặc kết nối với database không có sẵn.

## **Cú pháp**

**try** {

Xử lý

} **catch** (e: ExceptionName) {

Xử lý ngoại lệ

} **finally** {

Giải phóng tài nguyên

}

Để ném ra ngoại lệ chúng ta sử dụng throw. Tuy nhiên để bắt ngoại lệ, chúng ta sử dụng try…catch như trên. Đối với finally có thể bỏ qua khi viết try…catch và những câu lệnh bên trong nó luôn luôn được thực thi bất kể cho xảy ra ngoại lệ hay không.

Sau đây là một số exception thông dụng

1. ArithmeticException: Ngoại lệ liên quan đến tính toán số học
2. NumberFormatException: Ngoại lệ liên quan đến định dạng số
3. IllegalArgumentException: Ngoại lệ liên quan đến đối số không phù hợp
4. Exception: Ngoại lệ tổng quát nhất

Khi xử lý ngoại lệ, chúng ta có thể chỉ định cụ thể từng ngoại lệ trên hoặc sử dụng Exception.

## **Ví dụ**

**fun** **main**(args: Array<String>) {

**try** {

**val** age:Int = 12

**val** name:String = "Lê Hồng Kỳ"

name.toInt()

} **catch**(e:Exception) {

e.printStackTrace()

} **finally** {

println("Xử lý ngoại lệ trong Kotlin")

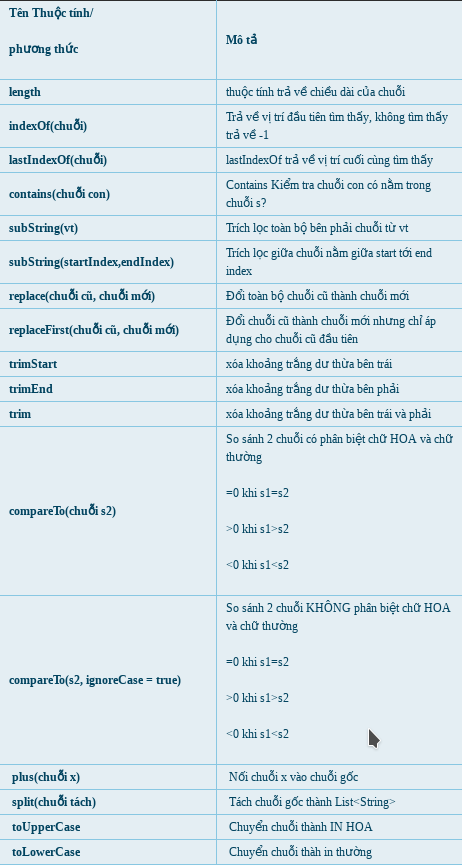
}

}

Khi chạy ví dụ trên, chương trình sẽ phát sinh một ngoại lệ tại dòng name.toInt() vì không thể chuyển một chuỗi sang dạng số int được. Với chương trình trên, nếu không xử lý ngoại lệ thì chương trình sẽ phát sinh lỗi và kết thúc bất thường.

# Xử lý chuỗi trong Kotlin

* Khai báo chuỗi giống java với C/C++…
* Lấy độ dài chuỗi thì .length
* So sánh 2 chuỗi thì sử dụng phương thức equals () hoặc sử dụng phương thức compareTo().



## Hàm IndexOf

– trả về vị trí đầu tiên tìm thấy, nếu không thấy trả về -1:

*/\*\**

*\* Created by cafe on 28/05/2017.*

*\*/*

fun **main**(args: Array<String>) {

**var** s:String="Obama"

**var** i = 0

*// Trả về i = 2*

i = s.indexOf("a")

println(i)

}

Kq: 2

## lastIndexOf

trả về vị trí cuối cùng tìm thấy, nếu không thấy trả về -1:

Thay indefOf = lastIndexOf thì giá trị trả về là 5

* Contains kiểm tra chuỗi con có tồn tại trong chuỗi gốc hay không, có thì trả về true, còn không trả về false

*/\*\**

*\* Created by cafe on 28/05/2017.*

*\*/*

fun **main**(args: Array<String>) {

**var** s :String = "Hello everybody !"

**var** s2="body"

**if**(s.contains(s2))

{

println("Có tồn tại [$s2]")

}

**else**

{

println("Ko tồn tại [$s2]")

}

}

Kết quả chạy ta thấy xuất “Có tồn tại [body]”. Hàm contains được sử dụng rất nhiều để kiểm tra sự tồn tại của chuỗi con.

## subString – trích lọc chuỗi

*/\*\**

*\* Created by cafe on 28/05/2017.*

*\*/*

fun **main**(args: Array<String>) {

val s = "Xin chào Obama! Tui là Putin"

val s2 = s.substring(9)

println(s2)

val s3 = s.substring(9, 14)

println(s3)

}

Kết quả khi chạy ta thấy:  
  
Muốn trích lọc bên phải chuỗi thì dùng substring với 1 đối số, muốn trích lọc giữa chuỗi thì dùng 2 đối số.

## replace – đổi chuỗi cũ thành chuỗi mới

*/\*\**

*\* Created by cafe on 28/05/2017.*

*\*/*

fun **main**(args: Array<String>) {

**var** s = "Xin chào Obama! Tui là Putin"

s = s.replace("Obama", "Kim Jong Un")

println(s)

}

Kết quả khi chạy ta thấy:  


## replaceFirst-Đổi chuỗi cũ thành chuỗi mới nhưng chỉ áp dụng cho chuỗi đầu tiên tìm thấy

*/\*\**

*\* Created by cafe on 28/05/2017.*

*\*/*

fun **main**(args: Array<String>) {

**var** s = "Obama Xin chào Michelle Obama"

s = s.replaceFirst("Obama", "Putin")

println(s)

}

Kết quả khi chạy ta thấy:  
  
Mặc dù chuỗi gốc ta có 2 chuỗi con Obama, nhưng chương trình chỉ áp dụng cho Obama đầu tiên

## Các hàm xóa khoảng trắng: trim,trimEnd,trimStart

*/\*\**

*\* Created by cafe on 28/05/2017.*

*\*/*

fun **main**(args: Array<String>) {

**var** s:String= " Trần Duy Thanh "

val s2=s.trimStart()

println(s2+"=>size="+s2.length)

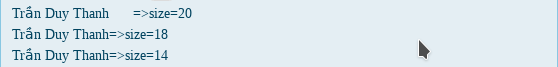
val s3=s.trimEnd()

println(s3+"=>size="+s3.length)

val s4=s.trim()

println(s4+"=>size="+s4.length)

}

Kết quả khi chạy ta thấy:  


## compareTo – so sánh chuỗi

*/\*\**

*\* Created by cafe on 28/05/2017.*

*\*/*

fun **main**(args: Array<String>) {

val s1 = "Hạnh phúc"

val s2 = "Hạnh PHÚC"

val x = s1.compareTo(s2, ignoreCase = **true**)

println(x)

val y = s1.compareTo(s2, ignoreCase = **false**)

println(y)

}*/\*\**

*\* Created by cafe on 28/05/2017.*

*\*/*

fun **main**(args: Array<String>) {

val s1 = "Hạnh phúc"

val s2 = "Hạnh PHÚC"

val x = s1.compareTo(s2, ignoreCase = **true**)

println(x)

val y = s1.compareTo(s2, ignoreCase = **false**)

println(y)

}

Kết quả khi chạy ta thấy:  
  
Ta thấy nếu so sánh không phân biệt chữ Hoa – thường(ignoreCase = true) thì kết quả =0, còn có phân biệt chữ HOA – thường(ignoreCase = false) thì kết quả >1 (32)

## plus– nối chuỗi

*/\*\**

*\* Created by cafe on 28/05/2017.*

*\*/*

fun **main**(args: Array<String>) {

**var** s:String="Obama"

s=s.plus(" ")

s=s.plus("Putin")

println(s)

}

Kết quả khi chạy ta thấy:  
  
Hàm plus ở trên nối chuỗi nhưng nó không làm thay đổi chuỗi gốc mà nó trả về 1 chuỗi mới do đó ta phải lưu lại địa chỉ mới này. Ta có thể dùng dấu + (nhưng không nên dùng khi nối chuỗi quá nhiều đặc biệt đọc từ file hay internet), Ngoài ra có thể dùng StringBuilder để xử lý nối chuỗi.

## split– tách chuỗi theo tiêu chí bất kỳ, trả về một List

*/\*\**

*\* Created by cafe on 28/05/2017.*

*\*/*

fun **main**(args: Array<String>) {

**var** s:String="Trần Duy Thanh"

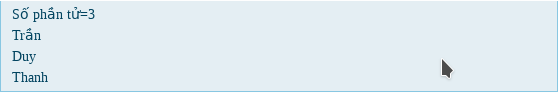
**var** arr:List<String> = s.split(" ")

println("Số phần tử="+arr.size)

**for**(x in arr)

println(x)

}

Kết quả khi chạy ta thấy:  
  
Hàm tách chuỗi rất quan trọng, vì hầu như ta đều gặp trong quá trình viết phần mềm. Input là 1 chuỗi có quy luật, và ta phải phân tích để tách chuỗi rồi mô hình hóa thành dữ liệu có cấu trúc(thường là OOP)

## toUpperCase, toLowerCase để chuyển CHỮ HOA, chữ thường

*/\*\**

*\* Created by cafe on 28/05/2017.*

*\*/*

fun **main**(args: Array<String>) {

**var** s:String="Trần Duy Thanh"

**var** s2=s.toUpperCase()

println(s2)

**var** s3=s.toLowerCase()

println(s3)

}

Kết quả khi chạy ta thấy:  


# Collections trong Kotlin

## **Mảng (Array)**

Array trong Kotlin có chiều dài cố định với phần tử đầu tiên bắt đầu ở vị trí 0. Array có thể chứa giá trị kiểu chuỗi, kiểu số, …

Tương tự như trên ta có thể sử dụng floatArrayOf(), longArrayOf(), … để chứa các kiểu dữ liệu tương ứng. Và để xác định kích thước của array, chúng ta có thể sử dụng lệnh size.

val san\_pham = arrayOf("iPhone", "iPad", "iMac")

val thu = intArrayOf(2, 3, 4, 5, 6, 7, 8)

## **List**

Giống như một mảng (Array), ngoại trừ chúng có thể có một số lượng các phần tử thay đổi tương tự như ArrayList trong Java. Các phần tử không chỉ được thêm vào mà còn có thể được xóa khỏi danh sách.

Ví dụ về việc sử dụng List trong Kotlin

**val** san\_pham = mutableListOf<String>("iPhone", "iPad", "iMac")

// Xóa phần tử cuối cùng

san\_pham.removeAt(2)

// Thêm phần tử vào cuối danh sách

san\_pham.add("iMatch")

## **Set**

Giống như List ngoại trừ mỗi phần tử phải là duy nhất. Chúng ta không thể lấy một phần tử theo vị trí và các phần tử hiện tại không thể thay đổi.

Ví dụ sử dụng Set trong Kotlin

**val** nt = mutableSetOf<Int>(3, 5, 7, 0, 13)

// Thêm một số vào cuối

nt.add(2)

// Xóa số 0, lưu ý 0 là số 0 chứ không phải vị trí

nt.remove(0)

## **Map**

Map trong Kotlin cũng tương tự HashMap trong Java. Chúng lưu trữ một cặp gồm key và value. Key là duy nhất và mỗi key chỉ lưu trữ được một value.

Ví dụ tạo Map chứa danh sách các hãng công nghệ

val hang = mutableMapOf("a" to "Apple", "s" to "Samsung")

println(hang["a"])