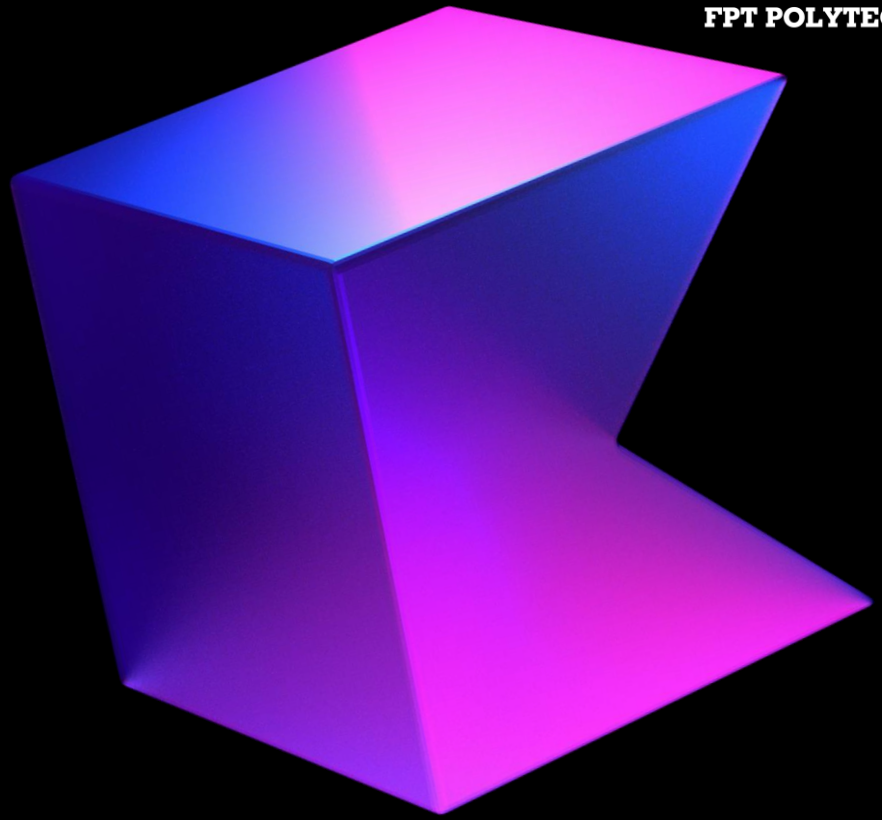
JET BRAINS Kotlin

FPT Education
FPT POLYTECHNIC

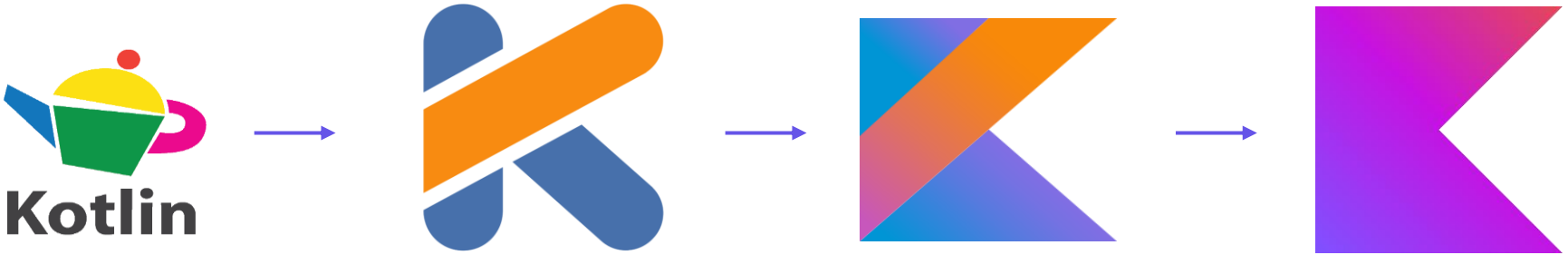
Giới thiệu về Kotlin



Nội dung

- Giới thiệu về Kotlin
- Cú pháp
- Cách tạo biến
- Kiểu dữ liệu
- Xử lý tính toán

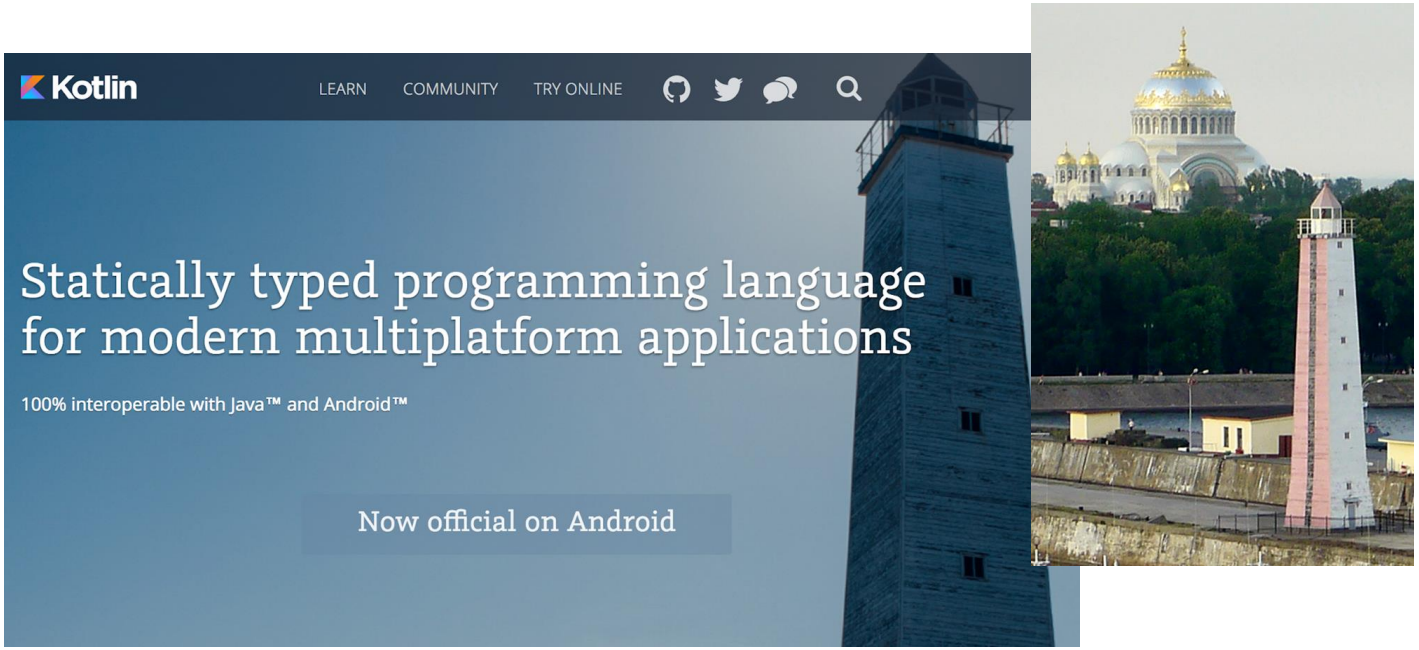
Logo



Kotlin là gì?

- **Kotlin** là ngôn ngữ lập trình được phát triển bởi JetBrains. Nó xuất hiện lần đầu năm 2011 khi JetBrains công bố dự án của họ mạng tên "Kotlin". Đây là một ngôn ngữ mã nguồn mở
- Về cơ bản, cũng như Java, C hay C++ , Kotlin cũng là "***ngôn ngữ lập trình kiểu tĩnh***". Nghĩa là các biến không cần phải định nghĩa trươc khi sử dụng. Kiểu tĩnh thực hiện việc kê khai nghiêm ngặt hoặc khởi tạo các biến trước khi chúng được sử dụng

Tại sao lại là Kotlin?



Kotlin được đặt theo tên một hòn đảo ở Vịnh Phần Lan.

Ưu điểm Kotlin

- Kotlin biên dịch thành JVM bytecode hoặc JavaScript - Giống như Java, Bytecode cũng là format biên dịch cho Kotlin. Bytecode nghĩa là một khi đã biên dịch, các đoạn code sẽ chạy thông qua một máy ảo thay vì một bộ xử lý. Bằng cách này, code có thể chạy trên bất kỳ nền tảng nào, khi nó được biên dịch và chạy thông qua máy ảo. Khi Kotlin được chuyển đổi thành bytecode, nó có thể truyền được qua mạng và thực hiện bởi JVM
- Kotlin có thể sử dụng tất cả các nền tảng và thư viện Java hiện có - Bất kể là nền tảng cao cấp dựa trên xử lý annotation. Điều quan trọng là Kotlin dễ dàng tích hợp với Maven, Gradle hay các hệ thống build khác.

Ưu điểm Kotlin

- Kotlin dễ học và dễ tiếp cận. Có thể học dễ dàng bằng những ví dụ tham khảo đơn giản. Cú pháp đơn giản và trực quan. Kotlin khá giống Scala nhưng dễ hiểu hơn
- Kotlin là mã nguồn mở nên không tốn kém gì để có thể sử dụng
- Chuyển đổi tự động Java thành Kotlin - JetBrains tích hợp chức năng mới vào IntelliJ để chuyển đổi Java thành Kotlin và điều này tiết kiệm một lượng thời gian rất lớn.

Ưu điểm Kotlin

- Null-safety của Kotlin: giúp chúng ta thoát khỏi NullPointerExceptions. Giúp chúng ta tránh những exeption kiểu con trỏ null. Trong Kotlin, hệ thống sẽ từ chối biên dịch đoạn code đang gán hay trả về giá trị null
- Review code không còn là vấn đề - Kotlin tập trung nhiều hơn vào việc cú pháp dễ hiểu, dễ đọc để review

Hello, world!

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

fun main() {
      println("Hello, world!")
}

fun main() = println("Hello, world!")
```

Dấu “;” ở đâu rồi???

Cú pháp cơ bản

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

- main: Điểm bắt đầu (Entry point) - Đây chính là chức năng chính của chương trình.
- Nó có thể chấp nhận số lượng đối số Chuỗi (String) thay đổi.
- Câu lệnh print sẽ in ra màn hình giá trị được truyền vào.
- Câu lệnh println hoạt động tương tự print nhưng sẽ xuống dòng.

Biến

`val/var myValue: Type = someValue`

- `var` – **mutable (Các biến có thể được gán lại giá trị)**
- `val` – **immutable (Các biến cục bộ chỉ cho phép chỉ đọc .Chúng chỉ có thể được gán một giá trị một lần)**
- Type can be inferred in most cases
- Assignment can be deferred

```
val a: Int = 1          // immediate assignment

var b = 2               // 'Int' type is inferred
b = a                           // Reassigning to 'var' is okay

val c: Int              // Type required when no initializer is provided
c = 3                           // Deferred assignment
a = 4                           // Error: Val cannot be reassigned
```

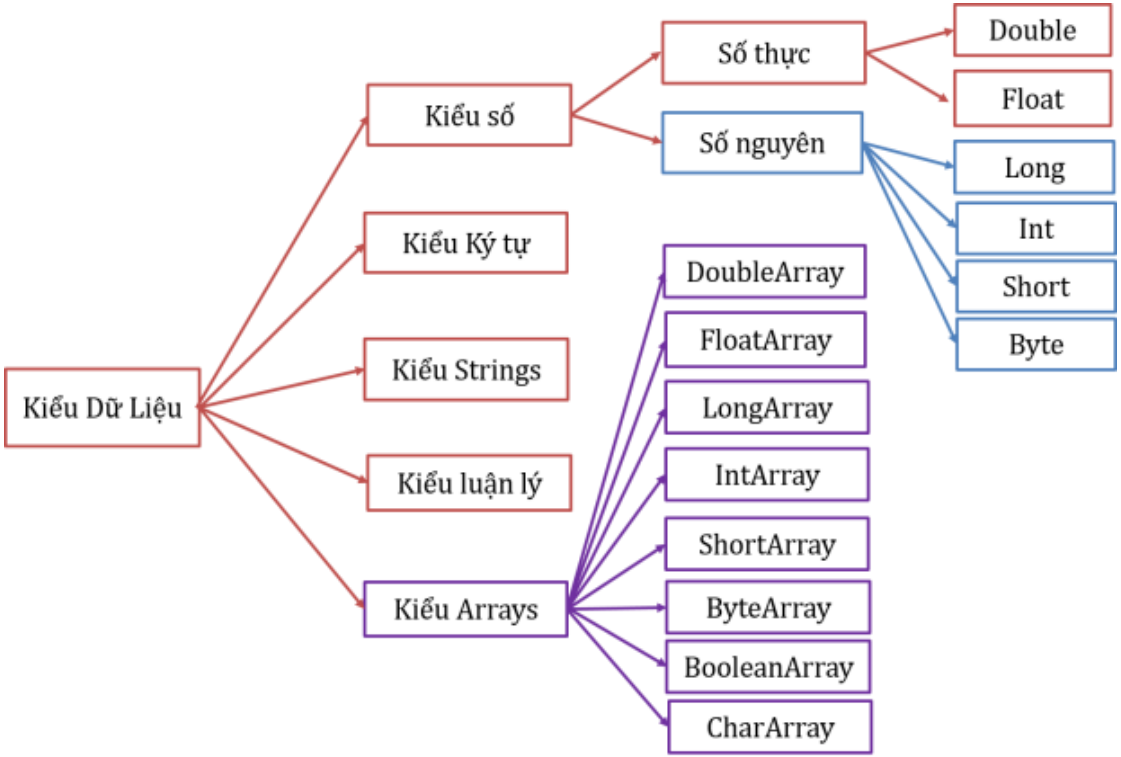
Biến

`const val/val myValue: Type = someValue`

- `const val` - compile-time const value
- `val` - immutable value
- for `const val` use uppercase for naming

```
const val NAME = "Kotlin" // can be calculated at compile-time
val nameLowered = NAME.lowercase() // cannot be calculated at compile-time
```

Kiểu dữ liệu


Kiểu Dữ Liệu
Kiểu số
Kiểu Ký tự
Kiểu Strings
Kiểu luận lý
Kiểu Arrays
Số thực
Số nguyên
Double
Float
Long
Int
Short
Byte
DoubleArray
FloatArray
LongArray
IntArray
ShortArray
ByteArray
BooleanArray
CharArray

Kiểu dữ liệu

Số thực sẽ có 2 loại đó là **Double** và **Float**, Nếu có hằng số để xác định nó là số thực nào thì ta có thể thêm ký tự **f** hoặc **F** đằng sau hằng số đó:

Ví dụ: 113.5 → kiểu **Double**, còn 113.5**F** hoặc 113.5**f** → kiểu **Float**

Số nguyên có 4 loại : **Long**, **Int**, **Short**, **Byte**. Ta chú ý trường hợp hằng số của **Long** và **Int** bằng cách thêm ký tự **L**

Ví dụ: 113 → kiểu **Int**, còn 113**L** → kiểu **Long** (***không dùng l thường***)

Kiểu dữ liệu

Cú pháp:

var tên_biến : [Kiểu_Dữ_Liệu] = Giá_Trị_Mặc_Định

var x = 100L **hoặc** var x:Long=100L

var y = 113.5 **hoặc** var y:Double=113.5

var f = 113.5f **hoặc** var f:Float=113.5f

var i = 113 **hoặc** var i:Int =113

var s = 8 **hoặc** var s:Short=8

var b = 1 **hoặc** var b:Byte=1

Kiểu dữ liệu

- Kiểu ký tự dùng để lưu trữ **một ký tự** nằm trong **nháy đơn**
- Kiểu chuỗi dùng để lưu tập các ký tự được để trong cặp **nháy đôi**, dùng đối tượng **String** để khai báo
- Ngoài ra ta có thể khai báo chuỗi trên nhiều dòng bằng cách để trong cặp **3 dấu nháy kép**

```
fun main() {
    val kytu = 'c'
    val ten = "Nguyễn Văn Anh"
    val diachi = """
        Quận 12,
        TPHCM
    """
    println(kytu)
    println(ten)
    println(diachi)
}
```

Kiểu dữ liệu

Với các kiểu dữ liệu **Mảng**, Kotlin cung cấp cho ta 8 loại mảng ứng với 8 kiểu dữ liệu được built-in trong Kotlin (***ngoại trừ String***).

Để khai báo và sử dụng kiểu dữ liệu ta làm như sau:

var Tên_Mảng = XXXArrayOf(giá trị 1, giá trị 2,..., giá trị n)

Với **XXX** là 8 kiểu dữ liệu tương ứng (***viết thường ký tự đầu***)

Kiểu dữ liệu

```
fun main() {
    val arrX = intArrayOf(1, 2, 3, 5)
    println(arrX[1])
    val arrY = doubleArrayOf(1.5, 2.6, 9.0, 10.3)
    println(arrY[3])
    val arrC = charArrayOf('a', 'b', 'c')
    println(arrC[0])
}
```

Hàm

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

fun mul(a: Int, b: Int) = a * b

fun printMul(a: Int, b: Int): Unit {
            println(mul(a, b))
}

fun printMul1(a: Int = 1, b: Int) {
            println(mul(a, b))
}

fun printMul2(a: Int, b: Int = 1) = println(mul(a, b))
```

Single expression function.

`Unit` means that the function does not return anything meaningful.

It can be omitted.

Arguments can have **default** values.

Toán tử một ngôi

Toán tử	Mô tả	Phương thức	Ví dụ
+	Số dương	a.unaryPlus()	var a:Int=-8 var b=a.unaryPlus()
–	Số âm	a.unaryMinus()	var a:Int=-8 var b=a.unaryMinus()

```
fun main() {
    val a: Int = -8
    val b = a.unaryPlus()
    val c = a.unaryMinus()
    println("a = " + a)
    println("b = " + b)
    println("c = " + c)
}
```

Kết quả ta được:

a = -8

b = -8

c = 8

Lưu ý: unaryPlus và **unaryMinus** không làm thay đổi giá trị nội tại của biến, mà nó trả về một giá trị, ta cần có biến để lưu lại giá trị này.

Toán tử số học

Toán tử	Mô tả	Phương thức	Ví dụ
+	Cộng	a.plus(b)	8 + 5 => 13 8.plus(5) =>13
–	Trừ	a.minus(b)	8-5 =>3 8.minus(5) =>3
*	Nhân	times	8*5 =>40 8.times(5) =>40
/	Chia	div	8/5 => 1 8.div(5) => 1
%	Chia lấy phần dư	rem	8 % 5 => 3 8.rem(5) =>3

Toán tử số học

```
fun main() {
    val a = 8
    val b = 5
    println("Phép cộng cách 1:" + a + "+" + b + "=" + (a + b))
    println("Phép cộng cách 2:" + a + "+" + b + "=" + (a.plus(b)))
    println("Phép trừ cách 1:" + a + "-" + b + "=" + (a - b))
    println("Phép trừ cách 2:" + a + "-" + b + "=" + (a.minus(b)))
    println("Phép nhân cách 1:" + a + "*" + b + "=" + (a * b))
    println("Phép nhân cách 2:" + a + "*" + b + "=" + (a.times(b)))
    println("Phép chia cách 1:" + a + "/" + b + "=" + (a / b))
    println("Phép chia cách 2:" + a + "/" + b + "=" + (a.div(b)))
    println("Lấy dư cách 1:" + a + "/" + b + "=" + (a % b))
    println("Lấy dư cách 2:" + a + "/" + b + "=" + (a.rem(b)))
    println(8.plus(9))
}
```

```
Phép cộng cách 1:8+5=13
Phép cộng cách 2:8+5=13
Phép trừ cách 1:8-5=3
Phép trừ cách 2:8-5=3
Phép nhân cách 1:8*5=40
Phép nhân cách 2:8*5=40
Phép chia cách 1:8/5=1
Phép chia cách 2:8/5=1
Lấy dư cách 1:8/5=3
Lấy dư cách 2:8/5=3
17
```

Toán tử gán

Toán tử	Mô tả	Ví dụ	Tương đương với
=	Phép gán giá trị bên phải cho biến bên trái dấu bằng	x=5	
+=	Cộng và gán	x=2 x+=5 ==>x=7	x=x+5
-=	Trừ và gán	x=2 x-=5 ==>x=-3	x=x-5
*=	Nhân và gán	x=2 x*=5 ==>x=10	x=x*5
/=	Chia và gán	x=7 x/=5 ==>x=1	x=x/5
%=	Chia lấy dư	x=7 x%=5 ==>x=2	x=x%5

Toán tử gán

```
fun main() {
    var x  = 5
    x += 2
    println("x=" + x)
    x -= 2
    println("x=" + x)
    x *= 2
    println("x=" + x)
    x /= 2
    println("x=" + x)
    x = 7
    x %= 3
    println("x=" + x)
}
```

```
x=7
x=5
x=10
x=5
x=1
```

Toán tử so sánh

Toán tử	Mô tả	Phương thức	Ví dụ
==	So sánh bằng	a.equals(b)	5 == 5 => kết quả True
!=	So sánh không bằng	!a.equals(b)	5 != 5 => kết quả False
<	So sánh nhỏ hơn	a.compareTo(b) < 0	5 < 5 => kết quả False
<=	So sánh nhỏ hơn hoặc bằgg	a.compareTo(b) <= 0	5 <= 5 => kết quả True
>	So sánh lớn hơn	a.compareTo(b) > 0	5 > 5.5 => kết quả False
>=	So sánh lớn hơn hoặc bằng	a.compareTo(b) >= 0	113>= 5 => kết quả True

Toán tử so sánh

```
fun main() {
    val a = 8
    val b = 5
    println(a == b)
    println(a.equals(b))
    println(!a.equals(b))
    println(a.compareTo(b))
    println(3.compareTo(3))
    println(3.compareTo(5))
    println(5.compareTo(3))
}
```

```
false
false
true
1
0
-1
1
```

Toán tử logic

Toán tử	Mô tả	Phương thức	Ví dụ
&&	Toán tử Và: Nếu cả hai điều kiện là True thì kết quả sẽ là True	a.and(b)	x=false y=true x&&y==>false
||	Toán tử Hoặc: Chỉ cần một điều kiện True thì nó True, tất cả điều kiện False thì nó False	a.or(b)	x=false y=truex||y==>true

Toán tử logic

```
fun main() {
    val x = true
    val y = false
    val z = false
    println("x=" + x)
    println("y=" + y)
    println("z=" + z)
    println("x&&y=" + (x && y))
    println("x.and(y)=" + x.and(y))
    println("x || y =" + (x || y))
    println("x.or(y)=" + x.or(y))
    println("x || z =" + (x || z))
}
```

```
x=true
y=false
z=false
x&&y=false
x.and(y)=false
x || y =true
x.or(y)=true
x || z =true
```

Toán tử logic

```
fun main() {
    val x = true
    val y = false
    val z = false
    println("x.or(z)=" + x.or(z))
    println("x && z =" + (x && z))
    println("x.and(z)=" + x.and(z))
    println("y || z =" + (y || z))
    println("y.or(z)=" + y.or(z))
    println("y && z =" + (y && z))
    println("y.and(z)=" + y.and(z))
    println("x && y && z =" + (x && y && z))
    println("x.and(y).and(z)=" + x.and(y).and(z))
    println("x|| y||z =" + (x || y || z))
    println("x.or(y).or(z)=" + x.or(y).or(z))
}
```

```
x.or(z)=true
x && z =false
x.and(z)=false
y || z =false
y.or(z)=false
y && z =false
y.and(z)=false
x && y && z =false
x.and(y).and(z)=false
x|| y||z =true
x.or(y).or(z)=true
```

Toán tử tăng dần/giảm dần

Toán tử	Mô tả	Phương thức	Ví dụ
++	Tăng nội tại biến lên một đơn vị	a.inc()	x=5 x++ =>x=6
—	giảm nội tại biến xuống một đơn vị	a.dec()	x=5 x–==>x=4

Toán tử tăng dần/giảm dần

Hàm **inc()** và **dec()** sẽ không làm thay đổi (tăng hay giảm) giá trị của biến, ta cần có biến khác để lưu giá trị bị thay đổi.

Với toán tử tăng dần và giảm dần này thì việc đặt ++ hay -- đằng trước và đằng sau biến có ý nghĩa khác nhau trong một biểu thức phức tạp, thông thường nó sẽ hoạt động theo nguyên tắc (có một số trường hợp đặc biệt phá vỡ nguyên tắc này ta không xét tới vì nó vô cùng hiếm gặp, chủ yếu do Testing Problem):

- Bước 1: Ưu tiên xử lý Prefix trước (các ++ hay -- đặt trước biến)
- Bước 2: Tính toán các phép toán còn lại trong biểu thức
- Bước 3: Gán giá trị ở bước 2 cho biến lưu trữ kết quả ở bên trái dấu bằng
- Bước 4: Xử lý Post fix (các ++ hay -- đặt sau biến)

Toán tử tăng dần/giảm dần

```
fun main() {
    var a = 5
    var b = 8
    var c = 2
    a--
    b++
    val z = a++ + ++b - --c + 7
    println("a=" + a)
    println("b=" + b)
    println("c=" + c)
    println("z=" + z)
}
```

```
a=5
b=10
c=1
z=20
```

Độ ưu tiên toán tử

Kotlin có ràng buộc thứ tự ưu tiên của các toán tử. Tuy nhiên tốt nhất là các bạn hay điều khiển nó bằng cách dùng cặp ngoặc tròn ( ) để nó rõ nghĩa hơn. Bảng dưới đây để tham khảo độ ưu tiên từ cao xuống thấp (tuy nhiên có thể quên nó đi mà hãy dùng ngoặc tròn () để chỉ định rõ).

Thứ tự ưu tiên	Toán tử	Miêu tả
1	* / %	Phép nhân, chia, lấy phần dư
2	+ –	Toán tử Cộng, Trừ
3	<= < > >=	Các toán tử so sánh
4	<> == !=	Các toán tử so sánh
5	= %= /= -= += *=	Các toán tử gán
6	&& , ||	Các toán tử logic

Tài liệu tham khảo

- kotlinlang.org
- kotlinlang.org/docs
- play.kotlinlang.org/byExample

Thanks!