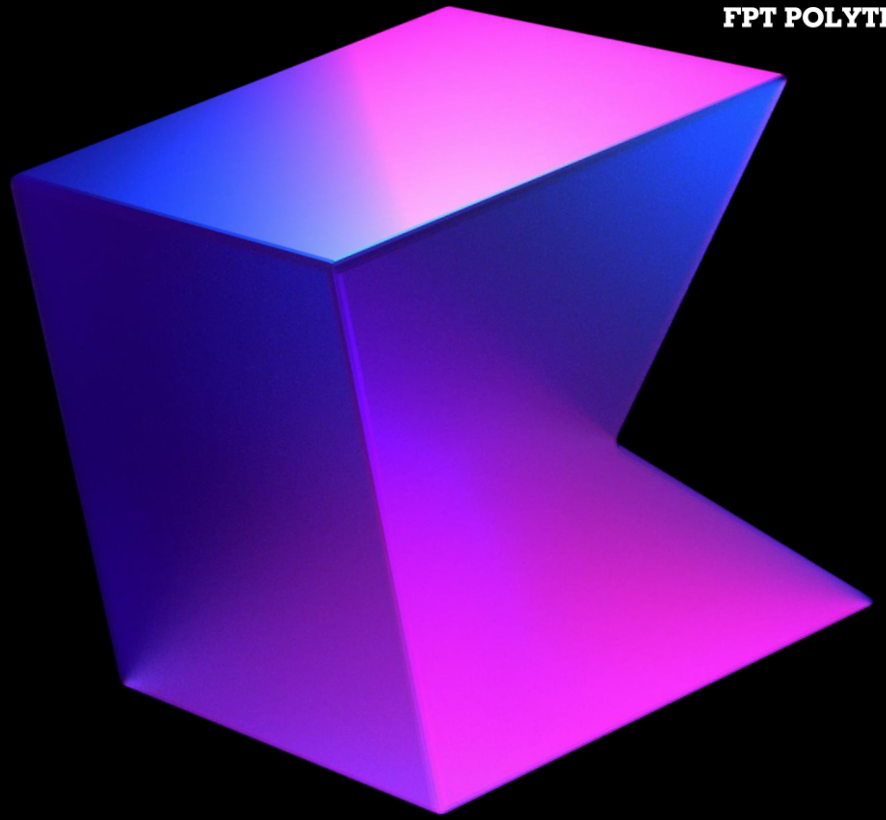
JET BRAINS Kotlin

FPT Education
FPT POLYTECHNIC

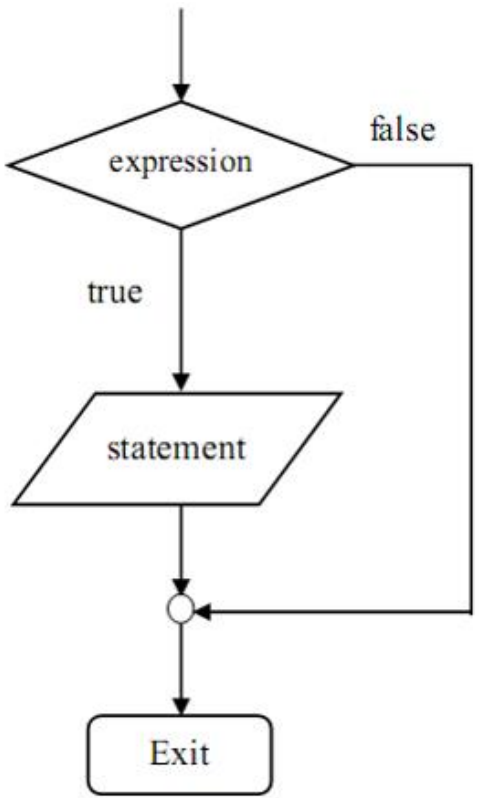
Giới thiệu về Kotlin (tiếp theo)

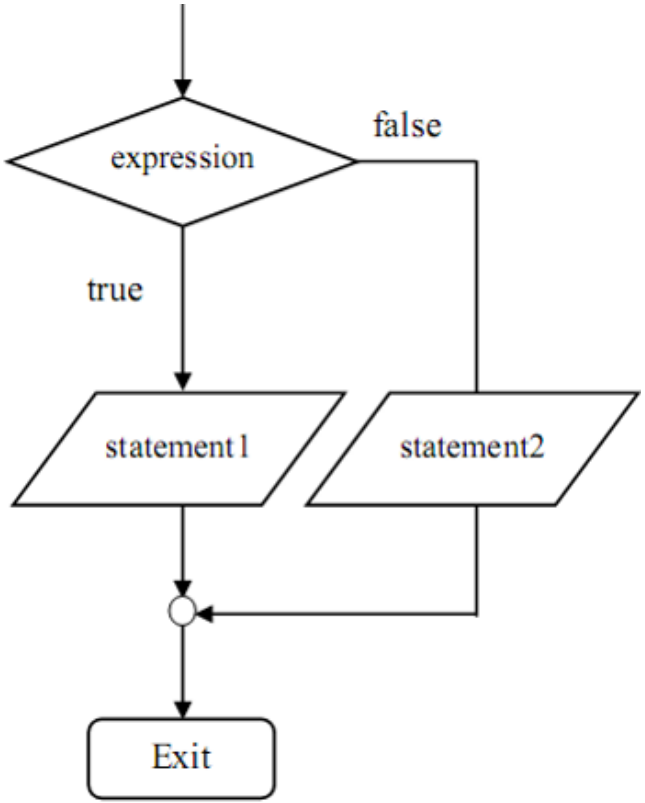


Nội dung

- Câu điều kiện
- Vòng lặp
- Xử lý logic

Câu điều kiện (if – if else)





Câu điều kiện (if, if else)

Cách viết truyền thống

```
fun maxOf(a: Int, b: Int): Int {
        if (a > b) {
                return a
        } else {
                return b
        }
}
```

tương đương

Cách viết theo biểu thức

```
fun maxOf(a: Int, b: Int) =
    if (a > b) {
        a
    } else {
        b
    }
```

Lưu ý: khi viết if với dạng biểu thức trả về kết quả thì bắt buộc phải có else

`if` có thể là một biểu thức (có thể trả về giá trị).

Có thể viết trên cùng một dòng:

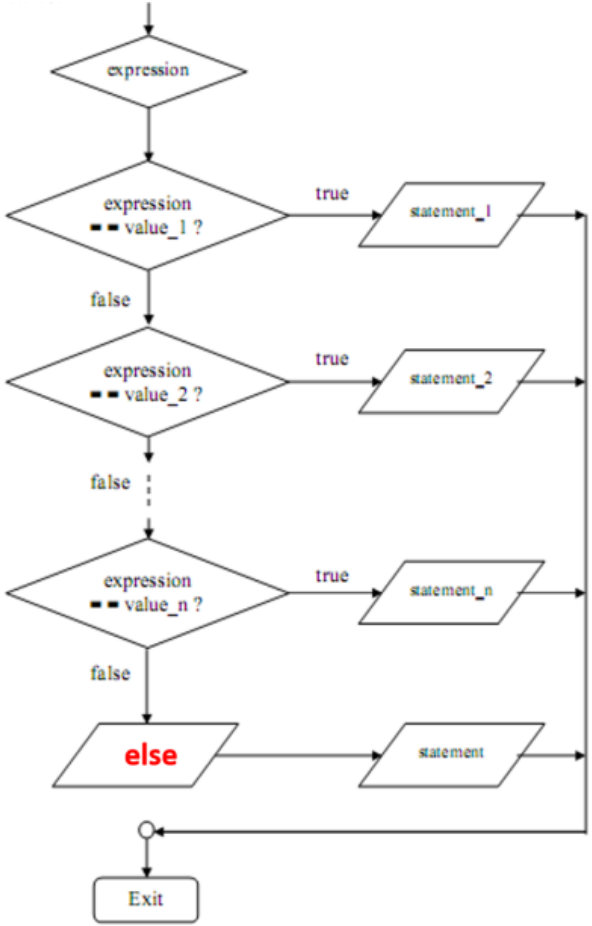
```
fun maxOf(a: Int, b: Int) = if (a > b) a else b
```

Biểu thức When

Cú pháp:

```
when(<expression>) {
        <value 1>   ->    <statement 1>
        <value 2>   ->    <statement 2>
        else        ->    <statement else>
}
```

when lấy giá trị trong **<expression>**, đem so sánh với các **<value>** bên trong , nếu trùng khớp với **value** nào thì **<statement>** đó sẽ được thực thi. Nếu tất cả **<value>** đều không khớp với **<expression>** thì **else** sẽ được thực hiện.



Biểu thức When

```
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> {
        print("x is neither 1 nor 2")
    }
}
```

`when` returns, the same way that `if` does.

```
when {
    x < 0 -> print("x < 0")
    x > 0 -> print("x > 0")
    else -> {
        print("x == 0")
    }
}
```

The condition can be inside of the branches.

Câu lệnh When

```
fun serveTeaTo(customer: Customer) {
            val teaSack = takeRandomTeaSack()

      when (teaSack) {
          is OolongSack -> error("We don't serve Chinese tea like $teaSack!")
          in trialTeaSacks, teaSackBoughtLastNight ->
                 error("Are you insane?! We cannot serve uncertified tea!")
      }

      teaPackage.brew().serveTo(customer)
}
```

`when` can accept several options in one branch. `else` branch can be omitted if `when` block is used as a *statement*.

&& vs and

```
if (a && b) { ... }     VS     if (a and b) { ... }
```

Unlike the `&&` operator, this function does not perform short-circuit evaluation.

The same behavior with OR:

```
if (a || b) { ... }     VS     if (a or b) { ... }
```

Vòng lặp (for)

1. Duyệt tuần tự hết giá trị trong danh sách (**closed range**)

Cú pháp:

```
for (i in a..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

Vòng lặp (for)

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

```
fun main() {
    var gt = 1
    val n = 5
    for (i in 1..n) {
        gt *= i
    }
    println("$n!=$gt")
}
```

Vòng lặp (for)

2. Duyệt tuần tự gần hết giá trị trong danh sách (**half-open range**)

Cú pháp:

```
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 **gần** b

Vòng lặp (for)

Ví dụ: Viết chương trình tính tổng từ 1 tới gần số nguyên dương n:

```
fun main() {
    var sum = 0
    val n = 5
    for (i in 1 until n) {
        sum += i
    }
    println("Tổng=$sum")
}
```

Vòng lặp (for)

3. Điều hướng bước nhảy step

Cú pháp:

```
for (i in a .. b step x)
{
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, nhưng mỗi lần duyệt nó tăng theo x đơn vị

Vòng lặp (for)

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

```
fun main() {
    var sum = 0
    val n = 10
    for (i in 2..n step 2)
        sum += i
    println("Tổng chẵn=$sum")
}
```

Vòng lặp (for)

4. Điều hướng bước nhảy downTo

Cú pháp:

```
for (i in b downTo a)
{
Xử lý biến 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

Vòng lặp (for)

4. Điều hướng bước nhảy downTo

Hoặc

Cú pháp:

```
for (i in b downTo a step x)
{
Xử lý biến 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

Vòng lặp (for)

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

```
fun main() {
    val a = 9
    val b = 6
    var ucscln = 1
    val 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")
}
```

Vòng lặp (for)

5. Lặp tập đối tượng

Cú pháp:

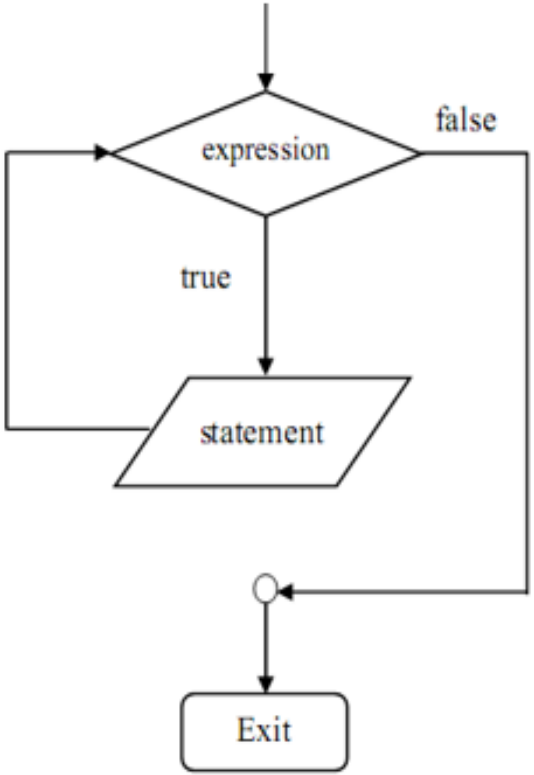
```
for (item in collection)
{
   println(item)
}
```

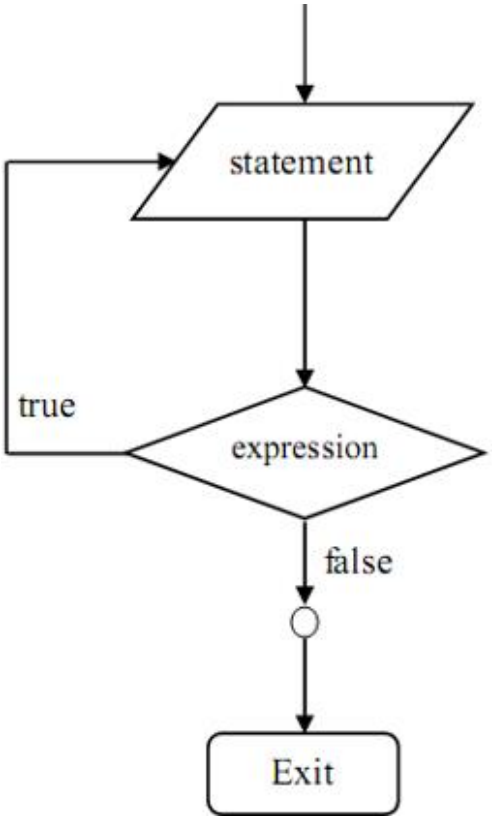
Cấu trúc for trên sẽ duyệt từng đối tượng trong một tập đối tượng

Vòng lặp (for)

```
val items = listOf("apple", "banana", "kiwifruit")

//Duyệt danh sách sản phẩm
for (item in items) {
    println(item)
}

//Duyệt theo vị trí
for (index in items.indices) {
    println("item at $index is ${items[index]}")
}

//Duyệt vừa lấy vị trí, vừa lấy giá trị
for ((index, item) in items.withIndex()) {
    println("item at $index is $item")
}
```

Vòng lặp (while, do..while)





Vòng lặp (while, do..while)

```
val items = listOf("apple", "banana", "kiwifruit")

var index = 0
while (index < items.size) {
    println("item at $index is ${items[index]}")
    index++
}

var toComplete: Boolean
do {
    ...
    toComplete = ...
} while(toComplete)
```

The condition variable can be initialized inside to the `do…while` loop.

Vòng lặp (break và continue)

There are `break` and `continue` labels for loops:

```
myLabel@ for (item in items) {
    for (anotherItem in otherItems) {
        if (...) break@myLabel
        else continue@myLabel
    }
}
```

Ranges

```
val x = 10
if (x in 1..10) {
    println("fits in range")
}

for (x in 1..5) {
    print(x)
}

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

`downTo` and `step` are extension functions, not keywords.
`'..'` is actually `T.rangeTo(that: T)`

Null safety

```
val notNullText: String = "Definitely not null"
val nullableText1: String? = "Might be null"
val nullableText2: String? = null

fun funny(text: String?) {
            if (text != null)
                        println(text)
            else
                        println("Nothing to print :(")
}

fun funnier(text: String?) {
            val toPrint = text ?: "Nothing to print :("
            println(toPrint)
}
```

Elvis operator ?:

Nếu biểu thức bên trái `?:` không phải là `null`, thì toán tử Elvis sẽ return về chính nó; ngược lại sẽ trả về biểu thức bên phải.

Lưu ý: Biểu thức ở bên phải được sử dụng khi biểu thức ở bên trái `null`.

```
fun loadInfoById(id: String): String? {
        val item = findItem(id) ?: return null
        return item.loadInfo() ?: throw Exception("...")
}
```

Safe Calls

`someThing?.otherThing` does not throw an NPE if `someThing` is `null`.

Safe calls khá hữu ích trong việc xử lý chuỗi. Ví dụ: Một nhân viên có thể làm việc ở một (hoặc không) phòng ban. Trong phòng ban sẽ có một nhân viên làm trưởng phòng (hoặc không)

```
fun printDepartmentHead(employee: Employee) {
            println(employee.department?.head?.name)
}
```

Nếu chỉ muốn lấy ra các giá trị khác null, có thể sử dụng safe calls với `let`:

```
employee.department?.head?.name?.let { println(it) }
```

Unsafe Calls

The not-null assertion operator (`!!`) converts any value to a non-null type and throws an NPE exception if the value is null.

```
fun printDepartmentHead(employee: Employee) {
        println(employee.department!!.head!!.name!!)
}
```

TRÁNH SỬ DỤNG Unsafe Calls

TODO

Always throws a NotImplementedError at **run-time** if called, stating that operation is not implemented.

```
// Throws an error at run-time if calls this function, but compiles
fun findItemOrNull(id: String): Item? = TODO("Find item $id")


// Does not compile at all
fun findItemOrNull(id: String): Item? = { }
```

String templates và the string builder

```
val i = 10
val s = "Kotlin"

println("i = $i")
println("Length of $s is ${s.length}")

val sb = StringBuilder()
sb.append("Hello")
sb.append(", world!")
println(sb.toString())
```

Biểu thức Lambda

```
val sum: (Int, Int) -> Int = { x: Int, y: Int -> x + y }
val mul = { x: Int, y: Int -> x * y }
```

According to Kotlin convention, if the last parameter of a function is a function, then a lambda expression passed as the corresponding argument can be placed outside the parentheses:

```
val badProduct = items.fold(1, { acc, e -> acc * e })

val goodProduct = items.fold(1) { acc, e -> acc * e }
```

If the lambda is the only argument, the parentheses can be omitted entirely (the documentation calls this feature "trailing lambda as a parameter"):

```
run({ println("Not Cool") })
run { println("Very Cool") }
```

Tài liệu tham khảo

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

Thanks!