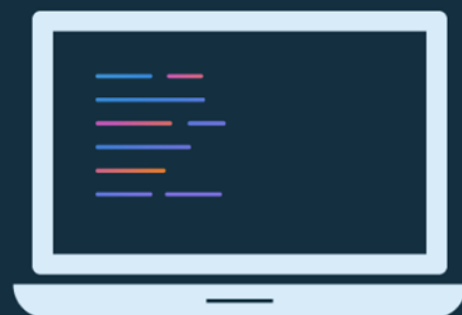




Bài học 2: Hàm



Giới thiệu về bài học này

Bài học 2: Hàm

- [Chương trình trong Kotlin](#)
- [\(Hầu hết\) Mọi thứ đều có giá trị](#)
- [Hàm trong Kotlin](#)
- [Hàm thu gọn](#)
- [Hàm lambda và các hàm bậc cao hơn](#)
- [Bộ lọc danh sách](#)
- [Tóm tắt](#)

Chương trình trong Kotlin

Thiết lập

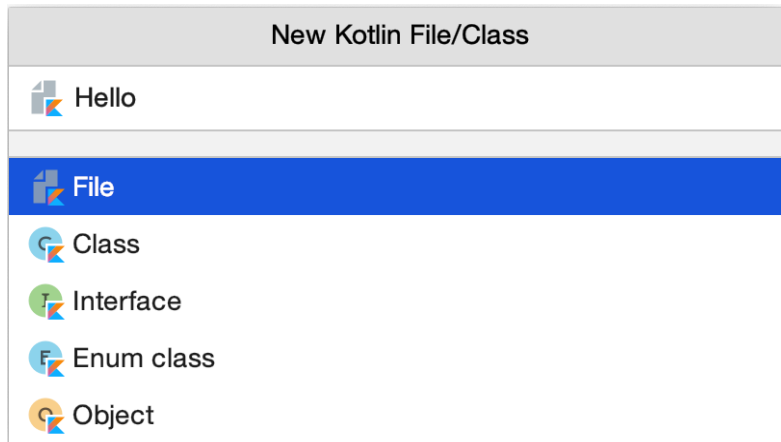
Trước khi có thể viết mã và chạy chương trình, bạn cần phải:

- Tạo một tệp trong dự án
- Tạo một hàm `main()`
- Chuyển các đối số cho hàm `main()` (Không bắt buộc)
- Dùng các đối số đã chuyển trong lệnh gọi hàm (Không bắt buộc)
- Chạy chương trình

Tạo một tệp mới trong Kotlin

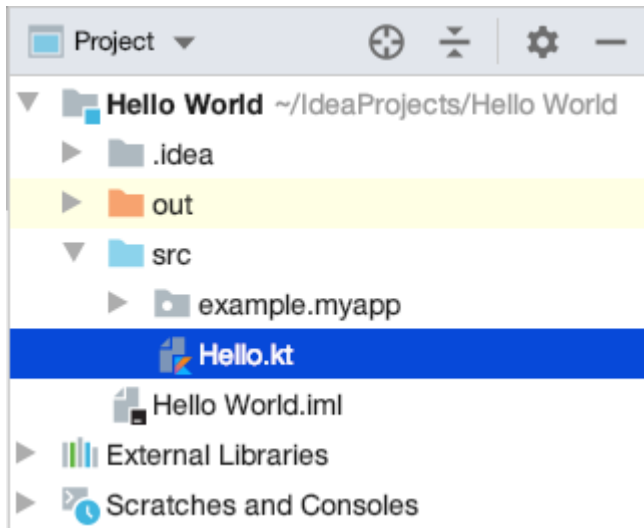
Trong ngăn Dự án của IntelliJ IDEA, bên dưới mục **Hello World**, hãy nhấp chuột phải vào thư mục `src`.

- Chọn **New > Kotlin File/Class** (Mới > Tập/lớp trong Kotlin).
- Chọn **File** (Tập), đặt tên cho tệp là `Hello` rồi nhấn phím **Enter**.



Tạo một tệp trong Kotlin

Bây giờ, bạn sẽ thấy một tệp trong thư mục `src` có tên là `Hello.kt`.



Tạo một hàm main()

`main()` là điểm bắt đầu để thực thi một chương trình trong Kotlin.

Trong tệp `Hello.kt`:

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

Các đối số trong hàm `main()` là không bắt buộc.

Chạy chương trình trong Kotlin

Để chạy chương trình, hãy nhấp vào biểu tượng Chạy (▶) ở bên trái hàm `main()`.

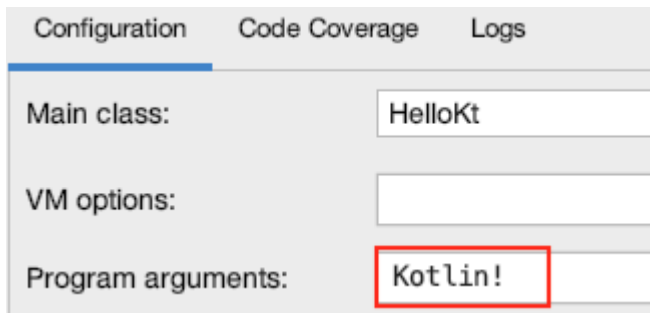
```
1 ▶ fun main(args: Array<String>) {  
2     println("Hello, world!")  
3 }  
4
```

IntelliJ IDEA sẽ chạy chương trình và hiển thị các kết quả trong bảng điều khiển.

```
HelloKt x  
/Library/Java/JavaVirtualMachines/jdk-13.0.2.jdk/Contents/Home/bin/java  
Hello, world!  
  
Process finished with exit code 0
```


Chuyển các đối số cho hàm main()

Chọn **Run > Edit Configurations** (Chạy > Chỉnh sửa cấu hình) để mở cửa sổ **Run/Debug Configurations** (Cấu hình chạy/gỡ lỗi).



Dùng các đối số trong hàm main()

Dùng `args[0]` để truy cập vào đối số đầu vào đầu tiên được chuyển cho hàm `main()`.

```
fun main(args: Array<String>) {  
    println("Hello, ${args[0]}")  
}
```

⇒ Hello, Kotlin!

(Hầu hết) Mọi thứ đều có giá trị



(Hầu hết) Mọi thứ đều ở dạng biểu thức

Trong Kotlin, hầu hết mọi thứ đều ở dạng biểu thức và có giá trị. Ngay cả biểu thức `if` cũng có giá trị.

```
val temperature = 20
```

```
val isHot = if (temperature > 40) true else false
```

```
println(isHot)
```

```
⇒ false
```

Giá trị biểu thức

Đôi khi, giá trị đó là `kotlin.Unit`.

```
val isUnit = println("This is an expression")  
println(isUnit)
```

⇒ This is an expression
kotlin.Unit

Hàm trong Kotlin

Giới thiệu về hàm

- Là một khối mã thực hiện tác vụ cụ thể
- Chia một chương trình lớn thành các phân đoạn mô-đun nhỏ hơn
- Được khai báo bằng từ khóa `fun`
- Có thể nhận các đối số có giá trị mặc định hoặc giá trị được đặt tên

Các thành phần của hàm

Trước đó, bạn đã tạo một hàm đơn giản để in "Hello World".

```
fun printHello() {  
    println("Hello World")  
}  
  
printHello()
```


Hàm trả về giá trị Unit

Nếu một hàm không trả về giá trị hữu ích nào, loại dữ liệu trả về sẽ là `Unit`.

```
fun printHello(name: String?): Unit {  
    println("Hi there!")  
}
```

`Unit` là loại dữ liệu chỉ có một giá trị: `Unit`.

Hàm trả về giá trị Unit

Không bắt buộc phải khai báo loại dữ liệu trả về là `Unit`.

```
fun printHello(name: String?): Unit {  
    println("Hi there!")  
}
```

tương đương với:

```
fun printHello(name: String?) {  
    println("Hi there!")  
}
```

Đổi số của hàm

Các hàm có thể chứa:

- Tham số mặc định
- Tham số bắt buộc
- Đổi số được đặt tên

Tham số mặc định

Các giá trị mặc định sẽ cung cấp giá trị dự phòng nếu không có giá trị tham số nào được chuyển.

```
fun drive(speed: String = "fast") {  
    println("driving $speed")  
}
```

Dùng "=" tiếp sau loại để xác định các giá trị mặc định

`drive()` ⇒ `driving fast`


`drive("slow")` ⇒ `driving slow`

`drive(speed = "turtle-like")` ⇒ `driving turtle-like`

Tham số bắt buộc

Nếu không chỉ định giá trị mặc định cho tham số, bạn sẽ phải cung cấp đối số tương ứng.

Tham số bắt buộc




```
fun tempToday(day: String, temp: Int) {  
    println("Today is $day and it's $temp degrees.")  
}
```

Tham số mặc định và tham số bắt buộc

Các hàm có thể dùng kết hợp tham số mặc định và tham số bắt buộc.

```
fun reformat(str: String,  
            divideByCamelHumps: Boolean,  
            wordSeparator: Char,  
            normalizeCase: Boolean = true){
```

 Có giá trị mặc định

Chuyển các đối số bắt buộc.

```
reformat("Today is a day like no other day", false, '_')
```

Đổi số được đặt tên

Để cải thiện khả năng đọc, hãy dùng các đổi số được đặt tên cho đổi số bắt buộc.

```
reformat(str, divideByCamelHumps = false, wordSeparator = '_')
```

Bạn nên đặt đổi số mặc định tiếp sau đổi số vị trí, bằng cách đó, phương thức gọi sẽ chỉ phải chỉ định các đổi số bắt buộc.

Hàm thu gọn

Hàm một biểu thức

Hàm thu gọn hoặc hàm một biểu thức giúp mã trở nên ngắn gọn và dễ đọc hơn.

```
fun double(x: Int): Int {  
    x * 2  
}
```

← Phiên bản đầy đủ

```
fun double(x: Int): Int = x * 2
```

← Phiên bản thu gọn

Hàm lambda và các hàm bậc cao hơn

Hàm Kotlin là lớp đầu tiên

- Hàm Kotlin có thể được lưu trữ ở dạng biến và dạng cấu trúc dữ liệu
- Các hàm này có thể được chuyển ở dạng đối số vào (và trả về từ) các hàm khác có bậc cao hơn
- Bạn có thể dùng các hàm bậc cao hơn để tạo hàm mới "tích hợp sẵn"

Hàm lambda

Hàm lambda là một biểu thức giúp tạo hàm không có tên.

Tham số và loại
Mũi tên hàm

```
var dirtLevel = 20  
val waterFilter = {level: Int -> level / 2}  
println(waterFilter(dirtLevel))  
⇒ 10
```

Mã cần thực thi

Cú pháp cho loại hàm

Cú pháp cho loại hàm của Kotlin có liên quan chặt chẽ với cú pháp cho hàm lambda. Khai báo một biến lưu giữ hàm.

```
val waterFilter: (Int) -> Int = {level -> level / 2}
```

Tên biến

Loại dữ liệu của biến
(loại hàm)

Hàm

Hàm bậc cao hơn

Hàm bậc cao hơn nhận các hàm ở dạng tham số hoặc trả về một hàm.

```
fun encodeMsg(msg: String, encode: (String) -> String): String {  
    return encode(msg)  
}
```

Phần nội dung mã sẽ gọi hàm đã được chuyển làm đối số thứ hai và chuyển đổi số đầu tiên cùng với đối số đó.

Hàm bậc cao hơn

Để gọi hàm này, hãy chuyển hàm ở dạng chuỗi và dạng hàm.

```
val enc1: (String) -> String = { input -> input.toUpperCase() }  
println(encodeMsg("abc", enc1))
```

Khi dùng loại hàm, việc triển khai sẽ tách biệt với việc sử dụng.

Chuyển tham chiếu hàm

Dùng toán tử `::` để chuyển một hàm được đặt tên ở dạng đối số cho một hàm khác.

```
fun enc2(input:String): String = input.reversed()
```

```
encodeMessage("abc", ::enc2)
```



Chuyển một hàm được đặt tên, không phải hàm lambda

Toán tử `::` sẽ cho Kotlin biết rằng bạn đang chuyển tham chiếu hàm ở dạng đối số, chứ không phải đang cố gọi hàm.

Cú pháp lệnh gọi tham số cuối cùng

Kotlin muốn rằng bất kỳ tham số nào nhận một hàm đều là tham số cuối cùng.

```
encodeMessage("acronym", { input -> input.toUpperCase() })
```

Bạn có thể chuyển hàm lambda ở dạng tham số hàm mà không cần đưa hàm này vào trong dấu ngoặc đơn.

```
encodeMsg("acronym") { input -> input.toUpperCase() }
```

Dùng các hàm bậc cao hơn

Nhiều hàm tích hợp sẵn của Kotlin được xác định bằng cú pháp lệnh gọi tham số cuối cùng.

```
inline fun repeat(times: Int, action: (Int) -> Unit)

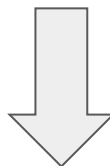
repeat(3) {
    println("Hello")
}
```

Bộ lọc danh sách

Bộ lọc danh sách

Lấy một phần danh sách dựa trên điều kiện nào đó

đỏ	đỏ cam	đỏ đậm	cam	cam sáng	vàng nghệ
----	--------	--------	-----	----------	-----------



Áp dụng `filter()` cho danh sách
Điều kiện: thành phần chứa chữ “đỏ”

đỏ	đỏ cam	đỏ đậm
----	--------	--------

Lặp lại theo danh sách

Nếu hằng hàm chỉ có một tham số, bạn có thể loại bỏ phần khai báo và "->". Tham số sẽ được khai báo ngầm bằng tên `it`.

```
val ints = listOf(1, 2, 3)
ints.filter { it > 0 }
```

Bộ lọc sẽ lặp qua một tập hợp, trong đó `it` là giá trị của thành phần trong quá trình lặp lại. Giá trị này tương đương với:

```
ints.filter { n: Int -> n > 0 } HOẶC ints.filter { n -> n > 0 }
```

Bộ lọc danh sách

Điều kiện lọc trong dấu ngoặc nhọn `{ }` sẽ kiểm tra từng mục khi bộ lọc lặp qua. Nếu biểu thức trả về `true`, mục sẽ được thêm vào.

```
val books = listOf("nature", "biology", "birds")  
println(books.filter { it[0] == 'b' })  
⇒ [biology, birds]
```

Bộ lọc eager và bộ lọc lazy

Đánh giá các biểu thức trong danh sách:

- **Eager:** diễn ra bất kể kết quả có được sử dụng hay không
- **Lazy:** chỉ diễn ra nếu cần vào thời gian chạy

Việc dùng phương thức đánh giá lazy cho các danh sách sẽ hữu ích nếu bạn không cần toàn bộ kết quả, hoặc nếu danh sách đặc biệt lớn và RAM không chứa được nhiều bản sao.

Bộ lọc eager

Các bộ lọc là eager theo mặc định. Một danh sách mới sẽ được tạo mỗi lần bạn dùng bộ lọc.

```
val instruments = listOf("viola", "cello", "violin")  
val eager = instruments.filter { it [0] == 'v' }  
println("eager: " + eager)  
⇒ eager: [viola, violin]
```


Bộ lọc lazy

Trình tự là các cấu trúc dữ liệu dùng phương thức đánh giá lazy, và có thể được dùng với các bộ lọc để chuyển thành lazy.

```
val instruments = listOf("viola", "cello", "violin")  
val filtered = instruments.asSequence().filter { it[0] == 'v' }  
println("filtered: " + filtered)  
  
⇒ filtered: kotlin.sequences.FilteringSequence@386cc1c4
```

Trình tự -> danh sách

Trình tự có thể được chuyển lại thành danh sách bằng `toList()`.

```
val filtered = instruments.asSequence().filter { it[0] == 'v' }
```

```
val newList = filtered.toList()
```

```
println("new list: " + newList)
```

```
⇒ new list: [viola, violin]
```

Các phép biến đổi danh sách khác

- `map()` thực hiện cùng một phép biến đổi trên mọi mục và trả về danh sách.

```
val numbers = setOf(1, 2, 3)
println(numbers.map { it * 3 })
=> [3, 6, 9]
```

- `flatten()` trả về một danh sách gồm tất cả các thành phần trong tập hợp lồng nhau.

```
val numberSets = listOf(setOf(1, 2, 3), setOf(4, 5), setOf(1, 2))
println(numberSets.flatten())
=> [1, 2, 3, 4, 5, 1, 2]
```

Tóm tắt



Tóm tắt

Trong Bài học 2, bạn đã tìm hiểu cách:

- Tạo tệp và hàm `main()` trong dự án, cũng như chạy chương trình
- Chuyển các đối số cho hàm `main()`
- Dùng giá trị được trả về của biểu thức
- Dùng các đối số mặc định để thay thế nhiều phiên bản của một hàm
- Dùng các hàm thu gọn để giúp mã dễ đọc hơn
- Dùng hàm lambda và các hàm bậc cao hơn
- Dùng các bộ lọc danh sách eager và lazy

Lộ trình

Thực hành những gì bạn đã học được bằng cách hoàn thành lộ trình này:

[Bài học 2: Hàm](#)



Câu hỏi 1

Các hàm trong Kotlin được tạo bằng từ khoá sau?

- A. void
- B. fun
- C. function
- D. def

Câu hỏi 2

Kiểu câu lệnh trả về mặc định của bất kỳ hàm nào được xác định trong Kotlin là gì?

- A. Int
- B. Double
- C. void
- D. Unit



Câu hỏi 3

Hàm in một dòng trong Kotlin là hàm nào?

- A. `println()`
- B. `printLine()`
- C. `linePrint()`
- D. `line()`

Câu hỏi 4

Câu lệnh "when" trong Kotlin có tương tự như câu lệnh trong các ngôn ngữ lập trình khác?

- A. if
- B. break
- C. switch
- D. continue

Câu hỏi 5

Lambdas trong Kotlin là gì?

- A. Lambda là một loại dữ liệu trong Kotlin
- B. Lambda là một loại lớp trong Kotlin
- C. Lambda là một giao diện trong Kotlin
- D. Lambda là một biểu thức mô tả hàm. Nhưng thay vì khai báo một hàm có tên, bạn sẽ khai báo hàm chưa có tên.