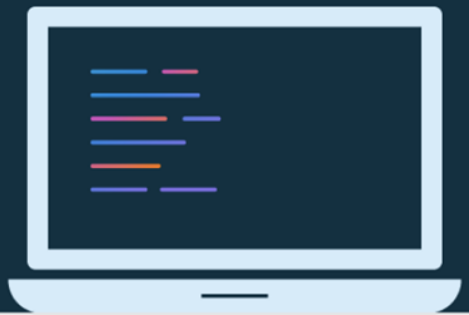
Bài học 1: Khái niệm cơ bản về Kotlin



Giới thiệu về bài học này

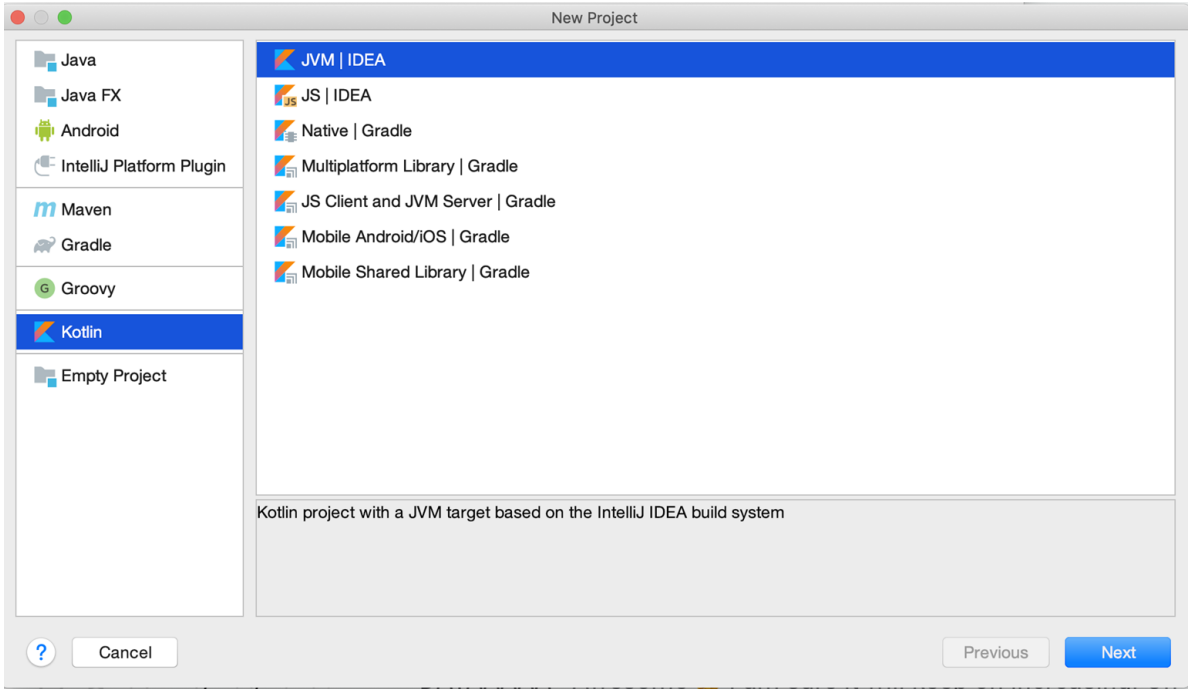
Bài học 1: Khái niệm cơ bản về Kotlin

- Bắt đầu
- Toán tử
- Loại dữ liệu
- Biến
- Điều kiện
- Danh sách và mảng
- Kiểm tra biến null an toàn
- Tóm tắt

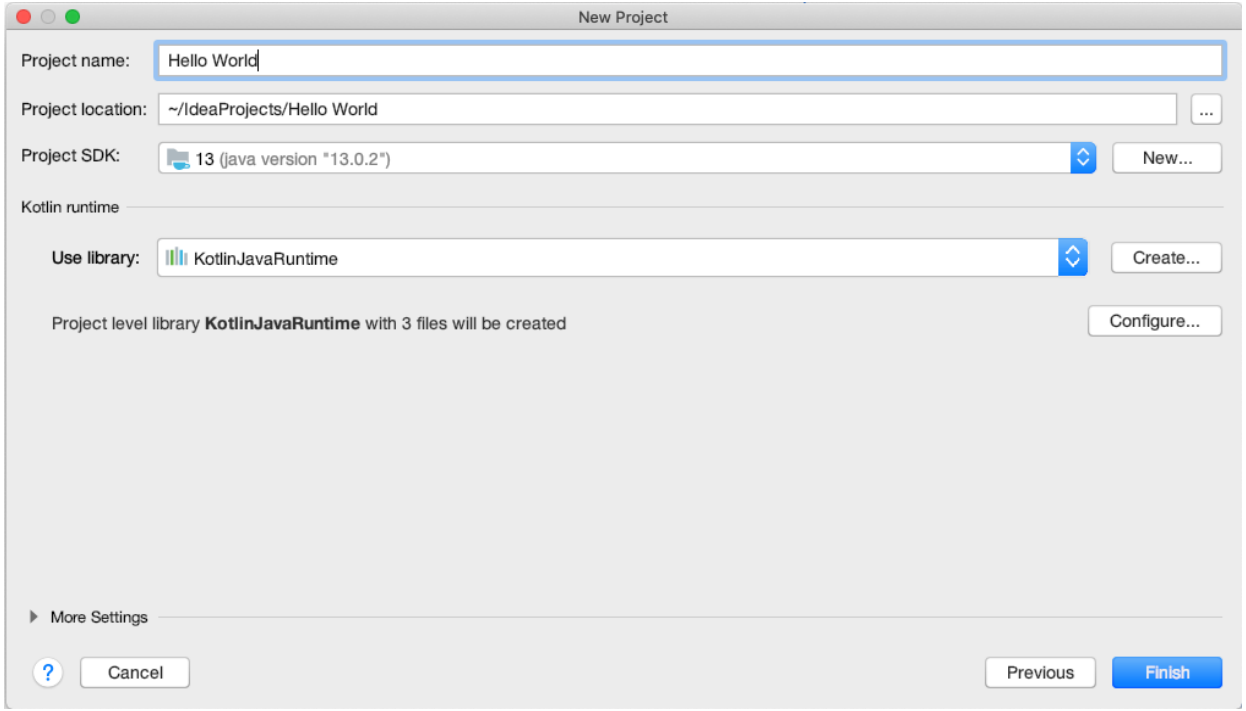
Bắt đầu

Mở IntelliJ IDEA

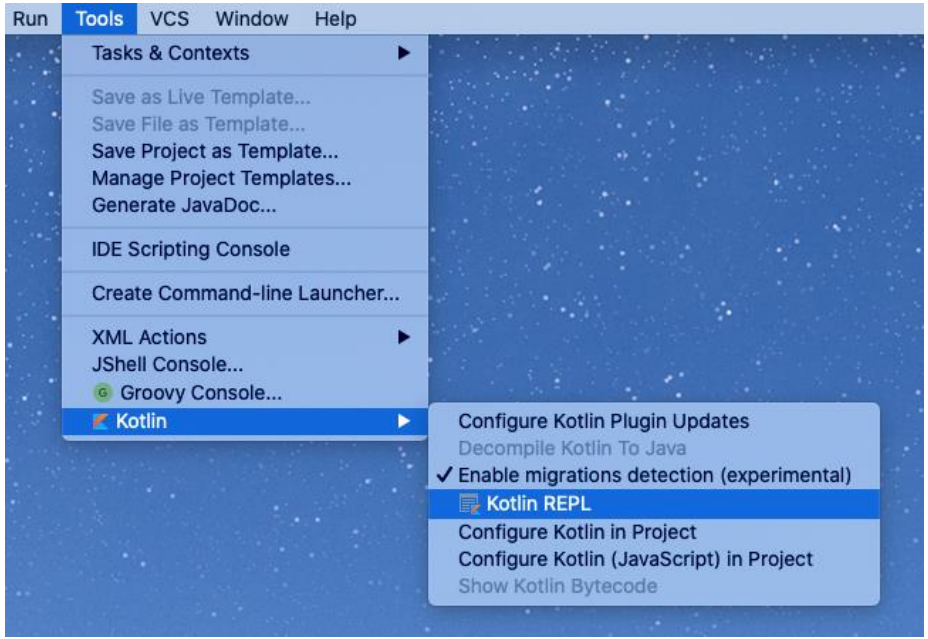
Tạo dự án mới



Đặt tên cho dự án

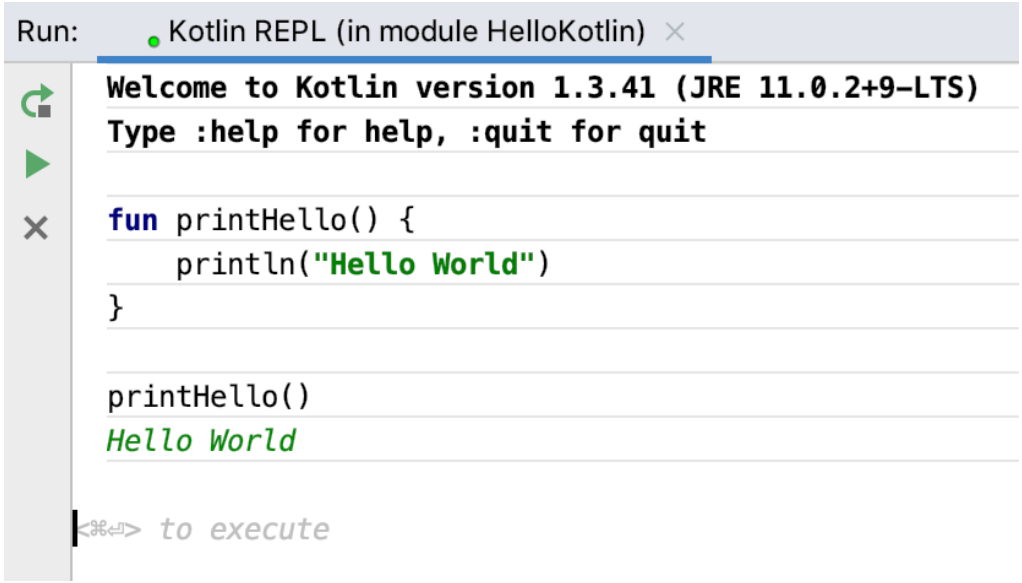


Mở REPL (Read-Eval-Print-Loop)



Trình đơn Kotlin có thể xuất hiện trong phần **Tools** (Công cụ) sau giây lát.

Tạo hàm printHello()



Nhấn tổ hợp phím **Control + Enter** (**Command + Enter** trên máy Mac) để thực thi.

Toán tử

Toán tử

- Toán tử toán học
- Toán tử tăng và toán tử giảm
- Toán tử so sánh
- Toán tử gán
- Toán tử bằng

```
+  -  *  /  %
++  --
<  <=   >  >=
=
==  !=
```

Toán tử toán học với số nguyên

```
1 + 1      =>      2

53 - 3     =>      50

50 / 10    =>      5

9 % 3      =>      0
```

Toán tử toán học với số thực

```
1.0 / 2.0   =>

2.0 * 3.5   =>
```

Toán tử toán học

```
1+1
⇒ kotlin.Int = 2
```

```
53-3
⇒ kotlin.Int = 50
```

```
50/10
⇒ kotlin.Int = 5
```

```
1.0/2.0
⇒ kotlin.Double = 0.5
```

```
2.0*3.5
⇒ kotlin.Double = 7.0
```

⇒ cho biết kết quả từ mã của bạn.

Kết quả bao gồm loại (**`kotlin.Int`**).

Phương thức toán tử số

Kotlin lưu giữ các số ở dạng nguyên thủy, nhưng cho phép bạn gọi phương thức cho các số đó như thể đó là đối tượng.

```
2.times(3)
⇒ kotlin.Int = 6

3.5.plus(4)
⇒ kotlin.Double = 7.5

2.4.div(2)
⇒ kotlin.Double = 1.2
```

Loại dữ liệu

Loại số nguyên

Loại	Bit	Ghi chú
Long	64	Từ -2^{63} đến $2^{63}-1$
Int	32	Từ -2^{31} đến $2^{31}-1$
Short	16	Từ -32768 đến 32767
Byte	8	Từ -128 đến 127

Dấu phẩy động và các loại số khác

Loại	Bit	Ghi chú
Double	64	16 – 17 chữ số có nghĩa
Float	32	6 – 7 chữ số có nghĩa
Char	16	Ký tự Unicode 16 bit
Boolean	8	Đúng hay sai. Phép toán bao gồm: || – tách từng phần, && – nối từng phần, ! – phủ định

Loại toán hạng

Kết quả của phép toán giữ nguyên loại toán hạng

```
6*50
⇒ kotlin.Int = 300
```

```
1/2
⇒ kotlin.Int = 0
```

```
6.0*50.0
⇒ kotlin.Double = 300.0
```

```
1.0*2.0
⇒ kotlin.Double = 0.5
```

```
6.0*50
⇒ kotlin.Double = 300.0
```

Chuyển đổi loại

Chỉ định `Int` cho `Byte`

```
val i: Int = 6
val b: Byte = i
println(b)
```

```
⇒ error: type mismatch: inferred type is Int but Byte
was expected
```

Chuyển đổi `Int` thành `Byte` bằng hàm chuyển đổi loại

```
val i: Int = 6
println(i.toByte())
```

```
⇒ 6
```

Dấu gạch dưới cho số dài

Dùng dấu gạch dưới để giúp các hằng số dài dễ đọc hơn.

```
val oneMillion = 1_000_000

val idNumber = 999_99_9999L

val hexBytes = 0xFF_EC_DE_5E

val bytes = 0b11010010_01101001_10010100_10010010
```

Chuỗi

Chuỗi là bất kỳ chuỗi ký tự nào được đưa vào dấu ngoặc kép.

```
val s1 = "Hello world!"
```

Hằng chuỗi có thể chứa ký tự thoát

```
val s2 = "Hello world!\n"
```

Hoặc bất kỳ văn bản tùy ý nào được phân tách bằng 3 dấu nháy (""")

```
val text = """
    var bikes = 50
"""
```

Ghép chuỗi

```
val numberOfDogs = 3
val numberOfCats = 2

"I have $numberOfDogs dogs" + " and $numberOfCats cats"

=> I have 3 dogs and 2 cats
```

Mẫu chuỗi

Biểu thức mẫu sẽ bắt đầu bằng ký hiệu đô la ($) và có thể là một giá trị đơn giản:

```
val i = 10
println("i = $i")
=> i = 10
```

Hoặc là một biểu thức bên trong dấu ngoặc nhọn:

```
val s = "abc"
println("$s.length is ${s.length}")
=> abc.length is 3
```

Biểu thức mẫu chuỗi

```
val numberOfShirts = 10
val numberOfPants = 5

"I have ${numberOfShirts + numberOfPants} items of clothing"

=> I have 15 items of clothing
```

Biến

Biến

- Khả năng dự đoán loại mạnh mẽ
  - Để trình biên dịch dự đoán loại
  - Bạn có thể khai báo rõ loại nếu cần

- Biến có thể thay đổi và biến không thể thay đổi
  - Tính bất biến không được thực thi nhưng lại được khuyến nghị

Kotlin là một ngôn ngữ loại tĩnh. Loại được phân giải lúc biên dịch và không bao giờ thay đổi.

Chỉ định loại biến

Ký hiệu dấu hai chấm

```
var width: Int = 12
var length: Double = 2.5
```

Lưu ý quan trọng: Sau khi bạn hoặc trình biên dịch chỉ định một loại, bạn không thể thay đổi loại đó, nếu không sẽ gặp lỗi.

Biến có thể thay đổi và biến không thể thay đổi

- Có thể thay đổi (Dễ thay đổi)

```
var score = 10
```

- Không thể thay đổi (Không thay đổi)

```
val name = "Jennifer"
```

Mặc dù không được thực thi nghiêm ngặt nhưng bạn vẫn nên dùng các biến không thể thay đổi trong hầu hết trường hợp.

var và val

```
var count = 1
count = 2

val size = 1
size = 2

=> Error: val cannot be reassigned
```

Điều kiện

Luồng điều khiển

Kotlin có một số cách để triển khai logic điều kiện:

- Câu lệnh If/Else
- Câu lệnh When
- Vòng lặp For
- Vòng lặp While

Câu lệnh if/else

```
val numberOfCups = 30
val numberOfPlates = 50

if (numberOfCups > numberOfPlates) {
    println("Too many cups!")
} else {
    println("Not enough cups!")
}
```

```
=> Not enough cups!
```

Câu lệnh if với nhiều trường hợp

```
val guests = 30
if (guests == 0) {
    println("No guests")
} else if (guests < 20) {
    println("Small group of people")
} else {
    println("Large group of people!")
}
```

⇒ Large group of people!

Phạm vi

- Loại dữ liệu chứa một khoảng giá trị so sánh (ví dụ: số nguyên từ 1 đến 100)
- Phạm vi được vạch ranh giới
- Các đối tượng trong một phạm vi có thể là dễ thay đổi hoặc không thể thay đổi

Phạm vi trong câu lệnh if/else

```
val numberOfStudents = 50
if (numberOfStudents in 1..100) {
    println(numberOfStudents)
}
=> 50
```

Lưu ý: Không có dấu cách quanh toán tử "range to" (phạm vi đến) `(1..100)`

Câu lệnh when

```
when (results) {
    0  -> println("No results")
    in 1..39 -> println("Got results!")
    else -> println("That's a lot of results!")
}
⇒ That's a lot of results!
```

Cũng như câu lệnh `when`, bạn cũng có thể xác định biểu thức `when` cung cấp giá trị trả về.

Vòng lặp for

```
val pets = arrayOf("dog", "cat", "canary")
for (element in pets) {
    print(element + " ")
}
⇒ dog cat canary
```

Bạn không cần xác định biến lặp và số gia cho mỗi lần chuyển.

Vòng lặp for: thành phần và chỉ mục

```
for ((index, element) in pets.withIndex()) {
    println("Item at $index is $element\n")
}
⇒ Item at 0 is dog
Item at 1 is cat
Item at 2 is canary
```

Vòng lặp for: kích cỡ bước và phạm vi

```
for (i in 1..5) print(i)
⇒ 12345

for (i in 5 downTo 1) print(i)
⇒ 54321

for (i in 3..6 step 2) print(i)
⇒ 35

for (i in 'd'..'g') print (i)
⇒ defg
```

Vòng lặp while

```
var bicycles = 0
while (bicycles < 50) {
    bicycles++
}
println("$bicycles bicycles in the bicycle rack\n")
⇒ 50 bicycles in the bicycle rack

do {
    bicycles--
} while (bicycles > 50)
println("$bicycles bicycles in the bicycle rack\n")
⇒ 49 bicycles in the bicycle rack
```

Vòng lặp repeat

```
repeat(2) {
    print("Hello!")
}
```

⇒ Hello!Hello!

Danh sách và mảng

Danh sách

- Danh sách là tập hợp các thành phần được sắp xếp theo thứ tự
- Các thành phần trong danh sách có thể được truy cập bằng cách lập trình thông qua các chỉ mục
- Các thành phần có thể xuất hiện nhiều lần trong một danh sách

Một ví dụ về danh sách là câu: đó là một nhóm từ, thứ tự các từ rất quan trọng và có thể lặp lại.

Danh sách không thể thay đổi sử dụng listOf()

Khai báo một danh sách bằng `listOf()` và in ra.

```
val instruments = listOf("trumpet", "piano", "violin")
println(instruments)
```

```
⇒ [trumpet, piano, violin]
```

Danh sách có thể thay đổi sử dụng mutableListOf()

Bạn có thể thay đổi các danh sách bằng `mutableListOf()`

```
val myList = mutableListOf("trumpet", "piano", "violin")
myList.remove("violin")

⇒ kotlin.Boolean = true
```

Với danh sách được xác định bằng `val`, bạn không thay đổi được danh sách mà biến tham chiếu đến, nhưng vẫn có thể thay đổi nội dung danh sách.

Mảng

- Mảng lưu trữ nhiều mục
- Các thành phần mảng có thể được truy cập bằng cách lập trình thông qua các chỉ mục
- Các thành phần mảng ở dạng có thể thay đổi
- Kích thước mảng là cố định

Mảng sử dụng arrayOf()

Bạn có thể tạo một mảng chuỗi bằng `arrayOf()`

```
val pets = arrayOf("dog", "cat", "canary")
println(java.util.Arrays.toString(pets))
```

Với mảng được xác định bằng `val`, bạn không thay đổi được mảng mà biến tham chiếu đến, nhưng vẫn có thể thay đổi nội dung mảng.

Mảng có loại kết hợp hoặc loại đơn lẻ

Một mảng có thể chứa nhiều loại.

```
val mix = arrayOf("hats", 2)
```

Một mảng cũng có thể chỉ chứa một loại (với trường hợp là số nguyên).

```
val numbers = intArrayOf(1, 2, 3)
```

Kết hợp mảng

Dùng toán tử +.

```
val numbers = intArrayOf(1,2,3)
val numbers2 = intArrayOf(4,5,6)
val combined = numbers2 + numbers
println(Arrays.toString(combined))

=> [4, 5, 6, 1, 2, 3]
```

Kiểm tra biến null an toàn

Kiểm tra biến null an toàn

- Trong Kotlin, theo mặc định, các biến không thể có giá trị null
- Bạn có thể chỉ định rõ một biến thành null bằng toán tử lệnh gọi an toàn
- Cho phép các trường hợp ngoại lệ con trỏ null bằng toán tử `!!`
- Bạn có thể kiểm tra biến null bằng toán tử elvis (`?:`)

Các biến không thể có giá trị null

Trong Kotlin, theo mặc định, các biến `null` ở trạng thái không được phép.

Khai báo `Int` và chỉ định `null` cho biến đó.

```
var numberOfBooks: Int = null
⇒ error: null can not be a value of a non-null type Int
```

Toán tử lệnh gọi an toàn

Toán tử lệnh gọi an toàn (`?`), theo sau loại biểu thị rằng một biến có thể có giá trị `null`.

Khai báo `Int?` ở dạng có thể có giá trị null

```
var numberOfBooks: Int? = null
```

Nhìn chung, bạn đừng đặt biến thành giá trị null vì việc này có thể dẫn đến hậu quả không mong muốn.

Kiểm tra biến null

Kiểm tra xem biến `numberOfBooks` có phải không có giá trị `null` hay không. Sau đó, giảm dần biến đó.

```
var numberOfBooks = 6
if (numberOfBooks != null) {
    numberOfBooks = numberOfBooks.dec()
}
```

Bây giờ, hãy xem cách viết của Kotlin, sử dụng toán tử lệnh gọi an toàn.

```
var numberOfBooks = 6
numberOfBooks = numberOfBooks?.dec()
```

Toán tử !!

Nếu bạn biết chắc một biến sẽ không có giá trị null, hãy dùng `!!` để buộc biến đó thành loại không có giá trị null. Sau đó, bạn gọi phương thức/thuộc tính cho biến đó.

```
val len = s!!.length
```

gửi NullPointerException nếu có giá trị null

Cảnh báo: Vì `!!` sẽ gửi một ngoại lệ nên chỉ khuyên dùng toán tử này khi đặc biệt cần lưu giữ giá trị null.

Toán tử Elvis

Kiểm tra biến null toàn chuỗi bằng toán tử `?:`.

```
numberOfBooks = numberOfBooks?.dec() ?: 0
```

Toán tử `?:` đôi khi được gọi là "toán tử Elvis" vì toán tử này trông giống như mặt cười với mái tóc phồng ở bên, tương tự kiểu tóc của Elvis Presley.

Tóm tắt

Tóm tắt

Trong Bài học 1, bạn đã tìm hiểu cách:

- Tạo dự án IntelliJ IDEA, mở REPL và thực thi hàm
- Dùng các toán tử và phương thức toán tử số
- Dùng các loại dữ liệu, chuyển đổi loại, chuỗi và mẫu chuỗi
- Dùng các biến và khả năng dự đoán loại, cũng như các biến có thể thay đổi và biến không thể thay đổi
- Dùng các điều kiện, luồng điều khiển và cấu trúc vòng lặp
- Dùng các danh sách và mảng
- Dùng các tính năng kiểm tra biến null an toàn của Kotlin

Lộ trình

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

Bài học 1: Khái niệm cơ bản về Kotlin

Câu hỏi 1

Tệp Kotlin được lưu bằng đuôi tệp nào?

A. .java

B. .kot

C. .kt hoặc .kts

D. .android

Câu hỏi 2

Từ khoá nào dùng để xác định biến không thể thay đổi?

A. var

B. val

C. immulvar

D. immutableVar

Câu hỏi 3

Kotlin có khả năng tương tác với ngôn ngữ lập trình Java không?

A. Có, ngôn ngữ này có khả năng tương tác

B. Không, ngôn ngữ này không có khả năng tương tác

Câu hỏi 4

Lệnh nào sẽ trả về độ dài của chuỗi trong Kotlin?

A. str.length

B. length(str)

C. str.lengthOf

D. str.getLength()

Câu hỏi 5

Lựa chọn nào sau đây được dùng để viết mã null-safe trong Kotlin?

A. Dãy

B. Lớp kín

C. Toán tử Elvis

D. Hàm Lambda

Câu hỏi 6

Làm sao để tạo Arraylist trong Kotlin đúng cách?

A. val map = hashMapOf(1 to "one", 2 to "two", 3 to "three")

B. enum class Color {RED, GREEN, BLUE}

C. val set = hashSetOf(1, 2, 3)

D. val list = arrayListOf(1, 2, 3)