NGÔN NGỮ LẬP TRÌNH

Kotlin

Trang Thanh Trí
tritrang@ctu.edu.vn

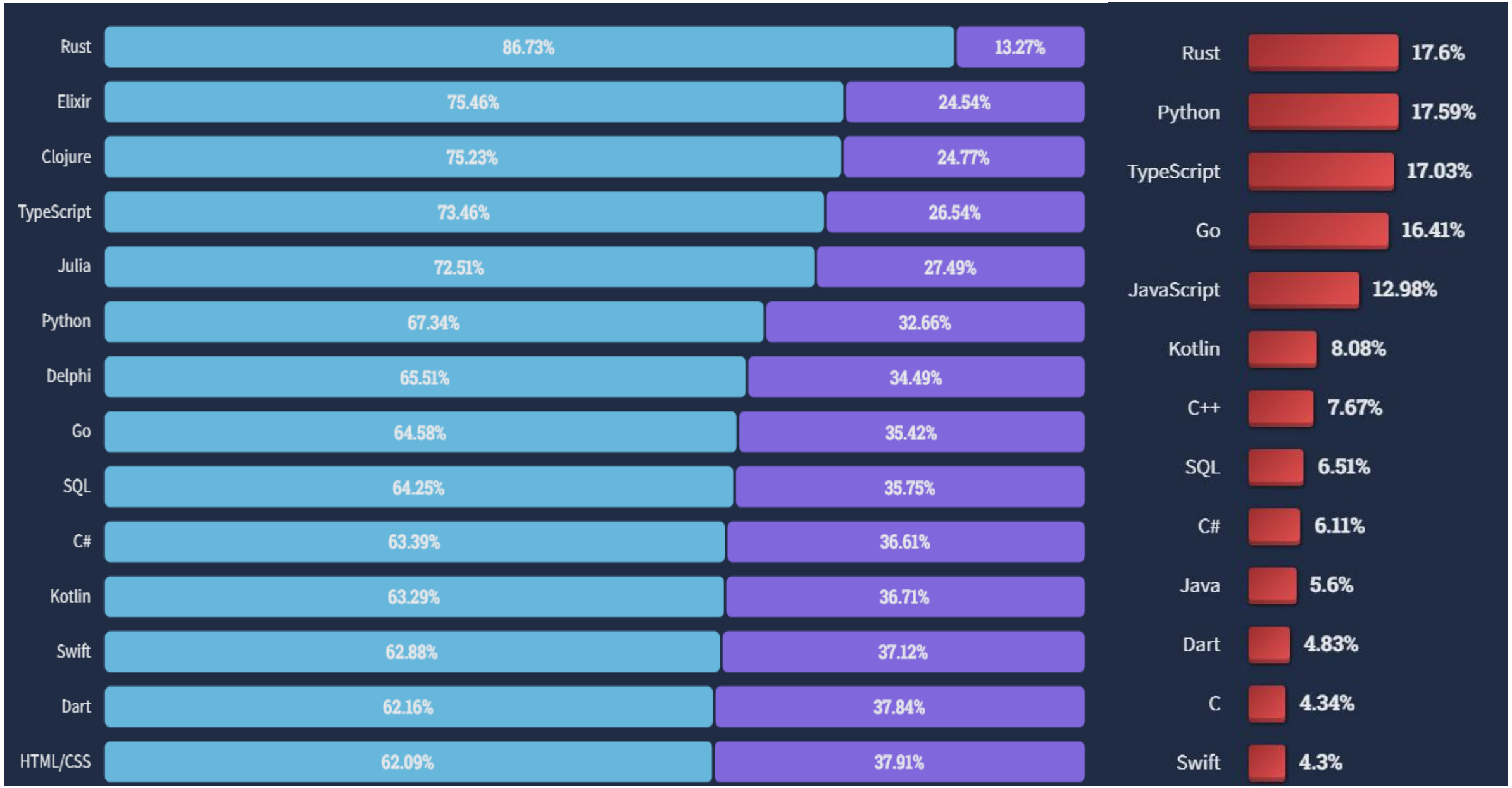
01-2022

Nội dung

- Giới thiệu
- Lập trình căn bản
- Các kiểu dữ liệu
- Lập trình hướng đối tượng
- Vào/ra, ngoại lệ
- Lập trình mạng
- Lập trình xử lý luồng

Giới thiệu

- **Kotlin** là một ngôn ngữ lập trình mã nguồn mở được phát triển bởi **JetBrains** (Cha đẻ của IDE đình đám **IntelliJ IDEA**), đăng ký theo Apache 2 license, miễn phí sử dụng.
- **Kotlin** là một ngôn ngữ lập trình kiểu tĩnh chạy trên máy ảo **Java** và có thể được biên dịch sang mã nguồn **Java** hay sử dụng cơ sở hạ tầng trình biên dịch LLVM. Đối với ứng dụng Android, chúng ta có thể sử dụng Kotlin thay thế cho mã Java hoặc sử dụng đồng thời cả hai trong cùng một project.
- Được sử dụng bởi các công ty lớn: Google, Uber, Nexflix, Trello, Pinterest,…
- Top 11 ngôn ngữ phổ biến được sử dụng 2022 (Python, Javascript, Swift, Go…)



https://survey.stackoverflow.co/2022/#most-loved-dreaded-and-wanted-language-want

Tính ứng dụng

- Lập trình đa nền tảng trên di động: KMM
- Server side: Spring, Ktor..
- Web Frontend: Kvision, Doodle…
- Android
- Tính toán khoa học
- Lập trình nhúng (Arduino/Raspberry…)

Lợi ích

1. Less code combined with greater readability.

2. Mature language and environment.

3. Kotlin support in Android Jetpack and other libraries. .

4. Interoperability with Java.

Lợi ích

5. Support for multiplatform development.

6. Code safety.

7. Easy learning.

8. Big community

Ưu điểm

- Concise
- Safe
- Expressive
- Interoperable
- Multiplatform

Ưu điểm

Concise

```
data class Employee(
    val name: String,
    val email: String,
    val company: String
) // + automatically generated equals(), hashCode(), toString(), and copy()

object MyCompany {                                      // A singleton
    const val name: String = "MyCompany"
}

fun main() {                                            // Function at the top
    val employee = Employee("Alice",                    // No `new` keyword
        "alice@mycompany.com", MyCompany.name)
    println(employee)
}
```

Safe

```
fun reply(condition: Boolean): String? =          // Nullability is part of Kotlin's type system
    if (condition) "I'm fine" else null

fun error(): Nothing =                            // Always throw an exception
    throw IllegalStateException("Shouldn't be here")

fun main() {
    val condition = true                          // Try replacing `true` with `false` and run the
sample!
    val message = reply(condition)                // The result is nullable
    // println(message.uppercase())               // This line doesn't compile
    println(message?.replace("fine", "okay"))     // Access a nullable value in a safe manner
    if (message != null) {                        // If you check that the type is right,
        println(message.uppercase())              // the compiler will smart-cast it for you
    }

    val nonNull: String =                            // If the null-case throws an error,
    reply(condition = true) ?: error()            // Kotlin can infer that the result is non-null
    println(nonNull)
}
```

Expressive

```
val map = mapOf(1 to "one", 2 to "two")
for ((k, v) in map) {                                  // Traverse a map or a list of pairs
    println("$k -> $v")
}

fun obtainKnowledge() = Pair("The Answer", 42)    // Single-expression functions

val (description, answer) = obtainKnowledge()     // Destructure into a pair of two variables
println("$description: $answer")

getText()?.let {                                       // Apply an action to a nullable expression
    sendEmailTo("alice@example.com", it)           // if it's not null
}

createEmptyWindow()
    .apply {                                      // Configure properties of an object
        width = 300
        height = 200
        isVisible = true
    }.also { w ->                                 // Perform an additional operation on a call
chain   showWindow(w)
    }

val fixedIssue = issueById["13456"]
?.takeIf { it.status == Status.FIXED }       // Use the value only if the condition is true
println(fixedIssue)
```

Interoperable

```
// Use any existing JVM library or framework
// Call Kotlin code from Java without an issue

@SpringBootApplication
class DemoApplication

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

@RestController
class MessageResource {
    @GetMapping
    fun index(): List<Message> = listOf(
        Message("1", "Hello!"),
        Message("2", "Bonjour!"),
        Message("3", "Privet!"),
    )
}

data class Message(val id: String?, val text: String)
```

Multiplatform

```
// Common
// Declare signatures to use them in the common code
// Provide platform-specific implementations in the platform modules
expect fun randomUUID(): String

expect class PlatformSocket(
        url: String
) {
    fun openSocket(listener: PlatformSocketListener)
    fun closeSocket(code: Int, reason: String)
    fun sendMessage(msg: String)
}

interface PlatformSocketListener {
    fun onOpen()
    fun onFailure(t: Throwable)
    fun onMessage(msg: String)
    fun onClosing(code: Int, reason: String)
}
```

Thực thi chương trình Kotlin

Soạn thảo chương trình: Website, Intellij IDEA, Android Studio...

Chương trình **hello.kt**

```
fun main() {
    print("Hello World!")
}
```

https://play.kotlinlang.org/

Keywords và toán tử trong Kotlin

1. Hard Keywords

2. Soft Keywords

3. Modifier Keywords

4. Special Identifier

5. Kotlin Identifiers

6. Quy ước đặt tên định danh Kotlin

Keywords và toán tử trong Kotlin

1. Hard Keywords: Những từ khóa này không thể được sử dụng làm định danh. (24 hard keywords)

as	for	null
as?	fun	object
break	if	package
class	in	return
continue	!in	super
do	interface	this
else	is	throw
false	!is	true

Keywords và toán tử trong Kotlin

2. Soft Keywords: Soft Keywords là những từ khóa chỉ được sử dụng trong một ngữ cảnh nhất định (17 soft keywords)

by	field	init	setparam
catch	file	param	where
constructor	finally	property	
delegate	get	receiver	
dynamic	import	set	

Keywords và toán tử trong Kotlin

3. Modifier Keywords: Modifier Keywords là những từ khóa chỉ được sử dụng trong một ngữ cảnh nhất định (17 modifier keywords)

actual	expect	noinline	reified
abstract	external	open	sealed
annotation	final	operator	suspend
companion	infix	out	tailrec
const	inline	override	vararg
crossinline	inner	private	
data	internal	protected	
enum	lateinit	public	

Keywords và toán tử trong Kotlin

4. Special Identifier: Có 2 định danh đặc biệt được định nghĩa bởi trình biên dịch trong ngữ cảnh nhất định

field	it

5. Kotlin Identifiers: Tên mà chúng ta đặt cho một variable, class, function

6. Quy ước đặt tên identifier Kotlin

1. identifier không thể có khoảng trắng.
2. identifier trong Kotlin phân biệt chữ hoa chữ thường.
3. Chúng không thể chứa các ký tự đặc biệt như @, #,%, v.v.
4. Một identifier có thể bắt đầu bằng một dấu gạch dưới
5. Cách tốt nhất để đặt cho các identifier là đặt tên có ý nghĩa. Ví dụ: add, multiply và divide là các định danh có ý nghĩa hơn a, m và d.
6. Nếu bạn muốn bao gồm hơn hai từ trong một identifier bạn có thể bắt đầu từ thứ hai bằng chữ in hoa. Ví dụ: sumOfTwo.

Các phép toán và kiểu dữ liệu cơ bản

Kiểu cơ bản: Int, Float, Double, Boolean, String, Char

Các phép toán số học: +, -, *, /, %

Phép toán so sánh: ==, !=, >, >=, <=

Phép toán luận lý: and (&&), or (||), not (!)

Kiểu dữ liệu cơ bản

1. Numbers
2. Booleans
3. Characters
4. Strings

Kiểu dữ liệu cơ bản | Numbers

Kiểu Integer

Type	Size	Min	Max
Byte	8	-128	127
Short	16	-32768	32767
Int	32	-2,147,483,648 (-2^31)	2,147,483,647 (2^31 - 1)
Long	64	-9,223,372,036,854,775,808 (-2^63)	9,223,372,036,854,775,807 (2^63 - 1)

Kiểu dữ liệu cơ bản | Booleans

Kiểu Boolean

The type Boolean represents boolean objects that can have two values: **true** and **false**.

Built-in operations on booleans include:: OR (**||**) , AND (**&&**), NOT (**!**)

```
fun main() {
//sampleStart
    val myTrue: Boolean = true
    val myFalse: Boolean = false
    val boolNull: Boolean? = null

    println(myTrue || myFalse)
    println(myTrue && myFalse)
    println(!myTrue)
//sampleEnd
}
```

Kiểu dữ liệu cơ bản | Characters

Characters are represented by the type Char. Character literals go in single quotes: '1'.

Special characters start from an escaping backslash \. The following escape sequences are supported: \t, \b, \n, \r, \', \", \\ and \$.

```
fun main() {
//sampleStart
    val aChar: Char = 'a'

    println(aChar)
    println('\n') //prints an extra newline character
    println('\uFF00')
//sampleEnd
}
```

Kiểu dữ liệu cơ bản | Strings

Strings in Kotlin are represented by the type String. Generally, a string value is a sequence of characters in double quotes (")

```
fun main() {
val str = "abcd"
//sampleStart
for (c in str) {
    println(c)
}
//sampleEnd
}
```

Kiểu dữ liệu cơ bản | Strings

A raw string is delimited by a triple quote (""""), contains no escaping and can contain newlines and any other characters:

```
val text = """
    for (c in "foo")
        print(c)
"""
```

Ép kiểu

toByte(): Byte

toShort(): Short

toInt(): Int

toLong(): Long

toFloat(): Float

toDouble(): Double

toChar(): Char

Chú thích trong kotlin

Sử dụng // để chú thích 1 dòng trong chương trình

```
// Đây là chú thích
```

Sử dụng /* */ để chú thích 1 đoạn

```
/*
Đây là chú thích 1 đoạn
*/
```

Lệnh print trong kotlin

Trong kotlin có 2 lệnh in: print và println

```kotlin
fun main() {
    // print prints its argument to the standard output.
    print("Hello ")
    print("world!")

    // println prints its arguments and adds a line break
    println("Hello world!")
    println(42)

}
```

Hàm trong kotlin

Ví dụ: Một hàm tính tổng 2 số có 2 tham số và trả về kiểu số nguyên

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

fun main() {
    print("sum of 3 and 5 is ")
    println(sum(3, 5))
}
```

Hàm trong kotlin

Ví dụ: Một hàm tính tổng 2 số có 2 tham số và trả về kiểu số nguyên (Viết theo dạng biểu thức)

```
//sampleStart
fun sum(a: Int, b: Int) = a + b
//sampleEnd

fun main() {
    println("sum of 19 and 23 is ${sum(19, 23)}")
}
```

Hàm trong kotlin

Ví dụ: Hàm không có kiểu trả về (Unit)

```
//sampleStart
fun printSum(a: Int, b: Int): Unit {
    println("sum of $a and $b is ${a + b}")
}
//sampleEnd

fun main() {
    printSum(-1, 8)
}
```

Khai báo biến trong kotlin

Sử dụng val để khai báo hằng số

```
fun main() {
//sampleStart
    val a: Int = 1  // immediate assignment
    val b = 2   // `Int` type is inferred
    val c: Int  // Type required when no initializer is provided
    c = 3       // deferred assignment
//sampleEnd
    println("a = $a, b = $b, c = $c")
}
```

Khai báo biến trong kotlin

Sử dụng var để biến có thể thay đổi

```
fun main() {
//sampleStart
    var x = 5 // `Int` type is inferred
    x += 1
//sampleEnd
    println("x = $x")
}
```

Khai báo lớp và khởi tạo

To define a class, use the class keyword.

```
class Shape
```

Properties of a class can be listed in its declaration or body.

```
class Rectangle(var height: Double, var length: Double) {
    var perimeter = (height + length) * 2
}
```

Khai báo lớp và khởi tạo

To define a class, use the class keyword.

```
class Shape
```

Properties of a class can be listed in its declaration or body.

```
class Rectangle(var height: Double, var length: Double) {
    var perimeter = (height + length) * 2
}
```

Khai báo lớp và khởi tạo

The default constructor with parameters listed in the class declaration is available automatically.

```
class Rectangle(var height: Double, var length: Double) {
    var perimeter = (height + length) * 2
}
fun main() {
//sampleStart
    val rectangle = Rectangle(5.0, 2.0)
    println("The perimeter is ${rectangle.perimeter}")
//sampleEnd
}
```

Khai báo lớp và khởi tạo

Kế thừa các lớp bằng ký tư (:). Mặc định các lớp là **final** (không thể kế thừa)

```
open class Shape

class Rectangle(var height: Double, var length: Double): Shape() {
    var perimeter = (height + length) * 2
}
```

String templates

```
fun main() {
//sampleStart
    var a = 1
    // simple name in template:
    val s1 = "a is $a"

    a = 2
    // arbitrary expression in template:
    val s2 = "${s1.replace("is", "was")}, but now is $a"
//sampleEnd
    println(s2)
}
```

Conditional expressions

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

fun main() {
    println("max of 0 and 42 is ${maxOf(0, 42)}")
}
```

Conditional expressions

Viết ngắn gọn

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

fun main() {
    println("max of 0 and 42 is ${maxOf(0, 42)}")
}
```

Lệnh for

```
fun main() {
//sampleStart
    val items = listOf("apple", "banana", "kiwifruit")
    for (item in items) {
        println(item)
    }
//sampleEnd
}
```

Lệnh for

```
fun main() {
//sampleStart
    val items = listOf("apple", "banana", "kiwifruit")
    for (index in items.indices) {
        println("item at $index is ${items[index]}")
    }
//sampleEnd
}
```

Lệnh while

```
fun main() {
//sampleStart
    val items = listOf("apple", "banana", "kiwifruit")
    var index = 0
    while (index < items.size) {
        println("item at $index is ${items[index]}")
        index++
    }
//sampleEnd
}
```

Lệnh when

```
//sampleStart
fun describe(obj: Any): String =
    when (obj) {
        1          -> "One"
        "Hello"    -> "Greeting"
        is Long    -> "Long"
        !is String -> "Not a string"
        else       -> "Unknown"
    }
//sampleEnd

fun main() {
    println(describe(1))
    println(describe("Hello"))
    println(describe(1000L))
    println(describe(2))
    println(describe("other"))
}
```

Ranges

Check if a number is within a range using in operator.

```
fun main() {
//sampleStart
    val x = 10
    val y = 9
    if (x in 1..y+1) {
        println("fits in range")
    }
//sampleEnd
}
```

Ranges

Check if a number is out of range.

```
fun main() {
//sampleStart
    val list = listOf("a", "b", "c")

    if (-1 !in 0..list.lastIndex) {
        println("-1 is out of range")
    }
    if (list.size !in list.indices) {
        println("list size is out of valid list indices range, too")
    }
//sampleEnd
}
```

Ranges

Iterate over a range.

```
fun main() {
//sampleStart
    for (x in 1..5) {
        print(x)
    }
//sampleEnd
}
```

Or over a progression.

```
fun main() {
//sampleStart
    for (x in 1..10 step 2) {
        print(x)
    }
    println()
    for (x in 9 downTo 0 step 3) {
        print(x)
    }
//sampleEnd
}
```

Bài tập trên lớp

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

Example:

- Input: nums = [-1,0,1,2,-1,-4]
- Output: [[-1,-1,2],[-1,0,1]]
- Explanation:
- nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
- nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
- nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
- The distinct triplets are [-1,0,1] and [-1,-1,2].
- Notice that the order of the output and the order of the triplets does not matter.

Bài tập trên lớp

Print all prime numbers from X to Y

Example:

- X = 10
- Y = 20
- Print : 11, 13, 17, 19