Kotlin for loop

While converting all my java code to kotlin, one of the unusual syntax changes I observed was the loop for loop in both languages. Later I realized in Kotlin, there are some concepts that are completely different from Java or any other language for loop. wait! They're not that hard. In fact, they are very easy, interesting and helpful. Let's check one by one. 1. Simple for loop in Java that iterates from some numbers to some incremental number one on each passing loop. String Java(int i = 0; i System.print(i); } equivalent code Kotlin(i in 0.10) { print(i) } there is no need to declare the type of variable if deleted on a range, we can use in the variable below and upper (including) limitations can be defined on both sides.. Operator. Java code(int j = 0; j &lt; 10; j ++{ System.out.print(j); / this will print only up to 9 } There are two ways to do the same in kotlin, the first one is to decrees the above it while coding and using ..operator or another way is used until operator. kotlin string(j in 0.9) { print(j) } for (j of 0 until 10) { print(j) } java code(int i = 0; i System.out.print(i); } We can use step operator here Kotlin codecode (i in 0.10 step 2) { print(i) } java codefo(int i = 10; i &gt; 0; i-) { System.out.print(i); } No, you cannot use 10..1 as .. operators never work on the reverse ranges. It won't give you a compile time or running time errors, but simply skip the loop by checking the conditions that will be false every time. You have to use downTo operators. Kotlin codecode (i in 10 downTo 1) { print(i) } You can also change the step size with step operator. string Java(int i = 10; i &gt; 0; i -= 3) { System.out.print(i); } Kotlin codecode (i'm in 10 downTo step 3) { print(i) } but please note that until operator doesn't work here. until operators can only be used forward increments. string java(int k = 2; k System.out.print(k); } Move in while loop, there is no other way Kotlin kodevar k = 2while(k) k * 2 } Kodinint Java[] arr = new int[5]; for (int i = 0; i &lt; arr.length; i ++) { System.out.print(arr[i]) } simple, we indicate they are in kotlin Kotlin stringval arr = IntArray(5)for (i in arr.indices) { print(ar[i]] I heard about some forech too. Can I use the same in kotlin? java kodint[] arr = new int[5];p on(int item: arr) { System.out.print(item); } Yes, you can. to loop iterate through anything that provides an iterateur. A to loop on a compile array of an index loop is based on that by creating an iterator object. Kotlin comfort(item in ar) { print(item) } Java CodeList arrayList = new ArrayList&lt;&gt;(); for (int i = 0; i &lt; arrayList.size(); i++) { System.out.print(arrayList.get(i) } list vector = new Vector&lt;&gt;(); for (int i = 0; i&lt; vector.size(); i++) { (vector.get(i)); } simple. Use indexes based on iteration. Kotlin kodval arrayList = ArrayList() for (i in opacity { print(arrayList[i]) } val vector = Vector() for (i in vector.indices) { print(vector[i]) } name, I'm a fan of foreach loop. String java(int item : arrayList) { System.out.print(item); } for (int item: vector) { System.out.print(item); } Ok, no problem, there you go. Kotlin codecode(item in arrayList) { print(item) } for (item in vector) { print(item) } Kotlin codefo((i, value) in arr.inIndex()) { print(the $i element is $value) } usually you don't need the Inindex function for iteration. And we'll do it. Watching me told you this will be very easy and interesting. Reference: Kotlin docs The for towns in Kotlin used to iterate or sugar the elements in array, ranges, collections etc. In this guide, we will learn how to use towns in Kotlin with the help of various examples. A simple example of for loop through Kotlin in this example we are deleting even if an integer range is using for loop. /*** created by Chaitanya for Beginnersbook.com */ package beginnersbook fun main(args : Array&lt;String&gt;) { for(n in 10..15) { $n print } } output: kotlin for loop using In the following example see have have have array myArray and see are displaying the elements of the array using for loop. package beginnersbook fun main(args : Array&lt;String&gt;) { val myArray = arrayOf(ab, bc, cd, da) for (str in myArray) { println(str) } output: ab bc cd da kotlin for loop iterating though array indices We can also use array indexes to iterate the array. /*** created by Chaitanya for Beginnersbook.com */ package beginnersbook fun main(args : Array&lt;String&gt;) { val myArray = arrayOf(steve, Robin, Kate, Lucy) for (n in myArray.indices) { println(myArray[$n]: ${myArray[n]}) } } output: Function withIndex() in for loop In the above example see have have haverated the array using array indices. Another way to do the same is with the use of InIndex() function. pack debutan main fun (args:&lt;String&gt;Array) { val MyArray = arrayOf(steve, Robin, Kate, Lucy) for (index, value) of MyArray.inIndex()) { print(Value in index $index is: $value) } Output: Value of Index 0 is: Steve Value in Index 1 is: Robin Value in Index 2 is: Kate Value in Index 3 is: Lucy Edit Page In Kolin, if that is an expression, it means it returns a value. That's why there are no terrain operators (conditions? Then: other things), because ordinary if works fine in this role. Traditional l'var max = a if(a&lt;b)=max=b with=else=var= max:=int=if= (a=&gt;b) { max =&lt;/String&gt;&lt;/String&gt;&lt;/String&gt; &lt;/String&gt; another { other { max = b } / As Expression val max = if (a&gt;b) else branch if branch can block, and the last expression is the value of a block: max val = if (a &gt; b) { print(Select a) else { print (Select b) b } if you are using if a rather than a statement (for example, returning its value or assigning it to a variable), the expression is required to have another branch. branch. the grammar for if. When the expression replaces the switch statement in C-like languages. In simplest form it looks this time (x) { 1 -&gt; print(x = 1) 2 -&gt; print (x = 2) other things -&gt; { / Do not open print block (x is neither 1 nor 2) } when matches its arguments against all sequential branches until some branch requirements are met. If it is used as an expression, the value of the satisfying branch becomes the value of the overall expression. If they use it as a statement, individual branch values are ignored. (Just like and if, each branch can be a block, and its value is the value of the last expression in the block.) The other branch is evaluated if none of the other branch requirements are met. If used as an expression, the other branch is required, unless the compilator can prove that all possible cases are covered with branch requirements (such as, for example, and enum class entries and subtle sealed classes). If many might be handled in the same way, branch conditions can be combined with a comma: when (x) { 0, 1 -&gt; print(x = 0 or x = 1) other things -&gt; print(otherwise) } We can use arbitrary expressions (not only constant) as branch condition when (x) { parseInt(s)-&gt; Print(s encoding x) other things -&gt; print(s not encoded x) } We can also check a value for being in or !in!in a row or a collection: when (x) { in 1.10 -&gt; print (x is in the range) in validNumbers -&gt; print (x is valid) ! ! 20 -&gt; print (x is outside the range) else -&gt; print (None of above) } Another possibility is to check that a value is or !is of a particular type. Note that, due to smart disposal, you can access the methods and properties of the type without any extra checks. fun hasPrefix(x: Any) = when (x) { is String -&gt; x.startsWith(prefix) else -&gt; false } when they can also be used as a replacement for an if-else if channel. If no arguments are supplied, branch conditions are merely boolean expressions, and a branch is executed when its condition is true: when { x.isOdd() -&gt; print(x is odd) y.isEven() -&gt; print(y is the same) else -&gt; print (x + they are the same. } Since Kotlin 1.3, it is possible to take when subject to a variable using syntax: Pleasure Request.getBody() = when (val response = executeRe()) { is successful -&gt; response.body is HttpError -&gt; throw HttpException(response.status) } Scope of variable, introduced in when subject, restriction on when body. See the grammar for when. to loop iterate through anything that provides an iterateur. This is equivalent to the foreach loop of languages such as C#. The syntax is as follows: for (items in collections) to print (items) the body can be a block. for (item: int ints) { / / ... As mentioned before, for iterates through everything that provides an iterator, this means there is a member- or extension-function iterator(), which type has a member- or extension-function next(), and has a member- or extension -function contains Next() that returns Boolean. All these three functions need to be marked as operators. To iterate over a range of numbers, use a row expression: main pleasure() { / sampleStart for (i in 1.3) { print(i) } for (i in 6 down step 0 2) { print(i) } / sample } A for loop over a row or a compile array of an index loop based on non-creating an iteraterator object. If you want to iterate through an array or a list with an index, you can do it this way: main fun() { array = array = array(a, b, c)/sampleStart for (i in array.indices) { print(array[i]) } /sampleEnd } Alternatively, you can use the Index library function: fun main() { array = array = array(, a, b, c)/sampleStart for (index, value) in array.inIndex()) { print(element in $index is $value) } / sample } See the grammar for. while and done.. while working as usual while (x &gt; 0) { x-} do {val y = retrieveData() } while (y != null) / they are visible here! See the grammar for forever. Kotlin supports traditional breaking and continuing operators in towns. See Back and jump. skip.