External vs. Internal Iterators

We'll cover the following

Programmers with experience in Java and other C-like languages are used to external iterators. They are common but complex; we're very familiar with them, but they involve many moving parts. Internal iterators are less familiar to a lot of programmers, but they are less complex than external iterators.

External iterators

External iterators in Kotlin are already an improvement when compared to external iterators in Java, but even then, the internal iterators have much less ceremony, are more concise, and are expressive when compared to Kotlin's external iterators. Let's compare Kotlin's external iterators with internal iterators using some examples.

Here's a list of numbers stored in a `numbers` variable:

```
// iterate.kts
val numbers = listOf(10, 12, 15, 17, 18, 19)
```

To print only the even numbers in this list we can use the `for...in` loop of Kotlin, like so:

```
for (i in numbers) {
    if (i % 2 == 0) {
        print("$i, ") //10, 12, 18,
    }
}
```

iterate.kts

As the iteration progresses, the variable `i` holds different values from the collection, one per iteration. The `if` expression, used as a statement here, checks if the value currently held in `i` is even, and if it is, we print the value. That's external iteration. We may easily add `break` and `continue` if we choose to further alter the flow of iteration.

Internal iterators

Let's rewrite the iteration using internal iterators.

```
numbers.filter { e -> e % 2 == 0 }
    .forEach { e -> print("$e, ") } //10, 12, 18,
```

iterate.kts

Kotlin standard library has added a number of extension functions to collections. In this example we use two of them, the `filter()` and the `forEach()` functions. Both are higher-order functions and so we pass lambdas to them.

The lambda that is given to the `filter()` function returns `true` if the value passed as a parameter to the lambda is even and returns `false` otherwise. Thus, the `filter()` function will return a list of only the even values from the given collection numbers. The `forEach()` function works on that resulting collection of even numbers and passes the values, one at a time, to the given lambda. The lambda attached to `forEach()` prints the given value.

Which is better?

Even though the `for` loop of the external iterator isn't verbose in Kotlin, the internal iterator to achieve the same result is a tad more concise. The code naturally reads like the problem statements: given the numbers, filter only even numbers and print each. The difference between these two styles will widen as the complexity of the tasks performed is increased. Let's dig in further to see this.

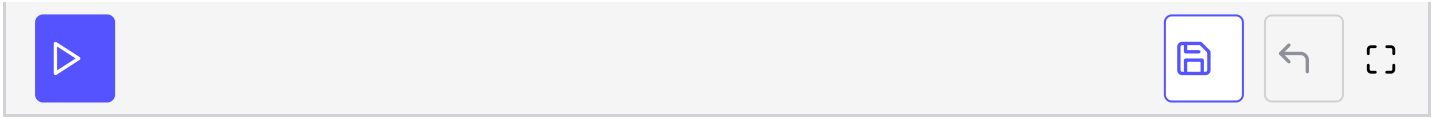
Suppose instead of printing the even numbers, we want to collect the doubles of the even numbers into another result collection. Here's the code using an external iterator to perform that task:

iterator to perform that task.

```
val doubled = mutableListOf<Int>()

for (i in numbers) {
    if (i % 2 == 0) {
        doubled.add(i * 2)
    }
}

println(doubled) //[20, 24, 36]
```



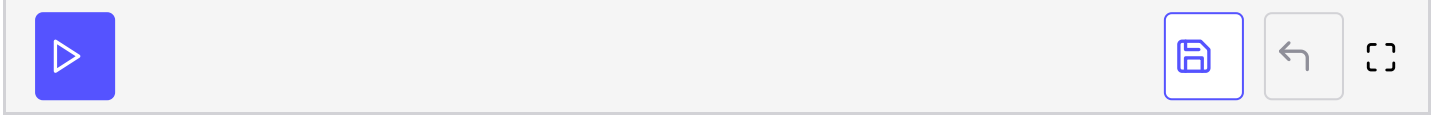
iterate.kts

The first smell in this code is the definition of an empty mutable list. Then, within the loop we add the double of each even number to that list. If you hear a programmer say “That’s not bad,” it’s just a sign of the Stockholm syndrome—it’s not their fault. The code has a few moving parts and is more complex than the corresponding internal iterator code, but many programmers are familiar with external iterators and often feel comfortable with that style.

Let’s take a look at the internal iterator version for the same task:

```
val doubledEven = numbers.filter { e -> e % 2 == 0 }
    .map { e -> e * 2 }

println(doubledEven) //[20, 24, 36]
```



iterate.kts

Instead of using the `forEach()` function, we used the `map()` function here. The `map()` function transforms the given collection into a result collection by applying the lambda to each element. The internal iterators form a *functional pipeline*—that is, a collection of functions that will be applied on objects or values that flow through the pipeline. The result of this functional pipeline is a read-only list containing the doubles of the even numbers from the original collection.

External iterators in Kotlin are good, but internal iterators are better. You can use whichever you’re comfortable with, and you also can refactor your code at any time to the style you feel is the best fit for the problem at hand.

time to the style you feel is the best fit for the problem at hand.

In the next lesson, let's dive in to see some internal iterators built into the Kotlin standard library.