Operator Functions

Learn how to define your own operators, which limitations Kotlin poses on such operators, and how they lead to succinct code.

WE'LL COVER THE FOLLOWING

Operator functions allow you to define the meaning of well-known symbols such as `+` and `-` for your own types.

Introducing Operators

Kotlin's operators allow calling functions using certain symbols. For instance, the infix `+` operator actually calls a `plus` function so that `a + b` translates to `a.plus(b)` under the hood.

Many operators use infix notation and therefore can be seen as an extension of infix functions. Instead of a function name, operator functions instead use an *operator symbol*. However, there are also operators using prefix and postfix notation:

- Prefix: `+a`, `-a`, `!a`
- Infix: `a * b`, `a..b`, `a == b`, ...
- Postfix: `a[i]`, `a()`

Kotlin has a predefined list of possible operators you can use (in contrast to languages like Scala or Haskell).

Important Kotlin Operators

Note: Some of the examples below require a custom operator definition to run. For instance, `"Coffee" * 3` doesn't work out of the box because Kotlin provides no such operator implementation.

Function Name	Operator	Example	Equivalent
Arithmetic Operators			
`plus`	`+`	`2 + 3`	`2.plus(3)`
`minus`	`-`	`"Kotlin" - "in"`	`"Kotlin".minus("in")`
`times`	`*`	`"Coffee" * 2`	`"Coffee".times(2)`
`div`	`/`	`7.0 / 2.0`	`(7.0).div(2.0)`
`rem`	`%`	`"android" % 'a'`	`"android".rem('a')`
Assignment Operators			
`plusAssign`	`+=`	`x += "Island"`	`x.plusAssign` (“island”)
`minusAssign`	`-=`	`x -= 1.06`	`x.minusAssign(1.06)`

timesAssign	*=	x *= 3	x.timesAssign(3)
divAssign	/=	x /= 2	x.divAssign(2)
remAssign	%=	x %= 10	x.remAssign(10)
Miscellaneous Operators			
inc	++	grade++	grade.inc()
dec	--	'c'--	'c'.dec()
contains	in	"A" in list	list.contains("A")
rangeTo	..	now..then	now.rangeTo(then)
get	[…]	skipList[4]	skipList.get(4)
set	[…] = …	array[5] = 42	array.set(5, 42)

For a full list of all operators, check out the Kotlin documentation.

Implementing an Operator Function

You can define the meaning of any of the allowed operators for any of your own types. **The declaration is similar as for infix functions, except that you use the `operator` modifier**:

```
operator fun Int.times(str: String) = str.repeat(this)
```

Remember that the function name can only be one of the predefined operators, whereas infix functions can have any name (recall your `withoutLast` function for instance). Also, not all operators have two parameters, and not all operators use infix notation.

With the function definition above, you can now repeat strings like this:

```
val greeting = 3 * "Ho "
```

This works because the `*` operator maps to the `times` operator function. Refer to the table above and the Kotlin documentation for all mappings.

Quiz

Operator Functions

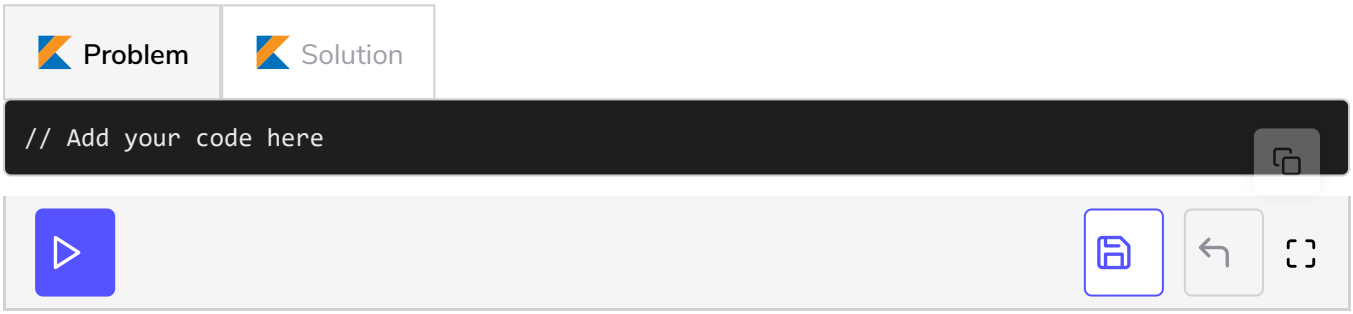
1 Which of the following function names can you use to define an operator in Kotlin? You can select multiple answers.

COMPLETED 0%

1 of 3

Exercises

Implement an operator function that returns whether a `String` contains *all* `Char`s in a given list (`List<Char>`).



You should be able to call the function as follows:

```
val requiredChars = listOf('x', 'y')
println(requiredChars in "xyzäöü")  // true
println(requiredChars in "x你z")  // false
println(requiredChars in "y好ä")  // false
```

Summary

Operators allow using predefined symbols to call a function.

- While this can improve code readability in certain cases, it should be used judiciously, since operators can quickly become confusing.
- There are dozens of allowed operators and many predefined operator implementations, such as `17++`, `"a" in "string"`, or “6 * 7”.
- To define a custom operator, you use the `operator` modifier and define either a member or extension function.

Congratulations! You now have a good understanding of the different types of functions in Kotlin and how to use them.

In the following section, you can recap the main points of each section in the course and find great resources to continue your Kotlin journey.