String Templates

We'll cover the following

Why string templates are needed

In programs, we often create strings with embedded values of expressions. Concatenating values to create strings using the `+` operator makes the code verbose and hard to maintain. String templates solve that problem by providing an elegant solution.

How to use string templates

Within a double-quoted string, the `$` symbol can prefix any variable to turn that into an expression. If the expression is more complex than a simple variable, then wrap the expression with `${}`.

A `$` symbol that's not followed by a variable name or expression is treated as a literal. You may also escape the `$` symbol with a backslash to use it as a literal.

Here's an example with a string template. Also, it contains a plain string with embedded `$` symbols that are used as literals.

```
val price = 12.25
val taxRate = 0.08

val output = "The amount $price after tax comes to $${price * (1 + taxRate)}"
val disclaimer = "The amount is in US$, that's right in \$only"

println(output)
println(disclaimer)
```

stringtemplate.kts

In the string template assigned to output, the first `$` symbol is used as a delimiter for the expression, the variable name, that follows it. The second `$` symbol is a literal since it's followed by another `$`, which isn't a variable or expression. The third `$` symbol prefixes an expression that's wrapped in `{}`. The other `$` symbols in the code are used as literals. Let's take a peek at the output of the code:

```
The amount 12.25 after tax comes to $13.23
The amount is in US$, that's right in $only
```

The earlier caution to prefer `val` over `var` applies here too. Let's take the code with `var` we saw previously and modify it slightly to use a string template.

```
var factor = 2

fun doubleIt(n: Int) = n * factor
var message = "The factor is $factor"

factor = 0

println(doubleIt(2))
println(message)
```

mutateconfusion.kts

Once again, don't run the code, but eyeball it and figure out the output of this code. Does it correspond with the following output?

```
0
The factor is 2
```

The variable `factor` within the function `doubleIt()` binds to the variable outside its immediate scope—that is, in its lexical scope. The value of `factor` at the time of the function call is used. The string template, on the other hand, is evaluated when the variable message is created, not when its value is printed out. These kinds of differences increase cognitive load and makes the code hard to maintain and also error prone. No need to torture fellow programmers with code like this. It's inhumane. Again, as much as possible prefer `val` over `var`.

QUIZ

Q

What will be the output of the following code snippet?

```
val factor = 2

fun doubleIt(n: Int) = n * factor
val message = "The factor is $factor"

factor = 0

println(doubleIt(2))
println(message)
```

Retake Quiz

In the next lesson, let’s look at using raw strings to remove some clutter and to create multiple lines of strings.