Raw Strings

We'll cover the following

- No escape
- Multiline strings

Dealing with escape characters makes the code messy. Instead of using escaped strings, in Kotlin we may use raw strings which start and end with three double quotes. We may use raw strings to place any character, without the need to use escapes, and may also use them to create multiline strings.

No escape

In an escaped string which starts and ends with a single double quote, we can't place a variety of characters, like new line or a double quote, for example, without using the escape character `\`. Even a simple case can be unpleasant to read, like this one:

```kotlin
val escaped = "The kid asked, \"How's it going, $name?\""
```

We had to escape the double quotes that were needed within the string. The more we use escaped strings, the messier it becomes. Instead of using escaped strings, in Kotlin we use raw strings. Just like escaped strings, raw strings can also be used as string templates, but without the mess of escaping characters. Here's the above escaped string changed to raw string—less clutter, more readable:

```kotlin
val raw = """The kid asked, "How's it going, $name?""""
```

Use escaped string, ironically, when you don't need to escape anything—for small, simple, plain vanilla strings. If you need anything more complex or multiple lines of string, then reach over to raw strings.

Multiline strings

The infamous + operator is often used to create multiple lines of strings, and that leads to nasty code that's hard to maintain. Kotlin removes that ceremony with a multiline string, which is a raw string that contains line breaks. Multiline strings can also act as string templates.

Let's create a string that runs across several lines, but without the + operator.

```
val name = "Eve"

val memo = """Dear $name, a quick reminder about the
party we have scheduled next Tuesday at
the 'Low Ceremony Cafe' at Noon. | Please plan to..."""

println(memo)
```

memo.kts

The multiline string starts with three double quotes, contains the string template expression to evaluate the variable name, and ends with three double quotes. The output of this code is multiple lines of string with the embedded expression evaluated.

```
Dear Eve, a quick reminder about the
party we have scheduled next Tuesday at
the 'Low Ceremony Cafe' at Noon. | Please plan to...
```

That worked beautifully, but—there always is a but—what if the multiline string were within a function, maybe within an if? Would the nesting mess things up? Let's find out.

```
fun createMemoFor(name: String): String {
  if (name == "Eve") {
    val memo = """Dear $name, a quick reminder about the
        party we have scheduled next Tuesday at
        the 'Low Ceremony Cafe' at Noon. | Please plan to..."""

    return memo
  }

  return ""
}

println(createMemoFor("Eve"))
```

nestedmemo.kts

The `createMemoFor()` function returns a multiline string if the parameter passed is equal to Eve. Let's see what the output beholds:

```
Dear Eve, a quick reminder about the
        party we have scheduled next Tuesday at
        the 'Low Ceremony Cafe' at Noon. | Please plan to...
```

The resulting string has preserved the indentation—yikes. Thankfully, it's not too hard to get rid of. Let's rework the example:

```
fun createMemoFor(name: String): String {
    if (name == "Eve") {
        val memo = """Dear $name, a quick reminder about the
            |party we have scheduled next Tuesday at
            |the 'Low Ceremony Cafe' at Noon. | Please plan to..."""
        return memo.trimMargin()
        }
        return ""
    }
println(createMemoFor("Eve"))
```

We made two changes. First, we placed a `|` on each line of the multiline string, starting with the second line. Second, we used the `trimMargin()` method, an extension function (we discuss these in Chapter 13, Fluency in Kotlin), to strip the margin out of the string. With no arguments, the `trimMargin()` method removes the spaces until the leading `|` character. The `|` character that's not in the leading position doesn't have any impact. Here's the output that shows the fix worked.

```
Dear Eve, a quick reminder about the
party we have scheduled next Tuesday at
the 'Low Ceremony Cafe' at Noon. | Please plan to...
```

If you do not want to use `|` as the leading delimiter, because maybe your text contains that character in arbitrary places, including the first character of a new line, then you may choose some other character—for example, let's go ahead and choose ~:

```
val memo = """Dear $name, a quick reminder about the
```

```
    ~party we have scheduled next Tuesday at


    ~the 'Low Ceremony Cafe' at Noon. | Please plan to..."""
return memo.trimMargin("~")
```

In the multiline string we use `~` as the delimiter instead of the default `|`, and in the call to `trimMargin()` we pass that specially chosen delimiter as argument. The output of this version is the same as the one where we used the default delimiter. So far in this chapter, we've looked at the improvements to expressions and statements in Kotlin when compared to languages like Java. But Kotlin prefers expressions over statements.

Let's discuss that in the next lesson.

QUIZ

Q

What will be the output of the following code snippet?

```
fun createMemoFor(name: String): String {
    if (name == "User") {
        val memo = """Dear $name,
          |Hope you are enjoying the course.
          |Have a good day $name..."""
        return memo
        }
        return ""
    }
println(createMemoFor("User").trimMargin())
```

Retake Quiz