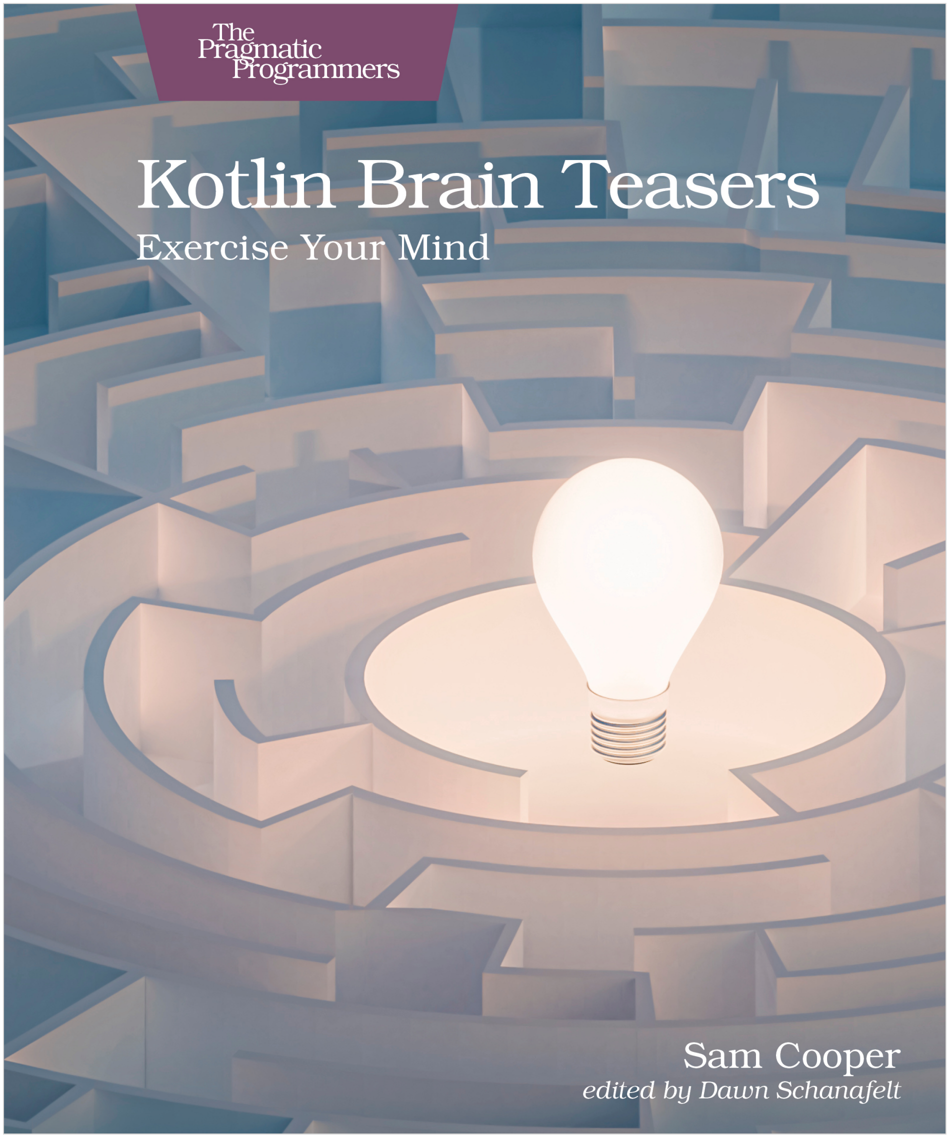
The Pragmatic Programmers
Kotlin Brain Teasers
Exercise Your Mind
Sam Cooper
edited by Dawn Schanafelt

The Vacuous Boomerang

src/commonMain/kotlin/com/example/puzzles/control/boomerang/ThrowReturn.kt

```kotlin
fun main() {
  val boomerang = throw return
  println("The value is: $boomerang")
}
```

Guess the Output

Try to guess the output (or error) before moving to the next page.

The program completes successfully, and displays no output.

Discussion

In a valid throw statement, the input slot on the right of the throw keyword must be filled by a Throwable exception. A return statement isn't a value at all, let alone a Throwable—so why is it a valid way to fill this slot?

Any time Kotlin's type checker is expecting a value of a specific type, it'll also accept an instruction that prevents the program from reaching the place where the value is required. In this puzzle, we're exploiting that twice in quick succession—first when we use throw in place of a real value for the boomerang variable, and again when we use return in place of the exception to throw.

Value Not Found

You probably already know that any time you don't want to return a value from a function, you can throw an exception instead:

```
fun getName(): String {
  if (hasName) { return myName } else { throw Exception("I have no name") }
}
```

Anyone calling this function can still safely use the result as a String:

```
val name: String = getName() // ✓ This is fine
```

Either getName() returns a String, or it throws an exception that causes the code to exit without assigning a value to the name variable at all. So if name receives a value, that value will always be a String. In fact, that's still true even if get-Name() always throws and never returns:

```
fun getName(): String { throw Exception("I have no name") }
val name: String = getName() // ✓ Still fine
```

If you think back to Puzzle 11, It's Over Nine Thousand!, on page ?, you might recognize this as another example of a *vacuous truth*. The getName() function never returns a value, so our code isn't lying when it says that every value returned by getName() will always be a String.

Meaningless Expressions

Now let's remove the function altogether, and just run its code directly:

```
val name: String = throw Exception("I have no name") // ✓ Still fine!
```

Just like a function, an expression in Kotlin is allowed to throw an exception instead of producing a value. And an exception isn't the only way to prevent the code from reaching the place where a value is used. For example, inside a function, a return instruction achieves the same by exiting the function:

```
fun main() {
  val name: String = return // ✓ This is fine: 'name' never gets a value
}
```

In our puzzle program, we use throw instead of providing a value for the boomerang variable, and then we use return instead of providing an exception for the throw statement. Since we didn't specify any particular type for the valueless boomerang variable, the compiler infers that its type is Nothing:

```
fun main() {
  val boomerang: Nothing = throw return
}
```

This doesn't mean that boomerang contains an object whose type is Nothing. Instead, it means that boomerang can never contain any value at all—because the assignment will always exit before it completes. The Nothing type is Kotlin's way of describing an outcome that can never happen. You can even use it to write your own functions that, like throw, can be used in place of any value:

```
fun myError(): Nothing { throw Exception("Something went wrong!") }
val name: String = myError() // ✓ 'myError' never returns a non-String value
```

Nothing or Never

Nothing is Kotlin's *bottom type*—a type that can act as a subtype of every other type, because it has zero possible values. Some other languages call their bottom type Never or NoReturn.

Here are the key points to remember:

- You can use jump instructions like return and throw in place of any value, because they divert the program before the value is ever actually used.
- Expressions that don't complete have a type of Nothing—Kotlin's way of describing a situation that never happens, or a value that can never exist.

Further Reading

Returns and jumps

https://kotlinlang.org/docs/returns.html

Short Circuits, Bottom Types, and the Vacuous Boomerang

https://sam-cooper.medium.com/2114bca82b3f