4. Learn about nullability

In this task, you learn about nullable versus non-nullable variables. Programming errors involving nulls have been the source of countless bugs. Kotlin seeks to reduce bugs by introducing non-nullable variables.

Step 1: Learn about nullability

By default, variables cannot be `null`.

1. Declare an `Int` and assign `null` to it.

```
var rocks: Int = null
⇒ error: null can not be a value of a non-null type Int
```

2. Use the question mark operator, `?`, after the type to indicate that a variable can be null. Declare an `Int?` and assign `null` to it.

```
var marbles: Int? = null
```

When you have complex data types, such as a list:

- You can allow the elements of the list to be null.
- You can allow for the list to be null, but if it's not null its elements cannot be null.
- You can allow both the list or the elements to be null.

Lists and some other complex data types are covered in a later task.

Step 2: Learn about the ? and ?: operators

You can test for `null` with the `?` operator, saving you the pain of writing many `if`/`else` statements.

1. Write some code the longer way to check whether the `fishFoodTreats` variable is not `null`. Then decrement that variable.

```
var fishFoodTreats = 6
if (fishFoodTreats != null) {
   fishFoodTreats = fishFoodTreats.dec()
}
```

2. Now look at the Kotlin way of writing it, using the `?` operator.

```
var fishFoodTreats = 6
fishFoodTreats = fishFoodTreats?.dec()
```

3. You can also chain null tests with the `?:` operator. Look at this example:

```
fishFoodTreats = fishFoodTreats?.dec() ?: 0
```

It's shorthand for "if `fishFoodTreats` is not `null`, decrement and use it; otherwise use the value after the `?:`, which is 0." If `fishFoodTreats` is `null`, evaluation is stopped, and the `dec()` method is not called.

Note: The `?:` operator is sometimes called the "Elvis operator," because it's like a smiley on its side with a pompadour hairstyle, the way Elvis Presley styled his hair. Read more about the Elvis operator in the Kotlin documentation.

A point about null pointers

If you really love `NullPointerExceptions`, Kotlin lets you keep them. The not-null assertion operator, `!!` (double-bang), converts any value to a non-null type and throws an exception if the value is `null`.

```
val len = s!!.length   // throws NullPointerException if s is null
```

Note: In programming slang, the exclamation mark is often called a " bang," so the not-null assertion operator is sometimes called the "double-bang" or "bang bang" operator. Because `!!` can throw an exception, it should only be used when it would be exceptional to hold a null value.