Variables without contents

In Python, a variable can have the special value `None` to indicate it has no value. A function can return `None` to indicate no result was found (perhaps because of an error).

In Java and Scala, a variable can have the special value `null`, and functions can return `null`. For instance, when you create an array, all its slots contain `null`.

However, you cannot perform any operation on `null` values, so if you forget to handle this case, you get a `NullPointerException`.

Kotlin does not normally allow `null` as a value.

You can use `null` for variables of nullable type.

Nullable types

Put a `?` behind the type name to make it nullable:

```
>>> var s: String? = "CS109"
>>> println(s)
CS109
>>> s = null
>>> println(s)
null
>>> s = "I'm a nullable string"
>>> println(s)
I'm a nullable string
```

With nullable types, the only allowed operations are equality comparisons and string conversion:

```
>>> if (s == null)
...   println("s is empty")
```


Working with nullable types

No other methods can be called on a variable of nullable type:

```
>>> s.length
error: only safe or non-null asserted calls are
allowed on a nullable receiver of type String?
```

Manually check for `null`:

```
>>> fun strlen(s: String?): Int =
        if (s == null) 0 else s.length
>>> strlen("Hello")
5
>>> strlen(null)
0
```


The `?.` operator

Use `?.` to call a method, or simply return `null`:

```
>>> s
I'm a nullable string
>>> s.length
error: only safe or non-null asserted calls ...
>>> s?.length
21
>>> s?.startsWith("I'm")
true
>>> s = null
>>> s?.length
null
>>> s?.startsWith("I'm")
null
```

The Elvis operator ?:

We can get another result by replacing `null` with another value using the Elvis operator `?:`

```
>>> fun check(s: String?): Boolean =
          s?.startsWith("I'm") ?: false
>>> check("I'm nullable")
true
>>> check("Bah")
false
>>> check(null)
false
```


The promise operator !!

We can also promise to the compiler that a value will never be `null`:

```
>>> var sn: String? = "I'm nullable"
>>> var s: String = sn
error: type mismatch
>>> var s: String = sn!!
>>> s
I'm nullable
>>> sn.length
error: only safe or non-null asserted calls
>>> sn!!.length
12
```

If your program violates the promise, an error will occur when the promise is executed.

Example

The Kotlin standard function `readLine()` returns a `String?`.

```
fun reverser() {
  var line: String? = readLine()
  while (line != null) {
    println(line.reversed())
    line = readLine()
  }
}

println("Enter lines to be reversed:")
reverser()
```

(Note: this function is different from `org.otfried.cs109.readString`.)