Functions

- Functions are defined with the `fun` keyword
- They have a name, parameters in brackets, a return type, and a definition

```
fun successor(x: Int): Int = x + 1
```

An Expression Body

We can use a `=` and a single expression on the right hand side to define the function - an 'expression body'.

```
fun even(x: Int): Boolean = x % 2 == 0
```

A Block Body

To define a function that operates as a sequence of steps we use a 'block body'. Here, we use braces instead of a `=`, and at the end we need a `return` statement.

```
fun turns(
    start: Double,
    end: Double,
    radius: Double,
): Double {
    val kmToMs = 1000
    val totalDistance = (end - start) * kmToMs
    val circumference = 2 * PI * radius
    return totalDistance / circumference
}
```

When Clause

When in Haskell we would use guards, in Kotlin we can use a `when` expression. The when expression can have multiple clauses.

```
fun signum(x: Int): Int =
    when {
        x < 0 -> -1
        x >= 0 -> 1
        else -> 0
    }
```

We can use a 'when with subject' when we want check various cases of 'n'. We can have multiple cases on the same line when the result is the same.

```
fun fib(n: Int): Int =
    when (n) {
        0, 1 -> 1
        else -> fib(n - 1) + fib(n - 2)
    }
```

Functions that don't return anything

The function does not return anything, so its return type is `Unit`. This function is an "action" because it causes a side-effect (printing) rather than returning a value.

```
fun doesItStartWithH(s: String): Unit =
    when {
        s.startsWith("H") -> println("yes, H")
        else -> println("no, not H")
    }
```

Default Values

We can provide default values for function arguments.

```
fun f1(a: Int = 4, b: Int = 6) = a * a + b * b
```

We can provide specific values using keyword arguments.

```
val t = f1(b = 8)
```

The `main` Function

Provides an entry point.

```kotlin
fun main() {
    println(square(2))
}
```