Functions as Objects
We have seen that Scala’s numeric types and the `Boolean` type can be implemented like normal classes.

But what about functions?
Functions as Objects

We have seen that Scala’s numeric types and the Boolean type can be implemented like normal classes.

But what about functions?

In fact function values are treated as objects in Scala.

The function type \( A \rightarrow B \) is just an abbreviation for the class scala.Function1[A, B], which is defined as follows.

```scala
package scala

trait Function1[A, B] {
  def apply(x: A): B
}
```

So functions are objects with apply methods.

There are also traits Function2, Function3, ... for functions which take more parameters (currently up to 22).
An anonymous function such as

\[ (x: \text{Int}) \Rightarrow x \times x \]

is expanded to:
An anonymous function such as

\[(x: \text{Int}) \Rightarrow x \times x\]

is expanded to:

```scala
{ class AnonFun extends Function1[Int, Int] {
    def apply(x: Int) = x * x
}
    new AnonFun
}
```
An anonymous function such as

\[(x: \text{Int}) \Rightarrow x \times x\]

is expanded to:

```scala
{ class AnonFun extends Function1[Int, Int] {
    def apply(x: Int) = x * x
}
    new AnonFun
}
```

or, shorter, using *anonymous class syntax*:

```scala
new Function1[Int, Int] {
    def apply(x: Int) = x * x
}
```
Expansion of Function Calls

A function call, such as \( f(a, b) \), where \( f \) is a value of some class type, is expanded to

\[
f.\text{apply}(a, b)
\]

So the OO-translation of

\[
\text{val } f = (x: \text{Int}) => x * x
\]

\( f(7) \)

would be

\[
\text{val } f = \text{new } \text{Function1[Int, Int]} \{ \\
\quad \text{def } \text{apply}(x: \text{Int}) = x * x \\
\} \\
\text{f.apply}(7)
\]
Functions and Methods

Note that a method such as

```python
def f(x: Int): Boolean = ...
```

is not itself a function value.

But if $f$ is used in a place where a Function type is expected, it is converted automatically to the function value

$$(x: \text{Int}) \Rightarrow f(x)$$

or, expanded:

```java
new Function1[Int, Boolean] {
    def apply(x: Int) = f(x)
}
```
Exercise

In package week4, define an

```java
object List {
    ...
}
```

with 3 functions in it so that users can create lists of lengths 0-2 using syntax

```java
List() // the empty list
List(1) // the list with single element 1
List(2, 3) // the list with elements 2 and 3.
```