# Functions as Objects

Principles of Functional Programming

## 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 => B` is just an abbreviation for the class `scala.Function1[A, B]`, which is defined as follows.

```
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.

## Expansion of Function Values

An anonymous function such as

```
(x: Int) => x * x
```

is expanded to:

## Expansion of Function Values

An anonymous function such as

```
(x: Int) => x * x
```

is expanded to:

```
new Function1[Int, Int]:
  def apply(x: Int) = x * x
```

## Expansion of Function Values

An anonymous function such as

```
(x: Int) => x * x
```

is expanded to:

```
new Function1[Int, Int]:
  def apply(x: Int) = x * x
```

This *anonymous class* can itself be thought of as a block that defines and instantiates a local class:

```
{ class $anonfun() extends Function1[Int, Int]:
    def apply(x: Int) = x * x
  $anonfun()
}
```

## 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.apply(a, b)
```

So the OO-translation of

```
val f = (x: Int) => x * x
f(7)
```

would be

```
val f = new Function1[Int, Int]:
  def apply(x: Int) = x * x

f.apply(7)
```

## Functions and Methods

Note that a method such as

```
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: Int) => f(x)
```

or, expanded:

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

## Exercise

In package `week3`, define an

```
object IntSet:
  ...
```

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

```
IntSet()        // the empty set
IntSet(1)       // the set with single element 1
IntSet(2, 3)    // the set with elements 2 and 3.
```