Functions

Effective Programming in Scala

Functions

The `map` operation takes as parameter a *function* applied to each element of the list.

```
contacts.map(contact => contact.name)
```

We pass a *function literal* as a parameter. In this example, the function takes one parameter, named `contact`, and returns the expression `contact.name`.

Functions: Examples

Here are some examples of functions.

```
val increment: Int => Int =
  x =>
    val result = x + 1
    result

val add =
  (x: Int, y: Int) => x + y

add(1, increment(2)) // 4
```

Functions: Reference

More generally, a value like:

```
(t1, t2, ..., tn) => e
```

Is a function that takes `n` parameters, and which returns the expression `e`.

In case the compiler can't infer the types of the function arguments, you have to write them explicitly:

```
(t1: T1, t2: T2, ..., tn: Tn) => e
```

Last, a function is a value, so it has a *function type*. The syntax for function types looks like the syntax for function literals:

```
(T1, T2, ..., Tn) => R
```

Puzzle

Question: what is the type of the following function?

```
(contact: Contact) => contact.email.endsWith("@sca.la")
```

O `Contact => String`

O `Contact => Boolean`

O `String => Contact`

O `String => Boolean`

Puzzle

Question: what is the type of the following function?

```
(contact: Contact) => contact.email.endsWith("@sca.la")
```

O `Contact => String`

X `Contact => Boolean`

O `String => Contact`

O `String => Boolean`

Functions vs Methods (1)

What is the difference between the following definitions of `increment`?

```
def increment(x: Int): Int = x + 1
val increment: Int => Int = x => x + 1
```

In both cases, we can call `increment` like the following:

```
increment(41) // : Int = 42
```

Functions vs Methods (1)

What is the difference between the following definitions of `increment`?

```
def increment(x: Int): Int = x + 1
val increment: Int => Int = x => x + 1
```

In both cases, we can call `increment` like the following:

```
increment(41) // : Int = 42
```

The difference is a bit technical, and comes from the fact that Scala supports both functional programming and object oriented programming.

Functions vs Methods (2)

```scala
def increment(x: Int): Int = x + 1
val increment: Int => Int = x => x + 1
```

A key difference is that the second version defines a **value** that can be passed as a parameter or returned as a result.

The runtime creates an object for it in memory.

Functions vs Methods (3)

```
val increment: Int => Int = x => x + 1
```

Calling a function means calling its method `apply`:

```
increment(41)
// is equivalent to
increment.apply(41)
```

When we write `increment(41)`, the compiler rewrites it to `increment.apply(41)`.

Functions vs Methods (4)

Most of the time, you don't need to think about whether you use a method or a function, because the compiler is able to automatically convert methods into functions, when necessary.

For instance, you can write:

```
val xs: List[Int] = List(1, 2, 3)
def increment(x: Int): Int = x + 1
xs.map(increment) // = List(2, 3, 4)
```

Here, the compiler sees that the operation `map` expects a function, so it converts the method `increment` into a function value.

Summary

Functions can be used as values.

You can define a function with the arrow syntax: `(x: Int) => x + 1`.

A function that takes a parameter of type `A` and returns a result of type `B` is a value of type `A => B`, which has an `apply` method:

```
def apply(a: A): B
```

Calling the function means calling its `apply` method.