Greetings - Scala fundamentals (part 5)

Aims

We're going to continue working on our `GreeterApplication` and introduce the concept of **flow control**.

Prerequisites

You have completed Greetings - Scala fundamentals (part 4).

Lesson steps

We are now going to introduce **flow control**.

If we want to perform the same operation on multiple elements in a collection, we can use flow control. We can use flow control to repeatedly perform the same task on all of the elements in a collection.

We can also use flow control to perform tasks based on whether or not a particular condition is met. We've seen this already, when we used the if/then control structure.

For comprehensions

```
for (name <- names) {
   println(name)
 }
```

In the above example, we are going to iterate over `names`.

Let's examine this declaration a little closer:

- `for(name <- names)` - we use our `names` value, then with the `<-` operator (the generator) we generate a new, temporary value called `name`. In wrapping these inside of `for`, we declare that we want to do *something* for each name within names.

- `println(name)` - inside of our braces, we use our temporary value `name` to print each element in `names` to the console.

In this instance, we have used the generator operator `<-`. It's one of many new operators we'll encounter!

Further reading

Scala Operators

For loop with ranges

In the example above we have create a *for* loop with a range, the range being the number of items in the list `names`.

There are other ways of defining a range in *for* loops in Scala.

We can specify a finite number of iterations:

```
for (a <- 1 to 10) {
    println("Value of a: $a")
}
```

This will print out ten statements to the console each with `value of a: X` where `X` is the numbers from `1 up and including 10`.

We could also specify:

```
for (a <- 1 until 10) {
    println("Value of a: $a")
}
```

This would do the same as the above example, however, it would print from `1 up and including 9`.

For loop with filters (guards)

You can also specify a filter in a for loop using the following syntax:

```
for (a <- Range if a != 3) {
```

```
    ...
}
```

This will iterate through the list of numbers in the `Range` i.e. `1 to 10` and **not** perform the statement at `...` if the current number at `a` is 3.

You can also have multiple filters separating by a `;` semicolon.

```
for (a <- Range if a != condition; if condition2) ...
```

For loop with yield

We can also use for loops to return a value after each loop has been completed and evaluated.

```
val names = List("adam", "david", "daniel", "david")

val filtered = for (
  name <- names if name != "david"; if name != "daniel"
) yield name

println(filtered)
```

In the example above, on each iteration the filter is going to be evaluated, if the name matches either *david* or *daniel* then it will ignore the yield statement on that iteration.

The result of this would be `List("adam")`.

We could also modify what is returned from the yield command, say for instance we had another list of numbers for their ages.

We could do the following:

```
  val ages = List(1, 25, 31, 14)
  val names = List("adam", "david", "daniel", "david")
  val filtered = for (
    name <- names if name != "david"; if name != "daniel";
    age <- ages
  ) yield s"name $name, age: $age"

  println(filtered)
```

Task

> What do you think the above statement will output?

Run the code and see for yourself! Not what you expected?

The actual output is:

```
List(
name adam, age: 1,
name adam, age: 25,
name adam, age: 31,
name adam, age: 14,
name david, age: 1,
name david, age: 25, ...)
```

Why do you think that is?

As you can see, the for loop is iterating over all the elements in the *age* list first and applying the name from the *names* list to each element at the current position. This is a *nested* loop.

Further reading

Control structures

Recap

We've covered for loops and for comprehensions, as methods of flow control. We'll be looking at other methods of flow control in future sections of this academy.

We also learned about more of the operators used in Scala.

Resources

- Scala Operators
- String Interpolation | Scala Documentation
- Control structures