Theory: Values and variables

10 minutes 0 / 5 problems solved

Skip this topic Start practicing

206 users solved this topic. Latest completion was about 3 hours ago.

If we can think about a program as a sequence of instructions or statements, we need some concepts for the intermediate results in between. Namely, we're talking about decomposing the logic to steps, saving the outcome of each step and passing it to the next one. Scala has two types of constructions especially for that purpose: values and variables.

§1. Values

You can give a name to some statement result:

```
scala> val a = 2 + 2
a: Int = 4
```

Constructions like above are **values** that use the **val** keyword. Referencing a value does not re-compute it. Moreover, values cannot be re-assigned:

```
scala> val a = 2 + 2
a: Int = 4
scala> a = 3
<console>:12: error: reassignment to val
       a = 3
         ^
```

The type of value can be either omitted and inferred automatically by the compiler, or stated explicitly:

```
scala> val b: Short = 2 + 3
b: Short = 5
```

In the example above, we force the value type to be `Short`. If we omit specifying the type, it will be `Int` as it is more convenient for the sum of two integers:

```
scala> val b = 2 + 3
b: Int = 5
```

§2. Variables

Variables are similar to values, except it's possible to re-assign them. You can define a variable with the **var** keyword:

```
scala> var c = 3 + 4
c: Int = 7
scala> c = 8
c: Int = 8
```

As it is with values, the type of a variable can be either omitted and inferred, or explicitly stated:

```
scala> var d: Long = 4 + 5
d: Long = 9
scala> var e = 4 + 5
e: Int = 9
```

§3. Naming

Values and variables give names to statements. Scala allows you to use any valid **identifier** as a name. An identifier can start with a letter and be followed by an arbitrary sequence of letters, digits or operator characters such as underscore (_) or plus (+):

Current topic:

Values and variables

Topic depends on:

✗ Basic literals

Topic is required for:

Functions introduction

Introduction to collections

Branching

Tuples

Table of contents:

```
scala> val x = 1
x: Int = 1
scala> var MAX_LEN2 = 2
MAX_LEN2: Int = 2
scala> val value_with_@ = 3
value_with_@: Int = 3
```

It is allowed for identifiers to start with an operator character followed by an arbitrary sequence of operator characters:

```
scala> val + = 4
+: Int = 4
scala> var +%*@! = 5
+%*@!: Int = 5
```

There are exceptions, of course: for example, it is impossible to define an identifier with the name `val` because it is a reserved keyword. Scala allows you to work around this exception using identifiers formed by an arbitrary string between back-quotes:

```
scala> val `val` = 6
val: Int = 6
scala> val `var` = 7
var: Int = 7
scala> var `A B C` = 8
A B C: Int = 8
```

As you can see, Scala has a great variety of options for naming values or variables. With these possibilities, you can be creative and use all instruments to define things as precisely as you can.

§4. (Im)mutability

So, why does Scala need two concepts, values and variables, and not just one? You could say that if variables have the same properties as values and in addition can be modified (re-assigned, mutated), then the choice is obvious: use variables everywhere. There is no mistake in this logic, but let's take a step back.

To put it short, more power is more responsibility. If we use variables in a program, we can change them from any place of our code. To control this modification, we have to use special techniques, build barriers and think twice before writing a new line of code. Could we avoid this and make it easier? Well, yes: we could use values.

The fact that values cannot be modified is a quality called **immutability**. In Scala, values are preferred due to this property. Nobody will scold you if you decide to use variables for a specific part of your code; in fact, there are a lot of cases when you have to modify things. However, if you can do without mutation and not use variables, it is an error-prone solution.

Report a typo

19 users liked this theory. 0 didn't like it. **What about you?**

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion