Programming Paradigms
Week 2

Jacek Wasilewski

1 Expressions

1. Scala command line and Scala Worksheet work like a calculator. You can evaluate commands that you type straight away. Check the result for `7 + 13`.

2. Everything in Scala gets a name - what name did your `7 + 13` expression get? (Hint: default names start with `res`)

3. Type of your result has been inferred - what is it?

4. Add `5` to your result, use the name that Scala gave you.

5. Expressions are one of the basic units that Scala offers you. Previously evaluated `7 + 13` is an expression. You can define named expression in the following way:

   ```
   def <name> = <expression>
   ```

   Example:

   ```
   def five = 1 + 4
   ```

6. Create your own named expression. Let it be a single number.

7. Try to manually modify the expression that you have just created. Is it possible?

8. The types of your expressions have been inferred automatically by Scala. Instead you can define them explicitly:

   ```
   def five: Int = 1 + 4
   ```

2 Functions

1. Using `def` keyword you can also define functions that take parameters. Actually, pure functions are expressions that take (or not) parameters. Consider the following example:

   ```
   def five: Int = 1 + 4
   def five(): Int = 1 + 4
   ```

   The above are equivalent. In this case, you could omit the return type definition.

2. Functions with parameters can be defined as in this example:

   ```
   def addOne(x: Int): Int = x + 1
   addOne(4)
   ```

   This should return `5`.

3. You can have as many parameters as you wish, separating them with a comma. To define a parameter you first create its name, then you specify its type.

4. Write a function that adds two numbers. Use `Double` as input and output types. Name the function `add`. Prove that it works.

5. Chaining of functions is possible:

   ```
   addOne(addOne(1 + 1))
   ```

6. Write a function that takes one `Int` parameter and returns its square plus 1. Use the `addOne` function defined before. Example: for input 5, it should output 26 (5 * 5 + 1).

3 Blocks and scopes

1. So far, our code has not been longer than one line. If we would like to write a function that is more complex, we can put our code into blocks defined by braces - and . That also defines the scope of definition, values and variables. Last expression in the block defines the block's value. Example:

```
def five(): Int = {
    1 + 4
}
```

is equivalent to:

```
def five(): Int = 1 + 4
```

2. Multi-statement function:

```
def five(): Int = {
    def four = 4
    1 + four
}
```

or you put them in one line, separating with a semicolon:

```
def five(): Int = {
    def four = 4; 1 + four
}
```

3. Definitions inside a block are only visible inside that block. If your definition uses name that has already been defined outside the current block, it overrides it only inside the block. Consider example:

```
def x = 0
def y = {
    def x = 4
    x * x
}
x + y
```