Exercises — Scala Day One (Part II)

Everything is an object

Spring term 2016

You need to have installed the Scala distribution before commencing these exercises. Don't forget the documentation for the distribution as that will come in very useful.

This exercise sheet provides additional examination of the basic imperative and object-oriented capabilities of the language.

1. What are the values and types of the following Scala literals?
   (a) `42`
   (b) `true`
   (c) `123L`
   (d) `42.0`
2. What is the difference between the following literals?
   (a) `'a'`
   (b) `"a"`

   What is the type and value of each?
3. What is the difference between the following expressions?
   (a) `"Hello world`"!
   (b) `println("Hello world`")!

   What is the type and value of each?
4. What is the type and value of the following literal? Try writing it using the REPL or in a Scala *worksheet* and see what happens!

   ```
   'Hello world!'
   ```
5. What are the types and values of the following conditionals?
   (a)
   ```
   val a = 1
   val b = 2
   if(a > b) "alien" else "predator"
   ```
   (b)
   ```
   val a = 1
   val b = 2
   if(a > b) "alien" else 2001
   ```

    (c) `if(true) "hello"`

6. What is the difference between the following expressions? What are the similarities?
    (a) `1 + 2 + 3`
    (b) `6`

7. Define an object called `calc` with a method `square` that accepts a `Double` as an argument and…you guessed it…squares its input. Add a method called `cube` that cubes its input and calls `square` as part of its result calculation.

8. Copy and paste `calc` from the previous exercise to create a `calc2` that is generalised to work with `Int`s as well as `Double`s. As you have Java experience, this should be fairly straightforward.

9. When entered on the REPL, what does the following program output, and what is the type and value of the final expression?

```
object argh {
  def a = {
    println("a")
    1
  }

  val b = {
    println("b")
    a + 2
  }

  def c = {
    println("c")
    a
    b + "c"
  }
}

argh.c + argh.b + argh.a
```

    Think carefully about the types, dependencies, and evaluation behaviour of each field and method.

10. (a) Define an object called `Person` that contains fields called `firstName` and `lastName`.
    (b) Define a second object called `Alien` containing a method called `greet` that takes your `Person` as a parameter and returns a greeting using their `firstName`.

    What is the type of the `greet` method? Can we use this method to greet other objects?

11. Are methods values? Are they expressions? Why might this be the case?

12. Create your own package containing three trivial classes (just define the classes, don't give them bodies). Now import one class, two classes, and all classes, and show that you've successfully imported them in each case.

13. Create a value of type `Range` that goes from `0` to `10` (not including 10). Satisfy the following tests:

    ```
    val r1 = // fill this in
    r1 is // fill this in
    ```

    Use `Range.inclusive` to solve the problem above. What changed?

14. Write a for loop that adds the values 0 through 10 (including 10). Sum all the values and ensure that it equals 55. Must you use a var instead of a val ? Why? Satisfy the following test: total is 55

15. Write a for loop that adds even numbers between 1 and 10 (including 10). Sum all the values and ensure that it equals 30. Hint: this conditional expression determines whether a number is even: if (number % 2 == 0) The % (modulo) operator checks to see if there is a remainder when you divide number by 2. Satisfy the following:

    ```
    `totalEvens is 30`
    ```

16. Write a for loop that adds even numbers between 1 and 10 (including 10) and odd numbers between 1 and 10. Calculate a sum for the even numbers and a sum for the odd numbers. Did you write two for loops? If so, try rewriting this with a single for loop. Satisfy the following tests:

    ```
    `evens is 30`
    ```

17. Create a `Vector` and populate it with words (which are `String`s). Add a for loop that prints each element in the `Vector`. Building on the previous exercise, append to a variable of type `String` to create a sentence. Satisfy the following test:

    ```
    sentence.toString() is "The dog visited the firehouse "
    ```

    That last space is unexpected. Use `String`'s method `replace` to replace `firehouse` with `firehouse!` Satisfy the following test:

    ```
    theString is "The dog visited the firehouse!"
    ```

18. Building on your solution from the previous exercise , write a *for loop* that prints each word, reversed. Your output should match:

    ```
    Output: ehT god detisiv eht esuoherif
    ```

19. Write a *for loop* that prints the words from the earlier exercise in reverse order (last word first, etc.). Your output should match:

    ```
    firehouse the visited dog The
    ```

20. (a) Create and initialise two `Vector`s, one containing `Int`s and one containing `Double`s. Call the `sum`, `min`, and `max` operations on each one and see what happens.

    (b) Create a `Vector` containing `String`s and apply the `sum`, `min`, and `max` operations. Explain the results. One of those methods won't work. Why?

    (c) In *For Loops*, we added the values in a `Range` to get the sum. Try calling the `sum` operation on a `Range`. Does this do the entire summation in one step?

21. `List` and `Set` are similar to `Vector`. Use the REPL to discover their operations and compare them to those of `Vector`. Create and initialise a `List` and `Set` with words, then print each one. Try the `reverse` and `sorted` operations and see what happens.

22. Palindromes are words or phrases that read the same forward and backward. Some examples include "mum" and "dad".

    Write a method to test words or phrases for palindromes.

    Hint: `String`'s `reverse` method may prove useful here.

    Satisfy the following tests:

    ```
    isPalindrome(``mum'')
    ```

    is true, and

    ```
    isPalindrome(``dad'')
    ```

    is true, and

    ```
    isPalindrome(``street'')
    ```

    is false.

23. Create a function `forecast` that represents the percentage of cloudiness, and use it to produce a "weather forecast". Strings such as "Sunny" (100), "Mostly Sunny" (80), "Partly Sunny" (50), "Mostly Cloudy" (20), and "Cloudy" (0).

    For this exercise use pattern matching and only match for the legal values 100, 80, 50, 20, and 0. Everything else should produce "Unknown".

    Your function should satisfy the following tests:

    ```
    forecast(100) is "Sunny"
    forecast(80) is "Mostly Sunny"
    forecast(50) is "Partly Sunny"
    forecast(20) is "Mostly Cloudy"
    forecast(0) is "Cloudy"
    forecast(15) is "Unknown"
    ```

24. Define a class `SimpleTime` that takes two arguments: an `Int` that represents hours, and an `Int` that represents minutes. Use *named arguments* to create a `SimpleTime` object which satisfies the following tests:

```
val t = new SimpleTime(hours=5, minutes=30)
t.hours is 5
t.minutes is 30
```

25. Using your answer to the previous question for `SimpleTime`, default the minutes to 0 so that you don't have to specify them. Your code should satisfy the following tests:

```
val t2 = new SimpleTime2(hours=10)
t2.hours is 10
t2.minutes is 0
```

26. Create a class `Planet` that has, by default, a single moon. The `Planet` class should have a `name` (a `String`) and `description` (a `String`). Use *named arguments* to specify the `name` and `description`, and a default for the number of moons. Satisfy the following tests:

```
val p = new Planet(name = "Mercury", description = "small and hot planet", moons =
p.hasMoon is false
```

27. (a) Create a *case class* to represent a `Person` in an address book. Complete the class with a `String` for the `name` and a `String` for contact information. Satisfy the following tests:

    ```
    val p = Person("Jane", "Smile", "jane@smile.com")
    p.first is "Jane"
    p.last is "Smile"
    p.email is "jane@smile.com"
    ```

    (b) Create some `Person` objects. Put the `Person` objects in a `Vector`. Satisfy the following tests:

    ```
    val people = Vector(
        Person("Jane","Smile","jane@smile.com"),
        Person("Ron","House","ron@house.com"),
        Person("Sally","Dove","sally@dove.com")
    )
    people(0) is "Person(Jane,Smile,jane@smile.com)"
    people(1) is "Person(Ron,House,ron@house.com)"
    people(2) is "Person(Sally,Dove,sally@dove.com)"
    ```

28. Maps store information using unique keys. An email address can serve as such a key. Create a class `Name` containing `firstName` and `lastName`.

    Create a `Map` that associates `emailAddress` (a `String`) with `Name`. Satisfy the following test:

```
val m = Map("sally@taylor.com"-> Name("Sally","Taylor"))
m("sally@taylor.com") is Name("Sally", "Taylor")
```

29. Adding to your solution for the previous exercise, add `Jiminy Cricket` to the map, where the email address is `jiminy@cricket.com`.

    Satisfy the following tests:

```
m2("jiminy@cricket.com") is Name("Jiminy","Cricket")
m2("sally@taylor.com") is Name("Sally", "Taylor")
```

30. Sets store distinct values. Create a set for the following languages: English, French, Spanish, German, and Chinese. Write a Scala class that uses a `Set` to represent this problem.
    (a) What happens when you try to add Turkish?
    (b) Try to add a language that already exists in the set (for example, French). What happens?
    (c) Remove "Spanish" from a the set containing English, French, Spanish, German, and Chinese.
    (d) Remove `jiminy@cricket.com` from a `Map` containing information for Jiminy Cricket, Mary Smith, and Sally Taylor. Test your solution.

Credits

Thanks to Noel Walsh and Bruce Eckel for the basis of these exercises…