BA-BEAD Big Data Engineering for Analytics

BEAD Workshop Series

W 02 - Scala Exercise Solution

1) Write a function that computes the area of a circle given its radius.

Answer

A circle's area is equal to the square of its radius times Pi. A quick and dirty solution of multiplying the radius twice with a reasonable hard coded version of Pi will get us close to the solution.

```
scala> def circleArea(r: Double) = r * r * 3.14159
circleArea: (r: Double)Double

scala> circleArea(8)
res0: Double = 201.06176
```

We can make use of scala.math to handle the square of the radius and a more approximate version of Pi, if more accuracy is desired.

```
scala> def circleArea2(r: Double) = math.pow(r,2) * math.Pi
circleArea2: (r: Double)Double

scala> circleArea2(8)
res1: Double = 201.06192982974676
```

2) Provide an alternate form of the function in #1 that takes the radius as a String. What happens if your function is invoked with an empty String ?

Answer

Providing the radius as a double number encoded as a string makes the function a little more complex. Not only will it be necessary to try to convert the string, but we'll have to figure out an acceptable result when the string is empty.

As this function takes up more than a single line, I'll add an explicit return type for clarity and also wrap the function body in curly braces. I'll also go ahead with the number zero as the return value when the input radius is empty.

A more advanced version of this function would also handle completely invalid radius values in the input string, such as "Hello" or "123abc". However, this function won't include that case, as it requires exception handling which we haven't yet covered.

```
scala> def circleArea3(r: String): Double = {
     |   r.isEmpty match {
     |     case true => 0
     |     case false => math.pow(r.toDouble,2) * math.Pi
     |   }
     | }
circleArea3: (r: String)Double
```

```
scala> circleArea3("8")
res2: Double = 201.06192982974676

scala> circleArea3("")
res3: Double = 0.0
```

3) Write a recursive function that prints the values from 5 to 50 by fives, without using for or while loops. Can you make it tail-recursive?

Answer

Recursive functions that have no return value are rather easy to write, as the result of invoking the function is ignored. The "@annotation.tailrec" will work easily as there is no return value which may be saved locally; we can simply exit immediately after the invocation.

Here’s my version of this recursive function. Can you find a shorter, simpler way to write this?

```
scala> @annotation.tailrec
     | def fives(cur: Int, max: Int): Unit = {
     |   if (cur <= max) {
     |     println(cur)
     |     fives(cur + 5, max)
     |   }
     | }
fives: (cur: Int, max: Int)Unit

scala> fives(0, 20)
0
5
10
15
20
```

4) Write a function that takes a milliseconds value and returns a string describing the value in days, hours, minutes and seconds. What’s the optimal type for the input value?

Answer

The common format for using milliseconds to describe a time is using the value zero to indicate 12:00 am GMT on January 1st, 1970. Also known as "epoch time", this format is supported by the JVM library classes java.util.Date and java.util.Calendar, among others.

However, we can write this function without making use of the JVM classes. Here's a version that expects the epoch time in milliseconds specified as a long value. Using a long value (64 bits) ensures that the code will work for millenia.

The first step is to reduce the milliseconds down to a more reasonable seconds value. The days is computed by dividing the seconds by the seconds in a day, while the remaining values are computed by modules of the nearest larger time unit.

```
scala> def descTime(epochMs: Long) = {
     |   val secs = epochMs / 1000
     |
     |   val days = secs / 86400
     |   val hours = (secs % 86400) / 3600
     |   val minutes = (secs % 3600) / 60
     |   val seconds = secs % 60
     |   s"$days days, $hours hours, $minutes minutes, $seconds
seconds"
     | }
descTime: (epochMs: Long)String

scala> descTime(123456789000L)
res20: String = 1428 days, 21 hours, 33 minutes, 9 seconds
```

5) Write a function that calculates the first value raised to the exponent of the second value. Try writing this first using math.pow, then with your own calculation. Did you implement it with variables? Is there a solution available that only uses immutable data? Did you choose a numeric type that is large enough for your uses?

Answer

Here's the easy way, using the math.pow library function.

```
scala> def pow(x: Double, y: Double): Double = math.pow(x,y)
pow: (x: Double, y: Double)Double

scala> pow(2, 5)
res0: Double = 32.0
```

Using a custom calculation, here's a version that stores the intermediate results in a variable. It's not especially robust, since it doesn't do validation on the exponent (eg, checking if a negative number was given).

```
scala> def pow(x: Double, y: Int): Double = {
     |   var p = 1.0; for (i <- 1 to y) p *= x; p
     | }
pow: (x: Double, y: Int)Double

scala> pow(2, 5)
res1: Double = 32.0
```

Hmm but using immutable data is more of a challenge. However, it looks like we can rewrite this version using tail recursion, where the final line in the function is a value passed to the next recursive call. No actual mutable data will be necessary.

```
scala> @annotation.tailrec
     | def pow(x: Double, y: Int, accum: Double = 1): Double = {
     |   if (y < 1) accum
     |   else pow(x, y - 1, accum * x)
     | }
pow: (x: Double, y: Int, accum: Double)Double

scala> pow(2, 5)
res2: Double = 32.0
```

6) Write a function that calculates the difference between a pair of 2d points (x and y) and returns the result as a point. Hint: this would be a good use for tuples ([tuples_section]).

Answer

A 2-sized tuple is a good parameter type for specifying a point in 2d space. It's components are unnamed, but we can imagine the first component as corresponding to x (the horizontal value) and its second as corresponding to y (the vertical value).

```
scala> def offset(src: (Int, Int), dest: (Int, Int)): (Int, Int) = {
     |   (dest._1 - src._1, dest._2 - src._2)
     | }
offset: (src: (Int, Int), dest: (Int, Int))(Int, Int)

scala> offset( (4, 9), (122, 27) )
res0: (Int, Int) = (118,18)
```

7) Write a function that takes a 2-sized tuple and returns it with the Int value (if included) in the first position. Hint: this would be a good use for type parameters and the isInstanceOf type operation.

Answer

This question should have stirred you to consider how you may change the type signature of a given tuple, and how changing the ordering of types in a tuple affects the return type of a function. Ultimately the resulting function must be non-type-safe, as the caller cannot expect to know in which order the types and values of a given tuple will be returned. Hopefully this solution is useful for learning the language, as it isn't really a recommended solution for actual coding (due to the lack of type safety).

```
scala> def intFirst[A,B](t: (A,B)): (Any,Any) = {
     |   def isInt(x: Any) = x.isInstanceOf[Int]
     |   (isInt(t._1), isInt(t._2)) match {
     |     case (false, true) => (t._2, t._1)
     |     case _ => t
     |   }
     | }
intFirst: [A, B](t: (A, B))(Any, Any)

scala> intFirst( ('a', 2) )
res0: (Any, Any) = (2,a)

scala> intFirst( (1, false) )
res1: (Any, Any) = (1,false)

scala> intFirst( (1, 4) )
res2: (Any, Any) = (1,4)
```

If the second item, but not the first item, is an integer than this function returns the tuple with the values switched. Otherwise it just returns the original tuple. Due to the reordering of the values, there is no way to alert the caller of the function what the types of each value will be. Thus the function is forced to return a tuple of the type (Any, Any).

8) Write a function that takes a 3-sized tuple and returns a 6-sized tuple, with each original parameter followed by its String representation. For example, invoking the function with (true, 22.25, "yes") should return (true, "true", 22.5, "22.5", "yes", "yes"). Can you ensure that tuples of all possible types are compatible with your function? When you invoke this function, can you do so with explicit types not only in the function result but in the value that you use to store the result?

Answer

The phrase "tuples of all possible types" indicates that we'll need to have type parameters for the input tuple. We can reuse them for the return tuple's tupe, since the return tuple should keep the same ordering of input types.

```
scala> def stringify[A,B,C](t: (A,B,C)):
(A,String,B,String,C,String) = {
     |   (t._1, t._1.toString, t._2, t._2.toString, t._3,
t._3.toString)
     | }
stringify: [A, B, C](t: (A, B, C))(A, String, B, String, C, String)
```

Let's invoke it and store the result in a value of an explicit type. The return type is based on interpolating the input tuple's types with String types so this should be easy to figure out.

```
scala> val t: (Int,String,Char,String,Boolean,String) = stringify(
(1, 'c', true) )
t: (Int, String, Char, String, Boolean, String) =
(1,1,c,c,true,true)
```