WORKING WITH XML

```
<?xml version="1.0"?>
<quiz>
 <qanda seq="1">
  <question>
   Who was the forty-second
   president of the U.S.A.?
  </question>
  <answer>
   William Jefferson Clinton
  </answer>
 </qanda>
 <!-- Note: We need to add
  more questions later.-->
</quiz>
```

XML

What is XML ?

XML is a form of semi-structured data.

It is more structured than plains strings, because it organizes the contents of the data into a tree

There are many forms of semi-structured data, but XML is the most widely used

XML Overview

XML is built out of two basic elements

01 Text

02 Tags

Text: As usual, any sequence of characters

Tags: Consist of a less-than sign, an alphanumeric label, and a greater than sign

Writing XML Tags

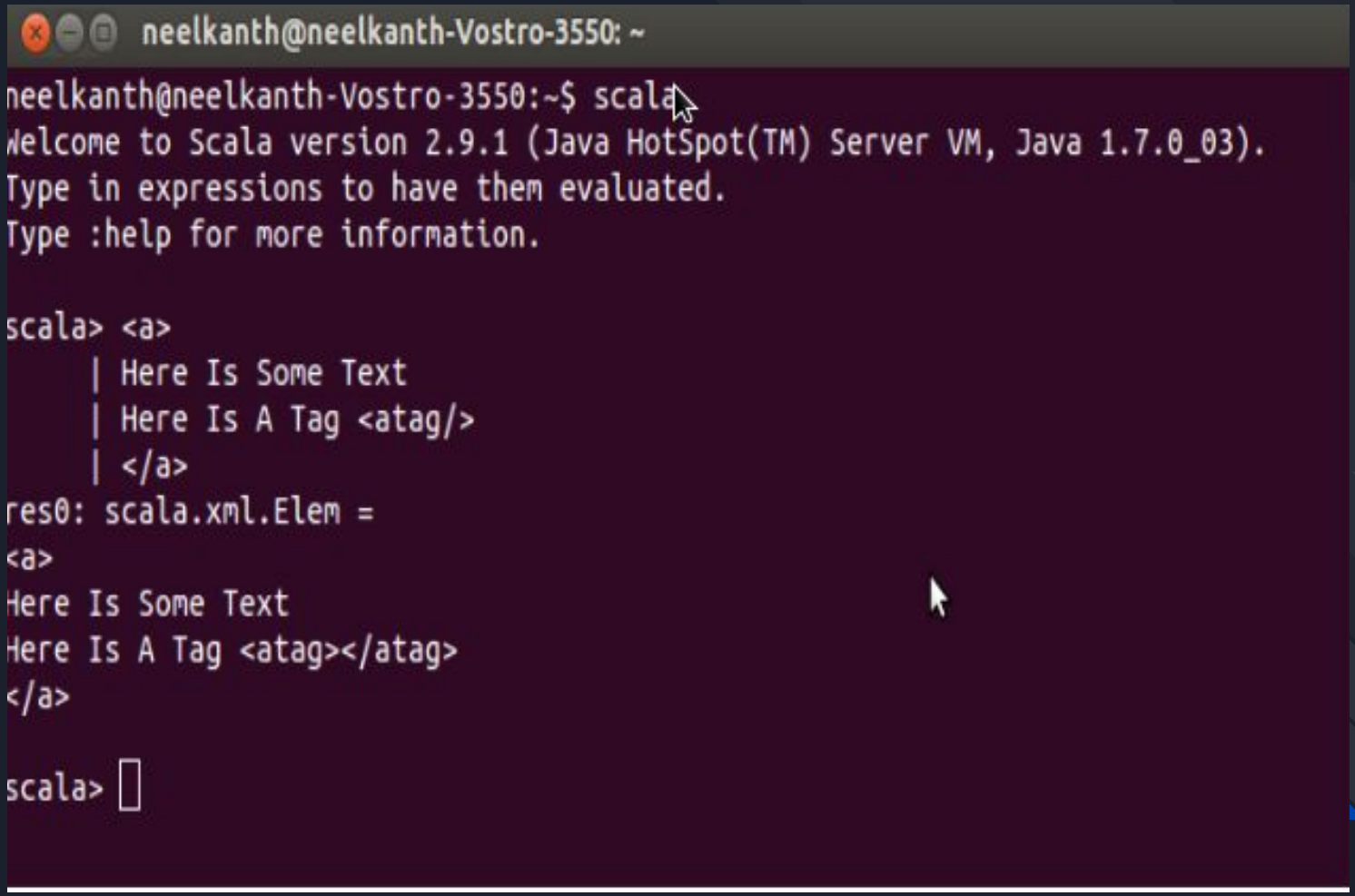
---There is a shorthand notation for a start tag followed immediately by its matching end tag.

---Simply write one tag with a slash put after the tag's label. Such a tag comprises an empty element.

e.g <pod>Three <peas/> in the </pod>

---Start tags can have attributes attached to them.

e.g <pod peas="3" strings="true"/>


neelkanth@neelkanth-Vostro-3550: ~
neelkanth@neelkanth-Vostro-3550:~$ scala
Welcome to Scala version 2.9.1 (Java HotSpot(TM) Server VM, Java 1.7.0_03).
Type in expressions to have them evaluated.
Type :help for more information.
scala> <a>
| Here Is Some Text
| Here Is A Tag <atag/>
|
res0: scala.xml.Elem =
<a>
Here Is Some Text
Here Is A Tag <atag></atag>

scala>

Important XML Classes

Class **Node** is the abstract superclass of all XML node classes.

Class **Text** is a node holding just text. For example, the “Here” part of **<a>Here** is of class Text.

Class **NodeSeq** holds a sequence of nodes.

Evaluating Scala Code

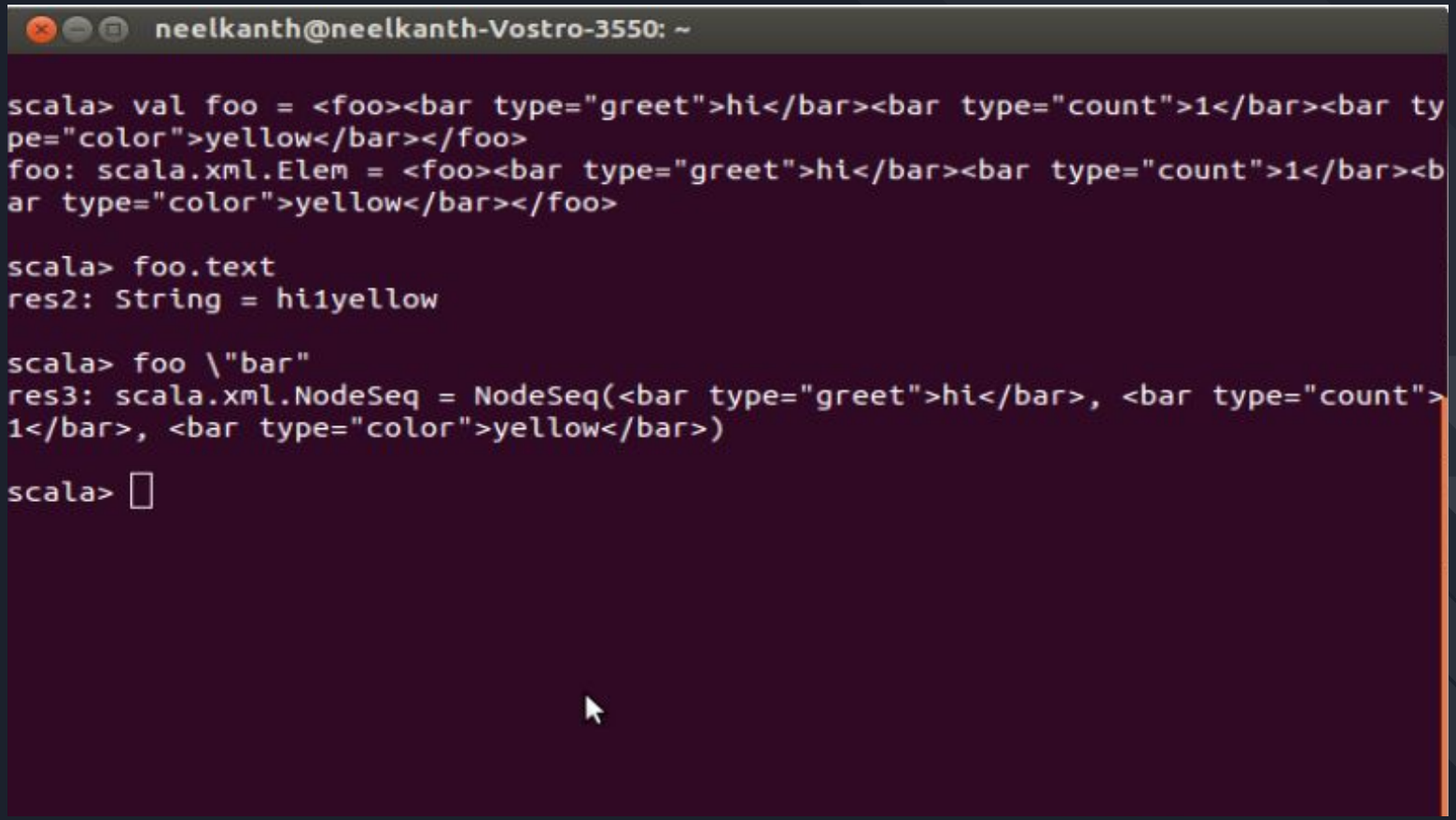
```
scala> <a> {"hello"+", world"} </a>
res4: scala.xml.Elem = <a> hello, world </a>

scala> val yearMade = 1955
yearMade: Int = 1955

scala> <a> { if(yearMade<2000) <old> { yearMade } </old> else xml.NodeSeq.Empty } </a>
res5: scala.xml.Elem = <a> <old> 1955 </old> </a>

scala>
```

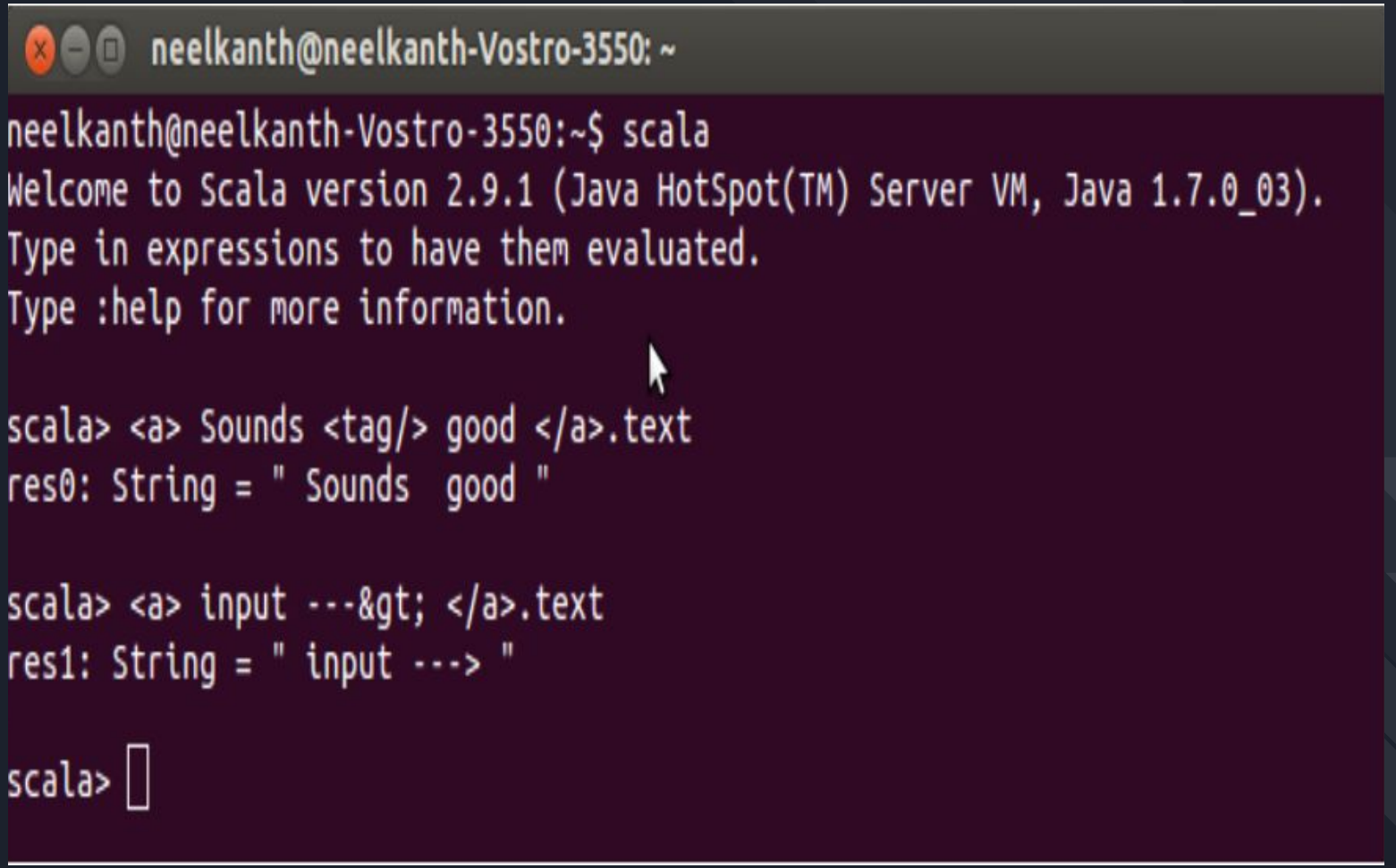
Example of XML



Taking XML apart

Extracting text :

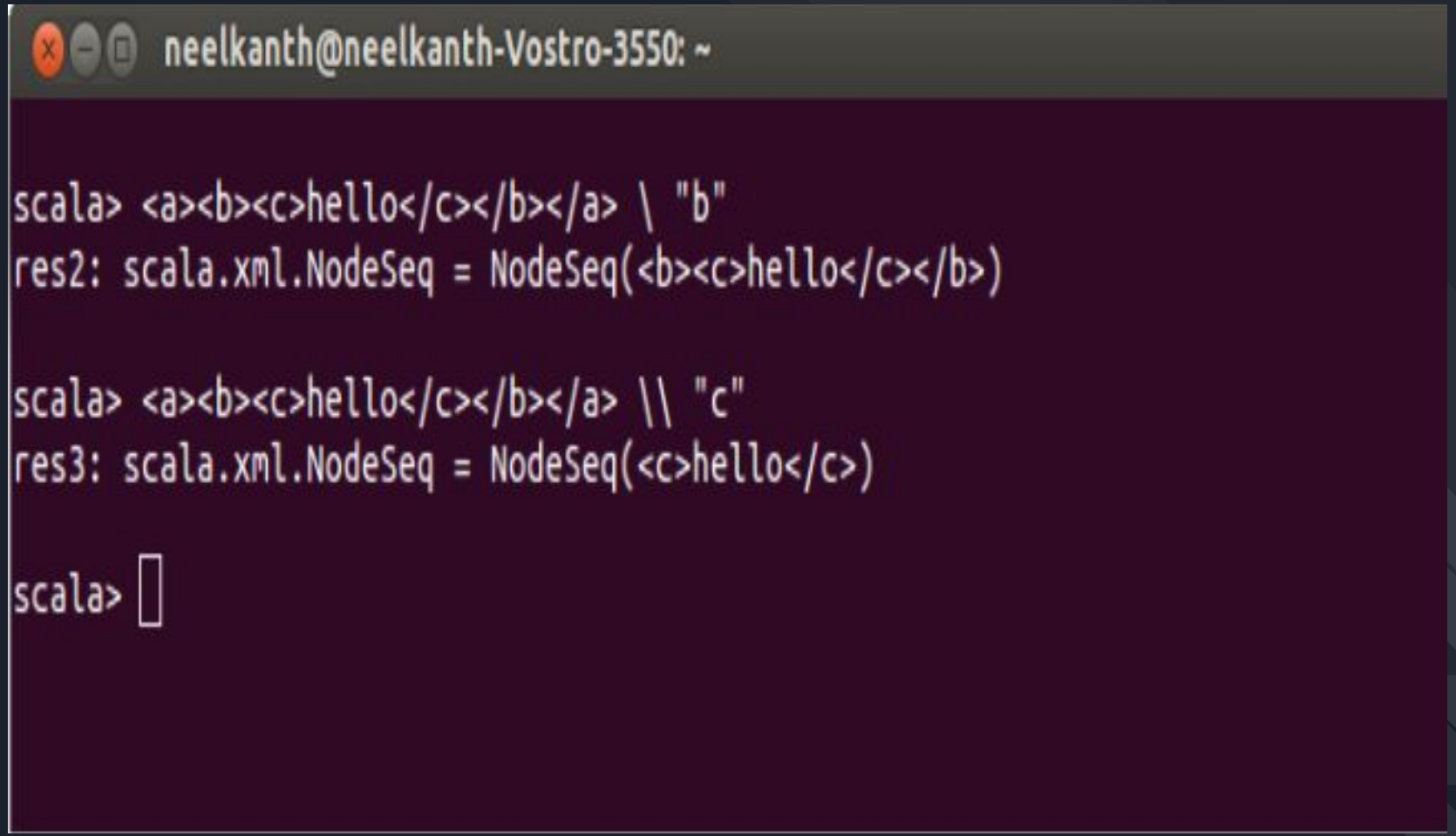
By calling the text method on
any XML node you retrieve all of the text within
that node, minus any element tags.

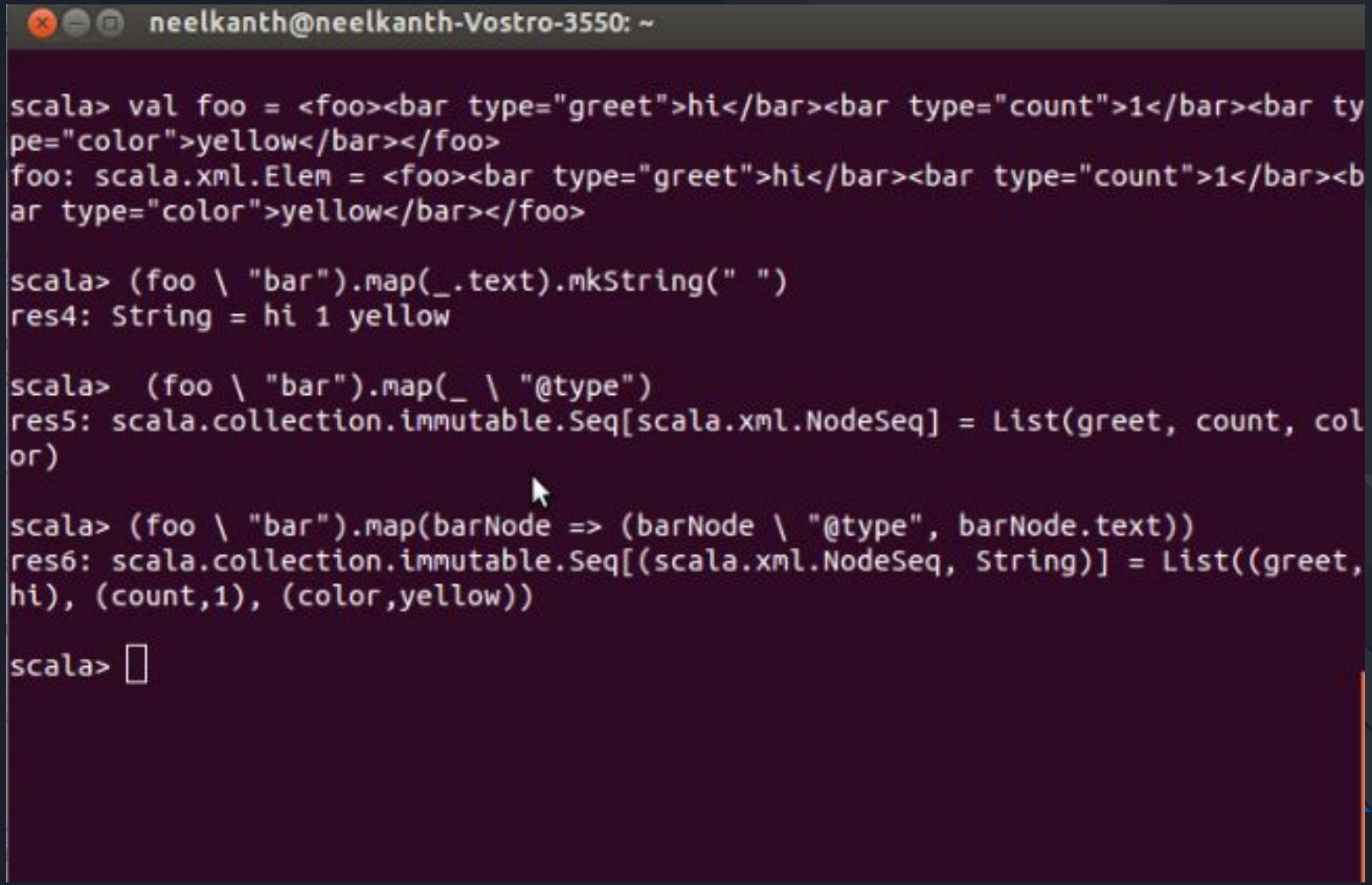

neelkanth@neelkanth-Vostro-3550: ~
neelkanth@neelkanth-Vostro-3550:~$ scala
Welcome to Scala version 2.9.1 (Java HotSpot(TM) Server VM, Java 1.7.0_03).
Type in expressions to have them evaluated.
Type :help for more information.
scala> <a> Sounds <tag/> good .text
res0: String = " Sounds good "
scala> <a> input ---> .text
res1: String = " input ---> "
scala>

Extracting sub-elements

If you want to find a sub-element by tag name, simply call \ with the name of the tag:

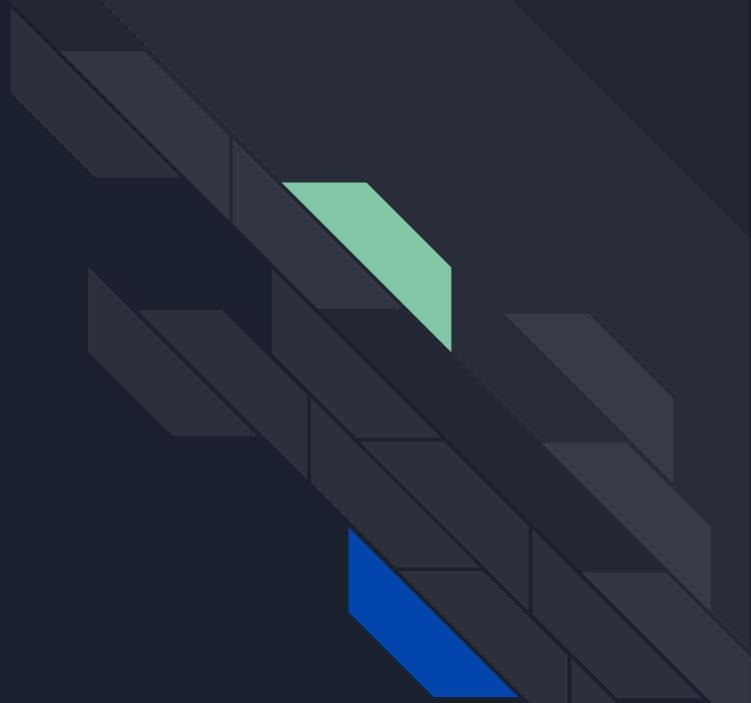
You can do a “deep search” and look through sub-sub-elements, etc., by using \\ instead of the \ operator.

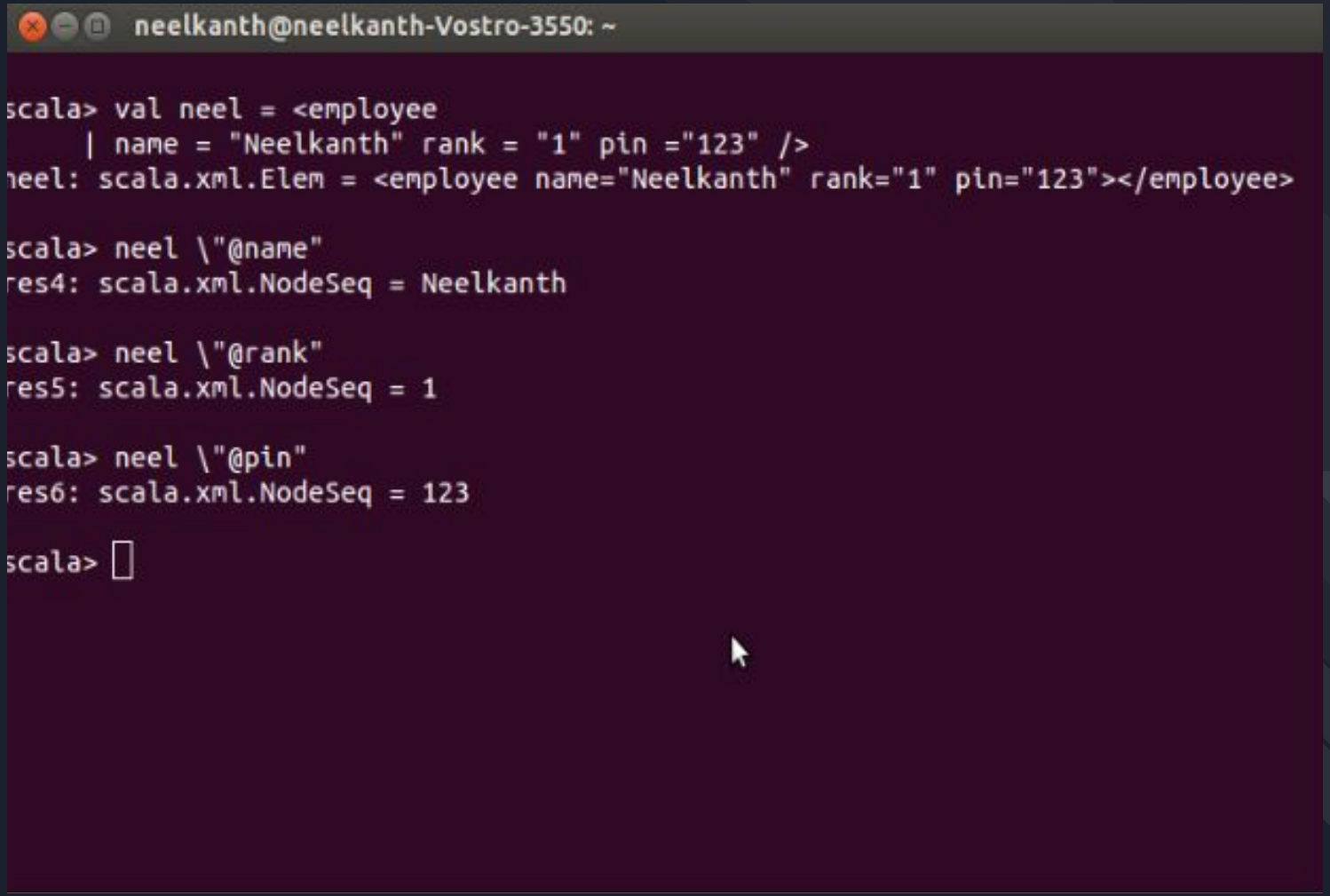

neelkanth@neelkanth-Vostro-3550: ~
scala> <a><c>hello</c> \ "b"
res2: scala.xml.NodeSeq = NodeSeq(<c>hello</c>)
scala> <a><c>hello</c> \\ "c"
res3: scala.xml.NodeSeq = NodeSeq(<c>hello</c>)
scala>


neelkanth@neelkanth-Vostro-3550: ~
scala> val foo = <foo><bar type="greet">hi</bar><bar type="count">1</bar><bar ty
pe="color">yellow</bar></foo>
foo: scala.xml.Elem = <foo><bar type="greet">hi</bar><bar type="count">1</bar><b
ar type="color">yellow</bar></foo>
scala> (foo \ "bar").map(_.text).mkString(" ")
res4: String = hi 1 yellow
scala> (foo \ "bar").map(_ \ "@type")
res5: scala.collection.immutable.Seq[scala.xml.NodeSeq] = List(greet, count, col
or)
scala> (foo \ "bar").map(barNode => (barNode \ "@type", barNode.text))
res6: scala.collection.immutable.Seq[(scala.xml.NodeSeq, String)] = List((greet,
hi), (count,1), (color,yellow))
scala>

Extracting attributes

You can extract tag attributes using the same \ and \\ methods. Simply put an at sign **(@)** before the attribute name:




neelkanth@neelkanth-Vostro-3550: ~
scala> val neel = <employee
| name = "Neelkanth" rank = "1" pin ="123" />
neel: scala.xml.Elem = <employee name="Neelkanth" rank="1" pin="123"></employee>
scala> neel \"@name"
res4: scala.xml.NodeSeq = Neelkanth
scala> neel \"@rank"
res5: scala.xml.NodeSeq = 1
scala> neel \"@pin"
res6: scala.xml.NodeSeq = 123
scala>

Runtime Representation

XML data is represented as labeled trees.

You can conveniently create such labeled nodes using standard XML syntax.

Consider the following XML document:

```
<html>
    <head>
        <title>Hello XHTML world</title>
    </head>
    <body>
        <h1>Hello world</h1>
        <p><a href="http://scala-lang.org/">Scala</a> talks XHTML</p>
    </body>
</html>
```

This document can be created by the following Scala program as :

```
object XMLTest1 extends Application {
    val page =
    <html>
        <head>
            <title>Hello XHTML world</title>
        </head>
        <body>
            <h1>Hello world</h1>
            <p><a href="scala-lang.org">Scala</a> talksXHTML</p>
        </body>
    </html>;
    println(page.toString())
}
```

It is possible to mix Scala expressions and XML :

```
object XMLTest2 extends Application {
        import scala.xml._
        val df = java.text.DateFormat.getDateInstance
        val dateString = df.format(new java.util.Date)
        def theDate(name: String) =
        <dateMsg addressedTo={ name }>
                Hello, { name }! Today is { dateString }
        </dateMsg>;
        println(theDate("Neelkanth Sachdeva").toString)
}
```