1. **Expressions which return/does not return value- [11/09/’15]**

val x = 2

Output

x: Int = 2

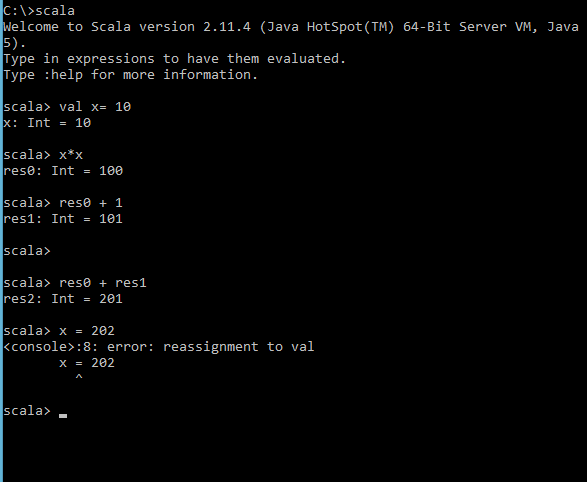
val x = println(2)

Output

2x: Unit = ()

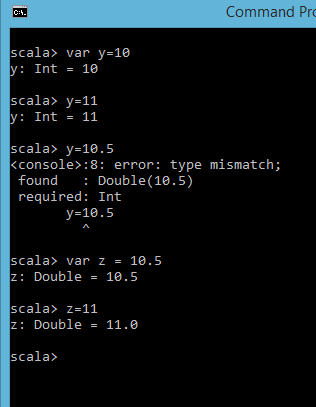
1. **Variables (var,val,lazy val)**

***Val -*** immutable variable

****

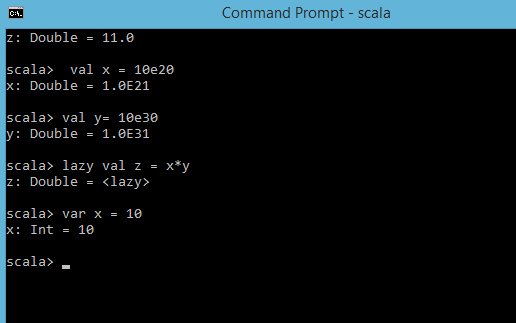
***Var*** *- mutable variable*

* We can Reassign a new value to y as y is mutable, but you cannot reassign the variable to a different type.
* Defining a variable of type Double and assigning it an Int value will work because Int numbers can be converted to Double numbers automatically.



***Lazy val*** –

* Lazy val variables are calculated once, the first time the variable is accessed.
* Only vals can be lazy variables.
* You would use a lazy val if the variable may not be used and the cost of calculating it is very long.



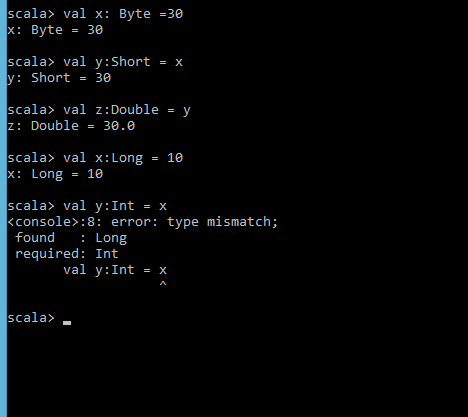
1. **Any, AnyVal and AnyRef Types**

* Class Any is the root of the Scala class hierarchy and is an abstract class. Every class in a Scala execution environment inherits directly or indirectly from this class.
* AnyVal and AnyRef extend Any type.
* The Any, AnyVal, and AnyRef types are the root of Scala’s type hierarchy. All other types descend from AnyVal and AnyRef.
* The types that extend AnyVal are known as value types.
* The types that extend AnyRef are known as reference types (i.e., nonvalue Scala classes and user-defined) classes.

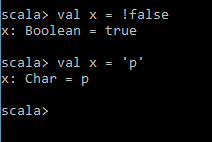
***Anyval***

* **Numeric Types -** The numeric data types in Scala constitute Float and Double types along with Integral data types such as Byte, Short, Int, Long, and Char.

*Scala supports the ability to automatically convert numbers from one type to another in the order Byte ➤ Short ➤ Int ➤ Long ➤ Float ➤ Double.*

**

* **Boolean Type** - The Boolean type is limited to the literal true or the literal false.
* **Char Type** - Char literals are written with single-quotes.

**

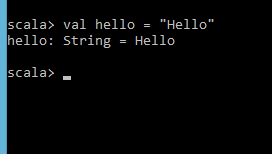
* **Unit type** - The Unit type is used to define a function that doesn’t return data. It is similiar to the void keyword in Java.(not worked)

*def main(args: Array[String]) : Unit = { }*

Output-

scala> *val empty = () empty: Unit = ()*

* **Strings**  - Scala’s String is built on Java’s String and adds additional features such as string interpolation to Java’s String.

**

***Anyref***

* Nothing and Null types are at the bottom of the Scala type hierarchy.
* Null is a subtype of all reference types that is, it is a subtype of all AnyRef types that exist to provide a type for the keyword null.
* One of the usages of Nothing is that it signals abnormal termination.
* Nothing is a trait that is guaranteed to have zero instances. It provides a return type for methods that never return normally.
* In best practiced Scala, you never use null values. If you’re coming from a language like Java, any time you feel like using a null, use an Option instead.

1. **String interpolation**

* String interpolation is a mechanism to combine your values inside a string with variables.
* The notation for interpolation in Scala is an s prefix added before the first double quote of the string. Then dollar sign operator $ can be used to reference the variable.

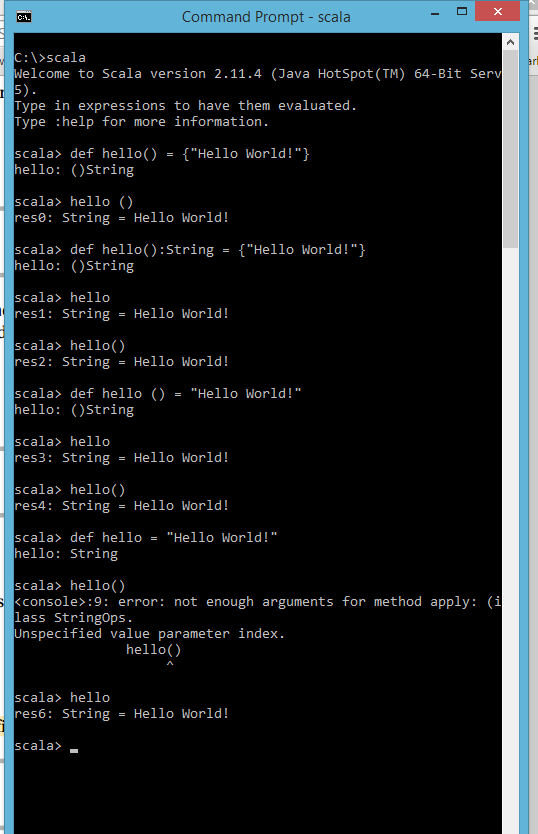
****

1. **Functions**

* Scala has both functions and methods.
* A Scala method is a part of a class that has a name and a signature, whereas a function in Scala is a complete object that can be assigned to a variable.
* A function definition can appear anywhere in a source file.

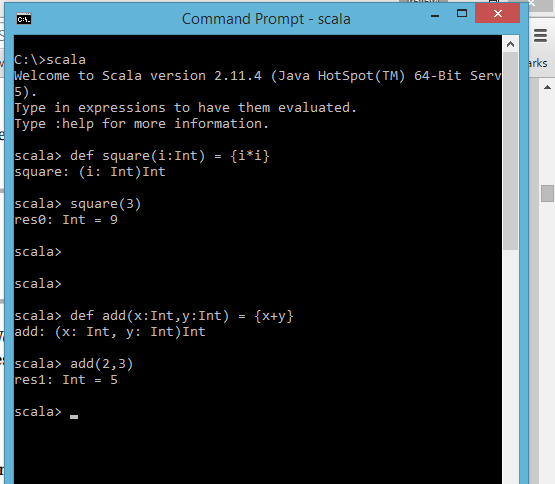
**Function without Parameter** – different scenarios

* Hello Function.
* Optional Return Type in the Method Definition.
* Method Body Without the Parentheses.
* Method Signature Without Parentheses.

****

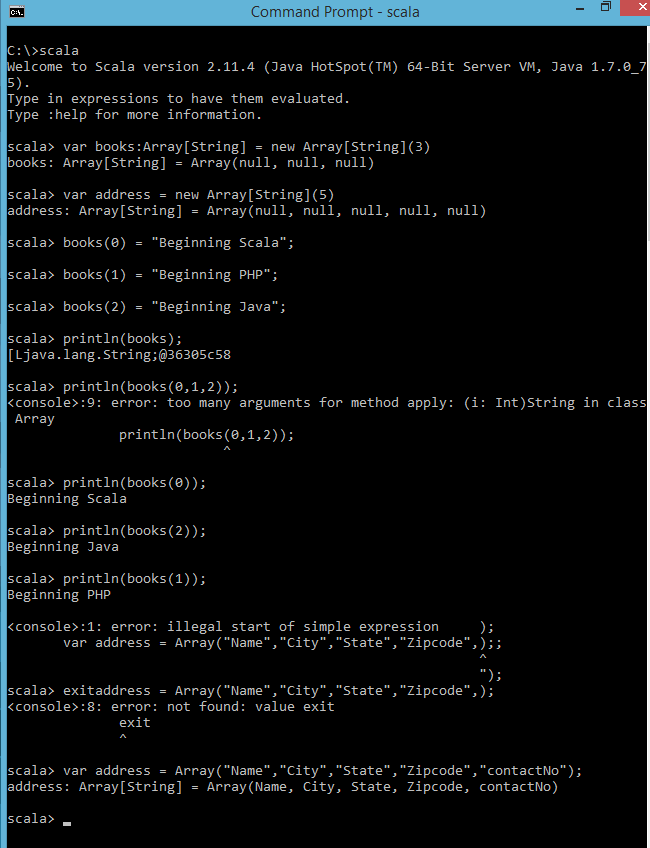
**Function with Parameters**– different scenarios

* Function with Parameter.
* Function with Multiple Parameters.

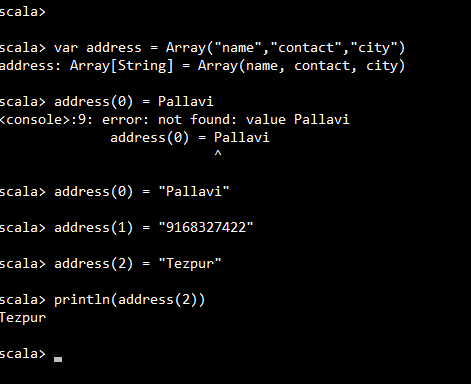
****

1. **Arrays**

* The array is a common data structure consisting of a collection of elements of the same type.
* Basically, there are two ways to define an array: either you can specify the total number of elements and then assigns values to the elements, or you can specify all values at once.

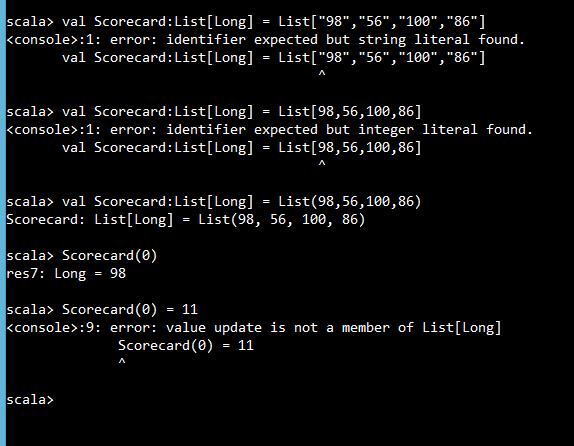
****

Note- “;” is not required in scala



1. **Lists [14/9/’15]**

* Unlike arrays, elements of a list cannot by changed by assignment. The list that has elements of type T is written as List[T].

****

* There are two ways to create a list: either you can create the list in the similar manner you create arrays or you can use :: cons operator.

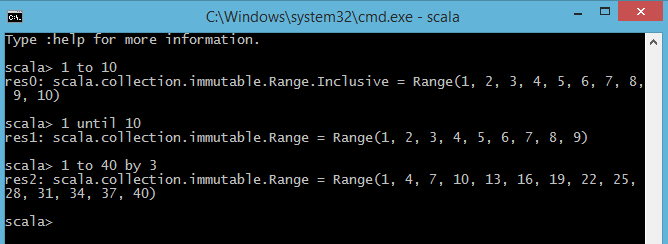
|  |  |  |
| --- | --- | --- |
|  | **Creating an empty list** | **Creating a list of books** |
| **Method 1** | scala> val empty: List[Nothing] = List()  empty: List[Nothing] = List()  scala>  ***Note-*** *The type of list is Nothing.* | scala> val books: List[String] = List("Beginning Scala", "Beginning Groovy", "Beginning Java")  books: List[String] = List("Beginning Scala", "Beginning Groovy", "Beginning Java")  scala>  **Note-** The type of list is String. |
| **Method 2** | scala> val empty = Nil  empty:scala.collection.immutable.Nil.type = List()  scala>  ***Note****- Using Nil* | scala> val books = " Beginning Scala" :: ("Beginning Groovy" :: ("Beginning Java" :: Nil))  books: List[String] = List(" Beginning Scala", Beginning Groovy, Beginning Java)  scala>  **Note-** Using a Tail Nill and ::  Accessing head and tail element->  scala> books.head  res0: String = " Beginning Scala"  scala> books.tail  res1: List[String] = List(Beginning Groovy, Beginning Java)  scala> |

* The operations on the lists can be expressed in terms of head and tail methods,

Where **HEAD** returns the first element of a list and **TAIL** returns a list consisting of all elements except the first element.

1. **Ranges**

Method “to” Method “Until” Method “By”



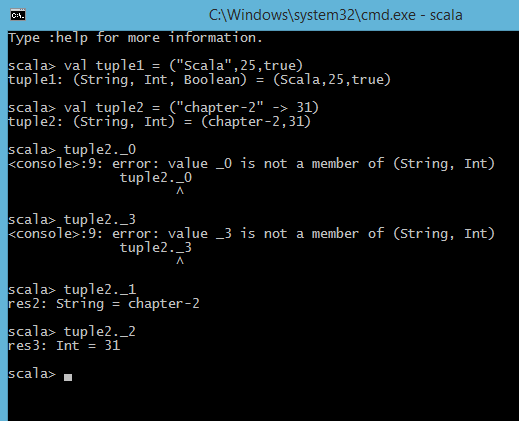
Note :- Ranges can be defined by their start, their end, and the stepping value.

1. **Tuples**

* A tuple is an ordered container of two or more values of same or different types.
* Unlike lists and arrays, however, there is no way to iterate through elements in a tuple.
* Tuples are useful when you need to group discrete elements and provide a generic means to structure data.
* Tuple creation

Writing your values separated by a comma and surrounded by a pair of parentheses

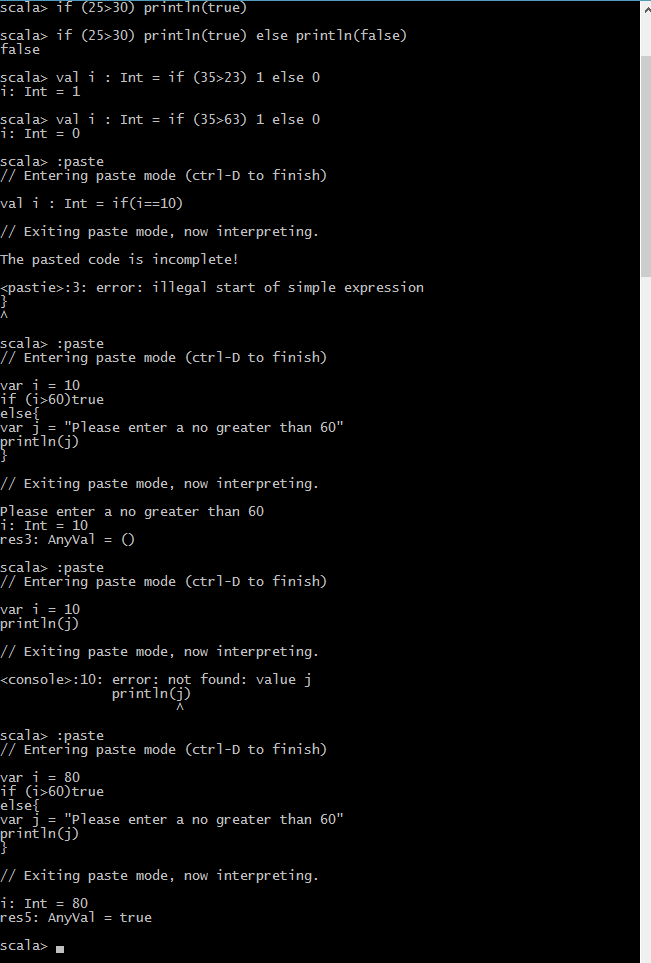
Using a relation operator (->)



Built-in Control Structures

1. **“If” Expressions**

* The result of if expressions in Scala is always Unit.
* The result of if/else is based on the type of each part of the expression.

****

1. **“While” Expressions**

* The while and dowhile constructs are called loops, not expressions, because they don’t result in any values.
* The type of the result is Unit.
* It is used very rarely in Scala code. It turns out that a value (and in fact, only one value) exists whose type is Unit.

Syntax-

while (exp) println("Working...")

while (exp) {

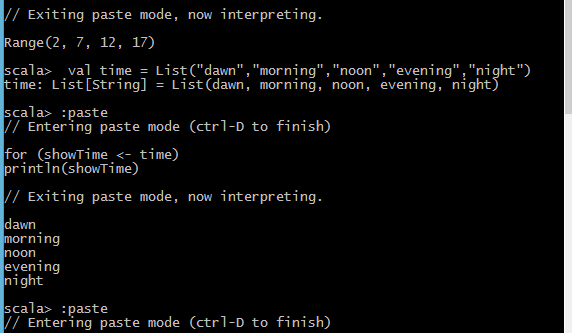
println("Working...")

}

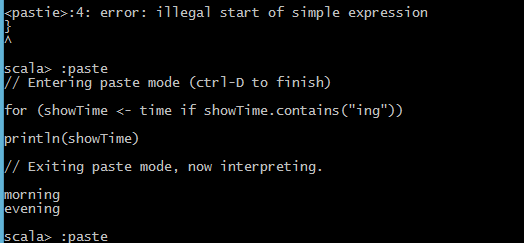
1. **“For” Expressions**

* “For” Comprehension is a very powerful control structure of Scala language.
* Not only does it offer the ability to iterate over a collection, but also provides filtering options and the ability to generate new collections.

Basic “for”

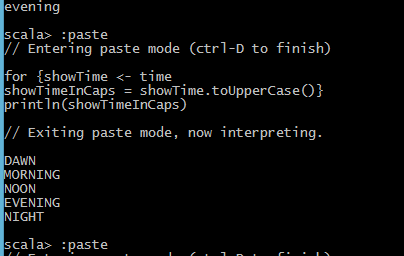


Defining “Filters” using “for”



Note - A filter is an if clause inside the “for” expression that is used to filter the collection when you do not want to iterate through the entire collection.

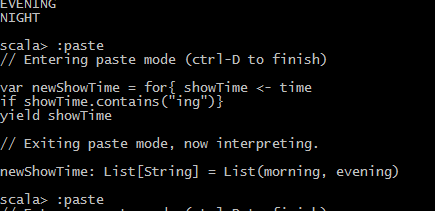
Variable Binding in “For” Comprehension



Note - You can define variables inside “for” expression which can be re-used within the body of your “for” expression.

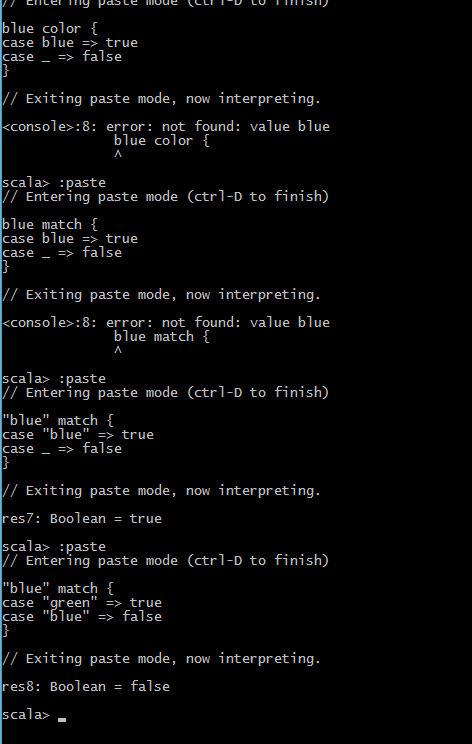
Using Yield in “For” Comprehension

* In Scala’s “for” expression you can use the yield keyword to generate new collections.
* The type of the collection generated from the “for” expression is inferred from the type of the collection being iterated over.
* The filtered result is yielded as a value named book. This result is accumulated with every run inside the “for” loop, and thus accumulated collection is assigned to the value scalabooks.



1. **“Match” Expression**

* Scala’s match expressions are used for pattern matching by allowing you to construct complex tests in very little code and are extremely flexible constructs that enable matching arbitrary items, such as types, content of the data structure, regular expressions, and even ranges of numeric types.
* Pattern matching is like Java’s switch statement, but you can test against almost anything, and you can even assign pieces of the matched value to variables.
* Like everything in Scala, pattern matching is an expression, so it results in a value that may be assigned or returned.



1. **Comments in scala**

* /\* This is a multiline comment. \*/
* // This is a single line comment

1. **Try/catch & Try/finaly**

* All exceptions in Scala are unchecked; there is no concept of checked exception.
* Try/catch in Scala is an expression that results in a value
* The exception in Scala can be pattern matched in the catch block instead of providing a separate catch clause for each different exception.
* Because try/catch in Scala is an expression, it becomes possible to wrap a call in a try/catch and assign a default value if the call fails.

Example

*try {*

*file.write(stuff)*

*} catch{*

*case e:java.io.IOException => // handle IO Exception*

*case n:NullPointerException => // handle null pointer*

*}*

Wrapping a Call in try/catch and Assigning a Default if the Call Fails

scala> try{Integer.parseInt("dog")}catch{case\_ => 0}

Output

res16:Int = 0

When the Call Does not Fail

scala> try{Integer.parseInt("44")} catch{case \_ => 0}

Output

res17:Int = 44