Welcome to Scala

Scala - Scalable Language

- Scala is a modern multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and type-safe way
- Integrates the features of object-oriented and functional languages

Why Scala?

- Statically Typed
    - Proven correctness before deployment
    - Performance
- Lightweight Composable Syntax
    - Low boilerplate
    - Syntax is very similar to other data centric languages
- Stable
    - Is being used in enterprises, financial sectors, retail, gaming and more for many years

Scala - History

2011	Corporate stewardship
2005	Scala 2.0 written in Scala
2003	First Experimental Release
2001	Decided to create even better Java
1990	Martin Odersky (Creator of Scala) made Java better via generics in the "javac" compiler

Scala - JVM

- Scala source code gets compiled to Java byte code
- Resulting bytecode is executed by a JVM - the Java Virtual Machine
- Java libraries may be used directly in Scala code and vice versa

Scala - Hello World - Example

```
// Create a file hello_world.scala

// using nano hello_world.scala

object HelloWorld {

    def main(args: Array[String]) {

    println("Hello, world!")

    }

}
```

Scala - Hello World - Compiling and Running

- Compile using scalac

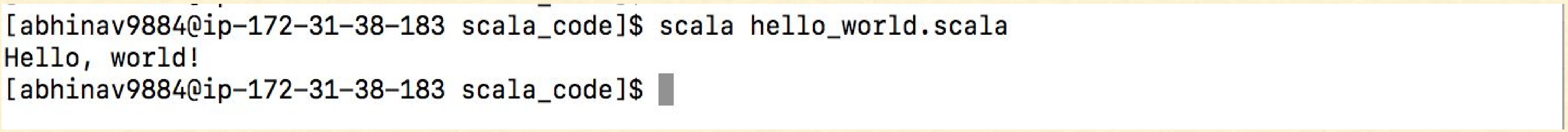
  scalac hello_world.scala

- To run it

  scala HelloWorld

Scala - Interpreter

- *scala hello_world.scala*



```
[abhinav9884@ip-172-31-38-183 scala_code]$ scala hello_world.scala
Hello, world!
[abhinav9884@ip-172-31-38-183 scala_code]$ 
```

Scala - Interpreter

To evaluate a single expression, you can use the following:

```
scala -e 'println("Hello, World!")'
```

Scala - Variables

- Variables are reserved memory locations to store values
- Compiler allocates a memory based on the data type of the variable

Scala - Variables - Types

- Mutable
- Immutable

Scala - Variables - Types

- Mutable variables are defined using “var” keyword
    - values can be changed
- Immutable variables are defined using “val” keyword
    - values can not be changed after assignment

Scala - Variables - Hands-on

- val x: Int = 8 // Immutable, Read Only
- var y: Int = 7 // Mutable

Scala - Immutable Variables - Importance

What is the importance of immutable variables?

- In large systems, we do not want to be in a situation where a variable's value changes unexpectedly
- In case of threads, APIs, functions and classes, we may not want some of the variables to change

Scala - Variables - Type Inference

- We can define the variables without specifying their data type
- Scala compiler can understand the type of the variable based on the value assigned to it

Scala - Variables - Type Inference

- var x = "hi" // Type Inference (Scala compiler determines the type)
- var x: String = "hi" // Explicitly specifying the type

Scala - Variables - Type Inference

- We should always define type of variables explicitly
    - Code will be compiled faster as compiler will not have to spend time in guessing the type
    - No ambiguity

Scala - Classes

- A class is a way of creating your own data type
- We can create a data type that represents a customers using a class
- A customer might have states like ***first name***, ***last name***, ***age***, ***address*** and ***contact number***
- A customer class might represent behaviours for how these states can be transformed like how to change someone's address or name
- A class is not concrete until it has been instantiated using a ***new*** keyword

Scala - Classes - Hands-on

```
class Person(val fname: String, val lname: String, val anage: Int) {
    val firstname:String = fname;
    val lastname: String = lname;
    val age = anage;
}
val obj = new Person("Robin", "Gill", 42);
print(obj.age)
print(obj.firstname)
```

Scala - Classes - Methods - Hands-on

```
class Person(val fname: String, val lname: String, val anage: Int) {
    val firstname:String = fname;
    val lastname: String = lname;
    val age = anage;
    def getFullName():String = {
        return firstname + " " + lastname;
    }
}
val obj = new Person("Robin", "Gill", 42);
print(obj.getFullName())
```

Scala - Singleton Objects

- A singleton is a class which can have only one instance at any point in time
- Methods and values that aren't associated with individual instances of a class belong in singleton objects
- A singleton class can be directly accessed via its name
- Create a singleton class using the keyword ***object***

Scala - Singleton Objects

```
object Hello {
    def message(): String = {
        return "Hello!!!";
    }
}
```

Scala - Singleton Objects

- Objects are useful in defining utility methods and constants as they are not related to any specific instance of the class

Scala - Functions - Representations

1.
```
def add(x : Int, y : Int) : Int = {
    return x + y
}
```

2.
```
def add(x : Int, y : Int) = { //return type is inferred
    x + y //"return" keyword is optional
}
```

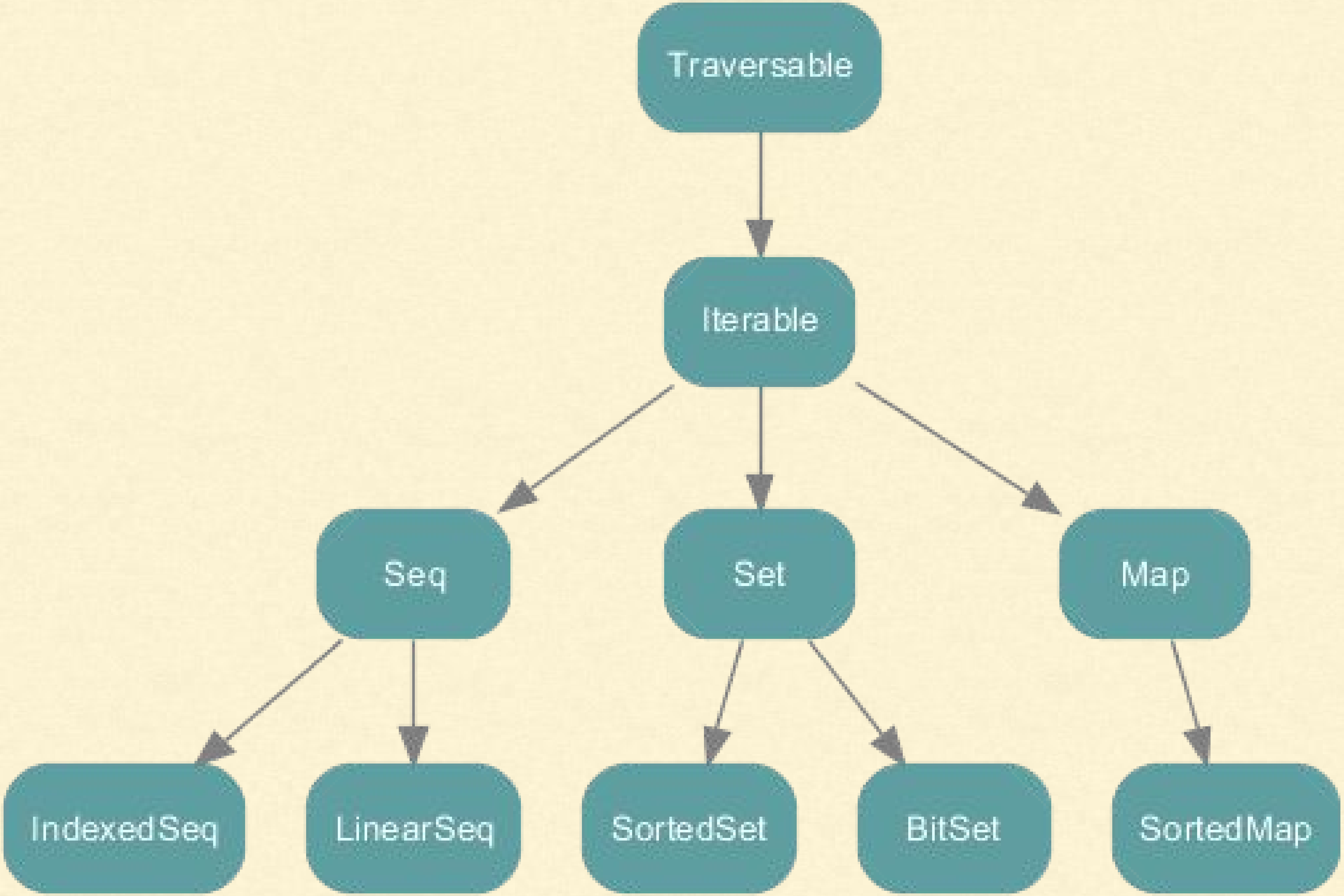
3.
```
//Curly braces are optional on single line blocks

def add(x : Int, y : Int) = x + y
```

Scala - Collections - Overview

- Collections provide different ways to store data
- Scala collections hierarchy represents a structure of collections that can contain data in different ways

Scala - Collections - Overview



Scala - Collections - Sequence

- An ordered collection of data
- May or may not be indexed
- Array, List, Vector

Scala - Collections - Sequence - Array

- Contains elements of same type
- Fixed size, ordered sequence of data
- Values are contiguous in memory
- Indexed by position

Scala - Collections - Sequence - Array

- *var languages = Array("Ruby", "SQL", "Python")*

- *languages(0)*

- *languages(1) = "C++"*

- *// Iterate over array*

  ```
  for(x <- languages) {
      println(x);
  }
  ```

Scala - Collections - Sequence - List

- List represents linked list with a value and a pointer to the next element
- Poor performance as data could be anywhere in the memory
- Theoretically unbounded in size

Scala - Collections - Sequence - List

- *var number_list = List(1, 2, 3)*
- *number_list :+ 4*

Scala - Collections - Sets

- Bag of data with no duplicates
- Order is not guaranteed

Scala - Collections - Sets

- *var set = Set(76, 5, 9, 1, 2);*
- *set = set + 9;*
- *set = set + 20;*
- *set(5);*
- *set(14);*

Scala - Collections - Tuples

- Unlike array or list, tuples can hold elements of different data types
- Example *var t = (14, 45.69, "Australia")* or
- *var t = Tuple3(14, 45.69, "Australia")* //Same result
- Can be accessed using a 1-based accessor for each value

Scala - Collections - Tuples

- *t._1 // 14*
- *t._3 // Australia*
- Can be deconstructed into names bound to each value in the tuple
- Tuples are immutable

Scala - Collections - Maps

- Collection of key/value pairs
- Allows indexing values by a specific key for fast access
- Java HashMap, Python dictionary

Scala - Collections - Maps

- var colors = Map("red" -> "#FF0000", "yellow" -> "#FFFF00")

- colors("yellow")

- colors += "green" -> "#008000"

- colors -= "red"

-
  ```
  for ((key,value) <- colors) {
      printf("key: %s, value: %s\n", key, value)
  }
  ```

Scala - Collections - Maps

- Two types of maps
    - Mutable
    - Immutable
- By default, Scala uses the immutable map
- For mutable maps explicitly import
    - *import scala.collection.mutable.Map*

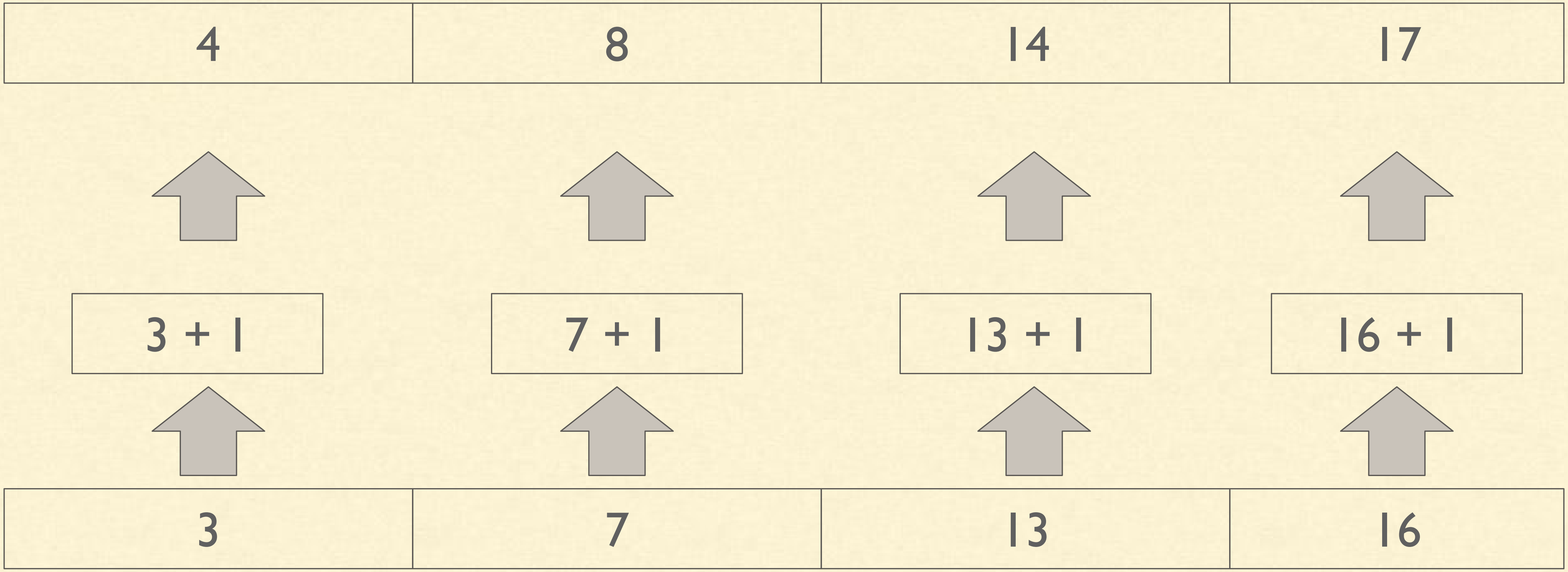
Scala - Higher Order Functions

- Higher order function is a function which takes another function as an argument
- Describes “how” the work to be done in a collection

Scala - Higher Order Functions - Map

- The map applies the function to each value in the collection and returns a new collection

Scala - Higher Order Functions - Map



4 | 8 | 14 | 17

3 + 1 | 7 + 1 | 13 + 1 | 16 + 1

3 | 7 | 13 | 16

Scala - Higher Order Functions - Map

- *var list = List(1,2,3)*
- *list.map(x => x + 1) // List(2, 3, 4)*
- *list.map(_+ 1) // Same output*

Scala - Higher Order Functions - flatMap

- flatMap takes a function as an argument and this function must return a collection
- Returned collection could be empty
- The flatMap applies the function passed to it as argument on each value in the collection just like map
- The returned collections from each call of function are then merged together to form a new collection

Scala - Higher Order Functions - flatMap

- *var list = List("Python", "Go")*
- *list.flatMap(lang => lang + "#") // List(P, y, t, h, o, n, #, G, o, #)*
- *list.flatMap(_+ "#") // Same output*

Scala - Higher Order Functions - Filter

- Filter applies a function to each value in the collection and returns a new collection with values that satisfy a condition specified in the function

Scala - Higher Order Functions - Filter

- *var list = List("Scala", "R", "Python", "Go", "SQL")*
- *list.filter(lang => lang.contains("S")) // List(Scala, SQL)*

Scala - Higher Order Functions - foreach

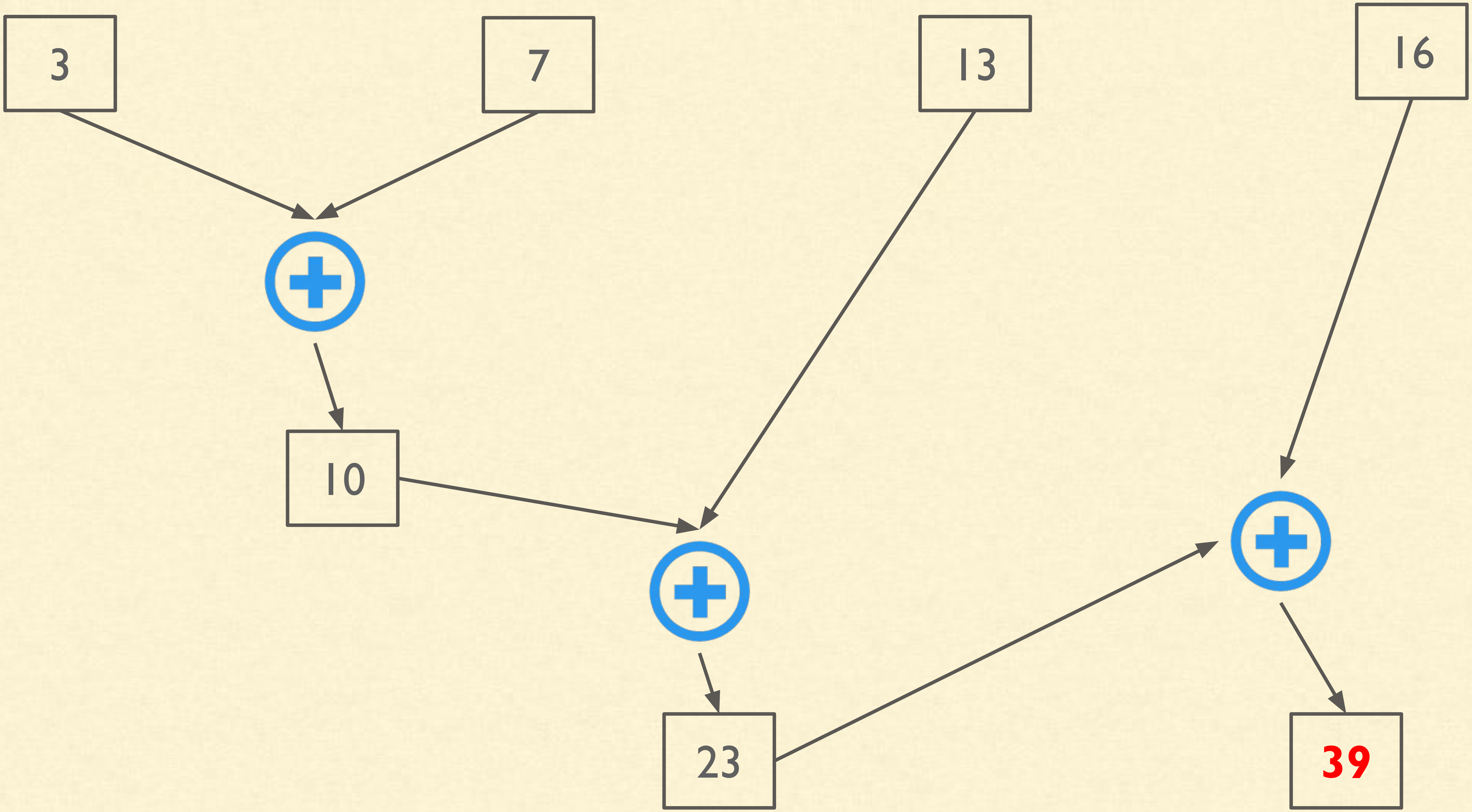
- Each of the previous higher order functions returned a new collection after applying the transformation
- At times we do not want the functions to return a new collection
    - Waste of memory resources on JVM if we do not want a return value
- foreach applies a function to each value of collection without returning a new collection

Scala - Higher Order Functions - foreach

- *var list = List("Python", "Go")*
- *list.foreach(println)*

Scala - Higher Order Functions - Reduce

Scala - Higher Order Functions - Reduce



Scala - Higher Order Functions - Reduce

- In previous example, addition of two variables was called again and again until all the values got reduced to a single value

- Please note, in case of reduce input collection should contain elements of same data type

Scala - Higher Order Functions - Reduce

- *var list = List(3, 7, 13, 16)*
- *list.reduce((x, y) => x + y) // 39*
- *list.reduce(_ + _) // same*

Scala - Interaction with Java

```
import java.util.{Date, Locale}
import java.text.DateFormat
import java.text.DateFormat._
object USDate {
    def getDate(): String = {
        val now = new Date
        val df = getDateInstance(LONG, Locale.US)
        return df.format(now)
    }
}
```

Scala - Build Tool - SBT

- If we are working on a big project containing hundreds of source files, it becomes really tedious to compile these files manually
- We need a build tool to manage the compilation of all these files
- SBT is build tool for Scala and Java projects, similar to Java's Maven or ant

Scala - Build Tool - SBT - Demo

- Sample Scala project is located at CloudxLab GitHub repository

  https://github.com/singhabhinav/cloudxlab/tree/master/scala/sbt

- Clone the repository

  ```
  git clone https://github.com/singhabhinav/cloudxlab.git ~/cloudxlab
  ```

- *Or update the repository*

  ```
  cd ~/cloudxlab && git pull origin master
  ```

Scala - Build Tool - SBT - Demo

- *cd ~/cloudxlab/scala/sbt*
- Look at the build.sbt file and the directory layout. Scala files must be located at

  src/main/scala
- Run the project using

  sbt run

Scala - Case Classes

- Case classes are regular classes that are:
  - Immutable by default
  - Can be pattern matched
  - Compiler automatically generates hashCode and equals methods, so less boilerplate code
  - Helps in writing more expressive and maintainable code

Scala - Case Classes - Demo

- Sample Scala project is located at CloudxLab GitHub repository

  https://github.com/singhabhinav/cloudxlab/tree/master/scala/case_classes

- Clone the repository

  ```
  git clone https://github.com/singhabhinav/cloudxlab.git ~/cloudxlab
  ```

- *Or update the repository*

  ```
  cd ~/cloudxlab && git pull origin master
  ```

Scala - Case Classes - Demo

- *cd ~/cloudxlab/scala/case_classes*
- *scala case_class.scala* // Run case class demo
- *scala pattern_matching.scala* // Run pattern matching demo

Scala - End

Thank You