Scala TÜRKİYE

String Interpolation
Streams
Regular Expressions
Process Execution

presenter
Volkan Yazıcı
https://twitter.com/yazicivo
http://vyazici.blogspot.com
http://github.com/vy
http://web.itu.edu.tr/~yazicivo/

Implicit Classes simplify the creation of classes which provide extension methods to another type.

http://docs.scala-lang.org/sips/pending/implicit-classes.html

Implicit Classes

Basics

```
scala> implicit class MyString(val str: String) {
     |   def quote: String = "'" + str + "'"
     | }
defined class MyString

scala> println("abc")
abc

scala> println("abc".quote)
'abc'
```

Implicit Classes

Basics

```
scala> implicit class MyString(val str: String) {
     |   def quote: String = "'" + str + "'"
     | }
defined class MyString

scala> println("abc")
abc

scala> println("abc".quote)
'abc'

------------------------------------------------------------------------------------

scala> implicit class MyInt(val n: Int) {
     |   def factorial: Int = {
     |     require(n >= 0)
     |     if (n < 2) 1
     |     else n * (n-1).factorial
     |   }
     | }
defined class MyInt

scala> 5.factorial
res3: Int = 120
```

String Interpolation allows users to embed variable references directly in processed string literals.

String Interpolation

Basics

```
scala> val (str, num) = ("Walter White", 0.128F)
str: String = Walter White
num: Float = 0.128
```

String Interpolation

Basics

```
scala> val (str, num) = ("Walter White", 0.128F)
str: String = Walter White
num: Float = 0.128

-------------------------------------------------------------------------------------

scala> println(s"str: $str, num: $num")
str: Walter White, num: 0.128
```

String Interpolation

Basics

```
scala> val (str, num) = ("Walter White", 0.128F)
str: String = Walter White
num: Float = 0.128

---------------------------------------------------------------------------------------

scala> println(s"str: $str, num: $num")
str: Walter White, num: 0.128

---------------------------------------------------------------------------------------

scala> println(s"1 + 1 = ${1 + 1}”)
1 + 1 = 2

scala> println(s"2 * $num = ${2 * num}”)
2 * 0.128 = 0.256
```

String Interpolation

Basics

```
scala> val (str, num) = ("Walter White", 0.128F)
str: String = Walter White
num: Float = 0.128

------------------------------------------------------------------------------------

scala> println(s"str: $str, num: $num")
str: Walter White, num: 0.128

------------------------------------------------------------------------------------

scala> println(s"1 + 1 = ${1 + 1}”)
1 + 1 = 2

scala> println(s"2 * $num = ${2 * num}”)
2 * 0.128 = 0.256

------------------------------------------------------------------------------------

scala> println(f"num: $num%.2f")
Num: 0.13
```

String Interpolation

Basics

```
scala> val (str, num) = ("Walter White", 0.128F)
str: String = Walter White
num: Float = 0.128

-------------------------------------------------------------------------------------

scala> println(s"str: $str, num: $num")
str: Walter White, num: 0.128

-------------------------------------------------------------------------------------

scala> println(s"1 + 1 = ${1 + 1}”)
1 + 1 = 2

scala> println(s"2 * $num = ${2 * num}”)
2 * 0.128 = 0.256

-------------------------------------------------------------------------------------

scala> println(f"num: $num%.2f")
Num: 0.13

-------------------------------------------------------------------------------------

scala> println(raw"a\nb \u0040")
a\nb @
```

String Interpolation

Internals

Compiler transformation:

`id”string” ↔ StringContext(“string”).id()`

Existing `StringContext` methods: `s`, `f`, and `raw`.

We can extend `StringContext` using implicit classes.

String Interpolation

Extending

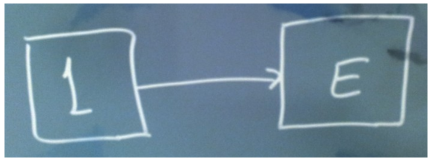
```
scala> implicit class MyStringContext(val sc: StringContext) {
     |   def q(args: Any*): String = s"args: '$args', parts: '$sc.parts'"
     | }
defined class MyStringContext

scala> println(q"abc ${1+1} def")
args: 'WrappedArray(2)', parts: 'StringContext(WrappedArray(abc ,  def)).parts'
```

A Stream is like a list except that its elements are computed lazily.

http://www.scala-lang.org/api/current/scala/collection/immutable/Stream.html

Streams

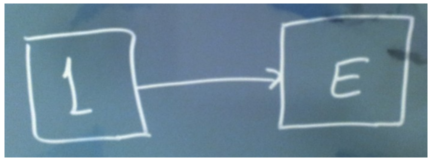
Basics



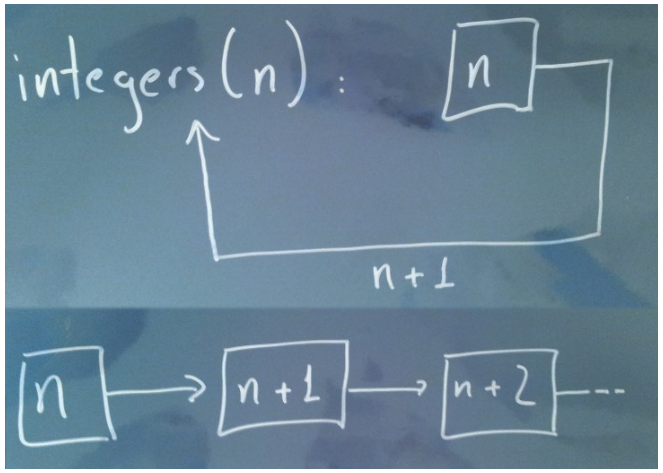
```
scala> Stream.cons(1, Stream.empty)
res0: Stream.Cons[Int] = Stream(1, ?)

scala> Stream.cons(1, Stream.empty): Stream[Int]
res1: Stream[Int] = Stream(1, ?)

scala> Stream.cons(1, Stream.empty).toList
res2: List[Int] = List(1)

scala> Stream.cons(1, Stream.cons(2, Stream.empty)).toList
res3: List[Int] = List(1, 2)
```

Streams

Basics



```
scala> Stream.cons(1, Stream.empty)
res0: Stream.Cons[Int] = Stream(1, ?)

scala> Stream.cons(1, Stream.empty): Stream[Int]
res1: Stream[Int] = Stream(1, ?)

scala> Stream.cons(1, Stream.empty).toList
res2: List[Int] = List(1)

scala> Stream.cons(1, Stream.cons(2, Stream.empty)).toList
res3: List[Int] = List(1, 2)

-----------------------------------------------------------------------------------------

scala> def integers(n: Int): Stream[Int] = Stream.cons(n, integers(n+1))
integers: (from: Int)Stream[Int]

scala> integers(0)
res4: Stream[Int] = Stream(0, ?)

scala> integers(0).take(10)
res5: Stream[Int] = Stream(0, ?)

scala> integers(0).take(10).toList
res6: List[Int] = List(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```



Streams

Usage

```
scala> def evenNumbers: Stream[Int] = integers(0).filter(_ % 2 == 0)
evenNumbers: ()Stream[Int]

scala> evenNumbers.take(10).toList
res7: List[Int] = List(0, 2, 4, 6, 8, 10, 12, 14, 16, 18)
```

Streams

Usage

```
scala> def evenNumbers: Stream[Int] = integers(0).filter(_ % 2 == 0)
evenNumbers: ()Stream[Int]

scala> evenNumbers.take(10).toList
res7: List[Int] = List(0, 2, 4, 6, 8, 10, 12, 14, 16, 18)

---------------------------------------------------------------------------

scala> Stream.continually(math.random).take(3).toList
res8: List[Double] = List(0.5932830094101399, 0.4377639789538802, 0.4522513999390704)
```

Streams

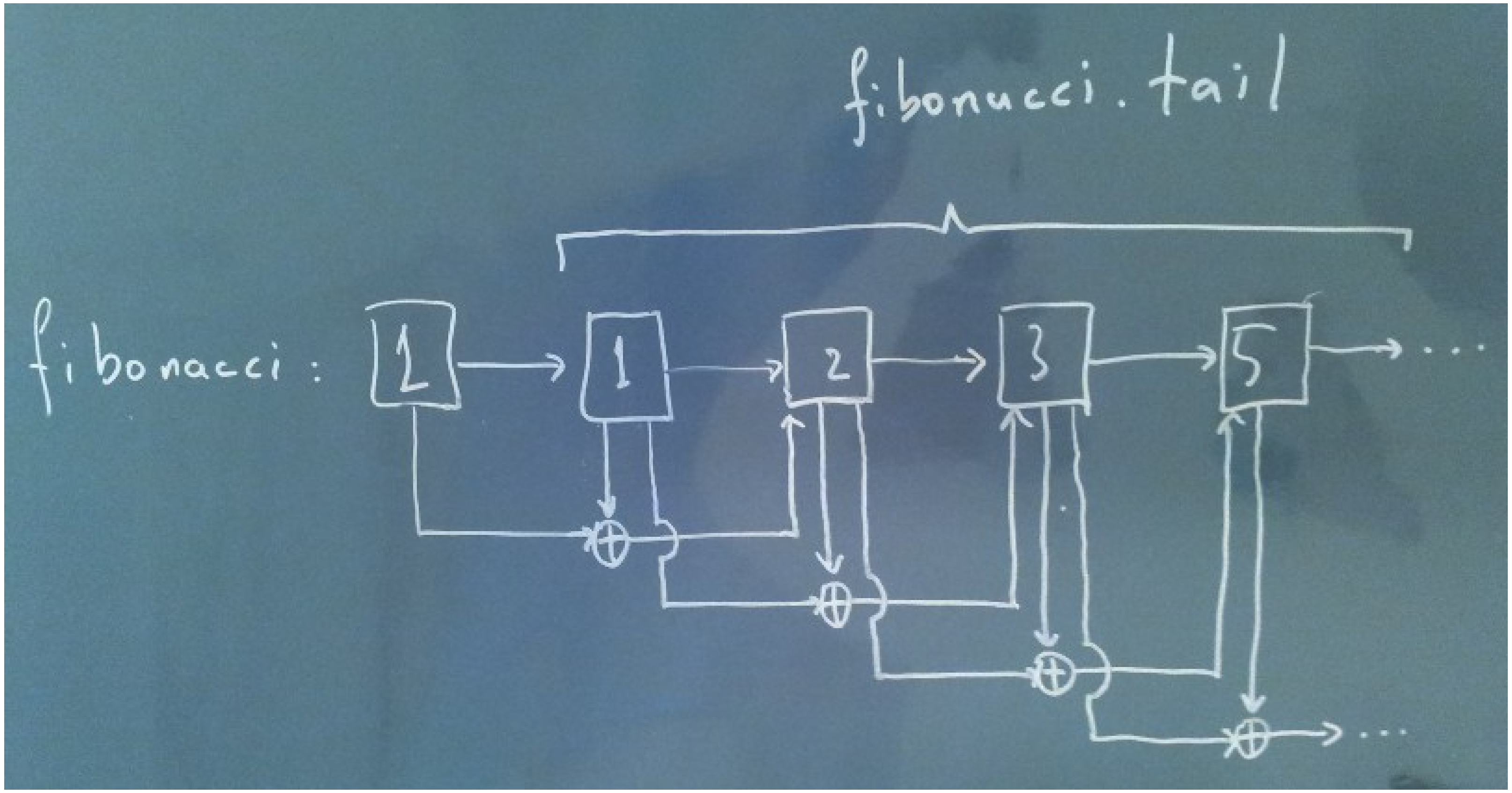
Usage

```
scala> def evenNumbers: Stream[Int] = integers(0).filter(_ % 2 == 0)
evenNumbers: ()Stream[Int]

scala> evenNumbers.take(10).toList
res7: List[Int] = List(0, 2, 4, 6, 8, 10, 12, 14, 16, 18)

-----------------------------------------------------------------------------------

scala> Stream.continually(math.random).take(3).toList
res8: List[Double] = List(0.5932830094101399, 0.4377639789538802, 0.4522513999390704)

-----------------------------------------------------------------------------------

scala> Stream.continually(List(1,2,3)).flatten.take(8).toList
res9: List[Int] = List(1, 2, 3, 1, 2, 3, 1, 2)
```

Streams

Recursion [1/2]

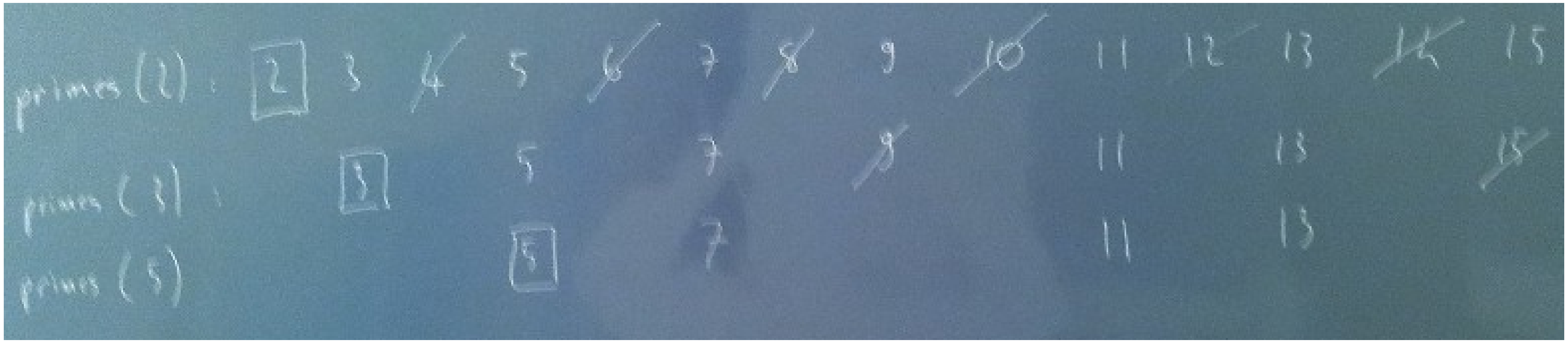
```
scala> def fibonacci: Stream[Int] =
           1 #:: 1 #:: fibonacci.zip(fibonacci.tail).map { case (x, y) => x + y }
fibonacci: Stream[Int]

scala> fibonacci.take(10).toList
res8: List[Int] = List(1, 1, 2, 3, 5, 8, 13, 21, 34, 55)
```



Streams

Recursion [2/2]

```
scala> def primes(implicit n: Int = 2): Stream[Int] =
         n #:: primes(n+1).filterNot(_ % n == 0)
primes: (implicit n: Int)Stream[Int]

scala> primes.take(10).toList
res17: List[Int] = List(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
```


primes (2) : 2 3 4 5 6 7 8 9 10 11 12 13 14 15
primes (3) : 3 5 7 9 11 13 15
primes (5) 5 7 11 13

Streams

Gotchas

```
scala> def naturalNumbers: Stream[Int] = integers(0)
naturalNumbers: ()Stream[Int]

scala> val naturalNumbers: Stream[Int] = integers(0)
naturalNumbers: Stream[Int] = Stream(0, ?)

scala> lazy val naturalNumbers: Stream[Int] = integers(0)
naturalNumbers: Stream[Int] = <lazy>
```

Regular Expressions

http://www.scala-lang.org/api/current/index.html#scala.util.matching.Regex

Regular Expressions

Basics

```
scala> "0xDeadBeef" matches "0x[0-9a-fA-F]+"
res1: Boolean = true

scala> "abc 0x12 def" matches "0x[0-9a-fA-F]+"
res2: Boolean = false
```

Regular Expressions

Basics

```
scala> "0xDeadBeef" matches "0x[0-9a-fA-F]+"
res1: Boolean = true

scala> "abc 0x12 def" matches "0x[0-9a-fA-F]+"
res2: Boolean = false

----------------------------------------------------------------------------

scala> val dateRegex = "([0-9]{4})-([0-9]{2})-([0-9]{2})".r
hexregex: scala.util.matching.Regex = ([0-9]{4})-([0-9]{2})-([0-9]{2})

scala> dateRegex findFirstIn "2013-04-08" match {
     |   case Some(dateRegex(y, m, n)) => s"y: $y, m: $m, n: $n"
     |   case None => "invalid date"
     | }
res3: String = y: 2013, m: 04, n: 08

scala> dateRegex findAllIn "2013-04-08 2012-05-07" map {
     |   case dateRegex(y, m, n) => (y, m, n)
     | } toList
res4: List[(String, String, String)] = List((2013,04,08), (2012,05,07))
```

Process Execution

http://www.scala-lang.org/api/current/index.html#scala.sys.process.package

Process Execution

Basics

```
scala> import scala.sys.process._
import scala.sys.process._
```

Process Execution

Basics

```
scala> import scala.sys.process._
import scala.sys.process._

------------------------------------------------------------------------------------------

scala> "/bin/false".!
res1: Int = 1

scala> "/bin/true".!
res2: Int = 0
```

Process Execution

Basics

```
scala> import scala.sys.process._
import scala.sys.process._

-------------------------------------------------------------------------------------

scala> "/bin/false".!
res1: Int = 1

scala> "/bin/true".!
res2: Int = 0

-------------------------------------------------------------------------------------

scala> "uptime".!!
res3: String =
" 17:30:38 up 4 days,  3:41,  5 users,  load average: 1.18, 0.97, 0.79
"
```

Process Execution

Basics

```
scala> import scala.sys.process._
import scala.sys.process._

--------------------------------------------------------------------------------

scala> "/bin/false".!
res1: Int = 1

scala> "/bin/true".!
res2: Int = 0

--------------------------------------------------------------------------------

scala> "uptime".!!
res3: String =
" 17:30:38 up 4 days,  3:41,  5 users,  load average: 1.18, 0.97, 0.79
"
--------------------------------------------------------------------------------

scala> Seq("ls").lines
res4: Stream[String] = Stream(Desktop, ?)

scala> Seq("ls").lines.toList
res5: List[String] = List(Desktop, Documents, Downloads, Music, …)

scala> Seq("ping", "-w", "1", "-c", "1", "google.com").lines
res6: Stream[String] = Stream(PING google.com (173.194.70.103) 56(84) bytes of data., ?)
```

Process Execution

Magic

```
scala> def compile(file: String) =
     |   s"ls $file" #&& s"scalac $file" #|| "echo nothing found" lines
compile: (file: String)Stream[String]
```

Process Execution

Magic

```
scala> def compile(file: String) =
     |   s"ls $file" #&& s"scalac $file" #|| "echo nothing found" lines
compile: (file: String)Stream[String]

-----------------------------------------------------------------------------------------

scala> import java.io.File
import java.io.File

scala> import java.net.URL
import java.net.URL

scala> new URL("http://www.scala-lang.org/") #> new File("/tmp/scala-lang.html") !
res7: Int = 0
```

Process Execution

Gotchas

```
scala> Seq("choke").lines
java.io.IOException: Cannot run program "choke": error=2, No such file or directory
```

Questions?