

Java
プログラマに
捧げる
Scala入門

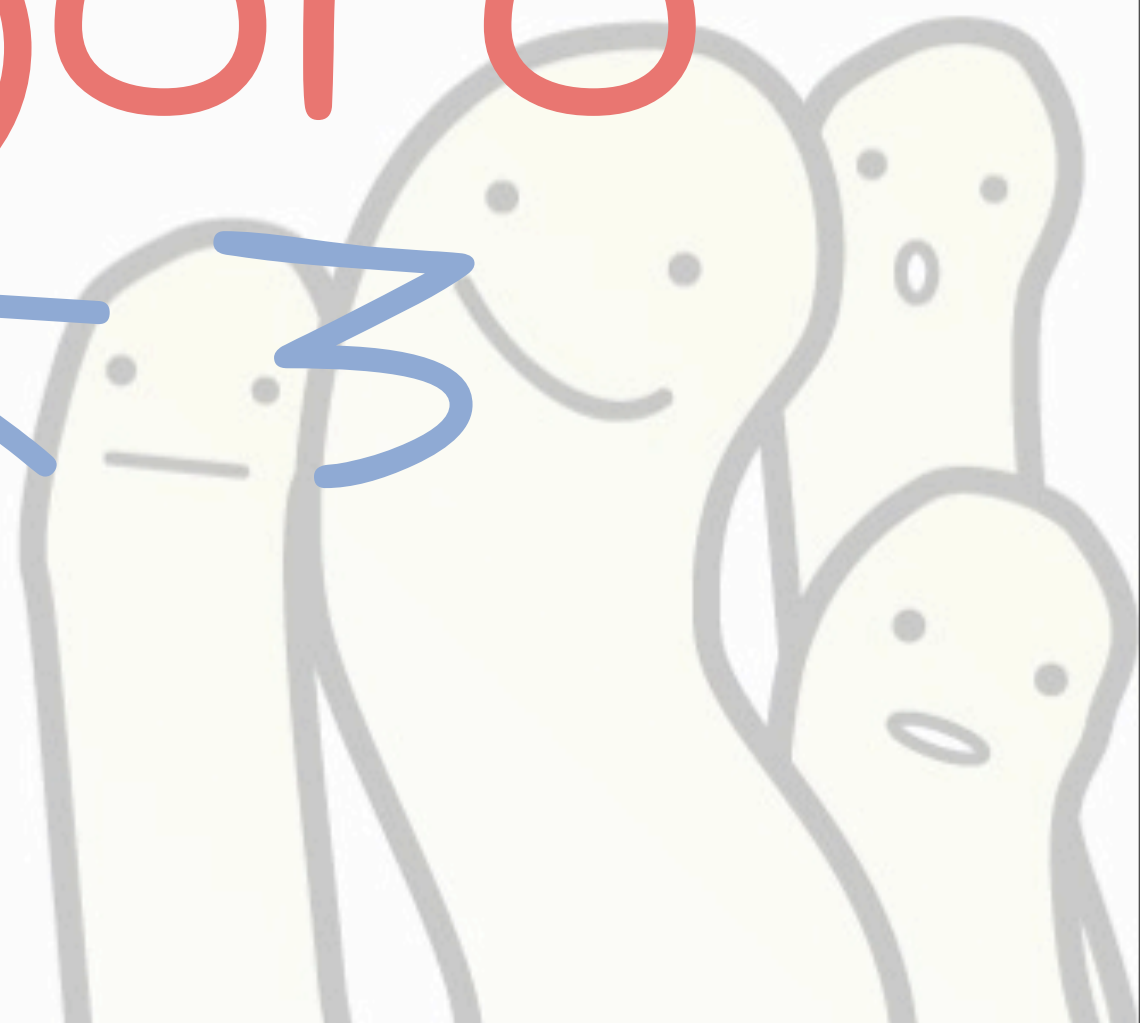
Seasar Conference 2009 Autumn

自己紹介

ID:

yuroyuro

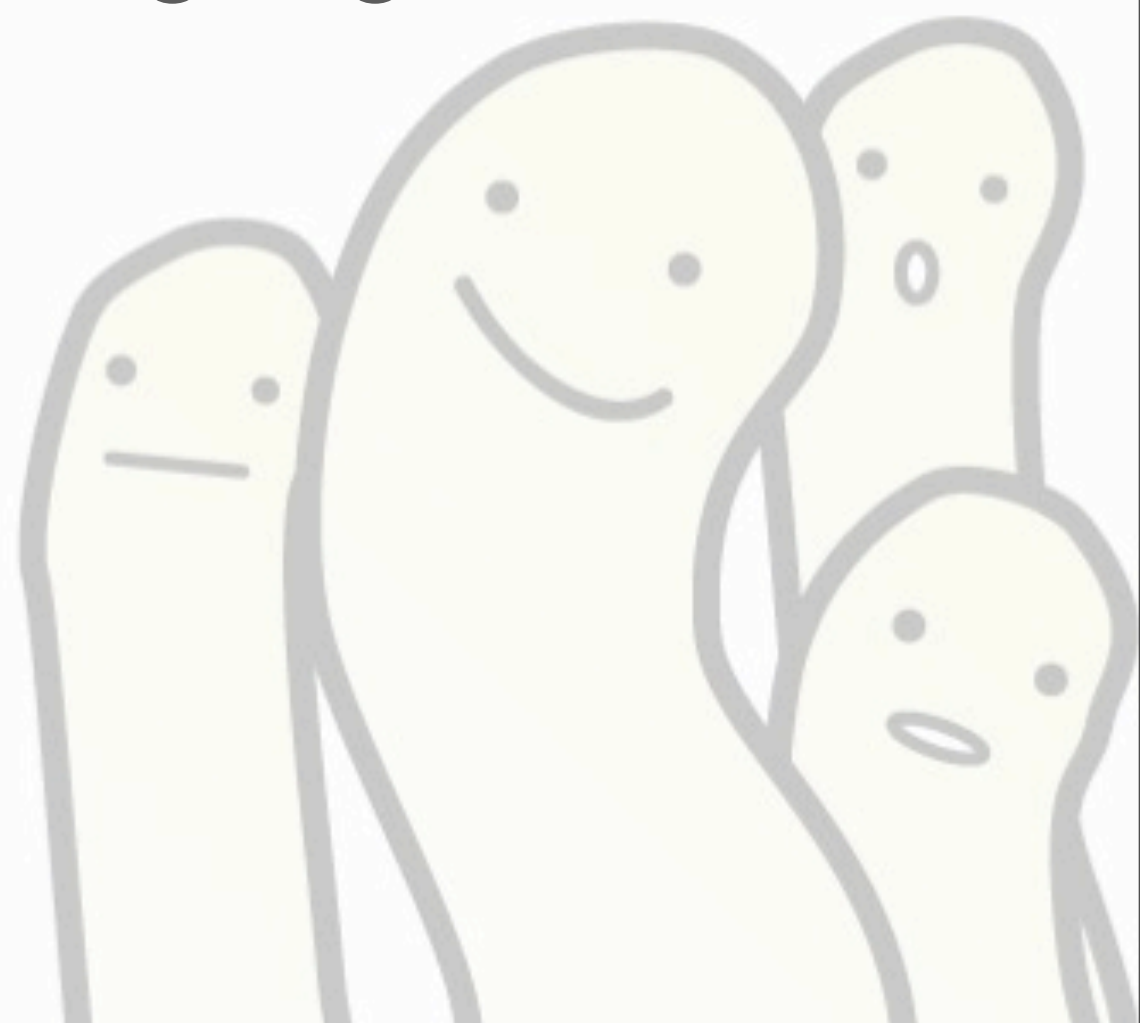
ゆるよる



自己紹介

- 尾崎 智仁(ゆろよろ)
- ID: yuroyoro
(はてな、twitter、wassr)
- フリーエンジニア
- JavaとかScalaとか
pythonとか

それでは・・・

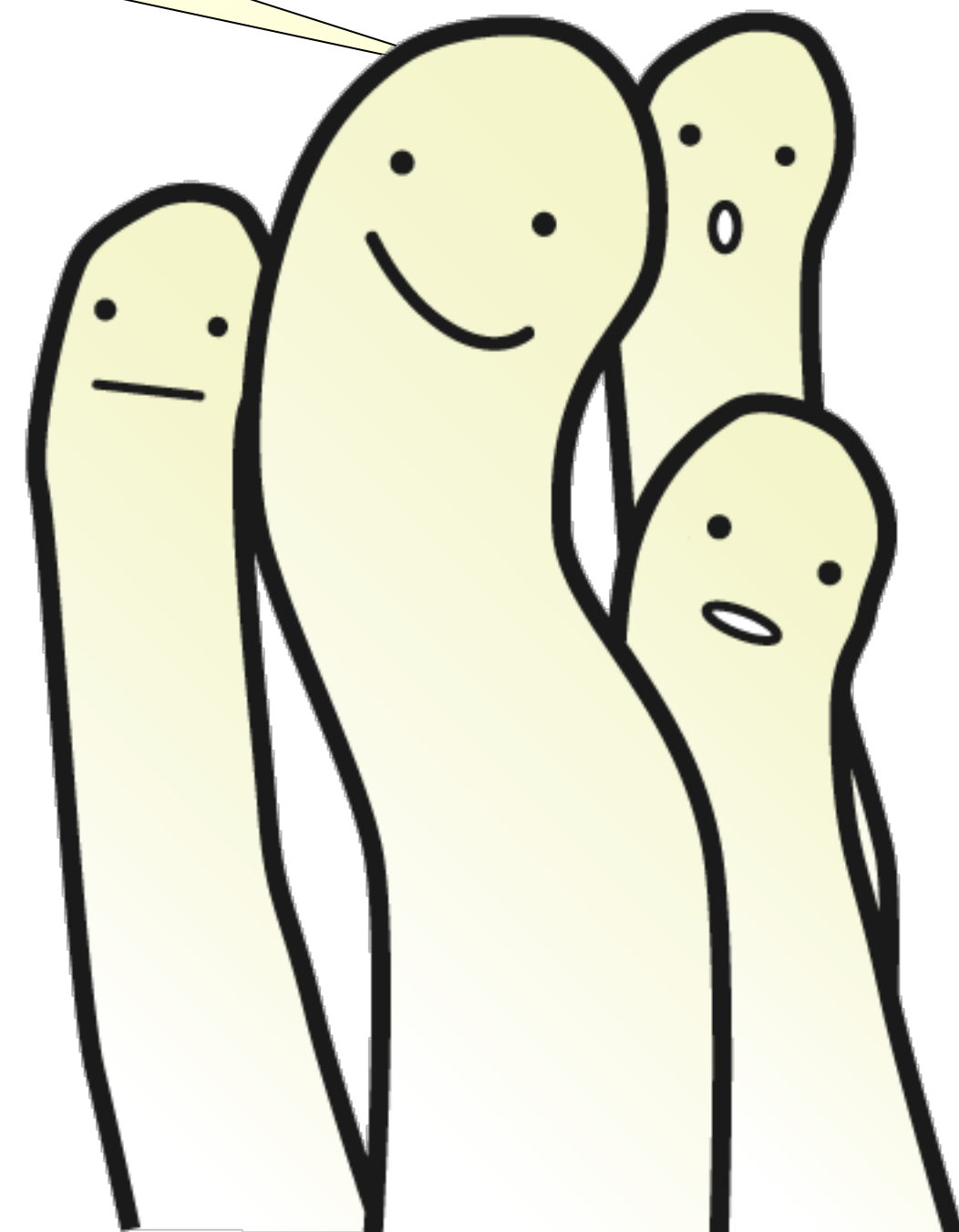
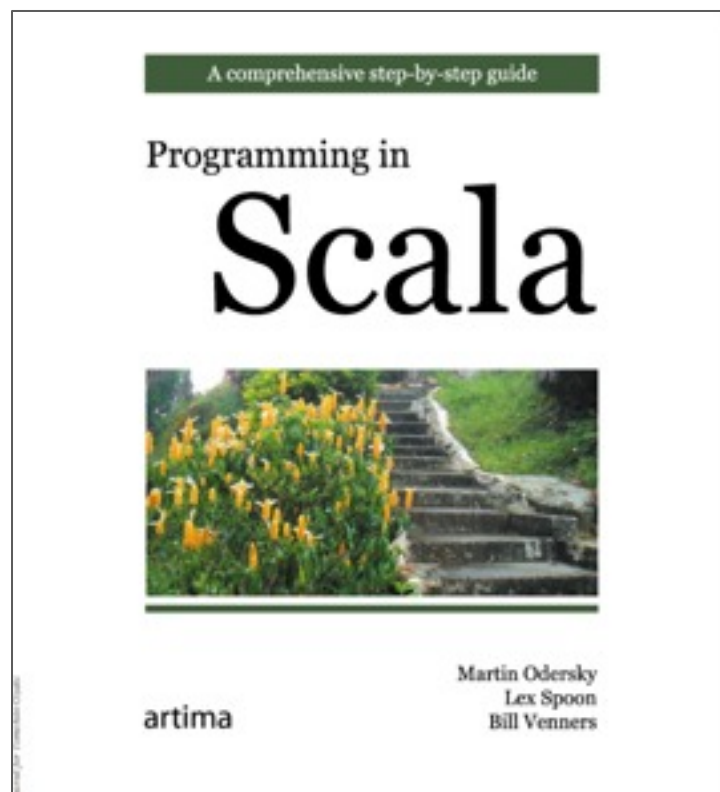


質問

たーいむ!!

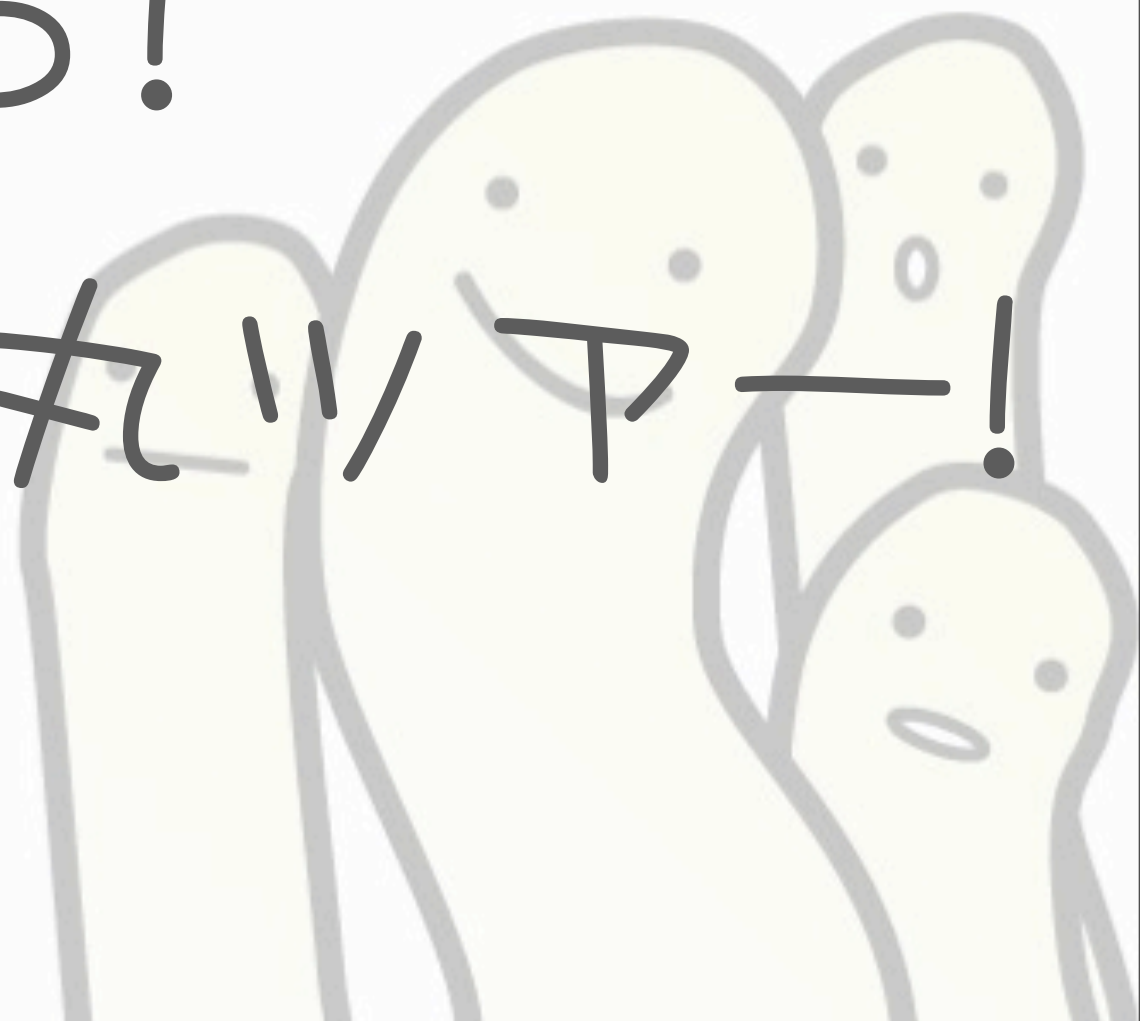


Q.Scala本
読みました?



今日の内容

- scala の 紹介!
- デモ2連発っ!
- 言語仕様弾丸リッアー!
- まとめっ!

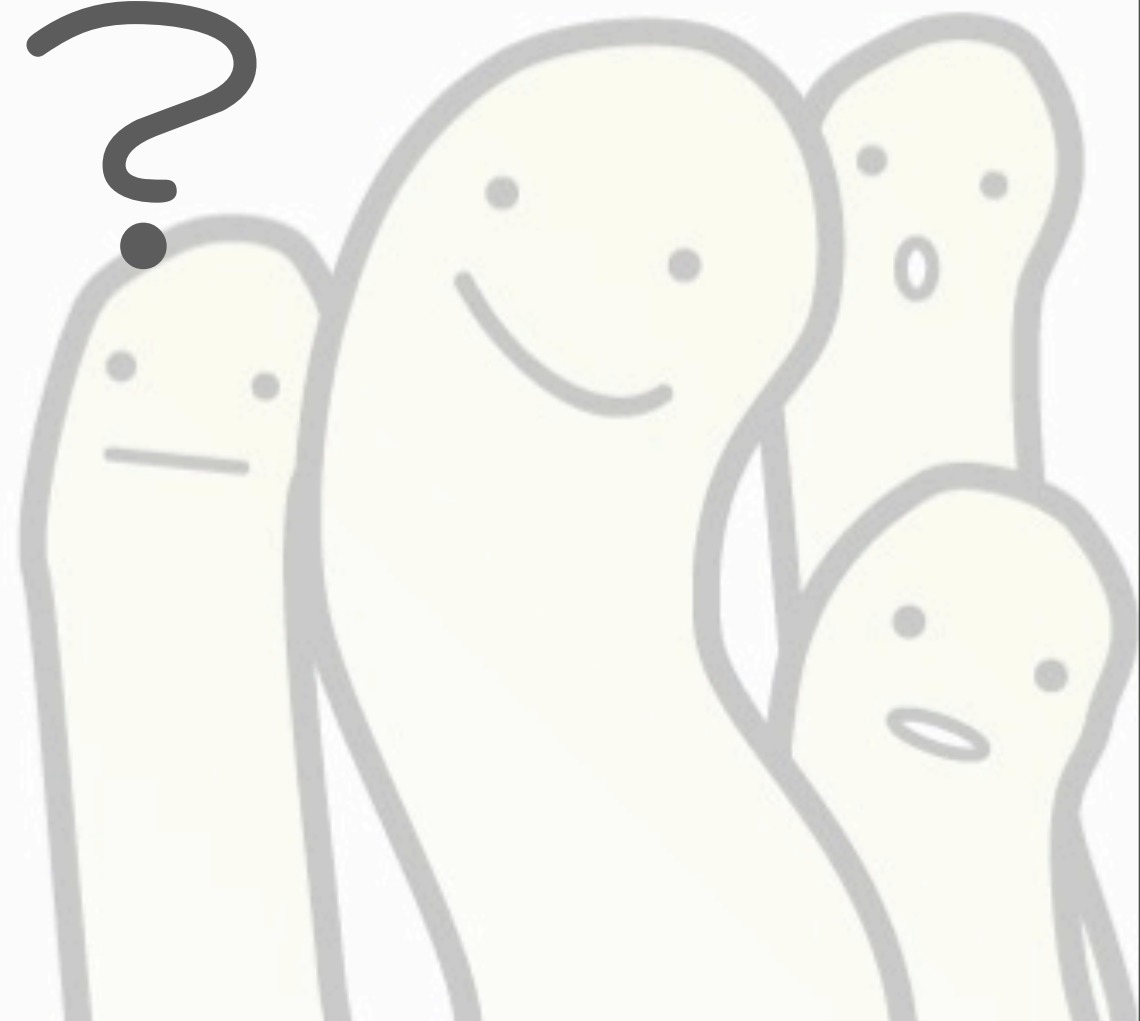


A white ceramic cup filled with dark coffee sits on a matching saucer. A silver spoon rests on the saucer. The cup and saucer are placed on top of an open book. The book's pages are visible, showing text in multiple languages including Dutch, Swedish, and Norwegian. The background is a soft, out-of-focus light blue.

scala の 紹介!

dra...
1965...
art and c...
Foreword b...
60 Pekka Toivanen...
Sweden 118 Lars Dub...
Norge Noorwegen Norway 205 Rudi Fuchs Zolang de kraanvogels het zwerk maar aan
Voorwoord Foreword Rudi Fuchs Zolang de kraanvogels het zwerk maar aan
de juiste kant en in de goede richting blijven doorkruisen is 'Het Transparante
Noorden' de laatste tentoonstelling in de Nieuwe Vleugel van het Stedelijk
Museum. Wanneer het stof is opgetrokken na de ingrijpende uitbreiding in de
tuin naar het ontwerp van de Portugese architect Alvaro Siza, kunnen we de
nieuwe ruimten verkennen en gaandeweg veroveren. Dan zal ook het tentoon-
stellingsprogramma veranderen. We blijven het huis van de actualiteit, maar we

Q.scalaって
なに？



みんな大好きついったーも
scalaで動いています



scalaの特徴

- `jvm`で動きます!
- オブジェクト指向 + 関数型
- ファーストクラス関数
- 静的型付け + 型推論
- `Java`との相互運用

scalaの気持ちいいところ

- インタプリタで動作確認ができる!
- 型推論で簡潔なコード
- でも静的型付け言語



scalaの気持ちいいところ

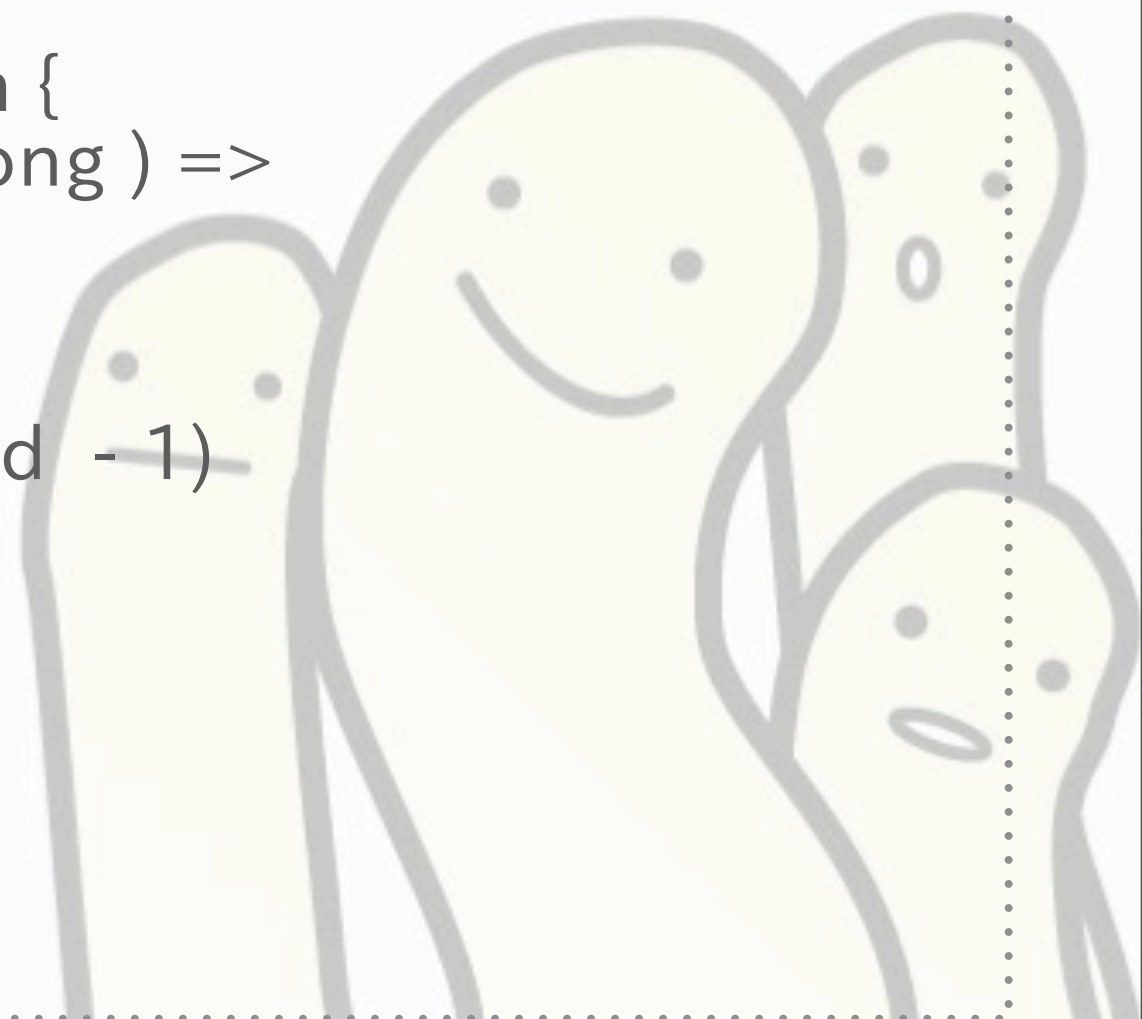
- クロージャ + 充実したコレクションライブラリ
- Object, traitで進化したオブジェクト指向

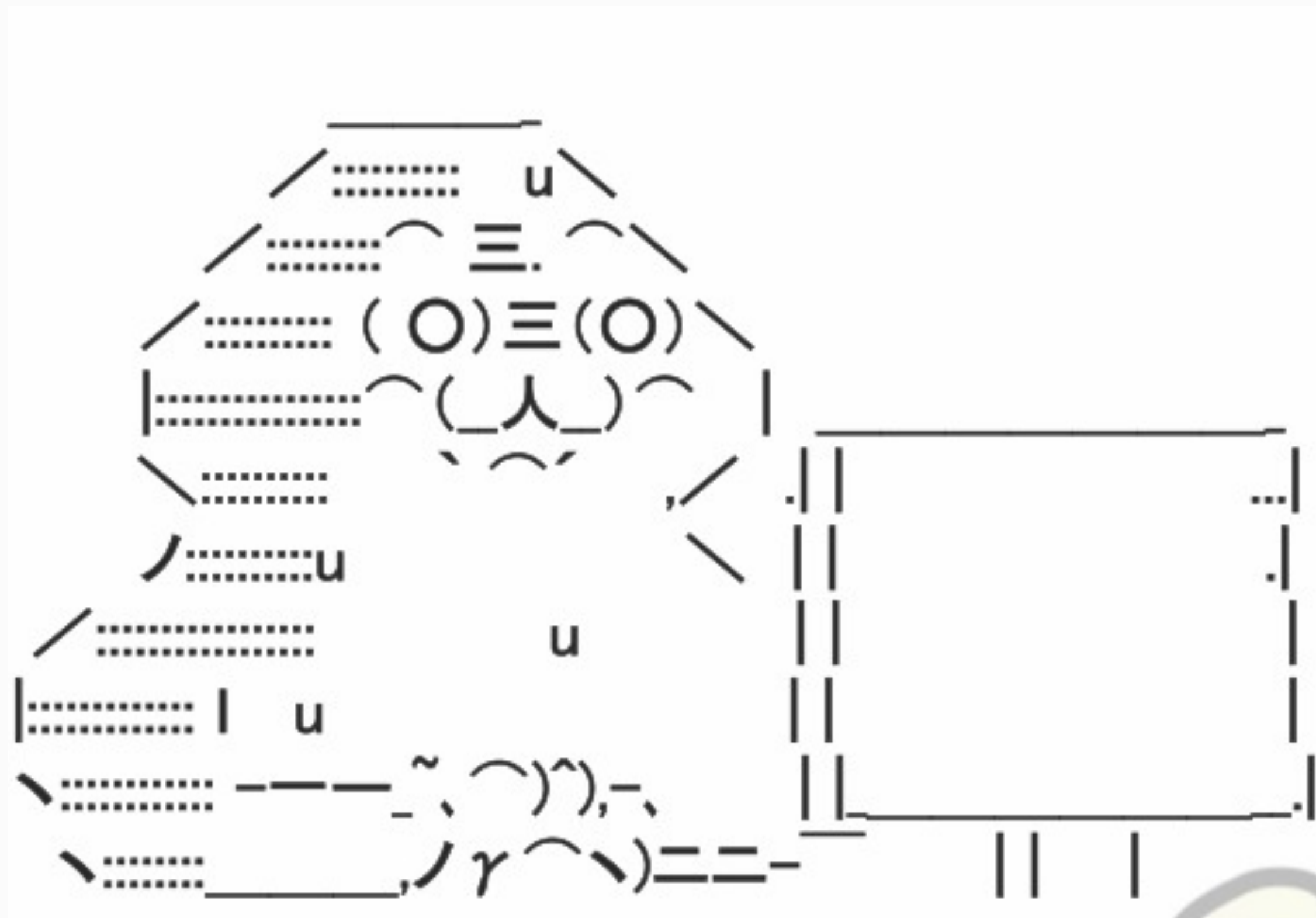
scalaのソース(例)

```
case class UserTimeline( maxId:Long )

object UserTimelineActor extends Actor{
  import Main._

  def act = {
    var cnt:Long = 0
    loop{
      react{
        case UserTimeline( maxId ) =>
          getUserTimeline( maxId ) match {
            case (xml:Node , nextMaxId:Long ) =>
              outputter.add( xml )
              cnt += 200
              Thread.sleep(interval)
              this ! UserTimeline( nextMaxId - 1)
            case _ =>
              outputter.close
              exit
          }
      }
    }
  }
}
```

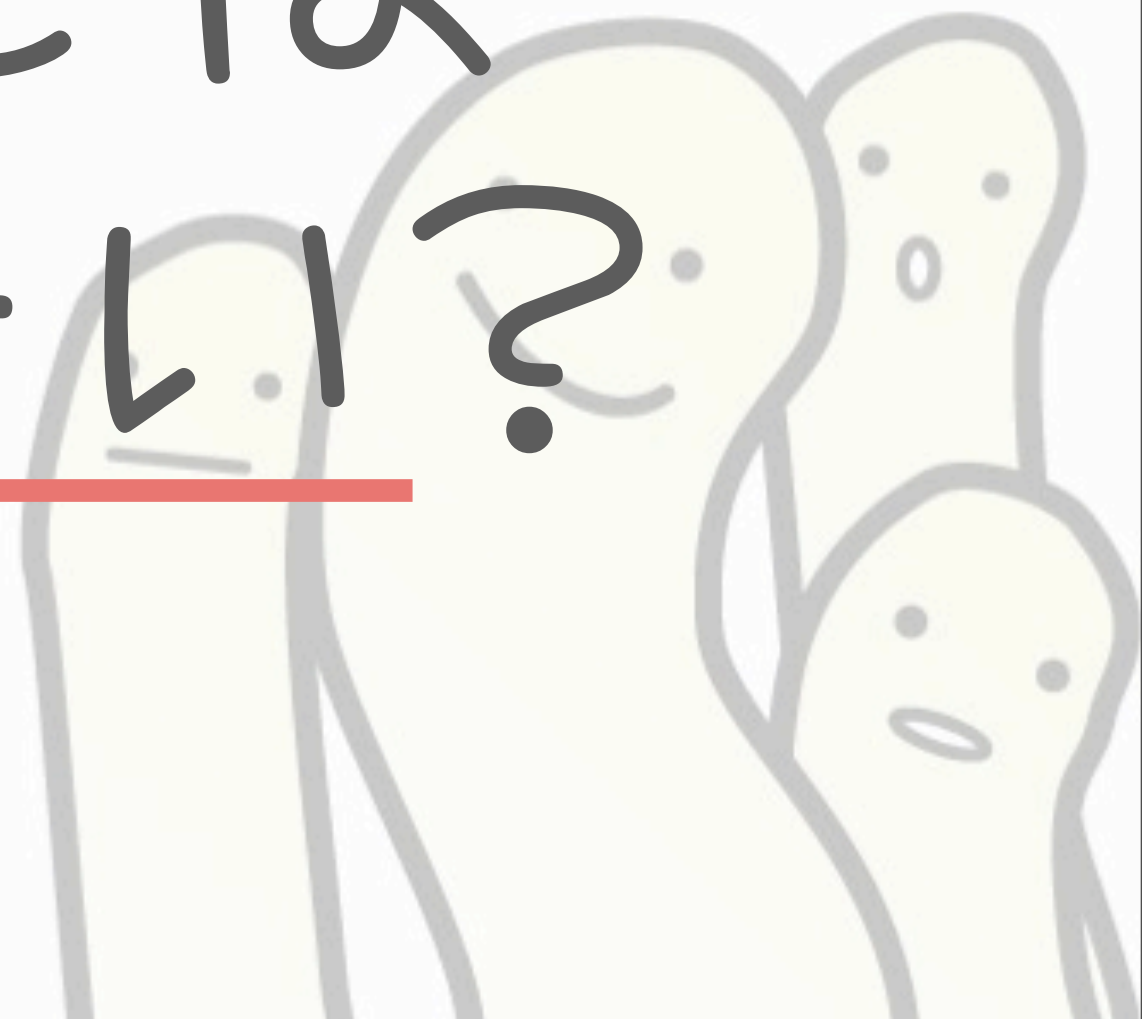




ちよwww
イww≡ww>www
なにwwwニれww



Scalaは
業務では
使えない？



No!!!



今から

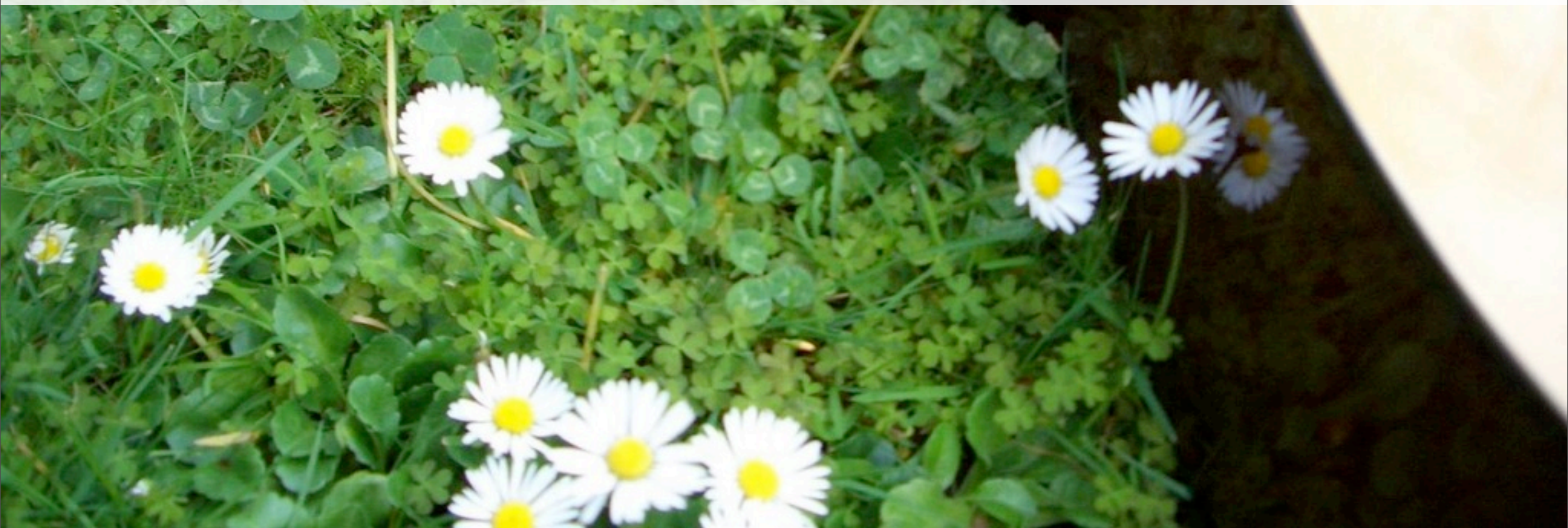
実証

しゃんよ!!





デモっ！



demo その1

- scalaで簡単なスクリプトを書いてみる!
- twitter検索でs2conのハッシュタグを検索
- 目標5分!



demo その2

- S2Console!
- Seasarプロダクトをイン
タプリタで動作確認
- LLフレームワークの対話
環境をJavaにも!



Scala 言語仕様 弾丸ッアー!

インターリタで
動作を
確認してみよう!



基本編



変数(var, val)

- var/val で変数を宣言
- var 変数名:型 = 値
- var は変更可能、val は変更不可(final)



変数(var, val)

```
scala> val i: Int = 1
```

```
i: Int = 1
```

```
scala> i = 2
```

```
<console>:13: error: reassignment to val
```

```
    i = 2
```

```
    ^
```

```
scala> var j = 10
```

```
j: Int = 10
```

```
scala> j = 20
```

```
j: Int = 20
```

valは再代入できない変数(javaのfinal)

代入しようとするときエラー

varは代入可能

型推論

- 変数、関数の戻り値は省略可能
- コンパイラーが推論する
- 型パラメータも含めて推論
- 関数の引数は省略できない

型推論

```
scala> val list:List[Int] = List(1,2,3,4)
```

```
list: List[Int] = List(1, 2, 3, 4)
```

省略できる

```
scala> val list = List(1,2,3,4)
```

```
list: List[Int] = List(1, 2, 3, 4)
```

省略!

```
scala> val a = List(1,2,3).map(_._toString)(0)
```

```
a: java.lang.String = 1
```

関数の戻り値
から推論!

if 文

- `if/else` 文は値を返す
- `if` の条件は `Boolean`
- `else` が無い場合は値を返さない(`Unit`型が結果となる)
- `if` 文の結果を変数に代入できる(三項演算子的な)

if 文

```
scala> val s = "abcdef"
s: java.lang.String = abcdef
```

条件は(Boolean)

```
scala> if( s.startsWith("abc") ) s.length
      else 0
```

```
res4: Int = 6
```

Int型が返る

結果を代入!

```
scala> val r = if(s.length > 1) s.substring(1)
      else ""
```

```
r: java.lang.String = bcdef
```

for式によるループ

- for式でコレクションの反復処理(javaの拡張forに相当)
- for(変数 ← コレクション) { ... }
- コレクション生成 - for(変数 ← コレクション) yield { ... }
- フィルター - for(n ← 1 to 10 ; if n % 2 == 0) yield { ... }
- break/continueはない!! (2.8で実装)

for式によるループ

```
scala> for( n <- 1 to 10 if n %2 == 0 ){  
    print(n * 2 + ",") }
```

4,8,12,16,20,

1から10のなか
で偶数のみ

結果を代入!

```
scala> val g = for( s <- List("a","b","c");  
    l = s.length )yield { s + ":" +  
    l }
```

```
g: List[java.lang.String] = List(a:1, b:1, c:1)
```

A close-up photograph of pink cherry blossoms in full bloom, with some flowers in sharp focus and others blurred in the background. The blossoms are a light pink color with darker pink centers.

関数編

関数

defで
関数定義

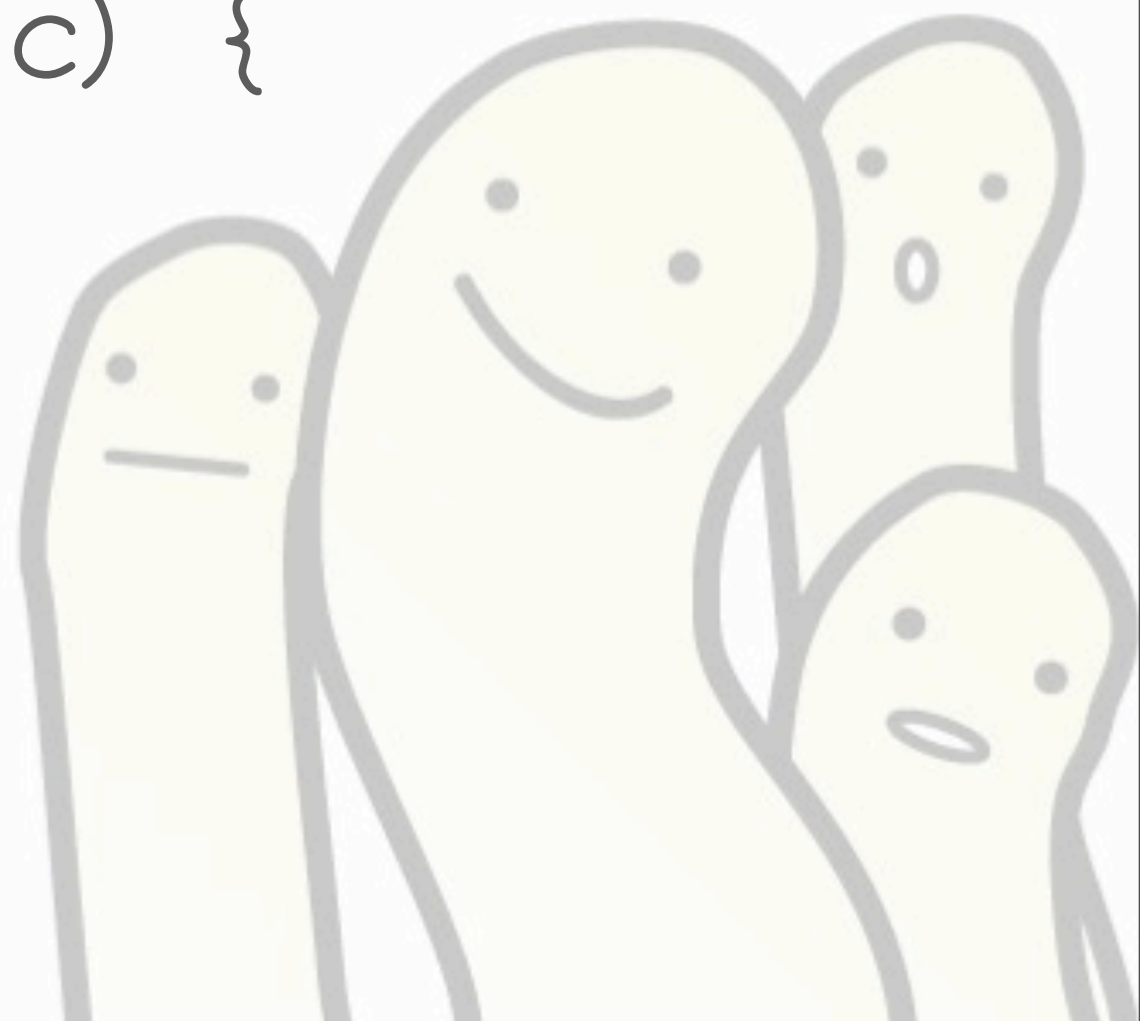
関数名

```
def countChar ( s:String, c:Char ) : Int =  
{  
  var cnt:Int = 0  
  for( ct <- st if ct == c ) {  
    cnt += 1  
  }  
  cnt  
}
```

引数
仮引数:型

戻り値の型

最後に評価された
値が戻り値



関数

- `def 関数名( 引数 ): 結果型 = { ... }`
- `val f = 関数名` で関数オブジェクト
- 関数リテラル - 無名関数的な
`val f = (x:Int, y:Int) => { x + y }`
 - 引数
仮引数: 型
 - 本体
- 関数リテラルでは `_` で引数のプレースホルダ

関数

```
scala> def isEven(i:Int):Boolean = {  
    return ( i % 2 == 0 )  
}
```

Javaっぽく書く
とこんな感じ

```
isEven: (Int)Boolean
```

```
scala> def isEven(i:Int) = i % 2 == 0
```

省略!

```
isEven: (Int)Boolean
```

```
scala> val func = isEven _  
func: (Int) => Boolean = <function>
```

関数を
オブジェクト
として変数に
代入!

```
scala> val f = (i:Int) => i % 2 == 0  
f: (Int) => Boolean = <function>
```

関数リテラル!

カリー化と部分適用

- `def hoge(i:Int)(j:Int) = i + j`
- 引数を`()`で分けるとカリー化
- `hoge(1)(_:Int)`
- 引数に`_`を指定して関数を部分適用



カーリー化と部分適用

```
scala> def add(n:Int)(m:Int)=n+m  
add: (n: Int)(m: Int)Int
```

カーリー化っ!!

```
scala> val add3 = add( 3 ) _  
add3: (Int) => Int = <function>
```

“_”で
部分適用!

```
scala> def curryingAdd( n:Int ) = add( n ) _  
curryingAdd: (Int)(Int) => Int
```

カーリー化され
た”関数”を
返す関数

ネスト関数とクロージャ

- 関数は入れ子(ネスト)にできる!
- 関数の外側の変数(自由変数)を参照できる
- 自由変数への参照を実行時に取り込む(クロージャ)



ネスト関数とクローージャ

```
def outer ( i:Int, s:String ) = {
```

```
  val v:String = s + ":" + i
```

```
  def inner( p:String ) = p + v
```

入れ子の関数
から外側の変数
を参照

```
  inner _
```

自由変数vを取り込んだ
inner関数オブジェクト
を返す

```
}
```

```
scala> val inf = outer( 5, "hoge" )  
inf: (String) => java.lang.String = <function>  
scala> inf("foo - ")  
res2: java.lang.String = foo - hoge:5
```

コレクションに関数を

- scalaのコレクションは31数
に関数を取るものが多い
- map, filter, foreach, exist ...
- 柔軟なコレクションの操作
- List, Map, Setなど



コレクションに関数を

```
scala> val list = List(1,2,3,4)
res1: List[Int] = List(1, 2, 3, 4)
```

```
scala> list.map( {(n:Int) => n * 2} )
res2: List[Int] = List(2, 4, 6, 8)
```

map操作
関数リテラルで
各要素を2倍

```
scala> list.filter( _ % 2 == 0 )
res3: List[Int] = List(2, 4)
```

偶数の要素を
抽出

```
scala> list.filter( func )
res4: List[Int] = List(2, 4)
```

関数オブジェクト
を引数に渡す!



クラス編

クラスとオブジェクト

- `class` <クラス名>(コンストラクタ引数)
- `object`でシングルトン定義
- ネスト `class`の中に`class`や`object`
- `static`がない`scala`では、コンパニオンオブジェクトを利用する
- 同ースコープに同じ名前の`class`と`object`があると特権的なアクセス権



クラスとオブジェクト

classで
class定義

クラス名

コンストラクタ引数

```
class Foo ( s:String, i:Int ) {
```

```
  val p1:String = s
```

```
  var p2:Int = i
```

フィールド
valは変更不可
varは変更化

```
  private var p3 = 1
```

```
  def hoge = s * p3
```

関数

objectで
Singleton
object

```
}
```

```
object Foo {
```

```
  val aaa = "aaa"
```

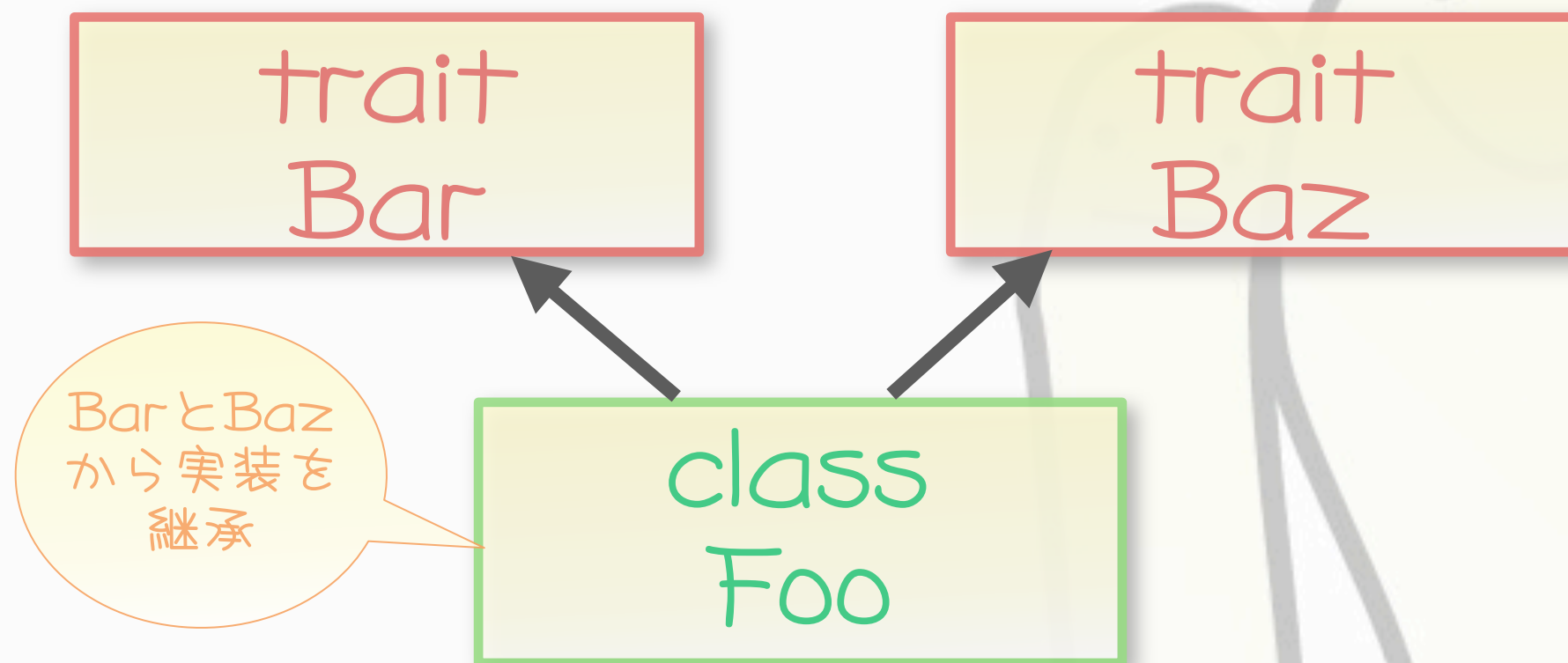
```
  def setP3( foo:Foo, i:Int ) = foo.p3 = i
```

```
}
```

同一クラス名の
クラスの
privateフィールドに
アクセスできる

trait と mixin

- trait は 実装をもてる interface
- class や object に複数の trait を Mix-In
- trait の中で実装を持たないメソッドは abstract



traitとMixin

traitで
trait定義

trait名

```
trait Bar {  
  val bs:String = "Bar"  
  def bar(n:Int) = bs * n  
}
```

traitはフィールド
関数をもてる

```
trait Baz { def baz(n:Int) = "Baz" * n }
```

```
class Foo extends Bar with Baz
```

withで複数の
traitをmixin

case クラスとパターンマッチ

- `case class` <クラス名>(コンストラクタ引数)
- `case class` は名前(...)でインスタンス生成(immutable object)
- `case` クラスはパターンとして使える
- `match` は `switch` の強力なやつ
- 値 `match { case パターン => 処理 ... }`
- パターン には、値/型/`case` クラスを指定

case クラスとパターンマッチ

抽象クラス

```
abstract class Pt
```

case class 定義

```
case class CC(s:String) extends Pt
```

```
case class DD(i:Int, s:String) extends Pt
```

```
def test(p:Pt) = p match {
```

```
  case CC(s) => println(s)
```

```
  case DD(3, _) => println("match DD")
```

```
}
```

matchした
case classの
値を利用

case classを
パターンとして
match

ちよっとだけよ・・・

- implicit conversion
 - ー 暗黙の型変換(StringからDoubleに自動変換etc)
- implicit parameter
 - ー 型による自動判別を行うデフォルト引数
- structural typing
 - ー type safeなダックタイピング
- lazy value
 - ー 遅延評価



ちょっとだけよ・・・

- XMLリテラル
 - xmlをリテラルで評価
- 共変/反変なGenerics
 - `Array[String]`は`Array[Any]`のサブクラス
- actorライブラリ
 - message base非同期処理
- Lift web framework
 - 関数型webフレームワーク





ま と め っ !

フリーダム

する
お話し仕様を
楽しめ!!

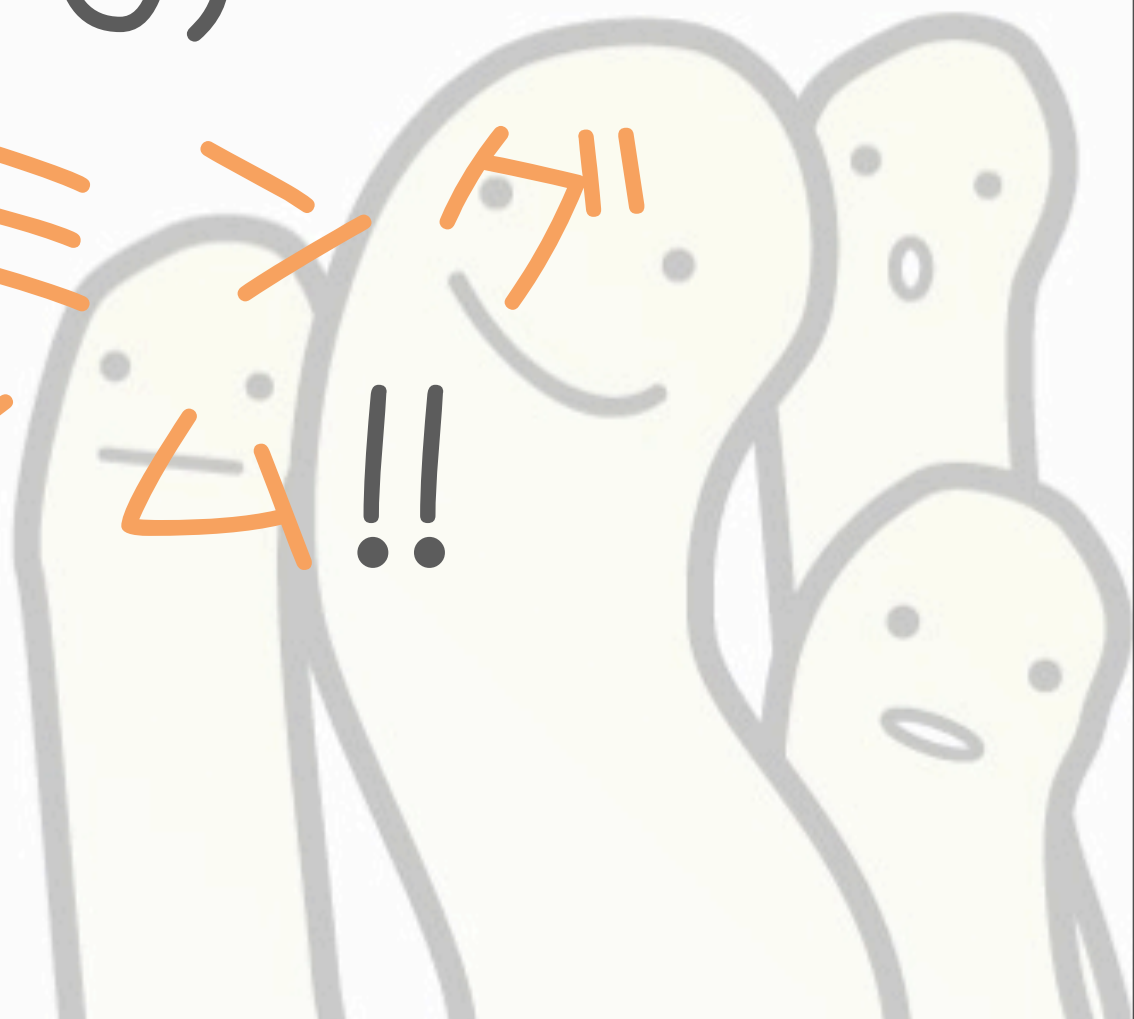
Javaと LLの 融合!!



進化した
オブジェクト指向で
Javaの常識を
破壊せよ!!



関数型言語の
概念を加えた
次世代の
プログラミング
パラダイム!!



情報源

- 公式サイト

<http://www.scala-lang.org/>

- Scala 東北

<https://sites.google.com/site/scalatohoku/>

- Scala-be

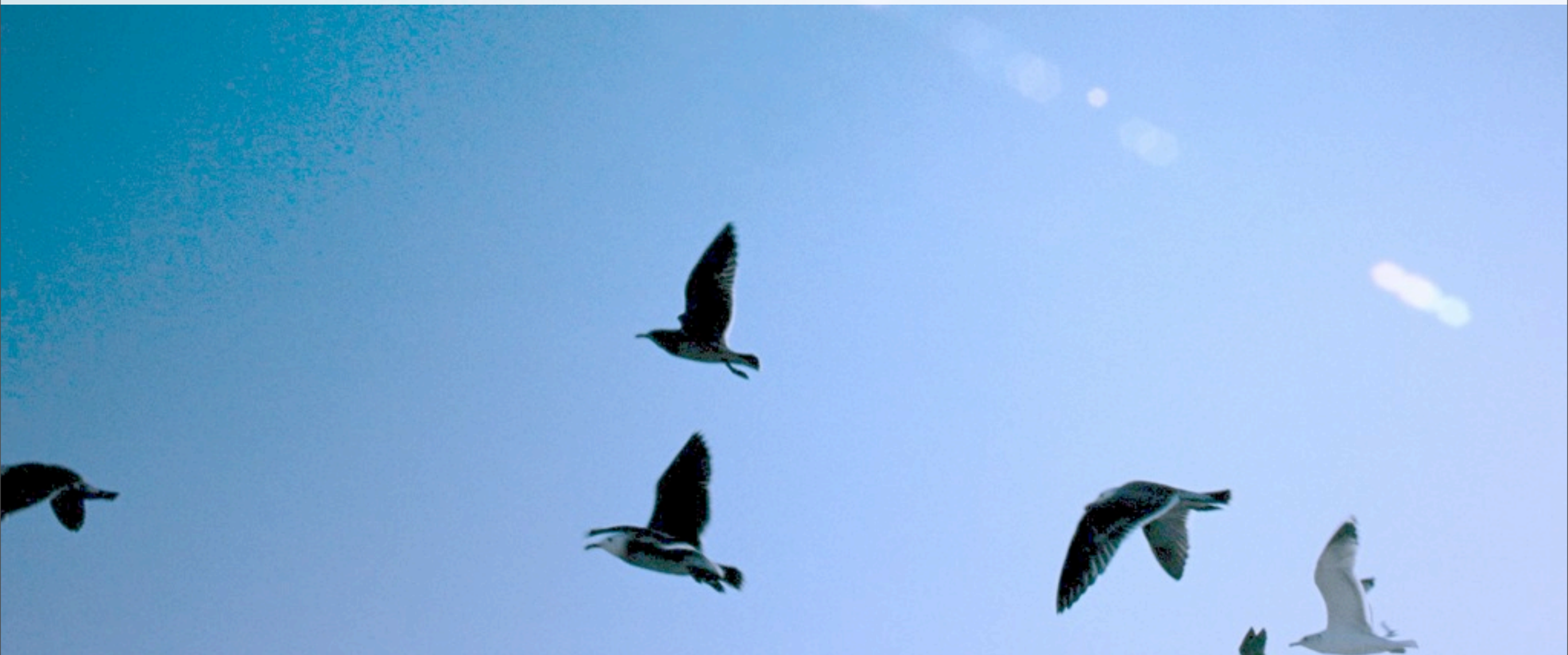
<http://groups.google.co.jp/group/scala-be>

- Programming in Scala

<http://www.amazon.co.jp/Programming-Scala-Comprehensive-Step-step/dp/0981531601>



Have fun
with
Scala !!



ご清聴
ありがとうございました!
ございました!

