Scala言語開発

Featuring
play framework

ダイキチ・ドットネット有限会社
DAIKICHI.net Co., Ltd.

池田成樹
Ikeda Naruki

池田成樹

twitter:@ikedanaruki

・猫好き。同居する3匹の兄弟猫は社員待遇
・酒好き。親族は造り酒屋。一番好きなのはスコッチウイスキー
・音楽好き。UKロック・プログレ、徳永英明が好き。EB・AG弾き
・時代／歴史小説好き。池波正太郎と岡本綺堂が特に好き
・落語好き。立川談志と柳家小三治が特に好き
・「血液型を性格判断に使うのを禁止させる会」代表（自称）
・野球はスワローズ、サッカーはFC東京・Gレンジャーズのファン
・絶賛発売中！iPhoneアプリ「Erahidaeon Audio Player」
・自著最新作「Scalaテキスト」カットシステム刊
・自著は大学,(高等)専門学校等の教科書採用多数
(注)コンピューターに興味ない方はこの先読まないでください
好きな言語：Scala,C++,アセンブラ,Objective C,Java
最近好きになった言語：JavaScript,Haskell,Clojure
ずっと嫌いな言語：C#,Visual Basic
好きなOS:UNIX系。特にAIX。Mac OSXも含む
嫌いなOS:Windows系　微妙なOS:Android,Plan 9
最近の関心：Scala/Play!,HTML5/Ajax,関数型言語

キジ(♂) ヒメ(♀) 小鉄(♂)
常に仕事ぶりを監視してマス

NPO法人JASIPA
理事・研修委員長
http://jasipa.jp/

I live in Karuizawa!

Scala 使ってます！

Lift WEB FRAMEWORK

play

BookStoreS.jp

dwango

twitter

chatwork

foursquare

CyberAgent

LinkedIn

NTT DATA

theguardian

GREE

BIZREACH

nulab

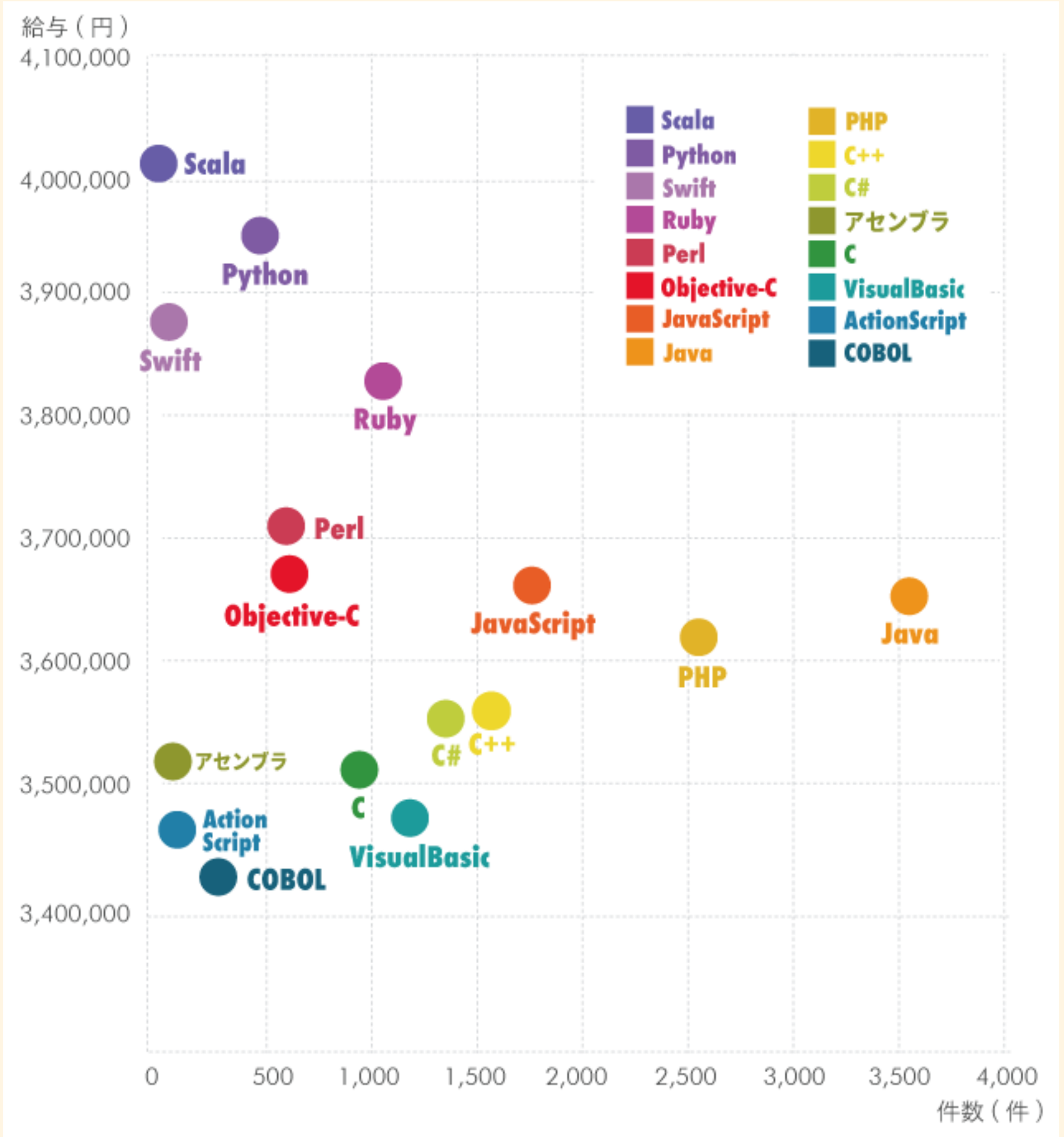
SmartNews

TIS IT Holdings Group

Spark

Hatena

PayPal

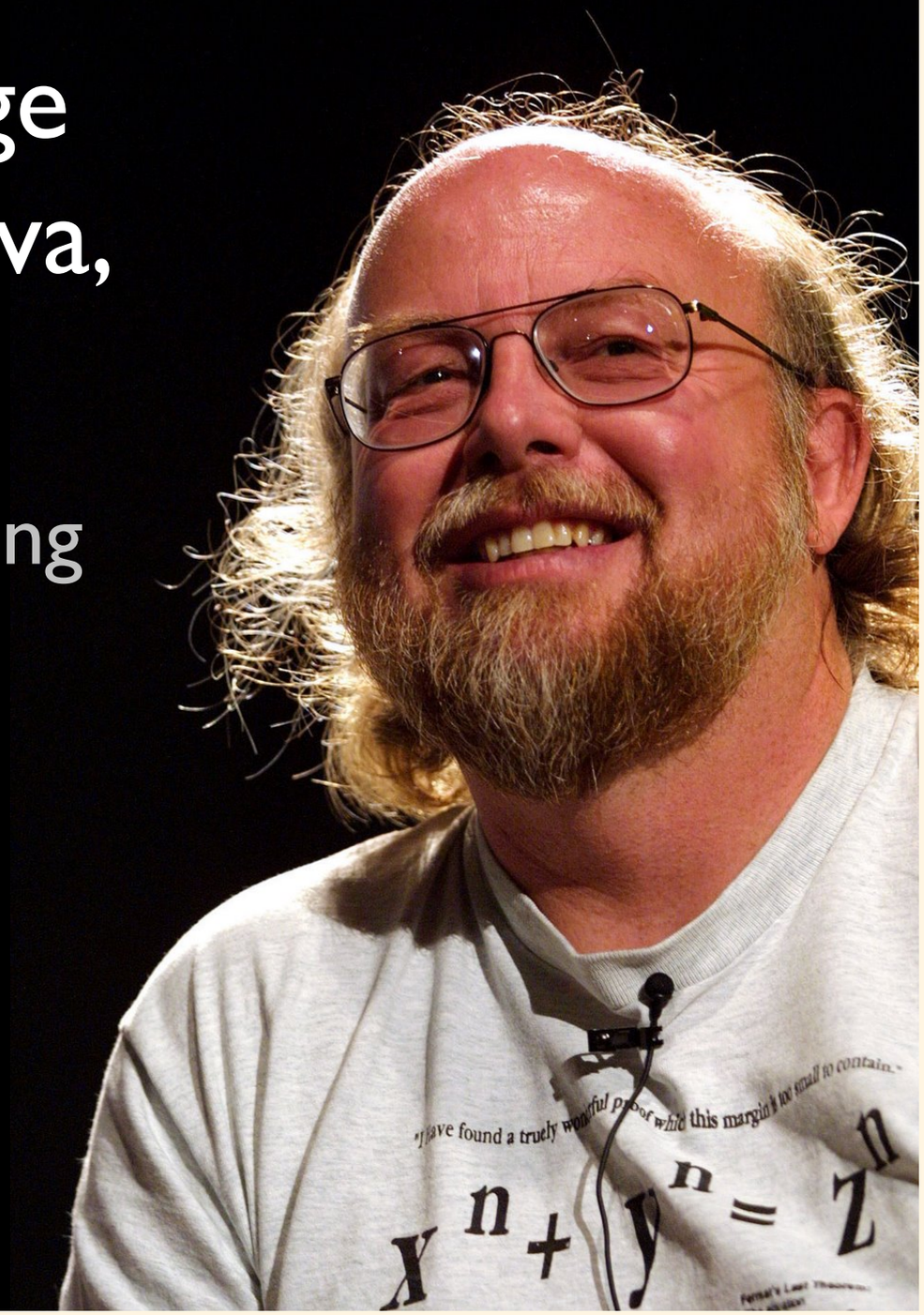


http://hrog.net/20151026249 91.html

ゴスリングのおっちゃんだっ
てベタ褒めですがな！

"If I were to pick a language to use today other than Java, it would be Scala"

James Gosling



CREATOR OF Scala

- Martin Odersky
  (マーティン・オーダスキー)

スイス連邦工科大学ローザンヌ校教授

初期のjavac開発者であり、Generics
仕様の提唱者


Scala = Scalable!
ワンライナーから１００万行
のコードまで対応!!

Scalable =

Functional + Objective

オブジェクト指向＋関数型

- 実はScala言語に限らずごろごろ存在

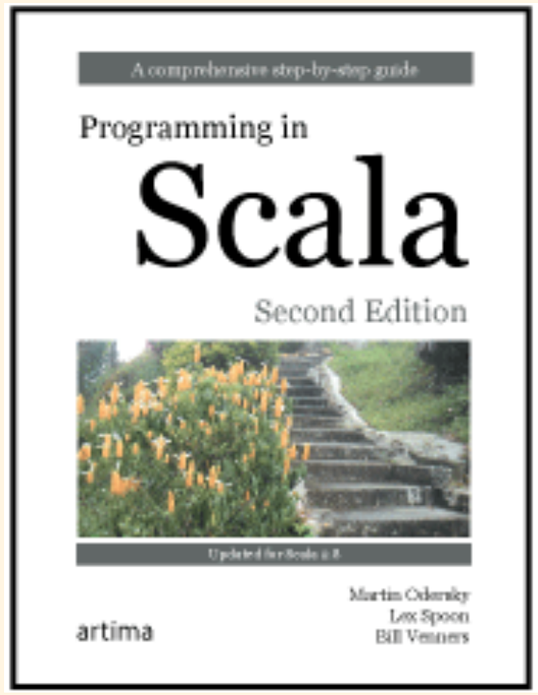
  Ruby
  JavaScript　→　実はチョー関数型言語！
  Groovy
  Python

  LL言語に多く見受けられるのは偶然？

  本格的な関数型言語なのか？という議論はある

James Strachan(Groovy作者)

- I can honestly say if someone had shown me the Programming in Scala book by by Martin Odersky, Lex Spoon & Bill Venners back in 2003 I'd probably have never created Groovy.



関数型言語ことはじめ

- いつも使ってる(作ってる)じゃないですか?

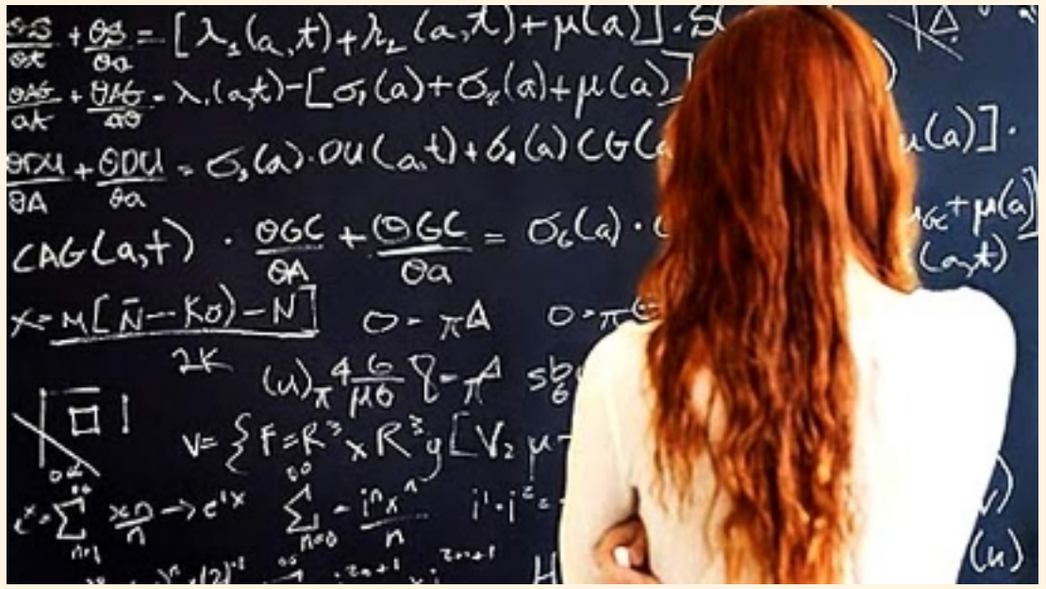
```
int func(int a, int b, int c) {
        xxxxxx
}
```

- じゃあ今日の話はオシマイ！

関数とは？

- 一度関数とかメソッドとかの概念を頭から消してください！！！
- 数学における「関数」にむしろ近い概念

  →　ルーツが数学から来ているから



ちなみに

- むかーしは、「函数」と書いていた

数学専門の世界だと 「関数」だけで1つの研究分野として 成立している

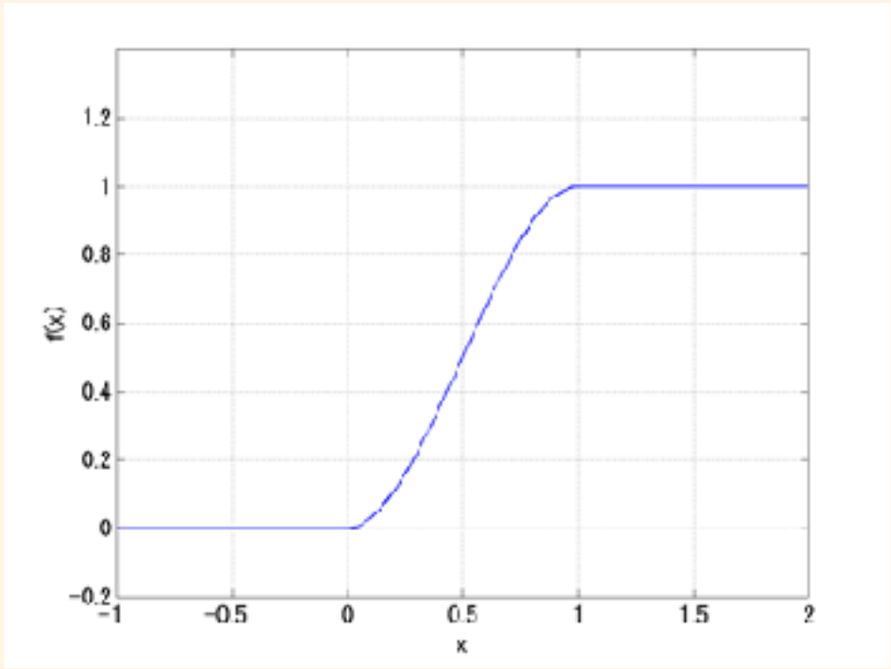
今日のところは

- 数学での定義についての深い議論は**もちろんしません**
- まず、最低限プログラミングの世界一般で流通している関数の概念はここで**一旦リセット!**
- 少しばかり端折ったり、簡略化しているけど、今回の場合の関数(数学上)の定義は・・・

関数とは？

入力値　　結果値

同じ入力値ならば
常に
結果値は同じ



ひっこいけど関数とは？

結果値を決める手がかりは

入力値だけである

関数型言語のパラダイム

状態を持たない

疎結合

状態を持たないとは？

- 入力値以外に結果値を決定する要素は存在しない

副作用と呼ぶ

- かつ、**結果値以外に影響を与える対象**は存在しない

（天気が良いから）
４本で２３０円！
持ってけドロボー

状態を持たないとは？(続)

- さきほどのトウモロコシの価格を決めるのを関数と考えると天気は入力値に入っているべき
- 関数内部では「天気の状態」は保持しない！
- テストの時に条件を単純化できる

疎結合

- 「昔々、あるクラスで他のクラスを使おうとして...」
- テストの時だけ別のクラスに入れ替えて使いたい、とか...
- DIコンテナとかなんとかかんとか...
- 関数型言語では言語の性質として疎結合となる！

ファーストクラスオブジェクト

- 第一級オブジェクトとも呼ぶ
- 変数に代入できる
- 関数の引数として渡せる
- 関数の戻り値として返せる

高階関数(higher-order function)

- 関数をファーストクラスオブジェクトにできる場合、その関数を高階関数と呼ぶ
- 仮にfunc1(func2(123))としていてもfunc2の結果の値をfunc1に渡しているのに過ぎないから**高階関数とは言わない！**

関数型言語とは

- 簡単に言えば、高階関数をサポートしている言語
- では「関数型言語には、関数型プログラミングのために、まずは第一級関数が必要不可欠である。」と記述されている。

- 第一級関数＝関数をファーストクラスオブジェクトとして扱えること

関数型プログラミング

- **関数と関数間の相互作用を掲示することがプログラミングとしての行為**

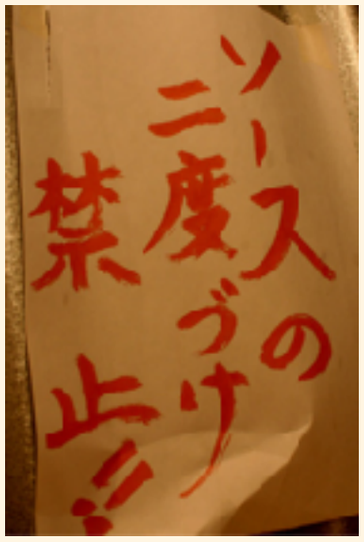
- **副作用が生じることは好まれず、むしろ禁じられる**と考えるのが良い

- こういった前提を基に検討をしていくと**決定的な概念**が導かれる!

変数は再代入不可！！

- 入力値が出力値に対して一意に決まっているならば、一旦決まった値が変わるはずがない！

- なので、変数は値に対する**ラベル**みたいなもの



束縛
と呼ぶ

参照透明性

- 仮に、f(10) = 123だとしたら

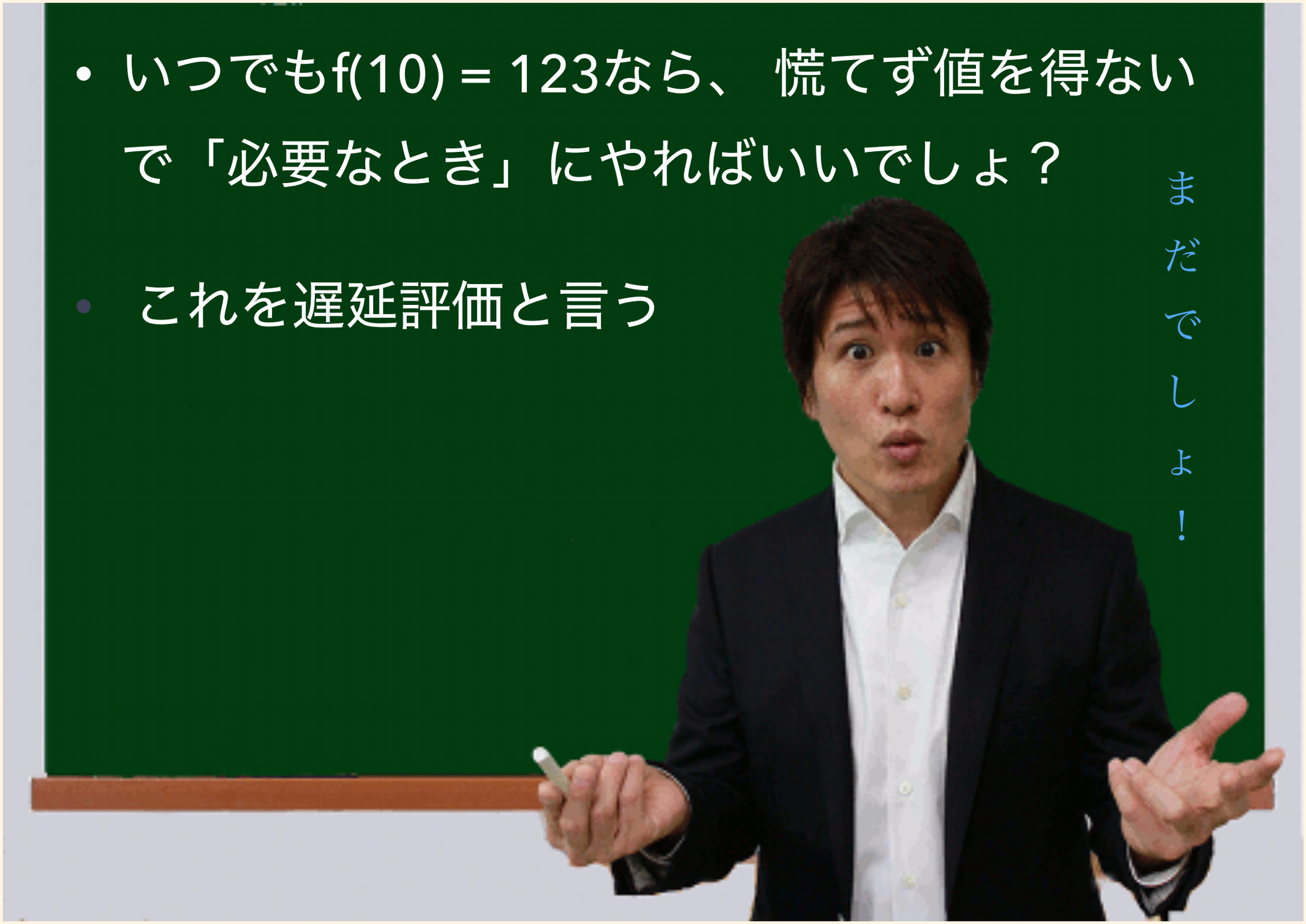
  (関数の中身はここではどうでもいいので略)

- いつでもどこでもf(10) = 123である。

  つまり、f(10) → 123 に置き換えて良い!

- これを参照透明性と言う

遅延評価


• いつでもf(10) = 123なら、 慌てず値を得ないで「必要なとき」にやればいいでしょ？
• これを遅延評価と言う
まだでしょ！

欠かせない概念

- 再代入不可
- 参照透明性
- 遅延評価

ただし、現実の関数型プログラミング言語が 必ずこれら3つを備えているか、その使用を 強制されるのかというと、実はそうでもな い。ガチガチで行ってしまうと不便なケース も現実にはあるので。。

関数型 vs 非関数型

非関数型

```
y = f(123)

r = g(y)

r = r + 1
```

再代入！

関数型

```
r = add(g(f(123)),1)
```

並列処理(関数型言語)

- 参照透明性と遅延評価によって逐次処理による制約を乗り越える試み

  →性質上並列処理と相性がいい

- 再代入禁止によるメリット

  →メモリへのアクセスで排他制御が必要ない

純粋関数型言語

- これまでに述べた関数型言語の特性

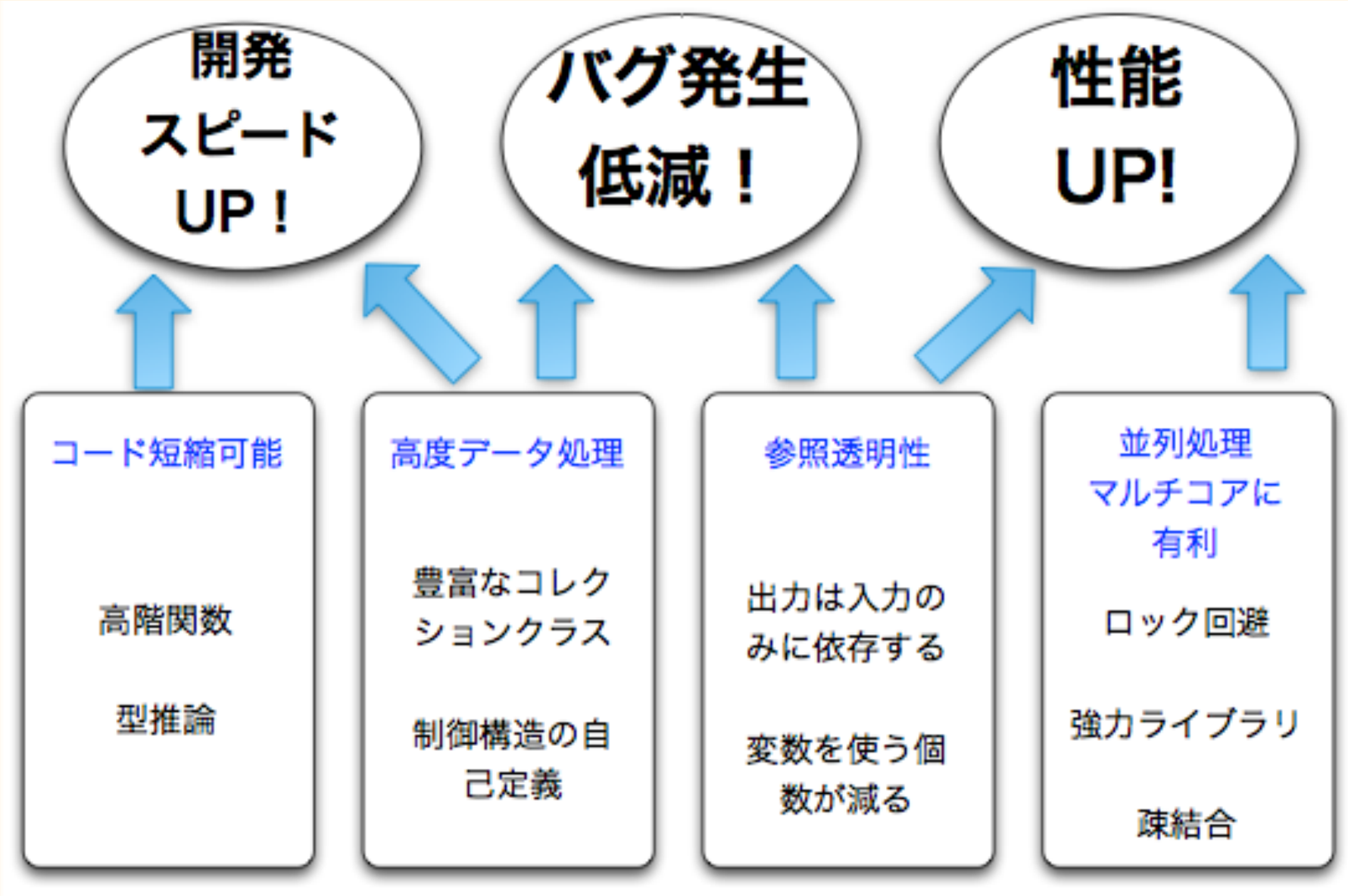
  →厳密に守るとかなり窮屈

  →画面出力もファイルI/Oも**副作用になる**

非純粋関数型言語

- 現実的には制約を緩めて言語仕様を定めていることが多い
- Scalaも非純粋関数型言語

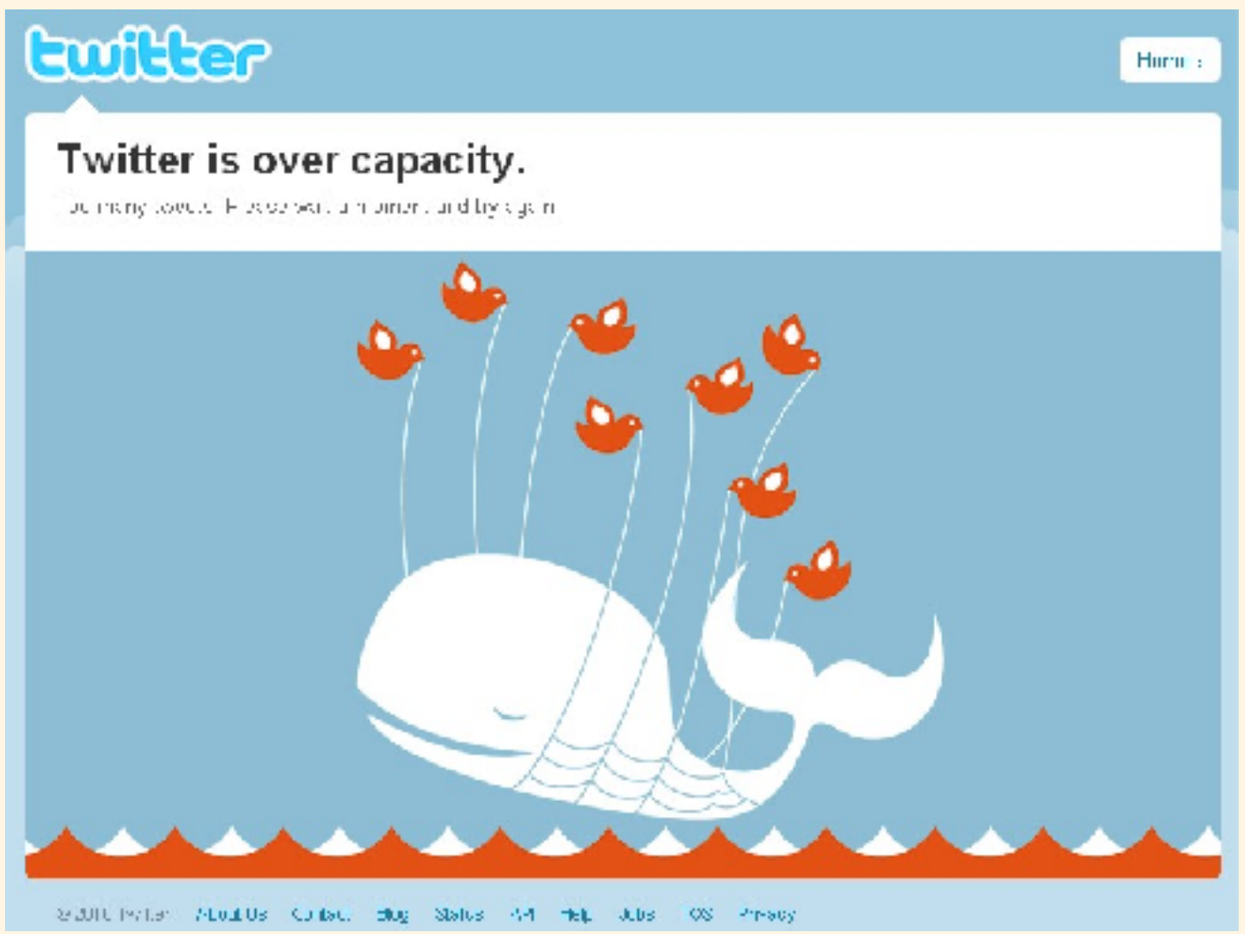
関数型言語の狙い


開発
スピード
UP！
バグ発生
低減！
性能
UP!
コード短縮可能
高階関数
型推論
高度データ処理
豊富なコレクションクラス
制御構造の自己定義
参照透明性
出力は入力のみに依存する
変数を使う個数が減る
並列処理
マルチコアに
有利
ロック回避
強力ライブラリ
疎結合

活用事例(１)

- 最近見なくないですか？

- Ruby on Railsから置き換え



活用事例(２)　Scalaではない

- 通信処理制御のためにErlang言語を設計
- ロバストなマルチスレッド処理を実現
- 99.9999999%の可用性を誇るライブラリ

  「OTP(Open Telecom Platform)」

活用事例(３)　Scalaではない

- 時価会計パッケージ「BancMeasure」を開発
- 開発規模は約２万３０００ステップ
- 金融機関２５社以上が採用
- 品質重視→バグがあると**有価証券報告書訂正へ**
- **javaと比べてコーディング量は５分の１**

活用事例(３)続

- 採用言語はもっとも純粋な関数型言語と言えるHaskell

ゲーテは言いました

- 「外国語を知らない人は、母国語を知らない」

(おまけ)JavaScript

- 実はjavaよりはるかに関数型言語っぽい
- JQueryなんかはそのエッセンスに満ちあふれている

ようやくScalaに注目！

Scalaの特色(1) JVM

- Javaクラスをそのまま使える
- 可用性等の実績はJavaのを転用可
- プロファイリングが容易
- 時間とカネをかけた実行系に載っかれる

ちなみにScalaはBSDライセンス

Scalaの特色(2) 型推論

- 型が無い言語＝エラーかは実行依存
- 型がある言語＝きっちり型決めろ～
- 型推論＝コンパイラがどの型かを推測

```
val formatter = new
SimpleDateFormat("yyyy/MM/dd HH:mm")
```

Scalaに型は、「ありま～す」

Scalaの特色(3) val

- 再度代入できない「変数」はvalで宣言
- 値変更が必要ならvarで宣言→驚くほど頻度少！
- valだとロックがない→並列化にグッド！
- valに慣れるとコードも綺麗になるぞ！
- valに慣れるとテストもラクになるぞ！

Scalaの特色(4) タプル

- 複数の値をカンタンにひとまとめ！
- メソッドの返り値に超便利に使える

```
scala> val t = (123,"abc")
scala> println(t._1)
123
scala> println(t._2)
abc
scala> val (a,b) = t → aとbそれぞれに値が入る
```

Scalaの特色(5) XML

- XMLリテラル

```
val x1 = <sample>hoge</sample>
val x2 =
<sample>
  <zzz>test</zzz>
  <foo/>
</sample>
```

- もちろんノードをパースしてあんなことやそんなことやえーそんなことも！！できちゃうの？！というメソッドなどはたくさん用意されてございます

Scalaの特色(6) Option

- Scalaではnull使いは野暮！！！
- Option[String]型
  中味ある→Some(value)という値
  スッカラカン→Noneという値
- スッカラカンのときの対策を強制できNull例外は回避可能
- Swiftでも取り入れられている

Scalaの特色(7)　パターンマッチ

- switchよりはるかに強力！

```
def matchTest(x: Any): Any
= x match {
    case 1 => "one"
    case "two" => 2
    case y: Int => "scala.Int"
  }
```

- if文を使う頻度が少なくなる

Scalaの特色(8)　Case Class

- こんなん(valフィールドになるのがデフォルト)

```
case class PaymentIDInfo(
        paymentID: String,
        status: String,
        transactionUrl: String,
        orderedTime: String)
```

- コンストラクタ・equals・hashCode・toString・copy メソッドを自動生成

- newいらずでオブジェクト生成可

Scalaの特色(9)　トレイト

- Javaのインターフェイスと異なりメソッドの中味（実装）そのものも突っ込める
- RubyではMix-Inと呼んでいる。多重継承問題へのScalaなりの解答
- 使いこなせるとかなり強力

Scalaの特色(10)　コレクション

- 高階関数を駆使
- 例えばmapメソッド(写像)

```
List("Book",
          "Car","Apple").map { _.length }
```

結果↓

```
List(4, 3, 5)
```

コードが、短くな〜〜〜る

```
import java.util.ArrayList;
public class Person {
  public final String name;
  public final int age;
  Person(String name, int age) {
      this.name = name;
      this.age = age;
  }
}
...
Person[] people;
Person[] minors;
Person[] adults;
{  ArrayList<Person> minorsList = new ArrayList<Person>();
   ArrayList<Person> adultsList = new ArrayList<Person>();
   for (int i = 0; i < people.length; i++)
       (people[i].age < 20 ? minorsList : adultsList)
           .add(people[i]);
   minors = minorsList.toArray(people);
   adults = adultsList.toArray(people);
}
```

人に関するクラス「Person」の宣言

大人や未成年などのオブジェクトを宣言

20歳未満なら未成年に分類

Javaのコード例
（人のデータの集合を未成年かどうかで分類する処理）

コード行数が約1/5に

```
class Person(val name: String, val age: Int)
val people: Array[Person]
val (minors, adults) = people partition (_.age < 20)
```

人に関するクラス「Person」の宣言

20歳未満なら未成年に分類

Scalaのコード例

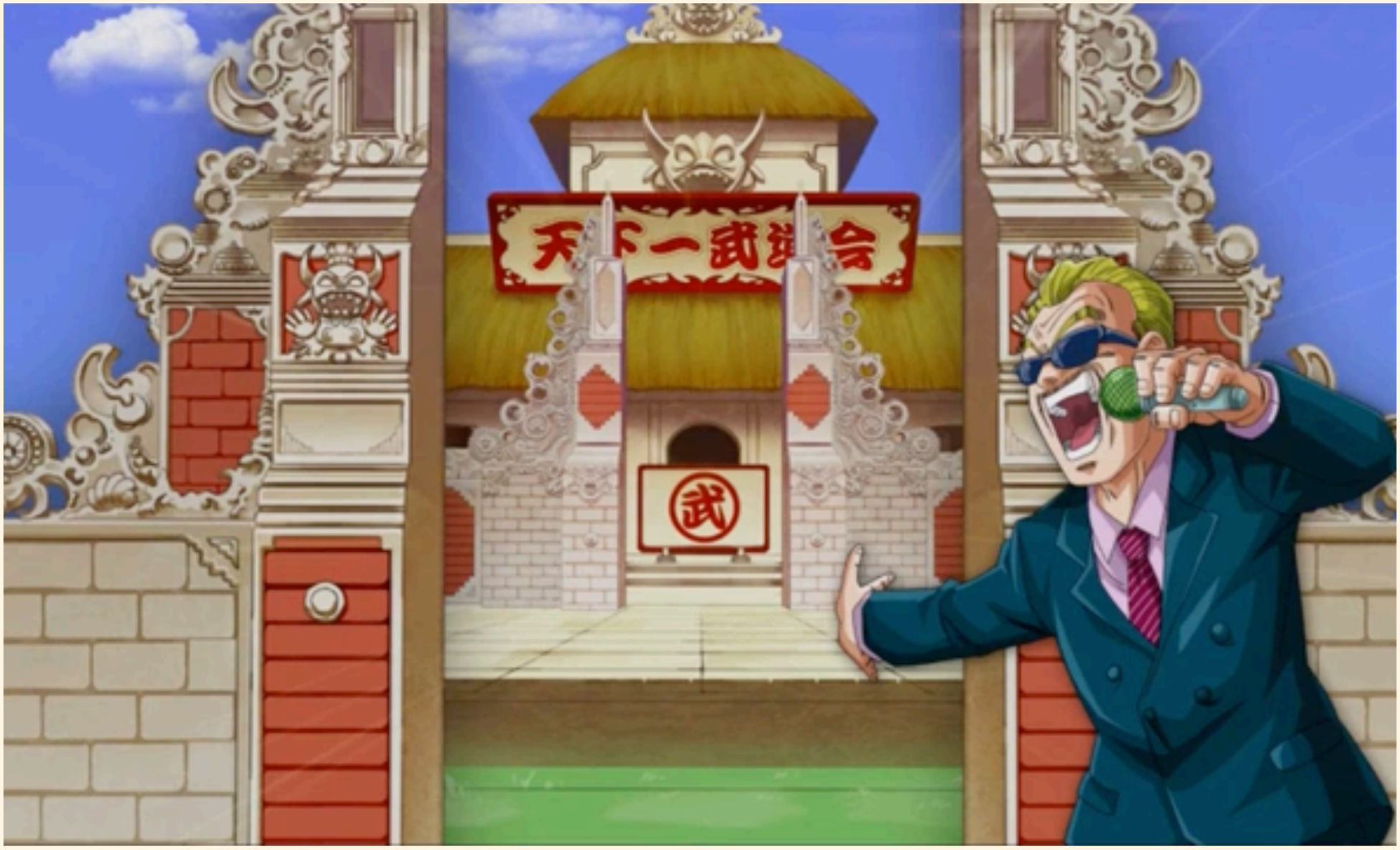
日経コンピュータ2012/9/27号より

バグよさらば！

- つまりは関数型故に
- 疎結合になる！
- 「変」数が減る
- テストケースも減る
- コーディングステップも減る

実行速度も速くなる

- マルチコア・スレッドに有利
- 磨き込まれたライブラリが使える
- ロックによる速度低下を回避
- ついでにロバストネスの獲得



ランゲエジ 武道会

Scala vs Java,PHP,C#,VB,Python and Javascript

	Functional	Robust	Type	Fast	Compact code	Cool	Learning difficulty
Scala	◎	◎	○	○	◎	◎	5
Java	○ 8.0	◎	○	◎	△	△	4
PHP	○ 5.3	×	×	◎	△	△	3~4
Ruby	◎	×	×	△	◎	◎	3~5
C#	○ 6.0	◎	○	○	△	○	4
VB	△	◎	△	○	△	×	2~4
Python	◎	△	×	△	○	◎	3~4
Javascript	◎	△	×	△	○	◎	2~5

試合結果発表のひとりごと

JAVAer,PHPer,C#er,VBer,PYTHONer and JSer

Java

Javaとは兄弟

JVMを使わせてくれてありがとう（オラクル様）

Javaのクラスライブラリも使えるし

Java8はScalaのマネだとは申しません

PHP

スパゲッティがお好きでしょ？

Ruby

実行するまでエラーは見たくない？
不安定な実行環境で頑張ってみたい？
ライブラリの互換性に困ってみたい？

C#

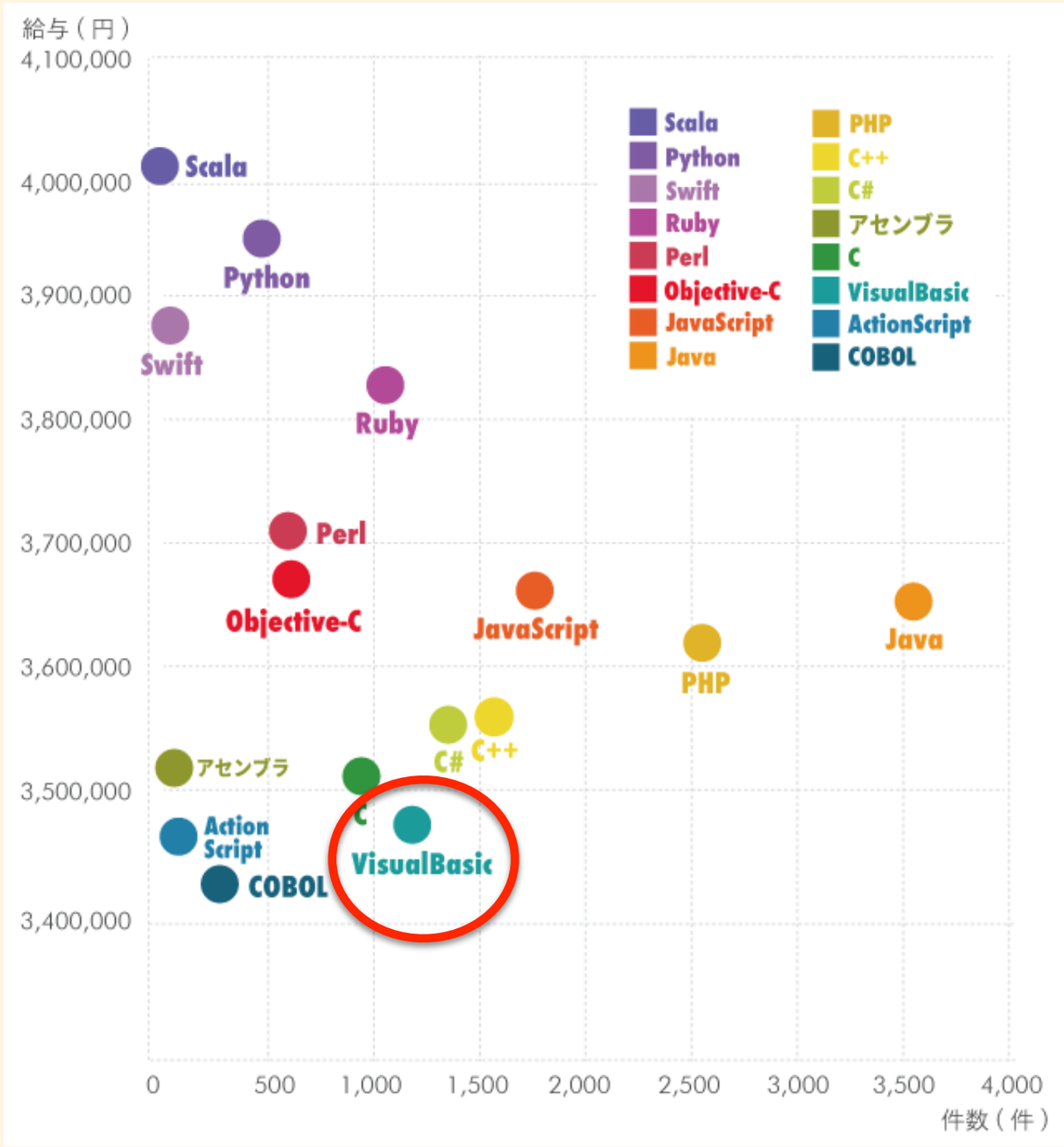
Windows使ってますよね？

サーフェース持ってますよね？

MSDNにお金払ってますよね？

転職、しないですよね？

Visual Basic



Python

インデントは何文字にしてますか？

ま、一番熱い言語ですよね、最近

Javascript

猫も杓子も

やってるよ！

まとめ

- Scala = 関数型とオブジェクト指向を融合
- カンタンに覚えられるとは言いがたい
- コード行数を削って生産性向上
- まずは試して楽しみましょう

play framework行きます！

play framework

最新Ver 2.5.12

- 様々な先行フレームワークのエッセンスを継承
- 「Ruby on Rails以後」のフレームワーク
- RoRの影響は確かに大きく受けている
- JavaとScalaで利用可能

Apacheライセンス

様々な先行フレームワーク

- Django(Python)
  強力なO/Rマッパーの提供

- Apache Wicket(Java)・Turbo Gears(Python)
  式言語の追放(cf. JSP)

HTMLと混じってカオス！

- Ruby on Rails(Ruby)
  高速開発のパラダイム。CoC(convention over configuration)
  XML地獄の回避

「Ruby on Rails以後」

- DRY
- CoC
- フルスタックフレームワーク
- 強力なO/Rマッパー
- コード量削減による高速開発

「Rails以後」は
当たり前

でもRoRのマネっこではない

- Cake PHPのようにRoRをリスペクトしているが、中味をマネしているわけではない
- Java WorldではStruts、Spring、Seaserとかしか選択肢が無い
- Grails使ってみます？

play framework

- RESTfulである
- ステートレスである
- フルスタックフレームワークである
- 強力なO/Rマッパーもある
- いわゆるMVCである

play framework - MODEL

- Slickという推奨O/Rマッパーが提供される
- RDBMSはJDBCを使いその上にレイヤー
- NoSQLにも対応可能
- データ構造はケースクラスを使って簡潔に
- スキーマからデータクラスを自動生成も可能

play framework - VIEW

- TwiriというScala実装のテンプレートエンジンが標準
- @()や@{}といった記述でコーディング指定
- ビルド時にコンパイルされる
- なんでもかんでもコードを入れ込むのは難しい
  (JSPやPHPで昔、あるいは今も>*０*<キャアアッ)
- JS＆JQuery&AJAXでいろいろやるのがモダンスタイル

play framework - CONTROLLER

- URLとコントローラーのマッピングはハッキリ指定

- Playではサーブレットは動かない！

- 認証などは優れたライブラリを利用するのが吉

- テスト実行時は専用コントローラー使用可

play framework - ETC

- DIコンテナが含まれている
- デバックモードと実稼働モードがある
- HTTPサーバーも含まれている！
- HTTPSは通常はリバースプロキシ使用

play framework - IDE

- エディタ＋コマンドラインでも可能は可能
- Eclipseでも可能
- IntelliJがオススメ
- play向け拡張使用は有料

ご清聴ありがとうございます