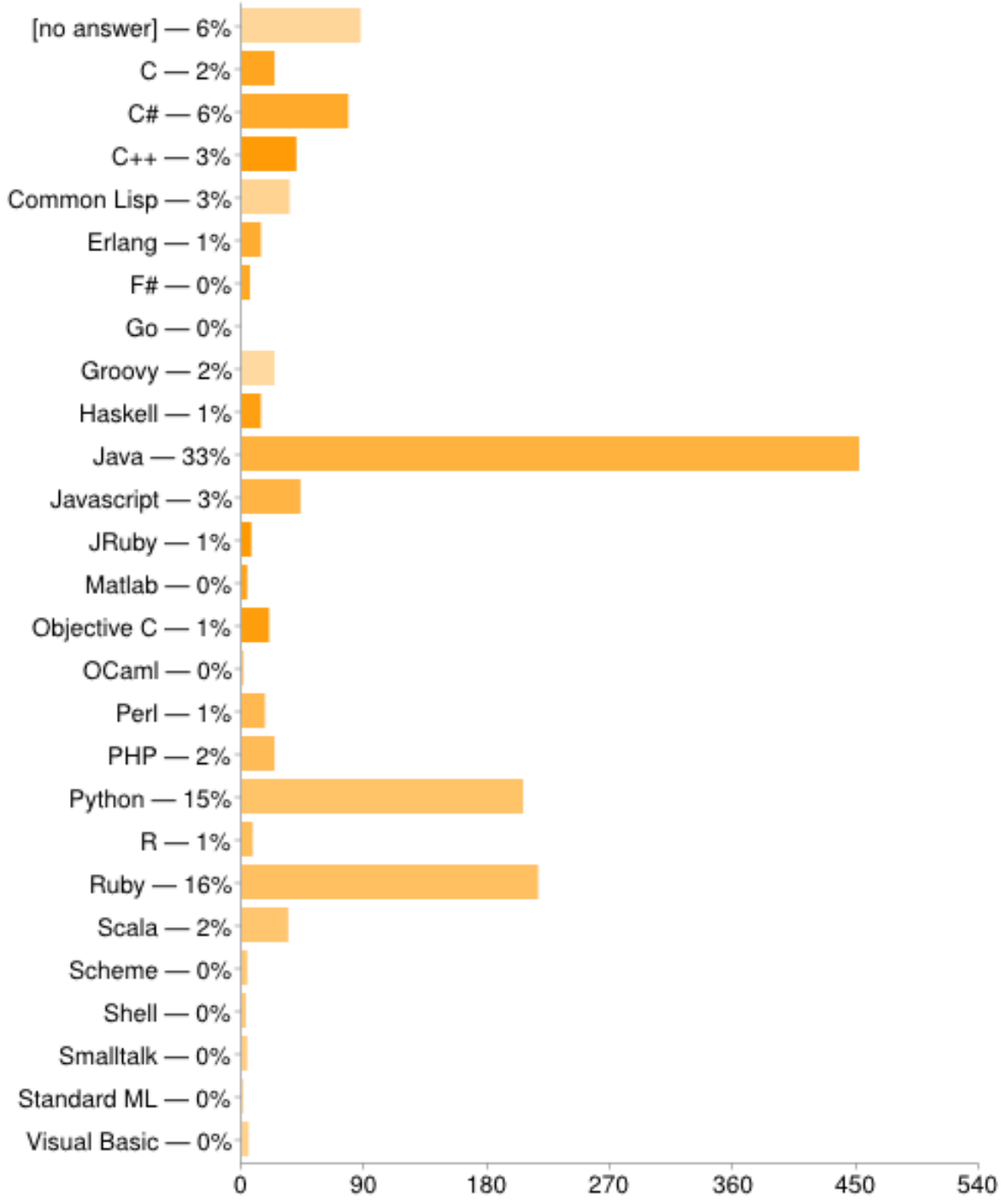
Clojure

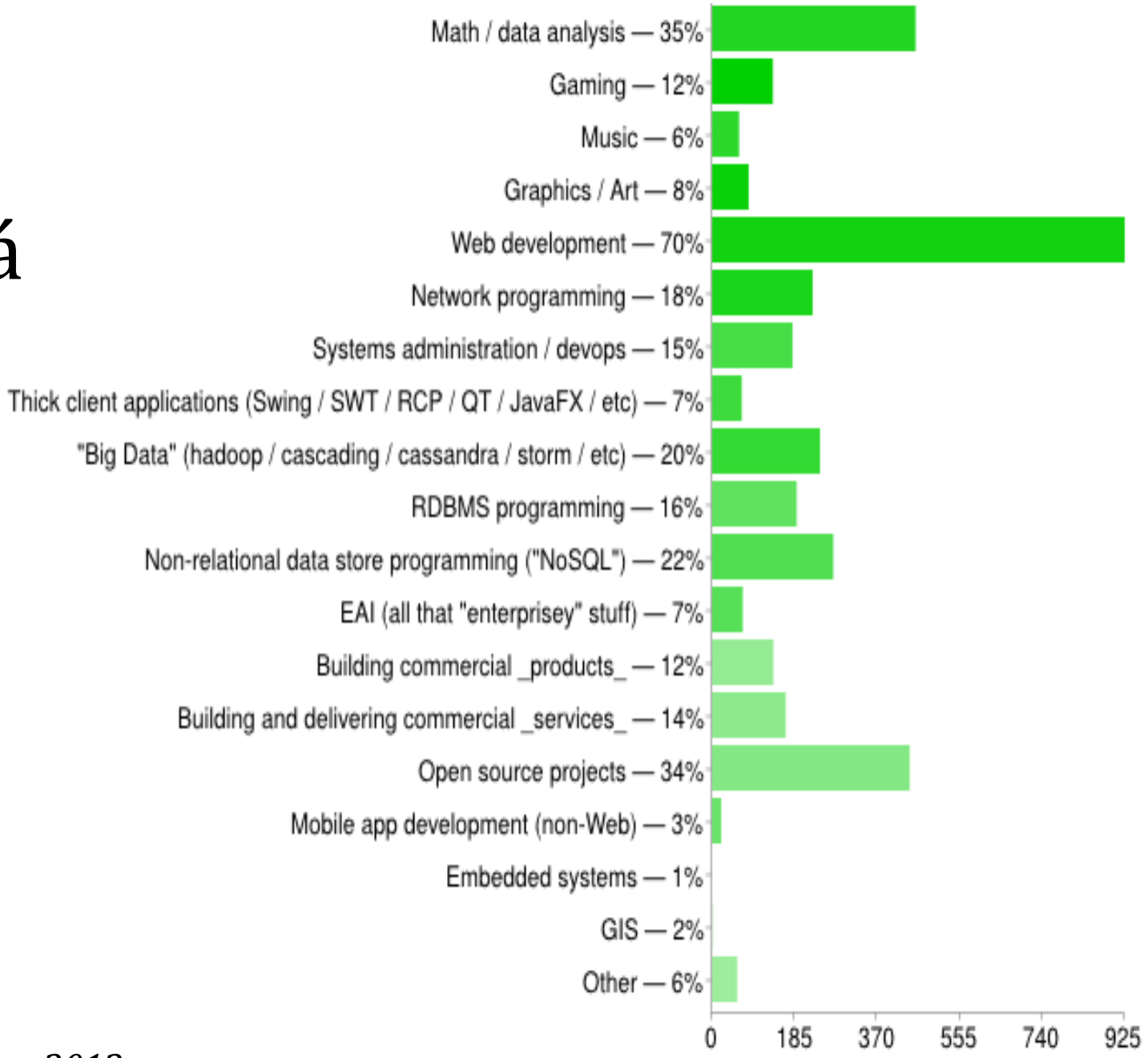
em busca de simplicidade

Qual linguagem você usava antes de adotar Clojure?



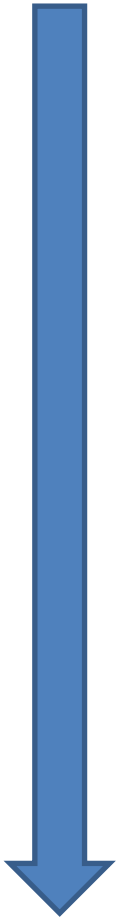
State of Clojure Survey 2012

Em qual domínio você está usando Clojure?



State of Clojure Survey 2012

remix disclaimer



2008 – Clojure, JVM Languages Summit

2009 – Are we there yet?, JVM Languages Summit

2011 – Simple Made Easy, Strange loop

2012 – The value of values, GOTO

Agenda

O que é simplicidade?

Idéias que simplificam

Introdução a linguagem

“Simple is better than complex.”

“Simple is better than complex.”

“Simplicidade é pre-requisito para um sistema confiável"

Edsger W. Dijkstra

simples (adj.) do Latim *simplex* ‘único’, de uma base Indo-Europeia *sem-* ‘um, único’ + *plicare* ‘dobrar’: aquilo que não apresenta complicações para ser aberto

simples (adj.) do Latim *simplex* 'único', de uma base Indo-Europeia *sem-* 'um, único' + *plicare* 'dobrar': aquilo que não apresenta complicações para ser aberto

1

conceito
tarefa
dimensão
papel
etc

simples (adj.) do Latim *simplex* 'único', de uma base Indo-Europeia *sem-* 'um, único' + *plicare* 'dobrar': aquilo que não apresenta complicações para ser aberto

1

conceito
tarefa
dimensão
papel
etc

Ausência de
entrelaçamentos
- não sobre *cardinalidade*

Simples vs complexo

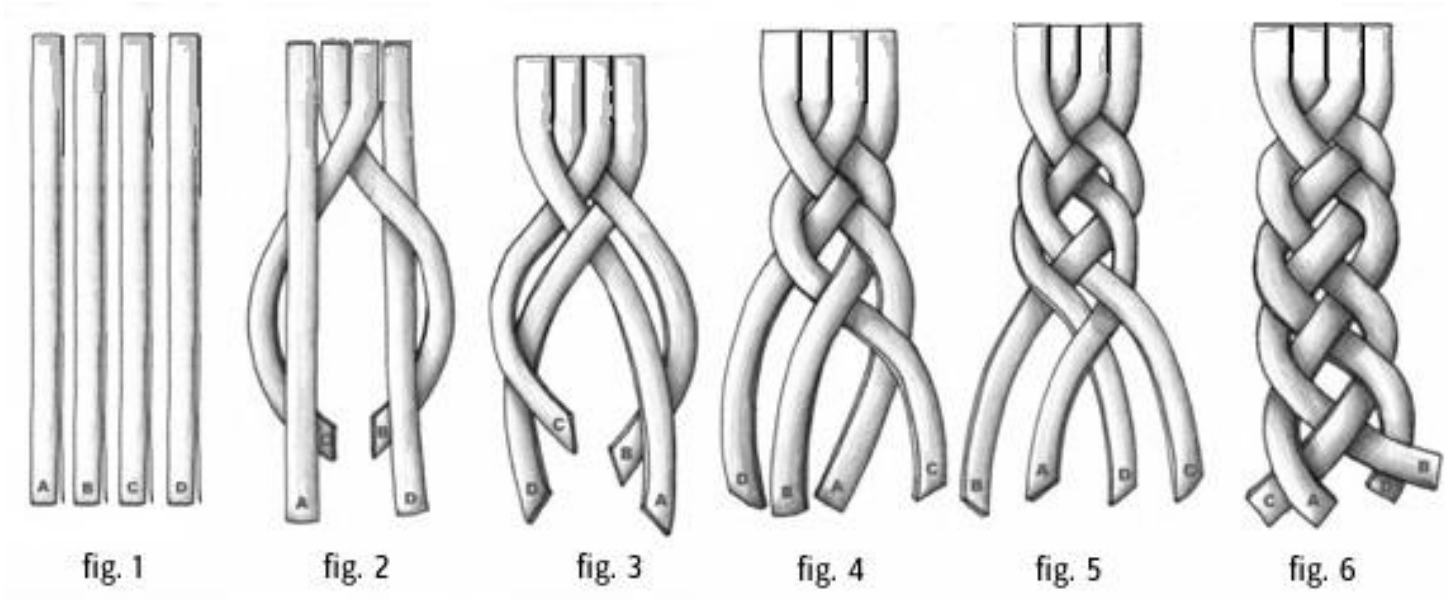
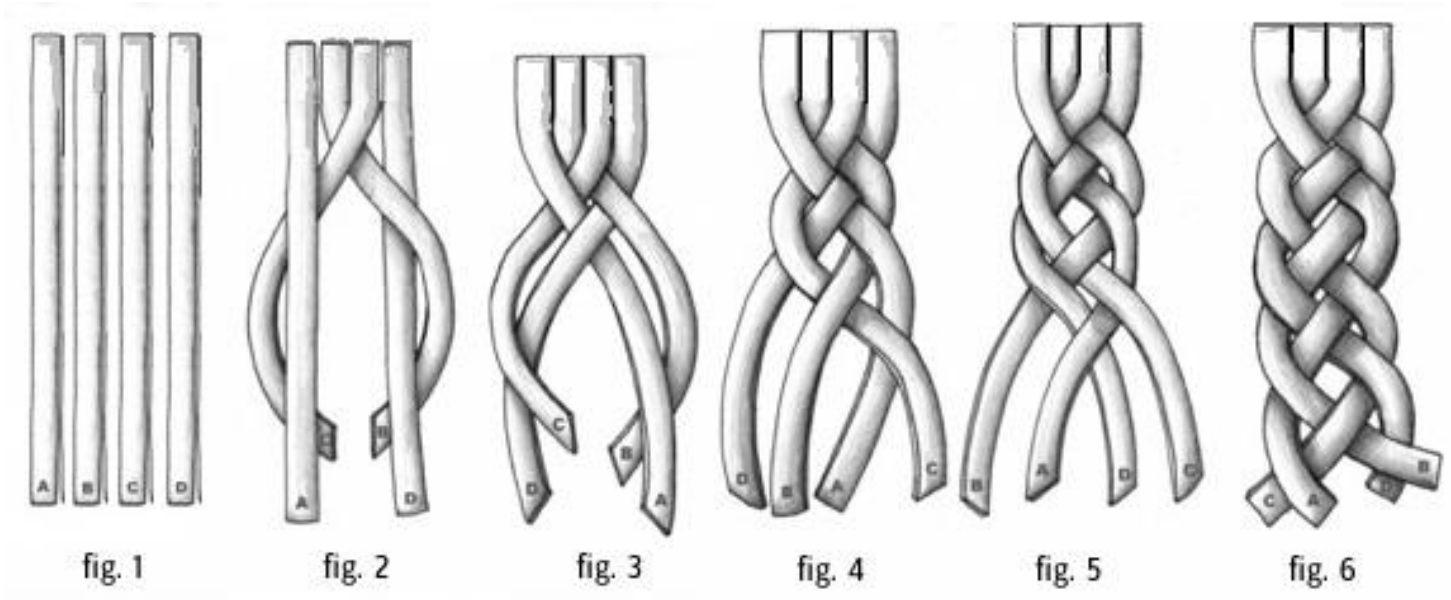


fig. 1 fig. 2 fig. 3 fig. 4 fig. 5 fig. 6

Simples vs complexo



Objetivo

fácil (adj.) do Francês *aisie* ‘conforto, sem esforço’ e relacionado com Latim *adjacens*, ‘estar perto’

fácil (adj.) do Francês *aisie* ‘conforto, sem esforço’ e relacionado com Latim *adjacens*, ‘estar perto’

Perto do alcance, acessível

já está instalado, apt-get install, pip install...

fácil (adj.) do Francês *aisie* ‘conforto, sem esforço’ e relacionado com Latim *adjacens*, ‘estar perto’

Perto do alcance, acessível

já está instalado, apt-get install, pip install...

Perto do nosso conhecimento

Familiar

fácil (adj.) do Francês *aisie* ‘conforto, sem esforço’ e relacionado com Latim *adjacens*, ‘estar perto’

Perto do alcance, acessível

já está instalado, apt-get install, pip install...

Perto do nosso conhecimento

Familiar

Perto de nossas capacidades

mas pouca variança aqui

fácil (adj.) do Francês *aisie* 'conforto, sem esforço' e relacionado com Latim *adjacens*, 'estar perto'
oposto de Difícil.

Perto do alcance, acessível

já está instalado, apt-get install, pip install...

Perto do nosso conhecimento

Familiar

Perto de nossas capacidades

mas pouca variança aqui

fácil (adj.) do Francês *aisie* 'conforto, sem esforço' e relacionado com Latim *adjacens*, 'estar perto'

oposto de Difícil.

Perto do alcance, acessível

já está instalado, apt-get install, pip install...

Perto do nosso conhecimento

Familiar

Perto de nossas capacidades

mas pouca variança aqui

Relativo

Linguagem vs Artefato

Linguagem vs Artefato

Abuso de foco no superficial

Linguagem vs Artefato

Abuso de foco no superficial

- Temos que avaliar o que é produzido

Linguagem vs Artefato

Abuso de foco no superficial

- Temos que avaliar o que é produzido

Familiaridade pode mascarar complexidade

- fixados em conveniência

Linguagem vs Artefato

Abuso de foco no superficial

- Temos que avaliar o que é produzido

Familiaridade pode mascarar complexidade

- fixados em conveniência

"programadores sabem os benefícios de tudo mas o custo de nada"

Limites cognitivos

Limites cognitivos

humanos só conseguem considerar poucas coisas ao *mesmo tempo*

Limites cognitivos

humanos só conseguem considerar poucas coisas ao *mesmo tempo*

só conseguiremos tornar confiável aquilo entendemos

Limites cognitivos

humanos só conseguem considerar poucas coisas ao *mesmo tempo*

só conseguiremos tornar confiável aquilo entendemos

coisas entrelaçadas precisam ser consideradas *juntas*

Limites cognitivos

humanos só conseguem considerar poucas coisas ao *mesmo tempo*

só conseguiremos tornar confiável aquilo entendemos

coisas entrelaçadas precisam ser consideradas *juntas*

Complexidade **impossibilita** entedimento.

Benefícios de Simplicidade

Mais fácil de alterar

Mais fácil de entender

Maior flexibilidade

- Alterar políticas, localização

Benefícios de Simplicidade

Mais fácil de alterar

Mais fácil de entender

Maior flexibilidade

- Alterar políticas, localização

Benefícios de testes e type checkers são *ortogonais* a esses

Complexidade acidental (subst.) é aquela não inerente ao problema, sinônimo de '*sua culpa*'.

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores
Métodos	Funções, Namespaces

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores
Métodos	Funções, Namespaces
Argumentos posicionais	Argumentos com nome ou mapa

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores
Métodos	Funções, Namespaces
Argumentos posicionais	Argumentos com nome ou mapa
Vars	Refs gerênciadas

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores
Métodos	Funções, Namespaces
Argumentos posicionais	Argumentos com nome ou mapa
Vars	Refs gerênciadas
Herança, switch, pattern matching	Poliformismo a la carte

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores
Métodos	Funções, Namespaces
Argumentos posicionais	Argumentos com nome ou mapa
Vars	Refs gerênciadas
Herança, switch, pattern matching	Poliformismo a la carte
Sintaxe	Dados

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores
Métodos	Funções, Namespaces
Argumentos posicionais	Argumentos com nome ou mapa
Vars	Refs gerênciadas
Herança, switch, pattern matching	Poliformismo a la carte
Sintaxe	Dados
Loop imperativo, fold	Funções de conjunto

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores
Métodos	Funções, Namespaces
Argumentos posicionais	Argumentos com nome ou mapa
Vars	Refs gerênciadas
Herança, switch, pattern matching	Poliformismo a la carte
Sintaxe	Dados
Loop imperativo, fold	Funções de conjunto
Condicionais	Regras

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores
Métodos	Funções, Namespaces
Argumentos posicionais	Argumentos com nome ou mapa
Vars	Refs gerênciadas
Herança, switch, pattern matching	Poliformismo a la carte
Sintaxe	Dados
Loop imperativo, fold	Funções de conjunto
Condicionais	Regras
Atores	Filas

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores
Métodos	Funções, Namespaces
Argumentos posicionais	Argumentos com nome ou mapa
Vars	Refs gerênciadas
Herança, switch, pattern matching	Poliformismo a la carte
Sintaxe	Dados
Loop imperativo, fold	Funções de conjunto
Condicionais	Regras
Atores	Filas
ORM	Data driven, manipulação declarativa

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores
Métodos	Funções, Namespaces
Argumentos posicionais	Argumentos com nome ou mapa
Vars	Refs gerênciadas
Herança, switch, pattern matching	Poliformismo a la carte
Sintaxe	Dados
Loop imperativo, fold	Funções de conjunto
Condicionais	Regras
Atores	Filas
ORM	Data driven, manipulação declarativa
CRUD	CR-only, Event Sourcing

O que você anda usando?

Complexidade	Alternativas mais Simples
Estado, objetos	Valores
Métodos	Funções, Namespaces
Argumentos posicionais	Argumentos com nome ou mapa
Vars	Refs gerênciadas
Herança, switch, pattern matching	Poliformismo a la carte
Sintaxe	Dados
Loop imperativo, fold	Funções de conjunto
Condicionais	Regras
Atores	Filas
ORM	Data driven, manipulação declarativa
CRUD	CR-only, Event Sourcing
Inconsistência	Consistência

Estado nunca é simples

Estado nunca é simples

Mistura valor e tempo

Estado nunca é simples

Mistura valor e tempo

Porém é muito *fácil*: acessível e familiar

Estado nunca é simples

Mistura valor e tempo

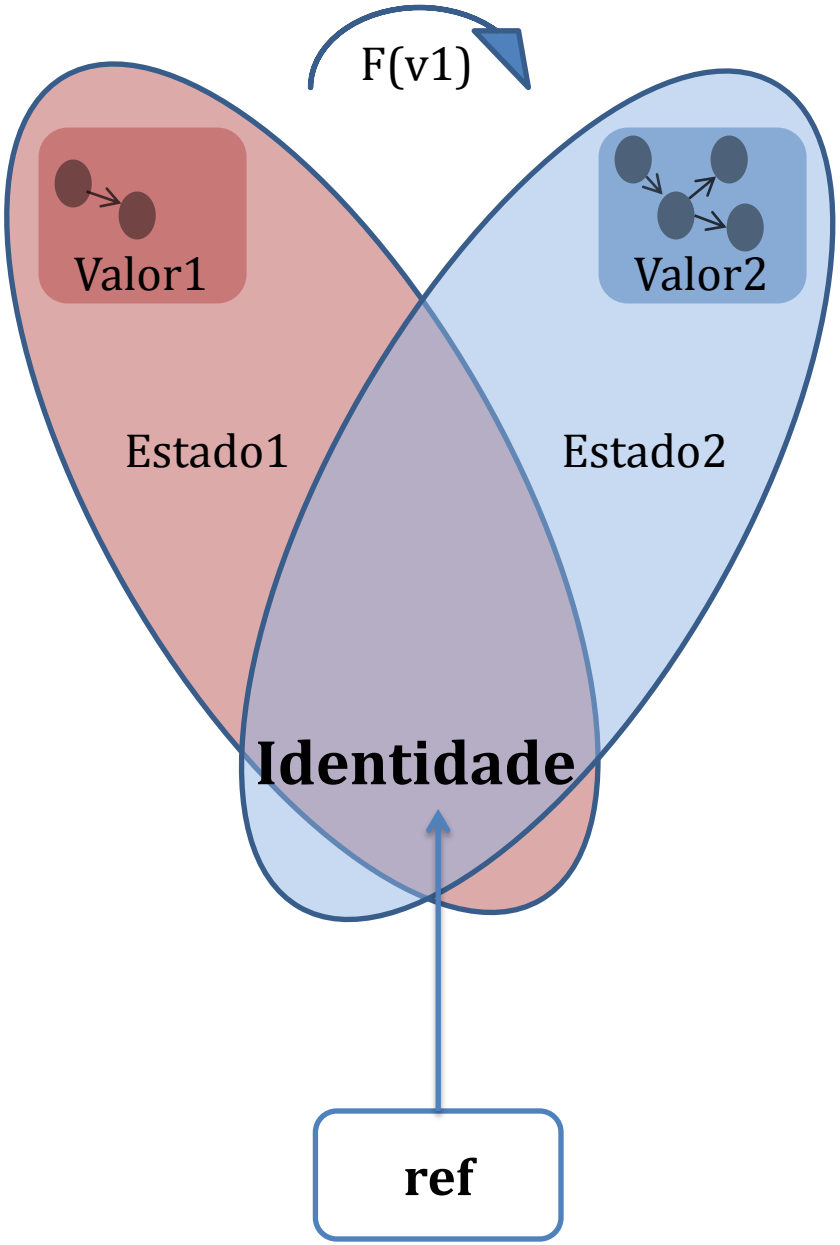
Porém é muito *fácil*: acessível e familiar

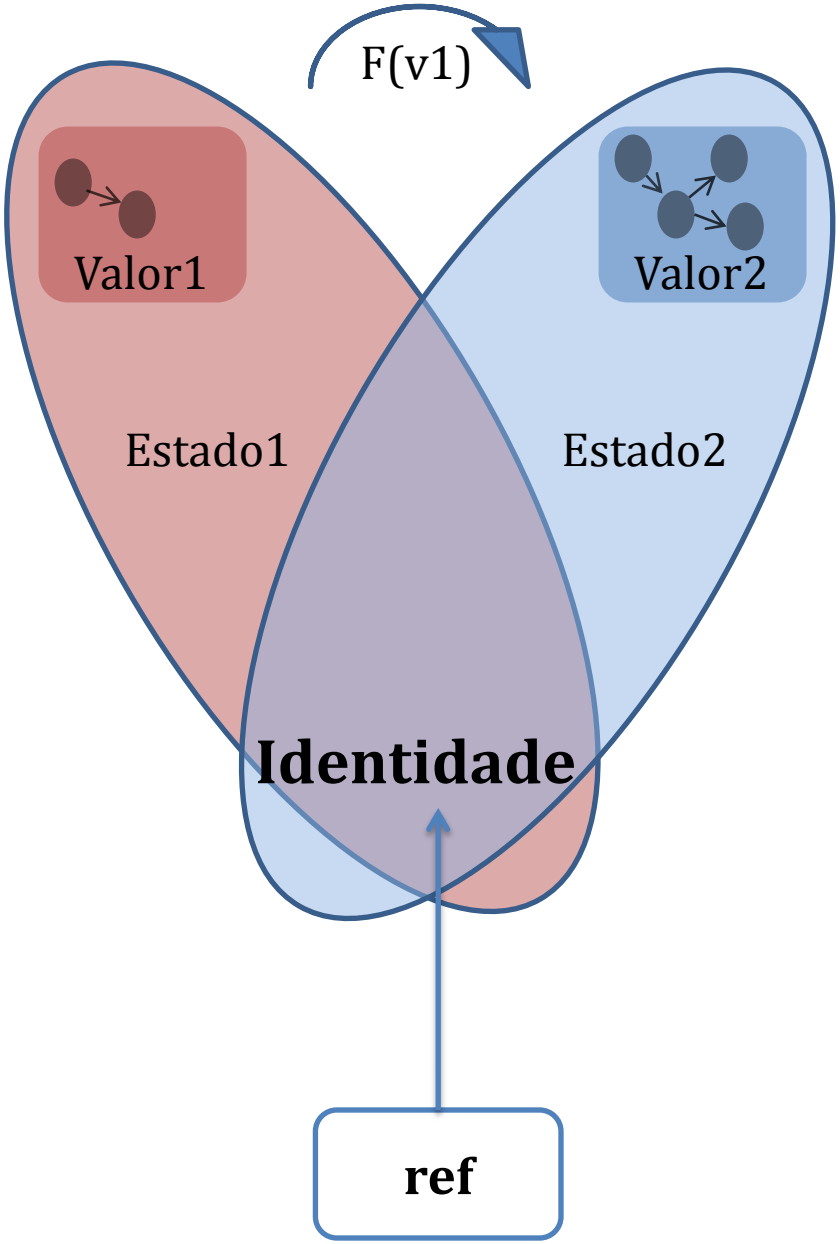
Entrelaça tudo que toca, diretamente ou indiretamente

- Não é possível mitigar por módulos ou encapsulamento

“Simplicity is Opportunity”

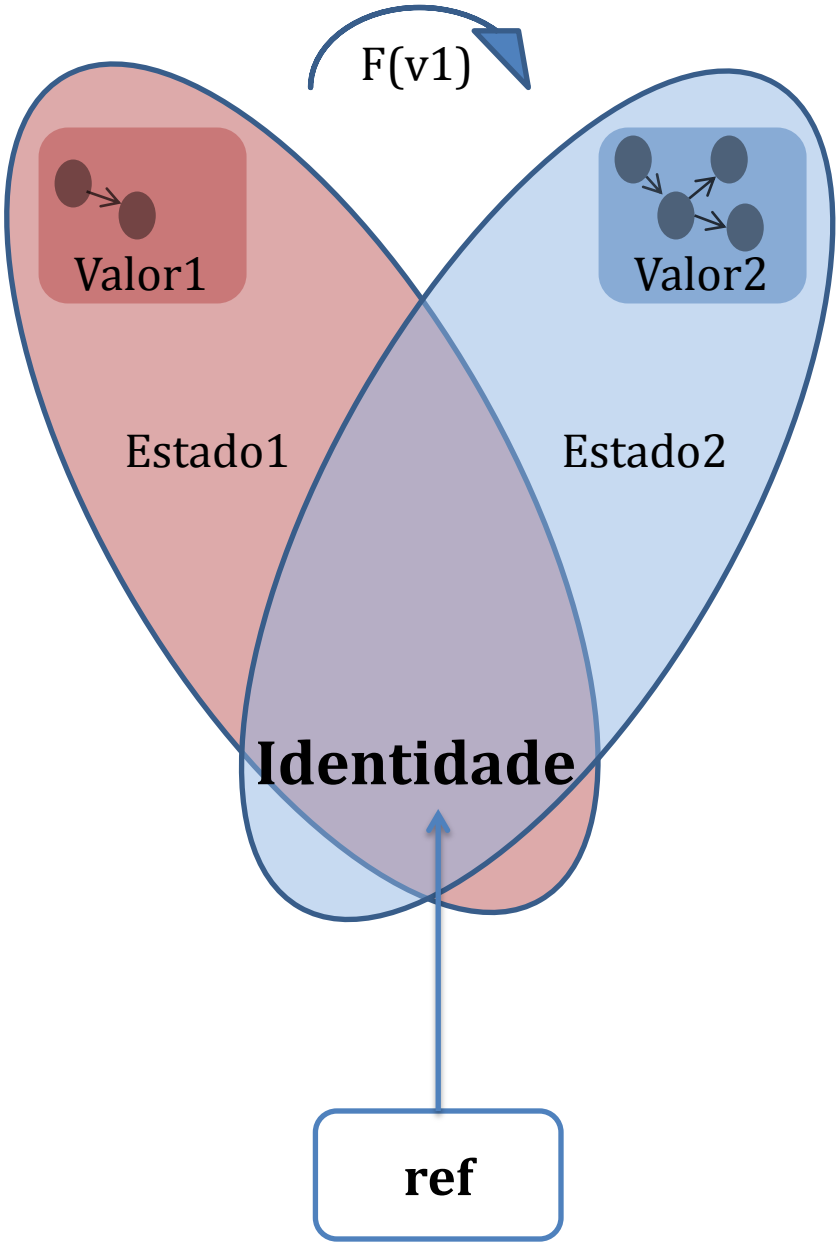
Rich Hickey


F(v1)
Valor1
Valor2
Estado1
Estado2
Identidade
ref



Estado

Valor de uma identidade em um ponto no tempo

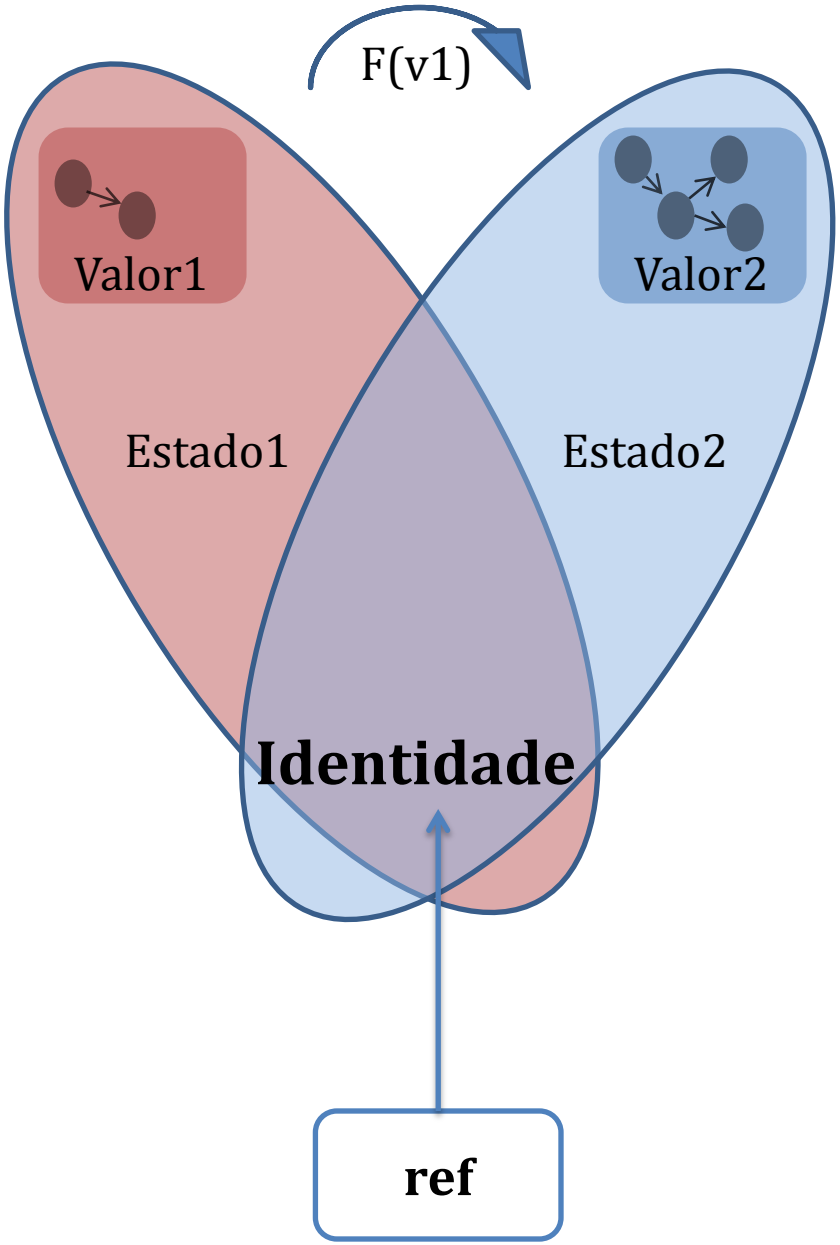


Estado

Valor de uma identidade em um ponto no tempo

Identidade

entidade lógica associada a uma série de estados no tempo



Estado

Valor de uma identidade em um ponto no tempo

Identidade

entidade lógica associada a uma série de estados no tempo

Referências gerenciadas

variáveis com semânticas de coordenação

permitem *compor* valor e tempo

O que é um Valor?

O que é um Valor?

Imutável

O que é um Valor?

Imutável

Agrega

Ex: [“hello word”, 42]

O que é um Valor?

Imutável

Agrega

- Ex: [“hello word”, 42]

Não precisa de métodos

- definidos independente de operações

O que é um Valor?

Imutável

Agrega

- Ex: [“hello word”, 42]

Não precisa de métodos

- definidos independente de operações

São genéricos:

- podemos compartilhar livremente
- representações em qualquer linguagem
- a ferramenta do poliglota!

T.I. hoje em dia...

em memória:

- objetos mutáveis como abstrações para *lugares*
- objetos tem métodos

T.I. hoje em dia...

em memória:

- objetos mutáveis como abstrações para *lugares*
- objetos tem métodos

em storage:

- tabelas/documentos/registros são *lugares*
- DBs tem CRUD

T.I. hoje em dia...

em memória:

- objetos mutáveis como abstrações para *lugares*
- objetos tem métodos

em storage:

- tabelas/documentos/registros são *lugares*
- DBs tem CRUD

PLOP: PLace-Oriented Programming

T.I. hoje em dia...

em memória:

- objetos mutáveis como abstrações para *lugares*
- objetos tem métodos

em storage:

- tabelas/documentos/registros são *lugares*
- DBs tem CRUD

PLOP: PLace-Oriented Programming

nova informação substitui antiga

T.I. hoje em dia...

em memória:

- objetos mutáveis como abstrações para *lugares*
- objetos tem métodos

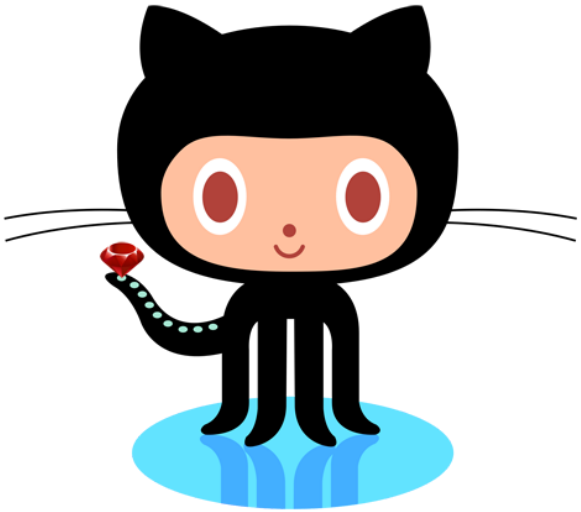
em storage:

- tabelas/documentos/registros são *lugares*
- DBs tem CRUD

PLOP: PLace-Oriented Programming

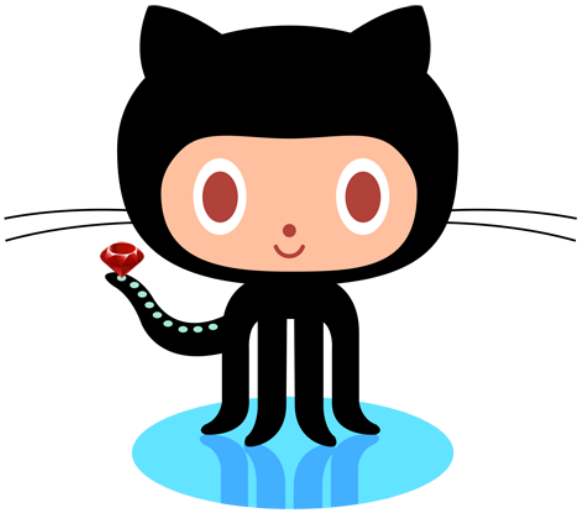
nova informação substitui antiga

T.I. do programador



Event Logs

T.I. do programador



Event Logs

A revanche dos usuários (a.k.a *big data*):

"Eu prefiro as suas ferramentas do que as que você me deu!"

Sistemas de informação

Informar

- comunicar conhecimento através de fatos

Sistemas de informação

Informar

- comunicar conhecimento através de fatos

Fatos são valores

- não mudam, *incorporam* tempo
- conhecimento é derivado de fatos

Sistemas de informação

Informar

- comunicar conhecimento através de fatos

Fatos são valores

- não mudam, *incorporam* tempo
- conhecimento é derivado de fatos

Informação é *simples*!

Sistemas de informação

Informar

- comunicar conhecimento através de fatos

Fatos são valores

- não mudam, *incorporam* tempo
- conhecimento é derivado de fatos

Informação é *simples*!

- não tem implementação

Sistemas de informação

Informar

- comunicar conhecimento através de fatos

Fatos são valores

- não mudam, *incorporam* tempo
- conhecimento é derivado de fatos

Informação é *simples*!

- não tem implementação
- não complique escondendo atráves de operações específicas por classe

O valor de valores

Mais *simples* que ‘place *oriented*’

- tanto no mesmo processo, entre processos e em storage

O valor de valores

Mais *simples* que ‘place *oriented*’

- tanto no mesmo processo, entre processos e em storage

Sistemas deveriam ser orientado a valores

- Mecanismo para percepção
- Mecanismo para memória
- Reduzem esforço de coordenação
- Essencial para tomar decisões

Complexidade

Conceito	Complica por que mistura
Estado	Tudo que tem contato

Complexidade

Conceito	Complica por que mistura
Estado	Tudo que tem contato
Objetos	Estado, identidade, valor, operações...

Complexidade

Conceito	Complica por que mistura
Estado	Tudo que tem contato
Objetos	Estado, identidade, valor, operações...
Métodos	Função e estado, namespaces

Complexidade

Conceito	Complica por que mistura
Estado	Tudo que tem contato
Objetos	Estado, identidade, valor, operações...
Métodos	Função e estado, namespaces
Variáveis	Valor, tempo

Complexidade

Conceito	Complica por que mistura
Estado	Tudo que tem contato
Objetos	Estado, identidade, valor, operações...
Métodos	Função e estado, namespaces
Variáveis	Valor, tempo
Sintaxe	Significado, ordem

Complexidade

Conceito	Complica por que mistura
Estado	Tudo que tem contato
Objetos	Estado, identidade, valor, operações...
Métodos	Função e estado, namespaces
Variáveis	Valor, tempo
Sintaxe	Significado, ordem
Herança	Tipos

Complexidade

Conceito	Complica por que mistura
Estado	Tudo que tem contato
Objetos	Estado, identidade, valor, operações...
Métodos	Função e estado, namespaces
Variáveis	Valor, tempo
Sintaxe	Significado, ordem
Herança	Tipos
Switch/matching	Múltiplos pares de ‘o que’ com ‘qual’

Complexidade

Conceito	Complica por que mistura
Estado	Tudo que tem contato
Objetos	Estado, identidade, valor, operações...
Métodos	Função e estado, namespaces
Variáveis	Valor, tempo
Sintaxe	Significado, ordem
Herança	Tipos
Switch/matching	Múltiplos pares de ‘o que’ com ‘qual’
Loop imperativo, fold	‘o que’ e ‘como’ (ordem)

Complexidade

Conceito	Complica por que mistura
Estado	Tudo que tem contato
Objetos	Estado, identidade, valor, operações...
Métodos	Função e estado, namespaces
Variáveis	Valor, tempo
Sintaxe	Significado, ordem
Herança	Tipos
Switch/matching	Múltiplos pares de ‘o que’ com ‘qual’
Loop imperativo, fold	‘o que’ e ‘como’ (ordem)
Atores	‘o que’ e ‘quem’

Complexidade

Conceito	Complica por que mistura
Estado	Tudo que tem contato
Objetos	Estado, identidade, valor, operações...
Métodos	Função e estado, namespaces
Variáveis	Valor, tempo
Sintaxe	Significado, ordem
Herança	Tipos
Switch/matching	Múltiplos pares de ‘o que’ com ‘qual’
Loop imperativo, fold	‘o que’ e ‘como’ (ordem)
Atores	‘o que’ e ‘quem’
Condicionais	‘o que/pq’ e ‘onde’

Complexidade

Conceito	Complica por que mistura
Estado	Tudo que tem contato
Objetos	Estado, identidade, valor, operações...
Métodos	Função e estado, namespaces
Variáveis	Valor, tempo
Sintaxe	Significado, ordem
Herança	Tipos
Switch/matching	Múltiplos pares de ‘o que’ com ‘qual’
Loop imperativo, fold	‘o que’ e ‘como’ (ordem)
Atores	‘o que’ e ‘quem’
Condicionais	onde/o que, controle de fluxo
ORM	OMG

Simplicidade

Conceito	Através de
Valores	final (imutável), coleções persistentes

Simplicidade

Conceito	Através de
Valores	final (imutável), coleções persistentes
Funções	a.k.a métodos sem estado

Simplicidade

Conceito	Através de
Valores	final (imutável), coleções persistentes
Funções	a.k.a métodos sem estado
Namespace	Depende da linguagem

Simplicidade

Conceito	Através de
Valores	final (imutável), coleções persistentes
Funções	a.k.a métodos sem estado
Namespace	Depende da linguagem
Managed refs	Clojure/Haskell refs

Simplicidade

Conceito	Através de
Valores	final (imutável), coleções persistentes
Funções	a.k.a métodos sem estado
Namespace	Depende da linguagem
Managed refs	Clojure/Haskell refs
Dados	Maps, arrays, conjuntos, json, etc

Simplicidade

Conceito	Através de
Valores	final (imutável), coleções persistentes
Funções	a.k.a métodos sem estado
Namespace	Depende da linguagem
Managed refs	Clojure/Haskell refs
Dados	Maps, arrays, conjuntos, json, etc
Poliformismo a la carte	Protocols, type classes

Simplicidade

Conceito	Através de
Valores	final (imutável), coleções persistentes
Funções	a.k.a métodos sem estado
Namespace	Depende da linguagem
Managed refs	Clojure/Haskell refs
Dados	Maps, arrays, conjuntos, json, etc
Poliformismo a la carte	Protocols, type classes
Funções de conjunto	Bibliotecas

Simplicidade

Conceito	Através de
Valores	final (imutável), coleções persistentes
Funções	a.k.a métodos sem estado
Namespace	Depende da linguagem
Managed refs	Clojure/Haskell refs
Dados	Maps, arrays, conjuntos, json, etc
Poliformismo a la carte	Protocols, type classes
Funções de conjunto	Bibliotecas
Filas	Bibliotecas

Simplicidade

Conceito	Através de
Valores	final (imutável), coleções persistentes
Funções	a.k.a métodos sem estado
Namespace	Depende da linguagem
Managed refs	Clojure/Haskell refs
Dados	Maps, arrays, conjuntos, json, etc
Poliformismo a la carte	Protocols, type classes
Funções de conjunto	Bibliotecas
Filas	Bibliotecas
Manipulação declarativa de dados	SQL, LINQ, Datalog

Simplicidade

Conceito	Através de
Valores	final (imutável), coleções persistentes
Funções	a.k.a métodos sem estado
Namespace	Depende da linguagem
Managed refs	Clojure/Haskell refs
Dados	Maps, arrays, conjuntos, json, etc
Poliformismo a la carte	Protocols, type classes
Funções de conjunto	Bibliotecas
Filas	Bibliotecas
Manipulação declarativa de dados	SQL, LINQ, Datalog
Regras	Bibliotecas, prolog

Simplicidade

Conceito	Através de
Valores	final (imutável), coleções persistentes
Funções	a.k.a métodos sem estado
Namespace	Depende da linguagem
Managed refs	Clojure/Haskell refs
Dados	Maps, arrays, conjuntos, json, etc
Poliformismo a la carte	Protocols, type classes
Funções de conjunto	Bibliotecas
Filas	Bibliotecas
Manipulação declarativa de dados	SQL, LINQ, Datalog
Regras	Bibliotecas, prolog
Consistência	Transações, valores

“I suppose I should learn Lisp, but it seems so foreign.”

Paul Graham, Nov 1983

(parênteses são *difíceis*)

(parênteses são *difíceis*)

Não é acessível para maioria

(parênteses são *difíceis*)

Não é acessível para maioria

Não é familiar

(parênteses são *difíceis*)

Não é acessível para maioria

Não é familiar

Mas são *simples*?

- *Não* em CL/Scheme
- Overload para chamadas e agrupamentos

(parênteses são *difíceis*)

Não é acessível para maioria

Não é familiar

Mas são *simples*?

- *Não* em CL/Scheme
- Overload para chamadas e agrupamentos

Adicionar novos primitivos é uma forma de simplificar

Linguagem é um instrumento para raciocínio humano, e não meramente um meio para expressão de pensamento”

George Boole

7 principais características

7 principais características

valores *por default*

7 principais características

valores *por default*

funcional *impura*

7 principais características

valores *por default*

funcional *impura*

lisp *reloaded*

7 principais características

valores *por default*

funcional *impura*

lisp *reloaded*

polimorfismo *a la carte*

7 principais características

valores *por default*

funcional *impura*

lisp *reloaded*

polimorfismo *a la carte*

interop *direto* com plataforma

7 principais características

valores *por default*

funcional *impura*

lisp *reloaded*

polimorfismo *a la carte*

interop *direto* com plataforma

STM gestão de estado automática

7 principais características

valores *por default*

funcional *impura*

lisp *reloaded*

polimorfismo *a la carte*

interop *direto* com plataforma

STM gestão de estado automática

open source

variantes

@github
clojure/clojure
clojure/clojurescript
clojure/clojure-clr
halgari/clojure-py

dados literais

tipo	exemplo
lista	`(1 2 3)`
vetor	`[1 2 3]`
mapa	`{:a 100` `:b 90}`
conjunto	`#{:a :b}`

tipo de dados atômicos

tipo	exemplo	equivalente em python
string	“python”	string
caracter	\f	char
integer a.p.	42	long/bignum
double	3.14159	double
double a.p.	3.14159M	dougle/Decimal(“)
booleano	true, false	boolean
nulo	nil	None
símbolo	foo, +	n/a
keyword	:foo, :ns/food	n/a
regex	#“py*”	n/a, import re
fração	22/7	Fraction(‘22/7')

chamando uma função

```
(inc 1)
```

= 2

funções são valores

```
(map inc [1 2 3])
```

```
= (2 3 4)
```

compondo funções

```
(def inc++ (comp inc inc))

(inc++ 1)
```

```
= 3
```

definindo símbolos

```
(def π 3.14159)

(def conf {:id “pybr8”})
```

definindo símbolos

```
(def π 3.14159)

(def conf {:id “pybr8”})

(get conf :id)

= “pybr8”
```

estruturas de dados persistentes

```
(def conf {:id “pybr8”})
```

estruturas de dados persistentes

```
(def conf {:id “pybr8”})

(assoc conf :n 42)

= {:id “pybr8” :n 42}
```

mapas são funções

```
(conf :id)
```

```
= “pybr8”
```

keywords são funções

```
(:id conf)
```

```
= “pybr8”
```

definindo uma função

```
(def say
 (fn [n] (str “hi ” n)))

(say “pybr8”)

= “hi pybr8”
```

definindo uma função

```
(def say
 (fn [n] (str “hi ” n)))
```

```
(defn say [name]
    (str “hi ” name))
```

read-eval

```
(eval (read-string “(inc 1)”))
```

= 2

read-eval

```
(eval (read-string "(inc 1)"))
```

reader retorna a lista `(inc 1)`

= 2

code = data

code = data

```
(count '(map inc [1 2]))
```

```
= 3
```

exemplo de macro

```
(when x
  (println “ok”))
```

```
(defmacro when
[test & body]
`(if ~test
   (do ~@body)))
```

exemplo de macro

```
(defmacro when
[test & body]
`(if ~test
   (do ~@body)))
```

```
(when x
  (println “ok”))
```


```
(if x
  (do (println “ok”)))
```

exemplo de macro

```
(defmacro when
[test & body]
`(if ~test
   (do ~@body)))
```

quote

```
(when x
  (println “ok”))
```

macroexpand

```
(if x
  (do (println “ok”)))
```

exemplo de macro

```
(defmacro when
[test & body]
`(if ~test
   (do ~@body)))
```

quote

unquote

```
(when x
  (println “ok”))
```

macroexpand

```
(if x
  (do (println “ok”)))
```

java interop

```
(.length “java”)
= 4
```

java interop

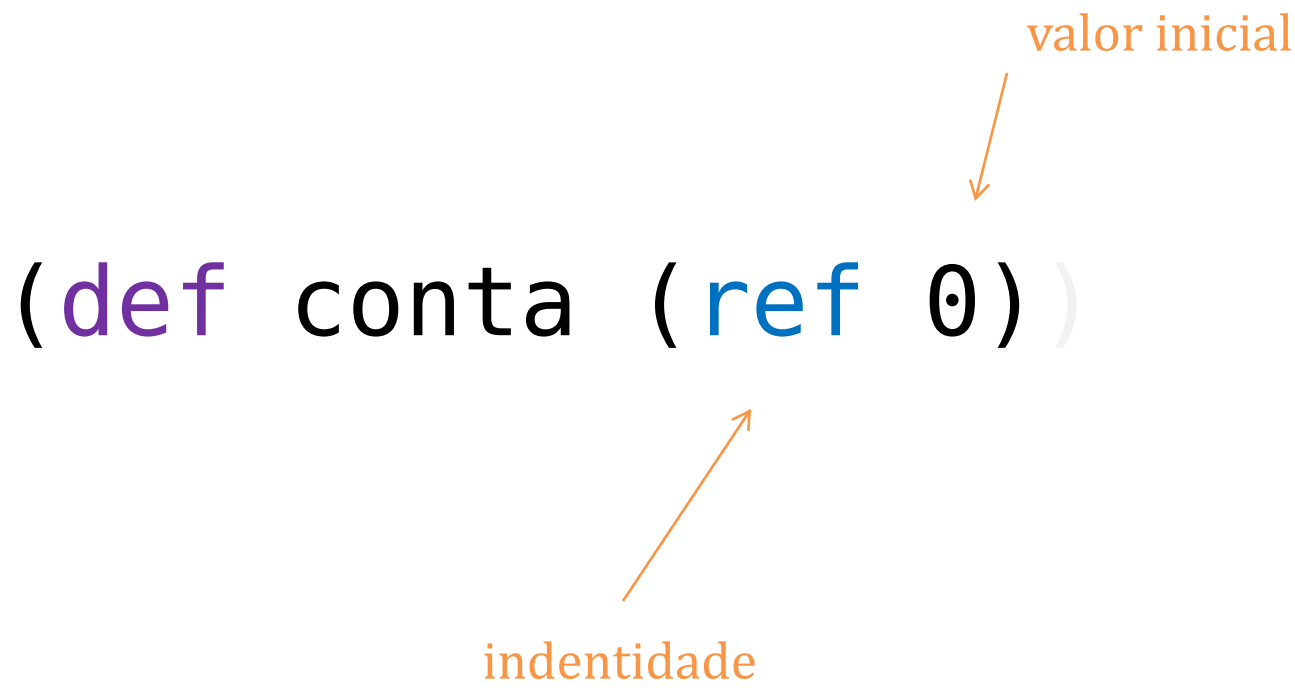
```
(.length “java”)

= 4


(.. “java”
    (substring 0 2)
    (toUpperCase))

 = “JA”
```

ref & stm



lendo um valor

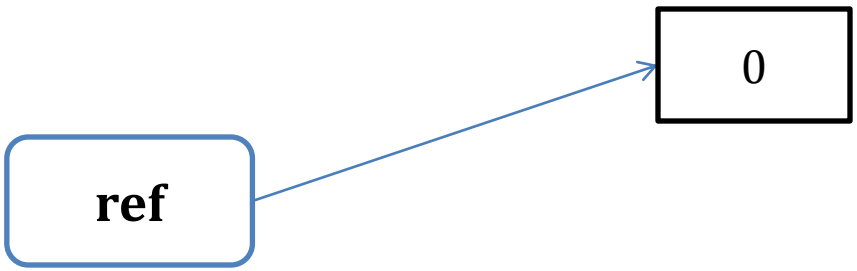
```
(deref conta)

@conta
```

```
= 0
```

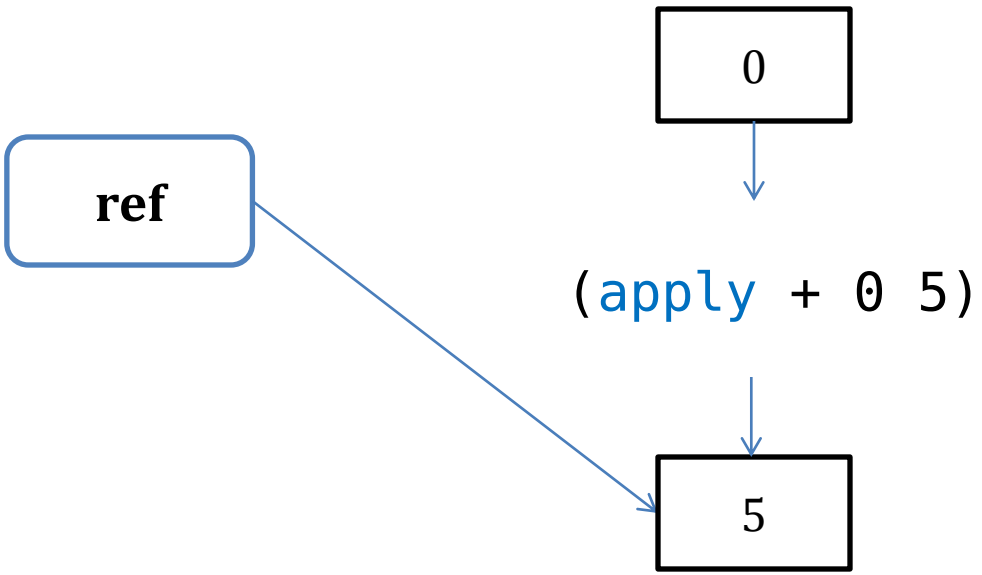
alter

```
(alter conta + 5)
```



alter

```
(alter conta + 5)
```

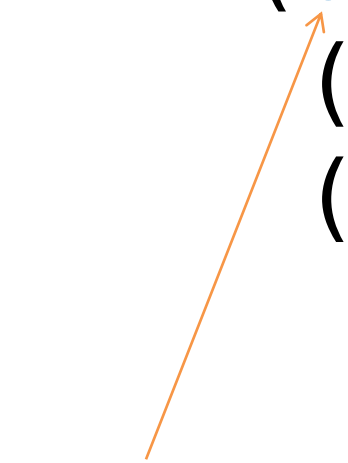


transação em memória

```
(dosync
 (alter conta + 5)
 (alter conta2 - 5))
```

transação em memória

```
(dosync
 (alter conta + 5)
 (alter conta2 - 5))
```



metadata

```
(def x (with-meta
         {:conf “pybr8”}
         {:cool true}))
```

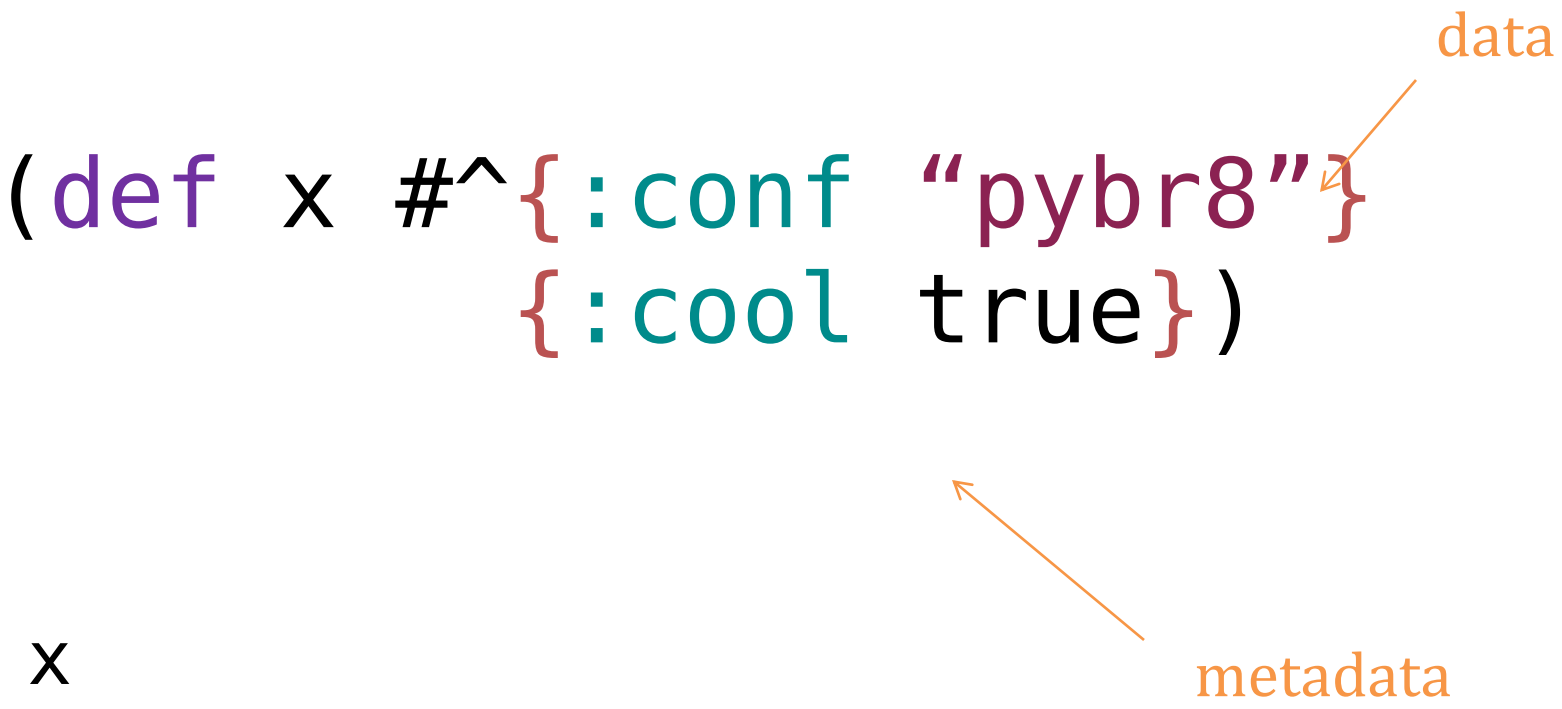
data

metadata

```
x
= {:conf “pybr8”}

(meta x)
= {:cool true}
```

metadata



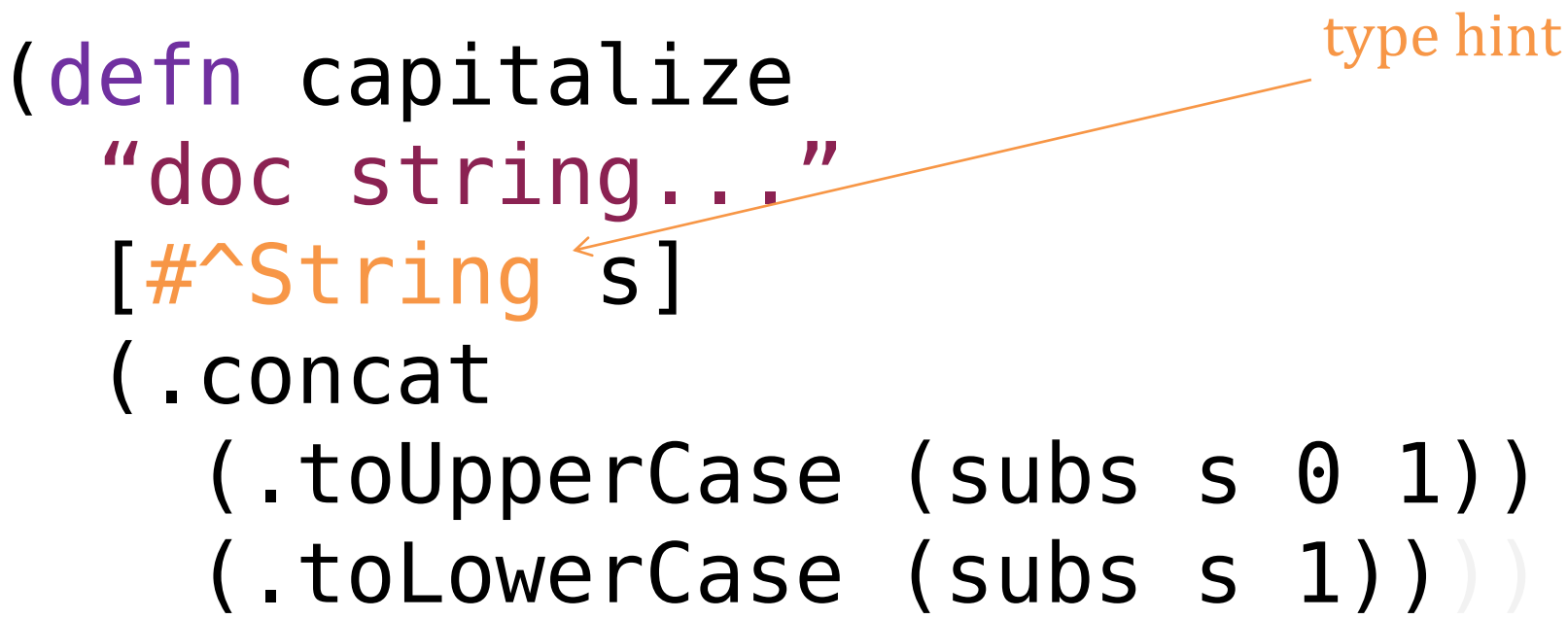

```
x
= {:conf “pybr8”}

^x
= {:cool true}
```

metadata

```
(defn capitalize
  “doc string...”
  [#^String s]
  (.concat
    (.toUpperCase (subs s 0 1))
    (.toLowerCase (subs s 1))))
```

metadata


type hint
(defn capitalize
"doc string..."
[#^String s]
(.concat
(.toUpperCase (subs s 0 1))
(.toLowerCase (subs s 1))))

metadata

```
(defn capitalize
  “doc string...”
  [#^String s]
  (.concat
    (.toUpperCase (subs s 0 1))
    (.toLowerCase (subs s 1))))
```

type hint

propagado

protocols

```
(defprotocol P
  (foo [x])
  (bar-me [x y]))
```

protocols

```
(defprotocol P
  (foo [x])
  (bar-me [x y]))


(extend-type MyType
 P
  (foo [x] (dec x) )
  (bar-me [x y] ... ))
```

reducers

```
(reduce + [1 2 3 4])
```

```
(require '[clojure.core.reducers :as r])
(r/fold + [1 2 3 4])
```

paralelo, usa ForkJoin framework

“[Clojure] is worth learning for the profound enlightenment experience you will have when you finally get it; that experience will make you a better programmer for the rest of your days, even if you never actually use [Clojure] itself a lot."

- Eric Raymond, "How to Become a Hacker"

referências

Talks:

Rich Hickey @ InfoQ

http://www.infoq.com/author/Rich-Hickey

Stuart Halloway @ Confreaks

http://www.confreaks.com/videos/191-rubyconf2009-clojure-for-ruby-programmers

Imagens:

http://www.benjamin-erb.de/slides/da-talk1/index.html

http://narkisr.github.com/clojure-concurrency/file/clojure/value-identity-state-and-time.svg

http://cemerick.com/2012/08/06/results-of-the-2012-state-of-clojure-survey/

pedro@**intelie.com.br**

twitter.com/pedroteixeira

https://github.com/intelie/quiz

trabalhe@intelie.com.br