

ThoughtWorks®

Teoria a serviço da prática

ALÉM DOS PARADIGMAS

*Entenda as características da linguagem
e saiba escolher e adaptar os padrões*

Luciano Ramalho
@ramalhoorg

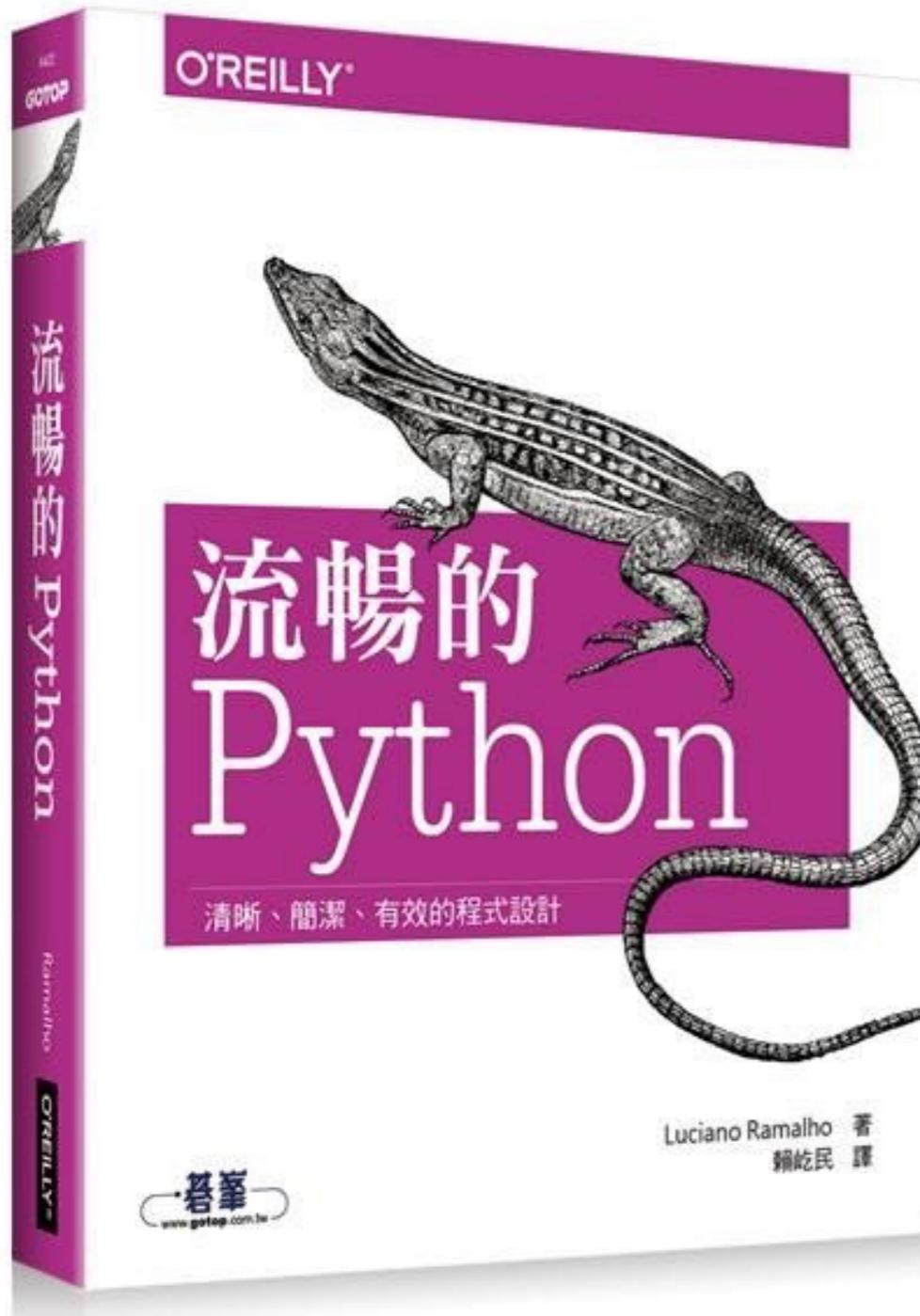
ThoughtWorks®

LUCIANO RAMALHO

Technical Principal

@ramalhoorg
luciano.ramalho@thoughtworks.com

FLUENT PYTHON, MEU 1º LIVRO



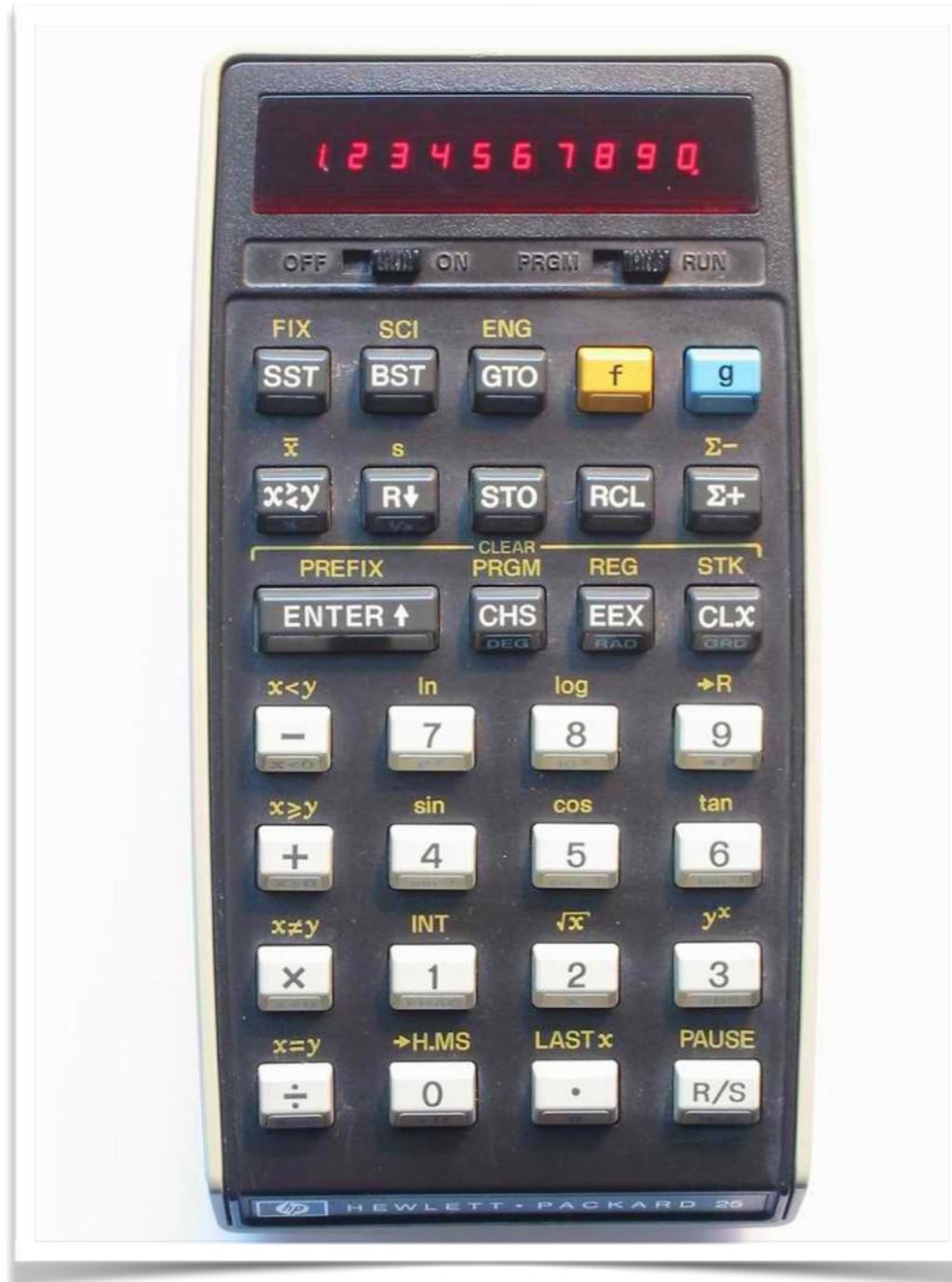
4.7 stars at
Amazon.com

Fluent Python (O'Reilly, 2015)
Python Fluente (Novatec, 2015)
Python к вершинам мастерства (DMK, 2015)
流暢的 Python (Gotop, 2016)
also in **Polish, Korean, etc...**

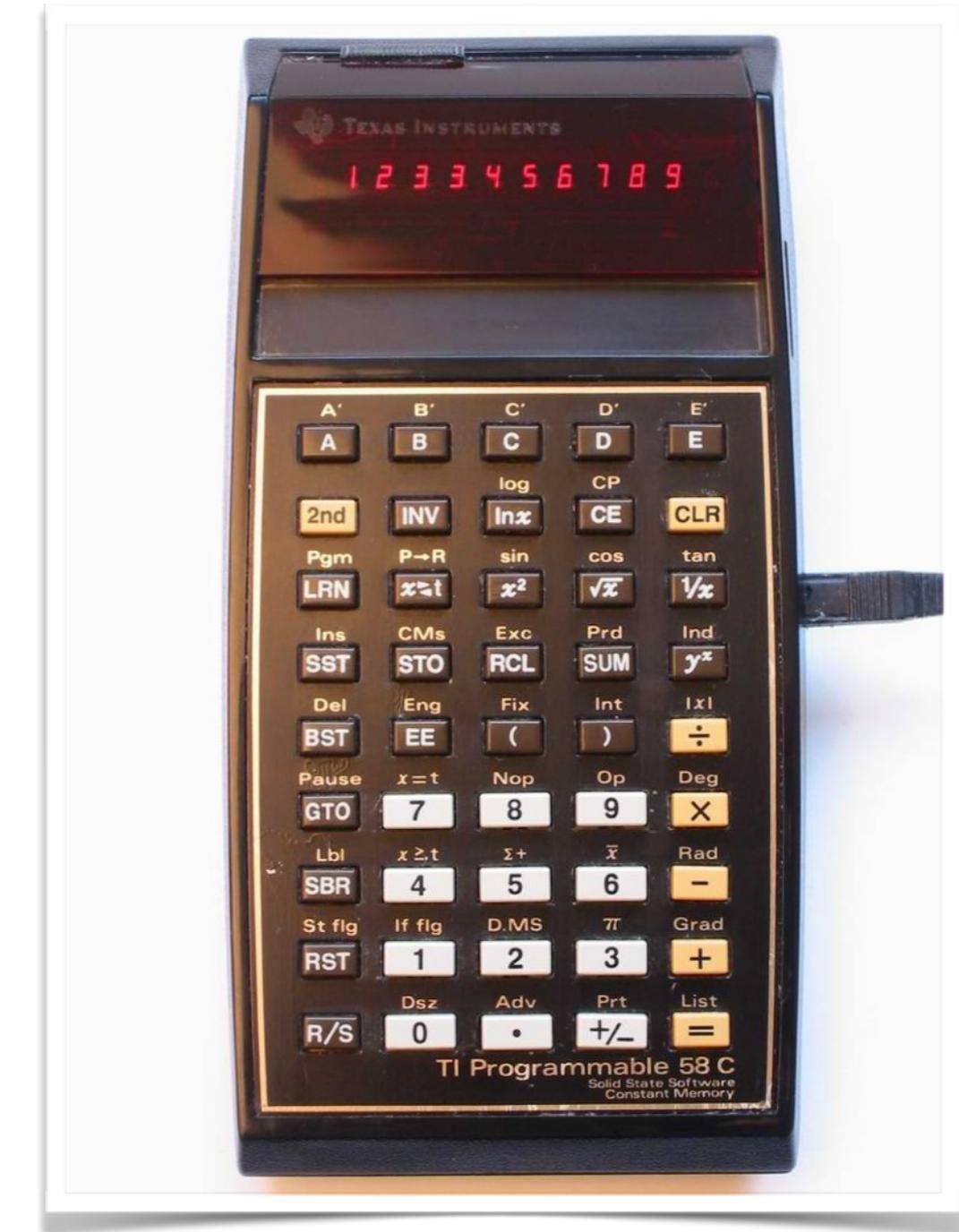
PARADIGMAS

Categorias de linguagens de programação

O PRIMEIRO "PARADIGMA" QUE APRENDI: IMPERATIVO



HP-25

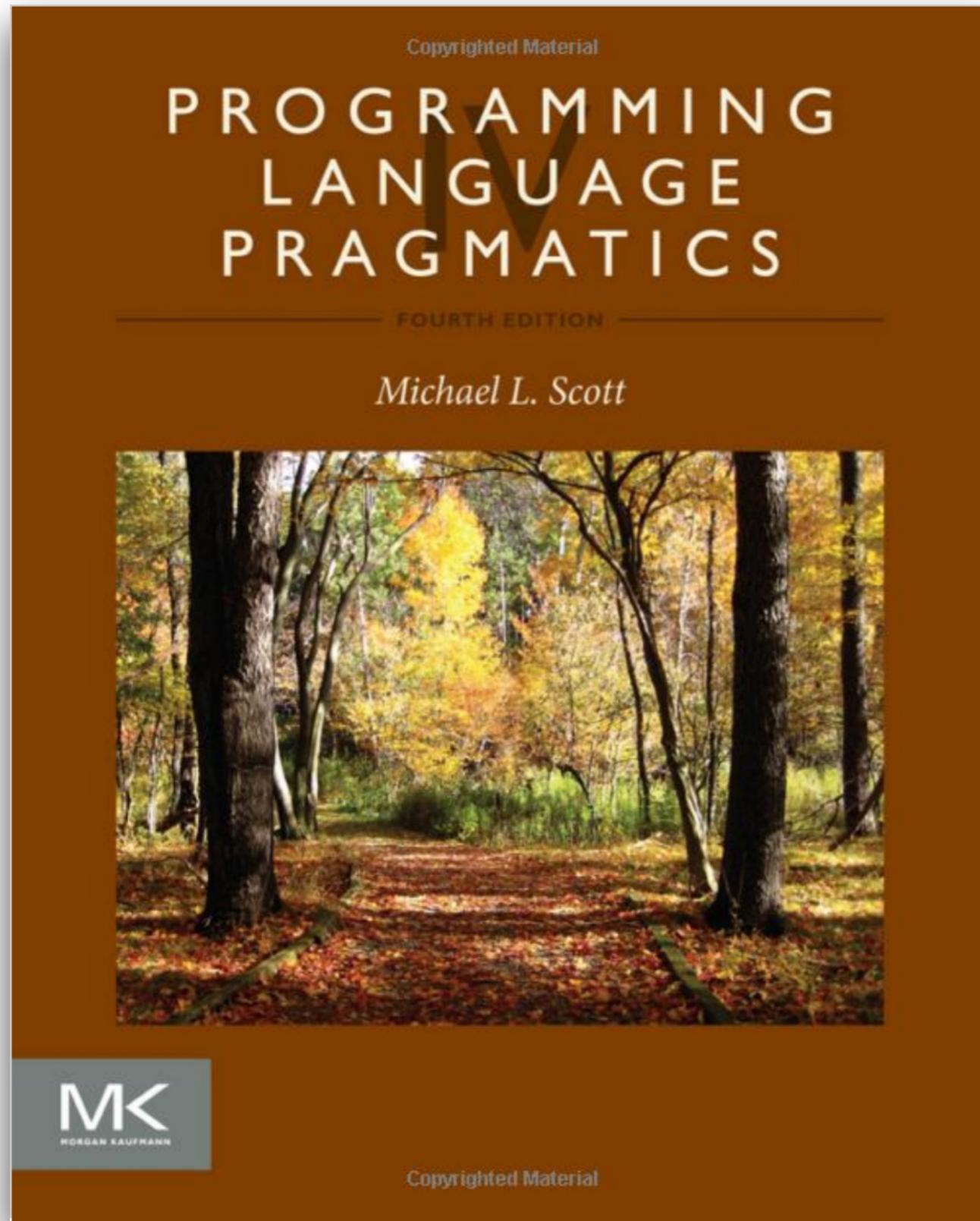


TI 58C

PROGRAMA PARA CALCULADORA HP-25

Linha	Visor Código	Intro- dução	X	Y	Z	T	
00							
01	14 11 04	f FIX 4					Apres
02	24 00	RCL 0	X				Apres
03	33	EEX	1.	00	X		
04	04	4	1.	04	X		
05	71	÷	X/10 ⁴				Divid
06	24 01	RCL 1	V		X/10 ⁴		
07	15 41	g x<0	V		X/10 ⁴		V é n
08	13 11	GTO 11	V		X/10 ⁴		Sim, t
09	51	+	V + X/10 ⁴				Não, t
10	13 13	GTO 13	V + X/10 ⁴				
11	21	x↔y	X/10 ⁴	V			V < 0
12	41	-	V - X/10 ⁴				
13	74	R/S	V.X				V.X =
14	24 02	RCL 2	F	B			Quein
15	14 41	f x<y	F	B			Comb
16	13 34	GTO 34	F	B			Sim, c
17	22	R↓	B			F	Não, c
18	23 41 02	STO - 2	B			F	Subtra
19	05	5	5	B			Gravi

PANORAMA SOBRE LINGUAGENS DE PROGRAMAÇÃO



Programming
Language
Pragmatics,
4th edition (2015)
Michael L. Scott

MDC: ASM X86

Máximo divisor
comum em
Assembly x86
(Scott, 2015)

```
pushl %ebp          # \
movl %esp, %ebp    # ) reserve space for local variables
subl $16, %esp     # /
call getInt         # read
movl %eax, -8(%ebp) # store i
call getInt         # read
movl %eax, -12(%ebp) # store j
A: movl -8(%ebp), %edi # load i
   movl -12(%ebp), %ebx # load j
   cmpl %ebx, %edi     # compare
   je D                # jump if i == j
   movl -8(%ebp), %edi # load i
   movl -12(%ebp), %ebx # load j
   cmpl %ebx, %edi     # compare
   jle B               # jump if i < j
   movl -8(%ebp), %edi # load i
   movl -12(%ebp), %ebx # load j
   subl %ebx, %edi     # i = i - j
   movl %edi, -8(%ebp) # store i
   jmp C
B: movl -12(%ebp), %edi # load j
   movl -8(%ebp), %ebx # load i
   subl %ebx, %edi     # j = j - i
   movl %edi, -12(%ebp) # store j
C: jmp A
D: movl -8(%ebp), %ebx # load i
   push %ebx           # push i (pass to putint)
   call putInt         # write
   addl $4, %esp        # pop i
   leave               # deallocate space for local variables
   mov $0, %eax         # exit status for program
   ret                 # return to operating system
```

Figure 1.7 Naive x86 assembly language for the GCD program.

MDC EM C, OCAML, PROLOG

```
int gcd(int a, int b) {                                // C
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}

let rec gcd a b =                                     (* OCaml *)
    if a = b then a
    else if a > b then gcd b (a - b)
    else gcd a (b - a)

gcd(A,B,G) :- A = B, G = A.                         % Prolog
gcd(A,B,G) :- A > B, C is A-B, gcd(C,B,G).
gcd(A,B,G) :- B > A, C is B-A, gcd(C,A,G).
```

Figure I.2 The GCD algorithm in C (top), OCaml (middle), and Prolog (bottom). All three versions assume (without checking) that their inputs are positive integers.

```
def gcd(a, b):  
    assert a > 0 and b > 0  
    while a != b:  
        if a > b:  
            a -= b  
        else:  
            b -= a  
    return a
```

Estilo imperativo

```
def gcd(a, b):  
    assert a > 0 and b > 0  
    if a == b:  
        return a  
    elif a > b:  
        return gcd(b, a - b)  
    else:  
        return gcd(a, b - a)
```

Estilo funcional

MDC EM PYTHON

```
def gcd(a, b):  
    assert a > 0 and b > 0  
    while a != b:  
        if a > b:  
            a -= b  
        else:  
            b -= a  
    return a
```

Estilo imperativo

~~def gcd(a, b):
 assert a > 0 and b > 0
 if a == b:
 return a
 elif a > b:
 return gcd(b, a - b)
 else:
 return gcd(a, b - a)~~

Estilo funcional

Inadequado para
Python que não faz
otimização de
chamada de cauda
(TCO)

UMA CLASSIFICAÇÃO

1.2 The Programming Language Spectrum

Example 1.3

Classification of programming languages

The many existing languages can be classified into families based on their model of computation. [Figure 1.1](#) shows a common set of families. The top-level division distinguishes between the *declarative* languages, in which the focus is on *what* the computer is to do, and the *imperative* languages, in which the focus is on *how* the computer should do it.■

declarative	
functional	Lisp/Scheme, ML, Haskell
dataflow	Id, Val
logic, constraint-based	Prolog, spreadsheets, SQL
imperative	
von Neumann	C, Ada, Fortran, ...
object-oriented	Smalltalk, Eiffel, Java, ...
scripting	Perl, Python, PHP, ...

FIGURE 1.1 Classification of programming

languages. Note that the categories are fuzzy and open to debate. In particular, it is possible for a functional language to be object-oriented, and many authors do not consider functional programming to be declarative.

Programming
Language
Pragmatics,
4th edition (2015)
Michael L. Scott

UMA CLASSIFICAÇÃO (ZOOM)

declarative	
functional	Lisp/Scheme, ML, Haskell
dataflow	Id, Val
logic, constraint-based	Prolog, spreadsheets, SQL
imperative	
von Neumann	C, Ada, Fortran, ...
object-oriented	Smalltalk, Eiffel, Java, ...
scripting	Perl, Python, PHP, ...

FIGURE 1.1 Classification of programming

languages. Note that the categories are fuzzy and open to debate. In particular, it is possible for a functional language to be object-oriented, and many authors do not consider functional programming to be declarative.

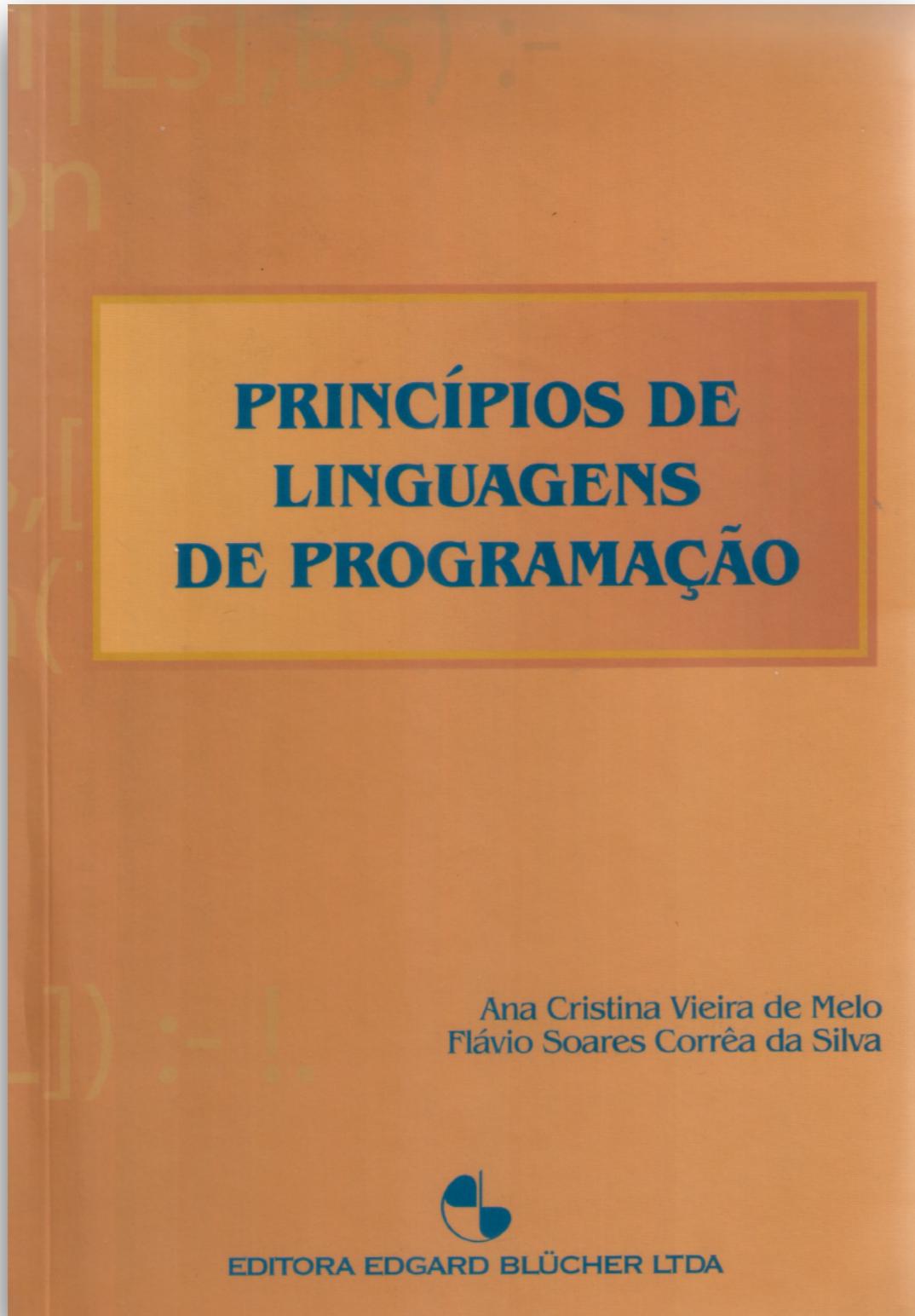
UMA CLASSIFICAÇÃO (ZOOM)

declarative	
functional	Lisp/Scheme, ML, Haskell
dataflow	Id, Val
logic, constraint-based	Prolog, spreadsheets, SQL
imperative	
von Neumann	C, Ada, Fortran, ...
object-oriented	Smalltalk, Eiffel, Java, ...
scripting	Perl, Python, PHP, ...
	???

FIGURE 1.1 Classification of programming

languages. Note that the categories are fuzzy and open to debate. In particular, it is possible for a functional language to be object-oriented, and many authors do not consider functional programming to be declarative.

OUTRO LIVRO, OUTRA CLASSIFICAÇÃO



Princípios de Linguagens de
Programação
(2003)

Ana Cristina Vieira de Melo
Flávio Soares Corrêa da Silva

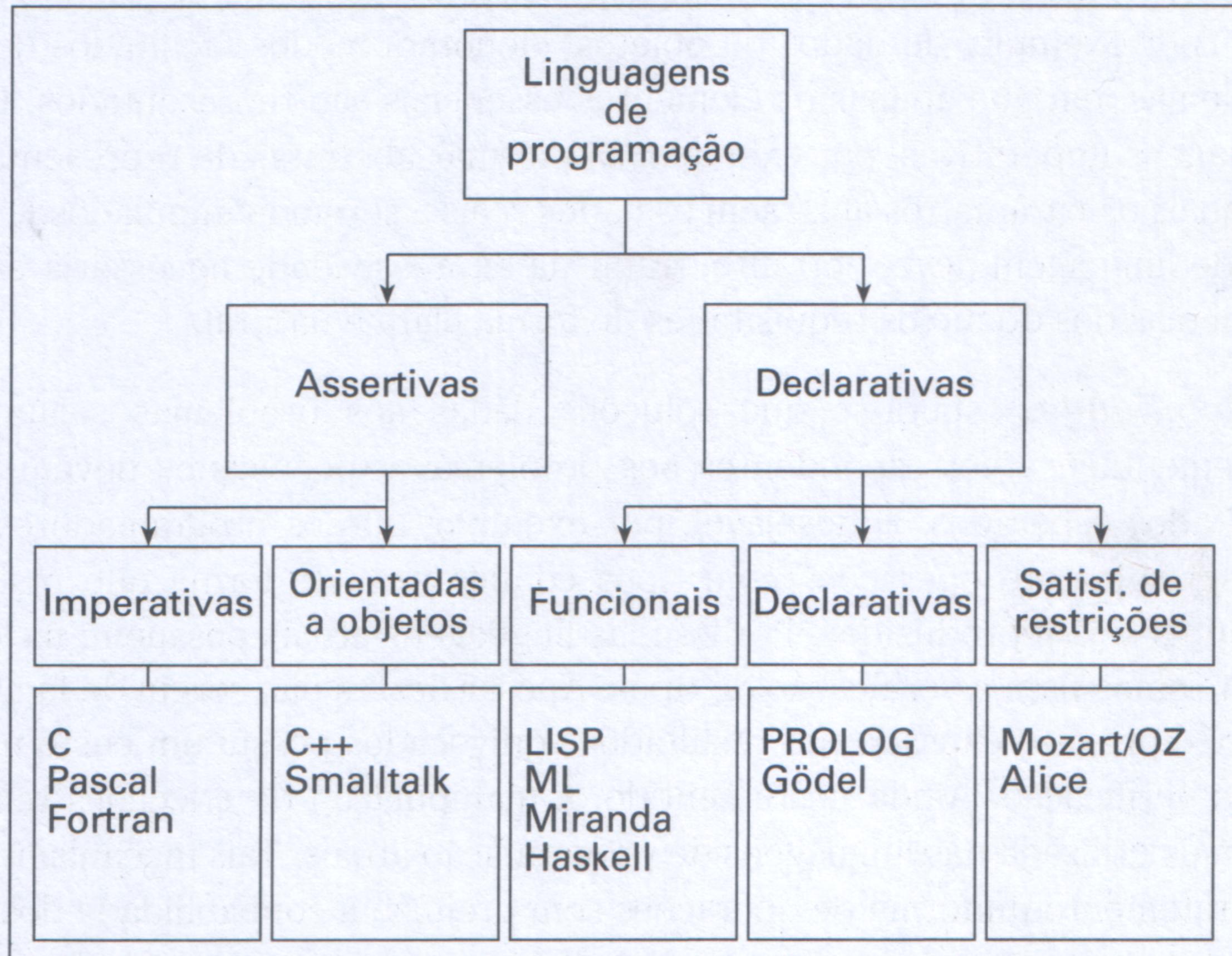


Figura 1 — Tipologia de linguagens de programação.

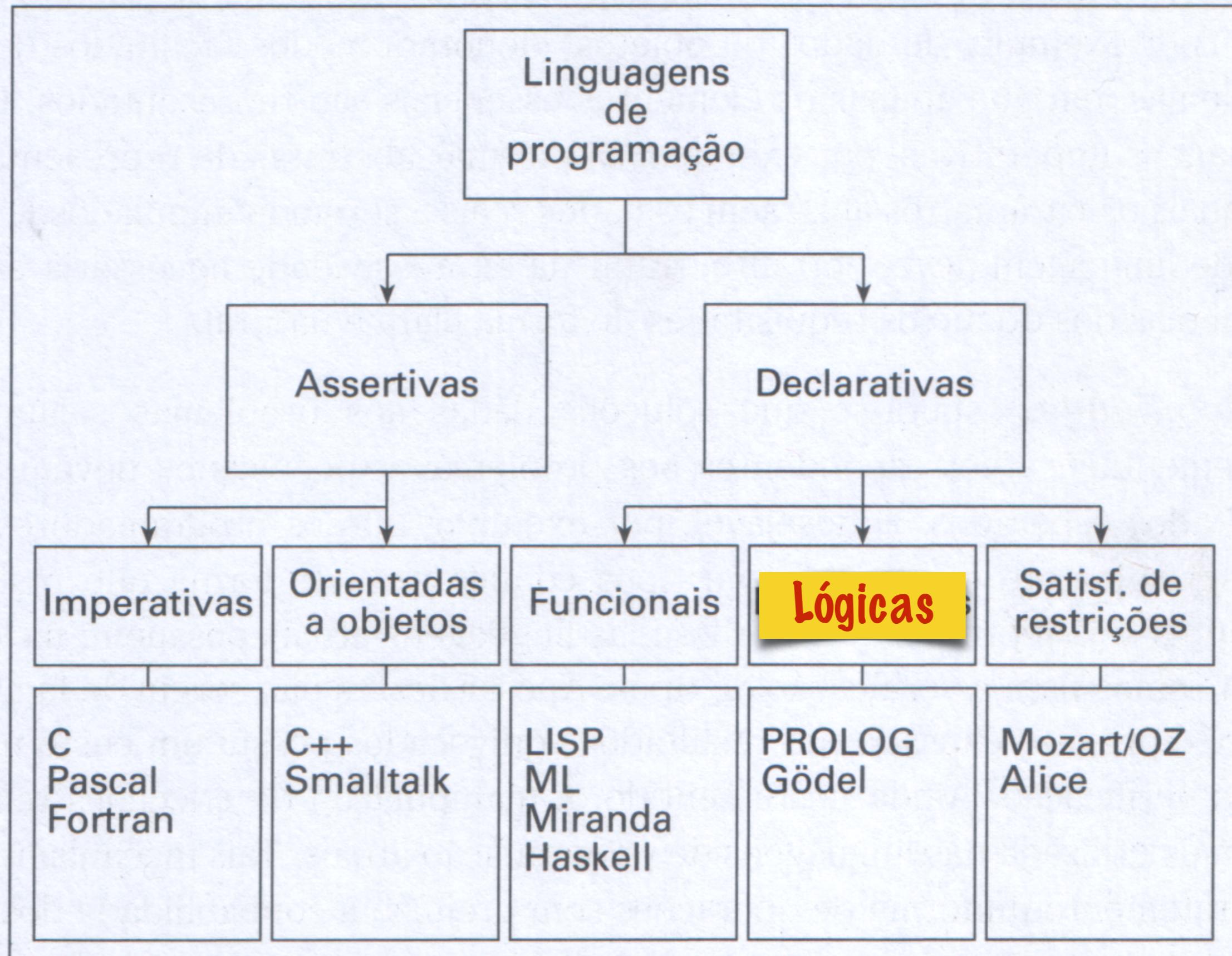


Figura 1 — Tipologia de linguagens de programação.

SITE: THE LANGUAGE LIST



The Language List

Collected Information On About 2500 Computer Languages, Past and Present.

Maintained by [Bill Kinnersley](#)

Welcome to The Language List! Early versions of this list were posted to comp.lang.misc beginning in 1991. Now a web site, our intention remains the same -- to become one of the most complete sources of information on computer programming languages ever assembled (or compiled :-).

The list does not pretend to be a definitive scholarly work. Its purpose is to collect and provide timely information in a rapidly growing field. Its accuracy and completeness depend to a great extent on the users of the Internet. If you know about a language that should be added, please share your knowledge.

[Start a Search](#)

[Contents of an Entry](#)

[What Languages Should be Included](#)

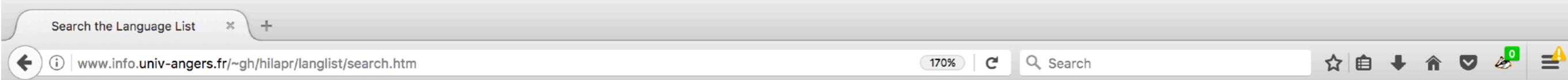
[Language Categories](#)

[Dialects, Variants, Versions and Implementations](#)

[References](#)

[A Chronology of Influential Languages](#)

SITE QUEBRADO, LINGUAGENS FALTANDO...



Search for a particular language entry:

<#> [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

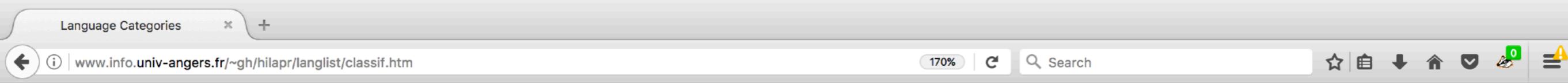
[Home](#)

Not Found

The requested URL
/~nkinners/LangList
/Indexes/gindex.htm
was not found on
this server.

*Apache/2.2.15 (Red
Hat) Server at
people.ku.edu Port
80*

CATEGORIAS DE LINGUAGENS



Language Categories

Procedural Language

A language which states how to compute the result of a given problem. This term encompasses both imperative and functional languages.

Imperative Language

A language which operates by a sequence of commands that change the value of data elements. Imperative languages are typified by assignments and iteration.

Declarative Language

A language which operates by making descriptive statements about data, and relations between data. The algorithm is hidden in the semantics of the language. This category encompasses both applicative and logic languages. Examples of declarative features are set comprehensions and pattern-matching statements.

Applicative Language

A language that operates by application of functions to values, with no side effects. A functional language in the broad sense.

Functional Language

In the narrow sense, a functional language is one that operates by use of higher-order functions, building operators that manipulate functions directly without ever appearing to manipulate data. Example: FP.

CATEGORIAS DE LINGUAGENS (2)

Language Categories 170% C ⌂ 0

www.info.univ-angers.fr/~gh/hilapr/langlist/classif.htm Search

Star Home Mail Notifications

Constraint Language

A language in which a problem is specified and solved by a series of constraining relationships.

Object-Oriented Language

A language in which data and the functions which access it are treated as a unit.

Concurrent Language

A concurrent language describes programs that may be executed in parallel. This may be either

- Multiprogramming: sharing one processor
 - Multiprocessing: separate processors sharing one memory
 - Distributed

Concurrent languages differ in the way that processes are created:

- Coroutines - control is explicitly transferred - examples are Simula I, SL5, BLISS and Modula-2.
 - Fork/join - examples are PL/I and Mesa.
 - Cobegin/coend - examples are ALGOL 68, CSP, Edison, Argus.
 - Process declarations - examples are DP, SR, Concurrent Pascal, Modula, PLITS and Ada.

and the ways in which processes interact:

- Semaphores - ALGOL 68
 - Conditional critical regions - Edison, DP, Argus
 - Monitors - Concurrent Pascal, Modula
 - Message passing - CSP, PLITS, Gypsy, Actors
 - Remote procedure calls - DP, *Mod
 - Rendezvous - Ada, SR

CATEGORIAS DE LINGUAGENS (3)

The screenshot shows a web browser window titled "Language Categories". The address bar contains the URL "www.info.univ-angers.fr/~gh/hilapr/langlist/classif.htm". The page content is a list of language categories:

- Message passing - CSP, PLTTS, Gypsy, Actors
- Remote procedure calls - DP, *Mod
 - Rendezvous - Ada, SR
 - Atomic transactions - Argus

Fourth Generation Language (4GL)

A very high-level language. It may use natural English or visual constructs. Algorithms or data structures may be selected by the compiler.

Query Language

An interface to a database.

Specification Language

A formalism for expressing a hardware or software design.

Assembly Language

A symbolic representation of the machine language of a specific computer.

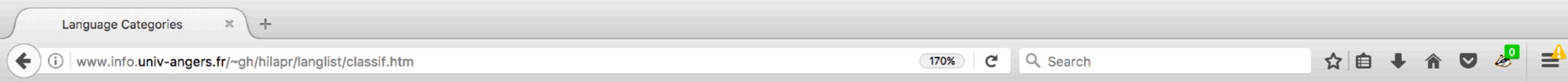
Intermediate Language

A language used as an intermediate stage in compilation. May be either text or binary.

Metalanguage

A language used for the formal description of another language.

CATEGORIAS DE LINGUAGENS (4)



Definitional Language

An applicative language containing assignments interpreted as definitions. Example: Lucid.

Single Assignment Language

An applicative language using assignments, with the convention that a variable may appear on the left side of an assignment only once within the portion of the program in which it is active.

Dataflow Language

A language suitable for use on a dataflow architecture. Necessary properties include freedom from side effects, and the equivalence of scheduling constraints with data dependencies. Examples: Val, Id, SISAL, Lucid.

Logic Language

A logic language deals with predicates or relationships $p(X,Y)$. A program consists of a set of Horn clauses which may be:

- facts - $p(X,Y)$ is true
- rules - p is true if q_1 and q_2 and ... q_n are true
- queries - is g_1 and g_2 and ... g_n true? (g_i 's are the goals.)

Further clauses are inferred using resolution. One clause is selected containing p as an assumption, another containing p as a consequence, and p is eliminated between them. If the two p 's have different arguments they must be unified, using the substitution with the fewest constraints that makes them the same. Logic languages try alternative resolutions for each goal in succession, backtracking in a search for a common solution.

- OR-parallel logic languages try alternative resolutions in parallel
- AND-parallel logic languages try to satisfy several goals in parallel.

Constraint Language

ThoughtWorks®

CATEGORIAS?

Ontologia é tão anos 90...

UMA CLASSIFICAÇÃO BASEADA EM FATOS CIENTÍFICOS

Periodic Table of the Elements

The Periodic Table is organized into groups (families) based on similar chemical properties:

- Alkali Metals:** Hydrogen (H), Lithium (Li), Sodium (Na), Potassium (K), Rubidium (Rb), Francium (Fr).
- Alkaline Earth Metals:** Beryllium (Be), Magnesium (Mg), Calcium (Ca), Strontium (Sr), Barium (Ba).
- Transition Metals:** Scandium (Sc), Titanium (Ti), Vanadium (V), Chromium (Cr), Manganese (Mn), Iron (Fe), Cobalt (Co), Nickel (Ni), Copper (Cu), Zinc (Zn), Rhodium (Rh), Palladium (Pd), Silver (Ag), Cadmium (Cd), Indium (In), Tin (Sn), Antimony (Sb), Tellurium (Te), Iodine (I), Xenon (Xe).
- Basic Metals:** Lanthanide Series (Lanthanum (La) to Lu), Actinide Series (Actinium (Ac) to Lr).
- Semimetal:** Boron (B), Silicon (Si), Germanium (Ge), Arsenic (As), Selenium (Se), Antimony (Sb), Tellurium (Te), Iodine (I).
- Nonmetal:** Carbon (C), Nitrogen (N), Oxygen (O), Fluorine (F), Neon (Ne), Phosphorus (P), Sulfur (S), Chlorine (Cl), Bromine (Br), Krypton (Kr), Xenon (Xe), Radon (Rn), Ununpentium (Uup), Livermorium (Lv), Ununseptium (Uus), Ununoctium (Uuo).
- Halogens:** Nitrogen (N), Oxygen (O), Fluorine (F), Chlorine (Cl), Bromine (Br), Iodine (I), Ununpentium (Uup), Livermorium (Lv), Ununseptium (Uus), Ununoctium (Uuo).
- Noble Gas:** Helium (He), Neon (Ne), Argon (Ar), Krypton (Kr), Xenon (Xe), Radon (Rn).
- Lanthanide:** Lanthanum (La), Cerium (Ce), Praseodymium (Pr), Neodymium (Nd), Promethium (Pm), Samarium (Sm), Europium (Eu), Gadolinium (Gd), Terbium (Tb), Dysprosium (Dy), Holmium (Ho), Erbium (Er), Thulium (Tm), Ytterbium (Yb), Lutetium (Lu).
- Actinide:** Actinium (Ac), Thorium (Th), Protactinium (Pa), Uranium (U), Neptunium (Np), Plutonium (Pu), Americium (Am), Curium (Cm), Berkelium (Bk), Californium (Cf), Einsteinium (Es), Fermium (Fm), Mendelevium (Md), Nobelium (No), Lawrencium (Lr).

Normal boiling points are in °C. SP = Triple Point. Pressure is listed if not 1 atm. Allotrope is listed if more than one allotrope.

		Atomic Number	Boiling Point	
				Symbol
				Name
				Atomic Mass
1	IA	1	-252.762	H
2	IIA	2	2471	Be
3	IIIIB	3	1342	Li
4	IVB	4	2471	Beryllium
11	VIB	12	882.940	Mg
19	VIIB	20	1090	Sodium
37	VIIIB	21	759	K
55	VIIIIB	22	40.078	Ca
87	VIIIB	23	44.956	Sc
	VIIIB	24	47.88	Ti
	VIIIB	25	50.942	V
	VIIIB	26	51.996	Cr
	VIIIB	27	58.933	Mn
	VIIIB	28	58.693	Fe
	VIIIB	29	54.938	Co
	VIIIB	30	58.933	Ni
	VIIIB	31	58.693	Cu
	VIIIB	32	54.938	Zn
	VIIIB	33	58.933	Ga
	VIIIB	34	58.693	Ge
	VIIIB	35	54.938	As
	VIIIB	36	58.933	Se
	VIIIB	37	58.693	Br
	VIIIB	38	54.938	Kr
	VIIIB	39	58.933	Xe
	VIIIB	40	58.693	Rb
	VIIIB	41	54.938	Sr
	VIIIB	42	58.933	Y
	VIIIB	43	58.693	Zr
	VIIIB	44	54.938	Nb
	VIIIB	45	58.933	Mo
	VIIIB	46	58.693	Tc
	VIIIB	47	54.938	Ru
	VIIIB	48	58.933	Rh
	VIIIB	49	58.693	Pd
	VIIIB	50	54.938	Ag
	VIIIB	51	58.933	Cd
	VIIIB	52	58.693	In
	VIIIB	53	54.938	Zn
	VIIIB	54	58.933	Ga
	VIIIB	55	58.693	Ge
	VIIIB	56	54.938	As
	VIIIB	57	58.933	Se
	VIIIB	58	58.693	Br
	VIIIB	59	54.938	Kr
	VIIIB	60	58.933	Xe
	VIIIB	61	58.693	Rb
	VIIIB	62	54.938	Sr
	VIIIB	63	58.933	Y
	VIIIB	64	58.693	Zr
	VIIIB	65	54.938	Nb
	VIIIB	66	58.933	Mo
	VIIIB	67	58.693	Tc
	VIIIB	68	54.938	Ru
	VIIIB	69	58.933	Rh
	VIIIB	70	58.693	Pd
	VIIIB	71	54.938	Ag
	VIIIB	72	58.933	Cd
	VIIIB	73	58.693	In
	VIIIB	74	54.938	Zn
	VIIIB	75	58.933	Ga
	VIIIB	76	58.693	Ge
	VIIIB	77	54.938	As
	VIIIB	78	58.933	Se
	VIIIB	79	58.693	Br
	VIIIB	80	54.938	Kr
	VIIIB	81	58.933	Xe
	VIIIB	82	58.693	Rb
	VIIIB	83	54.938	Y
	VIIIB	84	58.933	Zr
	VIIIB	85	58.693	Nb
	VIIIB	86	54.938	Mo
	VIIIB	87	58.933	Tc
	VIIIB	88	58.693	Ru
	VIIIB	89	54.938	Pd
	VIIIB	90	58.933	Ag
	VIIIB	91	58.693	Cd
	VIIIB	92	54.938	In
	VIIIB	93	58.933	Zn
	VIIIB	94	58.693	Ga
	VIIIB	95	54.938	Ge
	VIIIB	96	58.933	As
	VIIIB	97	58.693	Se
	VIIIB	98	54.938	Br
	VIIIB	99	58.933	Kr
	VIIIB	100	58.693	Xe
	VIIIB	101	54.938	Rb
	VIIIB	102	58.933	Y
	VIIIB	103	58.693	Zr
	VIIIB	104	54.938	Nb
	VIIIB	105	58.933	Mo
	VIIIB	106	58.693	Tc
	VIIIB	107	54.938	Ru
	VIIIB	108	58.933	Pd
	VIIIB	109	58.693	Ag
	VIIIB	110	54.938	Cd
	VIIIB	111	58.933	In
	VIIIB	112	58.693	Zn
	VIIIB	113	54.938	Ga
	VIIIB	114	58.933	Ge
	VIIIB	115	58.693	As
	VIIIB	116	54.938	Se
	VIIIB	117	58.933	Br
	VIIIB	118	58.693	Kr
	VIIIB	119	54.938	Xe
	VIIIB	120	58.933	Rb
	VIIIB	121	58.693	Y
	VIIIB	122	54.938	Zr
	VIIIB	123	58.933	Nb
	VIIIB	124	58.693	Mo
	VIIIB	125	54.938	Tc
	VIIIB	126	58.933	Ru
	VIIIB	127	58.693	Pd
	VIIIB	128	54.938	Ag
	VIIIB	129	58.933	Cd
	VIIIB	130	58.693	In
	VIIIB	131	54.938	Zn
	VIIIB	132	58.933	Ga
	VIIIB	133	58.693	Ge
	VIIIB	134	54.938	As
	VIIIB	135	58.933	Se
	VIIIB	136	58.693	Br
	VIIIB	137	54.938	Kr
	VIIIB	138	58.933	Xe
	VIIIB	139	58.693	Rb
	VIIIB	140	54.938	Y
	VIIIB	141	58.933	Zr
	VIIIB	142	58.693	Nb
	VIIIB	143	54.938	Mo
	VIIIB	144	58.933	Tc
	VIIIB	145	58.693	Ru
	VIIIB	146	54.938	Pd
	VIIIB	147	58.933	Ag
	VIIIB	148	58.693	Cd
	VIIIB	149	54.938	In
	VIIIB	150	58.933	Zn
	VIIIB	151	58.693	Ga
	VIIIB	152	54.938	Ge
	VIIIB	153	58.933	As
	VIIIB	154	58.693	Se
	VIIIB	155	54.938	Br
	VIIIB	156	58.933	Kr
	VIIIB	157	58.693	Xe
	VIIIB	158	54.938	Rb
	VIIIB	159	58.933	Y
	VIIIB	160	58.693	Zr
	VIIIB	161	54.938	Nb
	VIIIB	162	58.933	Mo
	VIIIB	163	58.693	Tc
	VIIIB	164	54.938	Ru
	VIIIB	165	58.933	Pd
	VIIIB	166	58.693	Ag
	VIIIB	167	54.938	Cd
	VIIIB	168	58.933	In
	VIIIB	169	58.693	Zn
	VIIIB	170	54.938	Ga
	VIIIB	171	58.933	Ge
	VIIIB	172	58.693	As
	VIIIB	173	54.938	Se
	VIIIB	174	58.933	Br
	VIIIB	175	58.693	Kr
	VIIIB	176	54.938	Xe
	VIIIB	177	58.933	Rb
	VIIIB	178	58.693	Y
	VIIIB	179	54.938	Zr
	VIIIB	180	58.933	Nb
	VIIIB	181	58.693	Mo
	VIIIB	182	54.938	Tc
	VIIIB	183	58.933	Ru
	VIIIB	184	58.693	Pd
	VIIIB	185	54.938	Ag
	VIIIB	186	58.933	Cd
	VIIIB	187	58.693	In
	VIIIB	188	54.938	Zn
	VIIIB	189	58.933	Ga
	VIIIB	190	58.693	Ge
	VIIIB	191	54.938	As
	VIIIB	192	58.933	Se
	VIIIB	193	58.693	Br
	VIIIB	194	54.938	Kr
	VIIIB	195	58.933	Xe
	VIIIB	196	58.693	Rb
	VIIIB	197	54.938	Y
	VIIIB	198	58.933	Zr
	VIIIB	199	58.693	Nb
	VIIIB	200	54.938	Mo
	VIIIB	201</		

UMA CLASSIFICAÇÃO BASEADA EM FATOS CIENTÍFICOS?

"Gases" "nobres"!?

The Periodic Table of the Elements is shown, highlighting the noble gas group (Group 18) in red. The noble gases are He, Ne, Ar, Kr, Xe, and Rn. The table includes atomic number, symbol, name, atomic mass, and boiling point.

	Atomic Number	Symbol	Name	Atomic Mass	Boiling Point
1	1	H	Hydrogen	1.008	-252.762
2	2	He	Helium	4.003	-268.93
3	3	Li	Lithium	6.941	-182.953
4	4	Be	Beryllium	9.012	-182.953
11	11	Na	Sodium	22.990	-182.953
12	12	Mg	Magnesium	24.305	-182.953
19	19	K	Potassium	39.098	-182.953
20	20	Ca	Calcium	40.078	-182.953
21	21	Sc	Scandium	44.956	-182.953
22	22	Ti	Titanium	47.88	-182.953
23	23	V	Vanadium	50.942	-182.953
24	24	Cr	Chromium	51.996	-182.953
25	25	Mn	Manganese	54.938	-182.953
26	26	Fe	Iron	55.933	-182.953
27	27	Co	Cobalt	58.933	-182.953
28	28	Ni	Nickel	58.693	-182.953
29	29	Cu	Copper	63.546	-182.953
30	30	Zn	Zinc	65.39	-182.953
31	31	Ga	Gallium	69.732	-182.953
32	32	Ge	Germanium	72.61	-182.953
33	33	As	Arsenic	74.922	-182.953
34	34	Se	Selenium	78.972	-182.953
35	35	Br	Bromine	79.904	-182.953
36	36	Kr	Krypton	84.80	-182.953
37	37	Rb	Rubidium	84.468	-182.953
38	38	Sr	Strontium	87.62	-182.953
39	39	Y	Yttrium	88.906	-182.953
40	40	Zr	Zirconium	91.224	-182.953
41	41	Nb	Niobium	92.906	-182.953
42	42	Mo	Molybdenum	95.95	-182.953
43	43	Tc	Technetium	98.907	-182.953
44	44	Ru	Ruthenium	101.07	-182.953
45	45	Rh	Rhodium	102.906	-182.953
46	46	Pd	Palladium	106.42	-182.953
47	47	Ag	Silver	107.868	-182.953
48	48	Cd	Cadmium	112.411	-182.953
49	49	In	Indium	114.818	-182.953
50	50	Sn	Tin	118.71	-182.953
51	51	Sb	Antimony	121.760	-182.953
52	52	Te	Tellurium	127.6	-182.953
53	53	I	Iodine	126.904	-182.953
54	54	Xe	Xenon	131.29	-182.953
55	55	Cs	Cesium	132.905	-182.953
56	56	Ba	Barium	137.327	-182.953
57	57	Hf	Hafnium	178.49	-182.953
58	58	Ta	Tantalum	180.948	-182.953
59	59	W	Tungsten	183.85	-182.953
60	60	Re	Rhenium	186.207	-182.953
61	61	Os	Osmium	190.23	-182.953
62	62	Ir	Iridium	192.22	-182.953
63	63	Pt	Platinum	195.08	-182.953
64	64	Au	Gold	196.967	-182.953
65	65	Hg	Mercury	200.59	-182.953
66	66	Tl	Thallium	204.383	-182.953
67	67	Pb	Lead	207.2	-182.953
68	68	Bi	Bismuth	208.980	-182.953
69	69	Po	Polonium	[208.982]	-182.953
70	70	At	Astatine	[209.987]	-182.953
71	71	Rn	Radon	222.018	-182.953
87	87	Fr	Francium	223.020	-182.953
88	88	Ra	Radium	226.025	-182.953
89	89	Rf	Rutherfordium	[261]	-182.953
90	90	Db	Dubnium	[262]	-182.953
91	91	Sg	Seaborgium	[266]	-182.953
92	92	Bh	Bohrium	[264]	-182.953
93	93	Hs	Hassium	[269]	-182.953
94	94	Mt	Meitnerium	[268]	-182.953
95	95	Ds	Darmstadtium	[269]	-182.953
96	96	Rg	Roentgenium	[272]	-182.953
97	97	Cn	Copernicium	[277]	-182.953
98	98	Uut	Ununtrium	unknown	-182.953
99	99	Fl	Flerovium	[289]	-182.953
100	100	Uup	Ununpentium	unknown	-182.953
101	101	Lv	Livermorium	[298]	-182.953
102	102	Uus	Ununseptium	unknown	-182.953
103	103	Uuo	Ununoctium	unknown	-182.953

Lanthanide Series

57	La	Lanthanum	138.906
58	Ce	Cerium	140.115
59	Pr	Praseodymium	140.908
60	Nd	Neodymium	144.24
61	Pm	Promethium	144.913
62	Sm	Samarium	150.36
63	Eu	Europium	151.966
64	Gd	Gadolinium	157.25
65	Tb	Terbium	158.925
66	Dy	Dysprosium	162.50
67	Ho	Holmium	164.930
68	Er	Erbium	167.26
69	Tm	Thulium	168.934
70	Yb	Ytterbium	173.04
71	Lu	Lutetium	174.967

Actinide Series

89	Ac	Actinium	227.028
90	Th	Thorium	232.038
91	Pa	Protactinium	231.036
92	U	Uranium	238.029
93	Np	Neptunium	237.048
94	Pu	Plutonium	244.064
95	Am	Americium	243.061
96	Cm	Curium	247.070
97	Bk	Berkelium	247.070
98	Cf	Californium	251.080
99	Es	Einsteinium	[254]
100	Fm	Fermium	257.095
101	Md	Mendelevium	258.1
102	No	Nobelium	259.101
103	Lr	Lawrencium	[262]

Classification Legend:

- Alkali Metal
- Alkaline Earth
- Transition Metal
- Basic Metal
- Semimetal
- Nonmetal
- Halogen
- Noble Gas
- Lanthanide
- Actinide

© 2014 Todd Helmenstine
sciencenotes.org

“Ontology is overrated.”

Clay Shirky

Ontologias são sobrevalorizadas.
Pesquise o artigo em inglês!

ThoughtWorks®

UMA ABORDAGEM MELHOR

Fundamental Features of Programming Languages

ENSINO MODERNO DE TEORIA DE LINGUAGENS

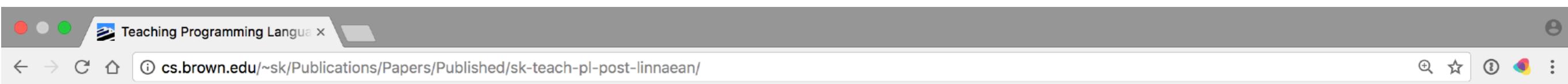
The screenshot shows a web browser window with the following details:

- Title Bar:** Teaching Programming Langua x
- Address Bar:** cs.brown.edu/~sk/Publications/Papers/Published/sk-teach-pl-post-linnaean/
- Content Area:**
 - Section Header:** Teaching Programming Languages in a Post-Linnaean Age
 - Author:** Shriram Krishnamurthi
 - Conference:** SIGPLAN Workshop on Undergraduate Programming
 - Abstract:** Programming language “paradigms” are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.
 - Comment:** The book discussed in this paper is available [here](#).
 - Paper:**
 - PDF:**
- Right Sidebar:** A yellow box containing the text "Ensinoando linguagens de programação na era pós-Lineu" (Ensinoando linguagens de programação na era pós-Lineu) in red font.

Sidebar (Vertical, Left): The sidebar features the text "Shriram Krishnamurthi" repeated vertically in white on a brown background.

Page Footer: These papers may differ in formatting from the versions that appear in print. They are made available only to support the rapid dissemination of results; the printed versions, not these, should be considered

ARTIGO APRESENTANDO A ABORDAGEM



Teaching Programming Languages in a Post-Linnaean Age

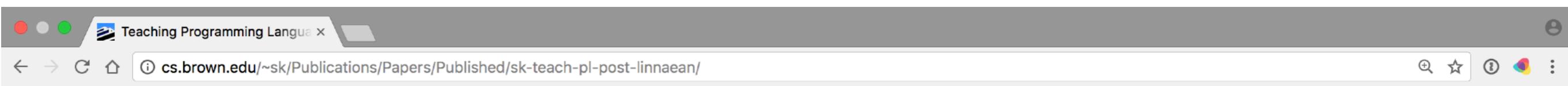
Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language “paradigms” are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.

ARTIGO APRESENTANDO A ABORDAGEM



Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

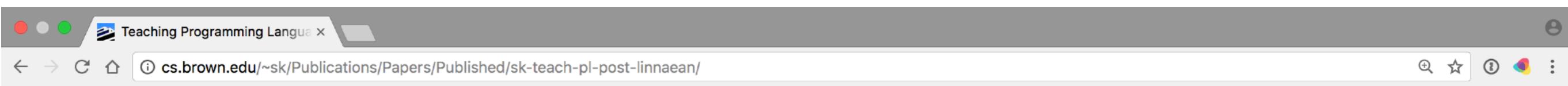
SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language “paradigms” are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly abandon this method of reconciles an important split in programming language education, and describes a textbook that explores these matters.

**“Paradigmas” de linguagens de programação
são um legado moribundo e tedioso...**

ARTIGO APRESENTANDO A ABORDAGEM



Teaching Programming Languages in a Post-Linnaean Age

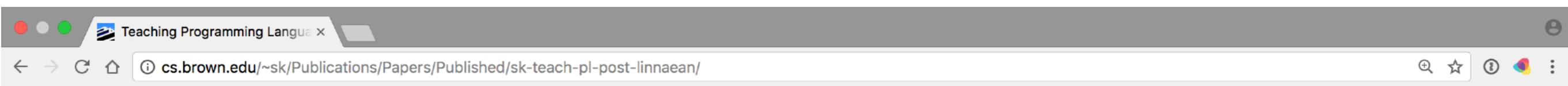
Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language “paradigms” are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.

ARTIGO APRESENTANDO A ABORDAGEM



Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

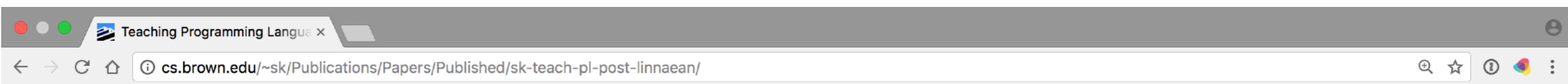
SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language “paradigms” are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching. It reconciles an important split in programming education: it describes a textbook that explores these matters.

Criadores de linguagens modernas
não têm o menor respeito por eles...

ARTIGO APRESENTANDO A ABORDAGEM



Teaching Programming Languages in a Post-Linnaean Age

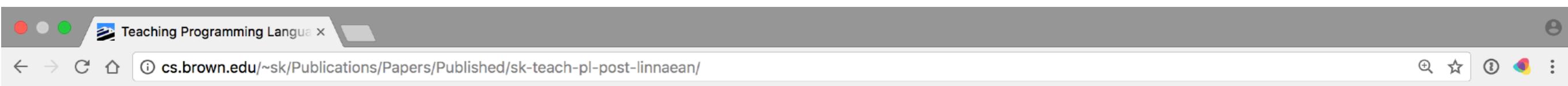
Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language “paradigms” are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.

ARTIGO APRESENTANDO A ABORDAGEM



Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

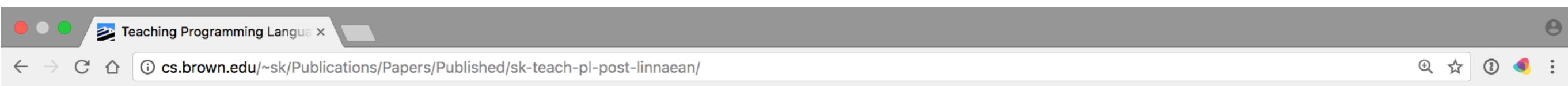
SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

...devemos abandonar esse método no ensino de linguagens...

do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.

ARTIGO APRESENTANDO A ABORDAGEM



Teaching Programming Languages in a Post-Linnaean Age

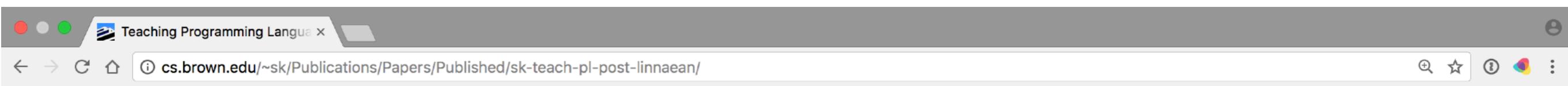
Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language “paradigms” are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching languages, offers an alternative, reconciles an important split in programming language education, and describes a textbook that explores these matters.

ARTIGO APRESENTANDO A ABORDAGEM



Teaching Programming Languages in a Post-Linnaean Age

Shriram Krishnamurthi

SIGPLAN Workshop on Undergraduate Programming Language Curricula, 2008

Abstract

Programming language “paradigms” are a moribund and tedious legacy of a bygone age. Modern language designers pay them no respect, so why do our courses slavishly adhere to them? This paper argues that we should abandon this method of teaching programming languages. It also discusses ...um livro-texto que explora esses assuntos. This reconciles an important split in programming language education, and describes a textbook that explores these matters.

TEORIA NA PRÁTICA COM RACKET (DIALETO DE SCHEME)

The screenshot shows a web browser window with the title "Programming Languages: Appl" and the URL "cs.brown.edu/courses/cs173/2012/book/". The page content is the table of contents for "Programming Languages: Application and Interpretation, Version Second Edition" by Shriram Krishnamurthi. The table of contents includes chapters 1 through 17, each with a list of sub-sections.

Version Second Edition

← prev up [next](#) →

Programming Languages: Application and Interpretation

by Shriram Krishnamurthi

1 Introduction

- 1.1 Our Philosophy
- 1.2 The Structure of This Book
- 1.3 The Language of This Book

2 Everything (We Will Say) About Parsing

- 2.1 A Lightweight, Built-In First Half of a Parser
- 2.2 A Convenient Shortcut
- 2.3 Types for Parsing
- 2.4 Completing the Parser
- 2.5 Coda

3 A First Look at Interpretation

- 3.1 Representing Arithmetic
- 3.2 Writing an Interpreter
- 3.3 Did You Notice?
- 3.4 Growing the Language

4 A First Taste of Desugaring

- 4.1 Extension: Binary Subtraction
- 4.2 Extension: Unary Negation

5 Adding Functions to the Language

- 5.1 Defining Data Representations
- 5.2 Growing the Interpreter
- 5.3 Substitution
- 5.4 The Interpreter, Resumed
- 5.5 Oh Wait, There's More!

6 From Substitution to Environments

- 6.1 Introducing the Environment

On this page:

- 1 Introduction
- 2 Everything (We Will Say) About Parsing
- 3 A First Look at Interpretation
- 4 A First Taste of Desugaring
- 5 Adding Functions to the Language
- 6 From Substitution to Environments
- 7 Functions Anywhere
- 8 Mutation: Structures and Variables
- 9 Recursion and Cycles: Procedures and Data
- 10 Objects
- 11 Memory Management
- 12 Representation Decisions
- 13 Desugaring as a Language Feature
- 14 Control Operations
- 15 Checking Program Invariants Statically: Types
- 16 Checking Program Invariants Dynamically: Contracts
- 17 Alternate Application Semantics

TEORIA NA PRÁTICA COM PASCAL (1990)

Programming Languages: Sam x Goodreads | Programming Lan x site:norvig.com kamin - Google x

Secure | https://www.goodreads.com/book/show/2082799.Programming_Languages?from_search=true

goodreads Home My Books Browse ▾ Community ▾ Search books

Recommend It | Stats | Recent Status Updates

Programming Languages: An Interpreter-Based Approach
by Samuel N. Kamin

★★★★★ 5.0 · Rating details · 1 Rating · 0 Reviews

GET A COPY

Amazon BR Online Stores ▾ Book Links ▾

Hardcover, 640 pages
Published January 1st 1990 by Addison Wesley Publishing Company

More Details... edit details

✓ Read My rating: ★★★★★

FRIEND REVIEWS

Recommend This Book None of your friends have reviewed this book yet.

READER Q&A

Ask the Goodreads community a question about Programming Languages

Ask anything about the book

Be the first to ask a question about Programming Languages

BOOKS BY SAMUEL N. KAMIN

BEGINNER'S GUIDE TO COMPUTER PROGRAMMING WITH PASCAL
An Introduction to Programming with Mathematica
An Introduction to Programming with Mathematica

LISTS WITH THIS BOOK

BBC micro:bit Go Bundle \$16.50 The British Invasion is here! No, not music...microcontrollers! New to the USA... Adafruit >

Science > Computer Science 1 user See top shelves...

More...



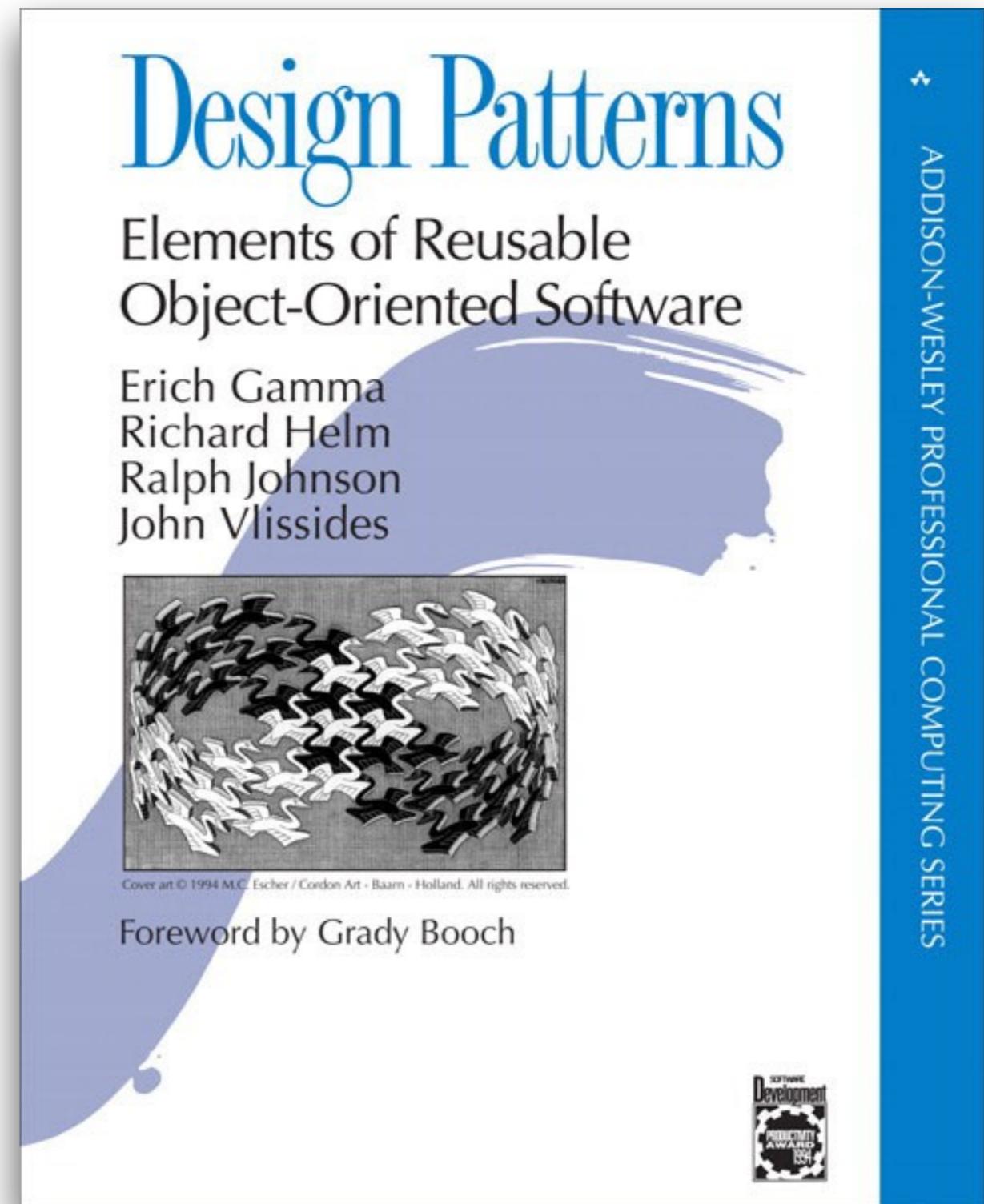
PADRÕES DE PROJETO

Quando as linguagens deixam a desejar

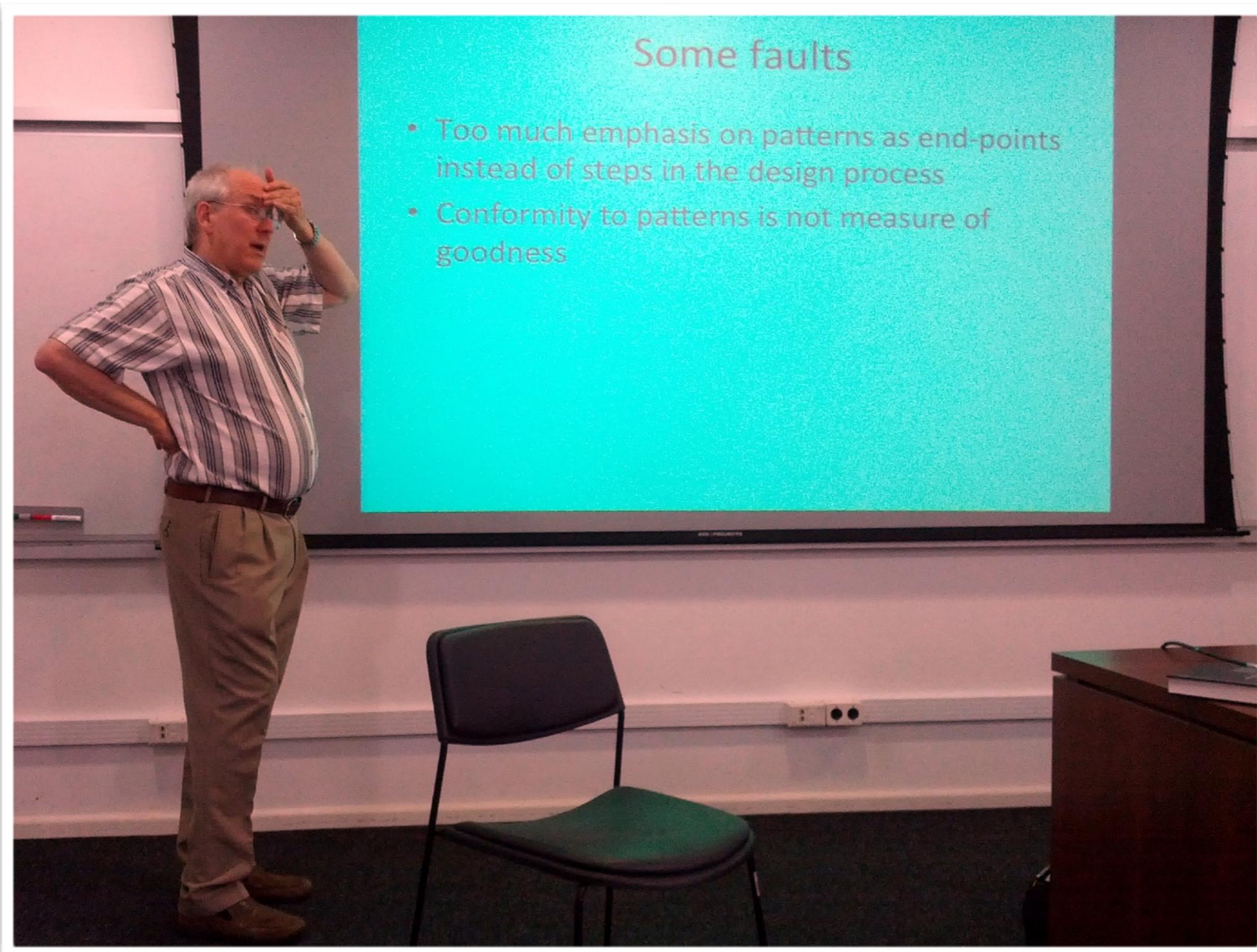
GOF: LIVRO CLÁSSICO DA “GANG OF FOUR”

Design Patterns:
Elements of Reusable
Object-Oriented
Software (1995)

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

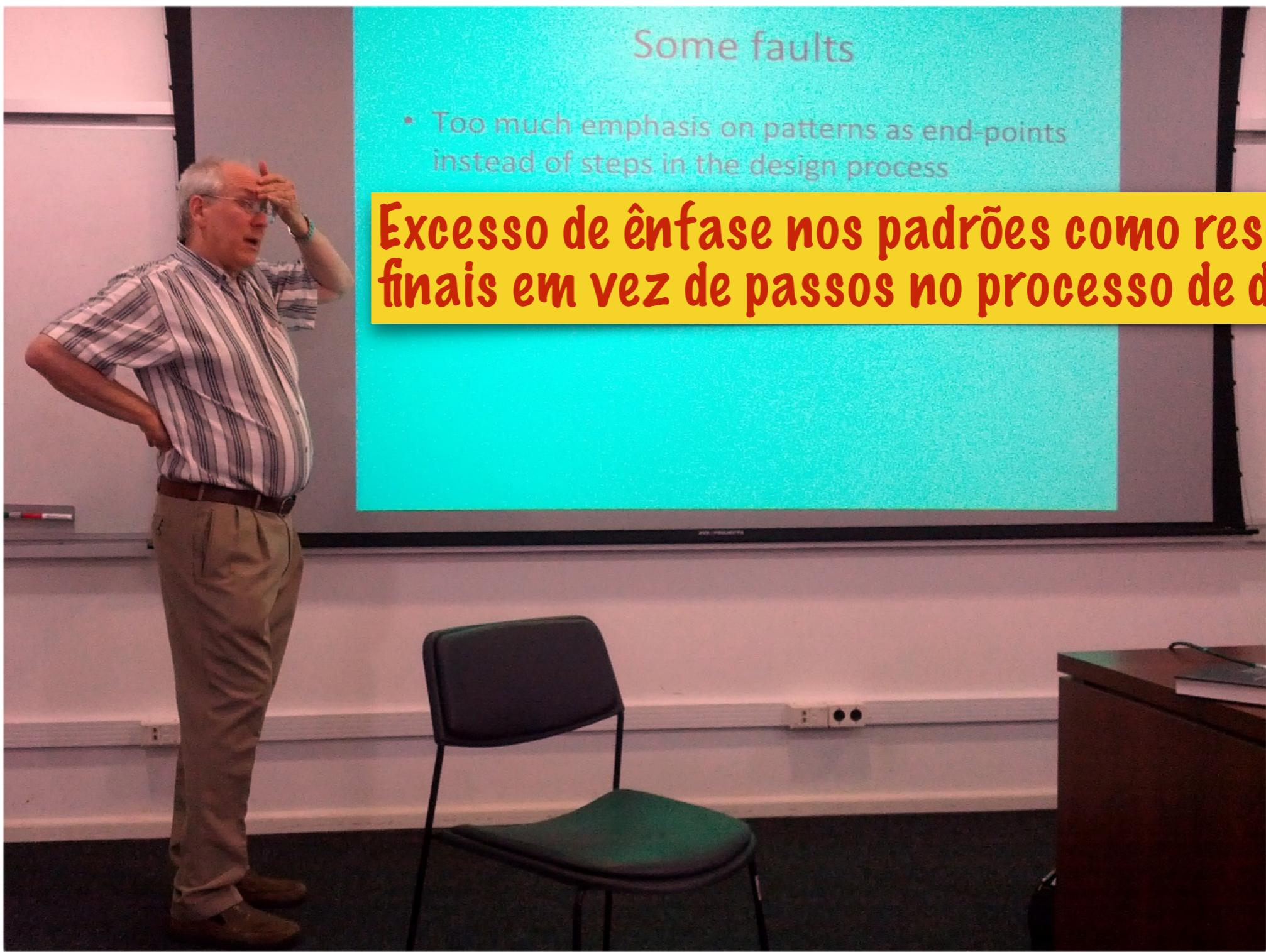


PROBLEMAS NA DIFUSÃO DOS PADRÕES



Ralph Johnson no IME/USP (2014?)

PROBLEMAS NA DIFUSÃO DOS PADRÕES



A photograph of Ralph Johnson giving a presentation. He is standing on a stage, facing a seated audience. He is wearing a striped shirt, khaki pants, and glasses. His right hand is on his hip, and his left hand is raised near his head. To his right is a large projection screen displaying a slide with the following content:

Some faults

- Too much emphasis on patterns as end-points instead of steps in the design process

Excesso de ênfase nos padrões como resultados finais em vez de passos no processo de design

Ralph Johnson no IME/USP (2014?)

PROBLEMAS NA DIFUSÃO DOS PADRÕES

A photograph of a man giving a presentation. He is standing on a stage, facing a seated audience. He is wearing a striped shirt, khaki pants, and glasses. His right hand is on his hip, and his left hand is raised near his head. To his right is a large projection screen. The screen has a teal header with the text "Some faults" and a yellow footer with the text "Conformidade com padrões não é uma medida de qualidade." The slide content is overlaid on the image.

Some faults

- Too much emphasis on patterns as end-points instead of steps in the design process
- Conformity to patterns is not measure of goodness

Conformidade com padrões não é uma medida de qualidade.

Ralph Johnson no IME/USP (2014?)

NEM TODO PADRÃO É UNIVERSAL

The choice of programming language is important because it influences one's point of view. Our patterns assume Smalltalk/C++-level language features, and that choice determines what can and cannot be implemented easily. If we assumed procedural languages, we might have included design patterns called "Inheritance," "Encapsulation," and "Polymorphism." Similarly, some of our patterns are supported directly by the less common object-oriented languages. CLOS has multi-methods, for example, which lessen the need for a pattern such as Visitor.²

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, 1995), p. 4.

NEM TODO PADRÃO É UNIVERSAL

The choice of programming language is important because it influences one's point of view. Our patterns assume Smalltalk/C++-level language features, and that choice determines what can and cannot be implemented easily. If we assumed procedural languages, we might have included “Iteration,” “Recursion,” and “Conditionals” and “Polymorphism.” Similarly, if we assumed a functional language, many common object-oriented patterns would not be needed, and others would lessen the need for a pattern such as Visitor.²

Nossos padrões assumem características no nível de Smalltalk/C++...

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley, 1995), p. 4.

NEM TODO PADRÃO É UNIVERSAL

The choice of programming language is important because it influences one's point of view. Our patterns assume Smalltalk/C++-level language features, and that choice determines what can and cannot be implemented easily. If we assumed procedural languages, we might have included design patterns called "Inheritance," "Encapsulation," and "Polymorphism." Similarly, some of our patterns are supported directly by the less common object-oriented languages. CLOS has multi-methods, for example, which lessen the need for a pattern such as Visitor.²

**Se tivéssemos assumido linguagens procedurais,
talvez tivéssemos incluído padrões como
"Herança", "Encapsulamento" e "Polimorfismo".**

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley, 1995), p. 4.

Design Patterns in Dynamic Programming

Peter Norvig

Chief Designer, Adaptive Systems
Harlequin Inc.

(2) Design Patterns in Dynamic Languages

- ◆ Dynamic Languages have fewer language limitations
 - Less need for bookkeeping objects and classes
 - Less need to get around class-restricted design
- ◆ Study of the *Design Patterns* book:
 - 16 of 23 patterns have qualitatively simpler implementation in Lisp or Dylan than in C++ for at least some uses of each pattern
- ◆ Dynamic Languages encourage new designs
 - We will see some in Part (3)

(2) Design Patterns in Dynamic Languages

- ◆ Dynamic Languages have fewer language limitations
Less need for bookkeeping objects and classes
Less need to get around class-restricted design
- ◆ Study of the *Design Patterns* book:
16 of 23 patterns have qualitatively simpler implementation in Lisp or Dylan than in C++ for at least some uses of each pattern
- ◆ Dynamic Languages encourage new designs

We will see so

Estudo do livro “Padrões de Projeto”: 16 dos 23 padrões têm implementações qualitativamente mais simples em Lisp ou Dylan do que em C++, ao menos para alguns usos de cada padrão.

Design Patterns in Dylan or Lisp

16 of 23 patterns are either invisible or simpler, due to:

- ◆ First-class types (6): Abstract-Factory, Flyweight, Factory-Method, State, Proxy, Chain-Of-Responsibility
- ◆ First-class functions (4): Command, Strategy, Template-Method, Visitor
- ◆ Macros (2): Interpreter, Iterator
- ◆ Method Combination (2): Mediator, Observer
- ◆ Multimethods (1): Builder
- ◆ Modules (1): Facade

16 dos 23 padrões são invisíveis ou mais simples, devido a essas características



CARACTERÍSTICAS

Características essenciais, não apenas açúcar sintático.

AMOSTRA DE CARACTERÍSTICAS × LINGUAGENS

	Common Lisp
Funções de 1 ^a classe	✓
Tipos de 1 ^a classe	✓
Iteradores	*
Modelo de variáveis	referência
Checagem de tipos	dinâmica
Expressão de tipos	estrutural

AMOSTRA DE CARACTERÍSTICAS × LINGUAGENS

	Common Lisp
Funções são objetos	✓
Classes são objetos	✓
Iteradores	*
Modelo de variáveis	referência
Checagem de tipos	dinâmica
Expressão de tipos	estrutural

AMOSTRA DE CARACTERÍSTICAS × LINGUAGENS

	Common Lisp	C
Funções de 1 ^a classe	✓	*
Tipos de 1 ^a classe	✓	
Iteradores	*	
Modelo de variáveis	referência	valor*
Checagem de tipos	dinâmica	estática
Expressão de tipos	estrutural	nominal

AMOSTRA DE CARACTERÍSTICAS × LINGUAGENS

	Common Lisp	C	Java
Funções de 1 ^a classe	✓	*	✓
Tipos de 1 ^a classe	✓		
Iteradores	*		✓
Modelo de variáveis	referência	valor*	valor e referência
Checagem de tipos	dinâmica	estática	estática
Expressão de tipos	estrutural	nominal	nominal

AMOSTRA DE CARACTERÍSTICAS × LINGUAGENS

	Common Lisp	C	Java	Go
Funções de 1 ^a classe	✓	*	✓	✓
Tipos de 1 ^a classe	✓			
Iteradores	*		✓	*
Modelo de variáveis	referência	valor*	valor e referência	valor* e referência
Checagem de tipos	dinâmica	estática	estática	estática
Expressão de tipos	estrutural	nominal	nominal	estrutural

AMOSTRA DE CARACTERÍSTICAS × LINGUAGENS

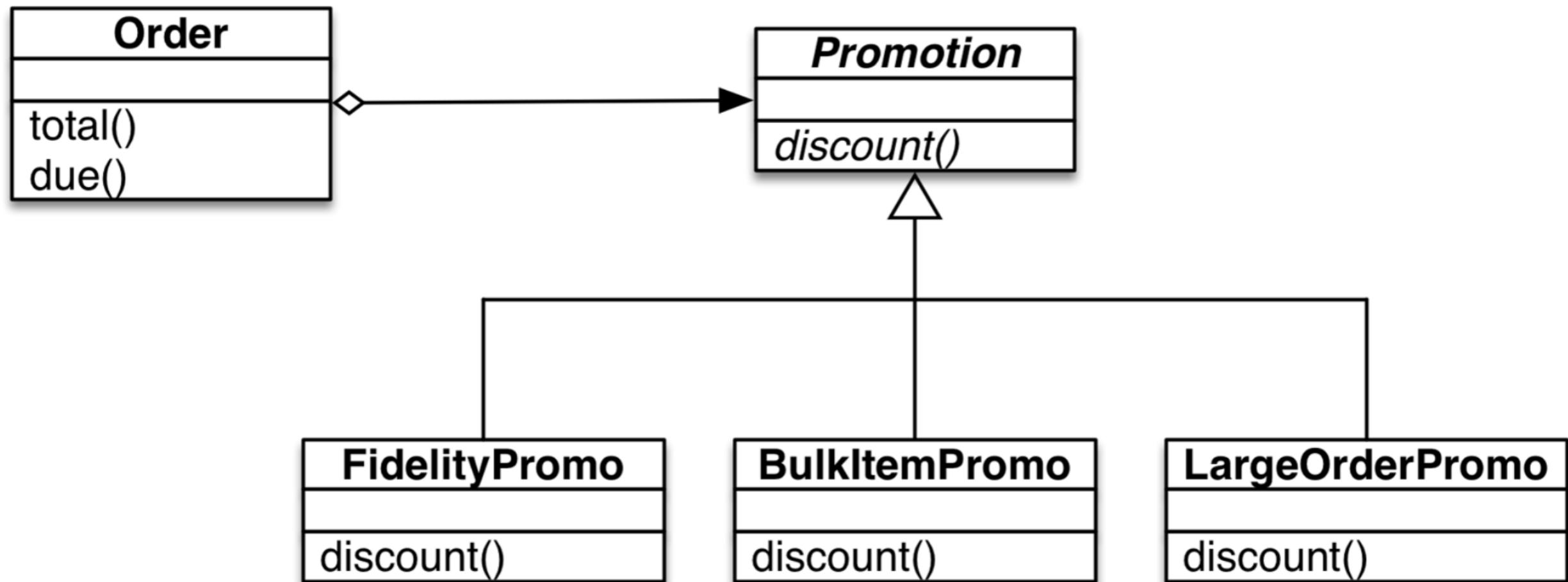
	Common Lisp	C	Java	Go	Python
Funções de 1 ^a classe	✓	*	✓	✓	✓
Tipos de 1 ^a classe	✓				✓
Iteradores	*		✓	*	✓
Modelo de variáveis	referência	valor*	valor e referência	valor* e referência	referência
Checagem de tipos	dinâmica	estática	estática	estática	dinâmica
Expressão de tipos	estrutural	nominal	nominal	estrutural	estrutural



STRATEGY EM PYTHON

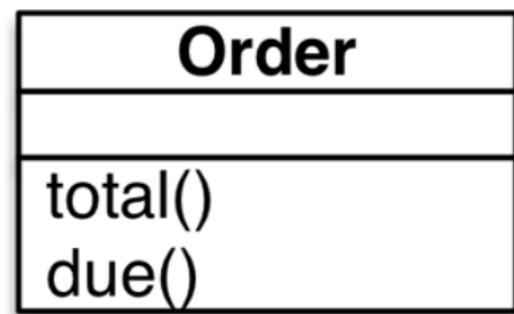
Aproveitando características de Python

SELEÇÃO DE UM ALGORITMO EM TEMPO DE EXECUÇÃO

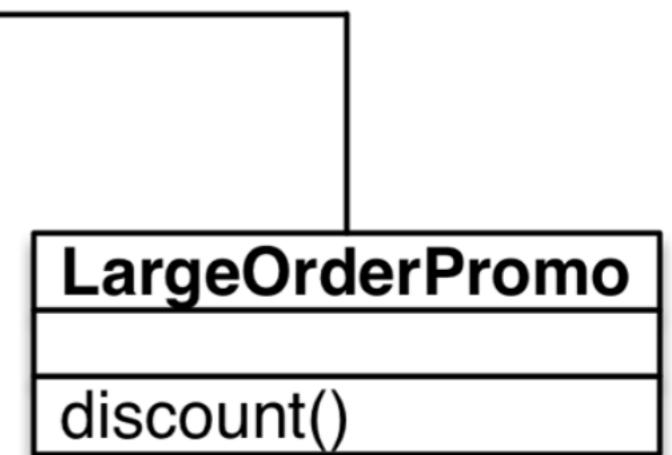
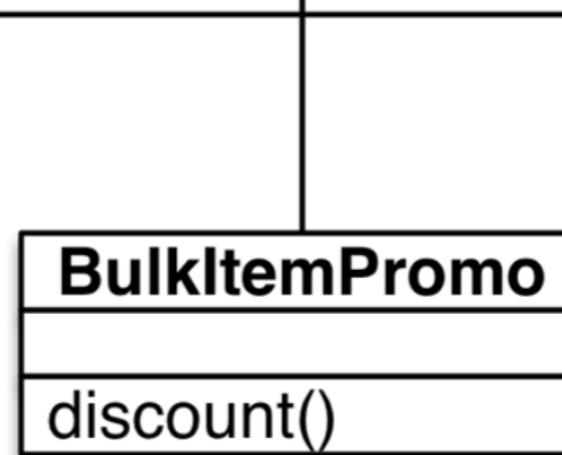
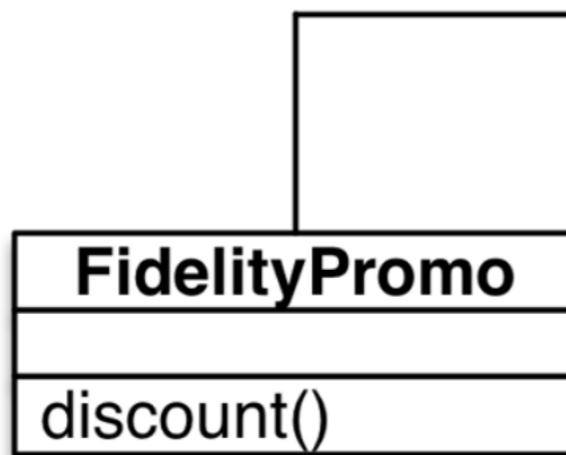
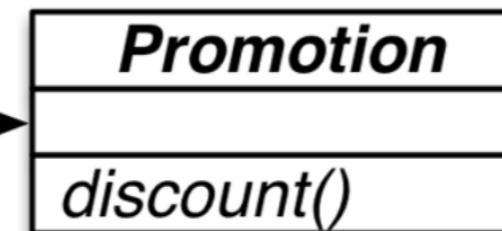


SELEÇÃO DE UM ALGORITMO EM TEMPO DE EXECUÇÃO

Contexto



Estratégia



Estratégias concretas

DOCTESTS: CONTEXTO E UMA ESTRATÉGIA CONCRETA

Create two customers, with and without "fidelity points":

```
>>> ann = Customer('Ann Smith', 1100) # ①  
>>> joe = Customer('John Doe', 0)
```

Create a shopping cart with some fruits:

```
>>> cart = [LineItem('banana', 4, .5), # ②  
...     LineItem('apple', 10, 1.5),  
...     LineItem('watermelon', 5, 5.0)]
```

The `FidelityPromo` only gives a discount to Ann:

```
>>> Order(joe, cart, FidelityPromo())  
<Order total: 42.00 due: 42.00>  
>>> Order(ann, cart, FidelityPromo())  
<Order total: 42.00 due: 39.90>
```

Instância de Estratégia (Promo) é passada ao construtor do Contexto (Order: o pedido)

DOCTESTS: MAIS DUAS ESTRATÉGIAS CONCRETAS

The `BulkItemPromo` gives a discount for items with 20+ units:

```
>>> banana_cart = [LineItem('banana', 30, .5), # ⑤
...                   LineItem('apple', 10, 1.5)]
>>> Order(joe, banana_cart, BulkItemPromo()) # ⑥
<Order total: 30.00 due: 28.50>
```

`LargeOrderPromo` gives a discount for orders with 10+ items:

```
>>> long_order = [LineItem(str(item_code), 1, 1.0) # ⑦
...                   for item_code in range(10)]
>>> Order(joe, long_order, LargeOrderPromo()) # ⑧
<Order total: 10.00 due: 9.30>
```

VARIAÇÕES DE STRATEGY EM PYTHON

Implementação **clássica** usando ABC

Implementação com **funções de 1ª classe**

Implementação parametrizada com **closure**

Implementação parametrizada com **classe invocável**

STRATEGY CLÁSSICO: A CLASSE DO CONTEXT

```
class Order: # the Context

    def __init__(self, customer, cart, promotion=None):
        self.customer = customer
        self.cart = list(cart)
        self.promotion = promotion

    def total(self):
        if not hasattr(self, '__total'):
            self.__total = sum(item.total() for item in self.cart)
        return self.__total

    def due(self):
        if self.promotion is None:
            discount = 0
        else:
            discount = self.promotion.discount(self)
        return self.total() - discount

    def __repr__(self):
        fmt = '<Order total: {:.2f} due: {:.2f}>'
        return fmt.format(self.total(), self.due())
```

Estratégia é passada para o construtor

Estratégia é usada aqui

STRATEGY: CLASSE ABSTRATA E SUBCLASSE CONCRETA

```
class Promotion(ABC): # the Strategy: an Abstract Base Class

    @abstractmethod
    def discount(self, order):
        """Return discount as a positive dollar amount"""

class FidelityPromo(Promotion): # first Concrete Strategy
    """5% discount for customers with 1000 or more fidelity points"""

    def discount(self, order):
        return order.total() * .05 if order.customer.fidelity >= 1000 else 0
```

MAIS DUAS ESTRATÉGIAS CONCRETAS

```
class BulkItemPromo(Promotion): # second Concrete Strategy
    """10% discount for each LineItem with 20 or more units"""

    def discount(self, order):
```

```
        discount = 0
        for item in order.cart:
            if item.quantity >= 20:
                discount += item.total() * .1
        return discount
```

```
class LargeOrderPromo(Promotion): # third Concrete Strategy
    """7% discount for orders with 10 or more distinct items"""

    def discount(self, order):
```

```
        distinct_items = {item.product for item in order.cart}
        if len(distinct_items) >= 10:
            return order.total() * .07
        return 0
```



STRATEGY COM FUNÇÃO DE 1^A CLASSE

CONTEXTO: FUNÇÃO DE ESTRATÉGIA COMO ARGUMENTO

Função de estratégia é
passada ao construtor
de Order

```
>>> banana_cart = [LineItem('banana', 30, .5),  
...                      LineItem('apple', 10, 1.5)]  
>>> Order(joe, banana_cart, bulk_item_promo)  
<Order total: 30.00 due: 28.50>  
>>> long_order = [LineItem(str(item_code), 1, 1.0)  
...                      for item_code in range(10)]  
>>> Order(joe, long_order, large_order_promo)  
<Order total: 10.00 due: 9.30>  
>>> Order(joe, cart, large_order_promo)  
<Order total: 42.00 due: 42.00>
```

CONTEXTO: USO DA FUNÇÃO DE ESTRATÉGIA

classic_strategy.py ↔ strategy.py



```
self.customer = customer
self.cart = list(cart)
self.promotion = promotion
```

```
def total(self):
    if not hasattr(self, '_total'):
        self._total = sum(item.total() for item in self.cart)
    return self._total
```

```
def due(self):
    if self.promotion is None:
        discount = 0
    else:
        - discount = self.promotion.discount(self)
    return self.total() - discount
```

```
def __repr__(self):
    fmt = '<Order total: {:.2f} due: {:.2f}>'
    return fmt.format(self.total(), self.due())
```

- ss Promotion(ABC): # the Strategy: an Abstract Base Class

-

- @abstractmethod

```
self.customer = customer
self.cart = list(cart)
self.promotion = promotion
```

```
def total(self):
    if not hasattr(self, '_total'):
        self._total = sum(item.total() for item in self.cart)
    return self._total
```

```
def due(self):
    if self.promotion is None:
        discount = 0
    else:
        + discount = self.promotion(self)
    return self.total() - discount
```

```
def __repr__(self):
    fmt = '<Order total: {:.2f} due: {:.2f}>'
    return fmt.format(self.total(), self.due())
```

+ fidelity_promo(order):

ESTRATÉGIAS CONCRETAS COMO FUNÇÕES

classic_strategy.py ↔ strategy.py



```
- class Promotion(ABC): # the Strategy: an Abstract Base Class
-     @abstractmethod
-         def discount(self, order):
-             """Return discount as a positive dollar amount"""
-
- class FidelityPromo(Promotion): # first Concrete Strategy
-     """5% discount for customers with 1000 or more
-     fidelity points"""
-     def discount(self, order):
-         return order.total() * .05 if order.customer.fidelity
-             >= 1000 else 0
-
- class BulkItemPromo(Promotion): # second Concrete Strategy
-     """10% discount for each LineItem with 20 or
-     more units"""
-     def discount(self, order):
-         discount = 0
-         for item in order.cart:
-             if item.quantity >= 20:
-                 discount += item.total() * .1
-         return discount
+ def fidelity_promo(order):
+     """5% discount for customers with 1000 or more
+     fidelity points"""
+     return order.total() * .05 if order.customer.fidelity
+         >= 1000 else 0
+
+ def bulk_item_promo(order):
+     """10% discount for each LineItem with 20 or
+     more units"""
+     discount = 0
+     for item in order.cart:
+         if item.quantity >= 20:
+             discount += item.total() * .1
+     return discount
```

ESTRATÉGIAS CONCRETAS COMO FUNÇÕES (2)

classic_strategy.py ↔ strategy.py



```
- class BulkItemPromo(Promotion): # second Concrete Strategy
    """10% discount for each LineItem with 20 or more units"""

    def discount(self, order):
        discount = 0
        for item in order.cart:
            if item.quantity >= 20:
                discount += item.total() * .1
        return discount

+ def bulk_item_promo(order):
    """10% discount for each LineItem with 20 or more units"""

    discount = 0
    for item in order.cart:
        if item.quantity >= 20:
            discount += item.total() * .1
    return discount

- class LargeOrderPromo(Promotion): # third Concrete Strategy
    """7% discount for orders with 10 or more distinct items"""

    def discount(self, order):
        distinct_items = {item.product for item in order.cart}
        if len(distinct_items) >= 10:
            return order.total() * .07
        return 0

+ def large_order_promo(order):
    """7% discount for orders with 10 or more distinct items"""

    distinct_items = {item.product for item in order.cart}
    if len(distinct_items) >= 10:
        return order.total() * .07
    return 0
```

ESTRATÉGIA PARAMETRIZADA COM CLOSURE

ESTRATÉGIA PARAMETRIZADA COM CLOSURE

função promo é
invocada com
porcentagem de
desconto

```
>>> banana_cart = [LineItem('banana', 30, .5),  
...                   LineItem('apple', 10, 1.5)]  
>>> Order(joe, banana_cart, bulk_item_promo(10))  
<Order total: 30.00 due: 28.50>  
>>> long_order = [LineItem(str(item_code), 1, 1.0)  
...                   for item_code in range(10)]  
>>> Order(joe, long_order, large_order_promo(7))  
<Order total: 10.00 due: 9.30>  
>>> Order(joe, cart, large_order_promo(7))  
<Order total: 42.00 due: 42.00>
```

IMPLEMENTAÇÃO COM CLOSURE

```
def bulk_item_promo(percent):
    """discount for each LineItem with 20 or more units"""
    def discounter(order):
        discount = 0
        for item in order.cart:
            if item.quantity >= 20:
                discount += item.total() * percent/100.0
        return discount
    return discounter
```

Função externa recebe argumento 'percent'

Função interna leva associação de `percent` em sua closure

LAMBDA: ATALHO PARA DEFINIR A FUNÇÃO INTERNA

strategy.py ↔ strategy_param.py

A set of small, light-gray navigation icons located at the bottom right of the slide. From left to right, they include: a double arrow pointing left, a single arrow pointing left, a single arrow pointing right, three vertical dots, and a double arrow pointing right.

```
- def fidelity_promo(order):
-     """5% discount for customers with 1000 or more
-     return order.total() * .05 if order.customer.
```

```
+ def fidelity_promo(percent):
+     """discount for customers with 1000 o
+     return lambda order: (order.total() *
+                           percent) if order.custom
```

```
- def bulk_item_promo(order):
-     """10% discount for each LineItem with 20 or
+     """discount for each LineItem with 20
+     def discounter(order):
+         discount = 0
+         for item in order.cart:
+             if item.quantity >= 20:
+                 discount += item.total() * .1
+         return discount
```

```
- def large_order_promo(order):
-     """7% discount for orders with 10 or more distinct items"""
+     """discount for orders with 10 or more distinct items"""
+     def discounter(order):
+         distinct_items = {item.product for item in order.items}
+         if len(distinct_items) >= 10:
+             return order.total() * .07
+         return 0
```

ESTRATÉGIA PARAMETRIZADA COM INVOCÁVEL

ESTRATÉGIA PARAMETRIZADA COM INVOCÁVEL

objeto promo é
instanciado com
porcentagem de
desconto

```
>>> banana_cart = [LineItem('banana', 30, .5),  
...                   LineItem('apple', 10, 1.5)]  
>>> Order(joe, banana_cart, BulkItemPromo(10))  
<Order total: 30.00 due: 28.50>  
>>> long_order = [LineItem(str(item_code), 1, 1.0)  
...                  for item_code in range(10)]  
>>> Order(joe, long_order, LargeOrderPromo(7))  
<Order total: 10.00 due: 9.30>  
>>> Order(joe, cart, LargeOrderPromo(7))  
<Order total: 42.00 due: 42.00>
```

CLASSE IMPLEMENTADA COMO INVOCÁVEL

strategy_param.py ↔ strategy_param2.py



- def fidelity_promo(percent):

"""discount for customers with 1000 or more fidelity points"""

return lambda order: (order.total() * percent) if order.customer.fidelity >= 1000 else 0

- def bulk_item_promo(percent):

"""discount for each LineItem with 20 or more units"""

- def discouter(order):

discount = 0

+ class Promotion():

"""compute discount for order"""

+ def __init__(self, percent):
+ self.percent = percent

+ def __call__(self, order):
+ raise NotImplementedError("Subclass re")

+ class FidelityPromo(Promotion):

"""discount for customers with 1000 or more fidelity points"""

+ def __call__(self, order):
+ if order.customer.fidelity >= 1000:
+ return order.total() * self.percent
+ return 0

+ class BulkItemPromo(Promotion):

"""discount for each LineItem with 20 or more units"""

+ def __call__(self, order):

discount = 0

SUB-CLASSES IMPLEMENTADAS COMO INVOCÁVEIS

strategy_param.py ↔ strategy_param2.py

```
- def bulk_item_promo(percent):
    """discount for each LineItem with 20 or more
- def discouter(order):
    discount = 0
    for item in order.cart:
        if item.quantity >= 20:
            discount += item.total() * percer
    return discount
- return discouter

- def large_order_promo(percent):
    """discount for orders with 10 or more distir
- def discouter(order):
    distinct_items = {item.product for item in order.cart}
    if len(distinct_items) >= 10:
        return order.total() * percent / 100.
    return 0
- return discouter

+ class BulkItemPromo(Promotion):
    """discount for each LineItem with 20 or more
+ def __call__(self, order):
    discount = 0
    for item in order.cart:
        if item.quantity >= 20:
            discount += item.total()
    return discount

+ class LargeOrderPromo(Promotion):
    """discount for orders with 10 or more distinct items
+ def __call__(self, order):
    distinct_items = {item.product for item in order.cart}
    if len(distinct_items) >= 10:
        return order.total() * self.percent
    return 0
```

ThoughtWorks®

QUAL A SOLUÇÃO MAIS IDIOMÁTICA?

Classes x funções

QUAL É MAIS IDIOMÁTICA?

Estratégia clássica *parece* muito verbosa em Python*

Funções de 1^a classe são muito comuns na biblioteca padrão

- Exemplo: argumento **key** nas funções embutidas **sorted**, **max**, **min**

* Sim, isto é subjetivo. Estamos falando de estilo!

QUAL É MAIS IDIOMÁTICA COM PARÂMETROS?

Use de closures é comum em Python

- Python 3 ganhou a declaração **nonlocal** para suportar melhor essa prática

Objetos invocáveis são característica típica de Python

- Graham Dumpleton recomenda invocáveis para construir **@decorators**



CONCLUINDO

Aprenda as características fundamentais

PORQUE APRENDER AS CARACTERÍSTICAS FUNDAMENTAIS

Aprender novas linguagens mais rápido

Aproveitar melhor as características de cada linguagem

Escolher entre implementações alternativas

Fazer bom uso de padrões de projeto

Depurar problemas difíceis

Emular características que fazem falta

* Inspirado por *Programming Language Pragmatics*

Michael L. Scott

MUITO GRATO!

Luciano Ramalho

luciano.ramalho@thoughtworks.com

Twitter: @ramalhoorg

Github: github.com/standupdev/paradigm-free

Slides: speakerdeck.com/ramalho

ThoughtWorks®