ThoughtWorks®

Simple is better than complex.

GO: A VISÃO DE UM PYTHONISTA

Minha opinão pessoal sobre Go e seu lugar diante das necessidades atuais de desenvolvimento.

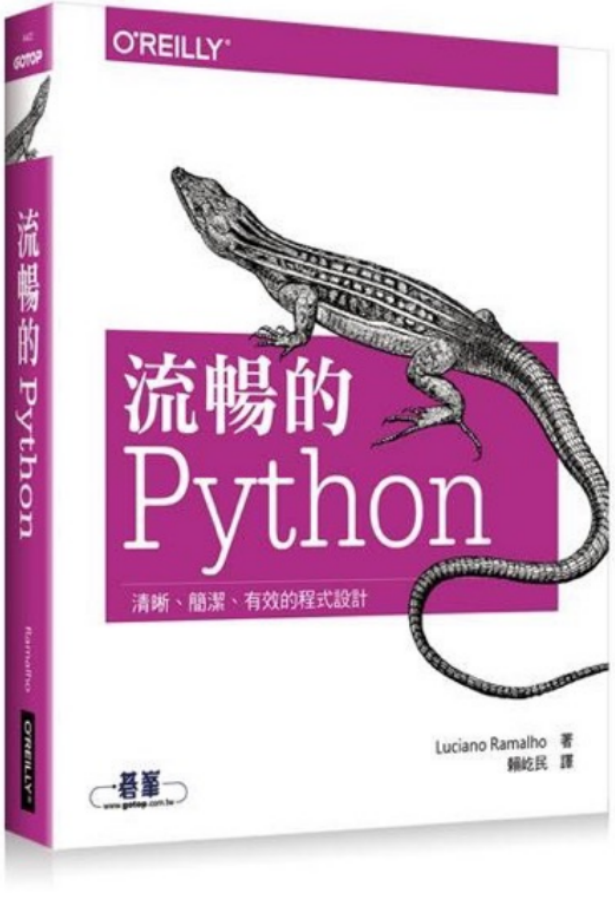
ThoughtWorks®

LUCIANO RAMALHO

Technical Principal

@ramalhoorg
luciano.ramalho@thoughtworks.com

FLUENT PYTHON, MEU PRIMEIRO LIVRO



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...**

** Python. To the heights of excellence*
† Smooth Python

ThoughtWorks®

MINHA JORNADA

Codando desde 1979

JÁ CODEI PROFISSIONALMENTE EM...

- BASIC
- xBase
- **Turbo Pascal**
- C++
- Hypertalk e Lingo
- Visual Basic
- **Delphi**
- Perl
- JavaScript (Netscape Enterprise Server)
- PHP
- Java
- Python
- JavaScript (Node)
- Ruby

JÁ ESTUDEI OU FIZ HACKS COM...

- TI-58/59, HP-25, HP-11c
- Assembly Z80, 6502, 8088
- Smalltalk
- Eiffel
- Scheme (um Lisp)
- C
- Elixir
- Go

PERSPECTIVA ATUAL

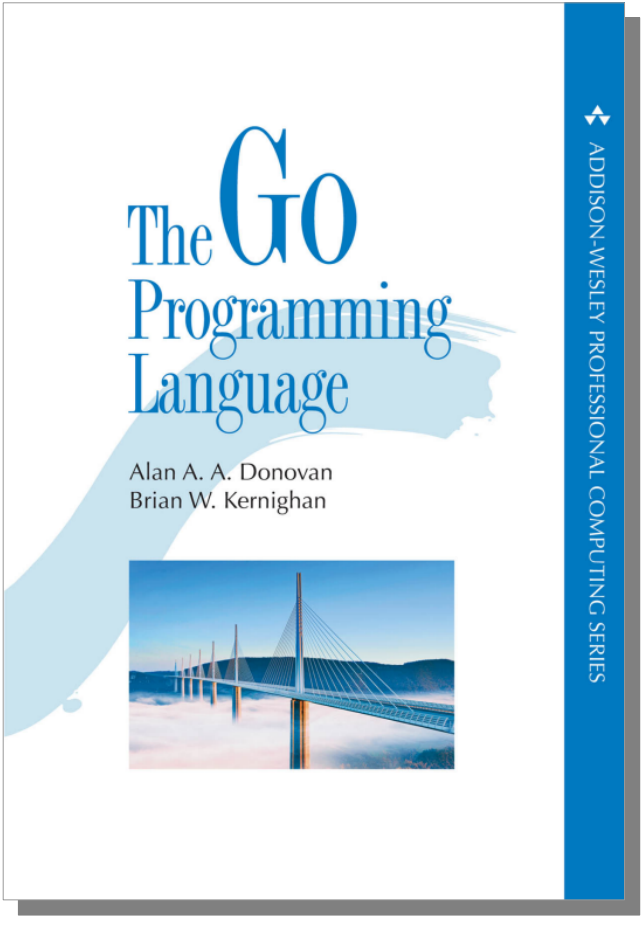
- Desde 2013 ativamente buscando reforçar minhas habilidades em **outras linguagens**. (pausa para escrever *Fluent Python*)
- Contratado pela **ThoughtWorks**, uma empresa de devs **poliglotas**
- Estudando arquitetura de **sistemas distribuídos** para construir **plataformas**
- Foco atual: **Go**, **Elixir**
- Na fila: Erlang, Elm, Clojure, Rust

ThoughtWorks®

GO

Pascal para o século 21

A “BÍBLIA”

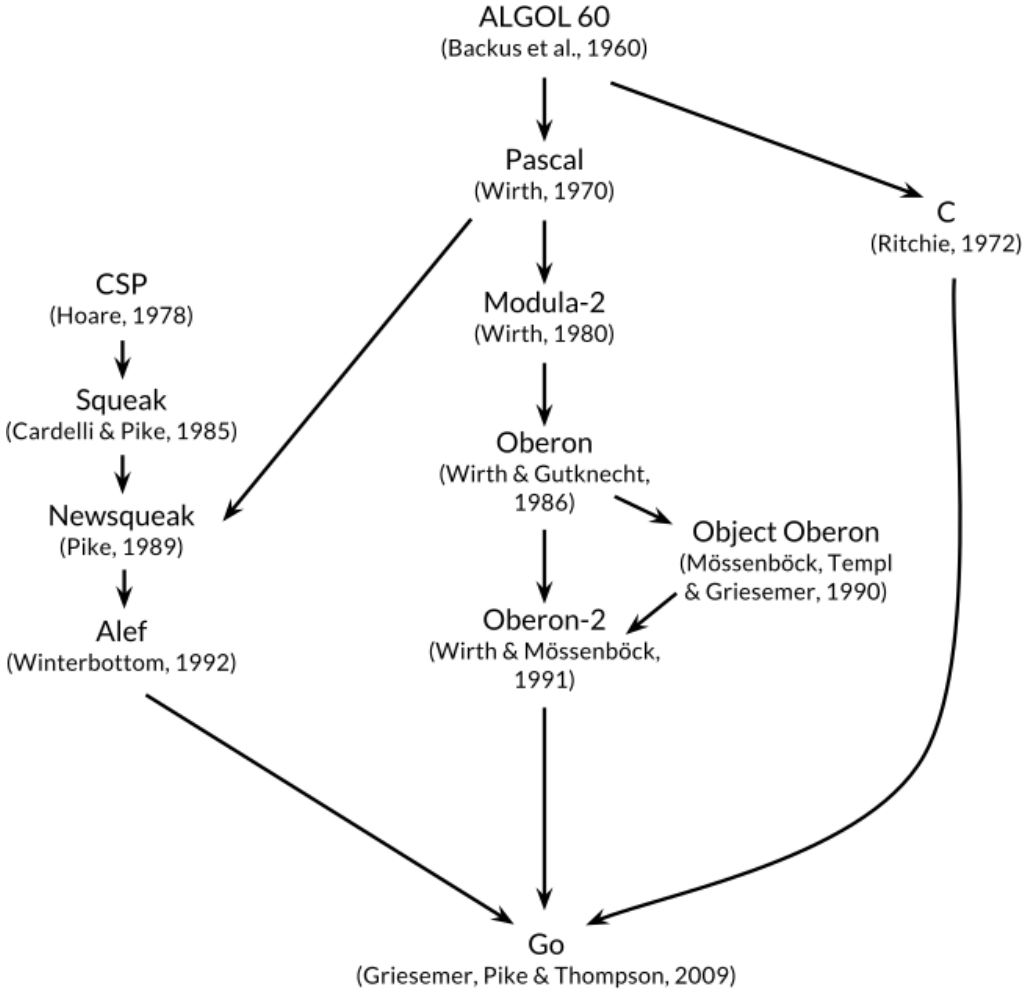


“GOPL”, de Alan Donovan e Brian Kernighan (o K de K&R)

Completo, preciso e com exemplos excelentes

No Brasil: *A Linguagem de Programação Go* (Novatec, 2017)

ORIGENS DE GO*



* Figura do GOPL

O SISTEMA DE TIPOS DE GO*

Go tem um sistema de tipos suficiente para evitar a maioria dos erros por descuido que aflige programadores de linguagens dinâmicas, porém seu sistema é mais simples que os de linguagens tipadas comparáveis a ela. Essa abordagem, às vezes, pode resultar em regiões isoladas de programação ‘‘não tipada’’ em um framework mais amplo de tipos, e programadores que usam Go não chegam ao ponto de fazer o que programadores de C++ e Haskell fazem para expressar propriedades de segurança como provas baseadas em tipos. Na prática, porém, Go oferece aos programadores uma boa dose de proteção e de vantagens quanto ao desempenho em tempo de execução, resultantes de um sistema de tipagem relativamente forte sem o peso de um sistema complexo.

* Trecho do prefácio de GOPL

PRINCIPAIS CARACTERÍSTICAS DE GO

- Compila direto para **binários nativos** otimizados: gera executáveis de alto desempenho que funcionam sem dependências externas
- Sistema de tipos simples: **structs**, **métodos** e **interfaces** (sem herança, sobrecarga etc.)
- Suporte a concorrência, paradigma CSP (Communicating Sequential Processes): **gorrotinas**, **canais** e instrução **select**
- Pilhas incluídas: **biblioteca** e **ferramentas**

SINTAXE INTELIGENTE

- Única instrução de laço: **for**
  (com a variante **for**/**range**)
- Instrução **defer**
- Instruções **go** e **select**
- Indicação de público/privado pela letra inicial do identificador (Xxx é público)
- Notação expressiva para literais de tipos compostos diversos
- Sintaxe bastante regular (poucas surpresas)
  + **go fmt**

PALAVRAS RESERVADAS EM LINGUAGENS

Table 16-1. Number of keywords in programming languages

Keywords	Language	Comment
5	Smalltalk-80	Famous for its minimalist syntax.
25	Go	The language, not the game.
32	C	That's ANSI C. C99 has 37 keywords, C11 has 44.
33	Python	Python 2.7 has 31 keywords; Python 1.5 had 28.
41	Ruby	Keywords may be used as identifiers (e.g., `class` is also a method name).
49	Java	As in C, the names of the primitive types (`char`, `float`, etc.) are reserved.
60	JavaScript	Includes all keywords from Java 1.0, many of which are unused.
65	PHP	Since PHP 5.3, seven keywords were introduced, including `goto`, `trait`, and `yield`.
85	C++	According to cppreference.com, C++11 added 10 keywords to the existing 75.
555	COBOL	I did not make this up. See this IBM ILE COBOL manual.
∞	Scheme	Anyone can define new keywords.

MAIS CARACTERÍSTICAS DE GO

- Tipos compostos embutidos:
  array, **slice**, **struct**, **map**
- Suporte a Unicode: tipo **string** é uma sequência de bytes, mas biblioteca padrão lida naturalmente com UTF-8 e o **for**/**range** itera por caracteres (**rune**) e não por bytes
- Biblioteca padrão excelente para sistemas distribuídos: já suporta HTTP/2!

COISAS QUE EU SINTO FALTA

- Protocolo de iteração (interface Iterable) aberto para o usuário da linguagem criar coleções que funcionam com **for/range**
- Métodos para coleções nativas (ex. **Contains**)
- Alguma solução para a necessidade de reimplementar de novo tipos básicos como **set**


ThoughtWorks®
SHOW ME THE CODE
Comparando um exemplo em Python e Go

UMA COMPARAÇÃO

Python: 73 linhas

Go: 138 linhas

```python
#!/usr/bin/env python3

import os, sys
import urllib.request
import threading

# URL UCD é a URL canônica do arquivo UnicodeData.txt mais atual
URL_UCD = 'http://www.unicode.org/Public/UNIDATA/UnicodeData.txt'


def analisar_linha(linha):
    campos = linha.split(';')
    código = int(campos[0], 16)
    nome = campos[1]
    palavras = set(nome.replace('-', ' ').split())
    if campos[10]:
        nome = '{} ({})'.format(nome, campos[10])
        palavras.update(campos[10].replace('-', ' ').split())
    return chr(código), nome, palavras


def listar(texto, consulta):
    consulta = set(consulta.replace('-', ' ').split())
    for linha in texto:
        runa, nome, palavras_nome = analisar_linha(linha)
        if consulta <= palavras_nome:
            try:
                print('U+{:04X}\t{}\t{}'.format(ord(runa), runa, nome))
            except UnicodeEncodeError:
                print('U+{:04X}\t\uFFFD\t{}'.format(ord(runa), nome))


def obter_caminho_UCD():
    caminho_UCD = os.environ.get('UCD_PATH')
    if caminho_UCD is None:
        caminho_UCD = os.path.join(os.environ['HOME'], "UnicodeData.txt")
    return caminho_UCD


def abrir_UCD(caminho):
    try:
        ucd = open(caminho)
    except FileNotFoundError:
        print('%s não encontrado\nbaixando %s' % (caminho, URL_UCD))
        feito = threading.Event()
        threading.Thread(target=baixar_UCD, args=(URL_UCD, caminho, feito)).start()
        progresso(feito)
        ucd = open(caminho)
    return ucd


def progresso(feito):
    while not feito.wait(.150):
        print('.', end='', flush=True)
    print()


def baixar_UCD(url, caminho, feito):
    with urllib.request.urlopen(url) as resposta:
        octetos = resposta.read()
    with open(caminho, 'wb') as arquivo:
        arquivo.write(octetos)
    feito.set()


def main():
    with abrir_UCD(obter_caminho_UCD()) as ucd:
        consulta = ' '.join(sys.argv[1:])
        listar(ucd, consulta.upper())


if __name__ == '__main__':
    main()
```

```go
package main

import (
  "bufio"
  "fmt"
  "io"
  "log"
  "net/http"
  "os"
  "os/user"
  "strconv"
  "strings"
  "time"
)

// URLUCD é a URL canônica do arquivo UnicodeData.txt mais atual
const URLUCD = "http://www.unicode.org/Public/UNIDATA/UnicodeData.txt"

func separar(s string) []string {
  s = strings.Replace(s, "-", " ", -1)
  return strings.Split(s, " ")
}

// AnalisarLinha devolve a runa, o nome e uma fatia de palavras
// que ocorrem no campo nome de uma linha do UnicodeData.txt
func AnalisarLinha(linha string) (rune, string, []string) {
  campos := strings.Split(linha, ";")
  código, _ := strconv.ParseInt(campos[0], 16, 32)
  nome := campos[1]
  palavras := separar(campos[1])
  if campos[10] != "" {
    nome += fmt.Sprintf(" (%s)", campos[10])
    for _, palavra := range separar(campos[10]) {
      if !contém(palavras, palavra) {
        palavras = append(palavras, palavra)
      }
    }
  }
  return rune(código), nome, palavras
}

func contém(fatia []string, procurado string) bool {
  for _, item := range fatia {
    if item == procurado {
      return true
    }
  }
  return false
}

func contémTodas(fatia []string, procurados []string) bool {
  for _, procurado := range procurados {
    if !contém(fatia, procurado) {
      return false
    }
  }
  return true
}

// Listar exibe na saída padrão o código, a runa e o nome dos
// caracteres Unicode onde ocorrem as palavras da consulta.
func Listar(texto io.Reader, consulta string) {
  termos := separar(consulta)
  varredor := bufio.NewScanner(texto)
  for varredor.Scan() {
    linha := varredor.Text()
    if strings.TrimSpace(linha) == "" {
      continue
    }
    runa, nome, palavrasNome := AnalisarLinha(linha)
    if contémTodas(palavrasNome, termos) {
      fmt.Printf("U+%04X\t%[1]c\t%s\n", runa, nome)
    }
  }
}

func check(e error) {
  if e != nil {
    panic(e)
  }
}

func obterCaminhoUCD() string {
  caminhoUCD := os.Getenv("UCD_PATH")
  if caminhoUCD == "" {
    usuário, err := user.Current()
    check(err)
    caminhoUCD = usuário.HomeDir + "/UnicodeData.txt"
  }
  return caminhoUCD
}

func progresso(feito <-chan bool) {
  for {
    select {
    case <-feito:
      fmt.Println()
      return
    default:
      fmt.Print(".")
      time.Sleep(150 * time.Millisecond)
    }
  }
}

func baixarUCD(url, caminho string, feito chan<- bool) {
  resposta, err := http.Get(url)
  check(err)
  defer resposta.Body.Close()
  arquivo, err := os.Create(caminho)
  check(err)
  defer arquivo.Close()
  _, err = io.Copy(arquivo, resposta.Body)
  check(err)
  feito <- true
}

func abrirUCD(caminho string) (*os.File, error) {
  ucd, err := os.Open(caminho)
  if os.IsNotExist(err) {
    fmt.Printf("%s não encontrado\nbaixando %s\n", caminho, URLUCD)
    feito := make(chan bool)
    go baixarUCD(URLUCD, caminho, feito)
    progresso(feito)
    ucd, err = os.Open(caminho)
  }
  return ucd, err
}

func main() {
  ucd, err := abrirUCD(obterCaminhoUCD())
  if err != nil {
    log.Fatal(err.Error())
  }
  defer func() { ucd.Close() }()
  consulta := strings.Join(os.Args[1:], " ")
  Listar(ucd, strings.ToUpper(consulta))
}
```

ThoughtWorks®

CONCLUSÃO

A melhor forma de prever o futuro é construí-lo (Alan Kay)

PORQUE GO É UM SUCESSO

Finalmente uma linguagem moderna que compila para **binários** em código de máquina de **alto desempenho**.

Design pragmático:

- Sintaxe simples, previsível e razoavelmente expressiva
- Compilação muito rápida
- Sistema de tipos simples e adequado
- Ferramental básico excelente
- “Pilhas incluídas”

GO E O ZEN DO PYTHON

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

GO E O ZEN DO PYTHON ✔

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit. ✔
Simple is better than complex. ✔
Complex is better than complicated. ✔
Flat is better than nested. ✔
Sparse is better than dense.
Readability counts. ✔
Special cases aren't special enough to break the rules. ✔
Although practicality beats purity. ✔
Errors should never pass silently. ?
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess. ✔
There should be one-- and preferably only one --obvious way to do it. ✔
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now. ✔
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those! ✔
```

O FUTURO DE GO

Go deve continuar sua trajetória de sucesso e será uma das 10+ do Tiobe Index em 2020*.

- Crescimento movido por sistemas distribuídos e **estratégia de plataformas**
- Gerenciamento de dependências na próxima versão (**go dep**)
- Go para **Android**: além dos experimentos
- **Generics?!?**

* Segundo minha bola de cristal

MUITO GRATO

Fale comigo:

Luciano Ramalho

Twitter: @ramalhoorg

luciano.ramalho@thoughtworks.com

ThoughtWorks®