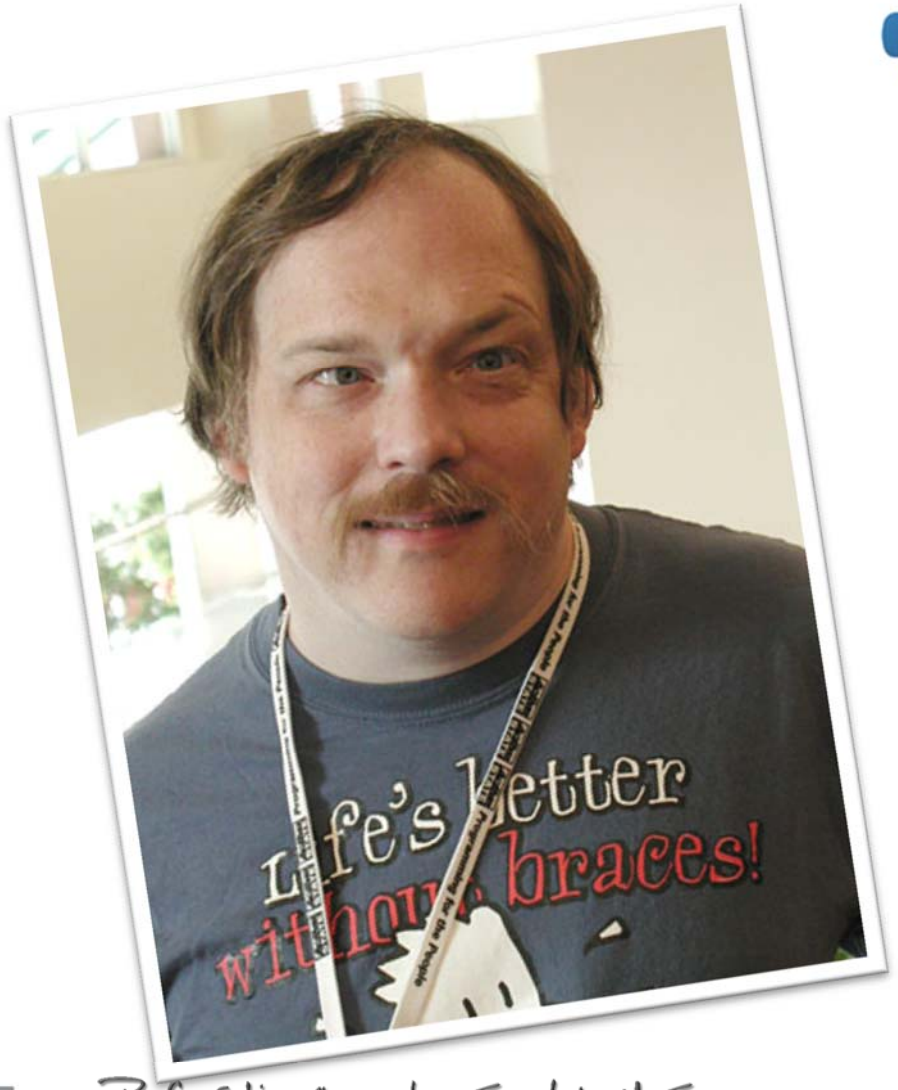

PYTHON
TRAINING
NO.1
ed. #4
UTFPR
Prof. Fabio Alexandre Spanhol, M.Sc.

Python?

- é uma linguagem *interpretada*, *interativa*, *funcional*, *orientada a objetos*, *dinamicamente tipada* e com *gerenciamento automático de memória*
  - Similar, em certos aspectos, a *Perl*, *Ruby*, *Scheme*, *Smalltalk* e *Tcl*

O que Dizem Sobre Python...



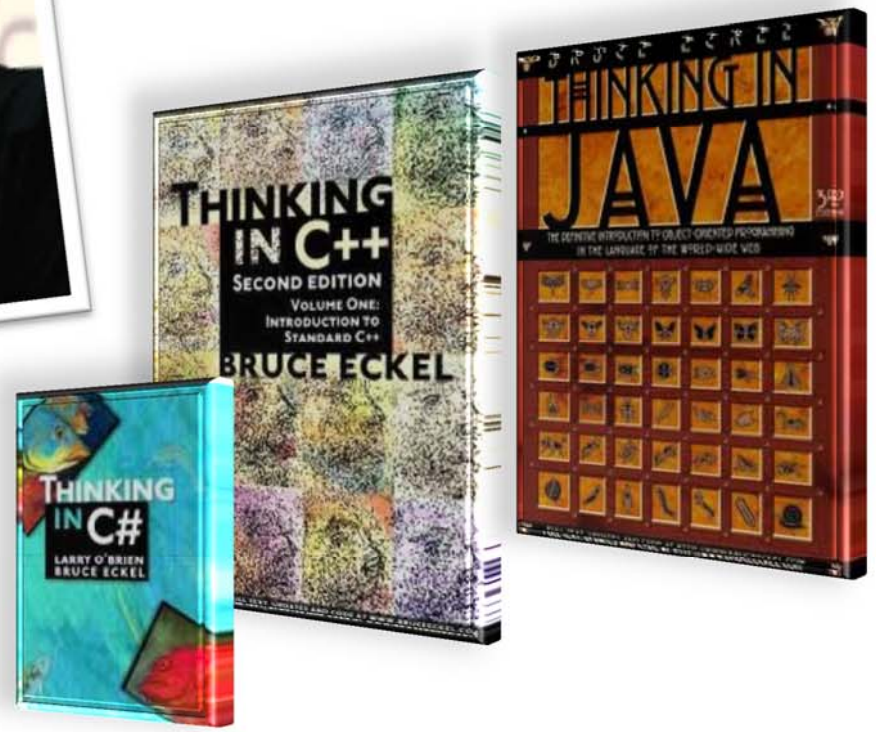
Entre todas as linguagens que aprendi, Python é a que menos interfere entre mim e o problema. É a mais efetiva para traduzir pensamentos em ações.

Eric Raymond

O que Dizem Sobre Python...

Life is Better Without Braces.

Bruce Eckel



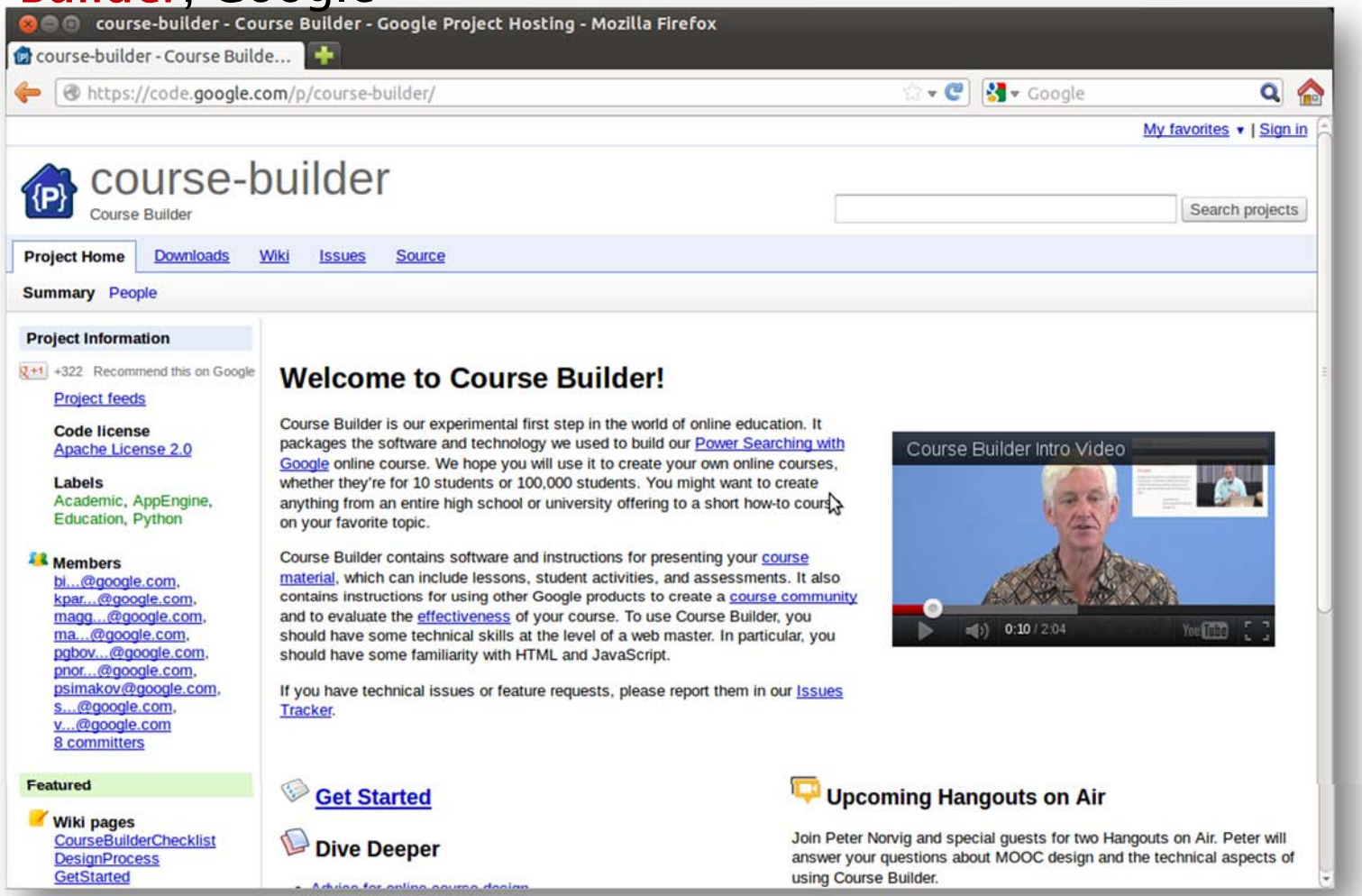
O que Dizem Sobre Python...

Python has been an important part of Google since the beginning, and remains so as the system grows and evolves. Today dozens of Google engineers use Python, and we're looking for more people with skills in this language.

Peter Norvik

Projeto saindo do forno...

Course-Builder, Google



E Quem criou?

Quem é Guido van Rossum?

Guido van Rossum é referido pela comunidade python, desde 1995, como *Benevolent Dictator For Life* (BDFL)

Desde dezembro de 2005 Guido trabalha na **Google**, dedicando 50% do seu tempo com a linguagem

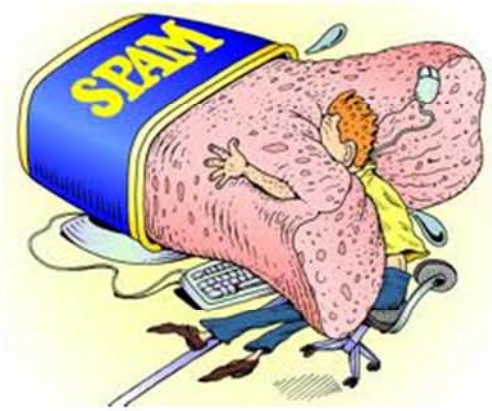
- www.python.org/~guido
- neopythonic.blogspot.com/

Mas, o nome Python...

Não veio da cobra!

"*Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose* ***Python*** *as a working title for the project, being in a slightly irreverent mood (and a big fan of* ***Monty Python´s Flying Circus****)*".

INTERCLÚDIO: PYTHONESQUE!!



Quanto custa?

NADA!

Python foi desenvolvido como um projeto *open source*, sem fins lucrativos e gerenciado pela PSF (*Python Software Foundation*)

python powered

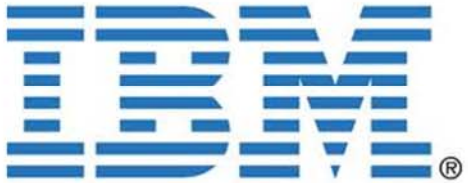
Onde pegar?

- Para obter o interpretador Python acessar *http://www.python.org/download/*
- Versões
  - `Python 2.7.3` foi liberada em 9 de abril de 2012
  - `Python 3.3.0*` foi liberada em 29 de setembro de 2012

Quem usa?

http://www.python.org/about/success/

- Google
- NASA
- Yahoo
- InfoSeek
- MCI Worldcom
- IBM
- Higway
- Industrial Light and Magic
- AstraZeneca
- Honeywell
- ...



gentoo linux

E no Brasil?

- Governo Federal
- Petrobras
- Serpro
- Embratel
- Globo.com
- StarOne
- Conectiva
- CPqD
- Async
- Haxent
- UTFPR
- ...

Apoio a Laboratórios

Alguém mais?

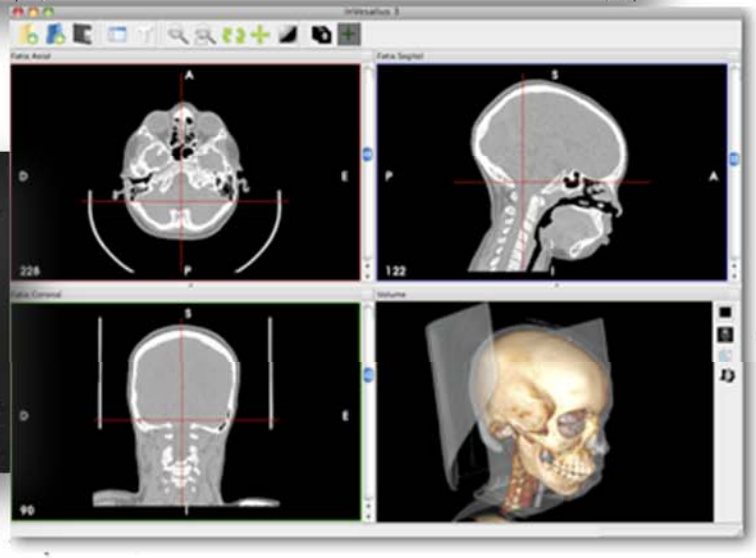
- Vários projetos da comunidade *Software Livre*
  - Blender
  - LibreOffice
  - Zope/Plone
  - MoinMoin
  - Mailman
  - BitTorrent
  - Chandler
  - Gimp
  - Plone
  - Django
  - OpenERP
  - inVesalius

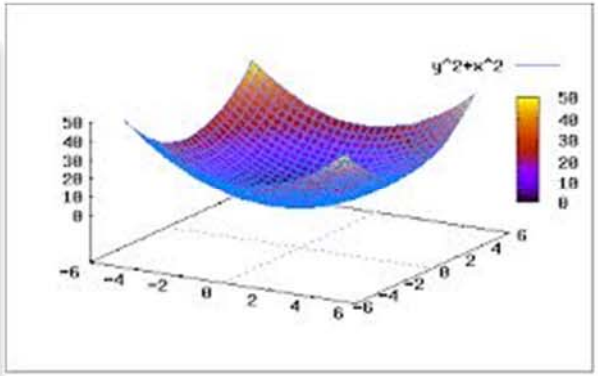


Onde aplico?

- Web e Internet
- Database
- GUIs
- Científico e processamento numérico
- Educação
- Programação de rede
- Construção e teste de software
- Desenvolvimento de jogos e renderização 3D
- Onde mais sua imaginação mandar!

APLICAÇÕES!

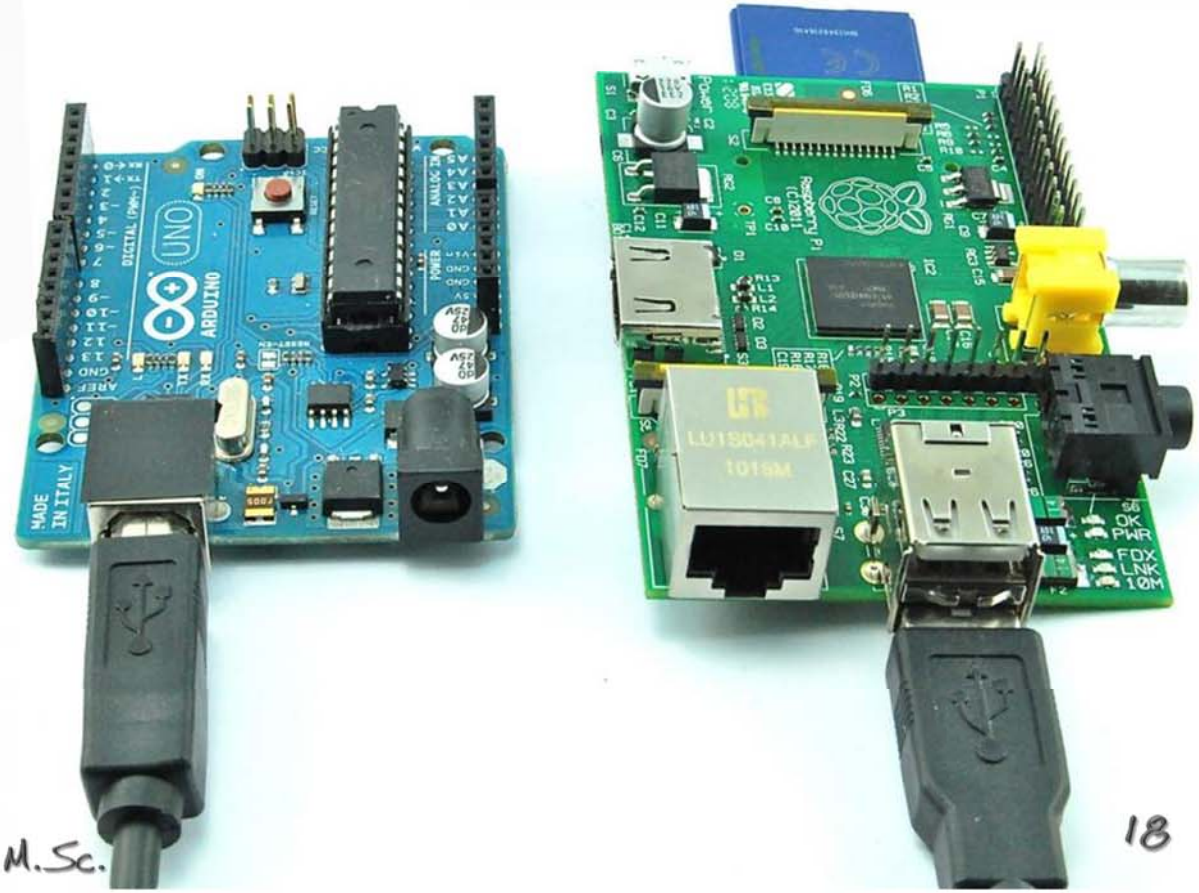




SymPy

Aplicações!

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

import serial
import time

# /dev/ttyACM0 = Arduino Uno on Linux
arduino = serial.Serial('/dev/ttyACM0', 9600)
time.sleep(2) #waiting the initialization...

arduino.write('H') #turns LED *on*
time.sleep(3) #zzz

arduino.write('L') #turns LED *off*
time.sleep(3) #zzz

arduino.close() #let's say goodbye
```



Python é Multiplataforma!

- Unix: HP-UX, Solaris ...
- Linux (freqüentemente pré-instalada)
- Mac OSX (sempre pré-instalada)
- Windows: 9x, ME, 2K, XP, Vista, Seven (ctypes, win32all)
- Apple iPhone
- Google Android

Python é portável!

Compilação Híbrida

- como Java, programas Python são compilados, porém para uma *linguagem intermediária*, destinada a um *interpretador*
  - Isola Python de muitas das excentricidades das máquinas reais na qual ele roda
  - provê um *nível de portabilidade*

Python é interoperável!

Prazer em lhe conhecer!

- Cython
  - onde existir um compilador C ISO/IEC 9899:1990
- Jython
  - máquina virtual Java
- PyPy
  - Python implementado em Python
- Python for .NET
  - [Brian Lloyd], IronPython (da M$)
- Python for Delphi
- LunaticPython
  - interoperando com Lua
- Ruby/Python
  - em Ruby importar módulos Python)

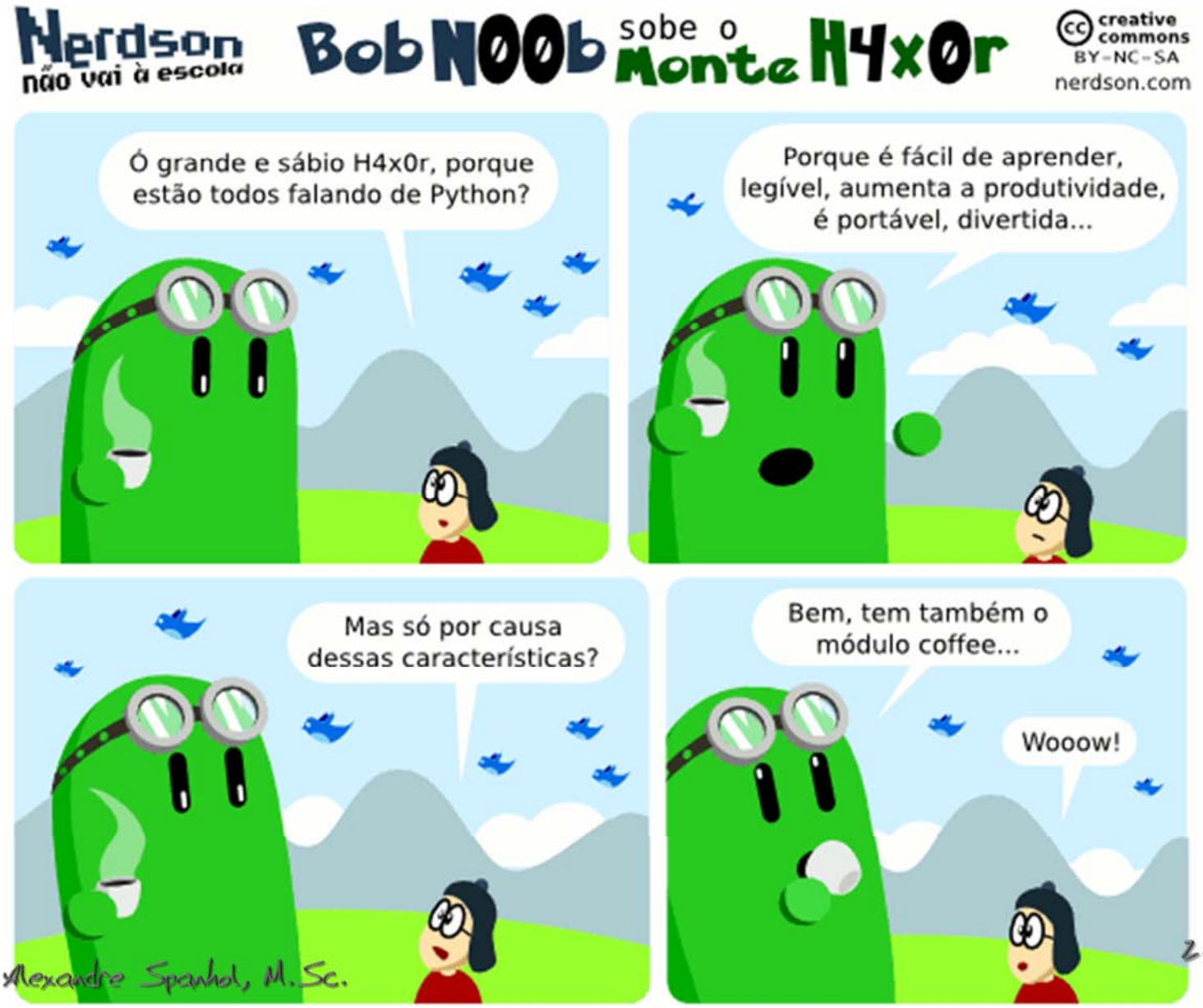
Resumindo, Python é...

- De altíssimo nível e poderosa
- Elegante, com sintaxe simples e concisa
- Fácil de aprender, "cabe" no seu cérebro !
- Multiparadigma
  - Funcional, Procedural e Orientado a Objetos
- Possui suporte nativo a estruturas de dados complexas

Python possui "baterias incluídas"!

- Extensa biblioteca padrão
- Um módulo para o que você precisar
- `sys`, `random`, `re`, `datetime`, `calendar`, `csv`, `os`, `webbrowser`, `string`, `urlib`, `Tkinter`, `codecs`, `pickle`, `types`, `array`, `mutex`, `queue`, `zlib`, `gzip`, `md5`, `thread`, `socket`, `email`, `xml.dom`, `locale`, etc.

Mas nem só de código vive o programador!



E LEMBRE-SE...

- Código é muito mais lido que escrito!
  - *Readability importa!* (muito)
    - Python te ajuda a escrever *código mais legível*

Agora, mãos na massa!

Básico

- Python é *case sensitive*
- Linhas são delimitadas por *enter*
- *Tipagem é forte e dinâmica*
  - Não há declaração de variáveis como em C, Java, Pascal, etc.
- Você pode utilizar o interpretador em modo interativo ou seu IDE/editor favoritos

Modo Interativo

IDLE File Edit Shell Debug Window Help

Python Shell

```
Python 3.1 (r31:73578, Jun 27 2009, 21:49:46)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for more information.
```

Python Shell

File Edit Shell Debug Options Windows Help

```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.1600
 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__', '__package__']
>>>
```

C:\Windows\system32\cmd.exe - python

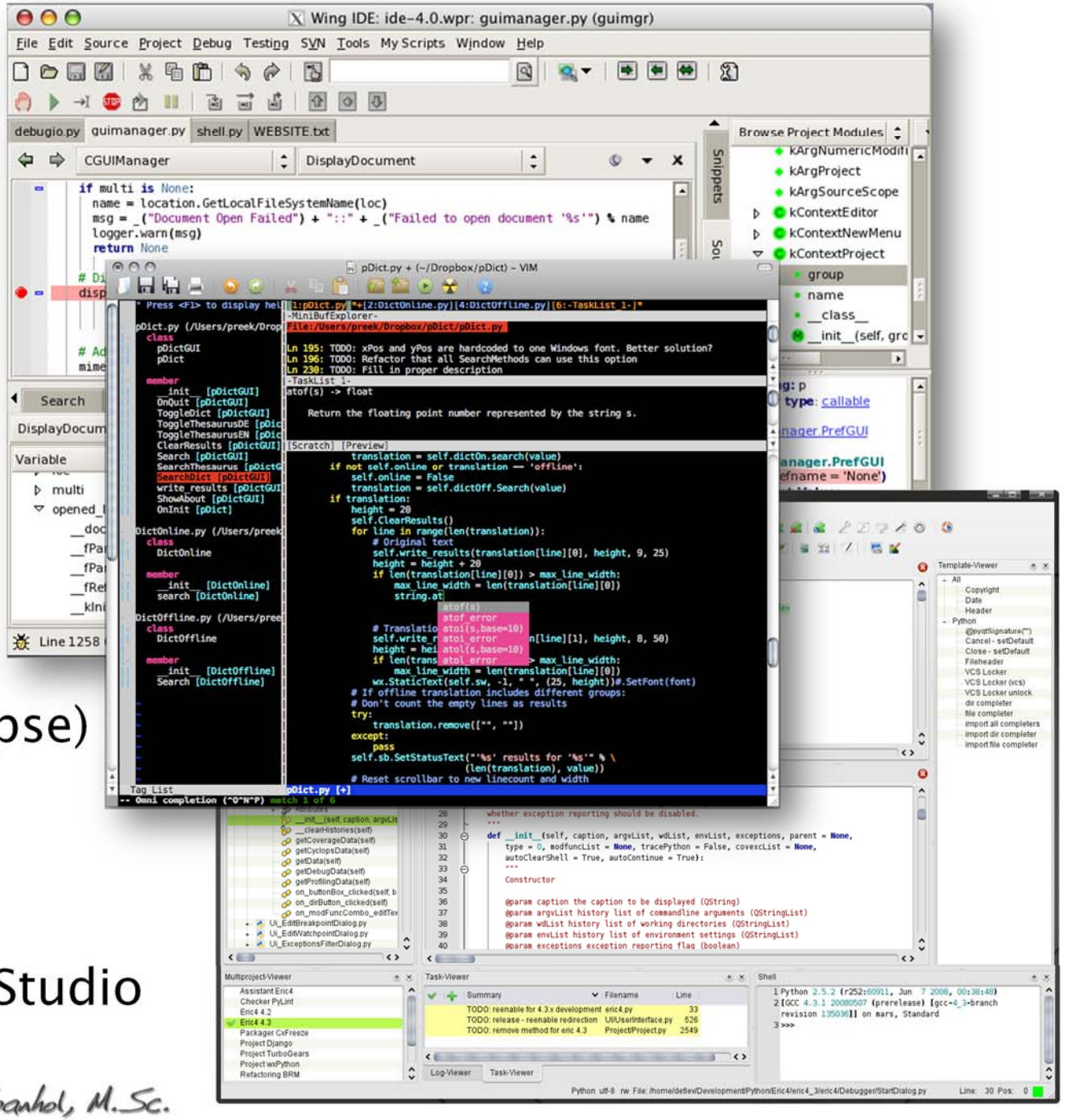
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\spanhol>python
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.1600 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> dir()
['__bui
>>>
```

spanhol@aragorn: ~

```
spanhol@aragorn:~$ python3
Python 3.2.3 (default, May  3 2012, 15:54:42)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
>>>
```

IDE

- Inúmeros
  - Geany
  - Vim
  - Aptana
  - Komodo
  - NetBeans
  - PyCharm
  - PyDev (Eclipse)
  - Wing IDE
  - Pyshield
  - Spyder
  - MS-Visual Studio
  - Etc.



Prof. Fabio Alexandre Spanhol, M.Sc.

UTFPR

IDE: Sugestão

Ninja IDE (*Ninja Is Not Just Another IDE*)

- Multiplataforma
- Editor de código poderoso
- Gerenciamento de Projetos
- Plugins
  - **http://ninja-ide.org/**


OPEN YOUR MIND!
USE FREE SOFTWARE!

Variáveis

- Em Python, *variáveis* são *referências a objetos*
  - não guardam os objetos em si
  - não têm tipo, mas os objetos aos quais elas se referem têm tipo
  - São criadas dinamicamente
- Uma variável não pode ser utilizada em uma expressão sem ter sido inicializada
  - não existe “criação automática” de variáveis

Variáveis: Atribuição

- Variáveis *não são "caixas" com valores*
  - *Variáveis são "rótulos" colados em objetos*
- São criadas pela atribuição
  - Operador =


```
>>> a=10
>>> b="5"
>>> a="3"
>>> a,b=b,a
>>> a
'5'
>>> b
'3'
>>> x=y=z=0.0
>>> x=c
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    x=c
NameError: name 'c' is not defined
```

Variáveis: "Desatribuição"

- Atribuição **não gera** uma cópia do objeto
  - Uma referência (variável) pode ser liberada
  - Usando **`del`**
    - Não existindo mais referências a um objeto, ele é removido da memória (*garbage collector*)

```
>>> x = 123
>>> x
123
>>> del x
>>> x
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    x
NameError: name 'x' is not defined
```

Execução

- Normalmente para programas maiores você irá codificar arquivos (*scripts*) em um IDE
  - extensão **py**
- O *script* pode ser executado diretamente da linha de comando
  - **python oi.py**
- Ou carregado para o interpretador interativo
  - **>>> import oi**

Módulos

- Módulos são arquivos que contêm qualquer estrutura python (classes, funções, variáveis, etc.) e podem ser *importados* nos programas
  - Quando importado pela primeira vez o módulo é compilado e um arquivo .pyc ou .pyo é gerado
- Um módulo é um objeto *singleton*
  - Apenas uma instância é carregada em memória
  - torna-se disponível globalmente para o programa que fez a importação

Módulos

- Funções importantes são disponibilizadas em módulos da biblioteca padrão
  - Ex.: módulo **`math`** tem funções como **`sin`**, **`cos`**, **`exp`** e outras
- Um módulo pode conter não só funções, mas também variáveis ou classes
  - Ex.: **`math`** define a constante **`pi`**

- Os módulos são localizados pelo interpretador pela lista de diretórios em `PYTHONPATH` (`sys.path`), que inclui o diretório atual

Módulos: Importando

- Para usar os elementos de um módulo, comando **import**
  - **import *modulo***
  - **from modulo import *nome*$_1$,...,*nome*$_n$**
  - **from modulo import ***
- A importação ocorre apenas uma vez!
  - Para carregar um módulo novamente (que tenha sido alterado) pode-se usar a função **reload**
    - No python > 3.x é **imp.reload**

Módulos: Importando

Exemplos

```
>>> import math
>>> sin(30)
Traceback (most recent call last):
  File "<pyshell#79>", line 1, in <module>
    sin(30)
NameError: name 'sin' is not defined
>>> math.sin(30)
-0.988031624092861 8
>>> from math import sin
>>> sin(30)
-0.988031624092861 8
>>> sin(radians(30))
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    sin(radians(30))
NameError: name 'radians' is not defined
>>> from math import*
>>> sin(radians(30))
0.4999999999999994
```

Módulo Principal

Se um programa pode ser executado isolado ou importado dentro de outro, como distinguir as duas situações?

- O módulo principal de um programa tem a variável **__name__** contendo **“__main__”**

- Para executar um código apenas se o módulo for o principal e não quando ele for importado

```
if __name__ == "__main__":
    #código a ser executado
```

Socorro! Ajuda!

```
>>> import math
>>> dir()
['__builtins__', '__doc__', '__name__', 'math']
>>>
>>> print math
<module 'math' (built-in)>
>>>
>>> dir(math)
['__doc__', '__name__', 'acos', 'asin', 'atan', 'atan2', 'ceil', 'cos', 'cosh',
'degrees', 'e', 'exp', 'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log'
, 'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh']
>>> help(math.pow)
Help on built-in function pow in module math:

pow(...)
    pow(x,y)

    Return x**y (x to the power of y).
```

```
>>> help(math)
Help on built-in module math:

NAME
    math

FILE
    (built-in)

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
```

Veja as PEPs!

Python Enhancement Proposal

- São documentos padronizados da comunidade Python
  - Uma PEP propõe um padrão, melhoria, funcionalidade, estrutura, explicações sobre funcionalidades, etc.

Veja a PEP #0

- http://www.python.org/dev/peps/pep-0000/

Blocos

- Um bloco inicia com **:**

- A estrutura dos blocos é definida pela indentação
  - A *PEP #8 – Style Guide for Python Code* – sugere 4 espaços
    - Seja consistente
    - Não misturar tabulações com espaços!!!

Blocos

```
1  for i in range(25):
2      if i%3 == 0:
3          print (i)
4          if i%5 == 0:
5              print ("Bingo!")
6              print ("#"*5)
7      print ("-"*i)
8  print ("FIM")
```

Comentários

- A partir do caractere #, o interpretador ignora o restante do código até o final da linha
  - Isso **não vale** dentro de strings
- Para várias linhas use aspas triplas (*docstring*)
  - Documentar funções, classes, módulos, etc.

```
1  def function(a, b):
2      """Do X and return a list."""
3      pass #do nothing
4
```

Números Inteiros

- Tipos básicos
  - **int** = normalmente 32 bits
  - **long** = limitado ao tamanho da memória
    - Há promoção automática de *int* para *long*
    - Divisão de inteiros em python < 3 resulta sempre *int*

```
>>> 568598989587113464651657987987156540000000000000**2
323304810959486366203142158661816361022953980726778177395047994464771600000000000000000000000000
>>> x = 10
>>> y = 568598989587113464651657987987156540000000000000**10
>>> y
353232647214080854803365995021719497772432721167815201669888470790347021557385266736154111590644315426
164363694298386910830633397619015653617178156478204837423398799737071999921745287034623067049511278543
321772299164789715890618452809778537913140785398013921668103962421206460624250690747176423987114482707
720465897599892294850586426512534917784576000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
>>> z = x + y
>>> z
353232647214080854803365995021719497772432721167815201669888470790347021557385266736154111590644315426
164363694298386910830633397619015653617178156478204837423398799737071999921745287034623067049511278543
321772299164789715890618452809778537913140785398013921668103962421206460624250690747176423987114482707
720465897599892294850586426512534917784576000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000010
```

Flutuantes e Complexos

- Também são tipos numéricos básicos
  - **float** = ponto flutuante de 32 bits
  - **complex**= para números complexos

```
>>> f = 13.1415926535
>>> f
13.1415926535
>>> c = 2 + 3j
>>> c
(2+3j)
>>> c.real
2.0
>>> c.imag
3.0
```

Convertendo Números

- Construtores ou funções de conversão
  - int (n)
  - float (n)
  - complex (n)
  - abs (n)

```
>>> int(3.4)
3
>>> int("101",2)
5
>>> abs(-12)
12
>>> complex(3)
(3+0j)
>>> complex(2,7)
(2+7j)
>>> int("123456789")
123456789
```

Números: Operadores

- Básicos
  - +, -, *, /, **
- Inteiros
  - %, //
- Bit a Bit
  - &, |, ^, ~, >>, <<

```
>>> 5/2
2.5
>>> 5//2
2
>>> 5%2
1
>>> 5**16
152587890625
>>> 5&5
5
>>> 5&4
4
>>> 5>>2
1
>>> 5<<2
20
```

Números: Funções Built-In

Módulo *math* e outros

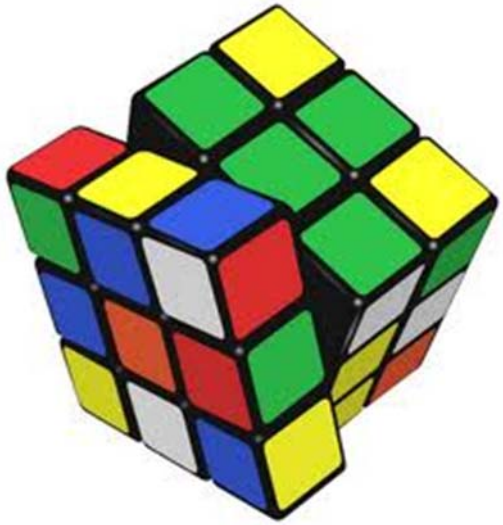
```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', 'acos', 'acosh',
 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos'
, 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'fac
torial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isfinit
e', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'lo
g2', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 't
anh', 'trunc']
>>> math.pi
3.141592653589793
>>> math.pow(2,30)
1073741824.0
>>> math.ceil(45.18)
46
>>> math.sqrt(36)
6.0
```

O "Não-valor" None

- Representa o valor nulo
  - Equivalente a **False** em operações booleanas
- Aplicações típicas
  - Valor *default* em parâmetros de funções
  - Valor de retorno de funções que será descartado
    - Como *void* de C/C++

Valores Lógicos

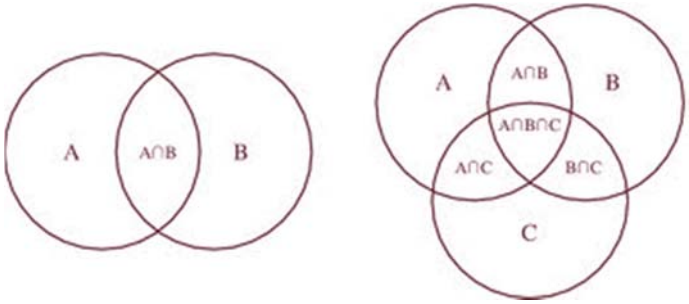
- Constantes *True* e *False*
  - Ocorre conversão automática entre tipos
  - Conversão explícita pode ser feita com **`bool(x)`**



```
>>> p = True
>>> p
True
>>> p = bool(0)
>>> p
False
>>> bool('')
False
>>> bool('0')
True
>>> bool(None)
False
>>> bool(False)
False
```

Valores Lógicos: Operadores

- Relacionais
  - `>`, `<`, `>=`, `<=`, `==`, `!=`, `is`, `is not`
  - Sempre retornam um *bool*
- Lógicos
  - **`and`**, **`or`**, **`not`**



Strings

- **str** – Sequência de bytes, com a acentuação dependente do *encoding*
  - Delimitadas por aspas, apóstrofo ou três aspas ou três apóstrofos

```
>>> s="string"
>>> s2='outra string'
>>> s
'string'
>>> s2
'outra string'
>>> s3="""string contendo
            varias
          linhas"""
>>> s3
'string contendo\n            varias\n          linhas'
>>> s4="string também com acentuação"
>>> s4
'string também com acentuação'
```

Enquanto isso, abaixo do equador...

- Codificações
  - **iso-8859-1**: padrão ISO Latin-1
  - **iso-8859-15**: idem, com símbolo € (Euro)
  - **cp1252**: MS-Windows codepage 1252
    - ISO Latin-1 aumentado com caracteres usados em editoração eletrônica (‘’ “” •)
  - **utf-8**: Unicode codificado em 8 bits compatível com ASCII até o código 127
    - utiliza 2 bytes para caracteres não-ASCII
    - padrão recomendado pelo W3C e está sendo adotado pela maioria dos sistemas

Codificação e scripts

- Constantes *str* ou *unicode* são interpretadas segundo a codificação declarada num comentário especial no início do arquivo .py

```
#!/usr/bin/env python
# coding: utf-8
```

- Nós (brasileiros) frequentemente lidamos com textos não ASCII!

Strings: Interpolação

Use caracteres de formatação, como na *printf* da linguagem C

%s, %d, %f são os mais comuns

```
>>> m='US$'
>>> r=2.028
>>> s='O %s está cotado a R$ %5.2f!'
>>> print(s % (m,r))
O US$ está cotado a R$  2.03!
```

Descobrindo os tipos!

Usando *type*

```
>>> x = 2
>>> type(x)
<class 'int'>
>>> x = 'dois'
>>> type(x)
<class 'str'>
>>> x = None
>>> type(x)
<class 'NoneType'>
```

Descobrindo os tipos!

Usando *type*

```
>>> x=2.3100574658
>>> type(x)
<class 'float'>
>>> f=round
>>> type(f)
<class 'builtin_function_or_method'>
>>> f(x)
2
>>> f(x,3)
2.31
```

Tuplas

- Tuplas são sequências *imutáveis*
  - **não é** possível modificar as referências contidas na tupla
  - Tuplas constantes são representadas como sequências de itens entre parênteses

```
>>> t1=1,1,2,3,5
>>> t1
(1, 1, 2, 3, 5)
>>> type(t1)
<class 'tuple'>
>>> t2=1,"bla",2.78
>>> t2
(1, 'bla', 2.78)
```

```
>>> t1[1]=0
Traceback (most recent call last):
  File "<pyshell#71>", line 1, in <module>
    t1[1]=0
TypeError: 'tuple' object does not support item assignment
```

Tuplas

- Tuplas são sequências *imutáveis*
  - Tuplas constantes são representadas como sequências de itens entre parênteses

- *CUIDADO: em certos contextos os parênteses ao redor das tuplas podem ser omitidos!*

```
>>> a=5
>>> b=3
>>> a,b=b,a
>>> a
3
>>> b
5
```

Listas

- Listas são coleções de itens heterogêneos que podem ser acessados sequencialmente ou indexados
  - São *mutáveis*
  - Constantes lista delimitadas por colchetes []

```
>>> l=['Fabio',36,False,89.5,186]
>>> l[0]
'Fabio'
>>> l[3]
89.5
>>> l[3]=89.0
>>> l
['Fabio', 36, False, 89.0, 186]
>>> l[5]
Traceback (most recent call last):
  File "<pyshell#82>", line 1, in <module>
    l[5]
IndexError: list index out of range
```

Gerando Valores com range

- Facilmente você pode criar uma lista numérica com
  - `range([inicio,] fim[, passo])`
  - Retorna uma sequência numérica conforme os argumentos dados
  - Normalmente usada em laços *for*
    - *Na verdade retorna um *iterator*, em Python >= 3.x

```
>>> for i in range(1,20,3): print (i)

1
4
7
10
13
16
19
```

Generalizando: Sequências

- Os tipos *string*, *lista*, *tupla*, *buffer* são sequências
  - Coleção ordenada e iterável de itens

- Operações
  - `s[i]` = acessa um item
  - `s[-i]` = acessa um item pelo final
  - `s+z` = concatena
  - `s*n` = produz n cópias de **s** concatenadas
  - `e in s` = elemento e está em s?
  - `e not in s` = elemento e não está em s?

Sequências: Fatiamento

- Operações de *slicing*
  - `s[a:b]` cópia de **a** (inclusive) até **b** (exclusive)
  - `s[a:]` cópia a partir de **a** (inclusive)
  - `s[:b]` cópia até **b** (exclusive)
  - `s[:]` cópia total de **s**
  - `s[a:b:n]` cópia de **n** em **n** itens

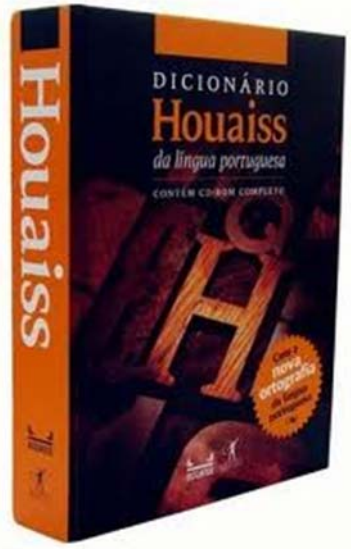
- Atribuição
  - `s[2:5] = [6,3,2,0]`
  - Aplicável somente em sequências mutáveis

Dicionários

Dicionários são estruturas de dados que implementam *mapeamentos*

- coleções de pares *chave:valor* que podem ser recuperados pela chave
- A chave pode ser qualquer imutável

```
>>> telefones={}
>>> type(telefones)
<class 'dict'>
>>> telefones['Fabio']='84052958'
>>> telefones
{'Fabio': '84052958'}
>>> telefones['Fabio']
'84052958'
```



E/S Básica

Entrada com `input`

```
>>> nome=input("Qual o seu nome?")
Qual o seu nome?Fabio
>>> nome
'Fabio'
>>> idade=int(input("Qual sua idade?"))
Qual sua idade?36
>>> idade
36
```

Saída com `print`

```
>>> print(nome)
Fabio
>>> print("Olá, %s. Sua idade é %d anos!" % (nome,idade))
Olá, Fabio. Sua idade é 36 anos!
```

Decisão!

if

```
if (expressao):
    bloco
[elif (expressao):
    bloco]
[else:
    bloco]
```

Decisão!

exemplo

```
1  #coding: utf-8
2  n = int(input("Número entre 0 e 100?"))
3
4  if not 0 < n < 101:
5      print ("Número inválido.")
6  elif n % 2 == 0:
7      print ("Número Par")
8  else:
9      print ("Número Impar")
```

Loops!

for

```
for variavel in sequencia:
    bloco
[else:
    bloco]
```

while

```
while (expressao):
    bloco
[else:
    bloco]
```

Loops!

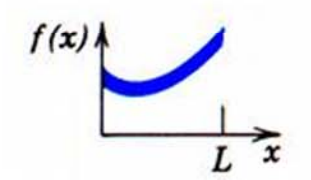
Exemplo1

```
1  # encoding: utf-8
2  cores=['verde','amarela','azul','branca']
3  for cor in cores:
4      print("%s\t é minha cor favorita" % cor)
5
```

Exemplo2

```
1  agenda = {"Fabio":84052968,
2            "Clau":84024792,
3            "Edson":9989977}
4
5  for nome,tel in agenda.items():
6      print ("%s - %d" % (nome.ljust(10),tel))
7
```

Funções



- Modularizam o código
  - Comando **`def`** inicia a definição de uma função
  - Comando **`return`** marca o fim da execução da função e define o resultado a ser devolvido
    - Pode ser *None*

Funções

Exemplo

```
1  #encoding: utf-8
2  def fibo(n):
3      """função que escreve n
4      termos da série de fibonacci"""
5      a,b,t=0,1,1
6      while t <= n:
7          print (b)
8          a, b = b, a + b
9          t+=1
10
11 if (__name__ =='__main__'):
12     x=int(input("Quantos termos?"))
13     fibo(x)
```

Funções

```
1  #encoding: utf-8
2  def fatorial(n):
3      if n==0:
4          return 1
5      else:
6          return n*fatorial(n-1)
7
8  def fibo(n):
9      if n==1 or n==2:
10         return 1
11     else:
12         return fibo(n-1) + fibo(n-2)
13
14 funcs=[fatorial,fibo]
15 for f in funcs:
16     print ([f(x) for x in range(1,10)])
```

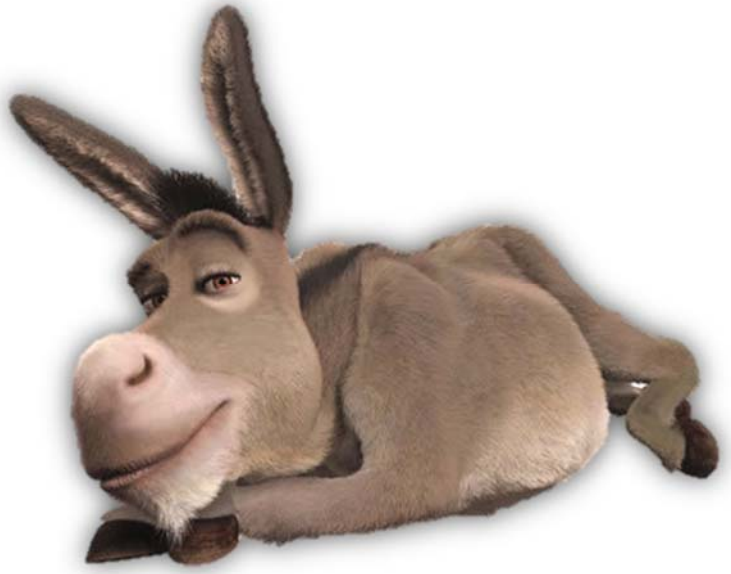
Funções: Parâmetros Default

```
1  # encoding: utf-8
2  def rgb2html(r=0,g=0,b=0):
3      """Converte RGB em #HHHHHH"""
4      return "#%02x%02x%02x" % (r,g,b)
5
6  def html2rgb(color="#000000"):
7      """Converte #HHHHHH em R,G,B"""
8      if color.startswith("#"):
9          color = color[1:]
10     r = int(color[:2],16)
11     g = int(color[2:4],16)
12     b = int(color[4:],16)
13     return r, g, b
14
15 if (__name__ == "__main__"):
16     print(rgb2html(255,205,42))
17     print(rgb2html(g=205,b=42,r=255))
18     print(html2rgb("#FFCD2A"))
19
```

Funções: Parâmetros Variáveis

Ei, não vamos complicar!

```
>>> def f(a=0, *b):
        c = 0
        for i in b:
            c += i
        c += a
        return c

>>> f()
0
>>> f(1,2,3)
6
>>> f(1,2,3,4,5,6,7,8,9)
45
```



Funções: Limites da Recursão

Estourar a pilha de execução é “fácil”

```
def fat(n):
    if n==0:
        return 1
    else:
        return (n*fat(n-1))
```

E aí, tentou **fat(1000)**?

Funções: Limites da Recursão

Mudando o tamanho da pilha de ativação

```
>>> sys.getrecursionlimit()
1000
>>> sys.setrecursionlimit(20000)
>>> fat(1000)
402387260077093773543702433923003985719374864210714632543799104299385123986290205920442084869694048
004799886101971960586316668729948085589013238296699445909974245040870737599188236277271887325197795
595099527612087497546249704360141827809464649629105639388743788648733711918104582578364784997701247
632889835955735432513185323958463075557409114262417474349347553428646576611667797396668820291207379
438537195882498081268678383745597317461360853795345242215865932019280908782973084313928444032812315
861103697680135730421616874760967587134831202547858932076716913244842623613141250878020800026168315
027341827977704784635868170164365024153691398281264810213092761244896359928705114964975419909342221
668325720808213331861168115536158365469840467089756029009505376164758477284218896796462449451607653
340819890138544248798495995331910172335555660213945039973628075013783761530712776192684903435262520
015888535147331611702103968175921510907788019393178114194545257223865541461062892187960223838971476
885062768629671466746975629112340824392081601537808898939645182632436716167621791689097799119037540
127462228998800519544441428201218736174599264295658174662830295557029902432415318161721046583203678
906117260158783520751516284225540265170483304226143974286933061690897968482590125458327168226458066
526769958652682272807075781391858178889652208164348344825993266043367660176999612831860788386150279
595513115655203609398818061213855860030143569452722420634463179746059468257310379008402443243846565
245014402821885252470935190620929023136493273497565513958720559654228749774011413346962715422845862
773875382304838656889764619273838149001407673104466402598994902222217659043399018860185665264850617
970235619389701786004081188972991831102117122984590164192106888438712185564612496079872290851929681
937238864261483965738229112312502418664935314397013742853192664987533721894069428143411852015801412
448280150513996942901534830776445690990731524332782882698646027898643211390835062170950025973898635
427719674282224875758676575234422020757363056949882508796892816275384886339690995982628095612145099
871701244516461260379029309120889086942028510640182154399457156805941872748998094254742173582401063
774045957417851608292301353580818400969963725242305608559037006242712434169090041536901059339838357
793941097002775347200000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Funções: Generators

- Funções que possuem a instrução **yeld** são *Generators*
  - Retornam objetos *iterators*
    - É mantida uma memória do último valor retornado

```
>>> def fgen(maximo):
        i = 0
        while i < maximo:
            yield i
            i += 1

>>> for k in fgen(5):
        print(k)


0
1
2
3
4
>>>
```

Funções: Generators

```
>>> print(sum(fgen(10)))
45
>>> l=fgen(8)
>>> l
<generator object fgen at 0x00000000034B15E8>
>>> list(fgen(8))
[0, 1, 2, 3, 4, 5, 6, 7]
>>>
```

Arquivos

- Classe **`file`** representam arquivos
  - Use a função **`open`** (ou construtor **`file()`**, são sinônimos)

- abrir arquivo binário para leitura
  - **`arq = file('imagem.png','rb')`**
- abrir arquivo texto para escrita
  - **`arq = open('log.txt','w')`**
- abrir arquivo para acrescentar (*append*)
  - **`arq = file('imagem.png','a')`**

Arquivos

- Cuidado: arquivo é uma sequência de bytes!
  - Deve ser interpretada por uma codificação de caracteres
  - Ex.: tentativa de ler um arquivo UTF-8 criado no MS-Windows
    - É assumida codificação padrão (do sistema): CP-1252

```
>>> f=open("idiomas.txt")
>>> for l in f.readlines():
        print(l)


Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    for l in f.readlines():
  File "c:\Python32\lib\encodings\cp1252.py", line 23, in decode
    return codecs.charmap_decode(input,self.errors,decoding_table)[0]
UnicodeDecodeError: 'charmap' codec can't decode byte 0x90 in position 93: chara
cter maps to <undefined>
>>>
```

Arquivos

- Boa prática: especificar a codificação na abertura!
  - O padrão é **dependente da plataforma**

```
>>> f=open("idiomas.txt",encoding="utf-8")
>>> for l in f.readlines():
        print(l)


Português-> Python: aprendendo a linguagem agora!

Chinês->    Python的：學習語言吧！

Russo->     Python: изучение языка прямо сейчас!

Eslovaco->  Python: učiť sa jazyk teraz!

Japonês->   Pythonは：今、言語を学ぶ！

Coreano->   파이썬 : 지금의 언어를 배우는!
>>>
```

Arquivos: Modo Binário

- Criando um `bitmap` de 2x2 pixels
  - Não entraremos em detalhes (*header*, *dib*, *data*, etc.) do formato bmp
  - **NÃO FAÇA isso em sã consciência!**
    - Existem bibliotecas especializadas para tratamento de imagens: `PIL`

```
>>> bmp=open("teste.bmp","wb")
>>> bmp.write(b"\x42\x4D\x46\x00\x00\x00\x00\x00\x00\x00\x36\x00\x00\x00\x28\x00
\x00\x00\x02\x00\x00\x00\x02\x00\x00\x00\x01\x00\x18\x00\x00\x00\x00\x00\x10\x00
\x00\x00\x13\x0B\x00\x00\x13\x0B\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\xFF\xFF\xFF\xFF\x00\x00\xFF\x00\x00\x00\xFF\x00\x00\x00")
70
>>> bmp.close()
```

    - Arquivo bmp de 70 bytes criado

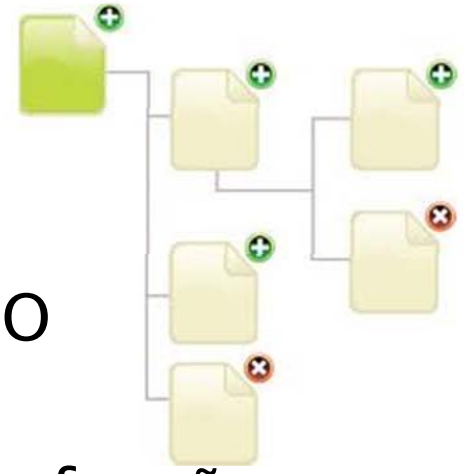
Arquivos: Modo Binário

- Alterando o nosso bmp de 2x2 pixels
  - **NÃO FAÇA isso em sã consciência!**
    - Existem bibliotecas especializadas para tratamento de imagens: PIL

```
>>> bmp=open("teste.bmp","r+b")
>>> bmp.seek(65)
65
>>> bmp.write(b"\x00\x00\xFF")
3
>>> bmp.close()
```

- Arquivo aberto para R/W
- Posicionado no byte 65
- Escritos 3 bytes

Arquivos



- Operações de E/S são realizadas pelo SO
- O módulo **`os`** possui diversas variáveis e funções que ajudam um programa Python adequar-se ao SO
  - **`os.getcwd()`** o diretório atual
  - **`os.chdir(`*`dir`*`)`** diretório atual para *dir*
  - **`os.sep`** caractere que separa componentes de um caminho (**`'/'`** para *Unix*, **`'\\'`** para *Windows*)
  - **`os.path.exists(`*`path`*`)`** diz se *path* é o nome de um arquivo existente

Arquivos

Módulo CSV (*Comma-Separated Values*)

```
1  """ Le um arquivo csv e mostra os campos na tela. """
2  import csv
3  f = csv.reader(open('agenda.csv'), delimiter=';')
4  print ('%s | %s | %s ' % ('nome'.ljust(30),
5                            'fone'.ljust(15),
6                            'email'.rjust(20)))
7  print ("-"*80)
8  for [nome,fone,email] in f:
9      print ('%s | %s | %s' % (nome.ljust(30),
10                              fone.ljust(15),
11                              email.rjust(20)))
12 print ("="*20)
13 print (f.line_num, 'linhas lidas\n')
```

Arquivos: Compressão

- Bibliotecas nativas para compressão
  - Baterias **gzip**, **bzip2**, **pkzip**, etc.

```
>>> import gzip
>>> with gzip.open("compactado.gz", mode="wb") as zf:
        zf.write("Numa toca no chão vivia um hobbit.".encode("utf-8"))
```

35

```
spanhol@aragorn: ~/curso

spanhol@aragorn:~/curso$ ls -lh
total 4,0K
-rw-rw-r-- 1 spanhol spanhol 66 Set 13 21:34 compactado.gz
spanhol@aragorn:~/curso$ gunzip compactado.gz
spanhol@aragorn:~/curso$ ls -lh
total 4,0K
-rw-rw-r-- 1 spanhol spanhol 35 Set 13 21:34 compactado
spanhol@aragorn:~/curso$ cat compactado
Numa toca no chão vivia um hobbit.spanhol@aragorn:~/curso$
```

POO... Uma pegada bem leve

- Lembra? Multiparadigma! Inclui *Orientação a Objetos*
  - Tudo em python é *objeto*. Mesmo!

```
>>> "objeto".upper()
'OBJETO'
>>> n = 2
>>> n.__pow__(10)
1024
>>> l = [1,1,2,3,5]
>>> l.append(8)
>>> l
[1, 1, 2, 3, 5, 8]
>>> def f():
        pass

>>> f.__sizeof__()
52
```

Criando uma classe

- Simplesmente defina a classe e use-a!
  - Classe (*class*) é um tipo de dados especial que define como construir objetos
  - Também armazena alguns dados que são compartilhados por todas as *instâncias* dessa classe
    - Instâncias são objetos criados segundo a definição da classe
  - Python **não separa** a interface da implementação da classe

Classe e Métodos

Não se desespere!

- para os métodos, defina funções dentro do escopo de *class*

```
class Person(object):
    """Classe que representa uma
       Pessoa """
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def get_age(self):
        return self.age
```

Criação de Instâncias

- A função **__init__** serve como construtor
- Nos métodos, o parâmetro **self** referencia a instância atual
  - NÃO É passado explicitamente

```
>>> p1 = Person("Fabio",35)
>>> p1.age
35
>>> dir(p1)
['__class__', '__delattr__', '__dict__', '__doc__', '__eq__', '__format__', '__g
e__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__',
'__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '_
_setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'age',
'get_age', 'name']
>>> p1.get_age()
35
```

Remoção de Instâncias

- Não é necessário liberar objetos explicitamente

- Esqueça *free* ou *delete*!
  - Coletor automático de lixo, lembra?

Tem (muito) mais POO...

- Herança, métodos de classe, métodos estáticos, atributos “privados”, propriedades, etc.
  - Mas, ...

I'll be back in #2!

E Programação GUI?

- TCL-TK é parte integrante, via **`tkinter`**
- Diversos outros *bindings* de *frameworks*
  - wxWidgets
  - QT
  - GTK
  - etc.

wxWidgets
Cross-Platform GUI Toolkit

GUI: Só um TIRA-GOSTO...

```
#coding: utf-8
from tkinter import messagebox
from tkinter import *

class OlaMundo:
  def __init__(self, master):
    self.frm = Frame(master)
    self.frm.pack()

    #criação do label
    self.lbl = Label(self.frm, text="Exemplo",
                     fg="#%02x%02x%02x" %(128, 192, 200),
                     font=("Helvetica", 16))
    self.lbl.pack()
    #botões
    self.bt = Button(self.frm, text="Sair", bg="black", fg="white",
                     height=2, width=10, command=self.exit)
    self.bt.pack(side=LEFT)

    self.bt2 = Button(self.frm, text="Olá mundo",
                      height=2, width=10, command=self.message)
    self.bt2.pack(side=LEFT)

  def exit(self):
    if messagebox.askyesno("Confirmação", "Sair?"):
      self.frm.quit()

  def message(self):
    messagebox.showinfo ( "Info","Olá Mundo!!!!")

root = Tk()                          #cria um widget janela, decorada pelo gerenciador de janelas
root.title("Exemplo de Tkinter") #seta o título da janela
root.geometry("300x100+400+300") #seta dimensões da janela e posição
root.resizable(width=TRUE, height=FALSE) #não permite redimensionar a altura
app = OlaMundo(root)
root.mainloop()                      #loop para processar eventos
```

Um pouco de "Magia Negra"

"List comprehensions"

- Produz uma lista a partir de qualquer objeto iterável
  - Sintaxe inspirada em Haskell

```
>>> qtd = [2,3,2]
>>> ingredientes=['cenouras','vagens','rabanetes']
>>> [(q,i) for q in qtd for i in ingredientes]
[(2, 'cenouras'), (2, 'vagens'), (2, 'rabanetes'),
(3, 'cenouras'), (3, 'vagens'), (3, 'rabanetes'),
(2, 'cenouras'), (2, 'vagens'), (2, 'rabanetes')]
```

Brincando com Baralho!

```
>>> valores = ['As'] + list(range(2,11))+  "Valete Dama Rei".split()
>>> naipes = "Paus Ouros Copas Espadas".split()
>>> baralho = ["%s de %s" % (v,n) for v in valores for n in naipes]
>>> print baralho
['As de Paus', 'As de Ouros', 'As de Copas', 'As de Espadas', '2 de Pa
us', '2 de Ouros', '2 de Copas', '2 de Espadas', '3 de Paus', '3 de Ou
ros', '3 de Copas', '3 de Espadas', '4 de Paus', '4 de Ouros', '4 de C
opas', '4 de Espadas', '5 de Paus', '5 de Ouros', '5 de Copas', '5 de
Espadas', '6 de Paus', '6 de Ouros', '6 de Copas', '6 de Espadas', '7
de Paus', '7 de Ouros', '7 de Copas', '7 de Espadas', '8 de Paus', '8
de Ouros', '8 de Copas', '8 de Espadas', '9 de Paus', '9 de Ouros', '9
 de Copas', '9 de Espadas', '10 de Paus', '10 de Ouros', '10 de Copas'
, '10 de Espadas', 'Valete de Paus', 'Valete de Ouros', 'Valete de Cop
as', 'Valete de Espadas', 'Dama de Paus', 'Dama de Ouros', 'Dama de Co
pas', 'Dama de Espadas', 'Rei de Paus', 'Rei de Ouros', 'Rei de Copas'
, 'Rei de Espadas']
>>>
```

Embaralhando o Baralho!

```
>>> from random import shuffle
>>> shuffle(baralho)
>>> baralho
['9 de Paus', '7 de Ouros', '7 de Espadas', '2 de Copas', '10 de Copas
', 'As de Espadas', '3 de Paus', '4 de Espadas', '6 de Copas', 'Valete
 de Espadas', '3 de Ouros', 'Valete de Copas', '8 de Paus', '10 de Esp
adas', 'Rei de Espadas', '8 de Copas', 'Dama de Paus', '9 de Copas', '
8 de Espadas', '9 de Ouros', 'Rei de Ouros', '5 de Ouros', '2 de Ouros
', 'Valete de Ouros', '2 de Paus', '5 de Copas', 'Dama de Copas', 'Val
ete de Paus', '2 de Espadas', '3 de Espadas', 'As de Paus', '4 de Copa
s', '4 de Ouros', '8 de Ouros', '9 de Espadas', '6 de Espadas', '7 de
Paus', 'Dama de Espadas', 'Rei de Copas', '4 de Paus', 'Rei de Paus',
'As de Copas', 'Dama de Ouros', 'As de Ouros', '10 de Ouros', '6 de Ou
ros', '5 de Paus', '5 de Espadas', '6 de Paus', '3 de Copas', '7 de Co
pas', '10 de Paus']
```

E QUE A FORÇA ESTEJA COM VOCÊ!

- Inverter a ordem das cartas
  - `baralho.reverse()`
- Tirar a carta do topo
  - `baralho.pop()`
- Adicionar 4 coringas
  - `baralho.extend(['Coringa']*4)`
- Colocar em ordem
  - `baralho.sort()`
    - Pelo número de letras:
      `baralho.sort(key=len)`

Já que estamos perto do Py (aquele outro...)

Baterias: *urllib* e re


```
1  #coding: utf-8
2  """
3  Busca em uma página web o valor do dólar
4  """
5  import urllib.request
6  import re
7  print ("=" *10, "Conversor para Dólares","=" *10)
8  print ("Valor em R$?"),
9  value = str(input())
10
11 url = """http://www.gocurrency.com/v2/
12         dorate.php?inV=%s
13         &from=BRL&to=USD&Calculate=Convert""" % value
14
15 url = re.sub(r'\s|\n', '', url)
16 print ("Consultando do site GoCurrency.com...")
17 currency = urllib.request.urlopen(url)
18 data = str(currency.read())
19 dolar = re.search('(\d*)\.(\d*) US Dollar', data)
20 print ("%s Reais equivalem a ..." % value)
21 print (dolar.group(1) + '.' + dolar.group(2), 'Dólares')
22 currency.close()
```

UFA, TERMINOU !!!!

Obrigado pela sua paciência...

faspanhol@gmail.com

@_photon_

www.slideshare.net/_photon_