Introdução à linguagem de programação Python

Rafael Sachetto Oliveira [1]

[1] Departamento de Ciência da Computação
Universidade Federal de São João del Rei

XIV Semana de Computação da UFJF

python™

Quem é esse cara?

- Formado em Ciência da Computação pela UFJF.
- Mestre em modelagem computacional pela UFJF.
- Doutorando em Ciência da Computação pela UFMG.

- Usuário Python desde 2006.
- Usuário Linux desde 2002.
- Entusiasta de software de código aberto.

python™

Sobre a linguagem

- Criada por Guido van Rossum em 1991
- Origem do nome: grupo de humoristas *Monty Python*
- Linguagem de altíssimo nível (VHLL)
- Sintaxe simples e fácil de ser assimilada
- Fácil de usar, aprender, ler
- Orientada à objetos, estruturada e funcional
- Tipagem forte e dinâmica
- Interpretada
- Ambiente interativo

python™

Mais sobre a linguagem

- Extremamente portável (Multiplataforma)
  - Unix/Linux, Windows, Mac, PalmOS, WindowsCE, RiscOS, VxWorks, QNX, OS/2, OS/390, AS/400, PlayStation, Sharp Zaurus, BeOS, VMS...
- Compila para byte code
  - compilação implícita e automática
- Gerenciamento automático memória (Garbage Collector)
- Poderosas estruturas de dados nativas
  - Listas
  - Dicionários
- Licença GPL-compatível

python™

Ainda mais sobre a linguagem

- Tudo é objeto
- Pacotes, módulos, classes, funções
- Tratamento exceções
- Sobrecarga de operadores
- Identação para estrutura de bloco
  - O resto é sintaxe convencional

python™

Por que usar Python?

- Uma das linguagens mais divertidas que se tem atualmente
- Já vem com “baterias inclusas” (vasto repertório de bibliotecas)
- Protótipos rápidos sem preocupação com detalhes de implementação da linguagem
  - Linguagem Interpretada: evita “codifica-compila-roda”
- Bem menos linhas de código comparando com Java, C/C++...

python™

Quem usa?

- Google (vários projetos)
- NASA (vários projetos)
- RedHat (ferramentas instalação Linux)
- Muitas Universidades, como MIT, e Stanford
- Globo.com

python™

Vamos Começar!

- Esqueça declarações de tipos de variáveis
- Esqueça begin e end
- Esqueça { e }
- Esqueça ;
- Se você já era organizado, não sofrerá!
- A identação é obrigatória! :)

python™

Shell Interativa

- Ótimo para aprender sobre a linguagem
- Experimentar bibliotecas
- Testar novos módulos
- Usar como calculadora

Exemplos

```
>> print "opa!"
opa!
>> x = 2 ** 3
>> x / 2
4
>>
```

python™

Tipos de dados básicos

- Números: int, long, float, complex
- Strings, Unicode: imutáveis
- Tuplas: imutáveis
- Listas, dicionários e conjuntos: “containers”, mutáveis

python™

Variáveis Numéricas

- São tipos imutáveis:
  - num_int = 13
  - num_int_long = 13L
  - num_real = 13.0

python™

Strings

- São imutáveis
- Armazenam texto (conjuntos de caracteres)
- Crescem até o limite da memória
- Substituem *char* e vetor de *char*
- Acesso sequencial, em fatias ou direto por índice
- Implementadas com arrays
- Possuem diversos métodos: procurar, contar, mudar caixa, etc.

python™

String - Exemplos

```
>>"hello"+"world" # concatenação
'helloworld'
>>"hello"*3 #repetição
'hellohellohello'
>>"hello"[0]
'h'
>>"hello"[-1] # (do final)
'o'
>>"hello"[1:4] # "slicing"
'ell'
>>len("hello") # tamanho
5
>>"hello" < "jello" # comparação
True
>>"e" in "hello" # busca
True
```

String - Métodos

Principais métodos: split, count, index, join, lower, upper, replace

split()

```
var = 'o guia do mochileiro das galaxias'
print var.split()
```

Saída

['o', 'guia', 'do', 'mochileiro', 'das', 'galaxias']

count()

```
var = 'o guia do mochileiro das galaxias'
print var.count('guia')
```

Saída

1

String - Métodos

join()

```
var = 'o guia do mochileiro das galaxias'
var2 = var.split()

print "/".join(var2)
```

Saída

o/guia/do/mochileiro/das/galaxias

Listas e tuplas

- Armazenam coleções de objetos heterogêneos
- Crescem até o limite da memória
- vetores, matrizes, registros e listas encadeadas
- Acesso sequencial, em fatias ou direto por índice
- Implementadas com arrays
- Possuem diversos métodos: adicionar, remover, ordenar, procurar, contar
- Listas são mutáveis e tuplas são imutáveis
- Listas são delimitadas por [e] e tuplas são delimitados por (e)

python™

Tuplas

Exemplo

```
tupla = ('MG', 'Juiz de Fora')
print tupla[1]
```

Saída

Juiz de Fora

Listas

- Arrays flexíveis
  - a = [98, "bottles of beer", ["on", "the", "wall"]]
- Operações comuns
  - a+b, a*3, a[0], a[-1], a[1:], len(a)
- Item e slice
  - a[0] -> 98
  - a[1:2] -> ["bottles", "of", "beer"]
  - del a[-1]
    - -> [98, "bottles", "of", "beer"]

python™

Listas

Exemplo

```
a = range(5)
print a
a.append(5)
print a
a.insert(0, 42)
print a
a.reverse()
print a
a.sort()
print a
```

Saída

```
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4, 5]
[42, 0, 1, 2, 3, 4, 5]
[5, 4, 3, 2, 1, 0, 42]
[0, 1, 2, 3, 4, 5, 42]
```

python™

Dicionários

- Armazenam pares chave/valor de forma desordenada
- Crescem até o limite da memória
- Acesso sequencial, em fatias ou direto pela chave
- Implementados com tabelas *hash*
- Possuem diversos métodos: adicionar, remover, procurar, contar

python™

Dicionários

Exemplo

```
>>>estoque = {'peras': 5, 'laranjas': 2}
>>>estoque['peras']
5
>>>estoque['peras'] = 4
>>>estoque['bananas'] = 2
>>>estoque.get('melancias','Em falta')
'Em falta'
>>>estoque
{'bananas': 2, 'laranjas': 2, 'peras': 4}
>>>'uvas' in estoque
False
>>>estoque.has_key('bananas')
True
>>>estoque.items()
[('laranjas', 2), ('bananas', 2), ('peras', 4)]
```

python™

Variáveis

- Identificadores: (underscore ou letra) + (qualquer número de dígitos ou underscores)
- Case sensitive (Var $\neq$ var)
- Nomes são criados quando atribuídos pela primeira vez
- Nomes devem ser atribuídos antes de serem referenciados

```
spam = 'Spam' #basico
spam, ham = 'yum', 'YUM' #tupla
spam = ham = 'lunch' #multiplo
spam + foo #Erro!!
```

Atribuição

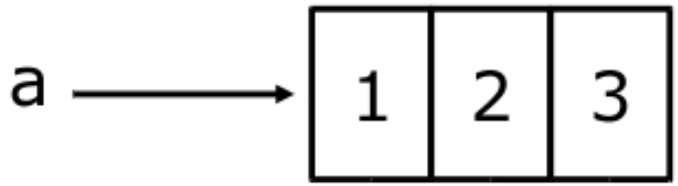
- Atribuição manipula referências
  - x = y não faz uma cópia de y
  - x = y faz x referenciar ao objeto que y referencia
- Bastante útil e eficiente, **mas cuidado!**
- Exemplo:

```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```

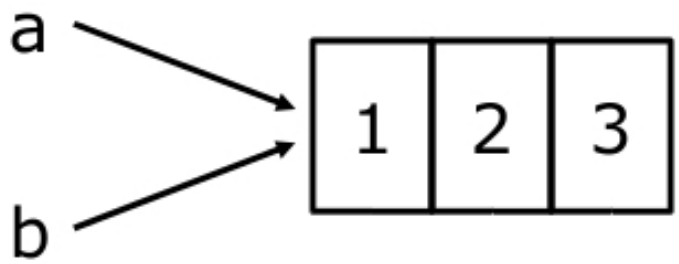
- Para obter um novo objeto, ao invés de uma referência para um objeto existente deve-se usar o módulo **copy**

Atribuição - Exemplo

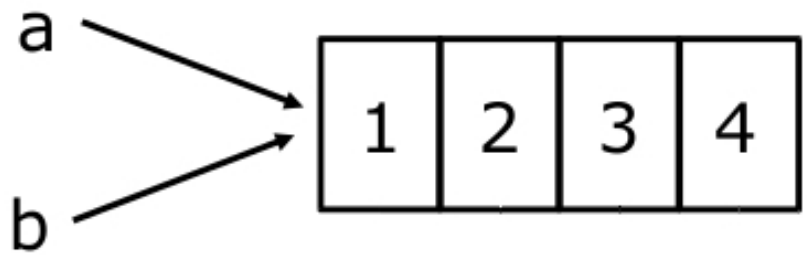
```
a = [1, 2, 3]
```




```
b = a
```

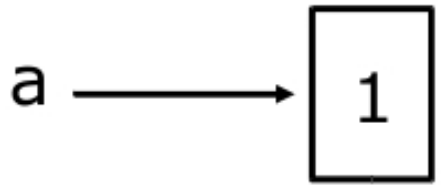



```
a.append(4)
```

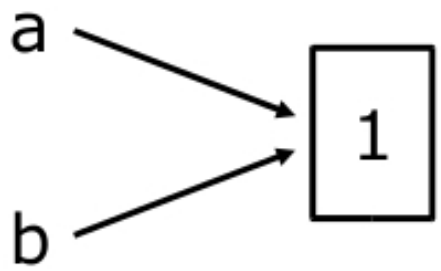


Atribuição - Outro Exemplo

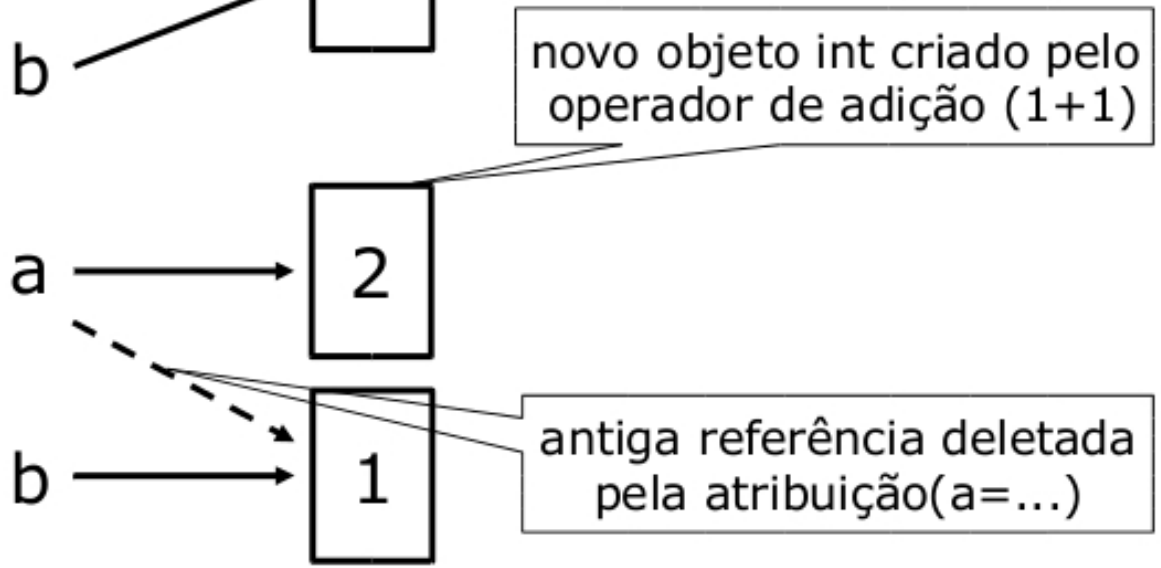
```
a = 1
```




```
b = a
```




```
a = a+1
```



Formatando a saída

```
>>> print "numeros: %d e %05d" % (1,2)
numeros: 1 e 00002

print "Linguagem: %s" % 'Python'
Linguagem: Python
```

Estruturas de Controle

If-Else

```
if exp:
    #comandos
else:
    #comandos
```

If-Else-If-Else

```
if exp:
    #comandos
elif exp:
    #comandos
else:
    #comandos
```

python™

Estruturas de Controle

For

```
for num in range(200):
    print num
```

For

```
for letra in 'python':
    print letra
```

While

```
while exp:
    #comandos
    if exp:
        break
    elif exp:
        continue
    #comandos
```

python™

list comprehensions e Generators

list comprehensions

```
pares = [i for i in range(100) if (i%2) == 0]
impares = [i for i in range(100) if (i%2) != 0]
lista_zeros = [0 for i in range(10)]
matriz = [ [ 0 for j in range(3) ] for i in range(5)]
```

Generators

```
pares = (i for i in range(100) if (i%2) == 0)
print pares.next() #0
print pares.next() #2

soma_impares = sum (i for i in range(100) if (i%2) != 0)
```

python™

Funções e procedimentos

```
def nome(arg1, arg2, ...):
    """Documentacao"""
    #comandos
    return #procedimento
    return expressao #funcao
```

Exemplo

```
def exemplo(a,b,c):
    """Soma tres valores"""
    return a + b + c

>>> exemplo(5,1,3)
9
```

python™

Passagem de parâmetros

```
def changer (x,y):
    x = 2 #modifica x local apenas
    y[0] = 'hi' #modifica o objeto compartilhado
```

Argumentos Opcionais

- É possível definir argumentos defaults que não precisam ser passados

```
def func(a, b, c=10, d=100):
    print a, b, c, d

>>> func(1,2)
1 2 10 100
>>> func(1,2,3,4)
1,2,3,4
```

Argumentos Opcionais

- **Aviso Importante:** O valor default é avaliado apenas uma vez. Isso faz uma diferença quando o default é um objeto mutável como uma lista, dicionário, ou instâncias de classes. Por exemplo, a seguinte função acumula os argumentos passados em chamadas subsequentes:

```
def f(a, L=[]):
    L.append(a)
    return L

>>>print f(1)
[1]
>>>print f(2)
[1, 2]
>>>print f(3)
[1, 2, 3]
```

Argumentos Opcionais

- Para que esse problema não aconteça você pode escrever a função anterior da seguinte forma:

```
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
```

Algumas funções úteis

- dir() -> lista atributos de um objeto
- help() -> help interativo ou help(objeto), info. sobre objeto
- type() -> retorna tipo do objeto
- raw_input() -> prompt de entrada de dados
- int(), str(), float()... -> typecast
- chr(), ord() -> ASCII
- max(), min() -> maior e menor de uma string, lista ou tupla

python™

Módulos

- Um módulo é um arquivo contendo definições de Python e declarações.
- O nome do arquivo é o nome do módulo com o sufixo ’.py’ anexado.
- Dentro de um módulo, o nome do módulo (como uma string) está disponível como o valor da variável global __name__.

Módulos

fibo.py

```
# modulo Fibonacci
def fib(n):
    """Escreve a serie de Fibonacci ate n"""
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

def fib2(n):
    """Retorna a serie de Fibonacci ate n"""
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result

if __name__=="__main__":
    fib(10)
```

Exemplo

```
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
>>> help(fibo)
Help on module fib:

NAME
    fibo

FILE
    fibo.py

FUNCTIONS
    fib(n)
        Escreve a serie de Fibonacci ate n

    fib2(n)
        Retorna a serie de Fibonacci ate n
```

Exceções

- Informar que uma condição anormal ocorreu

```
a = [ 1, 2, 3 ]
try:
    print a[5]
except IndexError:
    print "Posicao inexistente!"
finally:
    print "Fim do teste"
```

Manipulando Arquivos

Lendo arquivos

```
arq = open('teste.txt', 'r') #rb
for linha in arq:
    print linha
arq.close()
```

Escrevendo arquivos

```
arq = open('teste.txt', 'w')#'a','r+','w+','a+'
arq.write('linha1\n')
arq.write('linha2\n')
arq.close()
```

python™

Manipulando Arquivos

Alguns Métodos:

- read([nbytes]), readline(), readlines()
- write(string), writelines(list)
- seek(pos[, how]), tell()
- flush(), close()

Programa exemplo

Conta palavras

- Pedir Arquivo
- Ler Arquivo
- Contar quantas vezes aparece cada palavra
- Listar as palavras em ordem alfabética, junto com o seu número de repetições no texto

python™

Programa Exemplo

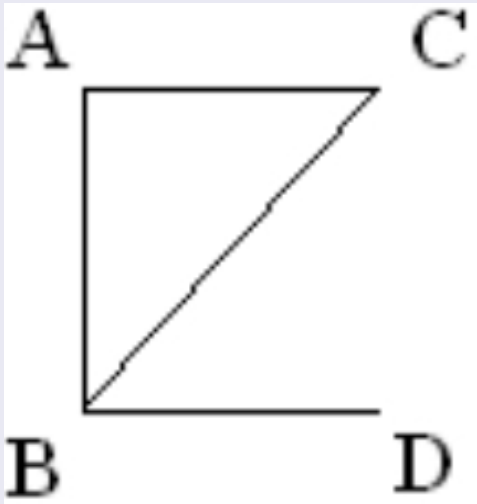
Conta palavras

```
p_count = {}
arqnom = raw_input('Digite o nome do Arquivo: ')
arq = open(arqnom, 'r')

for linha in arq:
    palavras = linha.split()
    for palavra in palavras:
        p_count[palavra] = p_count.get(palavra,0) + 1
        arq.close()

palavras = p_count.keys()
palavras.sort()
for palavra in palavras:
    print 'Palavra: %s, Quantidade: %05i' %
    (palavra, p_count[palavra])
```

python™

Grafos com dicionários

- Podemos utilizar dicionários para representar diversos tipos de dados, inclusive grafos!




```
grafo = {'A': ['B','C'], 'B': ['A','C','D'], 'C': ['A','B'], 'D': ['B']}
```

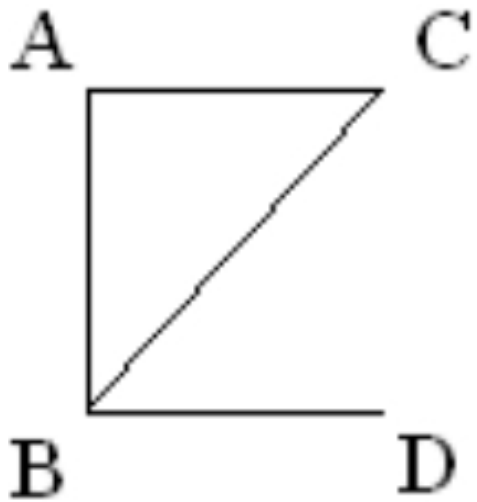
Achar caminho

```
def ache_caminho(grafo, inicio, final, caminho=None):

    if caminho is None:
        caminho = []

    caminho = caminho + [inicio]
    if inicio == final:
        return caminho
    for nodo in grafo[inicio]:
        if nodo not in caminho:
            novocaminho = ache_caminho(grafo, nodo, final,
                                       caminho)
            if novocaminho:
                return novocaminho
    return None
```

Grafos com dicionários

Exemplo

```
>>> ache_caminho(grafo, 'A', 'D')
['A', 'B', 'D']
>>> ache_caminho(grafo, 'A', 'D')
['A', 'B', 'D']
>>> ache_caminho(grafo, 'D', 'C')
['D', 'B', 'A', 'C']
```



python™

Classes

```
class Matriz(object):

    nome_da_classe = "Matriz"

    def __init__(self, n, m):
        self.matriz = [[None for j in range(m)] for i in range(n) ]
```

Classes

```
class Matriz(object):

    nome_da_classe = "Matriz"

    def __init__(self, n, m):
        self.matriz = [[None for j in range(m)] for i in range(n) ]

    def __getitem__(self, i):
        return self.matriz[i]
```

Classes

```
class Matriz(object):

    nome_da_classe = "Matriz"

    def __init__(self, n, m):
        self.matriz = [[None for j in range(m)] for i in range(n) ]

    def __getitem__(self, i):
        return self.matriz[i]
```

Classes

```
class Matriz(object):

    nome_da_classe = "Matriz"

    def __init__(self, n, m):
        self.matriz = [[None for j in range(m)] for i in range(n) ]
        self.linhas = n
        self.colunas = m

    def __getitem__(self, i):
        return self.matriz[i]

    def __add__(self, b):
        c = Matriz(self.linhas, self.colunas)
        for i in range(self.linhas):
            for j in range(self.colunas):
                c[i][j] = self[i][j] + b[i][j]
        return c

    @staticmethod
    def nome():
        print Matriz.nome_da_classe
```

python™

Classes

```
from matriz import Matriz

if __name__ == "__main__":
    a = Matriz(2,2)
    b = Matriz(2,2)
    print Matriz.nome()

    for i in range(a.linhas):
        for j in range(a.colunas):
            a[i][j] = i
            b[i][j] = j

    c = a+b

    print c.matriz
    print a.matriz
    print b.matriz
```

python™

Monkey patch

```
from matriz import Matriz

def sub(self, b):
    c = Matriz(self.linhas, self.colunas)
    for i in range(self.linhas):
        for j in range(self.colunas):
            c[i][j] = self[i][j] - b[i][j]

    return c

if __name__ == "__main__":
    a = Matriz(2,2)
    b = Matriz(2,2)

    for i in range(a.linhas):
        for j in range(a.colunas):
            a[i][j] = i
            b[i][j] = j

    Matriz.__sub__ = sub
    c = a-b
    print c.matriz
    print a.matriz
    print b.matriz
```

python™

Threading

```
import threading

t1 = threading.Thread(target=funcao,args=())
t1.start()
...
t1.join()

sem = threading.Semaphore(2)
sem.acquire()
...
sem.release()
```

Sockets

- Comunicação entre processos
- Cliente
  - Endereço e porta de conexão
- Servidor
  - Endereço e porta de escuta
- Comunicação
  - read()
  - write()

python™

Sockets

Cliente

```
from socket import socket, AF_INET, SOCK_STREAM

HOST = 'localhost'
PORT = 2223
s = socket(AF_INET, SOCK_STREAM)
s.connect((HOST, PORT))
s.send('Mensagem do Cliente!')
data = s.recv(1024)
print data

s.close()
```

Servidor

```
from socket import socket, AF_INET, SOCK_STREAM

HOST = ''
PORT = 2223
s = socket(AF_INET, SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1) # Numero de Conexoes
conn, addr = s.accept()
data = conn.recv(1024)
print data
conn.send('Mensagem do Servidor!')

conn.close()
```

:hon™

- anydbm
  - dumbdbm (lenta e limitada, todas plataformas)
  - dbm (somente em UNIX)
  - gdbm (somente em UNIX)
  - dbhash (biblioteca BSD, em UNIX e Windows)

Exemplo

```
import anydbm

db = anydbm.open('db.dat','c')
db['she'] = 'ela'
db['he'] = 'ele'

for k, v in db.iteritems():
        print k, '\t', v

db.close()
```

Bancos de dados

- Os bancos possuem uma API padrão
- Conexão
- Cursor
  - Comandos SQL
- Commit (Finaliza a operação)

Exemplo

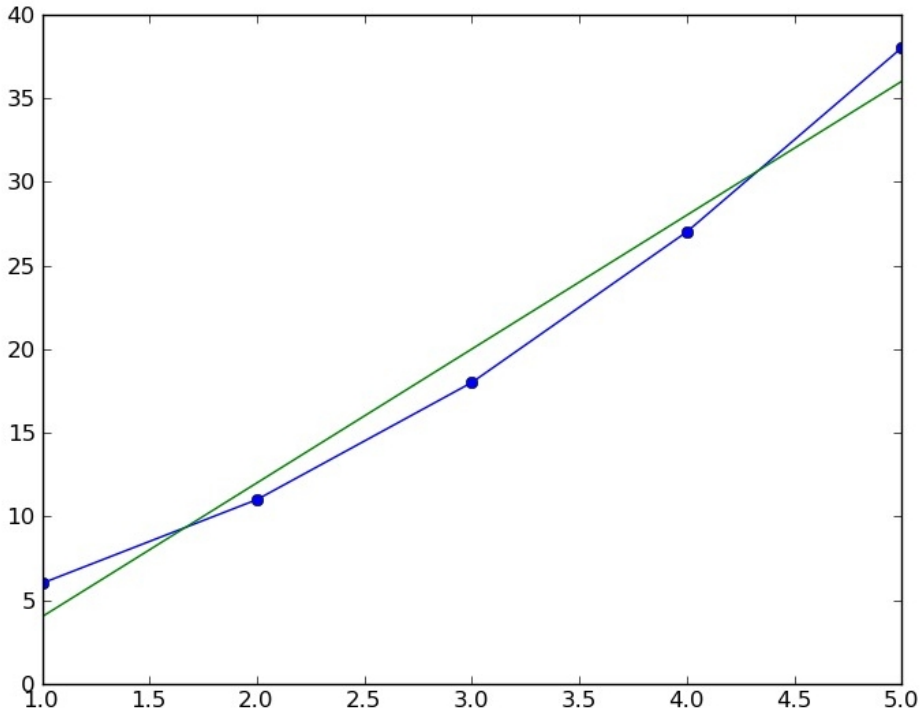
```
import sqlite3
db = sqlite3.connect('test.db')
cur = db.cursor()
cur.execute('create table empresa (cod numeric not null, \
            des character not null, primary key (cod))')
cur.execute('insert into empresa values(1,"empresa teste")')
db.commit()

cur.execute('select * from empresa order by des')
result = cur.fetchall()
for regs in result:
    print regs

db.close()
```

Interfaces Gráficas

- Tkinter - Padrão
- wxPython (antiga wxWindows)
  - GTK no Linux
  - MFC no Windows
- pyGTK
- pyQt
- pyFLTK, FxPy, Anygui

python™

NumPy e matplotlib



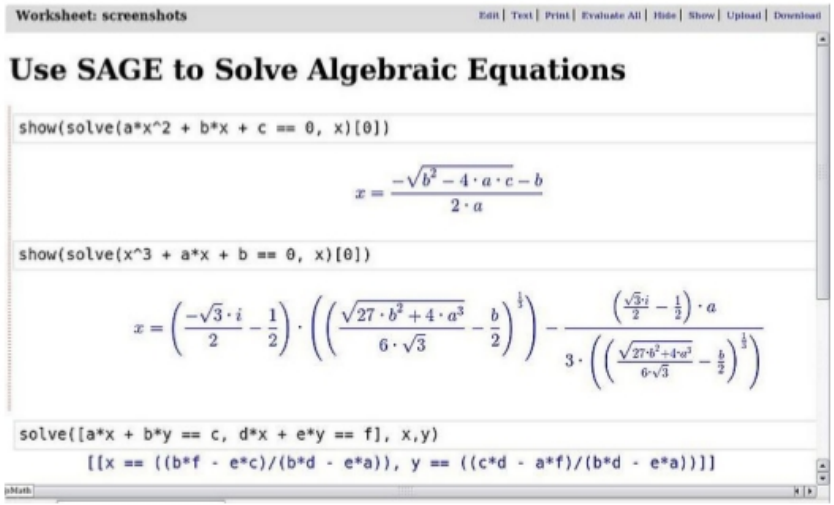

```
from numpy import *
import matplotlib.pyplot as plt

x = array([1,2,3,4,5])
y = array([6, 11, 18, 27, 38])

p1 = poly1d(polyfit(x,y,2))
p2 = poly1d(polyfit(x,y,1))

y1 = p1(x)
y2 = p2(x)

plt.plot(x, y, 'bo', x, y1, x, y2)
plt.show()
```

- Poderoso array N-dimensional de objetos
- integração com código C/C++ e Fortran
- funções de álgebra linear, etc...

python™

Sage

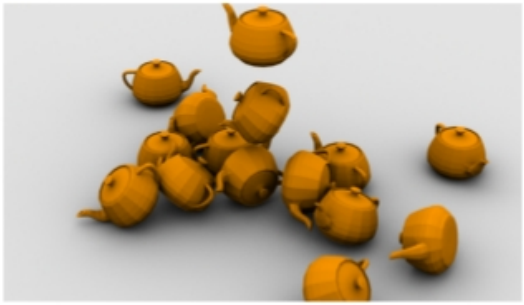
Software Matemático
livre e de código aberto

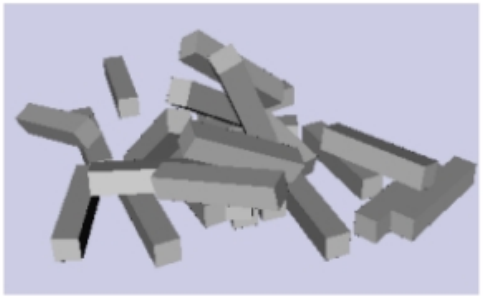


- Aplicativo de matemática GPL
- Combina o poder de muitos pacotes de código aberto existentes em uma interface Python comum
- Missão: Criar uma alternativa livre e de código aberto para Magma, Maple, Mathematica e Matlab

:hon™

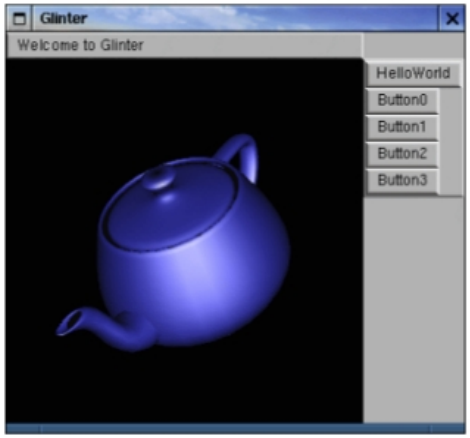
pyODE

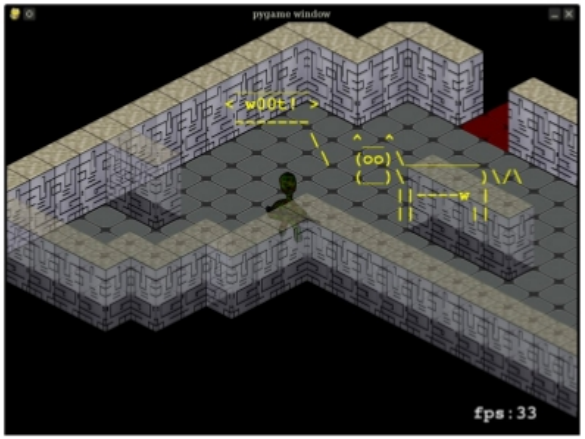




- PyODE é um *binding* de ODE para Python
- ODE: simulação da dinâmica de objetos rígidos
- Detecção de colisão com fricção
- Simulação de veículos, objetos e criaturas em ambientes virtuais
- Jogos de computador, ferramentas de autoria 3D e de simulação

PyOpenGL





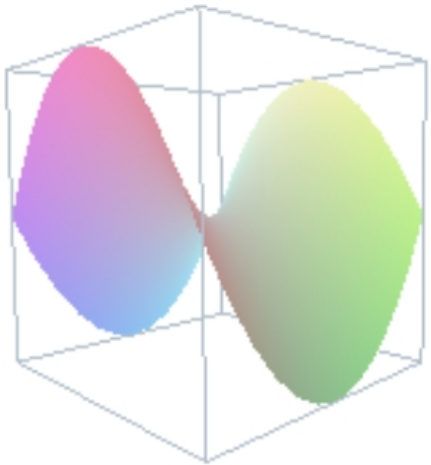
- PyOpenGL: binding de OpenGL para Python
- OpenGL: API livre utilizada na computação gráfica
- Desenvolvimento de aplicativos gráficos, ambientes 3D, jogos, entre outros

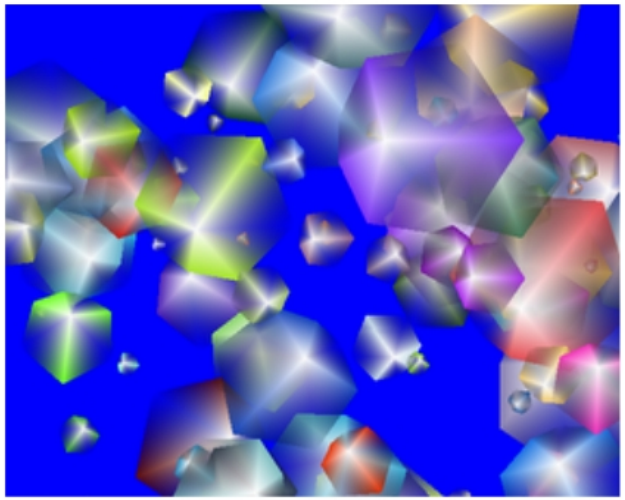
PIL

- suporte à abertura, gravação e manipulação de muitos formatos de imagem diferentes

python™

Pyglet





- Desenvolvimento de jogos e outras aplicações visualmente ricas
- Suporte para eventos de mouse e teclado
- Pode carregar diversos formatos de arquivos de multimídia

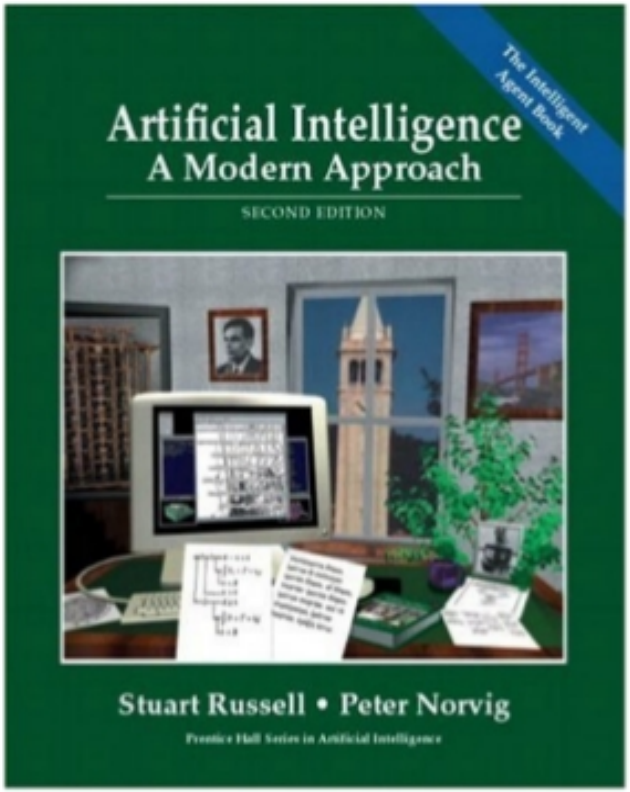
Pygame

- Módulo multi-plataforma projetado para escrever jogos.
- Inclui bibliotecas de computação gráfica e de som.
- construída sobre a biblioteca SDL (Simple DirectMediaLayer)

Panda3D

- Motor de jogo 3D escrito em C++ para ser usado com Python
- Desenvolvido pela Disney e Carnegie Mellon University's Entertainment Technology Center
- Possui motor de física, texturas animadas, suporte a shaders e som 2D e 3D

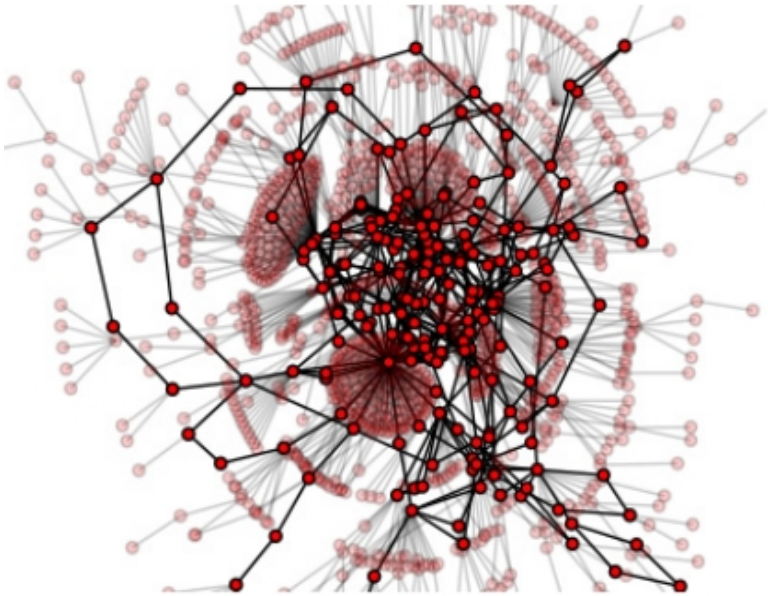
AIMA

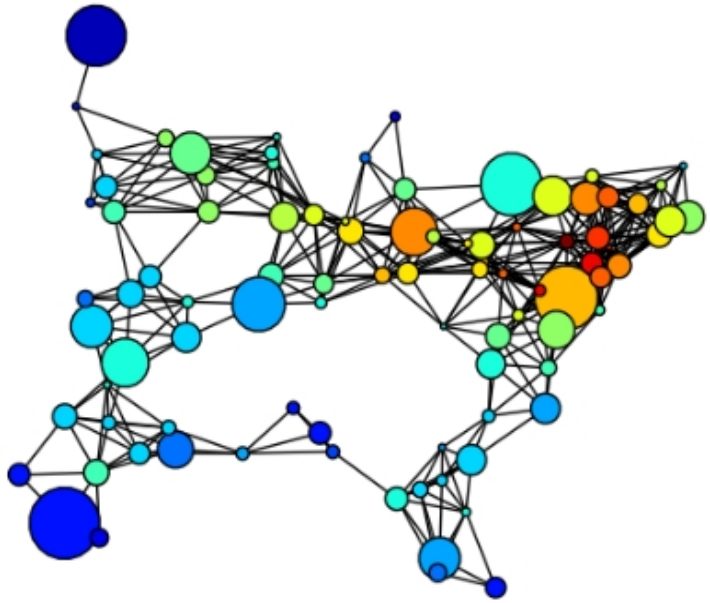


- Código em Python dos algoritmos do livro Artificial
- Intelligence: a Modern Approach, de Peter Norvig
- O livro-texto apresenta os algoritmos em pseudo-código

:hon™

NetworkX





- Criação, manipulação e estudo da estrutura, dinâmica e funções das redes complexas
- E adequada para operações em grandes grafos do mundo real
- Pode ser usada para análise de redes

Twisted

- Framework de rede de código aberto escrito inteiramente em Python
- Permite a criação de proxiesHTTP e servidores SSH (e mais) em Python com o mínimo esforço
- É assíncrono e orientado a eventos

python™

Pyro

- Comunicação de objetos em rede de forma simples
- Acesso a objetos em diferentes máquinas na rede
- Lembra Java Remote Method Invocation (RMI).

Onde buscar mais informação?

- http://www.python.org
- http://docs.python.org
- http://www.pythonbrasil.com.br
- http://www.pythonology.com/
- http://www.google.com :)

python™

Obrigado a Todos!

Referências

- Slides de Marco André Lopes Mendes - slideshare.com/marrcandre
- Slides de Vinicius T. Petrucci
- Alex Augusto da Luz dos Santos

```
import base64

print base64.b64decode('UGVyZ3VudGFzPw==')
print base64.b64decode('T2JyaWdhZG8=')
```

python™