

Introdução à programação em Python

Igor L. S. Russo

Mestrado em Modelagem Computacional
Universidade Federal de Juiz de Fora

9 de Abril de 2015

Sumário

Introdução

Estruturas de dados

Estruturas de controle

Funções

Módulos

Computação científica com Python

- Principais pacotes

- Exemplo - Integração numérica

Vantagens e desvantagens

Introdução

O que é Python?

- ▶ Python é uma linguagem de programação de propósito geral **interpretada**, **interativa** e **orientada a objetos**;

O que é Python?

- ▶ Python é uma linguagem de programação de propósito geral **interpretada**, **interativa** e **orientada a objetos**;
- ▶ Possui **tipagem dinâmica** e estruturas de dados com **altíssimo nível de abstração**;

O que é Python?

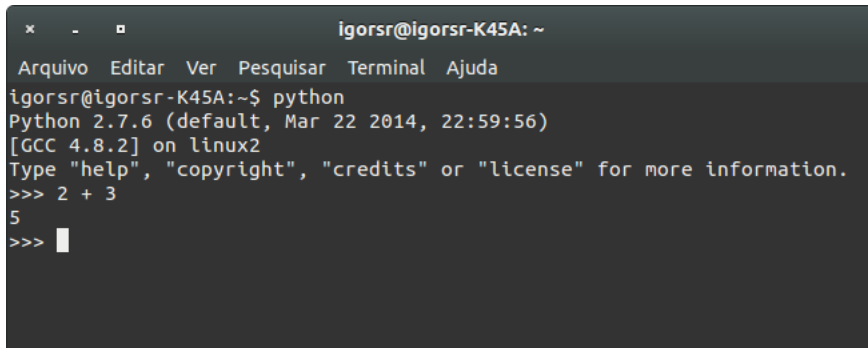
- ▶ Python é uma linguagem de programação de propósito geral **interpretada**, **interativa** e **orientada a objetos**;
- ▶ Possui **tipagem dinâmica** e estruturas de dados com **altíssimo nível de abstração**;
- ▶ Possui uma sintaxe clara e sucinta, favorecendo a legibilidade dos códigos e a produtividade do programador;

O que é Python?

- ▶ Python é uma linguagem de programação de propósito geral **interpretada**, **interativa** e **orientada a objetos**;
- ▶ Possui **tipagem dinâmica** e estruturas de dados com **altíssimo nível de abstração**;
- ▶ Possui uma sintaxe clara e sucinta, favorecendo a legibilidade dos códigos e a produtividade do programador;
- ▶ Finalmente, Python suporta integração com outras linguagens de programação, como C e Fortran.

Python no terminal

- ▶ Assim como Matlab, python é uma linguagem interativa, permitindo a execução de comandos diretamente no terminal:



```
igorsr@igorsr-K45A: ~  
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda  
igorsr@igorsr-K45A:~$ python  
Python 2.7.6 (default, Mar 22 2014, 22:59:56)  
[GCC 4.8.2] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 2 + 3  
5  
>>> 
```

Figura: Python no terminal

Existem várias IDEs para programação em Python:

- ▶ PyCharm
- ▶ PyDev (plugin Eclipse)
- ▶ Netbeans
- ▶ Spyder

Existem várias IDEs para programação em Python:

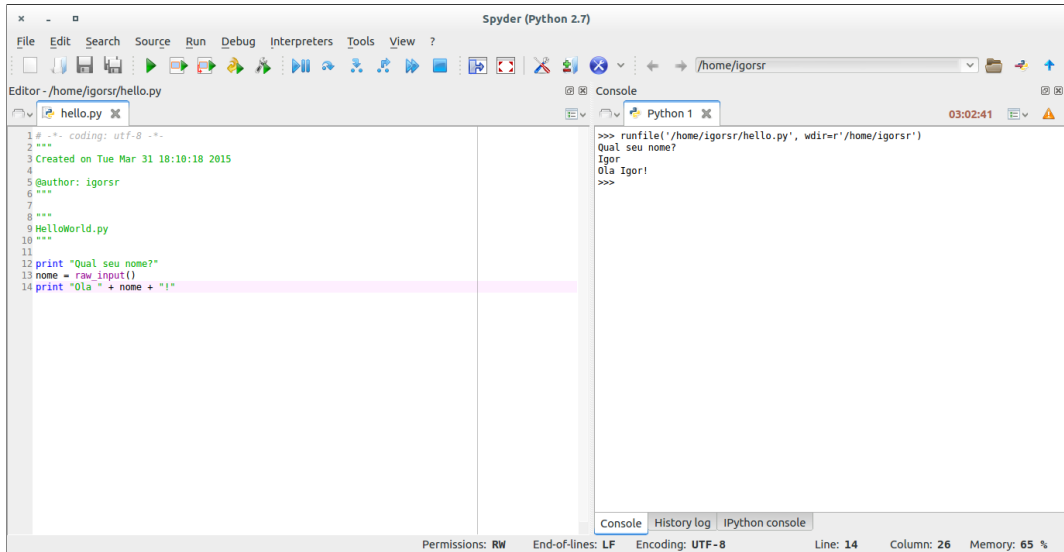
- ▶ PyCharm
- ▶ PyDev (plugin Eclipse)
- ▶ Netbeans
- ▶ Spyder

Scientific PYthon Development EnviRonment

Um ambiente de desenvolvimento interativo para a linguagem Python com recursos avançados de edição, testes, depuração, focado em computação científica.



Spyder



Olá mundo!

Código:

```
"""  
OlaMundo.py  
"""  
  
print "Qual seu nome?"  
nome = raw_input()  
print "Ola " + nome + "!"
```

Olá mundo!

Código:

```
"""  
OlaMundo.py  
"""  
  
print "Qual seu nome?"  
nome = raw_input()  
print "Ola " + nome + "!"
```

Execução:

```
>> python OlaMundo.py  
>> Qual seu nome?  
>> Igor  
>> Ola Igor!
```

- ▶ Comandos são separados por quebras de linha, ao invés de ";"
- ▶ Declaração de variáveis não indica seu tipo:
 - ▶ Este é inferido a partir do conteúdo atribuído na inicialização (tipagem dinâmica)
- ▶ Não é necessário compilar o código
- ▶ Diferentemente da linguagem C, não existe uma função "main"
- ▶ O gerenciamento de memória é automático:
 - ▶ *Garbage collector*

Estruturas de dados

Tipos de dados básicos

- ▶ Python suporta os tipos básicos de dados
 - ▶ Int, long int, float, double, complex, boolean, string

Tipos de dados básicos

- ▶ Python suporta os tipos básicos de dados
 - ▶ Int, long int, float, double, complex, boolean, string

```
# Inteiro
```

```
idade = 20
```

```
# Ponto flutuante
```

```
altura = 1.75
```

```
# Complexo
```

```
c = 10 + 5j
```

```
# Strings
```

```
frase = "Texto."
```

```
frase2 = 'Agora com aspas simples'
```

```
texto = """ Textos com quebras de linha  
tambem podem ser utilizados.  
"""
```

Entrada de dados

- ▶ A entrada de dados pelo console pode ser feita pela função `raw_input()`, como visto no exemplo "Hello World"
- ▶ Neste caso, como os dados são lidos como string, é necessário convertê-los para o tipo de dado desejado:

```
# Convertendo inteiro
n = int(raw_input("Informe o numero de alunos:"))

# Convertendo float
r = float(raw_input("Informe o raio:"))

piString = "3.4"
pi = float(piString)
```

Aritméticos:

- ▶ Soma: $3.86 + 2.5$
- ▶ Subtração: $2.0 - 2.5$
- ▶ Divisão: $8.35 / 2.5$
- ▶ Divisão inteira:
 $3.0 // 2.0$
- ▶ Módulo: $2 \% 3$
- ▶ Potência: $2^{**}3$

Aritméticos:

- ▶ Soma: $3.86 + 2.5$
- ▶ Subtração: $2.0 - 2.5$
- ▶ Divisão: $8.35 / 2.5$
- ▶ Divisão inteira:
 $3.0 // 2.0$
- ▶ Módulo: $2 \% 3$
- ▶ Potência: $2^{**}3$

Relacionais:

- ▶ $==$
- ▶ $>$
- ▶ $>=$
- ▶ $<$
- ▶ $<=$
- ▶ $!=$ ou $<>$

Aritméticos:

- ▶ Soma: $3.86 + 2.5$
- ▶ Subtração: $2.0 - 2.5$
- ▶ Divisão: $8.35 / 2.5$
- ▶ Divisão inteira:
 $3.0 // 2.0$
- ▶ Módulo: $2 \% 3$
- ▶ Potência: $2^{**}3$

Relacionais:

- ▶ $==$
- ▶ $>$
- ▶ $>=$
- ▶ $<$
- ▶ $<=$
- ▶ $!=$ ou $<>$

Lógicos:

- ▶ and
- ▶ or
- ▶ not

Listas

Introdução

- ▶ Python oferece uma estrutura de dados chamada **list**, que é uma coleção de dados homogêneos ou heterogêneos

- ▶ Python oferece uma estrutura de dados chamada **list**, que é uma coleção de dados homogêneos ou heterogêneos
- ▶ Exemplo:

```
a = [66.25, 333, 333, 1, 1234.5]
dias_da_semana = ["Domingo", "Segunda-feira", "Terça-feira",
                  "Quarta-feira", "Quinta-feira", "Sexta-feira", "Sabado"]

print a[0]
print dias_da_semana[1]
```

```
>> 66.25
>> Segunda-feira
```


Métodos disponíveis nas listas:

- ▶ **list.append(x)**: Insere um elemento ao final da lista
- ▶ **list.insert(i, x)**: Insere um elemento em uma posição específica na lista
- ▶ **list.remove(x)**: Remove um elemento da lista
- ▶ **list.pop([i])**: Remove e retorna um elemento da lista
- ▶ **list.index(x)**: Busca um elemento na lista e retorna seu índice
- ▶ **list.sort()**: Ordena a lista

Estruturas de controle

Estruturas de controle

Condicionais

- ▶ Estruturas de controle de fluxo, que executam ou não um bloco de código de acordo com a avaliação de uma condição lógica

```
if condicao1:  
    <bloco de codigo indentado>  
elif condicao2:  
    <bloco de codigo indentado>  
else  
    <bloco de codigo indentado>
```

Estruturas de controle

Laços de repetição

Laço **for**:

- ▶ Permite iterar através de elementos de uma sequência, como listas ou strings

```
#For em Python  
for i in range(0,30):  
    <bloco de codigo indentado>  
  
for dia in dias_da_semana:  
    <bloco de codigo indentado>
```

Estruturas de controle

Laços de repetição

Laço **for**:

- ▶ Permite iterar através de elementos de uma sequência, como listas ou strings

#For em Python

```
for i in range(0,30):  
    <bloco de codigo indentado>
```

```
for dia in dias_da_semana:  
    <bloco de codigo indentado>
```

//For em C

```
int i;  
for(i=0;i<30;i++)  
{  
    <bloco de codigo>  
}
```

Estruturas de controle

Laços de repetição

Laço **while**:

```
a = 0
while(a<30):
    ...
    a=a+1
```

Funções

Exemplo - Sequência de Fibonacci:

```
n = 10
sequence = [0,1]
for i in range(2,n):
    sequence.append(sequence[i-1]+sequence[i-2])
print sequence
```

```
>> [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```


Definindo uma função:

```
# fibo.py

def fibonacci(tamanhoSequencia):
    """Retorna a sequencia de Fibonacci ate o termo"""
    sequence = [0,1]
    for i in range(2,tamanhoSequencia):
        sequence.append(sequence[i-1]+sequence[i-2])
    return sequence

sequencia = fibonacci(10)
print sequencia
```

```
>> [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Módulos

Módulos são arquivos contendo funções reutilizáveis.

Módulos são arquivos contendo funções reutilizáveis.

Exemplo de utilização:

```
import fibo
sequencia = fibonacci(10)
print sequencia

import fibo as f
sequencia = f.fibonacci(10)
print sequencia
```

Computação científica com Python

Numpy (Numerical Python)

- ▶ Numpy é o pacote fundamental para computação científica com Python, implementando principalmente a manipulação de vetores e matrizes multidimensionais.

Contém, entre outras coisas:

- ▶ Implementação eficiente de vetores multidimensionais (escrita em C)
- ▶ Funções disponíveis para álgebra linear, transformada de Fourier e geração de números aleatórios

Numpy - Criação de arrays

```
import numpy as np

a = np.array([1,2,3]) # vetor
b = np.array([[1,2,3],[4,5,6]]) # matrix 3x3
c = np.zeros([3,3]) # matriz nula 3x3
c = np.ones([3,3]) # matriz 3x3 preenchida com 1's
d = np.identity(10) # matriz identidade

e = np.random.random((3)) # vetor de numeros aleatorios
f = np.linspace(0,10,1000) # vetor de numeros igualmente espaçados
```

Numpy - Métodos de arrays

```
# Soma dos elementos de um vetor  
print a.sum()  
  
# Produto escalar:  
print np.dot([1,2,3],[2,2,2])  
  
# Produto vetorial:  
print np.cross([1,2,3],[2,2,2])
```


- Considere o sistema:

$$\begin{cases} 3 * x1 + x2 = 9 \\ x1 + 2 * x2 = 8 \end{cases}$$

Numpy - Resolução de sistemas lineares

- Considere o sistema:

$$\begin{cases} 3 * x1 + x2 = 9 \\ x1 + 2 * x2 = 8 \end{cases}$$

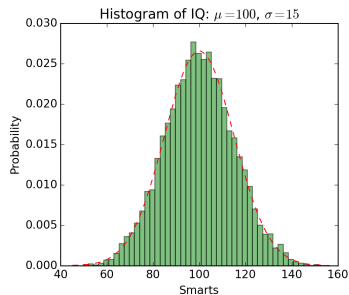
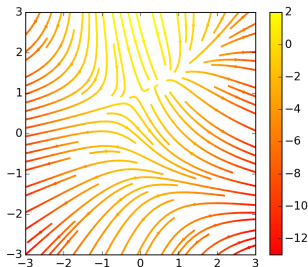
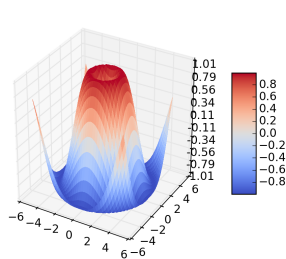
- Código:

```
a = np.array([[3,1], [1,2]])  
b = np.array([9,8])  
x = np.linalg.solve(a, b)  
print x
```

```
>> array([ 2.,  3.])
```

- ▶ SciPy é um "ecossistema" baseado em Python de software open-source para matemáticos, cientistas e engenheiros.
- ▶ Principais funcionalidades:
 - ▶ Integração numérica
 - ▶ Solução numérica de EDOs
 - ▶ Otimização
 - ▶ Matemática simbólica
 - ▶ Álgebra linear

O matplotlib é um pacote para visualização de dados 2D ao estilo do MATLAB e Mathematica. Gera vários tipos de gráficos e exporta para os principais formatos de imagem, como png, eps e pdf.



Exemplos

Matplotlib - Exemplo

Geração de um gráfico simples:

```
from matplotlib.pyplot import *

def f(t):
    return 0.5*exp(t)

t = linspace(0,3,51)
y = f(t)

title('Gráfico de 0.5*exp(t^2)')
pylab.plot(t, y)
pylab.show()
```

Matplotlib - Exemplo

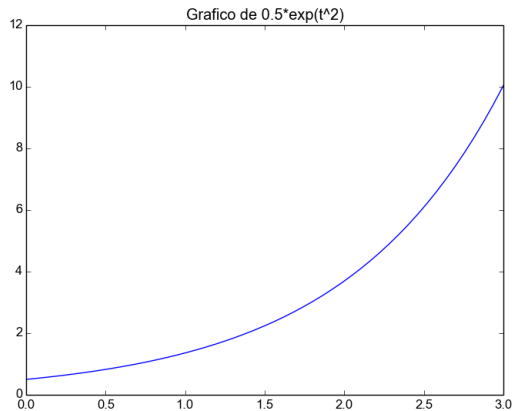


Figura: Gráfico gerado

Exemplo - Integração numérica

Regra do trapézio

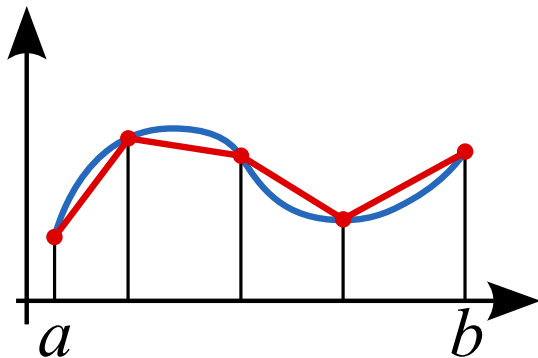


Figura: Regra do trapézio

Regra do trapézio

Regra do trapézio:

$$\int_a^b f(x)dx \approx \frac{h}{2} * (f(a) + f(b))$$

Regra do trapézio repetida:

$$\int_a^b f(x)dx \approx h * (f(x_0)/2 + f(x_1) + \dots + f(x_{n-1}) + f(x_n)/2))$$

Função **trap**:

```
def trap(g, inicio, fim, n):  
    h = (fim-inicio)/float(n);  
    estimativa = (f(inicio) + f(fim))/2;  
    for i in range(n):  
        x=inicio+i*h  
        estimativa+=g(x)  
    return estimativa*h
```

Implementação - 2

```
def exata(x):  
    return (x**4.0)/4.0  
  
def f(x):  
    return x**3  
  
print "Integral aproximada: " , trap(f, 0,1, 1000)  
print "Integral exata: ", (exata(1)-exata(0))
```

```
Integral aproximada: 0.249995000025  
Integral exata: 0.25
```

- ▶ Open Source!
- ▶ Linguagem de altíssimo nível
- ▶ Sintaxe simples -> Baixa curva de aprendizado
- ▶ Diversas bibliotecas disponíveis, para várias classes de problemas
- ▶ Código legível (geralmente)
- ▶ Pode ser utilizada como linguagem script, para automação de tarefas

- ▶ Linguagem interpretada \Rightarrow Geralmente apresenta desempenho computacional inferior às linguagens compiladas

Hans Petter Langtangen. Python Scripting for Computational Science, volume 3 of Texts in Computational Science and Engineering. Springer-Verlag, Berlin, second edition, 2006.

Alex Martelli. Python - in a nutshell: a desktop quick reference: covers Python 2.2. O'Reilly, 2003. ISBN 978-0-596-00188-9.

Python.org. Documentação python, 2015. URL <https://www.python.org/doc/>. [acessado em Março de 2015].

Zed A. Shaw. Learn Python The Hard Way. 2011. URL <http://learnpythonthehardway.org/book/>.

Obrigado!

