



Introdução a Linguagem de Programação Python

Flávio Ribeiro
flavio.ribeiro@avaty.com.br

www.avaty.com.br

- O que é Python?
- Por que Usar Python
- O Interpretador Interativo
- Variáveis e Tipos
- Regras e Sintaxe
- Funções
- Introdução a Orientação a Objetos em Python
- Algumas Práticas
 - Threads
 - Sockets
 - Extendendo Python com C



▪ Flávio Ribeiro

- Graduando em Engenharia Elétrica – IFPB
- Estagiário Avaty! Tecnologia e Inovação
- Fundador do Núcleo Comunicação Digital - IFPB
- Entusiasta
 - Python
 - Sistemas Embarcados
 - Dispositivos Móveis
- <http://flavioribeiro.com>
- flavio.ribeiro@avaty.com.br
- flavioribeiro @ freenode #python-br




- Linguagem de Altíssimo Nível (Very High Level Language)
- Suporta múltiplos paradigmas
 - Estruturada
 - Orientação a Objetos
- Case Sensitive
- Interpretada
 - Transformação source > bytecode
- Tipagem Dinâmica
- Multiplataforma
 - Symbian, Linux, Windows, OSX
- “Baterias Inclusas”



Por que usar Python?

- Python vai na web, no desktop, OLPC, Celulares, Internet Tablets...




Payable Help Admin

Show payments with status matching: 

Paid or due date: From: To:


#	Description	Supplier	Due Date	Paid Date	Status	Value
0021	1/1 Bill for order 1	Cia de Roupas São Ca...	04/09/2008		To Pay	\$7,930.00
0022	Phone bill		04/11/2008		To Pay	\$130.00
0023	Plumber		04/09/2008		To Pay	\$14.00
0024	1/1 Money for order 3	Cia de Roupas São Ca...	04/10/2008		Preview	\$673.00




Total: \$8,747.00

 Details  Pay order  Print





Stoq - Ponto de Vendas Admin

Vendas Pesquisar Caixa Ajuda

 Novo... Número do Pedido: 00009 Cliente: Alessandra Almeida It... Vendedor: Async Open Source

Código de Barras: Quantidade:  Adicionar  Pesquisar 

Referência	Descrição	Preço	Quantidade	Unidac	Total
00002	Bermuda Sarja	R\$ 149,00	2,0		R\$ 298,00
00004	Camiseta Tinturada Xcuba Comp	R\$ 89,00	1,0		R\$ 89,00
00008	Casaco Vilan	R\$ 503,00	1,0		R\$ 503,00

 Editar  Remover  Entrega  Confirmar

Total: R\$ 890,00

Por que usar Python?

- Python vai na web, no desktop, OLPC, Celulares, Internet Tablets...



Por que usar Python?

AVATY!

TECNOLOGIA E INOVAÇÃO

- Python vai na web, no desktop, OLPC, Celulares, Internet Tablets...



Por que usar Python?

- Python vai na web, no desktop, OLPC, Celulares, Internet Tablets...
- “7ª Linguagem mais utilizada” (TIOBE 2009)
 - É uma porcentagem, não ajustada, sobre a quantidade de hits em 5 engines de procura.
- Quem usa Python?



PHILIPS



▪ Python 2.x vs. Python 3.0

- Quebra de Compatibilidade
 - Print() é uma função
 - Dicionários remodelados (métodos de iteração retirados)
 - Strings agora são sempre unicode (modelo java-like)
 - Divisão de inteiros retorna float (// pra old-style)
- Como converter código 2.x pra 3.0?
- A continuação da série 2.x
 - multiprocessing



- O que é?
 - Aplicação nativa para testes de código
 - Interpretador run-in-time

```
flavio@avaty:~/dev$ python
Python 2.5.2 (r252:60911, Oct 5 2008, 19:24:49)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

▪ Tipagem Dinâmica

```
>>> a = 1
>>> type(a)
<type 'int'>
>>> a = 'flavio'
>>> type(a)
<type 'str'>
```

▪ Tipos Nativos

- float, int, long, boolean, strings, listas, tuplas e dicionários.



▪ Strings

- Objeto iterável
- Imutável
- Aspas simples (') e duplas (")
- Acessível através de índices
- Operador de concatenação: +
- **upper()** , **count()**, **title()**, **find()**, **startswith()**, **isalpha()**, **isdigit()**
- Slice e Substrings através de índices
- Membership

```
>>> 'io' in 'flavio'  
True
```



▪ Listas

- Objeto iterável
- Mutável
- Objetos dentro de colchetes []
- Acessível através de índices
- Aceita vários tipos de objetos diferentes
- Lista bi-dimensional
- Concatenação de listas
 - **+** e **extend()**
- Adição de Objetos
 - **append()** e **insert()**
- Remoção de Objetos
 - **pop()** e **remove()**
- **Index(), sort(), reverse(), count()**
 - Slicing



- **A Função range()**

- Função geradora de iteradores
- Útil na construção de laços for

```
>>> range(4)
[0, 1, 2, 3]
>>> range(4, 9)
[4, 5, 6, 7, 8]
>>> range(0, 10, 2)
[0, 2, 4, 6, 8]
```

- xrange() = range() no Py3k



▪ Tuplas

- Existência questionada
- Imutável
- Delimita objetos por ()
- Indicado para retorno de funções\métodos com múltiplos valores

```
>>> a = (1,2,3,'flavio','avaty!')
>>> a[0]
1
>>> a[3]
'flavio'
>>>
```



▪ Dicionários

- Mapping
- Estrutura mais poderosa em Python (na minha opinião ;-)
- Delimita os objetos com {}
- Par CHAVE:VALOR
 - Chave só pode ser tipos imutáveis
- Sem ordem fixa

```
>>> info = {'nome': 'Flavio', 'idade':  
21, 'interesses':  
['python', 'embedded', 'linux']}  
>>> info['idade']  
21
```



▪ Dicionários

- Gerando Iteradores
 - `dict.keys()`
 - `dict.values()`
- Adicionando pares chave: valor
- Buscando Valores
 - `dict.has_key('chave')`
- Somar dois dicionários
 - `dict1.update(dict2)`
- Apagar tudo
 - `dict.clear()`



- **Funções Embutidas**

- Int(), long(), float(), list(), dict(), tuple(), bool(), str()

- **Transformações string <> lista**

- O método join()
 - O método split()



- A função `raw_input()`
- A função `input()`
- A instrução `print`

Exercício!



Capturar uma string como entrada de dados de um usuário onde conterà seu nome, idade e profissão, todos separados por uma contra-barra. Armazenar esses dados em um dicionário e imprimir.

Ex:

Entrada: `flavio\21\programador`

Saida: `{'idade': 21, 'profissao': 'programador', 'nome': 'flavio'}`

▪ Palavras Reservadas

```
and  assert  break  class  continue  while  
def  del      elif   else   except   exec  
if   import  in      is     lambda   not  
or   pass    print  raise  return   try
```

▪ Blocos

- Identação obrigatória
 - TAB's ou 4 espaços



▪ Operadores de Comparação

- ==
- !=
- > <
- Is
- In

▪ if – elif – else

```
>>> if nota >= 7:
...     print 'Parabens.'
... elif nota >= 5:
...     print 'Voce esta na recuperacao!'
... else:
...     print 'Voce foi reprovado.'
```

▪ while

```
>>> numero = 20
>>> while numero > 10:
...     numero= input("Digite um numero: ")
...
Digite um numero: 11
Digite um numero: 12
Digite um numero: 2923929
Digite um numero: 9
>>>
```



- **For**

```
>>> for var in objeto_iteravel:
```

- **Exemplo:**

```
>>> for numero in [1,2,3,4]:  
...     print numero,  
...  
1 2 3 4  
>>>
```



▪ Mais Exemplos

```
>>> dic = {'flavio' : 21, 'theo' : 19, 'andre' : 26}
>>> for nome in dic.keys():
...     print nome, 'tem', dic[nome], 'anos.'
...
theo tem 19 anos.
andre tem 26 anos.
flavio tem 21 anos.
>>> for par in range(2,9,2):
...     print par, 'eh par.'
...
2 eh par.
4 eh par.
6 eh par.
8 eh par.
```



Exercício!



Capturar entradas de dados do usuário, checar se é inteiro ou string, e dependendo de qual for, adicionar as strings em uma lista ou somar os numeros entrados. O programa encerra quando o usuário digitar \$. Imprimir a lista de strings e a soma de numeros.

Ex:

Entrada:

1

flavio

30

ribeiro

\$

Saida:

['flavio','ribeiro']

31



- **Sintaxe:**

```
def nome_da_funcao(parametros):  
    bloco de comandos  
    return saida1,saida2
```

- **Exemplo**

```
>>> def soma(num1,num2):  
...     return num1+num2  
...  
>>> soma(3,5)  
8
```



▪ Parâmetros Opcionais

```
>>> def soma(num1, num2=10) :  
...     return num1+num2  
...  
>>> soma(3)  
13  
>>> soma(4, 10)  
14
```



▪ Inúmeros Parâmetros

```
>>> def soma(*args):  
...     return sum(args)  
...  
>>> soma(2, 3, 4, 5, 6, 7, 8)  
35  
>>> soma(2)  
2
```



Exercício!



Gerar uma função que retorna o número de parâmetros passados e uma string com todas os parâmetros concatenados como string.

Ex:

Entrada

`funcao(1,3,'flavio','avaty')`

Saída

`(4,'13flavioavaty')`



- **Conceitos necessários**

- Classe: Molde
- Objeto: Agente ativo na programação
- Método: Capacidades de ação do agente ativo
- Atributo: Características do agente ativo

- **Exemplo:**

- Classe: Humorista
- Objeto: Tiririca
- Método: Contar piadas, Imitar Pessoas
- Atributo: Baixo, 44 anos

- **Princípio Básico:** Capacidade de criar vários objetos a partir de uma mesma classe.



▪ Trazendo pro Python

- Um método tem a mesma sintaxe de uma função
 - self explícito
- Atributos recebem um prefixo self.
- Em python não existem atributos e métodos privados
 - Convenção de __ (underscores)
- Declaração de uma classe:

```
>>> class Nada(object):  
...     def faz_nada(self):  
...         print 'Objetos dessa classe\  
...         nao fazem nada.'  
...  
...
```



- **O Método Inicializador**


- O `__init__`
- Confundido com o método construtor (`__new__`)
- Exemplo:

```
>>> class Homem(object):  
...     def __init__(self):  
...         print 'Fui inicializado!'  
...  
>>> flavio = Homem()  
Fui inicializado!
```



▪ Passando Argumentos na inicialização

```
>>> class Homem(object):  
...     def __init__(self, nome):  
...         self.nome = nome  
...         print 'Fui inicializado!'  
...         print 'Meu nome eh', self.nome  
>>> flavio = Homem('Flavio')  
Fui inicializado!  
Meu nome eh Flavio  
>>> theo = Homem('Theoziran')  
Fui inicializado!  
Meu nome eh Theoziran
```



- **Criando Outros métodos**
 - **Método inicializador não necessário**

```
>>> class Jogador(object):  
...     def chutar(self, algo):  
...         print "Chutei", algo  
...     def correr(self):  
...         print "Estou Correndo!"  
...  
>>> ronaldo = Jogador()  
>>> ronaldo.chutar('bola')  
Chutei bola  
>>> ronaldo.chutar('pedra')  
Chutei pedra  
>>> ronaldo.correr()  
Estou Correndo!
```



Exercício!



Defina uma classe Humano com os atributos nome, sexo, idade e cor_do_cabelo. Faça com que o nome e a idade do objeto seja impresso ao inicializar, e crie um metodo que retorna a cor do cabelo.

Ex:

Entrada:

```
instancia1 = SuaClasse('Flavio','masculino',21,'marrom')
```

```
instancia1.getCorCabelo()
```

Saida:

Oi, sou Flavio e sou do sexo masculino.

Meu cabelo eh marrom.



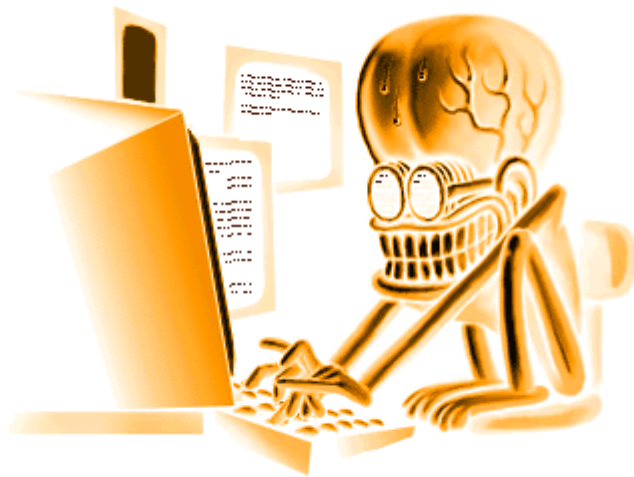
- **Herança**

- Classes new-style já usam a sintaxe de herança pra herdar object
- Python não chama o inicializador da classe base explicitamente



▪ Exemplo

```
class MembroDaFamilia(object):  
    def __init__(self,nome):  
        self.nome=nome  
        print 'meu nome eh',self.nome  
    def exhibe(self):  
        print 'sou',self.nome,'membro da  
familia.'  
  
class Pai(MembroDaFamilia):  
    def __init__(self, nome, salario):  
        MembroDaFamilia.__init__(self,nome)  
        self.salario = salario  
    def exhibe_salario(self):  
        print 'eu ganho', self.salario
```



- **Threads**
- **Sockets**
- **Extendendo Python com C**



- Threads são fluxos de execução que rodam dentro de um processo (aplicação).
- Ainda primitivo
 - Não tem stop()
 - Comunidade trabalhando nisso
 - Multiprocess
- Módulo thread
 - Método `start_new_thread(func,args)`
- Classe **threading.Thread**
 - Criar uma instância de `threading.Thread`
 - Herdar essa classe e sobrescrever o método `run()`



- **Criando uma instância de `threading.Thread`**

```
from threading import Thread  
def func(parametro):  
    print parametro
```

```
thr=Thread(target=funcao,args=('qualquercoisa',))  
thr.start()
```



- Herdar classe Thread e sobrescrever o método run()

```
from threading import Thread
import time
class Processo(Thread):
    def __init__(self,p):
        Thread.__init__(self)
        self.p = p
    def run(self):
        while True:
            print 'up and running :)'
            time.sleep(2)
```



- **Múltiplas Threads e o problema de regiões críticas**
 - **Python implementa lock's**
 - **Outra solução?**
 - **Variáveis de Travamento!**

```
from threading import Thread
```

```
from time import sleep
```

```
lock = 0
```

```
class Processo(Thread):  
    def __init__(self,p):  
        Thread.__init__(self)  
        self.p = p  
    def run(self):  
        global lock
```

```
while True:
```

```
    if lock == 0:
```

```
        lock = 1
```

```
        print 'Região
```

```
critica! Proc:', self.p
```

```
        lock = 0
```

```
    else:
```

```
        sleep(3)
```

```
        sleep(1)
```

- **Socket é o caminho mais simples de comunicação entre dois aplicativos localizados em máquinas diferentes.**
- **Stream Sockets != Datagram Sockets**
 - Stream Sockets: Conexão Constante
 - Datagram Sockets: Sem garantia



▪ Fazendo um Servidor

```
import socket
mySocket = socket.socket(socket.AF_INET, \
socket.SOCK_STREAM)
mySocket.bind(('localhost',2727))
mySocket.listen (1)
while True:
    channel, details = mySocket.accept()
    print 'Conectou: ', details
    print channel.recv ( 100 )
    channel.send('Mensagem do Servidor :)' )
    channel.close()
```



- **Fazendo um Cliente**
 - No interpretador interativo

```
import socket
s = socket.socket(socket.AF_INET, \
socket.SOCK_STREAM)
s.connect(('localhost',2727)
s.send('Enviando msg do cliente...')
s.recv(100)
s.close()
```



Exercício!



Fazer um servidor escutar 2 clientes e repassar a mensagem entre eles, gerando um chat com 2 clientes.

Sugestão: Colocar alguma flag de identificação.

Exercício!



Fazer um servidor escutar inúmeros clientes, com inúmeras conexões utilizando threads.

▪ **Coding Dojo?**



- **Existem várias maneiras de integrar C e Python**
 - Cython
 - Código híbrido
 - Python/C API
 - Interpretador embutido no código C
 - Chamada a funções de conversão de tipos C\Python
 - Main() inicializando as funções C
 - Ctypes
 - Código C puro
 - Módulo **ctypes** fazendo a 'cola'



▪ Cython em Um Exemplo

```
def primes(int kmax):  
    cdef int n, k, i  
    cdef int p[1000]  
    result = []  
    if kmax > 1000:  
        kmax = 1000  
    k = 0  
    n = 2  
    while k < kmax:  
        i = 0  
        while i < k and n % p[i] != 0:  
            i = i + 1  
        if i == k:  
            p[k] = n  
            k = k + 1  
            result.append(n)  
        n = n + 1  
    return result
```



- **Extendendo com a Python\C API**

- “Template” desenvolvido pra criação de módulos
- Disponível em:
<http://www.flavioribeiro.com/v2.0/2009/01/14/extendendo-python-com-c/>





Hora da

`random.choice([pergunta, dúvida, \n discussão])`

Obrigado! ;)

- <http://flavioribeiro.com>
- flavio.ribeiro@avaty.com.br
- flavioribeiro @ freenode #python-br



- <http://corecode.wordpress.com/2008/12/02/algoritmos-de-exclusao>
- <http://xintron.se/notebook/python-threads/>
- <http://blog.felisberto.net/2007/04/11/python-threads-and-the-gil/>
- <http://medeubranco.wordpress.com/2008/07/10/threads-em-python/>
- <http://docs.cython.org/>
- <http://corecode.wordpress.com/2008/12/02/algoritmos-de-exclusao-mutua-regioes-criticas-em-python-2/>
- <http://ldev.wordpress.com/2007/10/15/principios-de-sockets-em-python/>
- <http://www.flavioribeiro.com>

