Introdução Linguagem de Programação Python

V Jornada de Computação
Universidade Candido Mendes
Outubro/2003

Vinicius T. Petrucci
vinicius@telme.com.br

A linguagem Python

- Origem nome: *Monty Python's Flying Circus*
- Linguagem O-O de prototipação ágil
- Não apenas linguagem de script
- Fácil de usar, aprender, ler
- Extensível (adicionar novos módulos)
  - C/C++
  - Java (através Jython)
- Madura (13 anos)
- Design limpo, simples, fácil entender
  - ler como se fosse "pseudo-código"

Propriedades

- Extremamente portável
  - 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
- "Segura": sem "core dumps" causados por bugs do usuário
- Pouco punitiva: poucas regras arbitrárias; torna prazeroso seu uso

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++...

Quem está utilizando?

- Google (vários projetos)
- NASA (vários projetos)
- Yahoo! (Yahoo mail & groups)
- RealNetworks (function and load testing)
- RedHat (ferramentas instalação Linux)
- Zope Corporation (gerenciamento conteúdo)
- Apple, H.P., Microsoft
- Muitas Universidades, como MIT, e Stanford
- Jornal do Brasil, AOL Brasil

Características da Linguagem

- Tudo é objeto
- Pacotes, módulos, classes, funcões
- Tratamento exceções
- Tipagem dinâmica
- Sobrecarga de operador
- Identação para estrutura de bloco
  - O resto é sintaxe convencional

Interfaces para…

- XML
  - DOM, expat
  - XMLRPC, SOAP, Web Services
- Base de dados relacional
  - MySQL, PostgreSQL, Oracle , ODBC, Sybase, Informix
- Java (via Jython)
- Objective C
- COM, DCOM (.NET too)
- Many GUI libraries
  - cross-platform
    - Tk, wxWindows, GTK, Qt
  - platform-specific
    - MFC, Mac (classic, Cocoa), X11

Shell Interativa

- Ótimo para aprender a linguagem
- Experimentar bibliotecas
- Testar novos módulos
- Digite comandos ou expressões no prompt

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

Tipos de dados

- Números: int, long, float, complex
- Strings, Unicode: imutável
- Listas and dicionários: “containers”
- Módulos estendidos podem definir novos tipos de dados primitivos

Strings

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

Listas

- Arrays flexíveis
  - `a = [98, "bottles of beer", ["on", "the", "wall"]]`
- Mesmas operações para Strings
  - `a+b, a*3, a[0], a[-1], a[1:], len(a)`
- Item e slice
  - `a[0] = 98`
  - `a[1:2] = ["bottles", "of", "beer"]`

    `-> [98, "bottles", "of", "beer", ["on", "the", "wall"]]`
  - `del a[-1]  # -> [98, "bottles", "of", "beer"]`

Mais operações com Listas

```
>>> a = range(5)              # [0,1,2,3,4]
>>> a.append(5)               # [0,1,2,3,4,5]
>>> a.pop()                   # [0,1,2,3,4]
5
>>> a.insert(0, 42)                 # [42,0,1,2,3,4]
>>> a.pop(0)                        # [0,1,2,3,4]
5.5
>>> a.reverse()               # [4,3,2,1,0]
>>> a.sort()                        # [0,1,2,3,4]
```

Dicionários

- Hash tables, “arrays associativos"
  - `d = {"duck": "eend", "water": "water"}`
- Lookup:
  - `d["duck"] -> "eend"`
  - `d["back"] # ocorre exceção KeyError`
- Deleta, Insere, Sobrescreve:
  - `del d["water"] # {"duck": "eend", "back": "rug"}`
  - `d["back"] = "rug" # {"duck": "eend", "back": "rug"}`
  - `d["duck"] = "duik" # {"duck": "duik", "back": "rug"}`

Mais operacões com dicionários

- Chaves(keys), Valores (values), itens:
  - `d.keys() -> ["duck", "back"]`
  - `d.values() -> ["duik", "rug"]`
  - `d.items() -> [("duck","duik"), ("back","rug")]`
- Checar existência:
  - `d.has_key("duck") -> 1`
  - `d.has_key("spam") -> 0`
- Valores / Chaves vários tipos
  - `{"name":"Guido", "age":43, ("hello","world"):1, 42:"yes", "flag": ["red","white","blue"]}`

Tuplas

- `key = (lastname, firstname)`
- `point = x, y, z # parênteses opcional`
- `x, y, z = point # unpack`
- `lastname = key[0]`
- `empty = ()  # parênteses!`
- tuplas vs. listas
  - tuplas imútável

Variáveis

- Sintaxe: (underscore ou letra) + (qualquer número de dígitos ou underscores)
- Case sensitive
- 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`

Atribuição

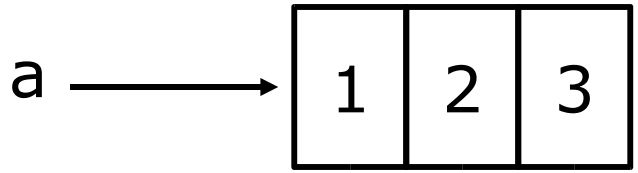
- Atribuição manipula referências
  - x = y **não faz uma cópia** de y
  - x = y faz x **referenciar** ao objeto que y referencia
- Bastante útil, mas cuidado!
- Exemplo:

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

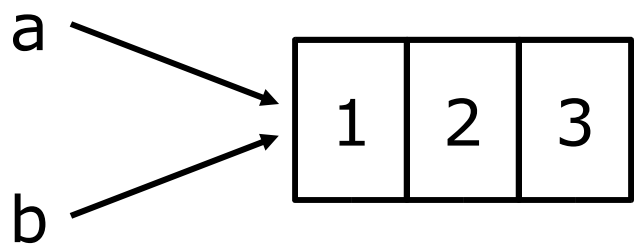
- Para obter um novo objeto, ao invés de um ponteiro para um objeto existente deve-se usar o módulo copy

Ilustração do Exemplo

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




```
b = a
```




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

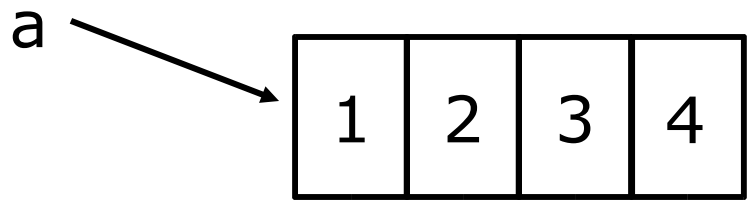
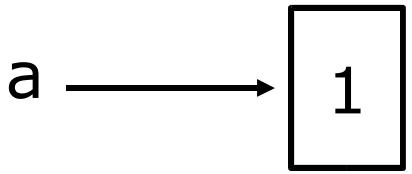
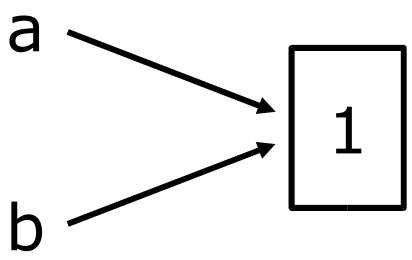


Ilustração de Outro Exemplo

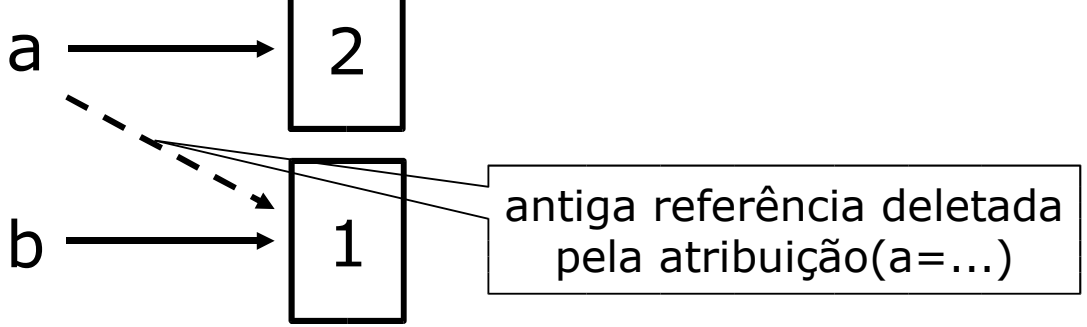
```
a = 1
```




```
b = a
```




```
a = a+1
```



Estruturas de controle

```
if condition:
    statements
[elif condition:
  statements] ...
else:
    statements
```

```
while condition:
    statements

for var in
  sequence:
    statements

break
continue
```

Identação

In Python:

```
for i in range(20):
    if i%3 == 0:
        print i
        if i%5 == 0:
            print
  "Bingo!"
   print "---"
```

In C:

```
for (i = 0; i < 20; i++)
{
    if (i%3 == 0) {
        printf("%d\n", i);
        if (i%5 == 0) {
            printf("Bingo!\n");
      }
     }
     printf("---\n");
}
```

Funções e procedimentos

```
def name(arg1, arg2, ...):
    """documentation"""# optional doc string
    statements

return                  # from procedure
return expression       # from function
```

Regras para escopo de funções

- Regra LGB
  - Referências buscam 3 escopos: local, global, built-in
  - Atribuições criam ou modificam nomes locais por default
  - Pode forçar argumentos a serem globais utilizando *global*
  - Exemplo

```
x = 99
def func(y):
    z = x+y #x não é atribuído então é global
    return z
func(1)
```

Passando argumentos para funções

- Argumentos são passados por atribuição
  - Argumentos são atribuídos a nomes locais
  - Atribuições não afetam o chamador
  - Modificando um mutável pode afetar o chamador

```
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
```

Expressões

- Chamadas para Funções
  - `spam(ham, eggs)`
- Referência para List/dictionary
  - `spam[ham]`
- Chamada para Métodos
  - `spam.ham`
  - `spam.ham(eggs)`
- Expressões Compostas
  - `spam < ham and ham != eggs`
- Testes de Intervalos
  - `spam < ham < eggs`

Módulos

- Módulos são funções definidas em arquivos separados
- Itens são importados utilizando *from ou import*

```
from module import function
function()

import module
module.function()
```

- Módulos são "namespaces"
  - Podem ser utlizados para organizar nomes de variáveis.

```
mod1.umValor = mod1.umValor - mod2.umValor
```

Exceções

- Informar que uma condição anormal ocorreu
- Formato **except**: except (tipo_excecao_1, tipo_excecao_2, ...) [, variavel_da_excecao]

```
>>> a = [ 1, 2, 3 ]
>>> try:
...     a[5]
... except IndexError:
...     print "Epa - o vetor nao possui
  tantas posicoes!"
... finally:
...     print "Fim do teste“
Epa - o vetor nao possui tantas posicoes!
Fim do teste
>>>
```

Objetos arquivos (file)

- `f = open(`*`filename`*`[, `*`mode`*`[, `*`buffersize`*`])`
  - mode can be "r", "w", "a" (tipo C stdio); default "r"
  - append "b" for text translation mode
  - append "+" for read/write open
- Alguns Métodos:
  - `read([`*`nbytes`*`]), readline(), readlines()`
  - `write(`*`string`*`), writelines(`*`list`*`)`
  - `seek(`*`pos`*`[, `*`how`*`]), tell()`
  - `flush(), close()`
  - `fileno()`

Algumas funcões 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 menos de uma string, list ou tuple

Links

- Courier New
  - `http://www.python.org`
- Estilo codificação Python:
  - `http://www.python.org/doc/essays/styleguide.html`
- Tkinter:
  - `http://www.pythonware.com/library/tkinter/introduction/`
- Projeto Jython:
  - `http://www.jython.org/`