Introdução à Linguagem Python

prof. Fernando Vanini
IC-UNICAMP
Klais Soluções

A linguagem: Python

Porque Python ?

- simples e de uso fácil
- modo interativo ou 'programado'
- disponível em praticamente todas as plataformas
- alto 'poder de expressão'
- extensível
- farta disponibilidade de bibliotecas de apoio
- fácil integração com outros sistemas
- uso disseminado

Python: conceitos básicos

- O interpretador
  - Unix: **`/usr/local/bin/python`**
  - Windows: **`C:\Python27`**

- Modo linha de comando

```
>>> print "Boa tarde!"
 Boa tarde!
 >>>
```

O interpretador Python como calculadora

- Expressões simples formadas por
  - variáveis
  - números
  - Operadores + - * / …
- Exemplos:

```
#isto é um comentário
>>> 45+3*2
51
>>> a = 10.0    # valor 10.0 (tipo float)
>>> b = 3       # valor 3    (tipo int)
>>> a/b         # o resultado é do tipo float
3.3333333333333335
```

Python: conceitos básicos

- Atribuição múltipla

```
a = b = c = 10
```

é equivalente a

```
a = 10
b = 10
c = 10
```

- Toda variável deve ser definida antes do seu uso:

```
>>> a = n  # n é indefinida

Traceback (most recent call last):
  File "<pyshell#96>", line 1, in <module>
  a = n  # n é indefinida
NameError: name 'n' is not defined
>>>
```

Python: conceitos básicos

- A variável ‘_’ mantém o último valor calculado pelo interpretador.

```
>>> taxa = 12.5 / 100  # 12.5%
>>> valor = 100.50
>>> valor * taxa       # juros
12.5625
>>> valor + _    # valor + juros
113.0625
>>> round(_, 2) # arredonda p/ 2 casas
113.06
```

Python: conceitos básicos

- *Strings*: cadeias de caracteres, delimitadas por aspas simples, duplas ou triplas (!)

```
>>> nome = "José"
>>> sobrenome = "Fernandes"
>>> print nome,sobrenome
José Fernandes
>>>
>>> # ex. operador '+' (concatenação)
>>> nomecompleto = nome+" "+sobrenome
>>> print nomecompleto
José Fernandes
>>>
```

Python: conceitos básicos

```
>>> str = "um string \
pode ser escrito em \
várias linhas"
>>> print str
um string pode ser escrito em várias linhas
>>>
```

- **Obs: nesse caso, cada linha deve ser terminada pelo caracter “\”**

Python: conceitos básicos

- O caracter “\n”:

```
>>> str = "uma linha\noutra linha"
>>> print str
uma linha
outra linha
>>>
```

Python: conceitos básicos

- Aspas triplas:

```
>>> str = """ um string
delimitado por aspas triplas
pode ocupar várias linhas """
>>> print str
um string
delimitado por aspas triplas
pode ocupar várias linhas
>>>
```

Python: conceitos básicos

- Indexação: cada caracter de um *string* pode ser acessado através do seu índice, entre '[' e ']'.
- Os índices de um *string* sempre começam em zero.

```
>>> nome = "José"
>>> sobrenome = "Fernandes"
>>> iniciais = nome[0]+"."+sobrenome[0]+"."
>>> print iniciais
J.F.
>>>
```

Python: conceitos básicos

- 'fatia' (trecho) de um string:

```
>>> nomeinteiro = "José Fernandes"
>>> nome = nomeinteiro[0:4]
>>> sobrenome = nomeinteiro[5:14]
>>> print nome,"-",sobrenome
José - Fernandes
>>>
```

- tamanho de um string:

```
>>> print len(sobrenome)
9
>>>
```

Python: conceitos básicos

- Listas
  - uma lista é uma sequência de valores (em Python, delimitada por '[' e ']').
  - Exemplo:

```
>>> primos = [2,3,5,7,11,13,17,19,23,29]
>>> primos[7]
19
>>> primos[0:5]  # 'fatia' de uma lista
[2, 3, 5, 7, 11]
>>> len(primos)  # número de elementos
10
>>>
```

Python: Operações com listas

```
>>> p1 = [3,4,5]
>>> p1.append(6)      # insere 6 ao final da lista
>>> p1
[3, 4, 5, 6]
>>> p1.insert(0,2)    # insere 2 na posição 0
>>> p1
[2, 3, 4, 5, 6]
>>> p1.remove(3)      # remove o valor 3
>>> p1
[2, 4, 5, 6]
>>> p1.pop()          # remove o último elemento
6
>>> p1
[2, 4, 5]
>>> p1.reverse()      # inverte a lista
>>> p1
[5, 4, 2]
>>>
```

Python: Operações com listas

- Os valores de uma lista
  - podem ser de qualquer tipo
  - Podem ter tipos diferentes
- Um valor de uma lista pode ser uma lista. Ex.:

```
>>> pares = [2,4,6,8]
>>> impares = [1,3,5,7,9]
>>> lista = ['um','dois',pares,'tres',impares]
>>> lista
['um', 'dois', [2, 4, 6, 8], 'tres', [1, 3, 5, 7, 9]]
>>>
```

- **Uma tabela pode ser representada como uma lista de listas**

Um 'programa' em Python

```
>>> # exemplo: série de Fibonacci
>>> a,b = 0,1
>>> while b < 100:
        print b,
        a,b = b, a+b


1 1 2 3 5 8 13 21 34 55 89
>>>
```

Python: O comando 'if'

```
>>> x = input('entre com um valor inteiro:')
>>> if x < 0:
...     print 'x é menor que zero'
... elif x == 0:
...     print 'x é igual a zero'
... else:
...     print 'x é maior que zero'
...
>>>
entre com um valor inteiro: 33
x é maior que zero
>>>
```

Python: Outro exemplo de ‘programa’

```
x = input('entre com o primeiro valor:')
y = input('entre com o segundo valor:')
while x != y:
    if x > y:
        x = x - y
    else:
        y = y - x
print 'mdc:',x

>>>
entre com o primeiro valor:10
entre com o segundo valor:35
mdc: 5
>>>
```

Python: O comando 'for'

- **O comando for opera sobre qualquer sequência de valores. Um exemplo:**

```
>>> dias = ['seg','ter','qua','qui','sex','sab','dom']
>>> for d in dias:
        print d,dias.index(d),

seg 0 ter 1 qua 2 qui 3 sex 4 sab 5 dom 6
>>>
```

Python: Comando 'for': mais exemplos

```
>>> dias = ['seg','ter','qua','qui','sex','sab','dom']
>>> for i in range(len(dias)):
        print i,dias[i],


0 seg 1 ter 2 qua 3 qui 4 sex 5 sab 6 dom
>>>
```

Python: Comando 'for': mais exemplos

```
>>> nome = 'Vasco da Gama'
>>> for c in nome: print c,

V a s c o   d a   G a m a
>>> for x in range(10,20): print x,

10 11 12 13 14 15 16 17 18 19
>>>
```

Python: 'break'

```
for n in range(2, 10):
     for x in range(2, n):
         if n % x == 0:
             print n, 'é igual a', x, '*', n/x
             break
          else: # (faz parte do 'for')
     # não achou um fator => o número é primo
     print n, 'é um número primo'

2 é um número primo
3 é um número primo
4 é igual a 2 * 2
5 é um número primo
6 é igual a 2 * 3
7 é um número primo
8 é igual a 2 * 4
9 é igual a 3 * 3
10 é igual a 2 * 5
```

Python: funções: definição e uso

```
>>> # definição da função
>>> def fib(n):
        """ Imprime a série de Fibonacci de 1 a n. """
        a, b = 0, 1
        while a < n:
                print a,
                a, b = b, a+b


>>> # chamada da função
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>>
```

Python: funções: valor de retorno

```
>>> def fib2(n):
        """ retorna uma lista uma lista contendo a
            série de Fibonacci de 1 a n. """
        res = []    # variável c/ o resultado da função
        a, b = 0,1
        while a < n:
                res.append(a)  # insere ao final da lista
                a, b = b, a+b
        return res  # retorna o resultado


>>> # chamada da função
>>> fib2(400)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377 ]
>>>
```

Python: funções: outro exemplo

```
def mdc(x,y):
    while x != y:
        if x > y:
            x = x - y
        else:
            y = y - x
    return x

def testamdc():
    a = input('entre com o primeiro valor:');
    b = input('entre com o segundo valor:');
    print 'mdc(',a,',',b,')=',mdc(a,b)

>>> testamdc()
entre com o primeiro valor: 3443
entre com o segundo valor:1234
mdc( 3443 , 1234 )= 1
>>>
```

Python: funções: outro exemplo

```
# troca dois elementos numa lista
def troca(s, i,j):
    s[i], s[j] = s[j], s[i]

# ordena uma lista (bubblesort)
def sort(lista):
    for i in range(0,len(lista)-1):
        for j in range(i,len(lista)):
            if(lista[i] > lista[j]):
                troca(lista,i,j)

>>> # exemplo de uso
>>> lista = ['seg','ter','qua','qui','sex','sab','dom']
>>> sort(lista)
>>> print lista
['dom', 'qua', 'qui', 'sab', 'seg', 'sex', 'ter']
>>>
```

Python: Tuplas

- Uma tupla é um valor composto:

```
>>> t1 = (2,4,6,8)
>>> t1[0]
2
>>> t1[3]
8
```

- Um valor de uma tupla pode ser de qualquer tipo, inclusive uma tupla :

```
>>> t2 = (0,(1,2,3),4)
>>> t3 = ([1,2,3],[4,6,8])
>>> aluno = (1200212,"Jose",34)
```

Python: 'List comprehension'

- Os valores de uma lista podem definidos através de uma a regra de formação baseada em valores de uma sequência:

```
>>> pares = [2,4,6,8]
>>> impares = [ x-1 for x in pares]
>>> cont = [ (n+1)/2 for n in impares ]
>>> novosPares = [ k*2 for k in range(1,200) ]
```

- É possível selecionar os valores da sequência:

```
>>> outros = [ x+1 for x in novosPares if x in cont ]
>>>
>>>
```

Dicionários em Python

- A estrutura de dados ‘dicionário’ (ou ‘mapa’) em Python é formada por um conjunto de pares da forma ‘chave’ e ‘valor associado’.
- Exemplos:

```
dic = { 'um':'one', 'dois':'two', 'tres':'three', 'quatro':'four'}
v = { }  # dicionario vazio
```

- O acesso a um elemento do dicionário :

```
print dic['um']     # imprime 'one'
print dic['quatro']  #imprime 'four'
```

- Acesso às chaves de um dicionário:
  - dic.keys() retorna uma lista com as chaves do dicionário ‘dic’.

```
print dic.keys() # imprime ['um','dois','tres','quatro']
```

Dicionários em Python

- Inclusão de um elemento no dicionário:

```
dic['cinco'] = 'five'
dic['seis'] = 'six'
```

  - **Importante:** ao se inserir os elementos no dicionário, a posição de inserção é definida pelas bibliotecas do Python – o programador não tem nenhum controle sobre a ordem dos elementos.
- Remoção de um elemento de um dicionário:

```
del dic['cinco']
```

- Tipos dos elementos:
  - Os elementos de um dicionário, chaves e valores, podem ser de qualquer tipo.
  - Um mesmo dicionário pode ter elementos de tipos diferentes:

```
dic = { 'um':1, 'dois': ['d','o','i','s'], 3: 'tres'}
```

Tratamento de Exceções

- Um programa em Python pode apresentar dois tipos de erros:
  - Erros de sintaxe:
    - acusados ao se tentar carregar o programa
    - Impedem a execução do programa
  - Erros em tempo de execução ou Exceções:
    - Ocorrem em função de uma situação anormal durante a execução do programa:
      - Índice de uma lista fora do intervalo válido
      - Tentativa de acesso a um arquivo inexistente
      - Acesso a um atributo ou método inexistente
      - Acesso a um elemento de um mapa através de uma chave inválida
      - Etc...

Tratamento de Exceções

- Python oferece um mecanismo que permite ao programa interceptar uma exceção e manter o controle da execução.

```
...
try
    x = lista[k+10]
    y = funcao(alfa)
except:
    print "erro no cálculo de x e y"
```

Tratamento de Exceções

- O tratamento da exceção pode ser seletivo em função do seu tipo. A linguagem define um conjunto de exceções e o programador pode definir as suas.

```
...
try
    x = lista[k+10]
    y = funcao(alfa)
except ValueError:
    print "erro no cálculo de x e y"
```

Orientação a Objetos em Python

- Python é uma linguagem Orientada a Objetos (embora o uso de objetos não seja obrigatório em qualquer caso).
- Mesmo sem utilizar explicitamente objetos, qualquer programa em Python usa a ‘base de objetos’ da linguagem:
  - Variáveis e funções
  - Tipos pré-definidos
  - Bibliotecas de apoio
- Todo valor usado pelo interpretador Python é um objeto.

Orientação a Objetos em Python

- Classes
  - Uma classe define
    - Um conjunto de valores ou atributos
    - Um conjunto de operações ou métodos sobre os valores associados
  - Uma vez definida uma classe, é possível criar 'objetos dessa classe' (ou *instâncias*):
    - Cada objeto de uma classe tem os atributos previstos na definição da mesma.
    - Cada objeto de uma classe pode ser submetido às operações definidas na classe.

Orientação a Objetos em Python

```
class Funcionario():
    empresa = "XBLUFT Ltda"
    aliq = 0.27

    def __init__(self,n,r,v):
        self.nome = n
        self.registro = r
        self.valorHora = v

    def calc(self,nh):
        return nh * self.valorHora

    def irpf(self,nh):
        return self.calc(nh) * self.aliq
```

Orientação a Objetos em Python

- Nesse exemplo
  - A classe Funcionário define
    - Os atributos: **`empresa`**, **`aliq`**, **`nome`**, **`registro`** e **`valorHora`**
    - Os métodos: **`__init__()`**, **`calc()`** e irpf()
  - O método **`__init__()`** é usado implicitamente na criação dos objetos da classe.
    - Nos métodos, o primeiro parâmetro (**`self`**) é utilizado para indicar o objeto ao qual o método se aplica.

Orientação a Objetos em Python

- Em uso:

```
def list(f,h):
    print f.empresa,f.nome,f.registro,f.calc(h),f.irpf(h)

def test():
    f1 = Funcionario("Joao","001",25.0);
    f2 = Funcionario("Jose","002",25.5);
    list(f1,180)
    list(f2,180)
```

Orientação a Objetos em Python

- Herança
  - Uma classe pode ser criada a partir de outra classe já definida
  - Nesse caso, a nova classe herda os atributos e métodos da classe original (a *classe filha* herda esses elementos da *classe mãe*).
  - Se necessário, a *classe filha* pode
    - Definir novos atributos e métodos
    - Redefinir métodos da *classe mãe*
  - *Python oferece a possib*

Orientação a Objetos em Python

- Um exemplo:

```
class Vendedor(Funcionario):
    ajCusto = 600.00

    def calc(self,nh):
        return nh*self.valorHora + self.ajCusto
```

Orientação a Objetos em Python

- Em uso:

```
def list(f,h):
    print f.empresa,f.nome,f.registro,f.calc(h),f.irpf(h)

def test():
    f1 = Funcionario("Joao","001",25.0)
    f2 = Funcionario("Jose","002",25.5)
    v1 = Vendedor("Pedro","003",23.2)
    list(f1,180)
    list(f2,180)
    list(v1,180)
```

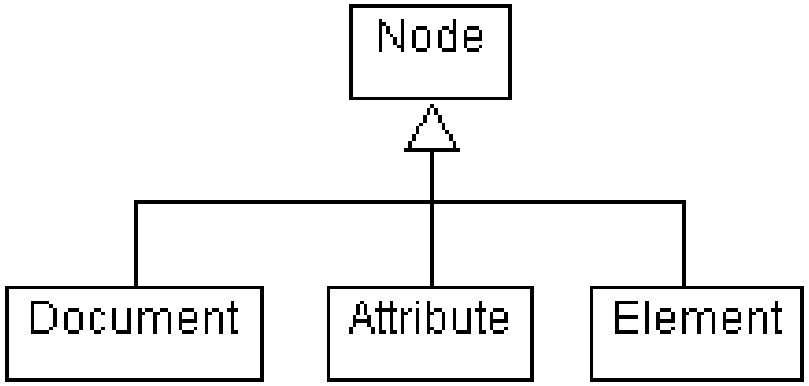
Python e XML (DOM e SAX)

- A distribuição padrão de Python oferece uma biblioteca padrão voltada ao tratamento de XML. Além disso, existem diversas implementações independentes.
- O pacote xml oferece
  - Parser operando em modo DOM e SAX.
  - xml.dom: implementação do modelo de objetos completa, de acordo com a padronização w3c.
  - xml.minidom: implementação simplificada do modelo de objetos, suficiente para a maioria das situações reais.

Python e XML (DOM)

xml.minidom: as classes (*tipos de dados*)

- **`Node`** – Representa qualquer parte de um documento XML. Pode ser o documento, uma subtag ou um atributo.
- **`Document`** – representa o 'elemento raiz' da estrutura que descreve o documento.
- **`Element`** – usado para representar as subtags do documento XML.
- **`Attribute`** – representa um atributo de uma tag.



Python e XML (DOM)

xml.minidom: principais funções:

- **`xml.dom.minidom.parse(`*`datasource`*`)`**
  - Faz o reconhecimento do texto xml disponível em datasource. Retorna um objeto **`Document`**, que representa a raiz da estrutura DOM descrita pelo texto em ***`datasource`***.
- Exemplo de uso:

```
import xml.dom.minidom
doc = xml.dom.minidom.parse("books.xml")
print doc.toxml()
```

Python e XML (DOM)

- O texto xml: “books.xml”

```
<catalog>
        <book isbn="1-56592-724-9">
                <title>The Cathedral &amp;amp; the Bazaar</title>
                <author>Eric S. Raymond</author>
                <publisher>Wiley</publisher>
                <year>1998</year>
        </book>
        <book isbn="1-56592-051-1">
                <title>Making TeX Work</title>
                <author>Norman Walsh</author>
                <publisher>O'Reilly</publisher>
                <year>2001</year>
        </book>
<!-- imagine more entries here... -->
</catalog>
```

Python e XML (DOM)

xml.minidom: principais funções:

- **`getElementsByTagName(`*`name`*`):`**
  - Função aplicada a um objeto **`Document`**, retorna a lista de objetos **`Element`** cujo nome é igual a ***`name`***.

- **`getAttribute(`*`name`*`):`**
  - Função aplicada a um objeto **`Node`**, retorna o string correspondente ao valor do atributo desse objeto cujo nome é igual a ***`name`***.

Python e XML (DOM)

Exemplo de uso:

```
import xml.dom.minidom
doc = xml.dom.minidom.parse("books.xml")
lista = doc.getElementsByTagName("book")
for node in lista:
      isbn = node.getAttribute("isbn")
      print "ISBN=",isbn
```

Python e XML (DOM)

- minidom: acesso ao texto contido numa tag

```
<publisher>Wiley</publisher>
```

  - O objeto Node tem o campo **`nodeType`** que indica o tipo do mesmo (valores possíveis: **`DOCUMENT_NODE, ELEMENT_NODE e TEXT_NODE`**)
  - O objeto Node , cujo tipo é TEXT_NODE tem um campo data, que contém o string correspondente ao texto associado ao mesmo.

- Exemplo de uso:

```
for n in tag.childNodes:
        if n.nodeType == Node.TEXT_NODE:
            text += n.data
```

Python e XML: exemplo completo(1)

```
import xml.dom.minidom
from xml.dom.minidom import Node

# devolve o texto associado a uma determinada tag
def getText(tag):
    text = ""
    for n in tag.childNodes:
        if n.nodeType == Node.TEXT_NODE:
            text += n.data
    return text

# devolve o texto associado à
# primeira subtag que tenha um dado nome
def getTextByTagName(tag,name):
    for n in tag.getElementsByTagName(name):
        return getText(n)
```

Python e XML: exemplo completo(2)

```
## teste para o arquivo 'books.xml'
def domTest():
    doc = xml.dom.minidom.parse("books.xml")
    for node in doc.getElementsByTagName("book"):
        isbn = node.getAttribute("isbn")
        title = getTextByTagName(node,"title")
        author = getTextByTagName(node,"author")
        publisher = getTextByTagName(node,"publisher")
        year = getTextByTagName(node,"year")
        print "<<",isbn,title,author,publisher,year,">>"
```

Python e XML (SAX)

- À medida em que o texto XML é analisado pelo parser XML este gera eventos que são passados à aplicação .
- O tratamento de eventos por parte da aplicação se baseia nos conceitos de orientação a objetos de Python, apresentados a seguir.

Python e XML (SAX)

- Para uso do parser XML é necessário criar uma classe derivada de **`xml.sax.handler.ContentHandler`**
- Essa classe prevê métodos que são chamados à medida em que as partes do documento XML são reconhecidas pelo parser:
  - Início de documento e fim de documento
  - Início de uma tag interna e fim de tag interna
  - Texto entre tags
- A nova classe deve redefinir os métodos necessários ao processamento desejado e um objeto dessa classe deve ser associado ao parser antes de processar o texto xml.

Python e XML (SAX) – um exemplo

```
import xml.sax.handler
class simpleSax(xml.sax.handler.ContentHandler):
    spaces = "    "

    def __init__(self):
        self.indent = ""

    def startElement(self, name, attributes):
        print self.indent,name
        self.indent += self.spaces

    def endElement(self, name):
        z = len(self.indent)-len(self.spaces)
        self.indent = self.indent[0:z]
```

Python e XML (SAX) – em uso

```
def test(file):
    parser = xml.sax.make_parser( )
    handler = simpleSax()
    parser.setContentHandler(handler)
    parser.parse(file)

test('cds.xml')
```

Python e XML (SAX)

- Acesso aos atributos (um exemplo):
  - os atributos são passados como parâmetro para o método `startElement()` e podem ser acessados como um mapa, através da indexação.

```
def startElement(self, name, attributes):
        self.buffer = ""
        if name == "book":
            self.isbn = attributes["isbn"]
```

Python e XML (SAX)

- Acesso ao texto entre tags:
  - O texto entre as tags é passado como parâmetro para o método `caracters()`. No exemplo abaixo o texto é acumulado no atributo `buffer` e tratado apropriadamente nos métodos `characters()` e `endElement()`:

```
def characters(self, data):
    self.buffer += data

def endElement(self, name):
    if name == "title":
        self.title = self.buffer
    elif name == "author":
        self.author = self.buffer
```

Python e a Web

- Python oferece diversos recursos para acesso Web, tanto do lado do servidor como do lado do cliente.
- Através da biblioteca **`urllib`**, por exemplo, é possível acessar recursos da web através do protocolo HTTP.
  - As operações GET e POST do protocolo HTTP podem ser acessadas diretamente através da função urlopen():
    - urlopen(url): acessa a url passada como parâmetro através de GET, retornando o descritor para um dispositivo de entrada. Os dados disponíveis nesse dispositivo podem ser acessados através do método read(), conforme mostrado abaixo:

```
dev  =
  urllib.urlopen('http://www.ic.unicamp.br')
pagina = dev.read()
```

Python e a Web

- Um exemplo completo:

```
import urllib

def getData(url):
    u = urllib.urlopen(url)
    data = u.read()
    return data

print getData("http://www.ic.unicamp.br/~vanini")
```

Python, a Web e XML

- Um recurso da web pode ser um texto xml
- O dispositivo criado por urllib.urlopen(url) pode ser passado ao *parser* xml:

```
import urllib
from xml.dom import minidom

def getXMLData(url):
    dom = minidom.parse(urllib.urlopen(url))
    return dom

print getXMLData('http://.../exemplos/cds.xml').toxml()
```

Python+Web+XML = Web Services

- Muitos web services públicos podem ser acessados no esquema descrito até aqui. Um exemplo é o serviço de previsão do tempo, disponibilizado pelo Yahoo:

  `http://xml.weather.yahoo.com/forecastrss?p=92120`

- Esse serviço devolve a previsão do tempo para uma região definida pelo seu zip code (na url acima, o zip code é 92120).
- Ao se passar essa url numa requisição GET, ela retorna um texto xml com a previsão do tempo para os próximos dias na área solicitada. O programa exemplo wsTest1.py acessa esse serviço, extrai as informações importantes e as apresenta ao usuário, conforme mostrado a seguir:

Python+Web+XML = Web Services

- Exemplo de consulta por wsTest1.py:

```
>>> test1(92120)
==>> Yahoo! Weather - San Diego, CA 92120 <<==
     cur temp: 53
     condition: Fair
     date: 27 Mar 2010
           condition: Sunny
           high: 77
           low: 55
     date: 28 Mar 2010
           condition: Sunny
           high: 78
           low: 57
```

Python e XML-Rpc

- Python oferece bibliotecas específicas para a publicação e acesso de Web Services baseados no protocolo XML-Rpc:
  - SimpleXMLRPCServer e XMLRPCServer: voltadas à publicação de um serviço em Python.
  - xmlrpclib: voltada à criação de um cliente para acesso ao um Web Service.
- Encapsulam os detalhes de codificação e decodificação dos objetos em XML:
  - A partir da definição das funções a serem publicadas são geradas as descrições do serviço em WSDL.
  - A partir da consulta à descrição em WSDL, são criadas as funções Python para acesso ao serviço do lado do cliente.
  - Simplificam a publicação e o acesso aos serviços.

Python e XML-Rpc – Um exemplo

```
from SimpleXMLRPCServer import SimpleXMLRPCServer

dddTab = { 'CPS':19, 'SP':11, 'Rio':21, 'Itu':11, 'Paulinia':19,
   'Sumare':19 }

servidor = SimpleXMLRPCServer(('localhost', 9000), logRequests=True)

# função publicada pelo serviço
# retorna o ddd da cidade passada como parâmetro
def consulta_ddd(cidade):
    if dddTab.has_key(cidade):
        return dddTab[cidade]
    return 'cidade não registrada'
```

Python e XML-Rpc – Um exemplo

```
# registro das funcoes 'publicadas' pelo servidor
# (toda função publicada deve ser registrada)
print 'registrando as funcoes...'
servidor.register_function(consulta_ddd)

try:
    print 'Para encerrar digite Control-C '
    servidor.serve_forever()
except KeyboardInterrupt:
    print 'Encerrando o servidor'
```

Python e XML-Rpc – Um exemplo

```
# Cliente exemplo
# Testes basicos p/ xmlRpcServidorEx1
# (o servidor deve rodar em outro processo (outra
   janela) )
import xmlrpclib

serv = xmlrpclib.ServerProxy('Http://localhost:9000')

print "Cliente ativo: serv.consulta_ddd(<cidade>)"
```

Referências na Web

- Python Official Website: http://www.python.org/
- Python Tutorial: http://docs.python.org/tutorial/
- Python Beginners Guide:http://www.sthurlow.com/python/
- Python Brasil: http://www.python.org.br/
- A Beginner's Python Tutorial:http://www.sthurlow.com/python/
- Penzilla.net's Python Tutorial:http://www.penzilla.net/tutorials/python/
- Tutorialspoint:http://www.tutorialspoint.com/python/
- Google's Python Class: http://code.google.com/intl/pt-BR/edu/languages/google-python-class/set-up.html
- Python Tutorial em Português:http://www.gpr.com.br/download/python21.pdf