



Minicurso de Python

Prof. Túlio Toffolo

<http://www.toffolo.com.br>

Baseado no trabalho de Alberto A.S. Lopes
Apresentação no II SSPGCC

Sobre o Minicurso

- Referência Bibliográfica
 - Lutz, M. e Ascher, D. – Aprendendo Python, 2^a ed. Bookman
- Sugestão de leitura
 - Borges, L. E. – Python para desenvolvedores
<http://ark4n.wordpress.com/python/>
 - Pilgrim, M. – Dive into Python – <http://diveintopython.org/>
 - Tutorial do Python em <http://docs.python.org/tutorial/>
 - Demais materiais referenciados em <http://www.python.org/doc/>, em especial os documentos em português em
<http://wiki.python.org/moin/PortugueseLanguage>

A linguagem Python

- Criada e conduzida por Guido von Rossum
- Mantida atualmente pela Python Software Foundation, em um processo comunitário
 - Mudanças fundamentais são discutidas em comunidade
 - Aprovação final das mudanças é revisada pelo criador
- Curiosidade
 - O nome Python vem do grupo de humor inglês **Monty Python**

Objetivos da linguagem

- Clareza e simplicidade do código
- Portabilidade
- Multi-propósito
- Multi-paradigma
- Linguagem Dinâmica
- Interoperabilidade com outras linguagens

Clareza e Simplicidade

- Eliminação de delimitadores de bloco
- “Recuo sintático”
- Tipagem dinâmica
- Gerenciamento de memória automático
- Parâmetros default

Como é um programa em Python?

```
# -*- coding: utf-8 -*-

import sys

print u"Bem vindo ao Python, versão %s" % sys.version
print u"Digite um número inteiro:",
n = int(raw_input())
for i in range(n):
    print u"Mensagem %s" % i

print u"Obrigado, e até logo!"
```

Portabilidade

- Modelo de execução baseado em máquina virtual
- Para executar em certa plataforma, basta haver um interpretador Python disponível
 - Windows, MAC, Linux, Unix básicos
 - Jython → Python para Java
 - IronPython → Python para .NET
 - PyObjC → Python para Cocoa

Multi-propósito

- Python inicialmente foi criada para ser usada como linguagem de script de shell, no sistema operacional Amoeba
- Atualmente, pode ser usada em diversos domínios de aplicação
 - Aplicativos “desktop” (Tk/Tcl, wxPython, Jython, IronPython)
 - Aplicativos web (**Django**, Grok, etc.)
 - Web Services
- Python é usada hoje entre outros no Youtube, Google, Globo.com, etc.

Multi-paradigma

- Python suporta construções principalmente nos paradigmas de programação
 - **Imperativa (estruturada)**
 - Funções, estruturas de controle, módulos
 - **Orientada a objetos**
 - Classes, objetos
 - **Funcional**
 - Manipulação de listas

Linguagem Dinâmica

- No Python, **tudo é um objeto**
- **As variáveis atuam como ponteiros**
- No Python, tudo ocorre em tempo de execução
- Os objetos no Python possuem uma estrutura dinâmica
 - Atributos podem ser acrescentados em tempo de execução
 - Classes podem ser modificadas em tempo de execução

Interoperabilidade

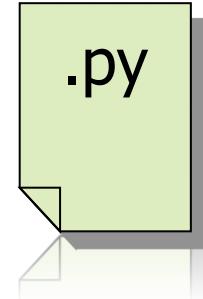
- Módulos do Python podem ser implementados em código nativo
 - E, portanto, escrito em linguagens como C++, C, etc.
- Programas em Python podem realizar chamadas ao sistema operacional, iniciar processos, etc.
- Implementações da Máquina Virtual Python em outras tecnologias podem oferecer acesso a bibliotecas
 - Programas em Python que rodam em Jython têm acesso à biblioteca padrão Java
 - O mesmo ocorre com o IronPython (bibliotecas do .NET Framework) e PyObjC (bibliotecas Cocoa)

Críticas ao Python

- **Tipagem dinâmica** dificulta a documentação e o entendimento de um sistema de maior tamanho e complexidade
- **Linguagem dinâmica** dificulta a programação com base em contratos
 - A ausência do suporte a interfaces na linguagem, por exemplo, torna menos formal a declaração de componentes
- **Ausência de bons ambientes de desenvolvimento** integrado vinha dificultando sua adoção

Execução de um programa Python

Criação do
programa
em Python



Compilação



Execução do
código

Realizados automaticamente pelo
interpretador do python

Iniciando o Python

- Interpretador interativo do Python
 - Use CTRL+D para sair (ou CTRL+Z no Windows)
 - No prompt >>> você tem duas opções
 - Digitar um comando
 - Digitar uma expressão
 - As expressões são avaliadas e o resultado é apresentado na linha abaixo, como a representação do resultado
 - Os **comandos são realmente executados**, afetando todo o ambiente de execução

Expressões básicas

- Expressões aritméticas
 - +, -, *, /
 - //
 - **
- Expressões de comparação
 - >, <, >=, <=, ==, !=
- Expressões lógicas
 - and, or, not

Literais

- Inteiros (long da linguagem C)
 - 0, 123, 4444, 231
- Longos (inteiros de precisão arbitrária - bignum)
 - 0L, 231L, 4444L
 - (o L pode ser minúsculo, mas é bom evitar)
- Ponto flutuante
 - 1.23, 444.4, 123.4e+56
- Inteiros podem ser em hexa ou octal
 - 0xFF, 0xbabaca, 0123, 02222, 010

Literais

- Números complexos
 - $100 + 2j$
- Strings (delimitados com aspas ou apóstrofos)
 - 'tulio', "toffolo"
- Booleans
 - True ou False (mas o Python considera como verdadeiro, além de True, qualquer valor diferente de 0, None, ou estruturas de dados vazias)
- Referência a nenhum objeto (equivalente a NULL)
 - None

Primeiro comando básico

- **print** (comando da linguagem)
 - A expressão depois do print é avaliada
 - Resultado é impresso na saída padrão
 - Se o print terminar com uma vírgula, não há quebra de linha ao final (só um espaço)
 - Se após o print houver expressões separadas por vírgula, todas serão impressas, separadas por espaço
- NÃO é a mesma coisa que a representação do resultado

Primeira função básica

- **raw_input** (função da linguagem)
 - É uma função cujo parâmetro é uma string (prompt)
 - O retorno é string
 - Você pode converter com as funções da linguagem int, float, bool, longpyth

Orientações básicas para escrever programas

- **# -*- coding: utf-8 -*-**
 - Coloque na primeira linha para indicar a codificação do arquivo
- Comentários
 - Comentários começam com o caracter **#** (fora de um literal string) e vão até o final da linha
- Espaços em branco e tabulações são ignorados, exceto no começo da linha (quando marcam o “recuo sintático”)
 - Lembre-se de manter o recuo sintático consistente
 - **Decida logo entre utilizar tabulações ou espaços !!!**

Orientações básicas para escrever programas

- Caso precise de mais de uma linha para escrever apenas uma “linha lógica”, coloque uma barra no final. Exemplo:
 - $x = 12 + 20 + 30 + \backslash$
 $40 + 60 + 99$
- Observações:
 - A linha lógica também continua se a linha física termina em uma lista (parâmetros de uma função, por exemplo)
 - Ou ainda, se termina em um literal string de delimitadores triplos não fechado

Orientações básicas para escrever programas

- É possível fazer atribuições utilizando tuplas
- Exemplo:

```
>>> a, b = 10, 20
```
- Qual será o valor de a?
- Qual será o valor de b?

Strings

- As strings no Python são delimitadas por apóstrofos ou aspas
 - Permite mais facilmente usar os delimitadores dentro das strings
 - Exemplo: "Eat at Joe's"
- Não pode haver quebras de linhas, a não ser que as strings sejam delimitadas por aspas ou apóstrofos triplos
 - """ Sobre este documento
O Python é uma linguagem muito boa
"""

Strings

- As strings pode ser prefixadas por ‘u’ ou ‘r’
 - u"Este é um literal unicode"
 - r"Este é um literal de string 'cru' (não interpreta \)"
 - ru"Este é um literal unicode cru"
- Strings podem ser convertidas de e para unicode com as funções internas **str** e **unicode**

Operações básicas com strings

- Concatenação
 - alfa + beta, “Hello” + ‘world’
 - Note que dois literais string justapostos subentende-se como concatenados, como “Mar” ‘portuguez’
- Repetição
 - ‘Liga’*30
- Busca simples
 - ‘dado’ in ‘soldado’

Slices

- Os strings podem ser manipulados com o operador de indexação
 - “alberto”[2] retorna ‘b’
- Mas isso não pode ser usado para alterar o string
 - Strings são consideradas imutáveis!
(será explanado mais adiante)
- Os valores de índice sempre começam com 0
- O operador de indexação no Python é muito poderoso, possuindo diversas opções:

Slices

- Para todos os exemplos a seguir, considere que:
 - alfa = "abcdefghijklmnopqrstuvwxyz"
- Obtenção de um determinado elemento
 - alfa[10] retorna 'k'
- Índices negativos: -1 é considerado o último elemento
 - alfa[-1] retorna 'z', alfa[-26] retorna 'a'
- Indicar um único índice fora do tamanho dá erro
 - alfa[300] ou alfa[-27] dão erro

Slices

- Dois índices separados por dois pontos retornam uma substring incluindo o caractere do primeiro índice e excluindo o caractere do segundo
 - alfa[0:3] retorna 'abc'
 - alfa[2:10] retorna 'cdefghij'
 - alfa[-24:-16] também retorna 'cdefghij'
- Se o primeiro índice corresponder a um caracter posterior ao do primeiro índice, nada é retornado
 - alfa[3:0] retorna ""

Slices

- Se o segundo índice é um valor além dos limites, o resultado é o mesmo de indicar a última posição
 - alfa[25:3000] retorna 'yz'
- Se o primeiro índice não for indicado, o default é a primeira posição
 - alfa[:2] retorna 'ab'
- Se o segundo índice não for indicado, o default é o índice “posterior ao da última posição”
 - alfa[20:] retorna 'uvwxyz'

Slices

- Um terceiro valor pode ser indicado como o “passo”
 - alfa[0:5:2] retorna ‘ace’
- Um passo negativo pode ser indicado para retornar a sequência invertida
 - alfa[6:2:-1] retorna ‘gfed’
 - Repare que agora o primeiro índice deve posterior ao segundo
 - E que o último caracter retornado corresponde à posição posterior à do segundo índice
 - Invertem também os defaults dos dois primeiros índices

Pergunta

- O que retorna:
 - `alfa[::]`
 - `alfa[:]`
 - `alfa[]`
 - `alfa[10:]`
 - `alfa[10::]`
 - `alfa[:10:]`
 - `alfa[2::2]`
 - `alfa[:2:2]`
 - `alfa[:2:-2]`

Formatação de strings

- O Python usa o operador % para a operação formatação de string
 - O operando do lado esquerdo é uma string de formato
 - O operando do lado direito é um objeto ou uma tupla de objetos (uma lista de objetos entre parênteses, separada por vírgulas)
 - O resultado é uma string no qual o valor do objeto é inserido no string de formato, em posições marcadas por sequências do tipo % β , onde β é um código de formatação
- Funciona igual ao **printf**

Exemplos comuns de códigos de formatação

| Símbolo | Descrição |
|---------|---|
| %s | str(objeto) |
| %d | Número inteiro |
| %x | Inteiro no formato hexa |
| %f | Float (muito usado com o número de casas: <code>%.2f</code> formata para duas casas decimais) |

```
# -*- coding: utf-8 -*-
peso = 85.542
idade = 25
nome = u"Túlio"
print "%s tem %d anos e pesa %.1fkg" % (nome, idade, peso)
```

Funções úteis com strings

| Nome | Descrição |
|---------|---|
| str | Converte um valor em string |
| unicode | Converte um valor em unicode |
| repr | Retorna a representação em string do objeto |
| len | Tamanho do string (ou da lista, da tupla, etc.) |

Estruturas de controle

- Principais estruturas de controle de código:
 - **if, elif, else, while, for**
- As estruturas utilizam a identação e operadores de comparação:

```
# -*- coding: utf-8 -*-
a = 1
if (a == 1):
    print "a = 1"
elif (a == 2 or a == 3):
    print "a = 2 or a = 3"
else:
    print u"condição else"
```

```
# -*- coding: utf-8 -*-
i = 0
while (i < 10):
    printf "i = %d" % i
for j in range(10):
    printf "j = %d" % j,
```

Estruturas de controle

- A função **range** gera uma lista contendo os valores passados por parâmetro. Testem:
 - range(10), range(0, 10) e range(2, 10)
 - range(0, 100, 5)
- Obtendo o n-ésimo item da sequência de Fibonacci:

```
# -*- coding: utf-8 -*-
n = 10 # n-ésimo item da sequência
a, b = 0, 1
for i in range(n):
    a, b = b, a+b
print "fib[%d] = %d" % (n, b)
```

Valores como objetos

- No Python **tudo é um objeto**
- Todos os valores, mesmo números, são objetos
- Um objeto é uma máquina de computação
 - Possui estado
 - Realiza determinadas computações
- Cada objeto é de um tipo
 - O tipo do objeto determina quais as informações que compõem o objeto
 - O tipo também determina as computações que o objeto pode executar

Valores como objetos

- Os objetos possuem atributos
 - Um atributo a de um objeto b nada mais é do que um objeto b associado a a
- Pode-se usar o operador ponto para identificar atributos de um objeto
 - Por exemplo, os objetos do tipo int possuem os atributos real e imag, indicando os valores das partes real e imaginária:
 - >>> a = 10
 - >>> a.imag
 - 0
 - >>> a.real
 - 10

Referências

- As variáveis guardam apenas referências a objetos
- Se uma variável aponta para um objeto, e receber uma atribuição de outro objeto, perde a referência do primeiro objeto
- Quando nenhuma variável apontar para um certo objeto, este objeto será excluído pelo **garbage collector**
 - Garbage collector = coletor de lixo
 - Python faz todo o gerenciamento de memória automaticamente

Referências

- **Repetindo**: a variável guarda sempre a referência (i.e., o endereço na memória) de um objeto
 - $a=1$ aloca o objeto 1 na memória, e guarda a referência dele na variável a
 - $b = a$ coloca em b a mesma referência que há em a
 - $b = 11$ aloca o objeto 11 na memória, e guarda a referência dele na variável b
- O prompt interativo sempre mostra a representação do valor do objeto apontado pela referência indicada (sempre imprime o valor e não o ponteiro)

Objetos imutáveis

- Alguns objetos são de **tipos imutáveis**
- As strings e os números são do tipo imutável
 - Não é possível alterar, acrescentar ou tirar atributos deles
- Exemplo:
 - alfa = 'Tulio'
alfa[0] = 'J' **retorna um erro: impossível fazer a atribuição**
- O que se pode fazer é obter novos objetos imutáveis por meio de operações, e atribuí-los às variáveis.

Exemplo de operações com objetos imutáveis

- Alterando o item na posição 0 de uma string

```
# -*- coding: utf-8 -*-
alfa = u'Túlio'
alfa = 'J' + alfa[1:]
```

- Alterando o item na posição 1 de uma string

```
# -*- coding: utf-8 -*-
alfa = u'Túlio'
alfa = alfa[0:1] + u'é' + alfa[2:]
```

Funções

- Funções são sempre iniciadas pela palavra **def** seguida do nome da função e seus argumentos:
- Uma função pode retornar um ou vários valores (tuplas)
- Exemplo:

```
def metade(x):  
    return x/2  
  
def metade_e_dobro(x):  
    return x/2, x*2
```

Funções: criando uma função Fibonacci

- Crie uma função que imprime os n primeiros elementos da sequência de fibonacci
- Teste a função para n = 10

```
# -*- coding: utf-8 -*-
def fib(n):
    a, b = 0, 1
    for i in range(n):
        print "fib[%d] = %d" % (i, b)
        a, b = b, a+b
fib(10) # executando a função fib com n = 10
```

Métodos de um objeto

- Conceitualmente, um método de um objeto é um procedimento que o objeto executa, ao receber uma mensagem correspondente
- Métodos são acessados com o operador `.`
- O método `dir()` retorna uma lista com todos os métodos disponíveis para um objeto
 - `a = 1`
 - `dir(a)`
 - `dir(1)`
 - `dir('Tulio')`

Exemplo: métodos úteis em strings

- S.capitalize()
- S.ljust(width [, fill])
- S.center(width [, fill])
- **S.lower()**
- S.count(sub [, start [, end]])
- S.lstrip([chars])
- S.decode([encoding[,errors]])
- S.encode([encoding [,errors]])
- S.maketrans(x[, y[, z]])
- S.endswith(suffix [, start [, end]])
- S.partition(sep)
- S.expandtabs([tabsize])
- **S.replace(old, new [, count])**
- **S.find(sub [, start [, end]])**
- S.rfind(sub [,start [,end]])
- S.format(fmtstr, *args, **kwargs)
- S.rindex(sub [, start [, end]])
- S.index(sub [, start [, end]])
- S.rjust(width [, fill])
- S.isalnum()
- S.rpartition(sep)
- S.isalpha()
- S.rsplit([sep[, maxsplit]])
- S.isdecimal()

Exemplo: métodos úteis em strings

- `S.rstrip([chars])`
- `S.isdigit()`
- `S.split([sep [,maxsplit]])`
- `S.isidentifier()`
- `S.splitlines([keepends])`
- `S.islower()`
- `S.startswith(prefix [, start [, end]])`
- `S.isnumeric()`
- **`S.strip([chars])`**
- `S.isprintable()`
- `S.swapcase()`
- `S.isspace()`
- `S.title()`
- `S.istitle()`
- `S.translate(map)`
- `S.isupper()`
- **`S.upper()`**
- **`S.join(iterable)`**
- `S.zfill(width)`

Listas

- As listas são objetos que guardam valores indexados por números (iniciados em 0) inteiros
- O tamanho da lista é variável e listas podem ser aninhadas (uma dentro da outra)
- Em cada posição da lista está a referência a um objeto de qualquer tipo
- É possível ter em uma única lista valores inteiros, booleanos e strings, por exemplo

Listas

- Os literais de lista são delimitados entre colchetes
 - [1,'a',True]
- Os operadores len, concatenação (+), repetição (*), e slices podem ser utilizados em listas
- Diferentemente das strings, as **listas são mutáveis**
 - As listas podem inclusive ser alteradas por atribuição a slices

Listas

- Exemplo
 - $a = [1, 'b', \text{True}, [2,3]]$
 - Veja que listas podem ser aninhadas (uma dentro da outra)
 - $a[0]$ retorna o quê?
 - $a[0][1]$ retorna o quê?
 - $a[3]$ retorna o quê?
 - $a[3][1]$ retorna o quê?
 - Faça $a[0] = [5,6,7]$ e observe o que acontece
 - Faça $a[0:0] = [5,6,7]$ e veja o que acontece

Operações de interesse para listas

- Métodos:
 - **sort** – ordena os itens
 - **append** – anexa itens
 - **extend** – anexa itens de uma lista ao final da lista
 - **pop** – retira o último elemento e o retorna
 - Dica: **pop** e **append** implementam uma pilha !!!
- Comando **del** – exclui um elemento
- Qual seria a diferença entre **append** e **extend** quando o parâmetro é uma lista?

Cuidado! Alerta!

- Alterar o valor de uma posição da lista (a[0] por exemplo) altera **apenas a lista**
 - a = "alfa"
 - b = [0,1,a,'c']
 - b[2] = 'd'
- Qual fica sendo o valor de **b**?
- E o valor de **a**?



Funções: criando uma função Fibonacci

- Crie uma função que retorna os n primeiros elementos da sequência de fibonacci
- Teste a função para n = 10

```
# -*- coding: utf-8 -*-

def fib(n):
    r = []
    a, b = 0, 1
    for i in range(n):
        r.append(a)
        a, b = b, a+b
    return r

fib(10) # executando a função para n = 10
```

Dicionários

- Dicionários são estruturas de dados que associam pares de objetos – chave e valor
- Exemplo:
 - $d1 = \{\text{'nome': 'Alberto', 'sobrenome': 'Lopes'}\}$
 - $d2 = \{\text{'a': 1, 2: 'b'}\}$
- Repare que qualquer objeto pode ser chave e qualquer objeto pode ser valor

Dicionários

- A indexação pode ser feita colocando o valor da chave entre parênteses
 - `d1['nome']` retorna 'Alberto'
 - `d2[2]` retorna 'b'
- Não é possível fazer *slicing* em um dicionário!
- Note que o dicionário não é ordenado

Dicionários

- É possível alterar um valor usando a atribuição a uma chave específica:
 - `d3[2] = 'alfa'`
 - Com isso, é possível também inserir chaves inexistentes no dicionário:
`d3['nova_chave'] = 'novo_valor'`
- Com o método **keys** retorna-se a lista de chaves
- Com o método **values** retorna-se a lista de valores

Dicionários

- O método `get` pode ser usado para recuperar o valor de uma chave, com a vantagem de não dar erro se a chave não existir
 - `d2.get(3, "nada")` retorna “nada” se a chave 3
 - `d2[3]` causa um erro se a chave 3 não existir
- O método `update` “concatena” o dicionário passado como parâmetro ao dicionário chamado
- Também é possível criar um dicionário com a função `dict`
 - `d = dict(nome="Alberto", sobrenome="Lopes")`
 - Aqui a desvantagem é que as chaves precisam ser strings

Tuplas

- São sequências imutáveis de objetos indexáveis pelo deslocamento (como nos strings e nas listas)
- Funcionam basicamente como as listas, exceto por serem imutáveis (não podem ser modificadas)
- Exemplo:
 - $a = 10$
 - $b = 20$
 - $c = 30$
 - $x = a,b,c$
 - x será uma tupla com três itens: $(10, 20, 30)$

Ler arquivos

- Utilizamos a função **open**, que funciona de forma análoga a **fopen** em C. Métodos:
 - **readline** – lê uma linha
 - **read** – lê todo o arquivo
 - **write** – escreve no arquivo
 - **close** – fecha o arquivo
- Exemplo:

```
i = 100
x = open('arquivo.txt', 'w')
x.write('imprime texto e o inteiro "%d" no arquivo' % i)
x.close()
```

Palavras reservadas

- Palavras reservadas:
 - and, del, from, not, while, as, elif, global, or, with, assert, else, if, pass, yield, break, except, import, print, class, exec, in, raise, continue, finally, is, return, def, for, lambda, try
- Alguns formatos de identificadores são usados de forma especial pelo interpretador:
 - `_` é o resultado da última avaliação no interpretador interativo
 - `__*__` nomes de sistema
 - `__*__` membros privados das classes

Mais sobre referências

- Dois objetos podem ser comparados de duas formas
 - `a == b`: verdadeiro se `a` e `b` tiverem o mesmo valor
 - `a is b`: verdadeiro se `a` e `b` são o mesmo objeto
- Isso é especialmente importante para objetos mutáveis
- Listas, strings, etc. podem ser comparados usando `<`, `>`, etc.
- Lembre-se de que `None` é um valor que significa “referência para objeto nenhum”

Mais técnicas de iteração

- Vimos o uso do for com sequências
 - `for i in [1,3,5,8,3]: print i`
 - `for i in range(5): print i`
- É possível ainda ter mais de uma variável de iteração
 - `for a,b,c in [[1,2,3],'456',(7,8,9)]: print b`
 - `For a,b in range(5), range(5,10): print a, b`

Mais técnicas de geração de listas

- Seja:
 - `lis = [10, 20, 5, 11, 100, 200, 1000]`
- Se quisermos gerar uma lista com todos os valores menores do que 30 multiplicados por 3, fazemos:

```
r = []
for x in lis:
    if x < 30:
        r.append(x*3)
```

- Ou podemos fazer simplesmente

```
r = [x*3 for x in lis if x < 30]
```



Perguntas?

Próximos minicursos

- Desenvolvendo sites usando Python e Django
 - Amanhã às 16h
- Desenvolvimento de jogos MMORPG usando Python
 - Sábado, às 16h

