
Programando em Python - Aula 1

1 Introdução

2 Por que Python

3 Hello World

Tudo começa com um *hello world*, e nossa aula não é diferente.

Como Python é uma linguagem interpretada, podemos executar um programa interativamente, sendo um ótimo meio para fazer cálculos avançados sem precisar abrir uma planilha de cálculo ou até mesmo fazer protótipos rápidos. Nosso primeiro código será direto no interpretador. Primeiro, inicie o interpretador Python, geralmente é só chamar **python**. Ele deverá mostrar algo assim na sua tela:

```
[gustavo@gustavo] ~/aulas_python/src$ python
Python 2.3.3a0 (#2, Nov 20 2003, 07:51:52)
[GCC 3.3.2 (Debian)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

Logo após já pode entrar com o código. Para um simples *Hello World*:

```
print "hello , world!"
```

Caso você queira o código em um arquivo, para que não precise digitar toda vez, salve o código acima em um arquivo e depois execute: **python arquivo_que_você_salvou_o_código**. Se você quiser gerar um executável, é só salvar o seguinte código em um arquivo e dar permissões de execução:

```
#!/usr/bin/env python
# -*- encoding: iso-8859-1 -*-

print "hello world!"
```

No qual a primeira linha é um comentário, comentários em Python começam com **#** e se estende até o final da linha.

Nota:

No caso da primeira linha temos algo comum em ambientes Unix, que são os comentários funcionais. Se na primeira linha de um arquivo existir **#!** seguido de um caminho para um executável, este executável será chamado e o nome do arquivo será passado como argumento. No nosso caso, o arquivo é *hello_world.py* e o Unix vai identificar a primeira linha como sendo um comentário funcional e executar da seguinte forma:

```
/usr/bin/env python hello_world.py
```

Na segunda linha temos outro comentário funcional, porém desta vez ele é usado pelo Python (e não pelo Unix, como visto na nota acima). Este comentário funcional especifica em qual codificação foi escrito o código em *hello_world.py* e tem o intuito de facilitar pessoas de outras línguas a escreverem código Python (chineses, japoneses e outros podem escrever as mensagens de texto em linguagem nativa). Este comentário funcional é utilizado a partir do Python 2.3.

Na terceira linha temos uma função interna do Python que imprime na tela, a função `print`. Você pode encontrar a documentação em <http://www.python.org/doc/current/ref/print.html>.

De agora em diante você já sabe como executar códigos Python. Fica a seu critério executar os exemplos aqui listados em modo interativo ou via arquivos.

3.1 A função print

Esta função recebe uma *string*¹ e a coloca em uma nova linha da saída padrão², isto é, após o texto dado ela envia um caractere de controle (`\n`) para o terminal de saída avançar para a próxima linha.

Caso queiramos suprimir este caractere de controle de nova linha, é só colocar uma vírgula no fim do comando, veja:

```
print "Texto SEM nova linha." ,
print "Texto COM nova linha."
```

Isto resultará em:

```
Texto SEM nova linha.Texto COM nova linha
```

A função `print` utiliza-se de um recurso das *strings* para simular o `printf` do C, C++, PHP e outras. Repetimos: este recurso é das *strings*, portanto pode ser utilizado em outros lugares. Ele tem a forma:

```
"STRING FORMATO" % ( LISTA DE VALORES )
```

No qual "STRING FORMATO" segue o padrão do `printf`, com `%s` para *strings*, `%f` para números reais (ponto flutuante), `%d` para decimais. Vide manual do Python para mais informações. Uma das principais diferenças é que o `%s` é mais relaxado, aceitando todos os tipos de conteúdo e convertendo-os para texto.

Exemplo:

```
print "decimal: '%d', flutuante: '%.2f'" % ( 1234, 1.234 )
print "string: '%s', string: '%s'" % ( "teste", 123 )
```

4 Variáveis

Em Python uma variável não tem tipo fixo, apenas o tipo do conteúdo atual. Elas podem ser declaradas (ou inicializadas) em qualquer parte do código — porém deve-se **tomar extremo cuidado para não utilizar uma variável antes desta ser sido inicializada!** Os nomes de variáveis só podem ter letras de **a** a **z**, o caractere de sublinhado, `_`, e números. Letras maiúsculas e minúsculas são consideradas diferentes! Para inicializar uma variável é só atribuir um valor a ela, por exemplo:

```
texto1 = "algum texto"
texto2 = 'outro texto'
inteiro = 123
real = 1.23
real2 = 1.23e6 # 1.23 * 10^6

lista = [ 1, 2, "teste", 1.23, inteiro ]

print "lista=%s" % lista # imprime: lista=[ 1, 2, 'teste', 1.23, 123 ]
print "lista[0]=%s, lista[1]=%s" % ( lista[ 0 ], lista[ 1 ] )
# imprime: lista[0]=1, lista[1]=2
print "pedaço da lista: %s" % lista[ 1 : 3 ]
```

¹*string* pode ser entendido como um pedaço de texto, uma cadeia de caracteres.

²O termo saída padrão é comumente referido por `stdout`, do inglês *standard output*.

```
# imprime: pedaço da lista: [ 2, 'teste' ]

dicionario = { "chave" : "valor",
               "inteiro" : inteiro,
               inteiro : "inteiro" }

print "dicionario=%s" % dicionario
# imprime: dicionario={'chave': 'valor', 123: 'inteiro', 'inteiro': 123}
print "dicionario[\"chave\"]=%s" % dicionario["chave"]
# imprime: dicionario[\"chave\"]=valor
print "dicionario[123]=%s" % dicionario[123] # inteiro == 123
# imprime: dicionario[123]=inteiro
```

5 Usando Condicionais: if ...elif ...else

Os elementos condicionais em Python funcionam tais como em outras linguagens, com a sintaxe:

```
if CONDIÇÃO:
    BLOCO DE CÓDIGO
elif CONDIÇÃO:
    BLOCO DE CÓDIGO
else:
    BLOCO DE CÓDIGO
```

veja um exemplo:

```
#!/usr/bin/env python
# -*- encoding: iso-8859-1 -*-

variavel1 = 1
variavel2 = 2

if variabel1 == 1:
    print "variavel1 igual a 1"
elif variabel1 !=1 or not variabel2 == 2:
    print "variavel1 diferente de 1 ou variabel2 diferente de 2"
elif variabel1 >= 1000 or variabel2 <= -1000:
    print "variavel1 maior que 1000 ou variabel2 menor que -1000"
else:
    print "nenhuma das alternativas anteriores"
```

Nota-se que as condições podem se utilizar de operadores == (igual), != (diferente), < (menor), > (maior), <= (menor ou igual), >= (maior ou igual) e que várias condições podem ser combinadas em uma condição maior utilizando os operadores lógicos **and** (e), **or** (ou) e **not** (negação).

Atenção:

Em Python os blocos de comandos são definidos pela indentação, portanto é muito importante ficar atento ao programar! Muitos dos erros advêm de má indentação.

6 Laços: while e for

A linguagem provê dois tipos de laços, um que testa uma condição e executa o bloco de código enquanto tal for verdadeira (**while**) e outro que interage com uma sequência (**for**). As sintaxes são:

```
while CONDIÇÃO:
    BLOCO DE CÓDIGO
```

```
for VARIÁVEL in SEQUÊNCIA:
    BLOCO DE CÓDIGO
```

Exemplo de laços:

```
#!/usr/bin/env python
# -*- encoding: iso-8859-1 -*-

for fruta in [ "banana", "maçã", "uva" ]:
    print "Fruta: " + fruta
# end for fruta

for i in range( 0, 10 ):
    print "i = " + str( i )
# end for i

i = 0
while i < 10:
    print "i = %d" % i
    i += 1
# end while i
```

Dentro de laços é possível usar dois comandos adicionais para ajudar no controle dos mesmos. Os comandos são:

- **continue** este comando vai para a próxima iteração do laço imediatamente.
- **break** este comando interrompe a execução do laço.

No exemplo utilizamos um comentário logo após o término de um laço, isto não é obrigatório porém ajuda a prevenir erros de indentação. Achemos uma boa prática e aconselhamos que utilizem-a.

7 Funções

Funções são um método de reaproveitamento e organização de código. Uma função pode receber alguns parâmetros e devolver um resultado.

Uma função é chamada pelo seu nome, seguido de parenteses contendo os parâmetros.

No exemplo sobre laços (Listagem 6) `range(0, 10)` é uma função chamada `range`, que recebe como parâmetros dois números inteiros, no exemplo passamos os valores 0 e 10. Esta função também retorna um valor, que é uma sequência de números desde 0 até 10 (não incluindo este último).

Em Python tudo são objetos, incluindo as funções. Elas tem alguns atributos, dentre os quais os mais úteis são `__doc__` o qual contém a documentação da função e `__name__` que contém o nome da função. Mas espera, pra que eu quero saber o nome da função se eu tenho que chamar ela pelo nome? Bem, você pode mudar o nome da função, por exemplo, se uma função tiver um nome muito grande, pra economizar digitação, você pode criar um novo nome apontando pra função que você quer (lembra-se que em Python tudo é um objeto?). Exemplo:

```
r = range
print r( 10 ) == range( 10 )
```

Para ver os atributos de um objeto (no caso, de uma função) utiliza-se a função `dir(OBJETO)`. Exemplo:

```
print dir( range )
for atributo in dir( range ):
    print "%s = %s" % ( atributo, getattr( range, atributo ) )
# end for atributo
```

Neste exemplo, `getattr` é uma função que recebe como primeiro parâmetro o objeto e como segundo o atributo que se deseja receber o valor, é equivalente a fazer `objeto.atributo`. Outra peculiaridade que vemos no exemplo anterior é que não importa o tipo do conteúdo dos atributos, eles são convertidos para *strings* pelo uso do `%s`, isto é feito pelo Python usando a função `str(OBJETO)`.

7.1 Criando Funções

Agora que você sabe usar funções, deve saber também como criar suas próprias. Trataremos só do básico, porém aconselhamos que você leia o manual do Python ou outros tutoriais mais avançados para saber os recursos disponíveis.

A sintaxe de criar uma função é simples:

```
def NOME DA FUNÇÃO( LISTA DE PARÂMETROS ):
    """
    DOCUMENTAÇÃO DA FUNÇÃO
    """
    BLOCO DE CÓDIGO
```

Onde **NOME DA FUNÇÃO** deve ser um nome com apenas caracteres de **a** a **z** (letras maiúsculas e minúsculas são consideradas diferentes!), o caractere de sublinhado, “_”, e números, **LISTA DE PARÂMETROS** é uma lista de nomes de variáveis separadas por vírgulas, tais variáveis estarão disponíveis no escopo da função com os valores recebidos.

O texto entre a sequência de três aspas duplas serve para documentar a função e chama-se “*docstring*”. Existem diversos aplicativos que geram a documentação em vários formatos (html, man, pdf, ...) a partir das *docstrings*. Também podemos acessar a documentação de uma função lendo o conteúdo do atributo `__doc__` da função (Vide exemplo, Listagem 7.1). Você também pode usar a função `help(função)` para obter a documentação desta.

O bloco de código pode conter várias saídas (pontos de retorno), basta somente usar o comando `return VALOR A RETORNAR` ou apenas `return`, caso não queira retornar valor algum.

Exemplo:

```
#!/usr/bin/env python
# -*- encoding: iso-8859-1 -*-

def fatorial( numero ):
    """
    Função recursiva que retorna o valor do fatorial do número dado.
    """

    if numero <= 1:
        return 1
    else:
        return ( numero * fatorial( numero - 1 ) )
# end fatorial

for n in range( 1, 11 ):
    print "Fatorial de %d é %d" % ( n, fatorial( n ) )
# end for n

print "\nDocumentação da função fatorial:\n" + fatorial.__doc__
```

Importante:

O Python passa como parâmetro para a função uma **cópia da referência para o objeto**, por isso se você mudar a referência para outro objeto, a variável passada como parâmetro não será alterada, caso você alterar o conteúdo de um objeto, a variável terá seu valor alterado. Um exemplo facilita o entendimento:

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

def nao_altera_lista( l ):
    l = [ 11, 22, 33 ]      # Muda a referência para outro
                           # objeto. Portanto não altera o
                           # conteúdo de "lista"

nao_altera_lista( lista )
print lista                # imprime: [ 1, 2, 3 ]

def altera_lista( l ):
    l.append( 123 )         # Altera o conteúdo do objeto.
                           # Portanto altera o conteúdo de
                           # "lista"

altera_lista( lista )
print lista                # imprime: [ 1, 2, 3, 123 ]
```