PYTHON e DJANGO

Desenvolvimento ágil de aplicações web

Osvaldo Santana
Thiago Galesi

Novatec

Editor: Rubens Prates
Capa: Victor Bittow
Revisão gramatical: Márcio Friedl
Editoração eletrônica: Camila Kuwabata e Carolina Kuwabata

ISBN: 978-85-7522-247-8

Histórico de impressões:

Outubro/2010 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos, 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Fax: +55 11 2950-8869
Email: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Santana, Osvaldo
Python e Django / Osvaldo Santana e Thiago Galesi . -- São Paulo : Novatec Editora, 2010.

ISBN 978-85-7522-247-8

1. Django (Recurso eletrônico) 2. Python (Linguagem de programação para computadores) 3. Web sites - Desenvolvimento I. Galesi, Thiago. II. Título.

10-08646 CDD-005.133

Índices para catálogo sistemático:

1. Python e Django : Desenvolvimento de aplicações Web : Programação : Processamento de dados 005.133

ORG20101006

CAPÍTULO 1

Introdução

Em se tratando de informática, o termo 'antigamente' refere-se a um tempo muito curto. Os primeiros computadores pessoais têm pouco mais de 30 anos. A internet como conhecemos tem pouco mais de 10 anos.

Posto isso, antigamente, havia computadores e seus aplicativos, pequenos, monousuários. Dada as limitações técnicas, tanto do hardware como dos sistemas operacionais da época, a grande maioria dos aplicativos era desenvolvida em linguagens de baixo nível (o resto era desenvolvido em linguagens de alto nível, porém pouco poderosas e muito vagarosas).

O tempo passou, a internet nasceu e, de repente, as aplicações que eram usadas por uma pessoa em seu computador passaram a ser usadas via internet, com todas as vantagens (e problemas) relacionados. Problemas que eram irrelevantes enquanto os sistemas tinham uma quantidade mínima de usuários simultâneos acabaram se tornando grandes problemas no momento em que as redes passaram a ter centenas ou até milhares de usuários ao mesmo tempo.

A complexidade dos sistemas aumentou. Entretanto, para nosso benefício, a capacidade computacional também. Assim, as linguagens interpretadas, que no início da nossa história eram lentas e sem muitos recursos, surgiram como uma alternativa poderosa às novas necessidades dos desenvolvedores que precisavam de flexibilidade sem perda de velocidade.

Ao mesmo tempo, outras necessidades e outras maneiras de se conceber sistemas foram surgindo. No início, as aplicações eram planejadas à exaustão, com os modelos tradicionais de engenharia de software; hoje, as necessidades comerciais e a grande complexidade dos sistemas favorecem os modelos de desenvolvimento ágeis.

Nesse contexto, temos Python, uma linguagem dinâmica com sua poderosa biblioteca padrão, e Django, um framework web que abstrai grande parte da complicação do desenvolvimento web. Ferramentas que combinam perfeitamente com as ideias das metodologias ágeis.

A história do Python

O desenvolvimento da linguagem Python começou em 1989 pelo holandês Guido van Rossum. O nome da linguagem origina-se do nome da série humorística britânica *Monty Python's Flying Circus*, do grupo humorístico britânico *Monty Python*. Guido queria criar uma linguagem de altíssimo nível, que agregasse características importantes de diversas linguagens de programação. Além disso, queria que essa linguagem de programação mantivesse uma sintaxe simples e clara.

Com essas ideias em mãos, iniciou o desenvolvimento de Python durante as festas de final de ano e, ao terminar, liberou o resultado como um projeto *open source*. Com o passar do tempo, o licenciamento dessa implementação referência da linguagem (chamada de CPython) foi mudando e hoje utiliza uma licença considerada compatível com a GPL, utilizada pela maioria dos softwares livres existentes.

A pronúncia correta do nome da linguagem Python é: "*Páithon*" (pode falar 't' no lugar do 'th').

Partindo para a ação

Para iniciarmos o estudo do Python, vamos implementar um programa simples e depois colocá-lo para executar. O exemplo contido no `programa1.py` implementa um pequeno jogo de adivinhação, no qual o computador sorteia um número aleatório e cabe ao usuário adivinhar que número é esse. Digite-o em seu editor de texto favorito e grave-o como `programa1.py`.

programa1.py

```
#!/usr/bin/env python                    (1)
# -*- encoding: utf-8 -*-                (2)
# programa1.py - Primeiro programa       (3)

"""
Importa o módulo random e sorteia
um número inteiro entre 1 e 100

"""
import random                            (4)

numero = random.randint(1, 100)          (5)
escolha = 0
```

```
tentativas = 0
while escolha != numero:                                    (6)
    escolha = input("Escolha um numero entre 1 e 100:")     (7)
    tentativas += 1                                         (8)
    if escolha < numero:                                    (9)
        print "O numero", escolha, "e menor que o sorteado."
    elif escolha > numero:
        print "O numero", escolha, "e maior que o sorteado."

print "Parabens! Voce acertou com", tentativas, "tentativas."   (10)
```

Linha	Explicação
Linha 1	Essa linha é usada em sistemas Unix para identificar qual será o interpretador que o sistema operacional deverá usar para executar o programa quando este tem permissão de execução e é chamado diretamente pela linha de comando (`./programa1.py`). Apesar de desnecessária na plataforma Windows, é sempre uma boa ideia mantê-la para fins de compatibilidade entre as plataformas.
Linha 2	Essa linha passou a ser exigida a partir da versão 2.3 do Python, quando utilizamos caracteres não ASCII (p. ex., caracteres acentuados) em nosso código. No meu caso, uso `UTF-8` (outra alternativa costuma ser `ISO-8859-1` ou `latin1`). Se essa linha for omitida e usarmos caracteres especiais em nosso código, o interpretador Python emitirá mensagens de aviso (*warning*).
Linha 3	Essa linha é um comentário. O Python ignora todos os caracteres à direita do caractere "`#`".
Linha 4	O comando `import random` importará o módulo Python que possui funções para geração de números aleatórios.
Linha 5	Nessa linha, chamamos a função `randint()` do módulo `random` para sortear um número inteiro entre 1 e 100 e o atribuímos à variável `numero`.
Linha 6	Entramos em um loop até que o número informado pelo usuário seja igual ao número sorteado.
Linha 7	Solicitamos a entrada do número pelo teclado com uma chamada à função `input()`. A função `input()` ficará aguardando que o usuário digite um valor e então tecle `Enter`.
Linha 8	Incrementamos o número de tentativas do usuário.
Linha 9	Verificamos se o número informado pelo usuário é maior ou menor do que o número sorteado e informamos isso a ele. O comando `print` é utilizado para imprimir uma mensagem na tela.
Linha 10	Quando a condição do loop não for mais satisfeita (o usuário acertou o número), informa-se ao usuário quantas tentativas foram necessárias para ele acertar.

Executando um programa Python

O sistema operacional Windows, atualmente, é o único que não vem com uma versão de Python instalada. Você pode obter mais informações sobre como instalar o interpretador Python em seu computador no apêndice B. Os outros sistemas operacionais mais importantes disponíveis no mercado (Linux, Mac e BSD) já possuem um interpretador Python pré-instalado.

Agora que temos o nosso primeiro programa, podemos executá-lo. Para isso, deve-se proceder da seguinte maneira:

Em Windows

Clicar duas vezes no arquivo `programa1.py` ou abrir um "prompt de comandos" e digitar:

```
C:\> c:\python26\python programa1.py
```

(o caminho pode variar conforme a versão do Python instalada)

Em Unix, GNU/Linux ou OS X

Você pode executar um programa Python por meio do terminal. Para isso, basta digitar:

```
$ python programa1.py
```

Outra alternativa é dar permissão de execução para o arquivo e depois chamá-lo diretamente do prompt:

```
$ chmod +x programa1.py
$ ./programa1.py
```

E, por último, quando se usa uma interface gráfica, basta apenas dar um duplo-clique no ícone do programa.

A execução do nosso primeiro programa terá o seguinte resultado:

```
$ python programa1.py
Informe um número entre 1 e 100: 50
Menor
Informe um número entre 1 e 100: 75
Menor
Informe um número entre 1 e 100: 87
Menor
Informe um número entre 1 e 100: 95
Maior
Informe um número entre 1 e 100: 92
```

```
Maior
Informe um número entre 1 e 100: 90
Maior
Informe um número entre 1 e 100: 88
Parabens! Voce acertou com 8 tentativas.
```

O interpretador

Agora que já sabemos fazer programas simples e executá-los, podemos nos aprofundar um pouco mais no ambiente Python.

A peça principal desse ambiente é o interpretador, uma vez que é o responsável pela execução dos nossos programas e também pela geração dos arquivos que irão conter a versão compilada para bytecode. Ao ser executado via linha de comando, ele apresenta a seguinte mensagem:

```
$ python
Python 2.7.0 (#2, Jan 1 1970, 00:00:00)
[GCC 3.4.3 19700101 (Linux Distribution)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

O sinal de prompt (>>>) indica que o interpretador Python está em modo interativo, pronto para receber comandos e processá-los:

```
>>> print "Olá mundo!"
Olá mundo!
```

Esse recurso é muito útil quando desejamos fazer teste de trechos de código antes de utilizá-los em nossas aplicações.

Para sair do interpretador, basta digitar CTRL-D (no Linux ou Mac OS X) ou CTRL-Z e ENTER (no Windows).

```
$ python
Python 2.7.0 (#2, Jan 1 1970, 00:00:00)
[GCC 3.4.3 19700101 (Linux Distribution)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> ^D
$
```

O projeto SciPy (*http://ipython.scipy.org*) desenvolveu uma versão aprimorada do interpretador Python com suporte a edição dos comandos de histórico, cores etc.

Parâmetros e variáveis de ambiente

Os parâmetros utilizados mais rotineiramente podem ser vistos em detalhes na tabela 1.1. Para obter mais informações sobre esses parâmetros, consulte a documentação oficial do Python.

Tabela 1.1 – Parâmetros do interpretador

Parâmetro	Descrição
`-h`	Exibe uma relação de todos os parâmetros e uma breve descrição de funcionalidades.
`-O`	Liga a opção de otimização do código bytecode compilado. O arquivo gerado passa a ter a extensão `.pyo` no lugar de `.pyc`.
`-OO`	Mesma função que `-O`, exceto por também remover as strings de documentação (`docstring`).
`-i`	Continua no interpretador após o término da execução do programa. Útil para verificar conteúdo de variáveis globais ao término da execução. Ignora o conteúdo da variável de ambiente PYTHONSTARTUP.
`-c` *cmds*	Executa os comandos (*cmds*) Python passados por parâmetro.
`-t`	Emite mensagens de aviso (*warning*) quando a endentação do programa é feita misturando Tab e espaços. O estilo de programação Python não recomenda essa prática (veja a PEP-008[1]).
`-u`	Desliga o buffer de saída para o terminal. Isso faz com que todo o conteúdo impresso no terminal seja exibido imediatamente.
`-E`	Ignora as variáveis de ambiente que modificam o comportamento do interpretador. Mais informações sobre essas variáveis de ambiente você encontra na tabela 1.2.
`-S`	Não adiciona o diretório de módulos *third-party* (`site-packages`) no caminho de busca de módulos.
`-v`	Mostra informações detalhadas sobre o carregamento dos módulos, a liberação e o diretório onde eles estão localizados.
`-W` *parm*	Controla o mecanismo de avisos (*warnings*) do interpretador. A opção `ignore` desabilita a exibição das mensagens de aviso; `default` serve para definir explicitamente o comportamento padrão do interpretador que exibe apenas um aviso para cada linha de código em que ocorrer o problema; `all` imprime as mensagens sempre que elas ocorrem; `module` faz o interpretador exibir o aviso apenas na primeira vez que ocorrer dentro de um módulo; `once` exibe o aviso uma única vez durante a execução do programa e `error` causa a interrupção do programa com erro sempre que um aviso ocorrer.
`-x`	Faz o interpretador ignorar a primeira linha do programa. Essa opção serve apenas para programas que serão executados em MS-DOS. Lembre-se de que o número da linha em que ocorre um erro não será exatamente aquela em que ocorreu o erro.
`-m` *módulo*	Executa o módulo como um programa.

Além dos parâmetros, o interpretador também pode ser configurado por meio de variáveis de ambiente do sistema operacional. Algumas dessas variáveis podem ser vistas na tabela 1.2; na documentação oficial da linguagem (*http://docs.python.org/using/cmdline.html*), é possível encontrar uma listagem completa.

Tabela 1.2 – Variáveis de ambiente usadas pelo interpretador

Variável	Descrição
PYTHONHOME	Define o diretório inicial em que foi instalado o Python. O Python utilizará essa variável para montar o caminho de diretórios que ele percorrerá para encontrar a sua biblioteca padrão. O caminho de busca dessas bibliotecas, por padrão, será PYTHONHOME/lib/python<*versão*>.
PYTHONPATH	Define a ordem de procura de diretórios usados para importar os módulos Python. Os diretórios devem ser separados por ":". Diretórios incorretos ou que não existam serão ignorados.
PYTHONSTARTUP	Nome de um arquivo com código Python que será executado pelo interpretador antes de exibir o prompt do ambiente interativo. Nesse arquivo, você pode definir algumas configurações para o interpretador.
PYTHONOPTIMIZE	Configurar essa variável para qualquer conteúdo é equivalente a passar o parâmetro -O para o interpretador Python. Se você usar um número *n* (inteiro), isso será equivalente a especificar -O*n* vezes.
PYTHONDEBUG	Configurar essa variável para qualquer conteúdo é equivalente a passar o parâmetro -d para o interpretador Python. Se você usar um número n (inteiro), isso será equivalente a especificar -d*n* vezes.
PYTHONINSPECT	Configurar essa variável para qualquer conteúdo é equivalente a passar o parâmetro -i para o interpretador.
PYTHONUNBUFFERED	Configurar essa variável para qualquer conteúdo é equivalente a passar o parâmetro -u para o interpretador Python.
PYTHONVERBOSE	Configurar essa variável para qualquer conteúdo é equivalente a passar o parâmetro -v para o interpretador Python. Se você usar um número n (inteiro), isso será equivalente a especificar -v*n* vezes.

Nos sistemas baseados em Linux, é possível utilizar a variável de ambiente PYTHONSTARTUP para especificar um módulo Python que habilita o suporte, a autocompletion de comandos no interpretador Python, bem como a gravação deles em um arquivo com o histórico do que foi digitado. Para habilitar essas funcionalidades:

1. Adicione a linha: `export PYTHONSTARTUP="$HOME/.pystartup"` em um arquivo interpretado pelo shell (ex. `~/.bashrc`)
2. Crie um arquivo `~/.pystartup` com o seguinte conteúdo:

```
import atexit
import os
import readline
import rlcompleter

readline.parse_and_bind("tab: complete")
historyPath = os.path.expanduser("~/.pyhistory")

def save_history(historyPath=historyPath):
    import readline
    readline.write_history_file(historyPath)

if os.path.exists(historyPath):
    readline.read_history_file(historyPath)

atexit.register(save_history)
del os, atexit, readline, rlcompleter, save_history, historyPath
```

Veja alguns exemplos de uso do interpretador:

```
$ python -c "import random; print random.randint(0, 10)"
3
$ python -c "import random; print random.randint(0, 10)"
2
$ python -c "import random; print random.randint(0, 10)"
10
```

```
$ ls -l *.html
-rw-r--r--  1 osantana  staff  2642 Jan  5 23:01 index.html
$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
localhost - - [05/Jan/2010 23:01:43] "GET /index.html HTTP/1.0" 200 -
200 OK
```

```
$ wget http://localhost:8000/index.html -O local.html
--2010-01-05 23:01:43--  http://localhost:8000/index.html
Resolving localhost... 127.0.0.1, ::1, fe80::1
Connecting to localhost|127.0.0.1|:8000... connected.
HTTP request sent, awaiting response...
Length: 2642 (2.6K) [text/html]
Saving to: `local.html'

100%[==============================>] 2,642       --.-K/s   in 0s
2010-01-05 23:01:43 (36.5 MB/s) - `local.html' saved [2642/2642]
```

```
$ ls -l *.html
-rw-r--r--  1 osantana  staff  2642 Jan  5 23:01 index.html
-rw-r--r--  1 osantana  staff  2642 Jan  5 23:01 local.html
```

```
$ python -c "print help(dir)" | more
Help on built-in function dir in module __builtin__:

dir(...)
dir([object]) -> list of strings

If called without an argument, return the names in the current scope.
Else, return an alphabetized list of names comprising (some of) the
attributes of the given object, and of attributes reachable from it.

If the object supplies a method named __dir__, it will be used;
otherwise the default dir() logic is used and returns:

    for a module object: the module's attributes.
    for a class object:  its attributes, and recursively the attributes
        of its bases.
    for any other object: its attributes, its class's attributes, and
        recursively the attributes of its class's base classes.
>>> ^D
$
```