Capítulo 2: Usando o interpretador Python

Disparando o interpretador

O interpretador é frequentemente instalado como `/usr/local/bin/python` nas máquinas onde está disponível; adicionando `/usr/local/bin` ao caminho de busca (search path) da shell de seu UNIX torna-se possível iniciá-lo digitando:

```
python
```

no console. Considerando que a escolha do diretório de instalação é uma opção de instalação, outras localizações são possíveis; verifique com seu guru local de Python ou com o administrador do sistema. (Ex.: `/usr/local/python` é uma alternativa popular para instalação.)

Em computadores com Windows, Python é instalado geralmente em `C:\Python27`, apesar de você poder mudar isso enquanto está executando o instalador. Para adicionar esse diretório ao path, você pode digitar o seguinte comando no console:

```
set path=%path%;C:\python27
```

Digitando um caractere EOF (end-of-file; fim de arquivo: `Control-D` em Unix, `Control-Z` em Windows) diretamente no prompt força o interpretador a sair com status de saída zero. Se isso não funcionar, você pode sair do interpretador digitando o seguinte: `quit()`.

As características de edição de linha não são muito sofisticadas. Sobre UNIX, quem instalou o interpretador talvez tenha habilitado o suporte à biblioteca GNU readline, que adiciona facilidades mais elaboradas de edição e histórico de comandos. Teclar `Control-P` no primeiro prompt oferecido pelo Python é, provavelmente, a maneira mais rápida de verificar se a edição de linha de comando é suportada. Se escutar um *beep*, você tem edição de linha de comando; veja o Apêndice *Edição interativa de entrada e substituição de histórico* para uma introdução às teclas especiais. Se nada acontecer, ou se ^P aparecer na tela, a opção de edição não está disponível; você apenas poderá usar o backspace para remover caracteres da linha atual.

O interpretador trabalha de forma semelhante a uma shell de UNIX: quando disparado com a saída padrão conectada a um console de terminal (dispositivo tty), ele lê e executa comandos interativamente; quando disparado com um nome de arquivo como parâmetro ou com redirecionamento da entrada padrão para ler um arquivo, o interpretador irá ler e executar o *script* contido em tal arquivo.

Uma segunda forma de rodar o interpretador é `python -c *comando* [arg] ...`, que executa um ou mais comandos especificados na posição *comando*, analogamente à opção de shell `-c`. Considerando que comandos Python frequentemente têm espaços em branco (ou outros caracteres que são especiais para a shell) é aconselhável que o *comando* esteja dentro de aspas duplas.

Alguns módulos Python são também úteis como scripts. Estes podem ser chamados usando

`python -m *módulo* [arg] ...`, que executa o arquivo fonte do *módulo* como se você tivesse digitado seu caminho completo na linha de comando.

Quando um arquivo de script é utilizado, as vezes é útil executá-lo e logo em seguida entrar em modo interativo. Isto pode ser feito acrescentando o argumento `-i` antes do nome do script.

Passagem de argumentos

Quando são de conhecimento do interpretador, o nome do script e demais argumentos da linha de comando da shell são acessíveis ao próprio script através da variável `argv` do módulo `sys`, que é uma lista de strings. Essa lista tem sempre ao menos um elemento; quando nenhum script ou argumento forem passados para o interpretador, `sys.argv[0]` será uma string vazia. Quando o nome do script for `-` (significando entrada padrão), o conteúdo de sys.argv[0] será `'-'`. Quando for utilizado `-c` *comando*, `sys.argv[0]` conterá `'-c'`. Quando for utilizado `-m` *módulo*, `sys.argv[0]` conterá o caminho completo do módulo localizado. Opções especificadas após `-c` *comando* ou `-m` *módulo* não serão consumidas pelo interpretador mas deixadas em `sys.argv` para serem tratadas pelo comando ou módulo.

Modo interativo

Quando os comandos são lidos a partir do console (tty), diz-se que o interpretador está em modo interativo. Nesse modo ele solicita um próximo comando através do *prompt primário*, tipicamente três sinais de maior (`>>>`); para linhas de continuação do comando atual, o *prompt secundário* padrão é formado por três pontos (`...`). O interpretador imprime uma mensagem de boas vindas, informando seu número de versão e um aviso de copyright antes de exibir o primeiro prompt:

```
python
Python 2.7 (#1, Feb 28 2010, 00:02:06)
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Linhas de continuação são necessárias em construções multi-linha. Como exemplo, dê uma olhada nesse comando `if`:

```
>>> o_mundo_eh_plano = 1
>>> if o_mundo_eh_plano:
...     print "Cuidado para não cair dele!"
...
Cuidado para não cair dele!
```

O interpretador e seu ambiente

Tratamento de erros

Quando ocorre um erro, o interpretador exibe uma mensagem de erro um *stack trace* (a situação da pilha de execução). No modo interativo, ele retorna ao prompt primário; quando a entrada vem de

um arquivo, o interpretador aborta a execução com status de erro diferente de zero após exibir o stack trace (Exceções tratadas por um `except` em um comando `try` não são consideradas erros neste contexto). Alguns erros são incondicionalmente fatais e causam a saída com status diferente de zero; isto se aplica a inconsistências internas e alguns casos de exaustão de memória. Todas as mensagens de erro são escritas na saída de erros padrão (standard error), enquanto a saída dos demais comandos é direcionada para a saída padrão.

Teclar o caractere de interrupção (tipicamente Control-C ou DEL) no prompt primário ou secundário cancela a entrada de dados corrente e volta ao prompt primário. [1] Provocar a interrupção enquanto um comando está em execução levanta a exceção `KeyboardInterrupt`, a qual pode ser tratada em um comando `try`.

Scripts executáveis em Python

Em sistemas UNIX, scripts Python podem ser transformados em executáveis, como shell scripts, pela inclusão desta linha no início do arquivo:

```
#! /usr/bin/env python
```

(assumindo que o interpretador foi incluído no `PATH` do usuário e que o script tenha a permissão de acesso habilitada para execução). Os caracteres `#!` devem ser os dois primeiros do arquivo. Em algumas plataformas esta linha inicial deve ser finalizada no estilo UNIX com (`'\\n'`), e não com a marca de fim de linha do Windows (`'\\r\\n'`). Observe que o caractere `'#'` inicia uma linha de comentário em Python.

Para atribuir modo executável ou permissão de execução ao seu script Python, utilize o comando `chmod` do shell do UNIX:

```
$ chmod +x meuscript.py
```

Em sistemas Windows, não há noção de um "modo executável". O instalador de Python associa automaticamente arquivos `.py` arquivos a `python.exe` para que um clique duplo sobre um arquivo Python o execute como um script. A extensão pode também ser `.pyw`; nesse caso, a janela de console que normalmente aparece é suprimida.

Codificação em arquivos de código-fonte

É possível usar codificação diferente de ASCII em arquivos de código Python. A melhor maneira de fazê-lo é através de um comentário adicional logo após a linha `#!`:

```
# -*- coding: codificacao -*-
```

Com essa declaração, todos os caracteres no código-fonte serão tratados de acordo com a codificação especificada, e será possível escrever strings Unicode diretamente, usando aquela codificação. A lista de codificações possíveis pode ser encontrada na Referência da Biblioteca Python, na seção `codecs`.

1 Um problema com o pacote Readline da GNU pode impedir isso.

Por exemplo, para escrever strings Unicode incluindo o símbolo monetário do Euro, a codificação ISO-8859-15 pode ser usada; nela símbolo do Euro tem o valor ordinal 164. Este script exibe o valor 8364 (código Unicode correspondente ao símbolo do Euro) e termina:

```
# -*- coding: iso-8859-15 -*-

currency = u"€"
print ord(currency)
```

Se o seu editor é capaz de salvar arquivos UTF-8 com *byte order mark* (conhecido como BOM), você pode usar isto ao invés da declaração de codificação. O IDLE é capaz de fazer isto se você habilitar `Options/General/Default Source Encoding/UTF-8`. Note que esta assinatura não é reconhecida por versões antigas (Python 2.2 e anteriores), e nem pelo sistema operacional para arquivos com a declaração `#!` (usada somente em sistemas UNIX).

Usando UTF-8 (seja através da assinatura ou de uma declaração de codificação), caracteres da maioria das línguas do mundo podem ser usados simultaneamente em strings e comentários. Não é possível usar caracteres não-ASCII em identificadores. Para exibir todos esses caracteres adequadamente, seu editor deve reconhecer que o arquivo é UTF-8, e deve usar uma fonte que tenha todos os caracteres usados no arquivo.

O arquivo de inicialização para o modo interativo

Quando usamos Python interativamente, pode ser útil executar uma série de comandos ao iniciar cada sessão do interpretador. Isso pode ser feito configurando a variável de ambiente `PYTHONSTARTUP` para indicar o nome de arquivo script que contém um script de inicialização. Essa característica assemelha-se aos arquivos `.profile` de shells UNIX.

Este arquivo só é processado em sessões interativas, nunca quando Python lê comandos de um script especificado como parâmetro, nem tampouco quando `/dev/tty` é especificado como a fonte de leitura de comandos (que de outra forma se comporta como uma sessão interativa). O script de inicialização é executado no mesmo namespace (espaço nominal ou contexto léxico) em que os comandos da sessão interativa serão executados, sendo assim, os objetos definidos e módulos importados podem ser utilizados sem qualificação durante a sessão interativa. É possível também redefinir os prompts `sys.ps1` e `sys.ps2` neste arquivo.

Se for necessário ler um script adicional de inicialização a partir do diretório atual, você pode programar isso a partir do script de inicialização global, por exemplo `if os.path.isfile('.pythonrc.py'): execfile('.pythonrc.py')`. Se você deseja utilizar o script de inicialização em outro script, você deve fazê-lo explicitamente da seguinte forma:

```
import os
filename = os.environ.get('PYTHONSTARTUP')
if filename and os.path.isfile(filename):
    execfile(filename)
```

Os módulos de customização

Python fornece dois hooks (ganchos) para que você possa personalizá-lo: `sitecustomize` e `usercustomize`. Para ver como funciona, você precisa primeiro encontrar o local de seu diretório site-packages de usuário. Inicie o Python e execute este código:

```
>>> import site
>>> site.getusersitepackages()
'/home/user/.local/lib/python2.7/site-packages'
```

Agora você pode criar um arquivo chamado `usercustomize.py` nesse diretório e colocar o que quiser nele. Isso afetará toda invocação de Python, a menos ele seja iniciado com a opção `-s` para desativar esta importação automática.

`sitecustomize` funciona da mesma forma, mas normalmente é criado por um administrador do sistema no diretório site-packages global, e é importado antes de `usercustomize`. Consulte a documentação do `site` para mais detalhes.

Notas