

Python: Módulos

Douglas Duarte

Módulos

- Módulos são programas feitos para serem reaproveitados em outros programas
- Eles tipicamente contêm funções, variáveis, classes e objetos que provêm alguma funcionalidade comum
- Por exemplo, já vimos que o módulo `math` contém funções matemáticas como `sin`, `exp`, etc, além da constante `pi`
- Toda a biblioteca padrão do Python é dividida em módulos e *pacotes* (veremos mais tarde)
- Alguns dos mais comuns são: `sys`, `os`, `time`, `random`, `re`, `shelve`

Escrevendo módulos

- Na verdade, qualquer programa que você escreva e salve num arquivo pode ser importado como um módulo
- Por exemplo, se você salva um programa com o nome prog.py, ele pode ser importado usando o comando `import prog`
 - Entretanto, a “importação” só ocorre uma vez
 - Python assume que variáveis e funções não são mudados e que o código do módulo serve meramente para inicializar esses elementos

Escrevendo módulos

- Após a importação de um módulo, este é compilado, gerando um arquivo .pyc correspondente
 - No exemplo, um arquivo prog.pyc será criado
 - Python só recompila um programa se o arquivo .py for mais recente que o arquivo .pyc

Exemplo (em Unix)

```
$ cat teste.py
```

```
def f():
```

```
    print "alo"
```

```
f()
```

```
$ python
```

```
...
```

```
>>> import teste
```

```
alo
```

```
>>> import teste
```

```
>>> teste.f()
```

```
alo
```

```
>>>
```

```
$ dir teste*
```

```
teste.py  teste.pyc
```

Tornando módulos disponíveis

- Em que diretório os módulos são buscados durante a importação?
 - No diretório corrente
 - Nos diretórios da lista `sys.path`
- Se for desejável especificar o local onde os módulos residem, há essencialmente duas opções
 - Alterar diretamente a variável `sys.path`
 - Alterar a *variável de ambiente* `PYTHONPATH`
 - É o método recomendável pois não requer que o programa que importará o módulo seja alterado

Exemplo

```
$ mkdir python
$ mv teste.py python/
$ cat python/teste.py
def f():
    print "alo"
f()
$ export PYTHONPATH=~/.python
$ python
Python 2.4.2 (#2, Sep 30 2005, 21:19:01)
...
>>> import teste
alo
```

A variável `__name__`

- Se um programa pode ser executado por si só ou importado dentro de outro, como distinguir as duas situações?
 - A variável `__name__` é definida para cada programa:
 - Se é um módulo, retorna o nome do módulo
 - Se é um programa sendo executado, retorna `'__main__'`
- Para saber se o código está sendo executado como módulo, basta testar:
 - `If __name__ == '__main__': código`
- Isto é útil em diversas circunstâncias
 - Por exemplo, para colocar código de teste, código para instalação do módulo ou exemplos de utilização

Exemplo

```
$ cat teste.py
```

```
def f():
```

```
    print "alo"
```

```
if __name__ == '__main__':
```

```
    f()
```

```
$ python teste.py
```

```
alo
```

```
$ python
```

```
Python 2.4.2 (#2, Sep 30 2005, 21:19:01)
```

```
...
```

```
>>> import teste
```

```
>>> print __name__
```

```
__main__
```

```
>>> print teste.__name__
```

```
teste
```

Pacotes

- São hierarquias de módulos
- Um pacote é um *diretório* que contém um arquivo chamado `__init__.py`
 - O pacote deve estar em um dos diretórios nos quais o Python busca por módulos
 - Para importar o pacote, use o nome do diretório
 - O programa correspondente ao pacote é `__init__.py`

Pacotes

- Os demais arquivos e diretórios dentro do pacote são encarados recursivamente como módulos
 - Por exemplo, se um pacote se chama `p` e contém um arquivo chamado `m.py`, então podemos importar
 - `p` (arquivo `p/__init__.py`)
 - `p.m` (arquivo `p/m.py`)
 - Semelhantemente, `p` poderia ter um outro pacote sob a forma de outro diretório contendo um arquivo `__init__.py`

Exemplo

```
$ dir python/  
pacote teste.py  
$ dir python/pacote/  
__init__.py teste2.py  
$ cat python/teste.py  
print "teste"  
$ cat python/pacote/__init__.py  
print "pacote"  
$ cat python/pacote/teste2.py  
print "teste2"  
$ python  
...  
>>> import teste  
teste  
>>> import pacote  
pacote  
>>> import pacote.teste2  
teste2
```