CAPÍTULO 06

6. Módulos

Ao sair e entrar de novo no interpretador Python, as definições anteriores (funções e variáveis) são perdidas. Portanto, se quiser escrever um programa maior, será mais eficiente usar um editor de texto para preparar as entradas para o interpretador, e executá-lo usando o arquivo como entrada. Isso é conhecido como criar um *script*. Se o programa se torna ainda maior, é uma boa prática dividi-lo em arquivos menores, para facilitar a manutenção. Também é preferível usar um arquivo separado para uma função que você escreveria em vários programas diferentes, para não copiar a definição de função em cada um deles.

Para permitir isso, o Python tem uma maneira de colocar as definições em um arquivo e então usá-las em um script ou em uma execução interativa do interpretador. Tal arquivo é chamado de *módulo*; definições de um módulo podem ser *importadas* para outros módulos, ou para o módulo *principal* (a coleção de variáveis a que você tem acesso num script executado como um programa e no modo calculadora).

Um módulo é um arquivo contendo definições e instruções Python. O nome do arquivo é o nome do módulo acrescido do sufixo `.py`. Dentro de um módulo, o nome do módulo (como uma string) está disponível como o valor da variável global `__name__`. Por exemplo, use seu editor de texto favorito para criar um arquivo chamado `fibo.py` no diretório atual com o seguinte conteúdo:

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

Agora, entre no interpretador Python e importe esse módulo com o seguinte comando:

```
>>> import fibo
```

Isso não coloca os nomes das funções definidas em `fibo` diretamente na tabela de símbolos atual; isso coloca somente o nome do módulo `fibo`. Usando o nome do módulo você pode acessar as funções:

```
>>> fibo.fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

Se pretender usar uma função muitas vezes, você pode atribui-lá a um nome local:

```
>>> fib = fibo.fib
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

6.1. Mais sobre módulos

Um módulo pode conter tanto instruções executáveis quanto definições de funções e classes. Essas instruções servem para inicializar o módulo. Eles são executados somente na *primeira* vez que o módulo é encontrado em uma instrução de importação. 1 (Também rodam se o arquivo é executado como um script.)

Cada módulo tem sua própria tabela de símbolos privada, que é usada como tabela de símbolos global para todas as funções definidas no módulo. Assim, o autor de um módulo pode usar variáveis globais no seu módulo sem se preocupar com conflitos acidentais com as variáveis globais do usuário. Por outro lado, se você precisar usar uma variável global de um módulo, poderá fazê-lo com a mesma notação usada para se referir às suas funções, `nomemodulo.nomeitem`.

Módulos podem importar outros módulos. É costume, porém não obrigatório, colocar todos os comandos `import` no início do módulo (ou script , se preferir). As definições do módulo importado são colocadas na tabela de símbolos global do módulo que faz a importação.

Existe uma variante do comando `import` que importa definições de um módulo diretamente para a tabela de símbolos do módulo importador. Por exemplo:

```
>>> from fibo import fib, fib2
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Isso não coloca o nome do módulo de onde foram feitas as importações na tabela de símbolos local (assim, no exemplo, `fibo` não está definido).

Existe ainda uma variante que importa todos os nomes definidos em um módulo:

```
>>> from fibo import *
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Isso importa todos as declarações de nomes, exceto aqueles que iniciam com um sublinhado (`_`). Na maioria dos casos, programadores Python não usam esta facilidade porque ela introduz um conjunto desconhecido de nomes no ambiente, podendo esconder outros nomes previamente definidos.

Note que, em geral, a prática do `import *` de um módulo ou pacote é desaprovada, uma vez que muitas vezes dificulta a leitura do código. Contudo, é aceitável para diminuir a digitação em sessões interativas.

Se o nome do módulo é seguido pela palavra-chave `as`, o nome a seguir é vinculado diretamente ao módulo importado.

```
>>> import fibo as fib
>>> fib.fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Isto efetivamente importa o módulo, da mesma maneira que `import fibo` fará, com a única diferença de estar disponível com o nome `fib`.

Também pode ser utilizado com a palavra-chave `from`, com efeitos similares:

```
>>> from fibo import fib as fibonacci
>>> fibonacci(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Nota

For efficiency reasons, each module is only imported once per interpreter session. Therefore, if you change your modules, you must restart the interpreter – or, if it's just one module you want to test interactively, use `importlib.reload()`,
e.g. `import importlib; importlib.reload(modulename)`.

6.1.1. Executando módulos como scripts

Quando você rodar um módulo Python com

```
python fibo.py <arguments>
```

o código no módulo será executado, da mesma forma que quando é importado, mas com a variável `__name__` com valor `"__main__"`. Isto significa que adicionando este código ao final do seu módulo:

```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

você pode tornar o arquivo utilizável tanto como script quanto como um módulo importável, porque o código que analisa a linha de comando só roda se o módulo é executado como arquivo “principal”:

```
$ python fibo.py 50
0 1 1 2 3 5 8 13 21 34
```

Se o módulo é importado, o código não é executado:

```
>>> import fibo
>>>
```

Isso é frequentemente usado para fornecer uma interface de usuário conveniente para um módulo, ou para realizar testes (rodando o módulo como um script executa um conjunto de testes).