CAPÍTULO 10

10. Um breve passeio pela biblioteca padrão

10.1. Interface com o sistema operacional

O módulo `os` fornece dúzias de funções para interagir com o sistema operacional:

```
>>> import os
>>> os.getcwd()      # Return the current working directory
'C:\\Python310'
>>> os.chdir('/server/accesslogs')   # Change current working
directory
>>> os.system('mkdir today')   # Run the command mkdir in the
system shell
0
```

Certifique-se de usar a forma `import os` ao invés de `from os import *`. Isso evitará que `os.open()` oculte a função `open()` que opera de forma muito diferente.

As funções embutidas `dir()` e `help()` são úteis como um sistema de ajuda interativa para lidar com módulos grandes como `os`:

```
>>> import os
>>> dir(os)
<returns a list of all module functions>
>>> help(os)
<returns an extensive manual page created from the module's
docstrings>
```

Para tarefas de gerenciamento cotidiano de arquivos e diretórios, o módulo `shutil` fornece uma interface de alto nível que é mais simples de usar:

```
>>> import shutil
>>> shutil.copyfile('data.db', 'archive.db')
'archive.db'
>>> shutil.move('/build/executables', 'installdir')
'installdir'
```

10.3. Argumentos de linha de comando

Scripts geralmente precisam processar argumentos passados na linha de comando. Esses argumentos são armazenados como uma lista no atributo *argv* do módulo `sys`. Por exemplo, teríamos a seguinte saída executando `python demo.py one two three` na linha de comando:

```
>>> import sys
>>> print(sys.argv)
['demo.py', 'one', 'two', 'three']
```

O módulo `argparse` fornece um mecanismo mais sofisticado para processar argumentos de linha de comando. O script seguinte extrai e exibe um ou mais nomes de arquivos e um número de linhas opcional:

```
import argparse

parser = argparse.ArgumentParser(prog = 'top',
    description = 'Show top lines from each file')
parser.add_argument('filenames', nargs='+')
parser.add_argument('-l', '--lines', type=int, default=10)
args = parser.parse_args()
print(args)
```

Quando executada a linha de comando `python top.py --lines=5 alpha.txt beta.txt`, o script define `args.lines` para `5` e `args.filenames` para `['alpha.txt', 'beta.txt']`.

10.6. Matemática

O módulo `math` oferece acesso às funções da biblioteca C para matemática de ponto flutuante:

```
>>> import math
>>> math.cos(math.pi / 4)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```

O módulo `random` fornece ferramentas para gerar seleções aleatórias:

```
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
```

```
>>> random.sample(range(100), 10)   # sampling without replacement
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random()    # random float
0.17970987693706186
>>> random.randrange(6)    # random integer chosen from range(6)
4
```

O módulo `statistics` calcula as propriedades estatísticas básicas (a média, a mediana, a variação, etc.) de dados numéricos:

```
>>> import statistics
>>> data = [2.75, 1.75, 1.25, 0.25, 0.5, 1.25, 3.5]
>>> statistics.mean(data)
1.6071428571428572
>>> statistics.median(data)
1.25
>>> statistics.variance(data)
1.3720238095238095
```

O projeto SciPy <https://scipy.org> tem muitos outros módulos para cálculos numéricos.

10.8. Data e hora

O módulo `datetime` fornece classes para manipulação de datas e horas nas mais variadas formas. Apesar da disponibilidade de aritmética com data e hora, o foco da implementação é na extração eficiente dos membros para formatação e manipulação. O módulo também oferece objetos que levam os fusos horários em consideração.

```
>>> # dates are easily constructed and formatted
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2003, 12, 2)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")
'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of December.'

>>> # dates support calendar arithmetic
>>> birthday = date(1964, 7, 31)
>>> age = now - birthday
>>> age.days
14368
```