10. 标准库简介

10.1. 操作系统接口

os 模块提供了许多与操作系统交互的函数:

```
>>> 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
```

一定要使用 `import os` 而不是 `from os import *` 。这将避免内建的 `open()` 函数被 `os.open()` 隐式替换掉，因为它们的使用方式大不相同。

内置的 `dir()` 和 `help()` 函数可用作交互式辅助工具，用于处理大型模块，如 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>
```

对于日常文件和目录管理任务， shutil 模块提供了更易于使用的更高级别的接口:

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

10.2. 文件通配符

glob 模块提供了一个在目录中使用通配符搜索创建文件列表的函数:

```
>>> import glob
>>> glob.glob('*.py')
['primes.py', 'random.py', 'quote.py']
```

10.3. 命令行参数

通用实用程序脚本通常需要处理命令行参数。这些参数作为列表存储在 sys 模块的 *argv* 属性中。例如，以下输出来自在命令行运行 `python demo.py one two three`

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

argparse 模块提供了一种更复杂的机制来处理命令行参数。 以下脚本可提取一个或多个文件名，并可选择要显示的行数:

```
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)
```

当在通过 `python top.py --lines=5 alpha.txt beta.txt` 在命令行运行时，该脚本会将 `args.lines` 设为 `5` 并将 `args.filenames` 设为 `['alpha.txt', 'beta.txt']`。

10.4. 错误输出重定向和程序终止

sys 模块还具有 *stdin* ， *stdout* 和 *stderr* 的属性。后者对于发出警告和错误消息非常有用，即使在 *stdout* 被重定向后也可以看到它们:

```
>>> sys.stderr.write('Warning, log file not found starting a new one\n')
Warning, log file not found starting a new one
```

终止脚本的最直接方法是使用 `sys.exit()` 。

10.5. 字符串模式匹配

`re` 模块为高级字符串处理提供正则表达式工具。对于复杂的匹配和操作，正则表达式提供简洁，优化的解决方案:

```
>>> import re
>>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
['foot', 'fell', 'fastest']
>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
'cat in the hat'
```

当只需要简单的功能时，首选字符串方法因为它们更容易阅读和调试:

```
>>> 'tea for too'.replace('too', 'two')
'tea for two'
```

10.6. 数学

`math` 模块提供对浮点数学的底层C库函数的访问:

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

`random` 模块提供了进行随机选择的工具:

```
>>> 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
```

`statistics` 模块计算数值数据的基本统计属性（均值，中位数，方差等）:

```
>>> 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
```

SciPy项目 <https://scipy.org> 有许多其他模块用于数值计算。

10.7. 互联网访问

有许多模块可用于访问互联网和处理互联网协议。其中两个最简单的 `urllib.request` 用于从URL检索数据，以及 `smtplib` 用于发送邮件:

```
>>> from urllib.request import urlopen
>>> with urlopen('http://worldtimeapi.org/api/timezone/etc/UTC.txt') as response:
...     for line in response:
...         line = line.decode()             # Convert bytes to a str
...         if line.startswith('datetime'):
...             print(line.rstrip())         # Remove trailing newline
...
datetime: 2022-01-01T01:36:47.689215+00:00

>>> import smtplib
>>> server = smtplib.SMTP('localhost')
>>> server.sendmail('soothsayer@example.org', 'jcaesar@example.org',
... """To: jcaesar@example.org
... From: soothsayer@example.org
...
... Beware the Ides of March.
... """)
>>> server.quit()
```

（请注意，第二个示例需要在localhost上运行的邮件服务器。）

10.8. 日期和时间

datetime 模块提供了以简单和复杂的方式操作日期和时间的类。虽然支持日期和时间算法，但实现的重点是有效的成员提取以进行输出格式化和操作。该模块还支持可感知时区的对象。

```
>>> # 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
```

10.9. 数据压缩

常见的数据存档和压缩格式由模块直接支持，包括：zlib, gzip, bz2, lzma, zipfile 和 tarfile。:

```
>>> import zlib
>>> s = b'witch which has which witches wrist watch'
>>> len(s)
41
>>> t = zlib.compress(s)
>>> len(t)
37
>>> zlib.decompress(t)
b'witch which has which witches wrist watch'
>>> zlib.crc32(s)
226805979
```

10.10. 性能测量

一些Python用户对了解同一问题的不同方法的相对性能产生了浓厚的兴趣。 Python提供了一种可以立即回答这些问题的测量工具。

例如，元组封包和拆包功能相比传统的交换参数可能更具吸引力。timeit 模块可以快速演示在运行效率方面一定的优势:

```
>>> from timeit import Timer
>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
0.57535828626024577
>>> Timer('a,b = b,a', 'a=1; b=2').timeit()
0.54962537085770791
```

与 timeit 的精细粒度级别相反， profile 和 pstats 模块提供了用于在较大的代码块中识别时间关键部分的工具。

10.11. 质量控制

开发高质量软件的一种方法是在开发过程中为每个函数编写测试，并在开发过程中经常运行这些测试。

doctest 模块提供了一个工具，用于扫描模块并验证程序文档字符串中嵌入的测试。测试构造就像将典型调用及其结果剪切并粘贴到文档字符串一样简单。这通过向用户提供示例来改进文档，并且它允许doctest模块确保代码保持对文档的真实:

```
def average(values):
    """Computes the arithmetic mean of a list of numbers.

    >>> print(average([20, 30, 70]))
    40.0
    """
    return sum(values) / len(values)

import doctest
doctest.testmod()   # automatically validate the embedded tests
```

unittest 模块不像 doctest 模块那样易于使用，但它允许在一个单独的文件中维护更全面的测试集:

```
import unittest

class TestStatisticalFunctions(unittest.TestCase):

    def test_average(self):
        self.assertEqual(average([20, 30, 70]), 40.0)
        self.assertEqual(round(average([1, 5, 7]), 1), 4.3)
        with self.assertRaises(ZeroDivisionError):
            average([])
        with self.assertRaises(TypeError):
            average(20, 30, 70)

unittest.main()  # Calling from the command line invokes all tests
```

10.12. 自带电池

Python有“自带电池”的理念。通过其包的复杂和强大功能可以最好地看到这一点。例如:

- xmlrpc.client 和 xmlrpc.server 模块使得实现远程过程调用变成了小菜一碟。 尽管存在于模块名称中，但用户不需要直接了解或处理 XML。
- email 包是一个用于管理电子邮件的库，包括MIME和其他符合 **RFC 2822** 规范的邮件文档。与 smtplib 和 poplib 不同（它们实际上做的是发送和接收消息），电子邮件包提供完整的工具集，用于构建或解码复杂的消息结构（包括附件）以及实现互联网编码和标头协议。
- json 包为解析这种流行的数据交换格式提供了强大的支持。 csv 模块支持以逗号分隔值格式直接读取和写入文件，这种格式通常为数据库和电子表格所支持。 XML 处理由 xml.etree.ElementTree， xml.dom 和 xml.sax 包支持。这些模块和软件包共同大大简化了 Python 应用程序和其他工具之间的数据交换。
- sqlite3 模块是 SQLite 数据库库的包装器，提供了一个可以使用稍微非标准的 SQL 语法更新和访问的持久数据库。
- 国际化由许多模块支持，包括 gettext， locale，以及 codecs 包。