迭代器与数组

续本达

清华大学 工程物理系

2024-07-05 清华

课前准备

- 安装 NumPy

```
apt update
apt install python3-numpy
```

- 验证是否安装成功

```
python3 -c "from numpy import version; print(version.version)"
```

```
1.24.2
```

复合数据结构

- 列表、元组、字典
- `defaultdict` 、 `Counter` 、 `namedtuple`
- 迭代器可以逐个访问它所包含的值的。 Python 的复合数据结构都可以看成迭代器。

函数

- 定义，名字空间
- 局部变量，全局变量（不推荐！用类的变量替代）

为什么

- Python 的内部实现
  - 一切都是对象

```
help(1)
```

给出一的是 help(int)，整数是一个类。

  - 对象的要素：封装、继承、多态
  - 迭代器：定义了 __iter__() 方法的类。
  - 运算：

+	__add__
-	__sub__

- 替代全局变量

极简例子

```
class status(object): # 继承自 "object"
    '''
    状态记录器
    '''
    def __init__(self, move=0, blank=(1,2,3)):
        self.move = move
        self.blank = blank
s = status(1)
print(s.move)
s.move = 3 # 一切成员都是 public
print(s.move)
```

```
1
3
```

- 将 s 传递给函数，避免使用全局变量

简介

- 函数用来代码复用（一次 原则）
- 模块：相关的函数和类集合起来，整理到名字空间 namespace 中
  - 模块可以用 Python 实现，也可以由 C 等编译语言实现

模块导入用 import

```
import math
help(math.factorial)
```

```
Help on built-in function factorial in module math:

factorial(x, /)
    Find x!.

    Raise a ValueError if x is negative or non-integral.
```

模块别名

- 加载模块时，可自定义名称。对长模块名有用

```
import math as m
m.factorial(10)
```

```
3628800
```

多层名字加载

1. 直接使用多层名字空间
2. 使用 `from`

```
import os
help(os.path.abspath)
```

```
Help on function abspath in module posixpath:

abspath(path)
    Return an absolute path.
```

```
from os.path import abspath
from os.path import abspath as absp
abspath is os.path.abspath, abspath is absp
```

```
(True, True)
```

自定义模块

- Python 可以方便地定义模块以进行代码复用
- 每个 Python 脚本都可以当作模块使用

```
cat physics_data/script.py
```

```
def spherical_harmonic_fitter(grid, order):
    '''
    求球谐函数拟合的系数

    输入
    ~~~
    grid: 球面上连续函数在固定格点上的取值
    order: 拟合时球谐函数近似截断的阶数

    输出
    ~~~
    拟合系数矩阵
    '''

    # 具体实现省略
    pass
```

自定义模块（二）

```
from physics_data import script
help(script.spherical_harmonic_fitter)
```

```
Help on function spherical_harmonic_fitter in module physics_data.script:

spherical_harmonic_fitter(grid, order)
    求球谐函数拟合的系数

    输入
    ~~~
    grid: 球面上连续函数在固定格点上的取值
    order: 拟合时球谐函数近似截断的阶数

    输出
    ~~~
    拟合系数矩阵
```

fractions 有理数才是良定义的

https://docs.python.org/3/library/fractions.html

```
from fractions import Fraction
Fraction(16, -10)
```

```
Fraction(-8, 5)
```

decimal 计算机对人类的妥协

https://docs.python.org/3/library/decimal.html

```
from decimal import *
print(Decimal(1) / Decimal(7))
print(1/7)
```

```
0.14285714285714285714285714286
0.14285714285714285
```

迭代器

- 迭代器是一个对象，允许逐个访问容器（如列表、元组、集合）中的元素，而不必知道容器的内部实现细节。
  - 实现“迭代器协议”，在类中定义 `__iter__()` 函数。

与容器（复合数据类型）区别

- 容器（如列表、元组、集合）是数据的 集合 ，而迭代器是遍历这些集合中元素的 方式 。
- 迭代器提供了一种统一的访问接口，无论容器类型如何，都可以通过迭代器进行逐个访问。
  - 例子：字典可以定义多种迭代器。
- 列表和元组的迭代器恰好是按顺序访问每个元素，因此容易与列表和元组本身混淆。

应用

- `for` 循环由迭代器实现。列表、元组等能从迭代器创建。

迭代器

- 迭代器是一个对象，允许逐个访问容器（如列表、元组、集合）中的元素，而不必知道容器的内部实现细节。
  - 实现“迭代器协议”，在类中定义 `__iter__()` 函数。

与容器（复合数据类型）区别

- 容器（如列表、元组、集合）是数据的 集合 ，而迭代器是遍历这些集合中元素的 方式 。
- 迭代器提供了一种统一的访问接口，无论容器类型如何，都可以通过迭代器进行逐个访问。
  - 例子：字典可以定义多种迭代器。
- 列表和元组的迭代器恰好是按顺序访问每个元素，因此容易与列表和元组本身混淆。

应用

- `for` 循环由迭代器实现。列表、元组等能从迭代器创建。

迭代器

- 迭代器是一个对象，允许逐个访问容器（如列表、元组、集合）中的元素，而不必知道容器的内部实现细节。
  - 实现“迭代器协议”，在类中定义 `__iter__()` 函数。

与容器（复合数据类型）区别

- 容器（如列表、元组、集合）是数据的 集合 ，而迭代器是遍历这些集合中元素的 方式 。
- 迭代器提供了一种统一的访问接口，无论容器类型如何，都可以通过迭代器进行逐个访问。
  - 例子：字典可以定义多种迭代器。
- 列表和元组的迭代器恰好是按顺序访问每个元素，因此容易与列表和元组本身混淆。

应用

- `for` 循环由迭代器实现。列表、元组等能从迭代器创建。

itertools 高级迭代器变换

https://docs.python.org/3/library/itertools.html

- 丰富多样的迭代器操作，巧妙运用则功能强大。

```
import itertools as it

data = [3, 4, 6, 2, 1, 9, 0, 7, 5, 8]
list(it.accumulate(data, max))
```

```
[3, 4, 6, 6, 6, 9, 9, 9, 9, 9]
```

```
list(it.permutations(range(1, 4)))
```

```
[(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)]
```

itertools 取笛卡尔积

```
list(it.product('ABCD', repeat=2))
```

```
[('A', 'A'),
 ('A', 'B'),
 ('A', 'C'),
 ('A', 'D'),
 ('B', 'A'),
 ('B', 'B'),
 ('B', 'C'),
 ('B', 'D'),
 ('C', 'A'),
 ('C', 'B'),
 ('C', 'C'),
 ('C', 'D'),
 ('D', 'A'),
 ('D', 'B'),
 ('D', 'C'),
 ('D', 'D')]
```

itertools 取可重组合

```
tuple(it.combinations_with_replacement('ABCD', 2))
```

```
(('A', 'A'),
 ('A', 'B'),
 ('A', 'C'),
 ('A', 'D'),
 ('B', 'B'),
 ('B', 'C'),
 ('B', 'D'),
 ('C', 'C'),
 ('C', 'D'),
 ('D', 'D'))
```

itertools 使用 zip 合并两个迭代器

```
for i, s in zip(range(11), it.accumulate(range(11))):
    print(f"从 0 加到 {i} 的和是 {s}")
```

```
从0加到 0 的和是 0
从0加到 1 的和是 1
从0加到 2 的和是 3
从0加到 3 的和是 6
从0加到 4 的和是 10
从0加到 5 的和是 15
从0加到 6 的和是 21
从0加到 7 的和是 28
从0加到 8 的和是 36
从0加到 9 的和是 45
从0加到 10 的和是 55
```

itertools 过滤器

```
list(it.filterfalse(lambda n: n % 13, range(100)))
```

```
[0, 13, 26, 39, 52, 65, 78, 91]
```

大批量输入输出

- `input()` `print()` 适合少量的信息传递
- 大批量的读写宜直接操作文件

文本文件读

- 文件是迭代器，逐行。每行是字符串。

```
import itertools as it
with open("departments.csv") as f_input:
    for l in it.islice(f_input, 5):
        print(l, end="")
```

```
数学系
工物系
致理书院
致理书院
探微书院
```

- with 用来帮助在文件用完后及时关闭，防止占用和争夺资源。

```
from collections import Counter
with open("departments.csv") as f_input:
    print(Counter(f_input))
```

```
Counter({'致理书院\n': 30, '工物系\n': 20, '未央书院\n': 20, '物理系\n': 7, '上海交大\n': 4, '
数学系\n': 1, '探微书院\n': 1})
```

文本文件写

```
with open("log.txt", "w") as f:
    f.write("第一天 概论\n")
    f.write("第二天 Python 基础\n")
```

```
cat log.txt
```

```
第一天 概论
第二天 Python 基础
```

字符处理

- Python 内建了丰富的字符处理函数

```
s = "今天的气温是 30 摄氏度, 明天是 29 摄氏度"
print(s.count("度"))
print(s.startswith("今天"))
print(s.split(", "))
```

```
2
True
['今天的气温是 30 摄氏度', '明天是 29 摄氏度']
```

字符处理（二）

- 娱乐奥利奥生成器

```
seed = bin(2324)
print(seed)
print(seed[2:].replace('0',"奥").replace('1',"利"))
```

```
0b100100010100
利奥奥利奥奥奥利奥利奥奥
```

参考资料

- help(str)
- https://docs.python.org/3.9/library/stdtypes.html#textseq

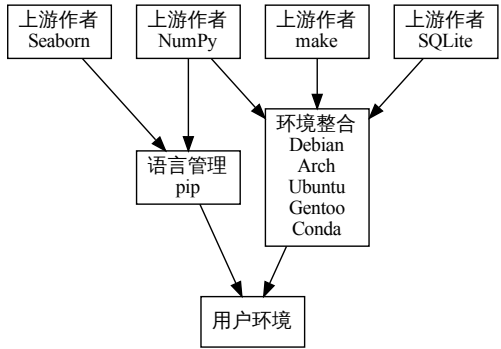
更多 Python 扩展库

- 优先使用 APT (Debian) 软件管理器

```
apt install python3-xxxxxx # Debian
```

- 混合 APT 与 pip 要极其小心，尽量不引入 pip
  - 提 issue 求助
  - 不要使用 conda

软件分发与环境整合：概念



- `pip` 没有整合测试，只适合早期试验个别最新 Python 软件，长期维护性差。
- 整体生产方案对应多种软件的稳定版本，经过应跨语言整合测试。

环境整合分发方案

- 第三方测试整合提供环境，防止被某上游垄断并偷藏私货
  - Malicious PyPI package opens backdoors on Windows, Linux, and Macs
    `Allows a remote attacker to gain unauthorized access to the application.`
  - `pip` 的软件没有经过第三方测试验证，使用意味者相信上游软件作者。
  - 环境整合方案会有负责安全、兼容和性能的团队，有大量同样环境的用户反馈问题。
- Debian/Arch/Gentoo 志愿者形态与 Ubuntu/Conda 公司免费形态
  - 志愿者形态的开发团队没有利益关系，人们由于共同的兴趣走到一起。
    - 无人能强制整个项目的走向，集体决策。
  - 公司免费形态利用免费服务吸引用户，构建潜在的客户池或者暗藏广告潜移默化改造用户。
    - 产品服务于公司的盈利或扩大影响的战略，代码由雇工产生。

不推荐用作科学计算和数据分析

- Ubuntu 的源代码 95% 从 Debian （合法）复制，外加公司定制和广告
- Conda 的依赖关系的计算效率极低，但在公司层面调动外宣经费公关推广

只有普通用户权限

- 安装和升级软件需要系统管理员权限，具体体现为在 `apt` 前加 `sudo` 。
- 课题组共享的服务器、超算中心的登录节点无权限怎么办？用户态软件管理

	成熟的管理工具	去中心研发	海量软件+物理学	超算中心采用	续本达召集
Gentoo Prefix	✓	✓	✓	✓	✓ ✓
nix/guix	✓	✓	✓		
Spack	✓			✓	
conda / mamba					

参考

Benda Xu, G. Amadio, F.Grffen, and M. Haubenwallner. “Gentoo Prefix as a Physics Software Manager.” EPJ Web of Conferences 245 (2020): 05036.

推荐：软件管理与环境配置

- 自己的机器用 Debian：工具安装用 `apt`
- 别人的机器用 Gentoo Prefix：工具安装用 `emerge`

NumPy: Numeric Python

- NumPy 起源于使用 Python 语言调用 fortran 进行线性代数运算的需求。
- 已经发展成为 Python 科学计算的基石
- 参考书：Scipy Lecture Notes

安装 NumPy 和相关工具

```
apt install python3-numpy python3-scipy python3-h5py
```

SciPy: Scientific Python

- NumPy 定义高效的数据结构
- SciPy 在 NumPy 的基础上提供的数值计算算法

scipy.cluster	Vector quantization / Kmeans
scipy.constants	Physical and mathematical constants
scipy.fftpack	Fourier transform
scipy.integrate	Integration routines
scipy.interpolate	Interpolation
scipy.io	Data input and output
scipy.linalg	Linear algebra routines
scipy.ndimage	n-dimensional image package
scipy.odr	Orthogonal distance regression
scipy.optimize	Optimization
scipy.signal	Signal processing
scipy.sparse	Sparse matrices
scipy.spatial	Spatial data structures and algorithms
scipy.special	Any special mathematical functions
scipy.stats	Statistics

创建和索引数组

```
import numpy as np

nv = np.array([1,2,3,4,3,2,1])
print(nv, nv[2], nv[5:])
print(nv[-1], nv[::2])
```

```
[1 2 3 4 3 2 1] 3 [2 1]
1 [1 3 3 1]
```

数组语法与列表相似，可能相互转换，区别在于：

- 数组要求元素的数据类型被预设且一致，列表 (List) 无此要求
- 数组的存储是一段连续的内存空间，列表不是
- 以上两点使得在数值计算中，数组的效率比列表高很多

二维数组用来表示矩阵

```
ma = np.array([[1,0], [0,1]])
print(ma)
```

```
[[1 0]
 [0 1]]
```

```
type(ma), ma.shape
```

```
(numpy.ndarray, (2, 2))
```

常数数组的创建

```
print(np.ones((3, 3)))
print(np.zeros((4, 4)))
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

随机矩阵的创建

```
print(np.random.rand(2,5))
```

```
[[0.80632296 0.53845746 0.00714721 0.84843748 0.32654466]
 [0.56930226 0.35817586 0.45108818 0.14484227 0.65463473]]
```

创建单位矩阵

- np.eye 读与大写字母 I 相同的读法

```
e = np.eye(4)
print(e)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

对角元

- 取矩阵对角元

```
print(np.diag(e))
```

```
[1. 1. 1. 1.]
```

- 由对角元生成对角矩阵

```
print(np.diag(np.arange(5)))
```

```
[[0 0 0 0 0]
 [0 1 0 0 0]
 [0 0 2 0 0]
 [0 0 0 3 0]
 [0 0 0 0 4]]
```

叠放

```
np.tile(e,2)
```

```
array([[1., 0., 0., 0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0., 0., 0., 1.]])
```

数组的形状

- 创建一个 10×10 的数组

```
m = np.arange(100, dtype=int)
print(m)
m.shape = (10, 10)
print(m)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
```

二维索引

```
m6 = m[:6, :6]
m6[:, 2]
```

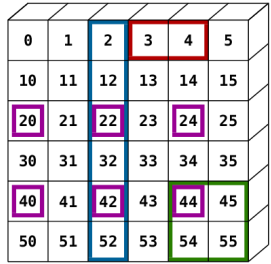
```
array([ 2, 12, 22, 32, 42, 52])
```

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 45],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



二维任意索引

- 分别传两个相等形状的索引数组，按索引数组形状排列对应的输出

```
print(m6)
print(m6[[1, 2, 3], [3, 4, 5]])
print(m6[[[1], [2]], [[3], [4]]])
```

```
[[ 0  1  2  3  4  5]
 [10 11 12 13 14 15]
 [20 21 22 23 24 25]
 [30 31 32 33 34 35]
 [40 41 42 43 44 45]
 [50 51 52 53 54 55]]
[13 24 35]
[[13]
 [24]]
```

整体运算

- 一般的运算符都可以在数组上使用
- 可以省去循环，使用程序比 map 和列表生成更简明易懂

```
n = np.arange(10)
print(n**2)
print([v**2 for v in n]) # 对比
```

```
[ 0  1  4  9 16 25 36 49 64 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

索引与运算结合

```
nr = n[::-1]
print(nr + n)
print(n[::2] + 100)
```

```
[9 9 9 9 9 9 9 9 9 9]
[100 102 104 106 108]
```

数组的总体特征

- 求和、平均、中位数、方差、标准差

```
np.sum(n), np.mean(n), np.median(n), np.var(n), np.std(n)
```

```
(45, 4.5, 4.5, 8.25, 2.8722813232690143)
```

- 选择性地对二维数组的某个维度求值，省去两重循环。

```
np.sum(m, axis=0), np.median(m, axis=1)
```

```
(array([450, 460, 470, 480, 490, 500, 510, 520, 530, 540]),
 array([ 4.5, 14.5, 24.5, 34.5, 44.5, 54.5, 64.5, 74.5, 84.5, 94.5]))
```

数组的扩展

- 把一维数组扩展成二维，补齐形状

```
n[None, :] + n[:, None]
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
       [ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11],
       [ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12],
       [ 4,  5,  6,  7,  8,  9, 10, 11, 12, 13],
       [ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14],
       [ 6,  7,  8,  9, 10, 11, 12, 13, 14, 15],
       [ 7,  8,  9, 10, 11, 12, 13, 14, 15, 16],
       [ 8,  9, 10, 11, 12, 13, 14, 15, 16, 17],
       [ 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]])
```

- 代替二重循环

```
np.array([x + y for x in n for y in n]).reshape(10, 10)
```