第 6 次课 字典和字符串

Python 科学计算

周吕文

宁波大学，机械工程与力学学院

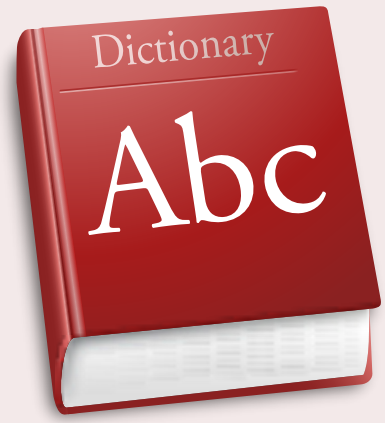
2024 年 9 月 1 日

提要

字典

```
import os
figs = {'fig1.pdf': 81761, 'fig2.png': 8754}
figs['NBU.pdf'] = os.path.getsize('figures/NBU.pdf')
for name in figs:
    print(' 文件 %8s 的大小是 %5d' % \
                (name, figs[name])  )
```

```
>>>
文件 fig1.pdf 的大小是 81761
文件 fig2.png 的大小是  8754
文件  NBU.pdf 的大小是  7967
```



字典可以看作一种广义上的列表

列表特性

- 列表中存储着一系列的元素
- 每个元素都是一个 Python 对象
- 通过序号（从 0 开始）访问每一个元素

字典

- 可以通过文本（词）索引对象
- Python 的字典在其它语言中也叫 Hash 表、Hash 映射、关联数组

使用字典建立映射关系

使用列表有时候会不自然

```
>>> temps = [34.3, 26.7, 32.5] # 宁波、北京、上海的气温
>>> print('宁波的温度: %.1f' % temps[0])
宁波的温度: 34.3
```

使用字典可以是城市和温度对应起来

```
>>> temps = {'宁波': 34.3, '北京': 26.7, '上海': 32.5}
>>> print('宁波的温度: %.1f' % temps['宁波'])
宁波的温度: 34.3
```

- 在字典中充当索引的称键（key），使用键访问的叫值（value）。
- 在上面的例子中，'宁波' 是键，34.3 是值。
- 字典是无序键-值对集合，通过键访问值：value = dictionary[key]

字典的初始化

两种初始化方法

```
>>> temps = {'nb': 34.3, 'bj': 26.7, 'sh': 32.5}
>>> temps
{'nb': 34.3, 'bj': 26.7, 'sh': 32.5}
>>> temps = dict(nb = 34.3, bj = 26.7, sh = 32.5)
>>> temps
{'nb': 34.3, 'bj': 26.7, 'sh': 32.5}
```

往字典中增加一个元素

```
>>> temps['hk'] = 30.5
>>> temps
{'nb': 34.3, 'bj': 26.7, 'sh': 32.5, 'hk': 30.5}
>>> temps['hk']
30.5
```

循环操作字典：对键的循环

```
>>> for city in temps:
...     print('%s 的温度是 %g C' % (city, temps[city]))
...
nb 的温度是 34.3 C
bj 的温度是 26.7 C
sh 的温度是 32.5 C
hk 的温度是 30.5 C
```

上面输出结果是无序的，怎么实现有序输出？

```
>>> for city in sorted(temps):
...     print('%s 的温度是 %g C' % (city, temps[city]))
...
bj 的温度是 26.7 C
hk 的温度是 30.5 C
nb 的温度是 34.3 C
sh 的温度是 32.5 C
```

查找键、删除元素

测试某个键是否在词典中

```
>>> if 'nb' in temps:
...     print('nb:', temps['nb'])
... else:
...     print('没有 nb 的温度')
...
nb: 34.3
>>> 'bj' in temps
True
```

删除某个元素

```
>>> temps
{'nb': 34.3, 'bj': 26.7, 'sh': 32.5, 'hk': 30.5}
>>> del temps['sh']
>>> temps
{'nb': 34.3, 'bj': 26.7, 'hk': 30.5}
```

可以通过迭代器或列表访问键和值

迭代器

```
>>> temps.keys()
dict_keys(['nb', 'bj', 'hk'])
>>> temps.values()
dict_values([34.3, 26.7, 30.5])
>>> for city in temps.keys():
...     print(city, temps[city])
...
nb 34.3
bj 26.7
hk 30.5
```

迭代器可以转成列表

```
>>> list(temps.keys())
['nb', 'bj', 'hk']
```

```
>>> list(temps.values())
[34.3, 26.7, 30.5]
```

字典的复制

```
>>> t1 = temps
>>> t1['nb'] = 35
>>> temps
{'nb': 35, 'bj': 26.7, 'hk': 30.5}
>>> t2 = temps.copy()
>>> t2['nb'] = 10
>>> temps
{'nb': 35, 'bj': 26.7, 'hk': 30.5}
```

复习：列表也有类似的性质和操作

```
>>> L = [1, 2, 3]
>>> M = L
>>> M[1] = 0
>>> L
[1, 0, 3]
```

```
>>> L = [1, 2, 3]
>>> M = L[:]   # = L.copy()
>>> M[2] = 0
>>> L
[1, 2, 3]
```

任意常量对象都可能作为字典的键

```
>>> a = {1: 65, 2: 67, 3: 89}      # 整数
>>> a
{1: 65, 2: 67, 3: 89}
>>> a = {1: 65, 2.3: 67, 3: 89}    # 整数, 浮点数
>>> a
{1: 65, 2.3: 67, 3: 89}
>>> a={(0,1): 45, (2,3): 67}       # 元组
>>> a
{(0, 1): 45, (2, 3): 67}

>>> a={[0,1]: 45, [2,3]: 67}       # 列表（不是常量对象）
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

例子：使用字典表示多项式

使用指数当键，系数当值

$$p(x) = -1 + x^2 + 3x^7 \quad \rightarrow \quad \texttt{p = \{0: -1, 2: 1, 7: 3\}}$$

定义计算多项式的函数 $p(x) = \sum c_i x^i$

eval_poly_dict.py

```
def poly_dict(poly, x):
    S = 0
    for power in poly:
        S += poly[power]*x**power
    # S = sum(poly[power]*x**power for power in poly)
    return S

p = {0: -1, 2: 1, 7: 3}
x = 2
print('p(%g) = %g' % (x, poly_dict(p,x)) )
```

```
>>>
p(2) = 387
```

也可以使用列表表示多项式

用列表序号表示指数

$$p(x) = -1 + x^2 + 3x^7 \quad \rightarrow \quad \texttt{p = [-1, 0, 1, 0, 0, 0, 0, 3]}$$

定义计算多项式的函数 $p(x) = \sum c_i x^i$ eval_poly_list.py

```
def poly_list(poly, x):
    S = 0
    for power in range(len(poly)):
        S += poly[power]*x**power
    # S = sum(poly[i]*x**i for i in range(len(poly)))
    return S

p = [-1, 0, 1, 0, 0, 0, 0, 3]
x = 2
print('p(%g) = %g' % (x, poly_list(p,x)) )
```

```
>>>
p(2) = 387
```

哪一种表示方法好?

字典更短: 字典只需要存储非 0 项, 如对于多项式 $1 - x^{100}$

- 字典: `p = {0: 1, 100: -1}` `# 长度 len(p) = 2`
- 列表: `p = [1, 0, 0, 0, ..., -1]` `# 长度 len(p) = 101`

字典可以表示指数为负的情况, 如 $x^{-3} + 2x^4$

```
p = {-3:1, 4:2}
x = 4
print('p(%g) = %g' % (x, poly_dict(p,x)) )
```

```
>>>
p(4) = 512.016
```

字典可表示指数为非整的情况, 如 $x^{-1/2} + 2x^{\pi}$

```
from math import pi
p = {-1/2: 1, pi: 2}
x = 4
print('p(%g) = %g' % (x, poly_dict(p,x)) )
```

```
>>>
p(4) = 156.26
```

课堂练习

1 计算多项式导数　　文件名：poly_diff.py

多项式的求导公式：

$$\frac{\mathrm{d}}{\mathrm{d}x}\sum_{j=0}^{n} c_j x^j = \sum_{j=1}^{n} j \cdot c_j x^{j-1}$$

在 python 中，多项式可以用一个字典来表示。请编写函数 diff 来计算这样一个多项式的导数，例如：

```
>>> p = {0: -3, 3: 2, 5: -1}   # -3 + 2*x**3 - x**5
>>> diff(p)                    # 6*x**2 - 5*x**4
{2: 6, 4: -5}
```

提示：`dp[j-1] = j*p[j]`

例子：读取文件数据到字典中

read_temps_dict.py

```
infile = open('cities_temp.txt', 'r')
temps = {}
for line in infile.readlines():
    city, temp = line.split(':')
    temps[city] = float(temp)
infile.close()
```

cities_temp.txt

```
Oslo:      21.8
London:    18.1
Berlin:    19
Paris:     23
Rome:      26
Helsinki:  17.8
```

temps

{'Oslo': 21.8, 'London': 18.1, 'Berlin': 19.0, 'Paris': 23.0, 'Rome': 26.0, 'Helsinki': 17.8}

例子：股价数据读入和可视化

stockprices_*.csv, * = {Apple, Google, Microsoft}

```
Date,Open,High,Low,Close,Volume,Adj Close
2014-02-03,37.74,37.99,35.69,37.42,42168900,37.42
2014-01-02,37.35,37.89,34.63,37.84,48732700,37.56
...
1986-03-13,25.50,29.75,25.50,27.50,155827200,0.07
```

写代码之前先设计算法

- **读数据**：1. 跳过第一行；2. 一行行读取文件数据，使用逗号对每一行的文本进行切分，将第一个词存入一个日期列表中，将倒数第一个词（调整收盘价）存入价格列表中；3. 以公司为关键码将日期和价格列表存入字典；4. 编写函数读取一个公司的数据。
- **绘制图**：将字符串形式的日期转换为年坐标（x 轴），注意不同公司股票开始于不同年。

例子：股价数据读入和可视化

stockprices.py（接下页） 读数据函数

```
from datetime import datetime
def read_file(filename):
    infile = open(filename, 'r')
    infile.readline() # 跳过第一行
    dates = []; prices = []
    for line in infile:
        words = line.split(',')
        dates.append(words[0])
        prices.append(float(words[-1]))
    infile.close()
    dates.reverse(); prices.reverse()
    dates = [datetime.strptime(_date, '%Y-%m-%d').date()
             for _date in dates]
    prices = np.array(prices)
    return dates, prices
```

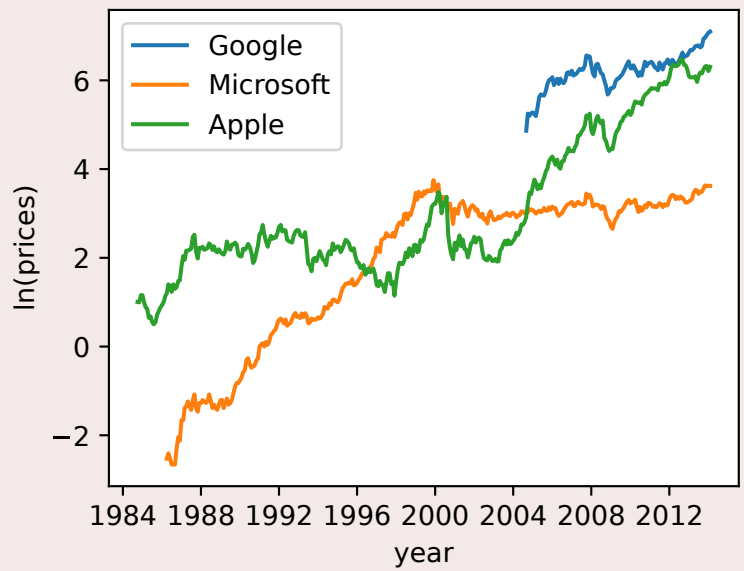
例子：股价数据读入和可视化

stockprices.py（接上页） 读入股价并可视化

```
import glob, numpy as np, matplotlib.pyplot as plt
dates = {};  prices = {}
filenames = glob.glob('stockprices_*.csv')
for filename in filenames:
    company = filename[12:-4]
    dates[company], prices[company] = read_file(filename)

for company in prices:
    x = dates[company]
    y = np.log(prices[company])
    plt.plot(x, y, label=company)

plt.xlabel('year')
plt.ylabel('ln(prices)')
plt.legend(); plt.show()
```



课堂练习

2 嵌套字典　　文件名：table2dict.py

表格文件 table.dat 的内容如下：

```
      A       B       C       D
1   11.7   0.035    2017    99.1
2    9.2   0.037    2019   101.2
3   12.2      no      no   105.2
4   10.1   0.031      no   102.1
5    9.1   0.033    2009   103.3
6    8.7   0.036    2015   101.9
```

建立字典结构（字典的字典），用 data[p][i] 存储第 i（1, 2, ...）行第 p（A, B, ...）列的数据。提示：

- 第一行：将第一行分成一个个词，将这些词作为键初始化字典，值为空。
- 剩余行：将每一行分成一个个词，第一个词作为键，后面的词，若不是 no，则作为值存入字典。

字典的一些方法和函数

`a = P{}`	创建一个空字典
`a = {'k': [2,7], 's': 7}`	创建一个字典
`a = dict(k=[2,7], s=3)`	创建一个字典
`a.update(d)`	利用 b 中的键值来添加/更新 d
`a.update(k=v)`	添加/更新 a 中的键-值对
`a['hide'] = True`	将新的键-值对添加到 a
`a['k']`	获取键 `'k'` 对应的值
`for key in a:`	以未指定顺序对键进行循环
`for key in sorted(a):`	按字母顺序对键进行循环
`'k' in a`	`'k'` 是 a 中的键则返回 `True`
`del a['k']`	从 a 中删除键为 `'k'` 的键-值对
`list(a.keys())`	键列表
`list(a.values())`	值列表
`len(a)`	a 中键-值对的个数
`isinstance(a, dict)`	如果 a 是字典，则返回 `True`
`get(a)`	返回键 a 的值，若未找到该键返回 `None`
`get(a, b)`	返回键 a 的值，若未找到该键返回 b

提要

字符串

字符串处理

```
>>> s = 'This is a string'
>>> s.split()
['This', 'is', 'a', 'string']
>>> 'This' in s
True
>>> s.find('is')
2
>>> ', '.join(s.split())
'This, is, a, string'
```

字符串处理是文件内容操作的关键

- 文本在 Python 中的表示为字符串
- 文件内容的处理需要字符串处理的支持

子串操作

获取子串：类似于列表和数组

```
>>> s = 'NB: 18.4 C at 4 pm'
>>> s[1]
'B'
>>> s[-1]
'm'
```

```
>>> s[8:]
' C at 4 pm'
>>> s[8:12]
' C a'
>>> s[8:-1]
' C at 4 p'
```

查找子串：返回子串序号

```
>>> s.find('at')
11
```

```
>>> s.find('BJ')
-1
```

判断是否包含某个子串

```
>>> 'NB' in s
True
```

```
>>> 'BJ' in s
False
```

字符串替换

s.replace(s1, s2): 将 s 中的 s1 替换成 s2

```
>>> s = 'NB: 18.4 C at 4 pm'
>>> s.replace(' ','-')
'NB:-18.4-C-at-4-pm'
>>> s.replace('NB','BJ')
'BJ: 18.4 C at 4 pm'
```

将第一个冒号前的字符串替换成 'SH'

```
>>> s
'NB: 18.4 C at 4 pm'
>>> s.replace(s[:s.find(':')], 'SH')    # s.find(':') =  2
'SH: 18.4 C at 4 pm'
>>> s.replace(s[:s.find(';')], 'SH')    # s.find(';') = -1
'SHm'
```

字符串拆分

s.split(sep): 将 s 用 sep 划分成子串列表，sep 缺省时使用空格

```
>>> s = 'NB: 18.4 C at 4 pm'
>>> s.split(':')
['NB', ' 18.4 C at 4 pm']
>>> s.split()
['NB:', '18.4', 'C', 'at', '4', 'pm']
>>> s.split(':')[1].split()[0] # 尝试理解一下这行程序
'18.4'
```

按行拆分: 多行文本行间由 \n（Linux）或 \r\n（Windows）隔开

```
>>> t = '1st line\n2nd line\n3rd line'
>>> t.split('\n')
['1st line', '2nd line', '3rd line']
>>> t.splitlines() # 跨平台
['1st line', '2nd line', '3rd line']
```

```
>>> print(t)
1st line
2nd line
3rd line
```

字符串是常量对象

字符串不能被改变

```
>>> s = 'NB: 18.4 C at 4 pm'
>>> s[4:8]
'18.4'
>>> s[4:8] = '20.5'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

所有对字符串的“修改”都是生成一个新串

```
>>> s = s[:4] + '20.5' + s[8:]
>>> s
'NB: 20.5 C at 4 pm'
```

删除字符串首尾空格

删除首尾空格

```
>>> s = '    NB: 18.4 C at 4 pm    '
>>> s.strip()
'NB: 18.4 C at 4 pm'
```

删除首（左）空格

```
>>> s.lstrip()
'NB: 18.4 C at 4 pm    '
```

删除尾（右）空格

```
>>> s.rstrip()
'    NB: 18.4 C at 4 pm'
```

其它常用字符串函数

判断对象是数字

```
>>> '315'.isdigit()
True
>>> ' 315 '.isdigit()
False
>>> '3.15'.isdigit()
False
```

转换大小写

```
>>> s = 'NB: 18.4 C at 4 pm'
>>> s.upper()
'NB: 18.4 C AT 4 PM'
>>> s.lower()
'nb: 18.4 c at 4 pm'
```

判断起始字符串

```
>>> s.startswith('NB')
True
>>> s.endswith('am')
False
```

字符串连接

用分隔符连接字符串

```
>>> slist = ['NBU', 'PKU', 'THU']
>>> ','.join(slist)
'NBU,PKU,THU'
>>> ','.join(slist).split(',') # 两个互逆的操作
['NBU', 'PKU', 'THU']
```

例子：去掉一行中的前两个单词

```
>>> line = 'This is a line of words separated by space'
>>> words = line.split()
>>> line2 = ' '.join(words[2:])
>>> line2
'a line of words separated by space'
```

例子：从文件读取数字对（x，y）

pairs.dat

```
(1.3,0)     (-1,2)      (3,-1.5)     (5,-7)   (3.4,2)
(0,1)       (1,0)       (1,1)        (8,2.9)  (6.2,7)
(0,-0.01)   (10.5,-1)   (2.5,-2.5)   (9,3.2)  (3,7.2)
```

算法

- 一行行读取文件内容
- 对每一行，分出一个个词
- 对每个词，去掉前后括号，然后用逗号分出两个数值

例子：从文件读取数字对（x, y）

read_pairs.py

```
with open('pairs.dat', 'r') as infile:
    lines = infile.readlines()

pairs = []
for line in lines:
    words = line.split()
    for word in words:
        word = word[1:-1]  # 去掉括号
        n1, n2 = word.split(',')
        n1, n2 = float(n1), float(n2)
        pair = (n1, n2)
        pairs.append(pair)

import pprint
pprint.pprint(pairs)
```

IDLE Shell

```
[(1.3, 0.0),
 (-1.0, 2.0),
 (3.0, -1.5),
 (5.0, -7.0),
 (3.4, 2.0),
 (0.0, 1.0),
 (1.0, 0.0),
 (1.0, 1.0),
 (8.0, 2.9),
 (6.2, 7.0),
 (0.0, -0.01),
 (10.5, -1.0),
 (2.5, -2.5),
 (9.0, 3.2),
 (3.0, 7.2)]
```

例子：从文件读取数字对（x，y）- 使用格式转换实现

```
>>> s = '(1.3,0)  (-1,2)  (3.4,2)\n(0,1)  (3,-1.5)  (5,-7)'
>>> s = '[' + s.replace(')', '),') + ']'
>>> s
'[(1.3,0),  (-1,2),  (3.4,2),\n(0,1),  (3,-1.5),  (5,-7),]'
>>> eval(s)
[(1.3, 0), (-1, 2), (3.4, 2), (0, 1), (3, -1.5), (5, -7)]
```

read_pairs2.py

```
with open('pairs.dat', 'r') as infile:
    text = infile.read()

text = '[' + text.replace(')', '),') + ']'
pairs = eval(text)
```

课堂练习

3 代码转换　　文件名：python2to3.py

Python3 和 Python2，并不完全兼容，一些语法上有差异。例如，从 Python2 变为 Python3，以下两种情况的代码就需要做相应的更改：

- 将字典 d 键转列表：`d.keys()` → `list(d.keys())`
- 使用 `print` 输出 x：`print x` → `print(x)`

请编写程序读入以下 Python2 版本的代码 python2.py，并通过字符串处理，将其转化为 Python3 版本的代码，并用 exec 函数执行并输出运行结果。

```
postcode = {'NBU': 315211, 'THU': 100084, 'ZJU': 310053}
U = postcode.keys() # -> U = list(postcode.key())
print postcode      # -> print(postcode)
print 'U =', U     # -> print('U =', U)
```

例子：从网上获取数据并读入

https://www.metoffice.gov.uk/.../oxforddata.txt

```
Oxford
Location: 450900E 207200N, Lat 51.761 Lon -1.262, 63 metres amsl
Estimated data is marked with a * after the value.
Missing data (more than 2 days missing in month) is marked by  ---.
Sunshine data taken from an automatic Kipp & Zonen sensor marked ...
   yyyy  mm    tmax    tmin      af    rain     sun
                degC    degC    days      mm   hours
   1853   1     8.4     2.7       4    62.8     ---
   1853   2     3.2    -1.8      19    29.3     ---
...
   1853  12     3.7    -1.3      19    10.7     ---
...
   2023   1     8.5     2.6       8    57.2    91.6
   2023   2    10.4     3.0       9     7.6    91.2
...
   2023  12    10.2     5.2       5   117.2    21.9*
   2024   1     8.0     2.6       6    76.3    79.7   Provisional
   2024   2    11.6*    5.4       2   111.9    57.3*  Provisional
```

例子：从网上获取数据并读入

算法

- 从文件头读取城市和位置信息，然后跳过接下来的 5 行
- 读取每一行信息存入字典，注意处理 '*' 和 'Provisional'

historic_weather.py（接下页）

```
import urllib.request
url = 'https://www.metoffice.gov.uk/pub/data/weather/uk\
/climate/stationdata/oxforddata.txt'
local_file = 'Oxford.txt'
urllib.request.urlretrieve(url, filename=local_file)
infile = open(local_file, 'r')
data = {};
data['place'] = infile.readline().strip()
data['location'] = infile.readline().strip()
for i in range(5): infile.readline()
```

historic_weather.py（接上页）

```
data['data'] ={}
for line in infile:
    columns = line.split()
    year, month = int(columns[0]), int(columns[1])
    if columns[-1] == 'Provisional': del columns[-1]
    for i in range(2, len(columns)):
        if columns[i] == '---':
            columns[i] = None
        elif columns[i][-1] == '*' or columns[i][-1] == '#':
            columns[i] = float(columns[i][:-1])
        else:
            columns[i] = float(columns[i])
    tmax, tmin, air_frost, rain, sun = columns[2:]
    if not year in data['data']: data['data'][year] = {}
    data['data'][year][month] = {'tmax': tmax,
          'tmin': tmin, 'air frost': air_frost, 'sun': sun}
infile.close()
```

The End!