高级程序设计
---Python与深度学习
4. 元组和文件
4. Tuple and file

李冰
副研究员
交叉科学研究院

回顾

- Python数据类型的常用操作及用法
  - 字符串
  - 列表
  - 字典
  - 动态类型
    - 对象引用

课程内容

- 元组
  - 元组操作
  - 元组方法
- 文件
  - 文件操作
  - 文件方法
- None对象

元组

- 元组通常写成圆括号的形式。
  - 任意对象的有序集合
    - 元组是一个位置有序的对象的集合（也就是其内容维持从左到右的顺序）（同字符串和列表）。
    - 可以嵌入到任何类别的对象中（同列表）。
  - 通过偏移存取
    - 在元组中的元素通过偏移来访问，它们支持所有基于偏移的操作。例如，索引和分片。（同字符串和列表）
  - 属于不可变序列类型
    - 元组是不可变的，它不支持任何原处修改操作（同字符串）。
  - 固定长度、异构、任意嵌套
    - 因为元组是不可变的，在不生成一个拷贝的情况下不能增长或缩短。
    - 元组可以嵌套包含其它复合对象（例如列表、字典和其它元组等）。
  - 对象引用的数组
    - 元组存储指向其它对象的引用，因此对元组进行索引操作的速度相对较快。

元组操作

```
x = (40)
x
```

```
40
```

```
y = (40, )
y
```

```
(40,)
```

圆括号里的唯一对象是一个表达式，则不是元组；
如果确认需要构造一个元组，需要在这个对象之后加一个逗号

元组操作

```
In [1]: type((1, 2))
Out[1]: tuple

In [2]: (1, 2) + (3, 4)          # Concatenation
Out[2]: (1, 2, 3, 4)

In [3]: (1, 2) * 4               # Repetition
Out[3]: (1, 2, 1, 2, 1, 2, 1, 2)

In [4]: T = (1, 2, 3, 4)         # Indexing, slicing
        T[0], T[1:3]
Out[4]: (1, (2, 3))
```

元组操作 – 排序

- 排序的两种实现方式
  - 转换为列表，用列表**sort()**方法

```
T = ('cc', 'aa', 'dd', 'bb')
tmp = list(T)                   # Make a list from a tuple's items
tmp.sort()                      # Sort the list
tmp
```

```
['aa', 'bb', 'cc', 'dd']
```

```
T = tuple(tmp)                  # Make a tuple from the list's items
T
```

```
('aa', 'bb', 'cc', 'dd')
```

元组操作 – 排序

- 排序的两种实现方式
  - 转换为列表，用列表**sort()**方法

```
T = ('cc', 'aa', 'dd', 'bb')
tmp = list(T)                   # Make a list from a tuple's items
tmp.sort()                      # Sort the list
tmp
```

```
['aa', 'bb', 'cc', 'dd']
```

```
T = tuple(tmp)                  # Make a tuple from the list's items
T
```

```
('aa', 'bb', 'cc', 'dd')
```

  - 元组内置**sorted()**方法

```
T = ('cc', 'aa', 'dd', 'bb')
sorted(T)                       # Or use the sorted built-in, and save two steps
```

```
['aa', 'bb', 'cc', 'dd']
```

元组操作

- 获取元素下标

```
T = (1, 2, 3, 2, 4, 2)        # Tuple methods in 2.6, 3.0, and later
T.index(4)                    # Offset of first appearance of 4
```

```
4
```

```
T.index(2)
```

```
1
```

```
T.index(2, 2)                 # Offset of appearance after offset 2
```

```
3
```

```
tu = ('hello', 333,  (44, 55,), [(888, 999,)], 54, 333, True)
v = tu.index(333, 4, 7)       # 待查询下标的元素，查询起始索引，查询终止索引
v
```

```
5
```

- 统计 ‘2’ 出现的次数。

```
T.count(2)                    # How many 2s are there?
```

```
3
```

元组操作-小结

方法	例子及含义
sorted(T)	原位排序
T.index()	返回某一索引区间某个已知元素的索引
T.count()	某一元素出现的次数

元组性质

- 不可变对象

```
T = (1, [2, 3], 4)
# T[1] = 'spam'          # This fails: can't change tuple itself
```

元组性质

- 不可变对象

```
T = (1, [2, 3], 4)
# T[1] = 'spam'          # This fails: can't change tuple itself
```

- 元组的不可变性只适用于元组本身顶层
  - 元组内部的可变对象是可以修改的

```
T[1][0] = 'spam'         # This works: can change mutables inside
T
```

```
(1, ['spam', 3], 4)
```

元组的不可变性提供了某种完整性。这样可以确保元组在程序中不会被另一个引用修改，而列表就没有这样的保证了。

几种对象对比

	列表	字典	元组
类型(type)	list	dict	tuple
创建	[], list()	{} ;dict(key, value); dict(zip(list1, list2))	(,); tuple()
增加	.append(), +, *	d.update()	+,*
删除	l.remove(); l.pop(); l.clear()	d.clear(), d.pop(key)	无
修改	l.append(); l.insert(); l.extend()	d[key]=value	不可更改
索引，切片	支持	不支持, d.get(key)	支持
闭包表达式	l=[i操作式 for l in list1] 返回列表	d={i:j for i,j in zip(list1,list2)}	t=(i操作式 for i in list1)

课程内容

- 元组
  - 元组操作
  - 元组方法
- 文件
  - 文件基本操作
  - 文件常用操作
- None对象

文件

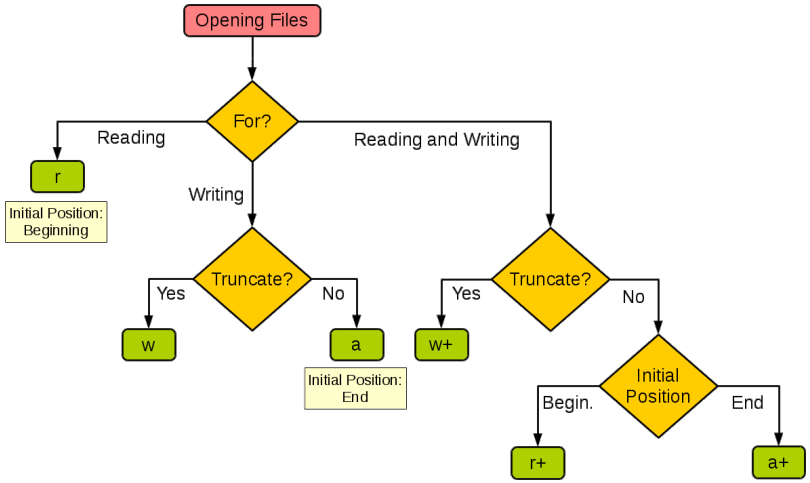
- 文件对象与前几种对象类型不同
  - 不是数字、序列,
  - 对**表达式操作符**（*、+）没有相关响应
  - 只包含通用文件处理的方法。
- 文件对象的多数方法都与执行外部文件的**输入输出操作**有关
  - 查找文件中新位置(seek())
  - 刷新输出缓存(flush())

文件基本操作

- 打开文件open()

```
afile = open(filename, mode)
afile.method()
```

  - mode: **处理模式使用字符串**
    - 'r' 代表读取文件（默认值）
    - 'w' 代表创建并写文件
    - 'a' 代表在打开文件并在文件尾部追加内容
    - 加上 '+' 代表同时进行读写操作；在模式字符串末尾加上 'b' 代表进行二进制数据处理



文件方法

- 打开文件，写入一些内容

```
myfile = open('myfile.txt', 'w')          # Open for text output: create/empty
myfile.write('hello text file\n')          # Write a line of text: string, return the number of characters written
```

```
16
```

返回字符数量

文件方法

- 打开文件，写入一些内容

```
myfile = open('myfile.txt', 'w')          # Open for text output: create/empty
myfile.write('hello text file\n')          # Write a line of text: string, return the number of characters written
```

```
16
```

返回字符数量

```
myfile.write('goodbye text file\n')
```

```
18
```

文件方法

- 打开文件，写入一些内容

```
myfile = open('myfile.txt', 'w')          # Open for text output: create/empty
myfile.write('hello text file\n')          # Write a line of text: string, return the number of characters written
```

```
16
```

返回字符数量

```
myfile.write('goodbye text file\n')
```

```
18
```

```
myfile.close()                             # Flush output buffers to disk
```

open ()创建了一个Python对象；
默认是文本打开，二进制打开要用**b**；
用 close（）方法关闭文件是一个很好的习惯。

文件读写

- 打开文件，读文件的内容
  - **readline()**，一行行读

```
myfile = open('myfile.txt', 'r')          # Open for text input: 'r' is default
myfile.readline()                         # Read the lines back
```

```
'hello text file\n'
```

```
myfile.readline()
```

```
'goodbye text file\n'
```

```
myfile.readline()
```

```
''
```

```
myfile.close()
```

  - **read()** 方法一次将整个文件读入到一个字符串中

```
print(open('myfile.txt').read())
```

```
hello text file
goodbye text file
```

文件读写

- 使用文件迭代器来一行一行读取文件。
  - open() 创建的临时文件对象将自动在每次循环迭代的时候读入并返回一行。

```
for line in open('myfile.txt'): # Use file iterators, not reads
    print(line, end='')
```

```
hello text file
goodbye text file
```

文件读写

- 在文件中存储并解析Python对象

```
S = 'Spam'                                  # Must be strings to store in file
X, Y, Z = 43, 44, 45                        # Native Python objects
L = [1, 2, 3]
D = {'a': 1, 'b': 2}

F = open('datafile.txt', 'w')               # Create output text file
F.write(S + '\n')                           # Terminate lines with \n
F.write('%s,%s,%s\n' % (X, Y, Z))           # Convert numbers to strings
F.write(str(L) + '$' + str(D) + '\n')       # Convert and separate with $
F.close()
```

- 创建文件后，就可以通过打开和读取字符串来查看文件的内容。

```
chars = open('datafile.txt').read()         # Raw string display
print(chars)
```

```
Spam
43,44,45
[1, 2, 3]${'a': 1, 'b': 2}
```

文件读写

- 我们用readline()一行行处理。

```
F = open('datafile.txt')          # Open again
line = F.readline()               # Read one line
line
```

```
'Spam\n'
```

去除多余的换行符。字符串处理的rstrip()方法

```
line.rstrip()                     # Remove end-of-line, 包括空格、换行(\n)、制表符(\t)等
```

```
'Spam'
```

文件读写

- 我们用readline()一行行处理。

```
F = open('datafile.txt')          # Open again
line = F.readline()               # Read one line
line
```

```
'Spam\n'
```

去除多余的换行符。字符串处理的rstrip()方法

```
line.rstrip()                     # Remove end-of-line, 包括空格、换行(\n)、制表符(\t)等
```

```
'Spam'
```

- 第二行，拿到所有数字，组成列表

```
line = F.readline()               # Next line from file
line                              # It's a string here
```

```
'43,44,45\n'
```

文件读写

- 我们用readline()一行行处理。

```
F = open('datafile.txt')          # Open again
line = F.readline()               # Read one line
line
```

```
'Spam\n'
```

去除多余的换行符。字符串处理的rstrip()方法

```
line.rstrip()                     # Remove end-of-line, 包括空格、换行(\n)、制表符(\t)等
```

```
'Spam'
```

- 第二行，拿到所有数字，组成列表

```
line = F.readline()
line
```

```
'43,44,45\n'
```

```
parts = line.split(',')           # Split (parse) on commas
parts
```

```
['43', '44', '45\n']
```

split() 方法，从逗号分隔符的地方断开
int()把数字字符转换为整数

```
numbers = [int(P) for P in parts]   # Convert all in list at once
numbers
```

```
[43, 44, 45]
```

文件读写– Python对象

- 第三行存储的列表和字典

eval() 这一内置函数，eval() 能将字符串视作可执行程序代码

```
line = F.readline()
line
```

```
"[1, 2, 3]${'a': 1, 'b': 2}\n"
```

```
parts = line.split('$')                    # Split (parse) on $
parts
```

```
['[1, 2, 3]', "{'a': 1, 'b': 2}\n"]
```

```
eval(parts[0])                             # Convert to any object type
```

```
[1, 2, 3]
```

```
objects = [eval(P) for P in parts]         # Do same for all in list
objects
```

```
[[1, 2, 3], {'a': 1, 'b': 2}]
```

文件读写 – Python对象

- 内置pickle库，实现Python 对象的读写处理，无需字符串来回转换

```
import pickle

D = {'a': 1, 'b': 2}
file = open('datafile.pkl', 'wb')

pickle.dump(D, file)         # Pickle any object to file
file.close()
```

pickle.dump()，将对象以二进制形式写入文件中，进行保存。

pickle.load() 将对象从文件中读取出来。

```
file = open('datafile.pkl', 'rb')
E = pickle.load(file)          # Load any object from file
file.close()
E
```

```
{'a': 1, 'b': 2}
```

文件上下文管理器(File Context Manager)

- 读写操作完成后，总是要close()
- 用 **with** 语句实现，以确保在退出时可以自动关闭文件。

```
with open('myfile.txt') as myfile:
    for line in myfile:
        print(line)
```

```
hello text file

goodbye text file
```

文件上下文管理器(File Context Manager)

- 读写操作完成后，总是要close()
- 用 **with** 语句实现，以确保在退出时可以自动关闭文件。

```
with open('myfile.txt') as myfile:
    for line in myfile:
        print(line)
```

```
hello text file

goodbye text file
```

```
myfile = open('myfile.txt')
try:
    for line in myfile:
        print(line)
finally:
    # 无论是否发生异常都将执行
    myfile.close()
```

```
hello text file

goodbye text file
```

Python 中提供的 try/finally 异常处理语句也可以实现类似的功能

文件操作-小结

方法	例子及含义
open(filename, mode)	打开文件对象，以读，写，读写，追加等模式
F.read()	raw字符串读取文件全部内容
F.readline()	读取文件一行
F.write()	将字符串写入文件
F.close()	关闭文件对象，将buffer内容写入磁盘
pickle.dump(D, F)	以二进制形式将对象D写入文件F
pickle.load(F)	从F中读取对象D，返回对象

课程内容

- 元组
  - 元组操作
  - 元组方法
- 文件
  - 文件操作
  - 文件方法
- None对象

None对象

- 起到一个空的占位作用
  - None 是一个特殊的常量
  - None 和 False 不同
  - None 不是 0
  - None 不是空字符串
  - None 和任何其他的数据类型比较永远返回 False
  - None 有自己的数据类型 NoneType
  - 可以将 None 赋值给任何变量，但是不能创建其它 NoneType 对象

例如，对于列表来说，无法为一个超出范围的偏移进行赋值操作。要预先分配一个10项的列表，你可以在10个位置上都预先设置为None：

```
L = [None] * 10
L
```

```
[None, None, None, None, None, None, None, None, None, None]
```

练习

- 写个表达式将元组（4，5，6）改为（1，5，6）。
- 读取一个二进制图像文件，并尝试进行图像处理（更改其像素数值）再写回文件

文件读写 – 二进制文件

- Python 的标准库包含一个处理模块：struct 模块，它包含了 pack() 和 unpack() 函数，能打包和解析打包的二进制数据。
  - 从某种意义上来说，它是一个数据转换工具，能把文件中的字符串解读为二进制数据。

例如，要生成一个二进制数据文件，用 'wb'（写入二进制）模式创建文件，并将格式化字符串对象传给 struct。这里用的格式化字符串包含一个4字节的整数、一个包含4个字符的字符串以及两个2字节的整数

```
import struct

F = open('data.bin', 'wb')                          # Open binary output file
data = struct.pack('<i4s2h', 18, b'spam', 8, 2)  # Make packed binary data
data
```

```
b'\x12\x00\x00\x00spam\x08\x00\x02\x00'
```

```
F.write(data)                                       # Write byte string
F.close()
```

文件读写 – 二进制文件

	18		‘spam’	8	2
>	i	4	s	2	h

‘>’ :大端字节顺序，第一个字符表示可用于指示打包数据的字节顺序

‘i’ 表示整数格式
‘4s’ 表示一个 10 字节的字节串
而 ’2h‘ 表示 2个整数。

```
import struct

F = open('data.bin', 'wb')                          # Open binary output file
data = struct.pack('>i4s2h', 18, b'spam', 8, 2)  # Make packed binary data
data
```

```
b'\x00\x00\x00\x12spam\x00\x08\x00\x02'
```

```
F.write(data)                                       # Write byte string
F.close()
```

文件读写 – 二进制文件

- 对齐方式

字符	字节顺序	大小	对齐方式
@	按原字节	按原字节	按原字节
=	按原字节	标准	无
<	小端	标准	无
>	大端	标准	无
!	网络（=大端）	标准	无

- 字节顺序可能为大端或是小端，取决于具体的主机系统。
  - 例如，Intel x86 和 AMD64（x86-64）是小端序；IBM z 和多数传统架构是大端序；
  - 而 ARM, RISC-V 和 IBM Power 具有可切换的字节顺序（双端，不过前两个系统实际上几乎总是小端序）。
  - 使用 sys.byteorder 来检查你的系统字节顺序。

文件读写 – 二进制文件

- 要将值解析为普通 Python 对象，可以简单地读取字符串，并使用相同格式的字符串把它解压出来就可以。

```
F = open('data.bin', 'rb')
data = F.read()                                    # Get packed binary data
F.close()
data
```

```
b'\x00\x00\x00\x12spam\x00\x08\x00\x02'
```

```
values = struct.unpack('>i4s2h', data)             # Convert to Python objects
values
```

```
(18, b'spam', 8, 2)
```

```
values = struct.unpack('<i4s2h', data)             # Convert to Python objects
values
```

```
(301989888, b'spam', 2048, 512)
```