collections— Container datatypes

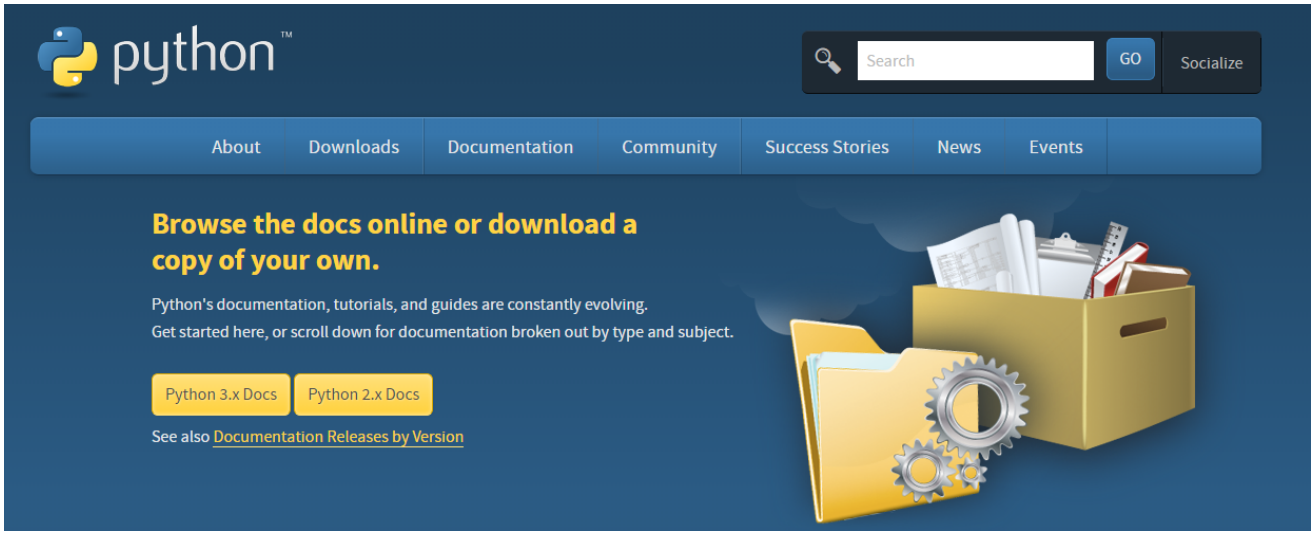
撰写：张启煊

优雅的语言中更优雅的模块.

0.放在前面的一些建议

1）Python瞬息万变，看官方文档尤其重要

Python瞬息万变，看官方文档尤其重要，着重了解版本间的差异、实现方法的改变等，有兴趣还可以看看源码



2）一些可能会帮到你的函数

- type()：对象类型
- id()：对象所在内存
- dir()：对象下所有的内容（模块下所有的函数、函数下所有信息）
- help()：对象的docs内容（查询某函数的使用方法等）
- print(func_name.__doc__)
- ……

3）Jupyter Notebook

记事本、分块代码运行、科学分析、md兼容

Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

- 这不是个IDE！！！

4）Python之禅

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
-----------------译文来自网络-----------------------------------------------------------------
Python之禅 by Tim Peters

优美胜于丑陋（Python 以编写优美的代码为目标）
明了胜于晦涩（优美的代码应当是明了的，命名规范，风格相似）
简洁胜于复杂（优美的代码应当是简洁的，不要有复杂的内部实现）
复杂胜于凌乱（如果复杂不可避免，那代码间也不能有难懂的关系，要保持接口简洁）
扁平胜于嵌套（优美的代码应当是扁平的，不能有太多的嵌套）
间隔胜于紧凑（优美的代码有适当的间隔，不要奢望一行代码解决问题）
可读性很重要（优美的代码是可读的）
```

即便假借特例的实用性之名，也不可违背这些规则（这些规则至高无上）

不要包容所有错误，除非你确定需要这样做（精准地捕获异常，不写 except:pass 风格的代码）

当存在多种可能，不要尝试去猜测
而是尽量找一种，最好是唯一一种明显的解决方案（如果不确定，就用穷举法）
虽然这并不容易，因为你不是 Python 之父（这里的 Dutch 是指 Guido ）

做也许好过不做，但不假思索就动手还不如不做（动手之前要细思量）

如果你无法向人描述你的方案，那肯定不是一个好方案；反之亦然（方案测评标准）

命名空间是一种绝妙的理念，我们应当多加利用（倡导与号召）

5）有了Anaconda你什么都有了

anaconda （一个开源的Python发行版本），不只是懒人包.

- 多个项目多个Python版本?
- 环境污染?
- 库需要CMaker编译?
- pip速度太慢?

- 部分库运行效率更高（据说）
- Anaconda：Conda+Python+工具包...

1. 这TM是个啥

- **module_name：** collections & collections.abc
  - collections.abc为Python3.3后从collection中被取缔内容，暂时保留
- **usage：** This module implements specialized container datatypes providing alternatives to Python's general purpose built-in containers, dict, list, set, and tuple.
- 一些很好的数据结构，一套优雅的解决方案

2.collections有什么用途

场景：issues on Hw5

- 路人甲：为什么我本地测试器全过，OJ全都RE?
- 路人乙：考虑使用与OJ相同版本的Python3.5测试. 可能是Python3.7 "新特性" 带来的差异
  - Python 3.7 中dict实现方法发生了改变，保留了输入时的顺序，兼容了sort方法
  - dict不该有顺序，这是个副产品；dict在检索时是随机的
  - 后续版本会取消这一特性
- 路人甲：那怎么办？是不是要重写?
- 路人乙：OrderedDict了解一下!

collections中的数据类型

Name	Usage
`namedtuple()`	factory function for creating tuple subclasses with named fields
`deque`	list-like container with fast appends and pops on either end
`ChainMap`	dict-like class for creating a single view of multiple mappings
`Counter`	dict subclass for counting hashable objects
`OrderedDict`	dict subclass that remembers the order entries were added
`defaultdict`	dict subclass that calls a factory function to supply missing values
`UserDict`	wrapper around dictionary objects for easier dict subclassing
`UserList`	wrapper around list objects for easier list subclassing
`UserString`	wrapper around string objects for easier string subclassing

3.collections中有趣的玩意儿

这里我们将介绍：

- OrderedDict
- Counter
- namedtuple

1）OrderedDict：带顺序的字典?

- OrderedDict([items])
- OrderedDict继承自dict类，拥有dict类的基本特性
- 优势（与Python3.7中dict呈现出来的样子差别不大）：
  - dict由于按照hash储存，所以是无序的（一般情况下；当然，这也决定了dict无法储存unhashable的东西），OrderedDict实现字典有序化
  - OrderedDict会根据放入元素的先后顺序进行排序（当然也可以sort后输出）
  - 通过双向链表保存（至少之前版本是这样）
  - ……
- 应用场景：
  - 大概就是Hw5?
  - ……

2）Counter：优雅的计数器

- collections.Counter([iterable-or-mapping])
- 用于统计某个词出现的次数（类似list.count）
- 应用场景
  - 词频分析?
  - 面向OJ编程?
  - ……

演示

```
>>> c = Counter('abracadabra')
>>> c
Counter({'a': 5, 'r': 2, 'b': 2, 'd': 1, 'c': 1})
>>> c.update('zzzbbe')  # 追加元素
>>> c
Counter({'a': 5, 'b': 4, 'z': 3, 'r': 2, 'e': 1, 'c': 1, 'd': 1})
>>> c.most_common(3)  # 获取出现频率最高的前 3 个字符
[('a', 5), ('b', 4), ('z', 3)]
--------------------------------------------------------------------------------
Counter(str1)==Counter(str2)
```

3）namedtuple：被命名的元组?

- namedtuple(typename, field_names, *, rename=False, defaults=None, module=None)
- namedtuple继承自tuple类，拥有tuple类的基本特性，如狭义上不可变更
- 优势（类似于C语言中的struct）：
  - tuple元组的item只能通过index访问，namedtuple不仅可以使用item的index访问item，还可以通过item的name进行访问

    - 能够用索引来访问数据
    - 能够迭代
    - 能够方便的通过属性名来访问数据。
- 应用场景：
    - 某个点的坐标?
    - 用户数据集暂存?
    - ……

演示

```
>>> # Basic example
>>> Point = namedtuple('Point', ['x', 'y'])
>>> p = Point(11, y=22)     # instantiate with positional or keyword arguments
>>> p[0] + p[1]             # indexable like the plain tuple (11, 22)
33
>>> x, y = p                # unpack like a regular tuple
>>> x, y
(11, 22)
>>> p.x + p.y               # fields also accessible by name
33
>>> p                       # readable __repr__ with a name=value style
Point(x=11, y=22)
```

More

- deque 双向队列（依然与C++类似）
    - 可以用于多线程的线程池或者消息队列
    - 对于deque来说从队列两端添加或弹出元素的复杂度都是O(1)。而从列表的头部插入或移除元素时，列表的复杂度为O(N)
- ChainMap映射链（Chain+Map）
    - 串联多组映射到一起
    - 变量名+作用域例子（逐层映射）
- defaultdict默认字典
    - 给字典的空键匹配默认值
    - 信息缺省处理
- UserDict、UserList、UserString（用于自定义）
    - 供实现自己的Dict、List、String继承来用