第9课　Python标准库

标准库

- 学习每个Python程序员必须知道的一些库
- Python标准库包括各种各样的有用工具和界面，涵盖了非常广泛的领域。

  - ✓ math
  - ✓ re
  - ✓ time/datetime
  - ✓ pickle
  - ✓ collections
  - ✓ itertools
  - ✓ sys
  - ✓ os

标准库

- 学习每个Python程序员必须知道的一些库
- Python标准库包括各种各样的有用工具和界面，涵盖了非常广泛的领域。
- 一些术语：

  •**模块**（module）：可复用的最小的Python代码单元。
    •包含Python定义和语句的文件。

  •**包**（package）：模块的集合。
    •例如，numpy聚合了数百个不同的模块以提供广泛的功能。

  •**标准库**（standard library）：
    •包和模块的集合，默认情况下与Python一起分发。

模块

```
#导入模块
import math
math.sqrt(16)

#将函数从模块导入
from math import ceil, floor
ceil(3.7)
floor(3.7)

#别名
from math import degrees as deg
deg(45.7)
```

- 每个Python文件都是一个模块；
- 可以将较大的项目组织成许多文件的集合；
- 然后使用import语句引用其他文件中的函数。

从Package导入

"sound" package 文件结构

```
sound/
├── __init__.py
├── formats/
│   ├── __init__.py
│   ├── wavread.py
│   ├── wavwrite.py
│   ├── aiffwrite.py
│   ├── auwrite.py
│   └── ...
├── effects/
│   ├── __init__.py
│   ├── echo.py
│   ├── reverse.py
│   └── ...
└── filters/
    ├── __init__.py
    ├── equalizer.py
    ├── karaoke.py
    └── ...
```

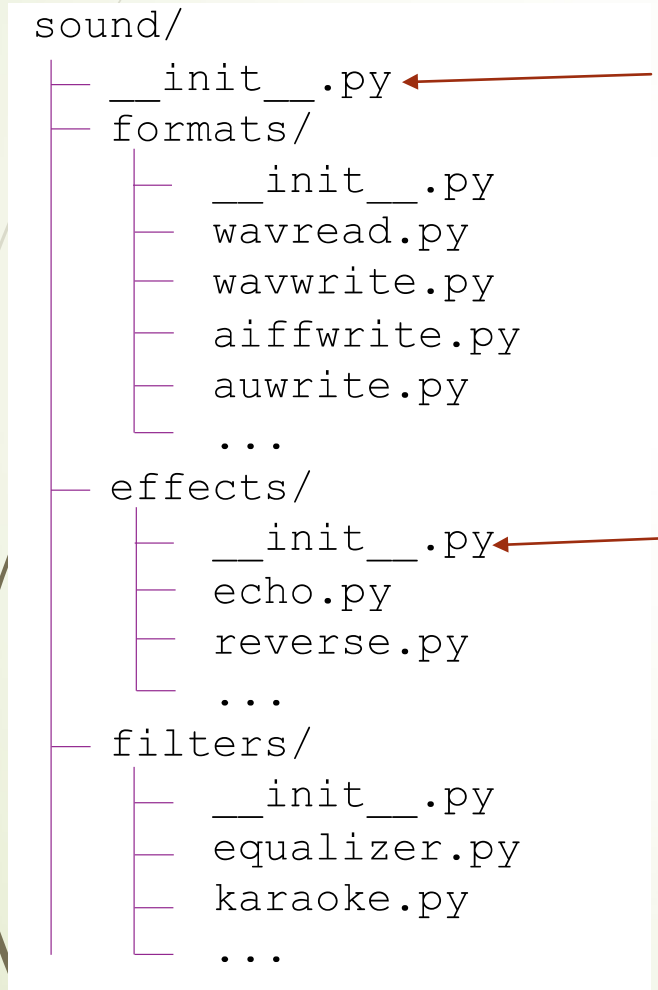
```
# 从最顶级导入
import sound
sound.effects.echo.echofilter(input, output)

# 从第二级导入
from sound.effects import echo
echo.echofliter(input, output)

# 从模块级别的导入
from sound.effects.echo import echofilter
echofilter(input, output)
```

https://docs.python.org/3/tutorial/modules.html

https://www.python-course.eu/sound1.tar.bz2

从Package导入

"sound" package 文件结构




```
__all__ = ["formats", "filters", "effects"]
```

```
>>> from sound import *
sound package is getting imported!
formats package is getting imported!
filters package is getting imported!
effects package is getting imported!
```

```
__all__ = ["echo", "surround", "reverse"]
```

```
>>> from sound.effects import *
sound package is getting imported!
effects package is getting imported!
Module echo.py has been loaded!
Module surround.py has been loaded!
Module reverse.py has been loaded!
```

一些导入的习惯

•导入一般写于Python文件的顶部，这样可以明确依赖关系，避免错误。

•首选import …，次选from … import …，因为这样可以避免可能的**名称冲突**。

•尽量避免from … import *, 这将导入所有内容，而使用时也不需要加模块名称前缀。

•有时候可能不知道要导入的所有内容，可能会导致名称冲突

标准库

- 不同的数据类型，如数字和列表
- 内置常数，如None，True，False
- 内置的很多函数
- 使用它们不需要导入任何内容，实际上是标准库定义了这些类型以及许多其他内置组件

		Built-in Functions		
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

time

- import time
- time模块负责提供与时间相关的功能和转换方法。

time.time() –自1970年1月1日00:00:00 UTC+00:00时区起的时间
time.sleep()–暂停执行秒。
time.clock()–返回当前处理器时间（秒）。

```
import time
def timer():
      s = time.time()
      time.sleep(5)
      e = time.time()
      print (e-s)

def cpu_timer():
      s = time.clock()
      time.sleep(5)
      e = time.clock()
      print (e-s)
```

```
timer()
#5.000328302383423
#time.time()只测量经过的绝对时间。


cpu_timer()
#9.400000000000038e-05
#time.clock()测量CPU运行该程序的时间。
```

当睡眠时，程序暂停一段时间，CPU在睡眠期间就不活动。

datetime

- datetime是Python处理日期和时间的标准库。

```
from datetime import datetime
now = datetime.now() # 获取当前日期时间
print(now)        #2020-05-03 21:09:57.164484
print(type(now)) #<class 'datetime.datetime'>
```

注意不同的datetime：datetime是模块，datetime模块还包含一个datetime类，通过from datetime import datetime导入的才是datetime这个类。

timestamp解释

在计算机中，时间实际上是用数字表示的。我们把1970年1月1日 00:00:00 UTC+00:00时区的时刻称为epoch time，记为0（1970年以前的时间timestamp为负数），当前时间就是相对于epoch time的秒数，称为timestamp。

datetime

datetime和timestamp的转换

```
timestamp = 0 即 1970-1-1 00:00:00 UTC+0:00

dt = datetime(2020, 5, 18, 19, 00)
                                # 用指定日期时间创建datetime
print(dt)                       #2020-05-18 19:00:00
print(dt.timestamp())           #1589799600.0
print(datetime.fromtimestamp(1589799650.0))
                                   #2020-05-18 19:00:50
```

datetime转换为str： 把datetime格式化为字符串显示给用户

```
now = datetime.now()
print(now.strftime('%a, %b %d %H:%M'))
Sun, May 03 21:28
```

（为什么？？？ 建议查找每个输出字母所代表的意思）

datetime

str转换为datetime: 用户输入的日期和时间是字符串，要处理日期和时间

```
cday = datetime.strptime('2020-5-18 20:00:00', '%Y-
%m-%d %H:%M:%S')
print(cday)
2020-05-18 20:00:00
```

时间的加减:timedelta

```
from datetime import datetime,timedelta
start = datetime(2020, 5, 18, 19, 00)
finish = start + timedelta(hours=1,minutes=20)

print(start)   #2020-05-18 19:00:00
print(finish)  #2020-05-18 20:20:00
```

collections

- **collections是Python内建的一个集合模块，提供了许多有用的集合类**

- 之前学过的双端队列deque

```
>>> from collections import deque
>>> q = deque(maxlen=5)                    #创建双端队列
>>> for item in [3, 5, 7, 9, 11]:
    q.append(item)
>>> q.append(13)
>>> q.append(15)
>>> q
deque([7, 9, 11, 13, 15], maxlen=5)
>>> q.appendleft(5)                  #从左侧添加元素，右侧自动溢出
>>> q
deque([5, 7, 9, 11, 13], maxlen=5)
>>> q.popleft()
5
```

collections

- 回顾元组：不可变的集合
- 例如:
- 一个点的二维坐标，p=(1,2)
- 迷宫游戏的东南西北方向通向的房间号码：direction=（“1”，“3”，“4”，“7”）

- 以上表示方法有什么问题？

- 元组的元素是由索引引用的，而不是由名称引用的（容易引起混乱）
- 在几千行代码中，可能很难看出 (1,2)就是个坐标
- 可能另外一个程序员不喜欢按照东南西北的顺序？喜欢东西南北？

namedtuple

- namedtuple用来创建一个自定义的tuple对象，并且规定了tuple元素的个数，并**可以用属性而不是索引**来引用tuple的某个元素。

```
from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])
p = Point(1, 2)
print(p.x) #1
print(p.y) #2

Maze_room = namedtuple("Maze_room", ["north", "east",
"south", "west"])
```

namedtuple可以很方便地定义一种数据类型，
它具备tuple的不变性，又可以根据属性来引用

namedtuple

```
from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])
p = Point(1, 2)
print(isinstance(p, Point)) #True
print(isinstance(p, tuple)) #True
```

类似于定义简单的类：不需要很多功能，只包含少量属性

```
namedtuple('名称', [属性list]):
```

```
Circle = namedtuple('Circle', ['origin','radius'])
mycircle = Circle (p,5)
print(mycircle.radius) #5
print(mycircle.origin) #Point(x=1, y=2)
```

collections

假设我们要处理数据：

```
# 输入数据:
input_data =
[('yellow', 1), ('blue', 2), ('yellow', 3), ('blue',
4), ('red', 1)]

# 我们想要得到的是:
processed_data =
       {'blue': [2, 4], 'red': [1], 'yellow': [1, 3]}

# 最好的方法是什么？
```

collections

```
#方法一
input_data = [('yellow', 1), ('blue', 2), ('yellow',
3), ('blue', 4), ('red', 1)]
d = {}
for k, v in input_data:
        if k not in d.keys():#需要初始化字典中没有的键的值
                d[k] = []
        d[k].append(v)


#方法二
input_data = [('yellow', 1), ('blue', 2), ('yellow',
3), ('blue', 4), ('red', 1)]
d = {}
for k, v in input_data:
        d[k] = d.get(k, []) + [v]
```

defaultdict

使用dict时，如果引用的key不存在，就会抛出KeyError。如果希望key不存在时，返回一个默认值，就可以用defaultdict：

```
# 好方法
input_data = [('yellow', 1), ('blue', 2), ('yellow', 3),
('blue', 4), ('red', 1)]

#将默认值设为空列表-这样我们就不需要为字典中没有的键设置值。
d = collections.defaultdict(list)
for k, v in input_data:
        d[k].append(v)

print(d)
print(d['yellow'])
print(d['white']) # key不存在，返回默认值

defaultdict(<class 'list'>, {'yellow': [1, 3], 'blue': [2,
4], 'red': [1]})
[1, 3]
[]
```

OrderedDict

使用dict时，Key是无序的。在对dict做迭代时，我们无法确定Key的顺序。 如果要保持Key的顺序，可以用OrderedDict：

```
from collections import OrderedDict
d = dict([('a', 1), ('b', 2), ('c', 3)])
print(d)  # dict的Key是无序的
{'a': 1, 'b': 2, 'c': 3}

od = OrderedDict([('a', 1), ('b', 2), ('c', 3)])
print(od) # OrderedDict的Key是有序的
OrderedDict([('a', 1), ('b', 2), ('c', 3)])
```

```
>>> od = OrderedDict()
>>> od['z'] = 1
>>> od['y'] = 2
>>> od['x'] = 3
>>> list(od.keys()) # 按照插入的Key的顺序返回
['z', 'y', 'x']
```

Counter

Counter是一个简单的计数器，例如，统计公仔（字符）出现的个数：

```
import collections

animals = "🐑🐑🐶🐶🐴🐴🐑🐙🐑🐶"
num_each_animal = collections.Counter(animals)

>>> print(num_each_animal)
Counter({'🐑': 4, '🐶': 3, '🐴': 2, '🐙': 1})

>>> num_each_animal.update("🐑🐑🐶🐑🐑🐶")

>>> print(num_each_animal)
Counter({'🐑': 8, '🐶': 5, '🐴': 2, '🐙': 1})
```

Counter实际上也是dict的一个子类，上面的结果可以看出每个字符出现的次数。

```
print(issubclass(collections.Counter,dict))#True
```

itertools

- **Python的内建模块itertools提供了非常有用的用于操作迭代对象的函数。**

itertools提供了创建无限迭代器的工具。

```
itertools.count(start, step) #count()会创建一个无限的迭代器
# start, start + step, start + 2*(step), start +
3*(step), ...

itertools.cycle("ABC") #cycle()会把传入的一个序列无限重复下去
# A, B, C, A, B, C, A, ...

itertools.repeat(32)   #repeat()负责把一个元素无限重复下去
# Loops over 32, 32, 32, 32, 32, 32, ...
```

itertools

```
import itertools
for i in itertools.count(10, 2):
        print(i)
        if i > 19:
                break
# 10 12 14 16 18 20

# 类似生成器，我们根据需求截取出一个有限的序列

natuals = itertools.count(1)
ns = itertools.takewhile(lambda x: x <= 10, natuals)
print(list(ns))
# [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# 通过takewhile()等函数根据条件判断来截取出一个有限的序列
```

itertools

练习

```
import itertools

counter = 0
for i in itertools.cycle([1, 2, 3]):
        print(i, end=' ')
        counter = counter + 1
        if counter > 12:
                break
```

```
1 2 3 1 2 3 1 2 3 1 2 3 1
```

itertools

- **itertools.product**
- 在数据集上实现一些漂亮的组合循环：迭代每个参数的所有元素组合。

```
itertools.product("ABCD", "EFGH")
# ('A', 'E'), ('A', 'F'), ('A', 'G'), ('A', 'H'), ('B',
'E') ...

itertools.product("ABCD", "EFGH", "IJKL")
# ('A', 'E', 'I'), ('A', 'E', 'J'), ('A', 'E', 'K') ...

itertools.product("ABCD", repeat=2)
# ('A', 'A'), ('A', 'B'), ('A', 'C'), ('A', 'D'), ('B',
'A') ...
# 即itertools.product("ABCD", "ABCD")
```

itertools

- **itertools.permutations itertools.combinations**
- 排列组合

```
itertools.permutations("ABCD", 2)
# 循环“ABCD”的所有长度为2的排列：
# ('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'A'),
('B', 'B') ...

itertools.combinations("ABCD", 2)
# 循环“ABCD”的所有长度为2的组合
# ('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'C'),
('B', 'D'), ('C', 'D')
```

- 实验8 第四题

itertools

- **itertools.chain**
- 把一组迭代对象串联起来，形成一个更大的迭代器
- 返回第一个迭代对象中的元素，直到它耗尽为止，然后继续返回下一个迭代对象，直到所有迭代对象都耗尽为止。

```
for c in itertools.chain(['a', 'b', 'c'], [1, 2, 3]):
        print(c)

# 迭代效果：'a' 'b' 'c' '1' '2' '3'
```

itertools

- **itertools.groupby**
- 把迭代器中相邻的重复元素挑出来放在一起

```
for key, group in itertools.groupby('AAABBBCCAAA'):
        print(key, list(group))

    A ['A', 'A', 'A']
    B ['B', 'B', 'B']
    C ['C', 'C']
    A ['A', 'A', 'A']

for key, group in itertools.groupby('AAabbBCCaa'):
        print(key, list(group))

    A ['A', 'A']
    a ['a']
    b ['b', 'b']
    B ['B']
    C ['C', 'C']
    a ['a', 'a']
```

sys

- sys模块提供对解释器使用或维护的一些变量的访问，以及与解释器交互的一些方法。它允许您接收有关运行时环境的信息并对其进行修改。

- 使用sys模块最常用的方法之一是访问传递给程序的参数。
- 通过sys.argv列表完成的。
- sys.argv列表的第一个元素始终是模块名，后跟空格分隔的参数。

模块 testargs.py

```
import sys

for i in range(len(sys.argv)):
        print("sys.argv[" + str(i) + "] is " + sys.argv[i])
```

sys

模块 testargs.py

```
import sys

for i in range(len(sys.argv)):
        print("sys.argv[" + str(i) + "] is " + sys.argv[i])
```

```
 $ python testargs.py here are some arguments
 sys.argv[0] is testargs.py
 sys.argv[1] is here
 sys.argv[2] is are
 sys.argv[3] is some
 sys.argv[4] is arguments
```

sys

- sys.path变量指定Python查找导入模块的位置。
- sys.path变量也是一个列表，可以由正在运行的程序自由操作。
- 第一个元素始终是模块所在的当前目录。

```
import sys

print("path has", len(sys.path), "members")
#path has 6 members
print("sys.path[0]:", sys.path[0])
#sys.ath[0]: /Users/Pan/Desktop

sys.path = []
import math
#报错!
Traceback (most recent call last):
  File "test.py", line 7, in <module>
    import math
ModuleNotFoundError: No module named 'math'
```

sys

- sys.stdin、sys.stdout和sys.stderr分别对应标准输入、标准输出和标准错误的文件对象。
- 就像sys模块中的其他属性一样，这些属性也可以随时更改！
- 如果要将标准文件对象还原为其原始值，使用sys.__stdin__, sys.__stdout__, sys.__stderr__ 。

```
import sys
f = open("somefile.txt", "w")
sys.stdout = f
print("This is going to be written to the file!")

sys.stdout = sys.__stdout__
print("This is going to be output to the Python
window!")
```

sys

```
import sys
f = open("somefile.txt", "w")
sys.stdout = f
print("This is going to be written to the file!")

sys.stdout = sys.__stdout__
print("This is going to be output to the Python
window!")
```

somefile.txt

```
This is going to be written to the file!
```

```
(base) Henry@PANHaoyuans-MacBook-Pro Desktop % python test.py
This is going to be output to the Python window!
```

sys

标准输入例子 （某公司校园招聘编程模拟试卷第一题）

```
import sys

for line in sys.stdin:
        a = line.split()
        print(int(a[0]) + int(a[1]))
```

```
(base) Henry@PANHaoyuans-MacBook-Pro Desktop % python test.py
1 2
3
3 4
7
^CTraceback (most recent call last):
  File "test.py", line 3, in <module>
    for line in sys.stdin:
KeyboardInterrupt
```

OS

os模块为依赖于操作系统的功能提供了通用接口。

我们之前遇到过的：

•os.chdir(*newcwd*)	改变当前工作目录到指定的路径
•os.rename(*current_name*, *new_name*)	重命名
•os.remove(*filename*)	删除文件
•os.getcwd()	当前工作目录
•os.mkdir(*newdirname*)	创建目录
•os.rmdir(*dirname*)	删除目录

OS

```
>>> import os
>>> os.getcwd()
'/Users/Pan/Desktop/python'
>>> os.makedirs("dir1/dir2/dir3")
>>> os.listdir()
['somefile.txt', 'test.py', 'dir1']

>>> f = open("dir1/dir2/dir3/test", "w")
>>> f.write("hi!")
>>> f.close()
>>> for line in open("dir1/dir2/dir3/test", "r"):
...     print(line)
...
hi!

>>> os.remove("dir1/dir2/dir3/test")
```

OS

os.walk（path）方法将遍历以path为根的目录树，为找到的每个目录生成一个元组（dirpath、dirnames、filenames）

```
>>> os.makedirs("dir1/dir2/dir3")
>>> os.listdir()
['somefile.txt', 'test.py', 'dir1']

>>> f = open("dir1/dir2/d2file", "w")

>>> path = os.getcwd()
>>> for (path, dirs, files) in os.walk(path):
...     print( "Path: ", path)
...     print( "Directories: ", dirs)
...     print ("Files: ", files)
...     print("---")
...
```

OS

os.walk（path）方法将遍历以path为根的目录树，为找到的每个目录生成一个元组（dirpath、dirnames、filenames）

```
Path: /Users/Henry/Desktop/python
Directories:  ['dir1']
Files:  ['somefile.txt', 'test.py']
---
Path:  /Users/Henry/Desktop/python/dir1
Directories:  ['dir2']
Files:  []
---
Path:  /Users/Henry/Desktop/python/dir1/dir2
Directories:  ['dir3']
Files:  ['d2file']
---
Path:  /Users/Henry/Desktop/python/dir1/dir2/dir3
Directories:  []
Files:  []
---
```

OS

os.stat（path）方法返回与提供的路径相关的文件属性

```
>>> stat_info = os.stat("somefile.txt")
>>> stat_info
os.stat_result(st_mode=33188, st_ino=8640756310,
st_dev=16777223, st_nlink=1, st_uid=501, st_gid=20,
st_size=41, st_atime=1588504503, st_mtime=1588496893,
st_ctime=1588504502)
```

- st_size: 文件大小（字节）
- st_atime: 最近访问的时间
- st_uid: 所有者的用户id
- …

OS

os.system(command) 函数在shell中执行命令行参数command，返回值是执行命令行的退出状态。

```
>>> os.system("ls")      #ls命令用于显示指定工作目录下之内容
dir1 somefile.txt test.py
0

>>> os.system("vim a.txt") # vim 文字编辑工具，退出后返回
0
0

>>> os.system("ls")
a.txt dir1 somefile.txt test.py
0
```

pickle

pickle: 对一个 Python 对象结构的二进制序列化和反序列化。

•所谓序列化，简单地说就是把内存中的数据在不丢失其类型信息的情况下转成对象的二进制形式的过程，对象序列化后的形式经过正确的反序列化过程应该能够准确无误地恢复为原来的对象。

•数据库文件、图像文件、可执行文件、音视频文件、Office文档等等均属于二进制文件。

•对于二进制文件，不能使用记事本或其他文本编辑软件进行正常读写，也无法通过Python的文件对象直接读取和理解二进制文件的内容。

pickle

pickle: 对一个 Python 对象结构的二进制序列化和反序列化。

- ✓ “Pickling” 是将 Python 对象及其所拥有的层次结构转化为一个字节流 (bytes structure)的过程，而 "unpickling" 是相反的操作
- ✓ 保留格式、保存数据的好工具

两大基本方法：

➡dump()方法是把对象保存到文件中。

格式：dump(obj, file, protocol=None)

```
data = np.array([.........]) # data可为任意Python对象
with open('data_file_name.pickle', 'wb') as f:
        pickle.dump(data, f)
```

➡load()方法是从文件中读数据，重构为原来的Python对象。

格式：load(file)

```
with open('data_file_name.pickle', 'rb') as f:
        data = pickle.load(f)
```

pickle

```
import pickle
dict1 = {'a':1, 'b':2, 'c':3}
output = open('data.pkl', 'wb')  # 二进制模式打开文件
pickle.dump(dict1, output)
                # 执行完导入操作，当前目录会生成data.pkl文件
output.close()  # 写入数据并关闭
```

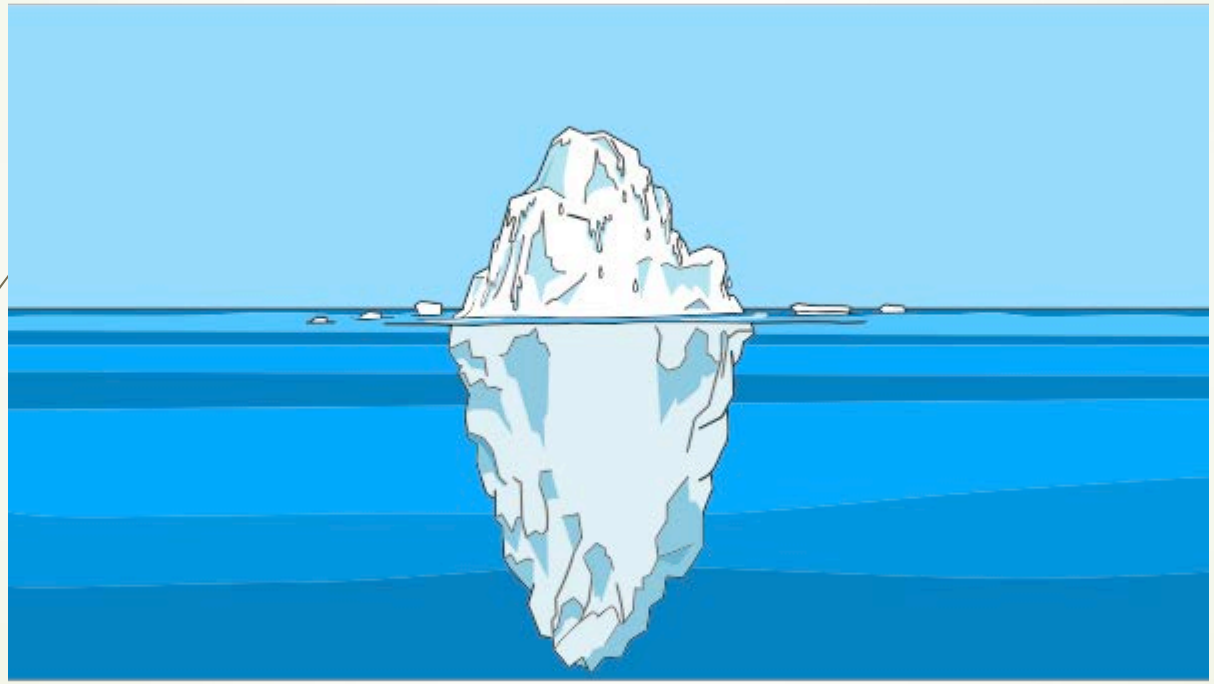
```
f = open('data.pkl','rb')
data = pickle.load(f)
print(data)
#{'a': 1, 'b': 2, 'c': 3}
```

pickle

```
import pickle
dict1 = {'a':1, 'b':2, 'c':3}
#dumps()返回一个pickle格式化的字符串
mystr=pickle.dumps(dict1)
print(mystr)
b'\x80\x03}q\x00(X\x01\x00\x00\x00aq\x01K\x01X\x01
\x00\x00\x00bq\x02K\x02X\x01\x00\x00\x00cq\x03K\x0
3u.'


#loads()解析pickle字符串
data = pickle.loads(mystr)
print(data)
{'a': 1, 'b': 2, 'c': 3}
```

标准库

今天讲的只是冰山一角…



更多请见

https://docs.python.org/zh-cn/3.7/library/index.html