Python数据分析

（4）

殷传涛 教授

chuantao.yin@buaa.edu.cn

代码复用

- 一个简单的程序可以在一个文件中用顺序、选择、循环语句来完成
- 但是，程序设计中几个很重要的理念：
- 代码的复用：代码能够被多个同样使用环境下，能够重复利用，目标在于减少代码量，同时，一个小的程序总是容易被测试的
- 大的程序总是分成若干小的程序，目标在于团队分组的开发
- 我们尽量希望一个程序模块就解决本身的功能，而不需要它做很多工作

代码复用

- 代码复用的几种形式：
  - 函数
  - 类
  - 模块
  - 包
  - 库

函数（function）定义

- 函数是一个能完成特定功能的代码块，可在程序中重复使用，减少程序的代码量和提高程序的执行效率。
- 在python中函数定义语法如下：
  - def function_name(arg1,arg2[,...]):
    - ✓ statement
    - ✓ [return value]
  - **def** 函数名（参数列表）:
    - ✓ 函数体
    - ✓ 返回值（可以缺省）

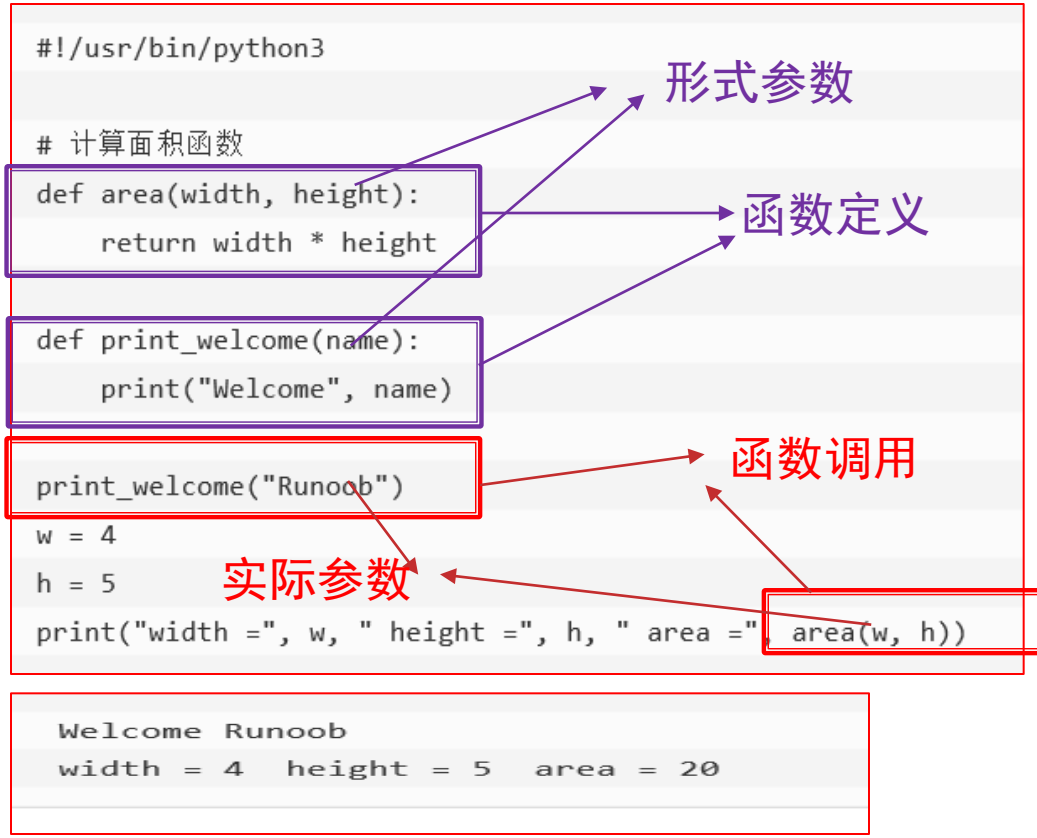
```
>>> def hello() :
    print("Hello World!")


>>> hello()
Hello World!
>>>
```

函数定义

- def是定义函数的关键字
- 函数的名称，根据函数功能进行命名
- 函数的输入：即函数的参数表，用圆括号括起来，逗号隔开
- 冒号，缩进的代码块（函数体）
- 函数的输出(可缺省)：return 返回值

函数参数与调用过程

- 函数的参数
  - 形式参数
  - 实际参数
- 程序执行过程
  1. 执行程序
  2. 通过函数名调用函数
  3. 将实际参数赋给形式参数
  4. 执行函数体
  5. 返回值
  6. 继续执行程序



- 调用时，实际参数将值赋给形式参数
- 实际参数可以是数据，变量，或表达式
- 实际参数要与形式参数的类型兼容

调用函数时对实际参数的要求

- 实际参数必须和形式参数一致
  - 数量一致
  - 类型一致

```
def printme(name):
    print(name)
    return name[0]

person="tom"
printme(person)
printme()
printme(5)
```

数量或类型不一致。最后两行程序均会报错

关键字参数

- 关键字参数允许函数调用时，实际参数与形式参数的顺序不一致。

```
def printme(name, age):
    print(age)
    print(name[0])

printme("Tom", 25)
printme(35, "Jenny")
```

最后一行程序均会报错

```
def printme(name, age):
    print(age)
    print(name[0])

printme("Tom", 25)
printme(age=35, name="Jenny")
```

OK，程序正确！

```
================ RE
25
T
35
J
>>>
```

参数的默认值

- 调用函数时，如果没有实际参数，则会使用形式参数的默认值
- 定义默认值

```
def printme(name="nobody",age=1):
    print(age)
    print(name[0])

printme()
printme("Tom")
printme(age=5)
```

* 默认的参数必须在最后，如printme(5)，则会出错
* 或者使用关键字参数，标明传入的是哪个参数

变量在参数传递时的变化

- Python中，数据类型属于对象，变量名仅仅只是名称，可以给任何对象进行命名
- 对象
  - 不可更改的对象：数字、字符串、元组
  - 可更改的对象：列表、集合、字典
- 作为函数的实际参数时
  - 不可更改的对象：将对象复制一个传到函数体内，不会被函数内部修改
  - 可更改的对象：将对象传递到函数体内，会被函数体内修改

函数的参数

- 实例

```
#!/usr/bin/python3

def ChangeInt( a ):
    a = 10

b = 2
ChangeInt(b)
print( b ) # 结果是 2
```

b的值不会被改变

```
#!/usr/bin/python3

# 可写函数说明
def changeme( mylist ):
   "修改传入的列表"
   mylist.append([1,2,3,4]);
   print ("函数内取值: ", mylist)
   return

# 调用changeme函数
mylist = [10,20,30];
changeme( mylist );
print ("函数外取值: ", mylist)
```

```
函数内取值:  [10, 20, 30, [1, 2, 3, 4]]
函数外取值:  [10, 20, 30, [1, 2, 3, 4]]
```

mylist的值发生了改变

变量的作用域

- 变量在创建时，作为名称赋给一个对象
- 变量在哪里能访问，取决于在哪里被赋值的
- 变量
  - 全局变量：在程序主体中被赋值，具有全局作用域，可在全局使用
  - 局部变量：在函数内部被赋值，具有局部作用域，仅在函数内使用

变量的作用域

- 实例

```
total = 0; # 这是一个全局变量
# 可写函数说明
def sum( arg1, arg2 ):
    #返回2个参数的和."
    total = arg1 + arg2; # total在这里是局部变量.
    print ("函数内是局部变量 : ", total)
    return total;

#调用sum函数
sum( 10, 20 );
print ("函数外是全局变量 : ", total)
```

```
函数内是局部变量 :  30
函数外是全局变量 :  0
```

变量的作用域

- 在函数内使用global关键字来注明全局变量

```
def fun1():
    global num
    num=2
    print("函数内",num)

fun1()
print("函数外",num)
```

```
----------------
函数内 2
函数外 2
>>>
```

函数的使用

- 函数的类型
  - 内建函数，直接使用：list( ), str( ), abs( )等

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()

  - 模块函数（math, time, random等），需导入模块后使用：
    - import math　math.pow(2,3)

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh'
, 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fm
od', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'is
inf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan'
, 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

  - 自定义函数，自定义和使用：def function_name( )

函数的使用

- 使用import导入模块
- 使用dir ()查看模块内容
- 使用help()查看函数使用方法

```
>>> help("modules")

Please wait a moment while I gather a list of all available modules...

__future__          autocomplete        idle_test           search
__main__            autocomplete_w      idlelib             searchbase
_ast                autoexpand          imaplib             searchengine
_asyncio            base64              imghdr              secrets
_bisect             bdb                 imp                 select
_blake2             binascii            importlib           selectors
_bootlocale         binhex              inspect             setuptools
_bz2                bisect              io                  shelve
_codecs             browser             iomenu              shlex
_codecs_cn          builtins            ipaddress           shutil
_codecs_hk          bz2                 itertools           signal
_codecs_iso2022     cProfile            json                site
_codecs_jp          calendar            keyword             smtpd
_codecs_kr          calltip_w           lib2to3             smtplib
```

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh'
, 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fm
od', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'is
inf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan'
, 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

```
>>> help("math.sin")
Help on built-in function sin in math:

math.sin = sin(...)
    sin(x)

    Return the sine of x (measured in radians).
```

面向对象

- 现实世界中，一切有形和无形的事物都可以用对象来描述
- 对象就是对客观世界的一种抽象
- 对象具有的特征
  - 静态特征：描述事物的属性，用属性来表示
  - 动态特征：描述事物的行为，用方法来表示
- 对象将事物的属性和方法封装在一起（称为封装）
- 面向对象的方法：从客观世界中把事物抽象成对象来构建软件，对象是软件系统中的基本构成单位

类

- 具有相似属性和方法的对象，可以用类来进行统一描述。
  - 例：张XX，李XX，都可以用学生类来描述
- 类是对象的抽象描述，对象是类的一个实例
  - 例：学生类是张XX的抽象描述，张XX是学生类的一个实例
- 类在抽象的时候，可以根据实际需求，进行不同层次的抽象
  - 例：人类、学生类、大学生类，均可以是张XX的抽象描述

类的定义

- 类是一组只有共同特征的所有对象成员的抽象描述
- 定义
  - 类的名称
  - 类的属性
  - 类的操作方法
- 已定义的类，进行实例化，就创建了一个类的对象
- 每个类的对象们的属性值可以不一样，但操作方法都一样
- 使用关键字class进行定义
  - class ClassName:

        类的主体

类的定义

```
class Student:
    def __init__(self, name, score):
        self.name=name
        self.score=score

    def print_name(self):
        print(self.name)

    def print_score(self):
        print(self.score)


stu1=Student("Tom",99)
stu2=Student("Jenny",90)
stu1.print_name()
stu2.print_score()
print(stu1.score)
print(stu2.name)
```

```
================ RES
Tom
90
99
Jenny
```

- __init__是类的初始化方法，当创建一个类的实例时会调用该方法
- print_name, print_score等，是类的方法，表示对类的操作
- self在类内，表示该类的实例对象本身
- self.name，self.score都是类的属性
- 可以通过类名( )创建一个类的实例对象
  - stu1 =Student(“Tom”,99)
  - 创建对象时，自动调用初始化方法
- 可以使用对象.属性和对象.方法来使用类定义的属性和方法

方法其实就是类内的函数，定义和使用方式与函数一样，但方法的第一个参数总是self，调用时不需要传递

类的数据封装

```
class Student:
    def __init__(self, name, score):
        self.__name=name
        self.__score=score

    def print_name(self):
        print(self.__name)

    def print_score(self):
        print(self.__score)

    def modify_name(self,name):
        self.__name=name
    def modify_score(self,score):
        self.__score=score


stu1=Student("Tom",99)
stu1.modify_name("Tommy")
stu1.modify_score(100)
stu1.print_name()
stu1.print_score()
```

```
==============
Tommy
100
```

- 将对象的属性和方法实现代码都封装起来，只需要通过方法来操作类
- 可以定义更多方法，实现对象及其属性的各种操作行为
  - 比如可以定义get_grade方法，实现对成绩的评级
- 通常不允许通过“对象.属性”这种方式直接访问和修改对象的内部属性
  - 私有属性（private）：在变量前使用两个下划线
    - 如：__name
- 通过调用“对象.方法”这种方式来访问或修改内部属性。因为方法可以做参数检查等，以规范的方式调用内部属性
- 私用方法：在方法前使用两个下划线，如__fun()

类的继承和多态

- 根据抽象程度不同，可以从父类继承子类
  - 如可以从Animal类，继承Dog类
- 子类自动具有父类的属性和方法，并且可以定义新的属性和方法
- 方法重载：如果父类中的方法不满足需求，可以在子类中重新定义该方法（类的多态）

类的继承和多态

```
class Animal:
    def __init__(self, name, color):
        self.name=name
        self.color=color

    def run(self):
        print(self.name,"is running")

    def get_info(self):
        print(self.name, self.color)

class Dog(Animal):
    def __init__(self,name,color,race):
        super().__init__(name,color)
        #利用super()调用其父类
        self.race=race
    #重新定义了初始化方法，并且增加了一个race属性

    def eat(self):
        print("the dog is eating")
    #增加了新的方法

    def get_info(self):
        print(self.name, self.color,self.race)
    #重新定义了get_info方法（多态）

a=Animal("duoduo","green")
a.run()
a.get_info()

b=Dog("benben","yellow","haski")
b.eat()
b.run()
#从父类中继承的方法
b.get_info()
```

```
================ REST
duoduo is running
duoduo green
the dog is eating
benben is running
benben yellow haski
>>>
```

模块

- 将一个完整的程序存储于一个文件中，这种方法仅适用于小型程序
- 当程序规模变大、参与人数变多、程序调用复杂时，典型的操作方法是将程序的不同部分存储于不同的文件当中,称为模块
- 可以在不同的模块文件之中构建程序代码，从而实现复杂的程序协同

模块

- 每个模块以文件的形式，存在于存储器上，文件后缀名为py，也称为脚本文件或源文件
- 在另外一个程序中，可以使用import导入模块，进行程序复用
- 复杂的程序，可以按功能分开，分别存放在不同的模块中
- 标准的python内建了200多个模块，还可以下载更多的免费模块
- 可编写自己的模块

模块

- 使用import 模块名 导入一个模块，需要使用模块名.函数名进行函数的调用
- 使用from 模块名 import 函数名 导入部分函数，可直接使用函数名进行函数调用

```
>>> import math
>>> math.sin(2)
0.9092974268256817
>>> sin(2)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    sin(2)
NameError: name 'sin' is not defined
>>> from math import sin
>>> sin(2)
0.9092974268256817
```

```
>>> from math import sin, cos, tan
>>> sin(1)
0.8414709848078965
>>> cos(1)
0.5403023058681398
>>> tan(1)
1.5574077246549023
>>>
```

- 也可以使用 from 模块名 import 函数1，函数2，....导入多个函数

模块

- 系统内部模块
- 使用dir( )查看模块内的变量与函数
- 使用help( )查看函数使用方法

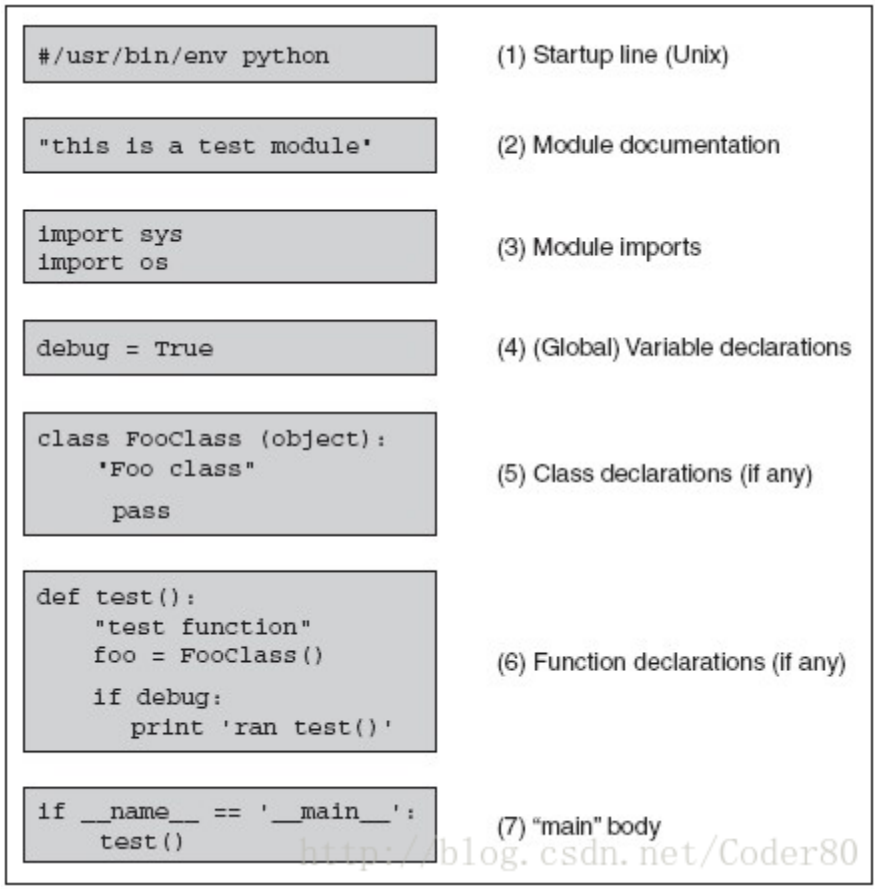
```
>>> help("modules")

Please wait a moment while I gather a list of all available modules...

__future__          autocomplete        idle_test           search
__main__            autocomplete_w      idlelib             searchbase
_ast                autoexpand          imaplib             searchengine
_asyncio            base64              imghdr              secrets
_bisect             bdb                 imp                 select
_blake2             binascii            importlib           selectors
_bootlocale         binhex              inspect             setuptools
_bz2                bisect              io                  shelve
_codecs             browser             iomenu              shlex
_codecs_cn          builtins            ipaddress           shutil
_codecs_hk          bz2                 itertools           signal
_codecs_iso2022     cProfile            json                site
_codecs_jp          calendar            keyword             smtpd
_codecs_kr          calltip_w           lib2to3             smtplib
```

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh'
, 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fm
od', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'is
inf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan'
, 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

```
>>> help("math.sin")
Help on built-in function sin in math:

math.sin = sin(...)
    sin(x)

    Return the sine of x (measured in radians).
```

```
>>> import time
>>> print(time.localtime())
time.struct_time(tm_year=2023, tm_mon=12, tm_mday=20, tm_hour=7, tm_min=44, tm_s
ec=39, tm_wday=2, tm_yday=354, tm_isdst=0)
>>> print(time.strftime("%Y-%X-%x", time.localtime()))
2023-07:44:45-12/20/23
>>>
```

Python的模块结构



1. 起始行
2. 模块文档（模块说明,可通过module.__doc__访问）
3. 模块导入（本模块用到的所有模块）
4. 变量定义（全局变量）
5. 类的定义
6. 函数定义
7. 主程序（检查变量__name__的值，确定是否执行）

模块

- Import导入模块时需要注意系统路径

```
>>> import sys
>>> sys.path
['', 'C:\\Users\\lenovo\\AppData\\Local\\Programs\\Python\\Python37\\Lib\\idleli
b', 'C:\\Users\\lenovo\\AppData\\Local\\Programs\\Python\\Python37\\python37.zip
', 'C:\\Users\\lenovo\\AppData\\Local\\Programs\\Python\\Python37\\DLLs', 'C:\\U
sers\\lenovo\\AppData\\Local\\Programs\\Python\\Python37\\lib', 'C:\\Users\\leno
vo\\AppData\\Local\\Programs\\Python\\Python37', 'C:\\Users\\lenovo\\AppData\\Lo
cal\\Programs\\Python\\Python37\\lib\\site-packages']
```

- 需要导入的模块，需要在系统路径下；或者添加自定义的路径到系统路径

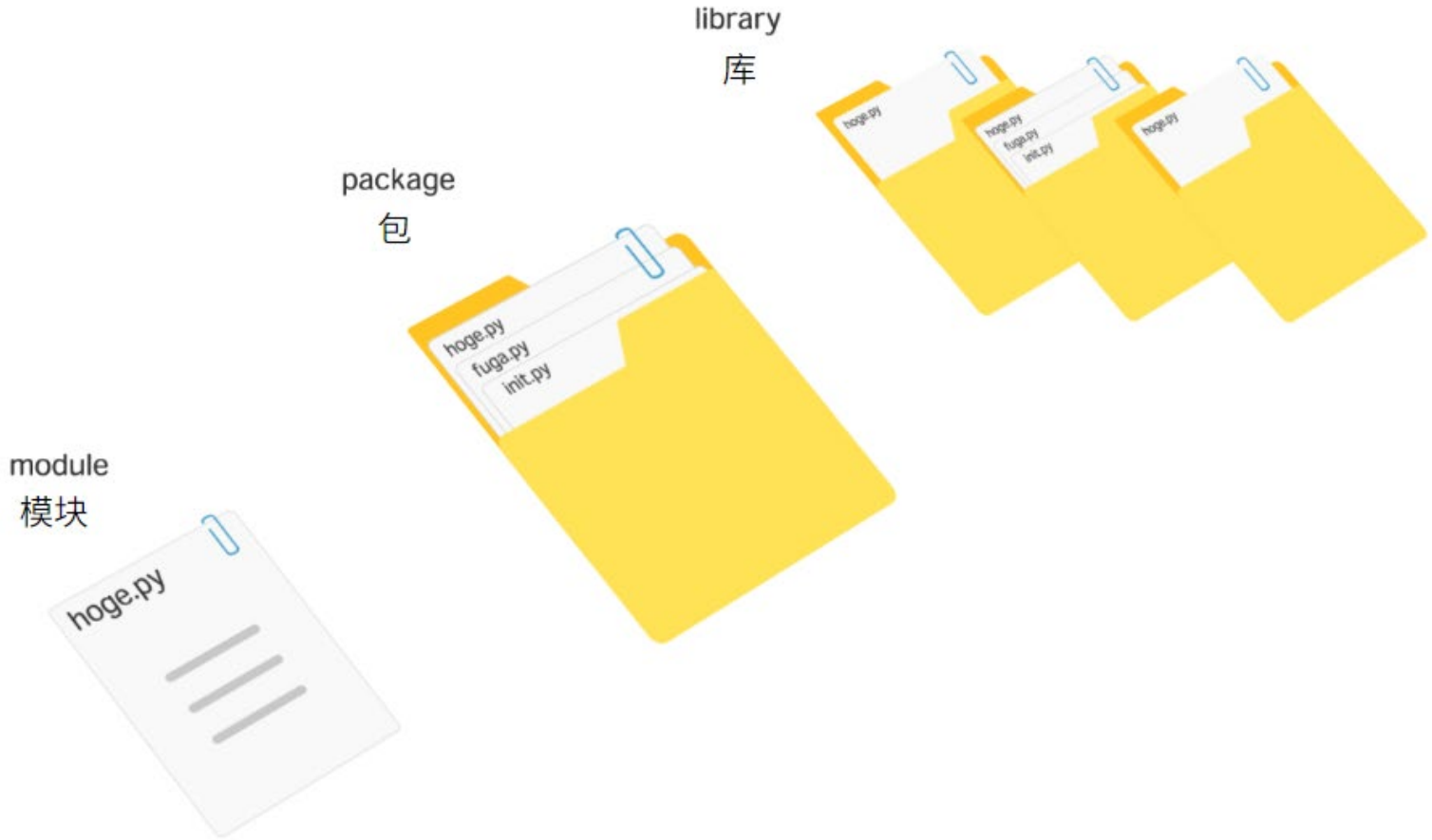
```
>>> sys.path.append("C:\\Users\\lenovo\\")
>>> sys.path
['', 'C:\\Users\\lenovo\\AppData\\Local\\Programs\\Python\\Python37\\Lib\\idleli
b', 'C:\\Users\\lenovo\\AppData\\Local\\Programs\\Python\\Python37\\python37.zip
', 'C:\\Users\\lenovo\\AppData\\Local\\Programs\\Python\\Python37\\DLLs', 'C:\\U
sers\\lenovo\\AppData\\Local\\Programs\\Python\\Python37\\lib', 'C:\\Users\\leno
vo\\AppData\\Local\\Programs\\Python\\Python37', 'C:\\Users\\lenovo\\AppData\\Lo
cal\\Programs\\Python\\Python37\\lib\\site-packages', 'C:\\Users\\lenovo\\']
```

包（package）

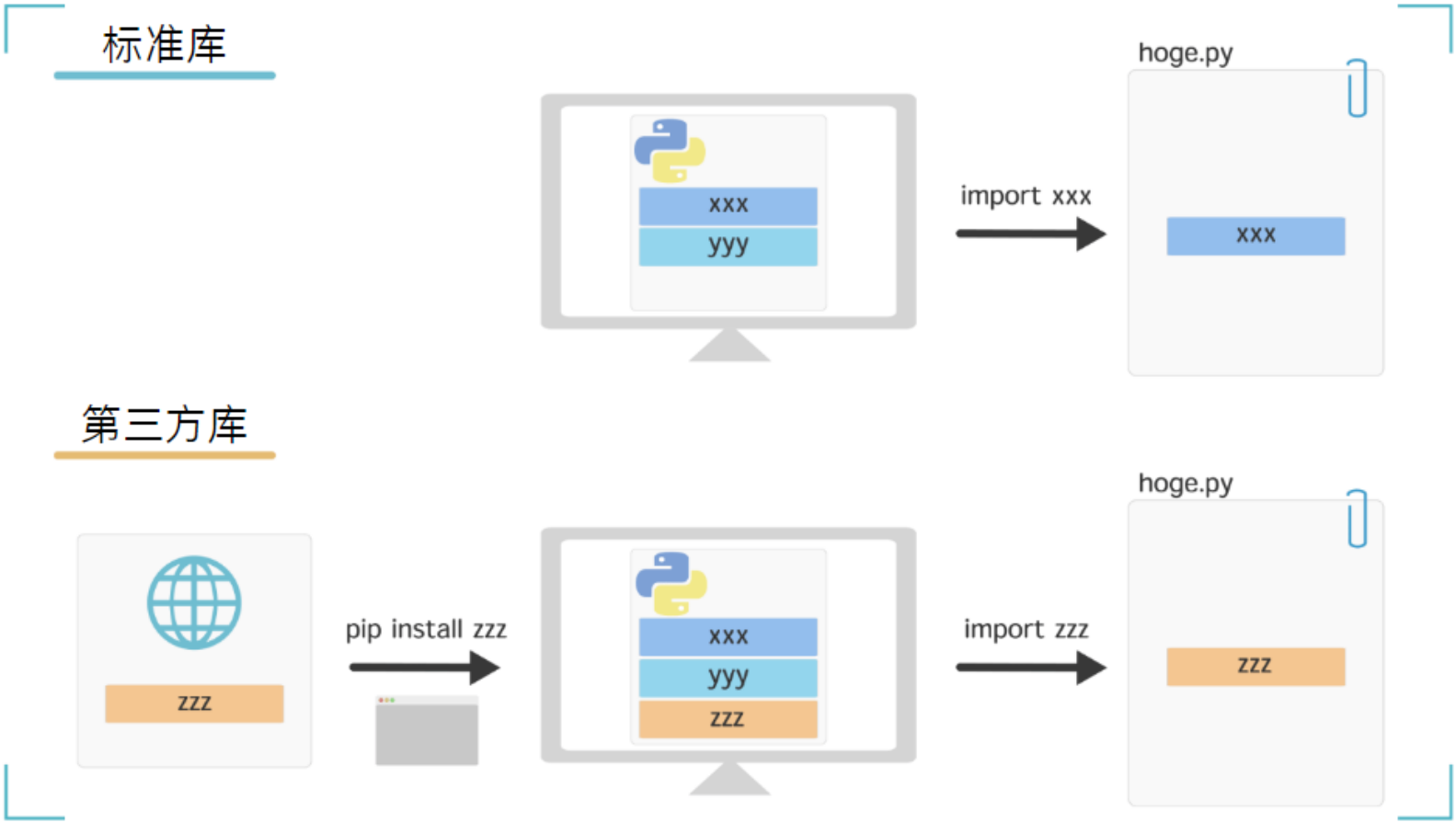
- 为了更好的组织模块，可以把模块分为包
- 包就是模块所在的目录
- 包下面包括:子包、模块以及一个__init__.py文件
  - __init__.py文件可以为空文件，仅供标识文件夹为一个包
- 可以导入包中的模块
  - import a.b.module1
    - 使用全名a.b.module1.fun1( )调用
  - from a.b import module1
    - 使用module1.fun1()进行调用

```
sound/                          顶层包
      __init__.py               初始化 sound 包
      formats/                  文件格式转换子包
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  声音效果子包
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
```

库、包、模块



库、包、模块


标准库
xxx
yyy
import xxx
hoge.py
xxx
第三方库
zzz
pip install zzz
xxx
yyy
zzz
import zzz
hoge.py
zzz

库（Library）

- 常用的第三方库（其他机构开发）：

数据分析
numpy：http://www.numpy.org/　开源数值计算扩展第三方库
scipy：https://pypi.org/project/scipy/　专为科学以及工程计算的第三方库
pandas：http://pandas.pydata.org/　可高效地操作大型数据集的第三方库

网络爬虫
requests：https://pypi.org/project/requests/　简洁且简单的处理HTTP请求的第三方库
scrapy：https://scrapy.org/　快速、高层次的Web获取框架

文本处理
pdfminer：https://pypi.org/project/pdfminer/　从PDF文档中提取各类信息的第三方库
openpyxl：https://pypi.org/project/openpyxl/　处理Microsoft Excel文档的Python第三方库
python-docx：https://pypi.org/project/python-docx/　处理Microsoft Word文档的Python第三方库
beautifulsoup4：https://pypi.org/project/beautifulsoup4/　从HTML和XML文件中解析出数据的第三方库

谢谢！