杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

第6章 函数与模块

6.1 函数的定义与调用

6.2 函数的参数传递

6.3 匿名函数

6.4 函数的递归

6.5 程序结构

6.3 匿名函数

- 函数定义的另一种方法是 lambda 表达式，它定义了一个匿名函数。

```
<函数对象名> = lambda <形式参数列表> : <表达式>
```

对比 def

```
>>> g = lambda x, y: x+y
>>> print( g(4,5) )
9
```

```
def <函数名> (<形式参数列表>) :
    return  <表达式>
```

```
def add(x, y):
    return x+y
```

6.3 匿名函数

- lambda 是一个表达式，而不是一个语句。
- 作为表达式，lambda 返回一个值（即一个新的函数）

```
>>> g = lambda x, y: x+y
>>> g
<function <lambda> at 0x021A12B8>
>>> (lambda x, y: x+y)(4, 5)
9
```

- Lambda能够出现在python语法不允许def出现的场合，用来编写简单的函数，而def定义的函数用来处理更强大的任务。

6.3 匿名函数

练习：程序如下，写出运行结果。

```
f=lambda a,b=2,c=5 : a*a-b*c     #使用默认值参数

print(f(10,15))
print(f(20,10,38))
print(f(c=20,a=10,b=38))    #使用关键字实参
```

运行结果：

```
25
20
-660
```

6.3 匿名函数

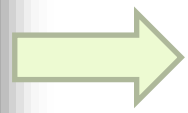
❖ 匿名函数也可嵌套条件分支，完成更复杂的逻辑。

例如，返回x和y中的较大值：

```
>>> mymax=lambda x,y: x if x>=y else y
>>> mymax(2,3)
3
```

练习：用匿名函数改造奇数判断函数isOdd(x)

```
def isOdd(n):
    if n%2==1:
        return 1
    else:
        return 0
```



?

6.3 匿名函数

- 匿名函数可以作为列表或字典的元素

```
>>> ls = [lambda x,y: x+y, lambda x,y: x-y, lambda x,y: x*y]
>>> ls[0](3,5)
8
>>> ls[2](3,5)
15
```

```
>>> dic = {'add':lambda x,y: x+y, 'sub':lambda x,y: x-y, 'mul':lambda x,y: x*y}
>>> dic['add'](3,5)
8
>>> dic['mul'](3,5)
15
```

6.3 匿名函数

- 匿名函数可作为其它函数的参数。

列表的操作函数	含义
list.sort(key=None, reverse=False)	对列表中的元素排序。key是用来进行比较的元素，reverse默认为升序。

```
>>> students = [('王一',89.5,15), ('方鹏',80.0,14), ('陈可',85.5,14)]
>>> # 对比sort的参数key的设置
>>> students.sort()
>>> students
[('方鹏', 80.0, 14), ('王一', 89.5, 15), ('陈可', 85.5, 14)]
>>>
>>> students.sort(key = lambda x:x[1])     #按成绩升序排列
>>> students
[('方鹏', 80.0, 14), ('陈可', 85.5, 14), ('王一', 89.5, 15)]
```

```
the clown ran after the car
and the car ran into the tent
and the tent fell down on the
clown and the car
```

6.3 匿名函数

例：对一段文本进行词频统计，并按词频降序输出统计结果。

```
text = open('1.txt').read()
words = text.split()
dic = {}
for word in words:
    dic[word] = dic.get(word,0) + 1    #词频加1
items = list(dic.items()) #所有数据项组成的列表

items.sort(key = lambda x: x[1], reverse=1) #按词频降序排
for item in items:
    word, count = item
    print('{} = {}'.format(word, count))
```

试运行，思考：如何让相同频次的单词按照字母升序排列？

6.4 函数的递归

- 在函数定义中调用函数自身的方式称为递归。递归在数学和计算机应用上非常强大，能够很简洁地解决问题。
- 数学上的经典递归例子——求阶乘

$$n! = \begin{cases} 1 & n = 0 \\ n * (n-1)! & otherwise \end{cases}$$

递归的实质是把问题分解成：规模缩小的同类子问题，然后递归调用来表示问题的解。

6.4 函数的递归

- 类似于数学归纳法
  - 先证明：当n取第一个值n_0时命题成立
  - 再假设当n_k时命题成立，证明当$n=n_{k+1}$时命题也成立
  - 得出结论：对所有n，命题都成立。

- 递归需要终止条件和递归条件：

$$n! = \begin{cases} 1 & n = 0 \\ n * (n-1)! & otherwise \end{cases}$$

当终止条件不满足时，递归前进；终止条件满足则递归返回。

6.4 函数的递归

❖ 递归的实现

递归出口（确定解）

$$n! = \begin{cases} 1 & n = 0 \\ n * (n-1)! & otherwise \end{cases}$$

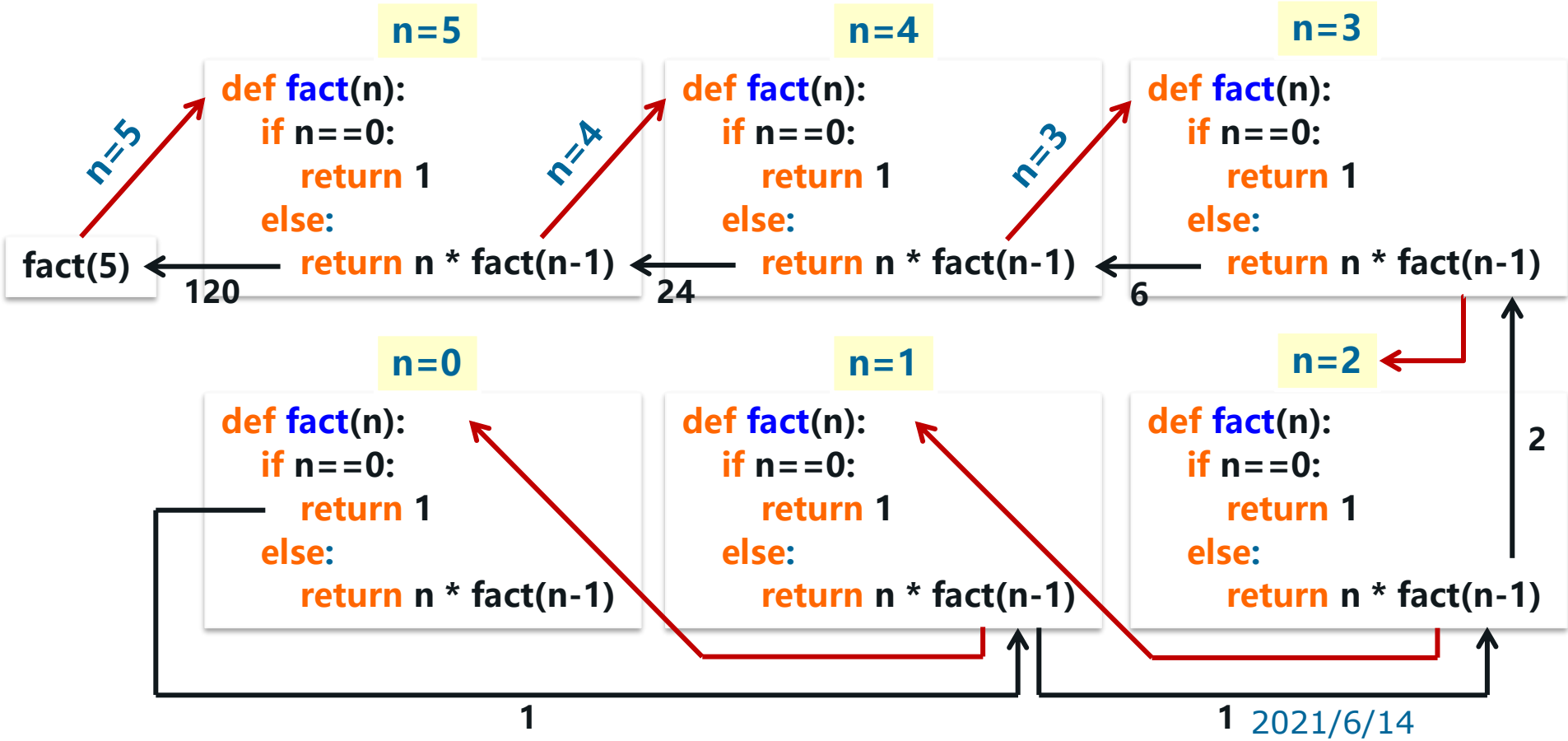
反复调用自身

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)
```

函数+分支结构

6.4 函数的递归

❖ 递归调用的过程



$$Fib(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ Fib(n-1)+Fib(n-2) & n>1\end{cases}$$

6.4

【例】计算 Fibonacci 数列第15项的值。

Fibonacci数列除了前两项之外，后面每项的值均等于前两项之和。

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

存在问题：由于有大量的重复运算，效率很低。

6.4 函数的

解决方法：用字典把中间结果保存下来，就不用重复运算了。

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

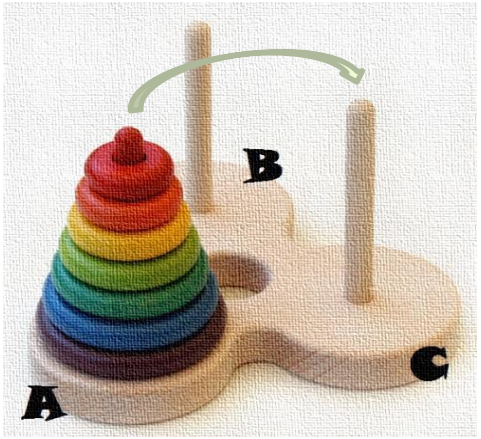
```
fib(5)
→fib(4)+fib(3)
→fib(3)+fib(2)+fib(3)
→fib(2)+fib(1)+fib(2)+fib(3)
→fib(1)+fib(0)+fib(2)+fib(3)
→1+1+fib(2)+fib(3)
                    +fib(3)


                    +fib(1)
```

```
pre = {0:1, 1:1}
def fib(n):
    if n in pre:
        return pre[n]
    else:
        newvalue =  fib(n-1)+fib(n-2)
        pre[n] = newvalue
        return newvalue
```

6.4 函数的递归

【例】汉诺塔问题



有 A、B、C 三根柱子，A 柱上叠放了 N 个上小下大的圆盘，现要把 A 柱上的所有圆盘移动到 C 柱上。移动规则：

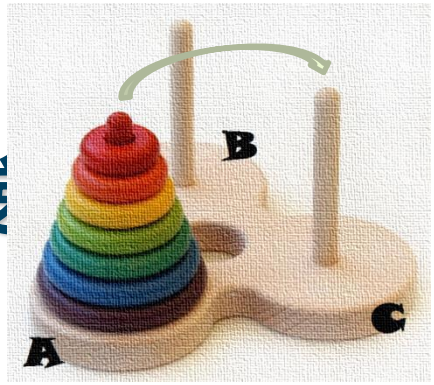
1) 一次只移一个盘；

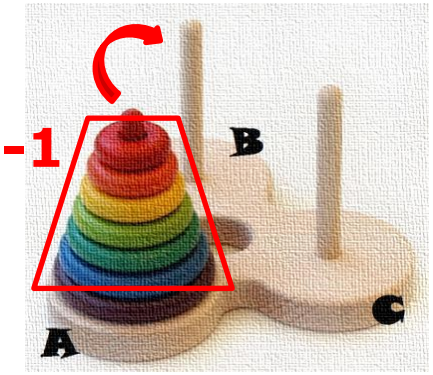
2) 每根柱上的圆盘必须始终保持上小下大的叠放顺序；

3) 可以借助 B 柱做中转。

求：移动轨迹。

6.4 函数的递





【例】汉诺塔问题

```
hanoi(n, A, B, C) # 将 n 个盘从 A 移到 C (B中转)
```

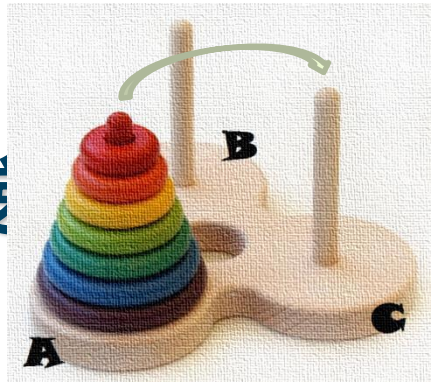
```
hanoi(n-1, A, C, B)
move(A, C)
hanoi(n-1, B, A, C)
```

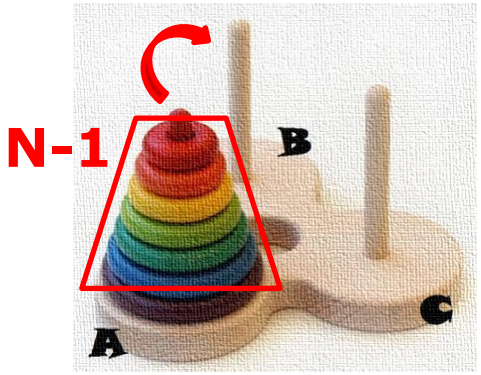
当n>1时

```
move(A, C)   # 直接把A上的盘移到C上
```

当n=1时

6.4 函数的递





【例】汉诺塔问题

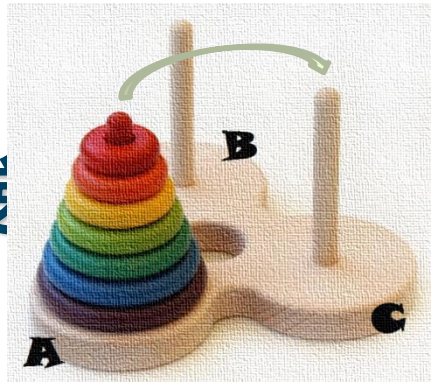
思考：如何同时得到移动次数?

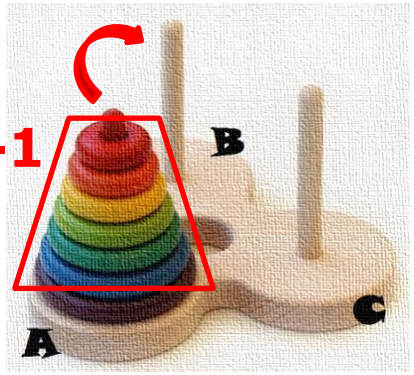
```
def move(src, dst):
    print("%c-->%c" % (src, dst))

def hanoi(n, src, tmp, dst):
    if n == 1:
        move(src, dst)
    else:
        hanoi(n-1, src, dst, tmp)
        move(src, dst)
        hanoi(n-1, tmp, src, dst)

hanoi(3, 'A','B','C')
```

6.4 函数的递





【例】汉诺塔问题

```
cnt = 0
def move(src, dst):
    global cnt
    cnt += 1
    print('%d: %c -> %c' % (cnt, src, dst))

def hanoi(n, src, tmp, dst):
    if n==1:
        move(src, dst)
    else:
        hanoi(n-1, src, dst, tmp)
        move(src, dst)
        hanoi(n-1, tmp, src, dst)

hanoi(3, 'A', 'B', 'C')
print('total = %d' % cnt)
```

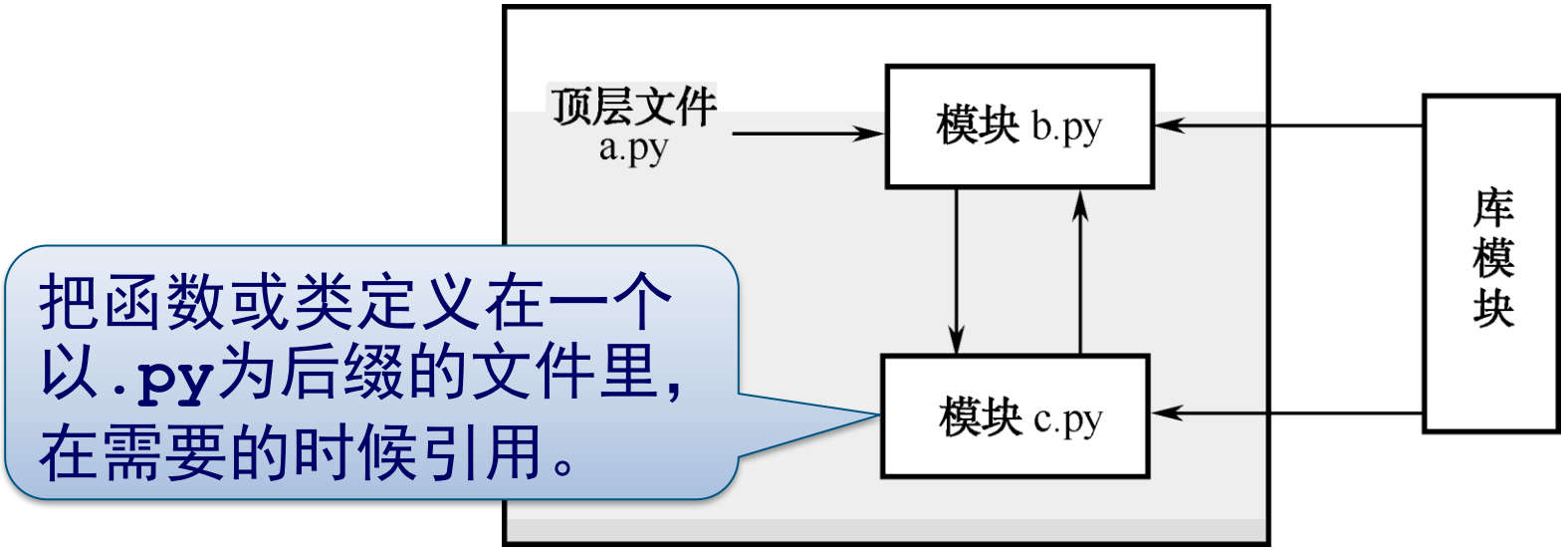
思考：如何同时得到移动次数?

运行结果：

```
1: A -> C
2: A -> B
3: C -> B
4: A -> C
5: B -> A
6: B -> C
7: A -> C
total = 7
```

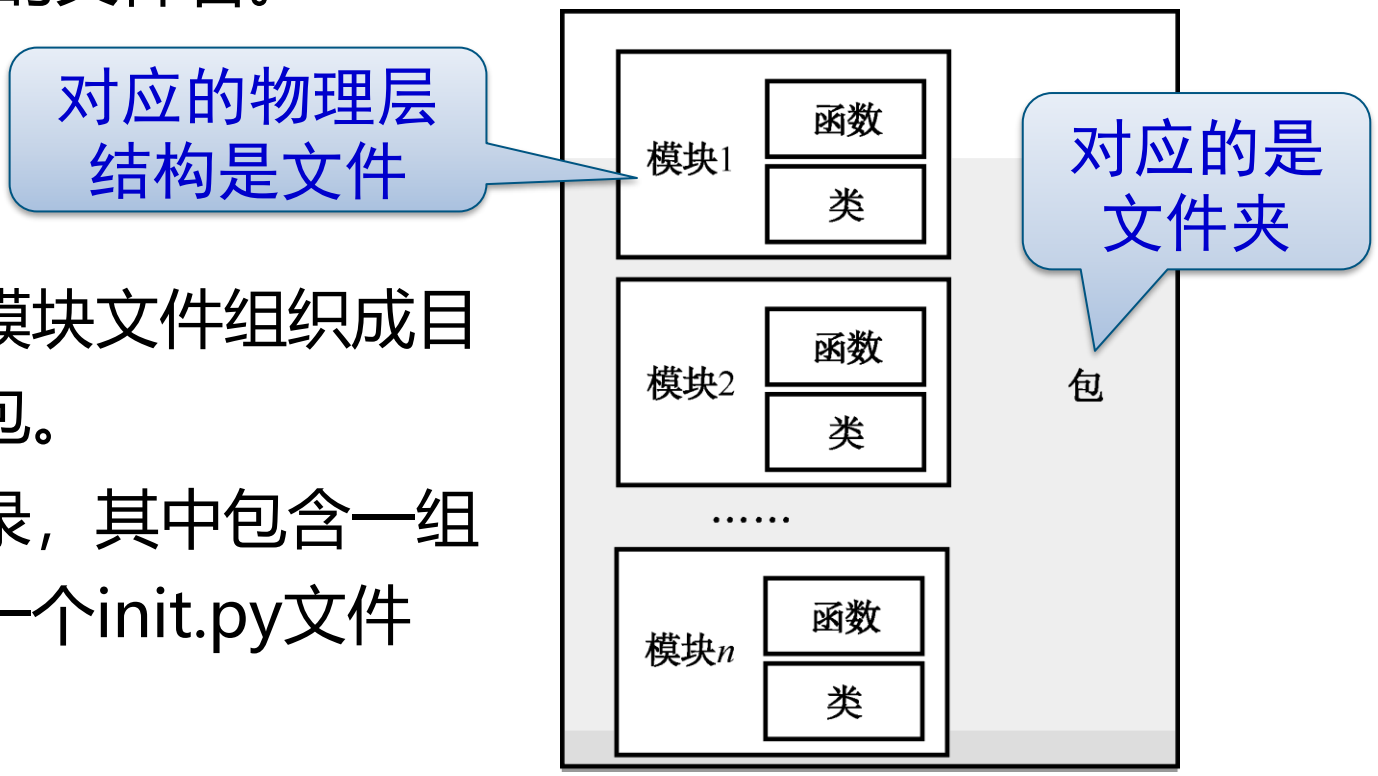
6.5 程序结构

- Python使用模块化的方法来组织程序，一个Python程序就是一个模块化的系统，它由一个顶层文件和多个模块文件组成。



6.5 程序结构

- 一个Python代码的文件就是一个模块，模块名就是Python文件的文件名。



- 可以把多个模块文件组织成目录，称之为包。
- 包是一个目录，其中包含一组模块文件和一个init.py文件

模块的定义

❖ 1. 标准库模块

标准库模块是Python自带的函数模块。Python提供了大量的标准库模块，实现了很多常见功能，包括数学运算、字符串处理、操作系统功能、网络和Internet编程、图形绘制、图形用户界面创建，等等，这些为应用程序开发提供了强大支持。

模块的定义

❖ 2. 用户自定义模块

用户自定义一个模块就是建立一个Python程序文件。下面是一个简单的模块，程序文件名为sample.py

```
def hello(name):
    print("Hello,", name)
```

可以通过执行import语句来导入Python模块，例如：

```
>>> import sample
>>> sample.hello("Python")
Hello, Python
```

模块的导入

1. 使用import语句导入模块

语法格式如下:

```
import <模块1>
import <模块2>
……
import <模块n>
```

模块的导入

用此方法将多个库导入当前命名空间时，可能造成新导入进来的名称覆盖掉当前命名空间中已有的相同函数名称，建议慎用这种方法。

2. 使用from语句导入特定的项目

从一个模块中导入指定的属性或名称到当前程序命名空间：

```
from <模块名> import <函数名>
```

也可以导入模块的所有项目到当前的命名空间：

```
from <模块名> import *
```

模块的引用

【例】创建一个fibo.py模块，其中包含两个求Fibonacci数列的函数，然后导入该模块并调用其中的函数。

```
def fib1(n):
    if n==0 or n==1:
        return 1
    else:
        return fib1(n-1) + fib1(n-2)

#用字典
pre = {0:1, 1:1}
def fib2(n):
    if n in pre:
        return pre[n]
    else:
        newvalue =  fib2(n-1) + fib2(n-2)
        pre[n] = newvalue
        return newvalue
```

常用标准库——random库

- 随机数在计算机应用中十分常见，Python内置的random库主要用于产生各种分布的伪随机数序列。

函数	描述
seed(a=None)	初始化随机数种子，默认值为当前系统时间
random()	生成一个[0.0, 1.0)之间的随机小数
randint(a, b)	生成一个[a,b]之间的整数
getrandbits(k)	生成一个k比特长度的随机整数
randrange(start, stop[, step])	生成一个[start, stop)之间以step为步数的随机整数
uniform(a, b)	生成一个[a, b]之间的随机小数
choice(seq)	从序列类型(例如：列表)中随机返回一个元素
shuffle(seq)	将序列类型中元素随机排列，返回打乱后的序列
sample(pop, k)	从pop类型中随机选取k个元素，以列表类型返回

常用标准库——random库

- random库的使用

```
>>> import random
>>> random.random()   #生成0~1.0之间的随机小数
0.7281639603025086
>>> random.randint(1,10)   #生成1~10之间的随机整数
4
>>> random.uniform(1,10)   #生成1~10之间的随机小数
1.9098371869738693
>>> ls = list(range(1,10))
>>> ls
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> random.shuffle(ls)
>>> ls
[2, 4, 5, 7, 9, 3, 1, 6, 8]
>>> random.choice(range(1,100))
37
```

常用标准库——random库

❖ 思考：

1）随机生成100内的10个整数

2）随机选取0到100之间的奇数

3）从字符串 'abcdefghij' 中随机选取4个字符

4）随机选取列表 ['apple', 'pear', 'peach', 'orange'] 中的1个字符串

```
>>> from random import *
>>> random()
0.04190433764860668
>>> sample(range(1,100),10)
[13, 88, 97, 75, 48, 3, 8, 98, 42, 7]
>>> randrange(1,100,2)
65
>>> sample('dafdasfa',4)
['s', 'd', 'a', 'f']
>>> choice(['dfad','323','dfk'])
'dfk'
```

```
>>> dic = {'Mike': 21, 'Anny': 23, 'Tom': 20}
>>> sorted(dic.items())
[('Anny', 23), ('Mike', 21), ('Tom', 20)]
>>> sorted(dic.items(), key = lambda x:x[1])
[('Tom', 20), ('Mike', 21), ('Anny', 23)]
```

内置函数

❖ sorted( )函数

对字符串、列表、元组、字典等对象进行排序操作。

```
sorted( iterable [, key [, reverse])
```

参数：

对比：列表的sort()方法

- iterable —— 序列
- key —— 用来进行比较的元素，可指定一个元素来进行排序
- reverse —— 排序规则，reverse=True降序，False升序（默认）

返回值：返回重新排序的列表。

内置函数

❖ map( )函数

根据提供的函数对指定序列做映射。

```
map( function, iterable )
```

参数:

- function —— 以参数序列中的每一个元素调用function函数
- iterable —— 序列

返回值：返回包含每次function函数返回值的新列表或迭代器。

```
>>> list(map(lambda x: x**2, [1, 2, 3, 4, 5]))
[1, 4, 9, 16, 25]
```

内置函数

❖ zip( )函数

以可迭代的对象作为参数，将对象中对应的元素打包成一个个的元组。

zip(iterable, ……)

参数： iterable —— 一个或多个序列

返回值：返回元组列表。

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> list(zip(a, b))
[(1, 4), (2, 5), (3, 6)]
```

```
>>> c = [7, 8, 9]
>>> list(zip(a, b, c))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
>>> list(zip(a))
[(1,), (2,), (3,)]
```

内置函数

❖ eval( )函数 和 exec( )函数

eval( ) 是计算表达式，返回表达式的值。

```
>>> x, y = 3, 7
>>> eval('x+3*y+4')
28
```

exec( ) 可运行Python的程序，返回程序运行结果。

```
>>> exec('print("hello world!")')
hello world!
```

内置函数

all( )函数 和 any( )函数

all( )和any( )函数将可迭代的对象作为参数。

- 参数都是True时，all( ) 函数才返回True，否则返回False。
- 参数只要有一个True，any( )函数就返回True，参数全是False时，才返回False。

```
>>> n = 47
>>> all([1 if n%k!=0 else 0 for k in range(2, n)])
True
>>> n = 15
>>> all([1 if n%k!=0 else 0 for k in range(2, n)])
False
```