第5章 函数与模块

AI程序设计课程组

目录页

函数的定义和调用

函数的定义

- 函数是组织好的，可重复使用的，用来实现单一或相关联功能的代码段，它能够提高应用的模块性和代码的重复利用率。
- 函数是一种功能抽象

函数定义与调用

- 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号 ()。
- 任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于定义参数。
- 函数内容以冒号起始，并且缩进。
- return [表达式] 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

def 函数名(参数列表):

函数体

return 返回值列表

函数的定义和调用

- 这是一个自定义的函数

```
def PrintClassInfo():
    print('------------------------------------')
    print('Welcome to the class of AI Programming')
    print('------------------------------------')
```

- 定义了函数之后，想要让这些代码能够执行，需要调用函数。通过“函数名()”即可完成调用。

```
PrintClassInfo()
```

```
------------------------------------
Welcome to the class of AI Programming
------------------------------------
```

函数的定义和调用

- 定义一个函数：给了函数一个名称，指定了函数里包含的参数，和代码块结构。
- 函数的基本结构完成以后，可以通过另一个函数调用执行，也可以直接从 Python 命令提示符执行。

```
#定义函数，执行输出课程信息的任务
#-------------------------------------
def PrintClassInfo(classname,duaration):
    print('{0:}的课程时长为：{1:d}课时'.format(classname,duaration))
```

```
#调用函数
#-------------------------------------
PrintClassInfo('AI Programming',64)
```

```
AI Programming的课程时长为：64课时
```

函数的定义和调用

■ 程序调用一个函数需要执行以下四个步骤：

（1）调用程序在调用处暂停执行；

（2）在调用时将实参复制给函数的形参；

（3）执行函数体语句；

（4）函数调用结束给出返回值，程序回到调用前的暂停处继续执行。

函数的参数

参数的概念


```
#定义函数，执行输出课程信息的任务
#-------------------------------------
def PrintClassInfo(classname,duaration):
    print('{0:}的课程时长为：{1:d}课时'.format(classname,duaration))
```

```
#调用函数
#-------------------------------------
PrintClassInfo('AI Programming',64)
```

```
AI Programming的课程时长为：64课时
```

- classname和duration为函数PrintClassInfo()的参数

参数传递

- 不可变类型，在被调用的函数中原对象的值不发生变化

  包括int，float，bool，str，元组

- 可变数据类型，在被调用的函数中原对象的值发生变化

  包括列表，字典，集合

不可变数据类型

- 调用函数后，变量赋值 a=6 后再赋值 a=a+18，这里实际是新生成一个 int 值对象 24

```
def ChangeInt(a):
    a+=18
    print('输出改变后的数据',a)

#--------------------------------------
#调用函数
#--------------------------------------
tempData=6
ChangeInt(tempData)
print('输出原始数据',tempData)
```

```
输出改变后的数据 24
输出原始数据 6
```

可变数据类型

- 变量赋值 listA=[1,3,5] 后再赋值 listA+=[6,8,10] 则是将 listA 的元素进行扩充，直接在原对象上修改。

```
def ChangeListData(listA):
    listA+=[6,8,10]
    print('输出改变后的数据',listA)

#调用函数
tempListData=[1,3,5]
ChangeListData(tempListData)
print('输出原始数据',tempListData)
```

```
输出改变后的数据 [1, 3, 5, 6, 8, 10]
输出原始数据 [1, 3, 5, 6, 8, 10]
```

参数设定

- 顺序传入
- 关键词参数传入
- 默认参数
- 不定长参数

函数调用过程中传入参数的4种方式

顺序传入

- 按照输入参数列表的顺序传入，所有参数不可为空，否则出错

```
def PrintClassInfo(classname,duaration):
    print('{0:}的课程时长为：{1:d}课时'.format(classname,duaration))
```

```
#必备参数，调用PrintClassInfo()必须传入，否则出错
PrintClassInfo('AI Programming',40)
PrintClassInfo('Machine Learning')
```

```
AI Programming的课程时长为：40课时
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-f6efc4d27224> in <module>
      1 #必备参数，调用PrintClassInfo()必须传入，否则出错
      2 PrintClassInfo('AI Programming',40)
----> 3 PrintClassInfo('Machine Learning')
```

关键词参数

- 关键字参数和函数调用关系紧密，函数调用使用关键字参数来确定传入的参数值。
- 使用关键字参数允许函数调用时参数的顺序与声明时不一致，因为 Python 解释器能够用参数名匹配参数值。

```
def PrintClassInfo(classname,duaration):
    print('{0:}的课程时长为：{1:d}课时'.format(classname,duaration))
```

```
#关键字参数，调用PrintClassInfo()时，如果利用关键字标识传入参数，则数
PrintClassInfo(duaration=40,classname='AI Programming')
PrintClassInfo('Machine Learning',duaration=40)
```

```
AI Programming的课程时长为：40课时
Machine Learning的课程时长为：40课时
```

默认参数

- 输入的参数可以是事先设定好赋值，也就是默认值。在调动函数的时候，可以不输入参数，函数内部会直接调用默认参数值。以下实例中如果没有传入 duaration 参数，则使用默认值。

```
def PrintClassInfo(classname,duaration=40):
    print('{0:}的课程时长为：{1:d}课时'.format(classname,duaration))

#关键字参数，调用PrintClassInfo()时，因为设置了默认参数，则默认参数可
#PrintClassInfo()
PrintClassInfo('Machine Learning')
```

不定长参数

- 当需要一个函数能处理比当初声明时更多的参数。这些参数叫做不定长参数。

```
def printinfo( arg1, *vartuple ):
    print('输出')
    print(arg1)
    for var in vartuple:
        print(var)
```

```
print('只传入1个参数------------------')
printinfo( 10 );
print('传入多个参数------------------')
printinfo( 70, 60, 50,90 );
```

```
只传入1个参数------------------
输出
10
传入多个参数------------------
输出
70
60
50
90
```

不定长参数

- 函数定义时，*可以将按位置传递进来的参数“打包”成元组(tuple)类型
- 函数调用时，*可以“解压”待传递到函数中的元组、列表、集合、字符串等类型，并按位置传递到函数入口参数中

```
def printinfo( arg1, *vartuple ):
    print('输出')
    print(arg1)
    for var in vartuple:
        print(var)
printinfo( 70, 60, 'demo',[78,86] )
```

```
输出
70
60
demo
[78, 86]
```

不定长参数

*args 传入的时候，如果调用函数使用关键词传入参数时会出错

**kwargs 的出现便是解决需要传入特定关键词参数的情况

```
def funa(classname,**args):
    if(args.get('duaration')):
        print(classname,'课时为：',args['duaration'])
    if(args.get('times')):
        print(classname,'上课次数为：',args['times'])
    print('Welcome to {}'.format(classname))

funa(classname='AI',duaration=64)
funa(classname='AI',duaration=64,times=8)
```

```
AI 课时为： 64
Welcome to AI
AI 课时为： 64
AI 上课次数为： 8
Welcome to AI
```

return 语句

- return语句选择性地向调用方返回一个表达式。return语句用来退出函数，并将程序返回到函数被调用的位置继续执行

```
def sum1(tempA, tempB ):
    # 返回2个参数的和."
    total = tempA + tempB
    print('传入的两个数的加和为：%d'%total)
    return total;

# 调用sum函数
total = 30+sum1(10, 20)
print('输出总和为:{}'.format(total))
```

```
传入的两个数的加和为：30
输出总和为:60
```

- **函数可以没有return，此时函数并不返回值。函数也可以用return返回多个值，多个值以元组类型保存。**

变量的作用域

全局变量和局部变量

- 一个程序的所有的变量并不是在哪个位置都可以访问的。变量的作用域决定了在哪一部分程序你可以访问哪个特定名称的名称。
- 两种最基本的变量作用域如下：
- 全局变量
- 局部变量

局部变量

- 局部变量就是在函数内部定义的变量。
- 局部变量的作用域是函数内部，意味着它只在定义它的函数中有效，一旦函数结束就会消失。

```
listdemo=['AI','Programming']
def demofunc(arg1):
    listdemo=[]
    listdemo.append(arg1)
```

```
demofunc('Python')
print(listdemo)
```

```
['AI', 'Programming']
```

全局变量

- 全局变量定义在函数外，拥有全局作用域。
- 全局变量可以在整个程序范围内访问。
- 如果出现全局变量和局部变量名字相同的情况，则在函数中访问的是局部变量。

In [38]:
```
listdemo=['AI','Programming']
def demofunc(arg1):
    #listdemo=[]
    listdemo.append(arg1)
```

In [39]:
```
demofunc('Python')
print(listdemo)
```

```
['AI', 'Programming', 'Python']
```

global函数

- **Python中定义函数时，若想在函数内部对函数外的变量进行操作，就需要在函数内部将其声明其为global 变量。**
  - **global是Python中的全局变量关键字。**
  - **变量分为局部变量与全局变量，局部变量又可称之为内部变量。**
  - **局部变量只能被对象或函数内部引用，无法被其它对象或函数引用。**
  - **全局变量既可以是某对象函数创建，也可以是在本程序任何地方创建。全局变量是可以被本程序所有对象或函数引用。**
  - **global关键字的作用是可以使得一个局部变量为全局变量。**

global函数


```
def sum2(tempA, tempB ):
    # 返回2个参数的和.
    global total
    total = tempA + tempB
    print('传入的两个数的加和为: %d'%total)
    return total;

# 调用sum函数
sum2(10, 20)
print('输出总和为:{}'.format(total))
```

递归函数

目录页

递归函数

- 一个函数的内部可以调用其他函数。但是，如果一个函数在内部不调用其它的函数，而是自己本身的话，这个函数就是递归函数。

- 使用递归，实现阶乘 n! = n * (n-1) * (n-2) * … * 2 * 1 的计算。
- 这个关系给出了另一种方式表达阶乘的方式：

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

■阶乘的例子揭示了递归的2个关键特征：

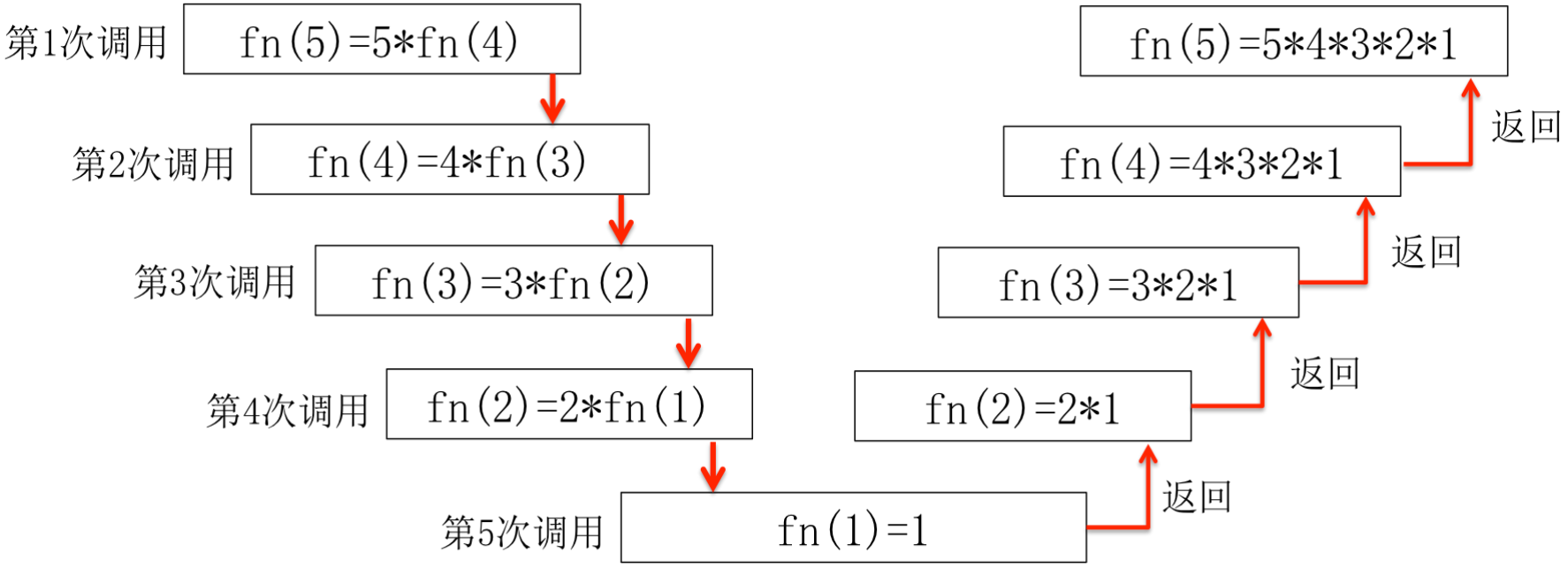
（1）存在一个或多个基例，基例不需要再次递归，它是确定的表达式；

（2）所有递归链要以一个或多个基例结尾。

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

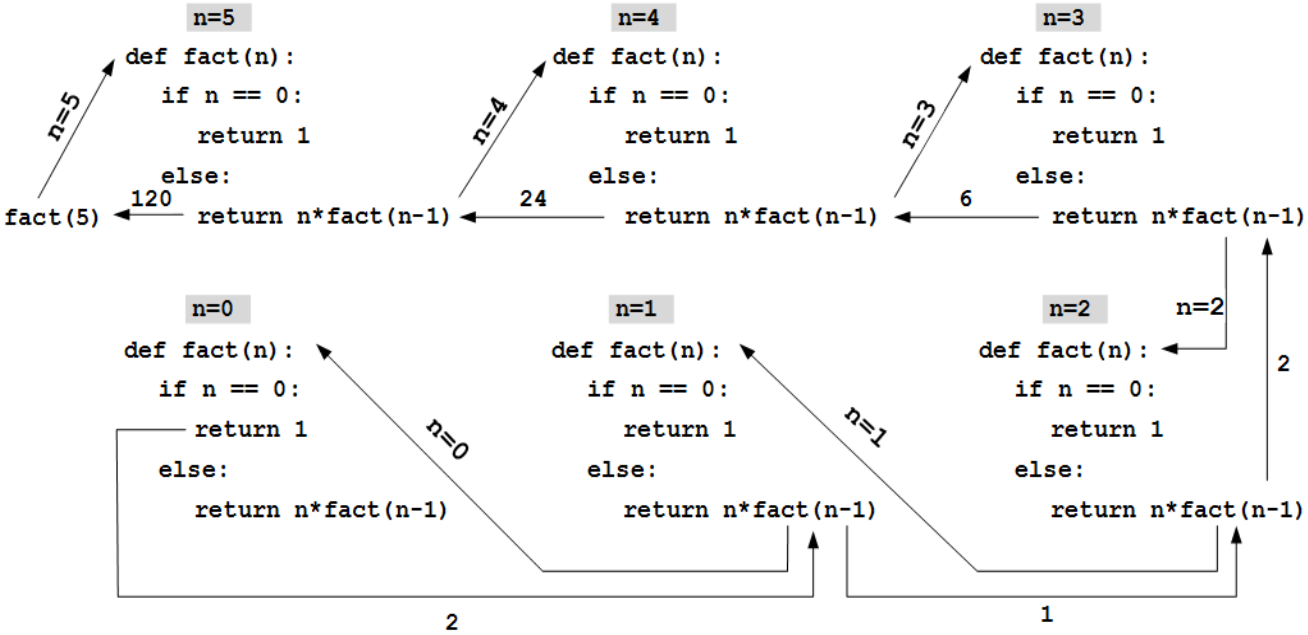
递归函数

■ 使用递归，实现阶乘 n! = n * (n-1) * (n-2) * … * 2 * 1 的计算。



递归函数

- 使用递归，实现阶乘 n! = n * (n-1) * (n-2) * … * 2 * 1 的计算
  -



递归函数

- 使用递归，实现阶乘 n! = n * (n-1) * (n-2) * … * 2 * 1 的计算。

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

fact(5)
```

```
120
```

递归函数

- 字符串反转。对于用户输入的字符串s，输出反转后的字符串
- 解决这个问题的基本思想是把字符串看作一个递归对象

```
import sys
sys.setrecursionlimit(1000)

def reverse(s):
    if len(s) == 0:
        return s
    return reverse(s[1:]) + s[0]

reverse('ABC')
```

'CBA'

设置停止条件

拓展延伸——汉诺塔问题

- 有三个立柱A、B、C。A柱上穿有大小不等的圆盘N个，较大的圆盘在下，较小的圆盘在上。要求把A柱上的圆盘全部移到C柱上，保持大盘在下、小盘在上的规律（可借助B柱）。每次移动只能把一个柱子最上面的圆盘移到另一个柱子的最上面。请输出移动过程。

拓展延伸——汉诺塔问题

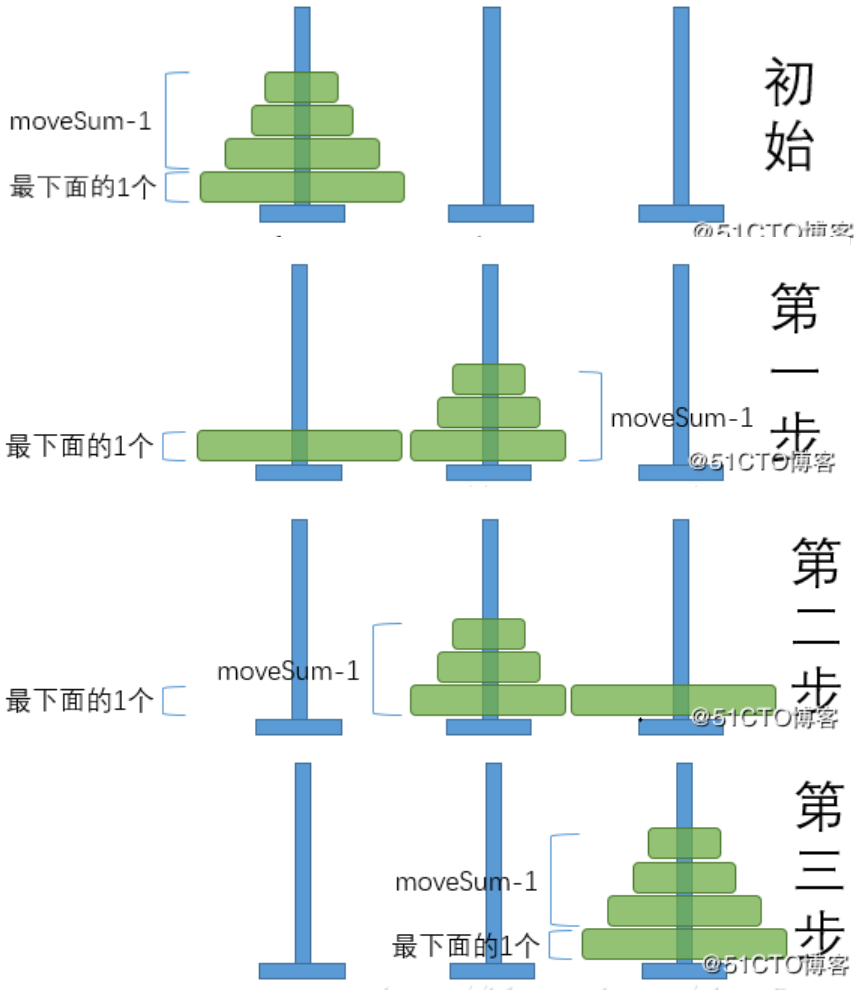
- 规则：
  - 1. 每次移动一个盘子
  - 2. 任何时候大盘子在下面，小盘子在上面
- 方法：
  - 1. n=1: 直接把A上的一个盘子移动到C上，A→C
  - 2. n=2:
    - A. 把小盘子从A放到B上，A→B
    - B. 把大盘子从A放到C上，A→C
    - C. 把小盘子从B放到C上，B→C
  - 3. n=3:
    - A. 把A上的两个盘子，通过C移动到B上，调用递归实现
    - B. 把A上剩下的一个最大的盘子移动到C上，A→C
    - C. 把B上的两个盘子，借助A，移到C上，调用递归

拓展延伸——汉诺塔问题

- 方法：
  - 3. n=3:
    - A. 把A上的两个盘子，通过C移动到B上，调用递归实现
    - B. 把A上剩下的一个最大的盘子移动到C上，A→C
    - C. 把B上的两个盘子，借助A，移到C上，调用递归
  - 4. n=n:
    - A. 把A上的n-1个盘子，通过C移动到B上，调用递归实现
    - B. 把A上剩下的一个最大的盘子移动到C上，A→C
    - C. 把B上的n-1个盘子，借助A，移到C上，调用递归

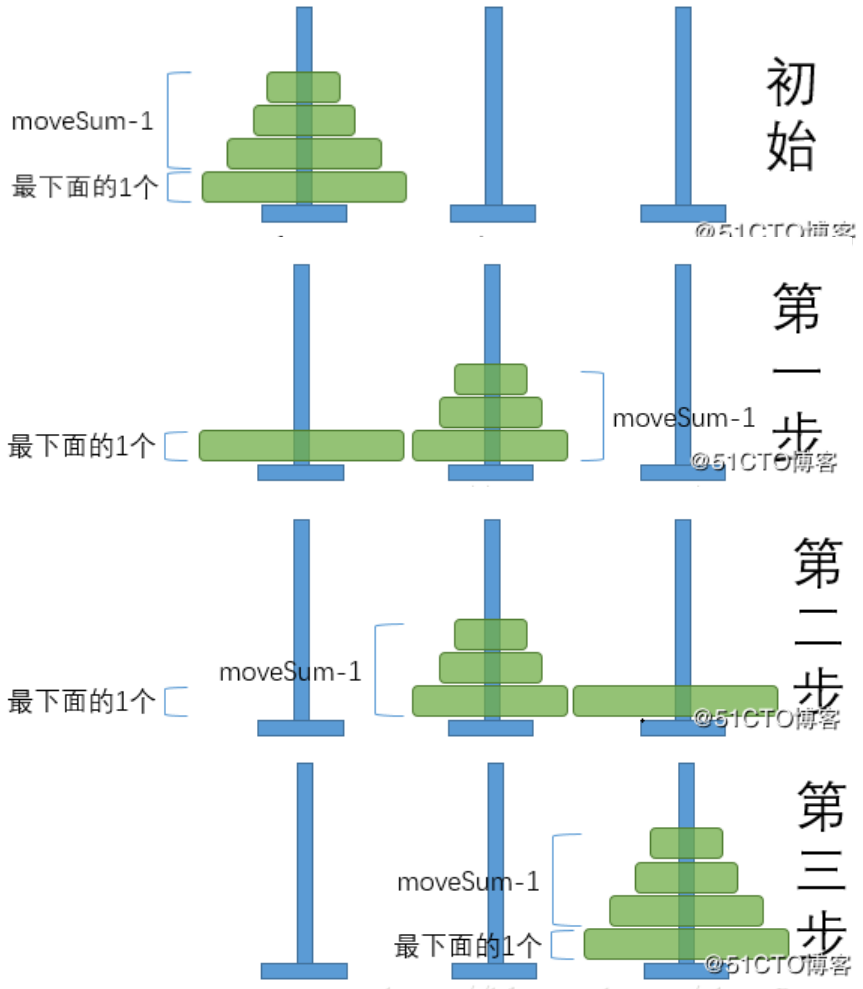
拓展延伸——汉诺塔问题

- 对于"将moveSum个圆盘从from柱移动到to柱（借助by柱）"这个问题，我们可以通过以下三步实现：
  - 将from柱最上面的moveSum-1个圆盘移动到by柱（借助to柱）
  - 将from柱上剩下的那1个圆盘直接移动到to柱
  - 将by柱上的moveSum-1个圆盘移动到to柱（借助from柱）



拓展延伸——汉诺塔问题

- 对于"将moveSum个圆盘从from柱移动到to柱（借助by柱）"这个问题，我们可以通过以下三步实现：
  - 将from柱最上面的moveSum-1个圆盘移动到by柱（借助to柱）
  - 将from柱上剩下的那1个圆盘直接移动到to柱
  - 将by柱上的moveSum-1个圆盘移动到to柱（借助from柱）



拓展延伸——汉诺塔问题

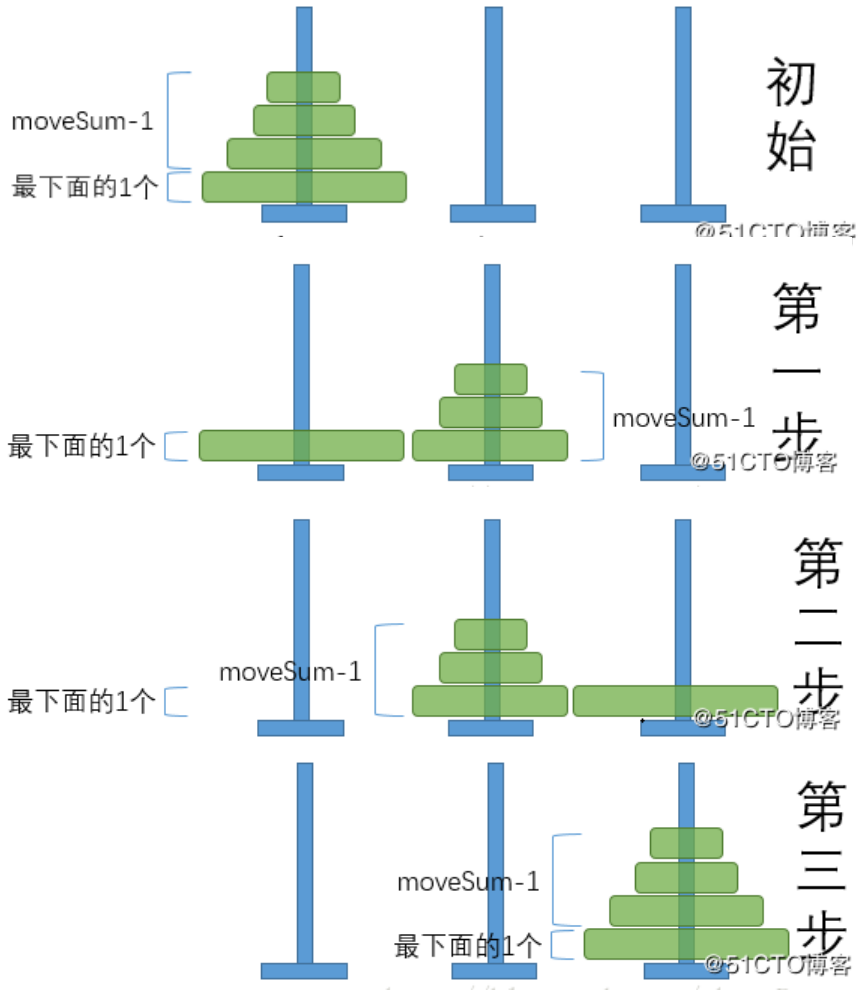
- 对于
  到to
  以通
  -
  -
  -

```
def hano(n, a, b, c):
    if n == 1:
        print(a, "-->", c)
        return None

    hano(n-1, a, c, b)
    print(a, "-->", c)
    hano(n-1, b, a, c)

n1, a1, b1, c1 = 3, "A", "B", "C"
hano(n1, a1, b1, c1)
```

```
A --> C
A --> B
C --> B
A --> C
B --> A
B --> C
A --> C
```



模块化与代码复用

代码的复用和模块化设计

- 函数是程序的一种基本抽象方式，它将一系列代码组织起来通过命名供其他程序使用。
- 函数封装的直接好处是代码复用，任何其他代码只要输入参数即可调用函数，从而避免相同功能代码在被调用处重复编写。
- 代码复用产生了另一个好处，当更新函数功能时，所有被调用处的功能都被更新。

代码的复用和模块化设计

- 当程序的长度在百行以上，如果不划分模块就算是最好的程序员也很难理解程序含义程序的可读性就已经很糟糕了。解决这一问题的最好方法是将一个程序分割成短小的程序段，每一段程序完成一个小的功能。无论面向过程和面向对象编程，对程序合理划分功能模块并基于模块设计程序是一种常用方法，被称为“模块化设计”。

代码的复用和模块化设计

- 模块化设计一般有两个基本要求：
  - 紧耦合：尽可能合理划分功能块，功能块内部耦合紧密；
  - 松耦合：模块间关系尽可能简单，功能块之间耦合度低。
- 要想进行模块化设计，根据计算需求合理划分函数十分重要。一般来说，完成特定功能或被经常复用的一组语句应该采用函数来封装，并尽可能减少函数间参数和返回值的数量。

Python模块

- 模块是一个包含所有你定义的函数和变量的文件，其后缀名是.py。模块可以被别的程序引入，以使用该模块中的函数等功能。

```
import sys
print('命令行参数如下:')
for i in sys.argv:
    print(i)

print('\n\nPython 路径为: ', sys.path, '\n')
```

```
命令行参数如下:
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel
-f
C:\Users\LiuYang\AppData\Roaming\jupyter\runtime\ker
b2634.json
```

import语句

- 想使用 Python 源文件，只需在另一个源文件里执行 import 语句。
- **import 语句**

  import module1[, module2[,... moduleN]

- **from … import 语句**

  from modname import name1[, name2[, ... nameN]]

  from … import *

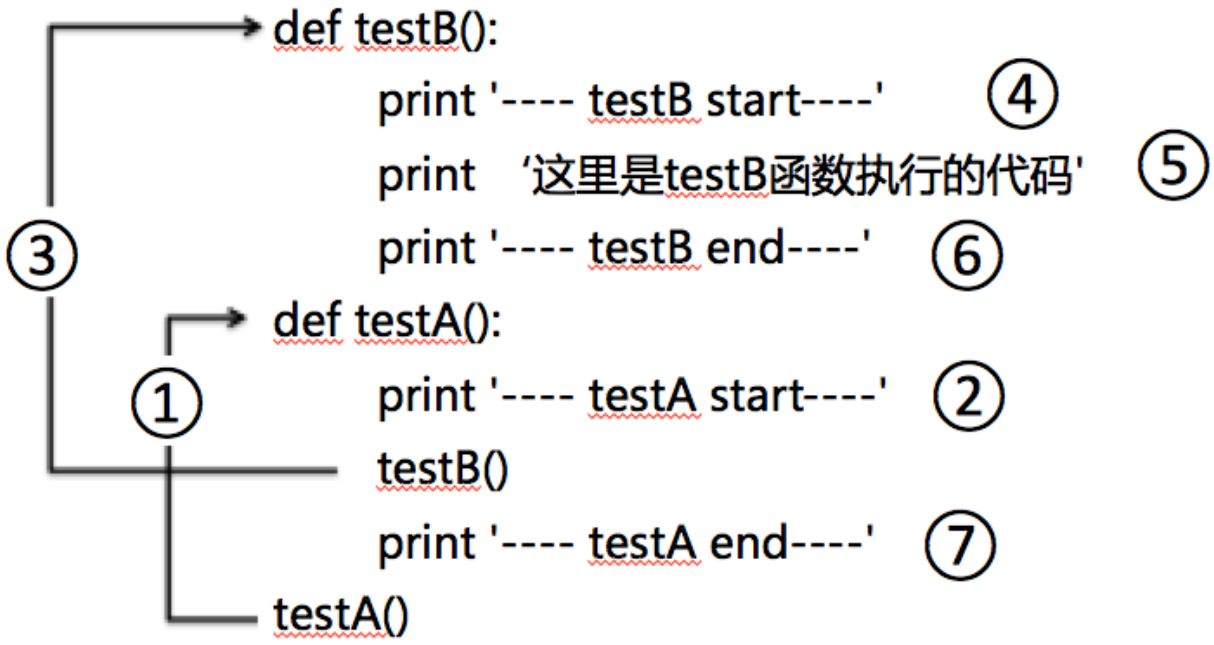
- **as语句**

  from modname import name1 as n1

  import module1 as m1

函数的高级用法

函数的嵌套调用


def testB():
print '---- testB start----'
print '这里是testB函数执行的代码'
print '---- testB end----'
def testA():
print '---- testA start----'
testB()
print '---- testA end----'
testA()
①
②
③
④
⑤
⑥
⑦

闭包

- Python函数是支持嵌套的。如果在一个内部函数中对外部函数作用域（非全局作用域）的变量进行引用，那么内部函数就会被称为闭包。闭包需要满足如下3个条件：
  - 存在于两个嵌套关系的函数中，并且闭包是内部函数；
  - 内部函数引用了外部函数的变量（自由变量）；
  - 外部函数会把内部函数的函数名称返回。

闭包


```
def outer(start=0):
    count = [start]
    def inner():
        count[0] += 1
        return count[0]
    return inner

out = outer(5)
print(out())
```

```
6
```

装饰器

- 假设我们已经开发了一个本有的函数，后续可能会增加临时的需求，例如插入日志，我们可以增加一个包裹函数，由它来负责这些额外的需求，这个包裹函数就是装饰器 。
- 装饰器主要应用在如下场景：
  - 引入日志
  - 函数执行时间统计
  - 执行函数前预备处理
  - 执行函数后清理功能
  - 权限校验
  - 缓存

装饰器

■ 装饰器是一个函数，它需要接收一个参数，该参数表示被修饰的函数。例如，有如下一个装饰器函数：

```
def myDecoration(func):
    def inner():
        print("正在执行内部函数")
        func()
    return inner

def printMessage():
    print("---------欢迎您--------")

pm = myDecoration(printMessage)
pm()
```

```
正在执行内部函数
---------欢迎您--------
```

- 装饰器是个嵌套函数
- 内部函数是一个闭包
- 外部函数接收的是被修饰的函数（func）

装饰器

- 通过在函数定义的前面添加@符号和装饰器名，实现装饰器对函数的包装。给**f1**函数加上装饰器，示例如下：

```
@w1
def f1():
    print('f1')
```

- 此时，程序会自动编译生成调用装饰器函数的代码，等价于：

```
f1 = w1(f1)
```

- 多个装饰器应用在一个函数上，调用顺序是从下至上。

```
@w1
@w2
def f1():
    print('f1')
```

执行顺序：

先执行@w2，后执行@w1

装饰器


```
def w1(func):
    def inner(a, b):
        print("开始验证权限")
        func(a, b)
    return inner

@w1
def test(a, b):
    print("a=%d, b=%d" % (a, b))

test(1, 2)
```

```
开始验证权限
a=1, b=2
```

装饰器

- 如果给装饰器添加参数，需要增加一层包装，先传递参数，然后再传递函数名。

```
def func_arg(args):
    def func(functionName):
        def func_in():
            print("----记录日志-args=%s" % args)
            functionName()
        return func_in
    return func

@func_arg('20230101')
def test():
    print("this is the test")

test()
```

```
----记录日志-args=20230101
this is the test
```

匿名函数——lambda函数

- 匿名函数就是没有名称的函数，也就是不再使用def语句定义的函数。如果要声明匿名函数，则需要使用lambda关键字

<函数名> = lambda <参数列表>: <表达式>

- lambda函数与正常函数一样，等价于下面形式：

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

匿名函数——lambda函数

- 匿名函数就是没有名称的函数，也就是不再使用def语句定义的函数。如果要声明匿名函数，则需要使用lambda关键字

  <函数名> = lambda <参数列表>: <表达式>

- 注意：使用Lambda声明的匿名函数能接收任何数量的参数，但只能返回一个表达式的值。匿名函数不能直接调用print，因为lambda需要一个表达式。

匿名函数——lambda函数

- 简单说，**lambda函数用于定义简单的、能够在一行内表示的函数，返回一个函数类型，实例如下。**

```
f = lambda x, y: x + y
print(type(f))

f(10, 12)
```

```
<class 'function'>

22
```

map函数

- map函数会根据提供的函数对指定的序列做映射。

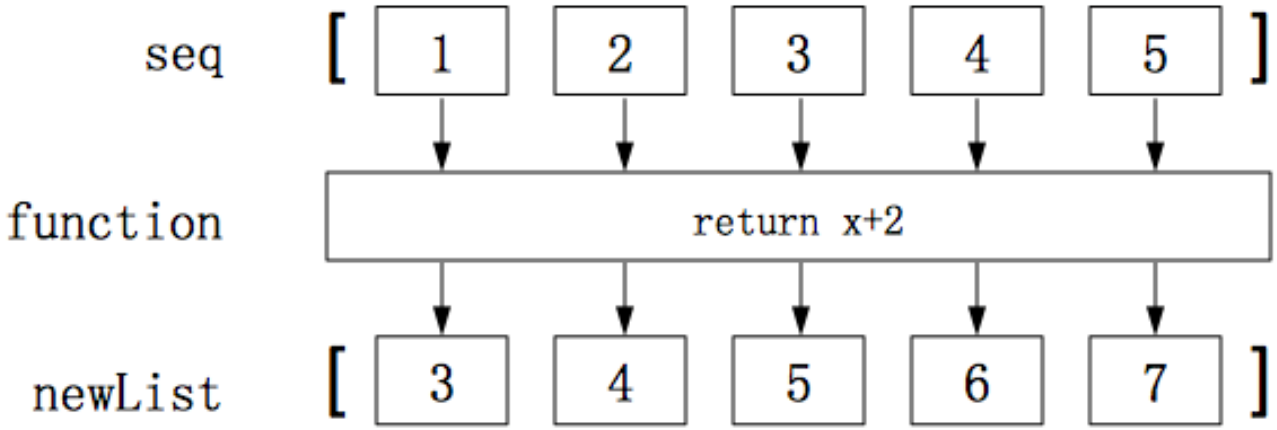
map(function, iterable,…)

- 第1个参数是函数的名称；第2个参数表示支持迭代的容器或者迭代器。
- map函数的作用是以参数序列中的每个元素分别调用function函数，把每次调用后返回的结果保存为对象。

```
func = lambda x:x+2
result = map(func, [1, 2, 3, 4, 5])
print(list(result))
```

```
[3, 4, 5, 6, 7]
```

- 执行过程



filter函数

- **filter函数会对指定序列执行过滤操作**

filter(function, iterable)

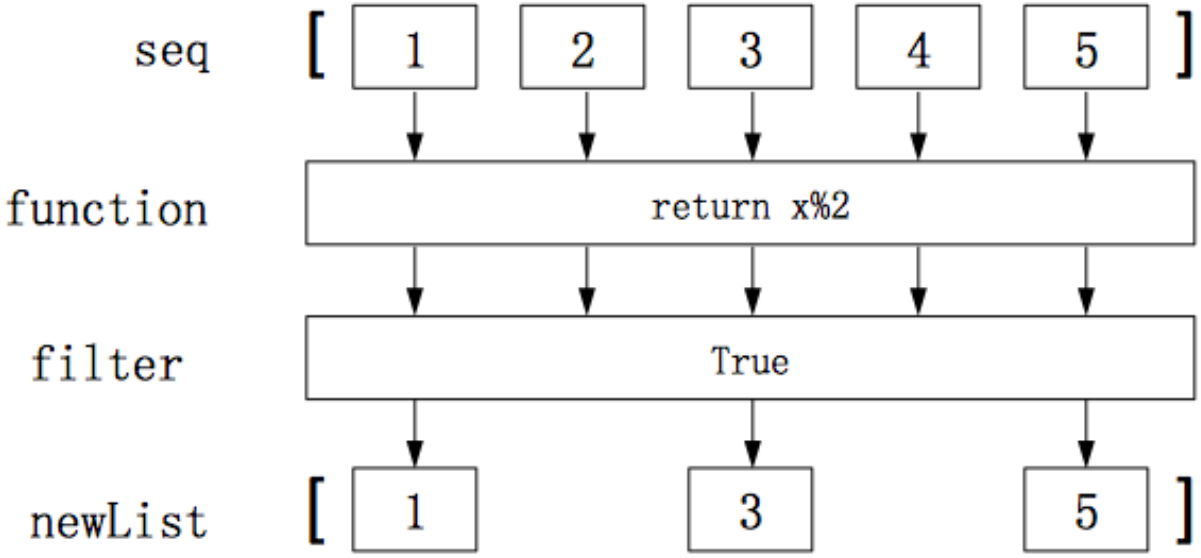
- **第1个参数函数的名称；第2个参数表示的是序列、支持迭代的容器或迭代器。**

```
func = lambda x:x%2
result = filter(func, [1, 2, 3, 4, 5])
print(list(result))
```

```
[1, 3, 5]
```

filter函数

- 执行过程



常用函数简介

datetime库的使用

- 以不同格式显示日期和时间是程序中最常用到的功能。

  Python提供了一个处理时间的标准函数库datetime，它提供了一系列由简单到复杂的时间处理方法。datetime库可以从系统中获得时间，并以用户选择的格式输出。

datetime库的使用

- **datetime库以类的方式提供多种日期和时间表达方式：**
  - **datetime.date：日期表示类，可以表示年、月、日等**
  - **datetime.time：时间表示类，可以表示小时、分钟、秒、毫秒等**
  - **datetime.datetime：日期和时间表示的类，功能覆盖date和time类**
  - **datetime.timedelta：时间间隔有关的类**
  - **datetime.tzinfo：与时区有关的信息表示类**

datetime库的使用

- 使用**datetime.now()**获得当前日期和时间对象
- 使用**datetime.utcnow()**获得当前日期和时间对应的**UTC**（世界标准时间）时间对象

```
from datetime import datetime
today1 = datetime.now()
today2 = datetime.utcnow()
print(today1)
print(today2)
```

```
2023-10-06 18:55:18.060548
2023-10-06 10:55:18.060548
```

datetime库的使用

- 可以直接使用datetime()构造一个日期和时间对象

  datetime(year, month, day, hour=0, minute=0,second=0, microsecond=0)

- 进一步可以利用这个对象的属性显示时间

```
someday = datetime(2016,9,16,22,33,32,7)
someday
```

```
datetime.datetime(2016, 9, 16, 22, 33, 32, 7)
```

```
print(someday)
```

```
2016-09-16 22:33:32.000007
```

属性	描述
someday.min	固定返回datetime的最小时间对象，datetime(1,1,1,0,0)
someday.max	固定返回datetime的最大时间对象，datetime(9999,12,31,23,59,59,999999)
someday.year	返回someday包含的年份
someday.month	返回someday包含的月份
someday.day	返回someday包含的日期
someday.hour	返回someday包含的小时
someday.minute	返回someday包含的分钟
someday.second	返回someday包含的秒钟
someday.microsecond	返回someday包含的微秒值

datetime库的使用

- **datetime对象有3个常用的时间格式化方法**

属性	描述
someday.isoformat()	采用ISO 8601标准显示时间
someday.isoweekday()	根据日期计算星期后返回1-7,对应星期一到星期日
someday.strftime(format)	根据格式化字符串format进行格式显示的方法

```
someday = datetime(2016, 9, 16, 22, 33, 32, 7)
someday.isoformat()
```

```
'2016-09-16T22:33:32.000007'
```

```
someday.isoweekday()
```

```
5
```

datetime库的使用

- **strftime()方法是时间格式化最有效的方法，几乎可以以任何通用格式输出时间**

```
someday.strftime('%Y-%m-%d %H:%M:%S')
'2016-09-16 22:33:32'
```

格式化字符串	日期/时间	值范围和实例
%Y	年份	0001~9999，例如：1900
%m	月份	01~12，例如：10
%B	月名	January~December，例如：April
%b	月名缩写	Jan~Dec，例如：Apr
%d	日期	01 ~ 31，例如：25
%A	星期	Monday~Sunday，例如：Wednesday
%a	星期缩写	Mon~Sun，例如：Wed
%H	小时（24h制）	00 ~ 23，例如：12
%I	小时（12h制）	01 ~ 12，例如：7
%p	上/下午	AM, PM，例如：PM
%M	分钟	00 ~ 59，例如：26
%S	秒	00 ~ 59，例如：26

datetime库的使用

- **strftime()格式化字符串的数字左侧会自动补零，上述格式也可以与print()的格式化函数一起使用**

```
from datetime import datetime
now = datetime.now()
now.strftime('%Y-%m-%d')
```

```
'2023-10-06'
```

```
now.strftime('%A, %d. %B %Y %I:%M%p')
```

```
'Friday, 06. October 2023 07:59PM'
```

time库的使用

- 通常来讲，时间戳表示的是从1970年1月1日00:00:00开始按秒计算的偏移量。

```
import time
ticks = time.time()
print('当前时间戳为：', ticks)
```

```
当前时间戳为：  1696594084.1062696
```

time库的使用

■ 可以使用time模块的strftime方法来格式化日期。

```
print(time.strftime('%Y-%m-%d %H:%M:%S', time.localtime()))
```

```
2023-10-06 20:09:57
```

格式化符号	含义
%y	两位数的年份表示（00-99）
%Y	四位数的年份表示（000-9999）
%m	月份（01-12）
%d	月内中的一天

格式化符号	含义
%H	24小时小时数（0-23）
%l	12小时制小时数（01-12）
%M	分钟数（00-59）
%S	秒（00-59）

time库的使用

- 时间元组
  - time.altzone
  - time.asctime([tupletime])
  - time.clock()
  - time.ctime([secs])
  - time.gmtime([secs])
  - time.localtime([secs])
  - time.mktime(tupletime)
  - time.sleep(secs)
  - time.strftime(fmt[,tupletime])
  - time.time()
  - time.tzset()
  - time.timezone
  - time.tzname

自行探索

日历函数


```
import calendar
print(calendar.calendar(2023, w=2, l=1, c=6))
```

```
                                  2023

      January                   February                  March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                   1                1  2  3  4  5            1  2  3  4  5
 2  3  4  5  6  7  8       6  7  8  9 10 11 12       6  7  8  9 10 11 12
 9 10 11 12 13 14 15      13 14 15 16 17 18 19      13 14 15 16 17 18 19
16 17 18 19 20 21 22      20 21 22 23 24 25 26      20 21 22 23 24 25 26
23 24 25 26 27 28 29      27 28                     27 28 29 30 31
30 31

      April                     May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2       1  2  3  4  5  6  7                1  2  3  4
 3  4  5  6  7  8  9       8  9 10 11 12 13 14       5  6  7  8  9 10 11
10 11 12 13 14 15 16      15 16 17 18 19 20 21      12 13 14 15 16 17 18
17 18 19 20 21 22 23      22 23 24 25 26 27 28      19 20 21 22 23 24 25
24 25 26 27 28 29 30      29 30 31                  26 27 28 29 30
31

      July                      August                    September
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2          1  2  3  4  5  6                   1  2  3
 3  4  5  6  7  8  9       7  8  9 10 11 12 13       4  5  6  7  8  9 10
10 11 12 13 14 15 16      14 15 16 17 18 19 20      11 12 13 14 15 16 17
17 18 19 20 21 22 23      21 22 23 24 25 26 27      18 19 20 21 22 23 24
24 25 26 27 28 29 30      28 29 30 31               25 26 27 28 29 30
31
```

- calen
- 返回
  离为c
  l是每

一行，间隔距

W+18+2* C。

日历函数

- calendar.firstweekday()
- calendar.isleap(year)
- calendar.leapdays(y1,y2)
- calendar.month(year,month,w=2,l=1)
- calendar.monthcalendar(year,month)
- calendar.monthrange(year,month)
- calendar.prcal(year,w=2,l=1,c=6)
- calendar.setfirstweekday(weekday)
- calendar.timegm(tupletime)
- calendar.prmonth(year,month,w=2,l=1)
- calendar.weekday(year,month,day)

自行探索

random库的使用

- **random.random()**
- **用于生成一个0到1的随机符点数: 0 <= n < 1.0。**

```
import random
# 生成第一个随机数
print("random():", random.random())

# 生成第二个随机数
print("random():", random.random())
```

```
random(): 0.9028749696046191
random(): 0.17446487152725354
```

random库的使用

- **random.uniform(a,b)**
- **返回a, b之间的随机浮点数，a不一定要比b小。**

```
import random
print("random:",random.uniform(50,100))
print("random:",random.uniform(100,50))
```

```
random: 72.99870587308828
random: 90.70466104058092
```

random库的使用

- **random.randint(a,b)**
- **返回a, b之间的整数。传入参数必须是整数，a一定要比b小。**

```
import random
#生成的随机整数n: 12 <= n <= 20
print(random.randint(12,20))
```

```
13
```

random库的使用

- **random.randrang([start], stop[, step])**
- **返回区间内的整数，可以设置step。只能传入整数，random.randrange(10, 100, 2)，结果相当于从[10, 12, 14, 16, … 96, 98]序列中获取一个随机数。**

```
random.randrange(10, 100, 2)
```

```
46
```

random库的使用

- **random.choice(sequence)**
- **从sequence（序列，是有序类型的）中随机获取一个元素，列表、元组、字符串都属于sequence。**

```
random.randrange(10, 100, 2)
# 结果等价于
random.choice(range(10,100,2))
```

```
58
```

random库的使用

- **random.shuffle(x)**
- 用于将列表中的元素打乱顺序，俗称为洗牌。

```
lst = ["Python", "is", "powerful", "simple"]
random.shuffle(lst)
print(lst)
```

```
['is', 'Python', 'simple', 'powerful']
```

random库的使用

- **random.sample(sequence,k)**
- **从指定序列中随机获取k个元素作为一个片段返回，**
- **sample函数不会修改原有序列**

```
list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
slice = random.sample(list, 5)
print(slice)
print(list)
```

```
[1, 10, 2, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```