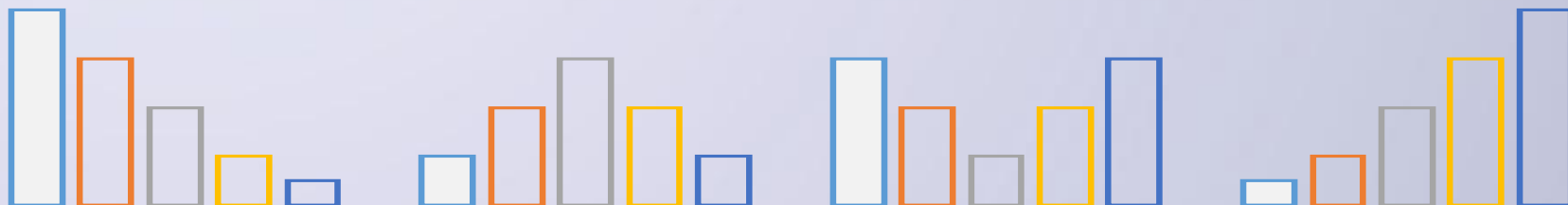
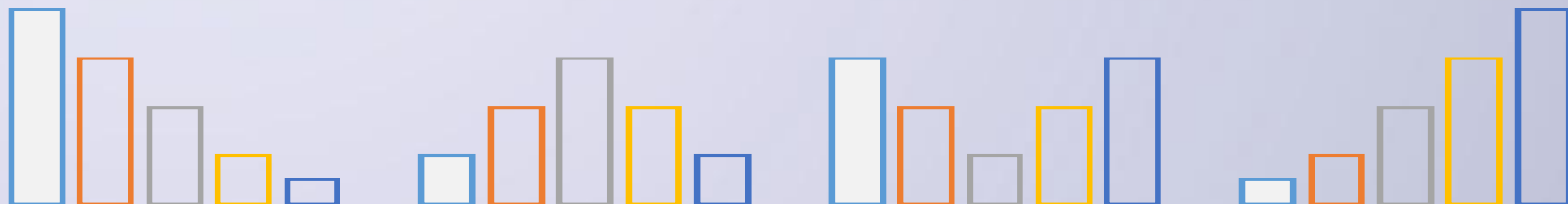


Python语言程序设计

北京理工大学 嵩天



第5节 函数和代码的复用





函数的基本使用



函数的定义

- 函数是一段具有特定功能的、可重用的语句组，用函数名来表示并通过函数名进行完成功能调用。
- 函数也可以看作是一段具有名字的子程序，可以在需要的地方调用执行，不需要在每个执行地方重复编写这些语句。每次使用函数可以提供不同的参数作为输入，以实现对不同数据的处理；函数执行后，还可以反馈相应的处理结果。

■ 函数是一种功能抽象



函数的定义

Python定义一个函数使用def保留字，语法形式如下：

```
def <函数名>(<参数列表>):
```

```
    <函数体>
```

```
    return <返回值列表>
```



函数的定义

微实例5.1：生日歌。

过生日时要为朋友唱生日歌，歌词为：

Happy birthday to you!

Happy birthday to you!

Happy birthday, dear <名字>

Happy birthday to you!


编写程序为Mike和Lily输出生日歌。最简单的实现方法是重复使用print()语句



函数的定义

最简单的实现方法是重复使用print()语句，如下：

```
1 print("Happy birthday to you!")  
2 print("Happy birthday to you!")  
3 print("Happy birthday, dear Mike!")  
4 print("Happy birthday to you!")
```



函数的定义

微实例5.1

m5.1HappyBirthday.py

```
1 def happy():
2     print("Happy birthday to you!")
3 def happyB(name):
4     happy()
5     happy()
6     print("Happy birthday, dear {}".format(name))
7     happy()
8     happyB("Mike")
9     print()
10    happyB("Lily")
```

>>>

```
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Mike!
Happy birthday to you!
```

```
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Lily!
Happy birthday to you!
```




函数调用的过程

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。



函数调用的过程

```
name="Mike"
```

```
happyB("Mike") → def happyB(name):  
print()             happy()  
happyB("Lily")      happy()  
                    print("Happy birthday, dear!".format(name))  
                    happy()
```

微实例5.1中happyB()的被调用过程



函数调用的过程

`name="Mike"`

```
happyB("Mike")  →  def happyB(name):  
print()          happy() → def happy():  
happyB("Lily")   happy()  ↓ print("Happy birthday to you!")  
                  print("Happy birthday, dear!".format(name))  
                  happy()
```

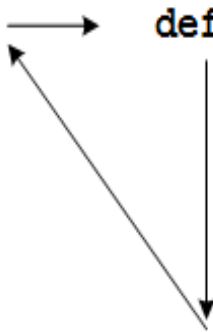


函数调用的过程

```
name="Mike"

happyB("Mike")
print()
happyB("Lily")

def happyB(name):
    happy()
    happy()
    print("Happy birthday, dear!".format(name))
    happy()
```





lambda函数

Python的有33个保留字，其中一个lambda，该保留字用于定义一种特殊的函数——匿名函数，又称lambda函数。

匿名函数并非没有名字，而是将函数名作为函数结果返回，如下：

```
<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>
```



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
>>>type(f)
<class 'function'>
>>>f(10, 12)
22
```



函数的参数传递



可选参数和可变数量参数

在定义函数时，有些参数可以存在默认值

```
>>>def dup(str, times = 2):  
    print(str*times)  
>>>dup("knock~")  
knock~knock~  
>>>dup("knock~", 4)  
knock~knock~knock~knock~
```




可选参数和可变数量参数

在函数定义时，可以设计可变数量参数，通过参数前增加星号（*）实现

```
>>>def vfunc(a, *b):  
    print(type(b))  
    for n in b:  
        a += n  
    return a  
>>>vfunc(1,2,3,4,5)  
<class 'tuple'>  
15
```



参数的位置和名称传递

Python提供了按照形参名称输入实参的方式，调用如下：

```
result = func(x2=4, y2=5, z2=6, x1=1, y1=2, z1=3)
```

由于调用函数时指定了参数名称，所以参数之间的顺序可以任意调整。



变量的返回值

- 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。
- return语句同时可以将0个、1个或多个函数运算完的结果返回给函数被调用处的变量，例如。

```
>>>def func(a, b):  
    return a*b  
>>>s = func("knock~", 2)  
>>>print(s)  
knock~knock~
```



变量的返回值

函数可以没有return，此时函数并不返回值，如微实例5.1的happy()函数。函数也可以用return返回多个值，多个值以元组类型保存，例如。

```
>>>def func(a, b):  
    return b,a  
>>>s = func("knock~", 2)  
>>>print(s, type(s))  
(2, 'knock~') <class 'tuple'>
```



函数对变量的作用

一个程序中的变量包括两类：全局变量和局部变量。

- 全局变量指在函数之外定义的变量，一般没有缩进，在程序执行全过程有效。
- 局部变量指在函数内部使用的变量，仅在函数内部有效，当函数退出时变量将不存在。



变量的返回值

```
>>>n = 1      #n是全局变量
>>>def func(a, b):
        c = a * b      #c是局部变量, a和b作为函数参数也是局部变量
        return c
>>>s = func("knock~", 2)
>>>print(c)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print(c)
NameError: name 'c' is not defined
```


这个例子说明，当函数执行完退出后，其内部变量将被释放。如果函数内部使用了全局变量呢？



变量的返回值

```
>>>n = 1      #n是全局变量
>>>def func(a, b):
        n = b      #这个n是在函数内存中新生成的局部变量，不是全局变量
        return a*b
>>>s = func("knock~", 2)
>>>print(s, n)    #测试一下n值是否改变
knock~knock~ 1
```

- 函数func()内部使用了变量n，并且将变量参数b赋值给变量n，为何全局变量n值没有改变？



变量的返回值

如果希望让func()函数将n当作全局变量，需要在变量n使用前显式声明该变量为全局变量，代码如下。


```
>>>n = 1      #n是全局变量
>>>def func(a, b):
    global n
    n = b      #将局部变量b赋值给全局变量n
    return a*b
>>>s = func("knock~", 2)
>>>print(s, n)  #测试一下n值是否改变
knock~knock~ 2
```




变量的返回值

如果此时的全局变量不是整数n，而是列表类型ls，会怎么样呢？理解如下代码。


```
>>>ls = []      #ls是全局列表变量
>>>def func(a, b):
    ls.append(b)    #将局部变量b增加到全局列表变量ls中
    return a*b
>>>s = func("knock~", 2)
>>>print(s, ls)   #测试一下ls值是否改变
knock~knock~ [2]
```



变量的返回值

如果func()函数内部存在一个真实创建过且名称为ls的列表，则func()将操作该列表而不会修改全局变量，例子如下。


```
>>>ls = []      #ls是全局列表变量
>>>def func(a, b):
    ls = []      #创建了名称为ls的局部列表变量列
    ls.append(b)  #将局部变量b增加到全局列表变量ls中
    return a*b
>>>s = func("knock~", 3)
>>>print(s, ls)  #测试一下ls值是否改变
knock~knock~ []
```



变量的返回值

Python函数对变量的作用遵守如下原则：

- 简单数据类型变量无论是否与全局变量重名，仅在函数内部创建和使用，函数退出后变量被释放；
- 简单数据类型变量在用global保留字声明后，作为全局变量；
- 对于组合数据类型的全局变量，如果在函数内部没有被真实创建的同名变量，则函数内部可直接使用并修改全局变量的值；
- 如果函数内部真实创建了组合数据类型变量，无论是否有同名全局变量，函数仅对局部变量进行操作。



datetime库的使用



datetime库概述

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库概述

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库解析

使用datetime.now()获得当前日期和时间对象，使用方法如下：

datetime.now()

作用：返回一个datetime类型，表示当前的日期和时间，精确到微秒。

```
>>> from datetime import datetime
>>> today = datetime.now()
>>> today
datetime.datetime(2016, 9, 20, 10, 29, 43, 928549)
```



datetime库解析

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，使用方法如下：

datetime.utcnow()

作用：返回datetime类型，表示当前日期和时间的UTC表示，精确到微秒。

```
>>> today = datetime.utcnow()  
>>> today  
datetime.datetime(2016, 9, 20, 2, 35, 1, 427954)
```




datetime库解析

`datetime.now()` 和 `datetime.utcnow()` 都返回一个 `datetime` 类型的对象，也可以直接使用 `datetime()` 构造一个日期和时间对象，使用方法如下：

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

作用：返回一个 `datetime` 类型，表示指定的日期和时间，可以精确到微秒。



datetime库解析

调用datetime()函数直接创建一个datetime对象，表示2016年9月16日22:33，32秒7微秒，执行结果如下：

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

程序已经有了一个datetime对象，进一步可以利用这个对象的属性显示时间，为了区别datetime库名，采用上例中的someday代替生成的datetime对象



datetime库解析

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库解析

datetime对象有3个常用的时间格式化方法，如表所示

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

isoformat()和isoweekday()方法的使用如下：

```
>>> someday = datetime(2016,9,16,22,33,32,7)
>>> someday.isoformat()
'2016-09-16T22:33:32.000007'
>>> someday.isoweekday()
5
```



datetime库解析

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'
```



datetime库解析


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库解析

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

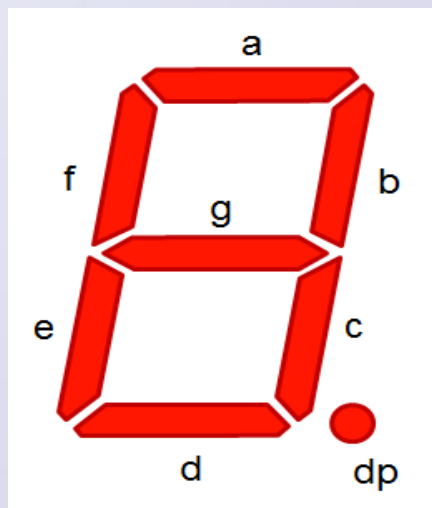
```
>>>from datetime import datetime
>>>now = datetime.now()
>>>now.strftime("%Y-%m-%d")
'2016-09-20'
>>>now.strftime("%A, %d. %B %Y %I:%M%p")
'Tuesday, 20. September 2016 01:53PM'
>>>print("今天是{0:%Y}年{0:%m}月{0:%d}日".format(now))
今天是2016年09月20日
```



七段数码管绘制

七段数码管绘制

七段数码管（seven-segment indicator）由7段数码管拼接而成，每段有亮或不亮两种情况，改进型的七段数码管还包括一个小数点位置，如图所示。



七段数码管绘制


七段数码管能形成 $2^7=128$ 种不同状态，其中部分状态能够显示易于人们理解的数字或字母含义，因此被广泛使用。图5.5给出了十六进制中16个字符的七段数码管表示。





七段数码管绘制

每个0到9的数字都有相同的七段数码管样式，因此，可以通过设计函数复用数字的绘制过程。进一步，每个七段数码管包括7个数码管样式，除了数码管位置不同外，绘制风格一致，也可以通过函数复用单个数码段的绘制过程。




七段数码管绘制

实例代码7.1

e7.1[DrawSevenSegDisplay.py](#)

```
1  #e7.1DrawSevenSegDisplay.py
2  import turtle, datetime
3  def drawLine(draw):    #绘制单段数码管
4      turtle.pendown() if draw else turtle.penup()
5      turtle.fd(40)
6      turtle.right(90)
7  def drawDigit(d): #根据数字绘制七段数码管
8      drawLine(True) if d in [2,3,4,5,6,8,9] else drawLine(False)
9      drawLine(True) if d in [0,1,3,4,5,6,7,8,9] else drawLine(False)
10     drawLine(True) if d in [0,2,3,5,6,8,9] else drawLine(False)
11     drawLine(True) if d in [0,2,6,8] else drawLine(False)
12     turtle.left(90)
13     drawLine(True) if d in [0,4,5,6,8,9] else drawLine(False)
14     drawLine(True) if d in [0,2,3,5,6,7,8,9] else drawLine(False)
15     drawLine(True) if d in [0,1,2,3,4,7,8,9] else drawLine(False)
```



七段数码管绘制

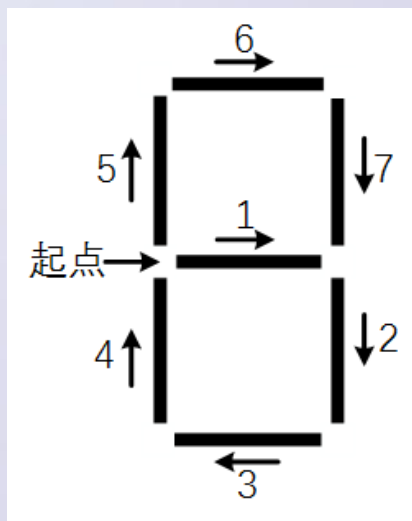
实例代码7.1

e7.1[DrawSevenSegDisplay.py](#)

```
16  turtle.left(180)
17      turtle.penup()
18      turtle.fd(20)
19  def drawDate(date):    #获得要输出的数字
20      for i in date:
21          drawDigit(eval(i))    #注意：通过eval()函数将数字变为整数
22  def main():
23      turtle.setup(800, 350, 200, 200)
24      turtle.penup()
25      turtle.fd(-300)
26      turtle.pensize(5)
27      drawDate(datetime.datetime.now().strftime('%Y%m%d'))
28      turtle.hideturtle()
29  main()
```

七段数码管绘制

实例代码定义了drawDigit()函数，该函数根据输入的数字d绘制七段数码管，结合七段数码管结构，每个数码管的绘制采用图所示顺序。





七段数码管绘制

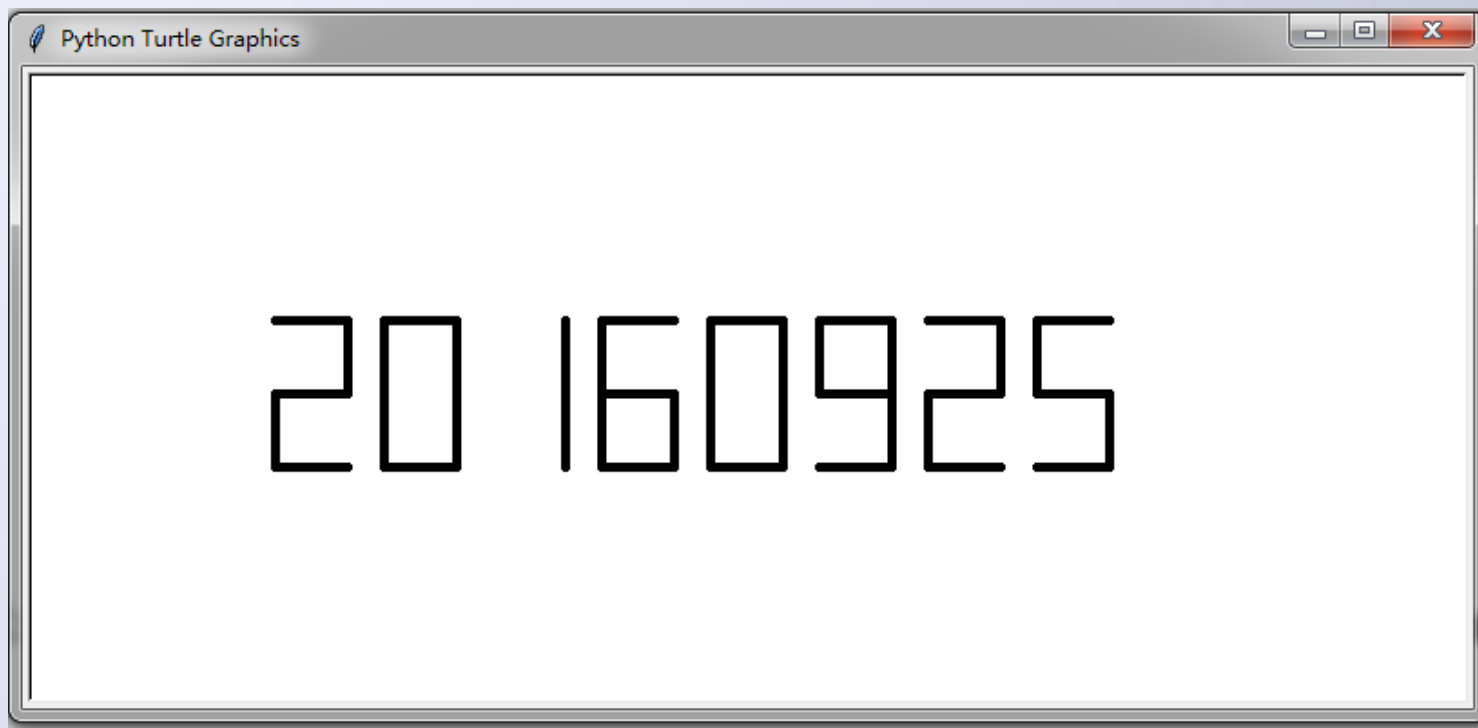
绘制起点在数码管中部左侧，无论每段数码管是否被绘制出来，turtle画笔都按顺序“画完”所有7个数码管。对于给定数字d，哪个数码段被绘制出来采用if...else...语句判断。

8

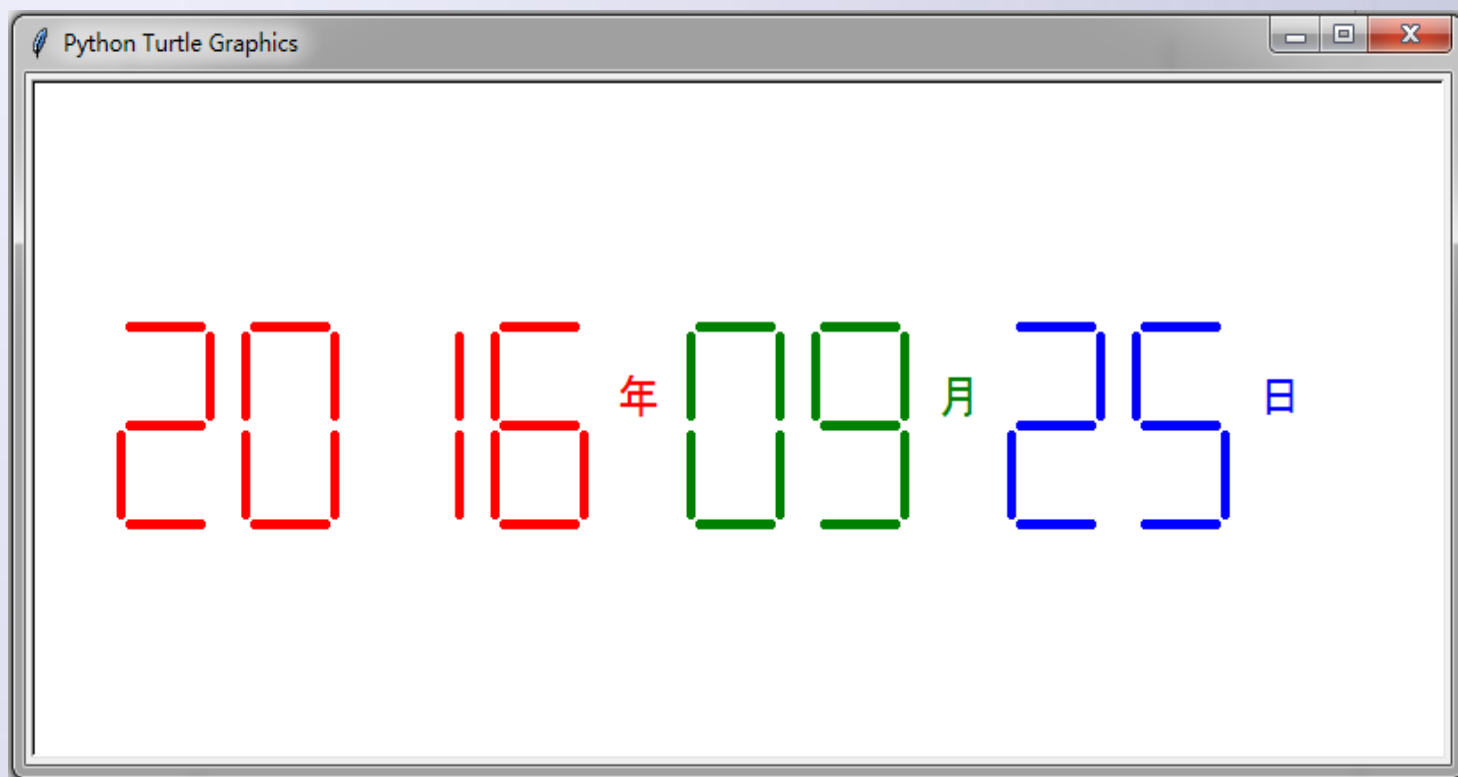
```
drawLine(True) if d in [2,3,4,5,6,8,9] else drawLine(False)
```



七段数码管绘制



七段数码管绘制



```
1  #e7.2DrawSevenSegDisplay.py
2  import turtle, datetime
3  def drawGap(): #绘制数码管间隔
4      turtle.penup()
5      turtle.fd(5)
6  def drawLine(draw): #绘制单段数码管
7      drawGap()
8      turtle.pendown() if draw else turtle.penup()
9      turtle.fd(40)
10     drawGap()
11     turtle.right(90)
12 def drawDigit(d): #根据数字绘制七段数码管
13     drawLine(True) if d in [2,3,4,5,6,8,9] else drawLine(False)
14     drawLine(True) if d in [0,1,3,4,5,6,7,8,9] else drawLine(False)
15     drawLine(True) if d in [0,2,3,5,6,8,9] else drawLine(False)
16     drawLine(True) if d in [0,2,6,8] else drawLine(False)
17     turtle.left(90)
18     drawLine(True) if d in [0,4,5,6,8,9] else drawLine(False)
19     drawLine(True) if d in [0,2,3,5,6,7,8,9] else drawLine(False)
20     drawLine(True) if d in [0,1,2,3,4,7,8,9] else drawLine(False)
21     turtle.left(180)
22     turtle.penup()
23     turtle.fd(20)
24
```

```
25 def drawDate(date):
26     turtle.pencolor("red")
27     for i in date:
28         if i == '-':
29             turtle.write('年', font=("Arial", 18, "normal"))
30             turtle.pencolor("green")
31             turtle.fd(40)
32         elif i == '=':
33             turtle.write('月', font=("Arial", 18, "normal"))
34             turtle.pencolor("blue")
35             turtle.fd(40)
36         elif i == '+':
37             turtle.write('日', font=("Arial", 18, "normal"))
38         else:
39             drawDigit(eval(i))
40 def main():
41     turtle.setup(800, 350, 200, 200)
42     turtle.penup()
43     turtle.fd(-350)
44     turtle.pensize(5)
45     drawDate(datetime.datetime.now().strftime('%Y-%m=%d+'))
46     turtle.hideturtle()
main()
```



代码的复用和模块化设计



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

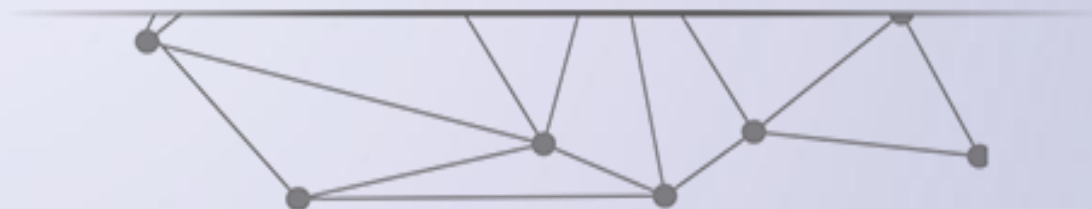
模块化设计一般有两个基本要求：

- 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。

使用函数只是模块化设计的必要非充分条件，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。



函数的递归





递归的定义

函数作为一种代码封装，可以被其他程序调用，当然，也可以被函数内部代码调用。这种函数定义中调用函数自身的方式称为递归。就像一个人站在装满镜子的房间中，看到的影像就是递归的结果。递归在数学和计算机应用上非常强大，能够非常简洁的解决重要问题。



递归的定义

数学上有个经典的递归例子叫阶乘，阶乘通常定义为：

$$n! = n(n-1)(n-2)\dots(1)$$

这个关系给出了另一种方式表达阶乘的方式：

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



递归的定义

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。



递归的使用方法

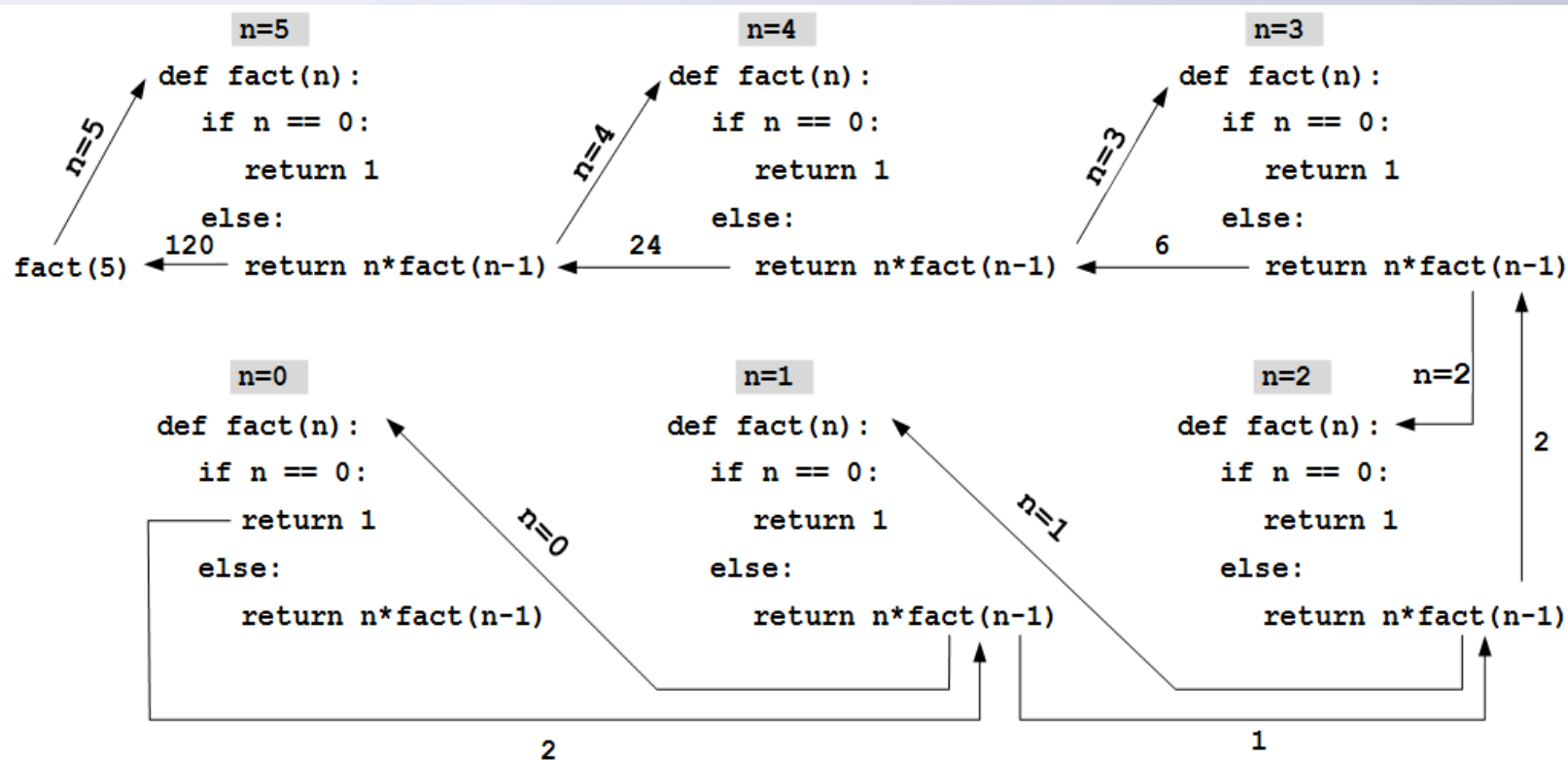
微实例5.21：阶乘的计算。


根据用户输入的整数 n ，计算并输出 n 的阶乘值。

微实例5.21 m5.1CalFactorial.py

```
1 def fact(n):
2     if n == 0:
3         return 1
4     else:
5         return n * fact(n-1)
6 num = eval(input("请输入一个整数: "))
7 print(fact(abs(int(num))))
```

递归的使用方法





递归的使用方法

微实例5.32：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。

解决这个问题基本思想是把字符串看作一个递归对象。

```
1 def reverse(s):  
2     return reverse(s[1:]) + s[0]
```



递归的使用方法

观察这个函数的工作过程。s[0]是首字符，s[1:]是剩余字符串，将它们反向连接，可以得到反转字符串。执行这个程序，结果如下

```
>>>def reverse(s):  
    return reverse(s[1:]) + s[0]  
>>>reverse("ABC")  
...  
RecursionError: maximum recursion depth exceeded
```




科赫曲线绘制



科赫曲线绘制

自然界有很多图形很规则，符合一定的数学规律，例如，蜜蜂蜂窝是天然的等边六角形等。科赫(Koch)曲线在众多经典数学曲线中非常著名，由瑞典数学家冯·科赫(H·V·Koch)于1904年提出，由于其形状类似雪花，也被称为雪花曲线。



科赫曲线绘制

科赫曲线的基本概念和绘制方法如下：

正整数 n 代表科赫曲线的阶数，表示生成科赫曲线过程的操作次数。科赫曲线初始化阶数为0，表示一个长度为 L 的直线。对于直线 L ，将其等分为三段，中间一段用边长为 $L/3$ 的等边三角形的两个边替代，得到1阶科赫曲线，它包含四条线段。进一步对每条线段重复同样的操作后得到2阶科赫曲线。继续重复同样的操作 n 次可以得到 n 阶科赫曲线。



科赫曲线绘制

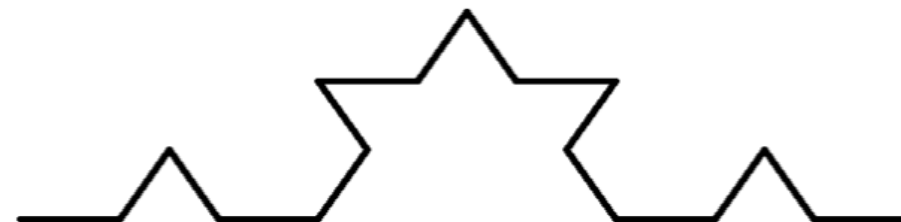
0阶科赫曲线



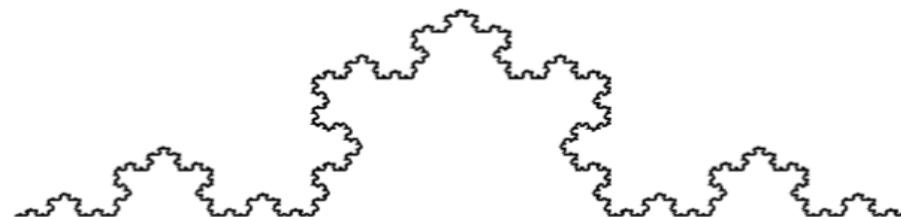
1阶科赫曲线




2阶科赫曲线



5阶科赫曲线






科赫曲线绘制

科赫曲线属于分形几何分支，它的绘制过程体现了递归思想，绘制过程代码。

实例代码8.1

e8.1DrawKoch.py

```
1  #e8.1DrawKoch.py
2  import turtle
3  def koch(size, n):
4      if n == 0:
5          turtle.fd(size)
6      else:
7          for angle in [0, 60, -120, 60]:
8              turtle.left(angle)
9              koch(size/3, n-1)
10 def main():
11     turtle.setup(800,400)
12     turtle.speed(0)    #控制绘制速度
13     turtle.penup()
14     turtle.goto(-300, -50)
15     turtle.pendown()
16     turtle.pensize(2)
17     koch(600,3)        # 0阶科赫曲线长度，阶数
18     turtle.hideturtle()
19 main()
```



科赫曲线绘制

科赫曲线的雪花效果

实例代码8.2

e8.2DrawKoch.py

```
1  #e8.2DrawKoch.py
2  import turtle
3  def koch(size, n):
4      if n == 0:
5          turtle.fd(size)
6      else:
7          for angle in [0, 60, -120, 60]:
8              turtle.left(angle)
9              koch(size/3, n-1)
10 def main():
11     turtle.setup(600, 600)
12     turtle.speed(0)
13     turtle.penup()
14     turtle.goto(-200, 100)
15     turtle.pendown()
16     turtle.pensize(2)
17     level = 5
18     koch(400, level)
19     turtle.right(120)
20     koch(400, level)
21     turtle.right(120)
22     koch(400, level)
23     turtle.hideturtle()
24 main()
```



Python内置函数



Python内置函数

Python解释器提供了68个内置函数，其中，前36个已经将结果，需要掌握。

<code>abs()</code>	<code>id()</code>	<code>round()</code>	<code>compile()</code>	<code>locals()</code>
<code>all()</code>	<code>input()</code>	<code>set()</code>	<code>dir()</code>	<code>map()</code>
<code>any()</code>	<code>int()</code>	<code>sorted()</code>	<code>exec()</code>	<code>memoryview()</code>
<code>ascii()</code>	<code>len()</code>	<code>str()</code>	<code>enumerate()</code>	<code>next()</code>
<code>bin()</code>	<code>list()</code>	<code>tuple()</code>	<code>filter()</code>	<code>object()</code>
<code>bool()</code>	<code>max()</code>	<code>type()</code>	<code>format()</code>	<code>property()</code>
<code>chr()</code>	<code>min()</code>	<code>zip()</code>	<code>frozenset()</code>	<code>repr()</code>
<code>complex()</code>	<code>oct()</code>		<code>getattr()</code>	<code>setattr()</code>
<code>dict()</code>	<code>open()</code>		<code>globals()</code>	<code>slice()</code>
<code>divmod()</code>	<code>ord()</code>	<code>bytes()</code>	<code>hasattr()</code>	<code>staticmethod()</code>
<code>eval()</code>	<code>pow()</code>	<code>delattr()</code>	<code>help()</code>	<code>sum()</code>
<code>float()</code>	<code>print()</code>	<code>bytearray()</code>	<code>isinstance()</code>	<code>super()</code>
<code>hash()</code>	<code>range()</code>	<code>callable()</code>	<code>issubclass()</code>	<code>vars()</code>
<code>hex()</code>	<code>reversed()</code>	<code>classmethod()</code>	<code>iter()</code>	<code>import()</code>