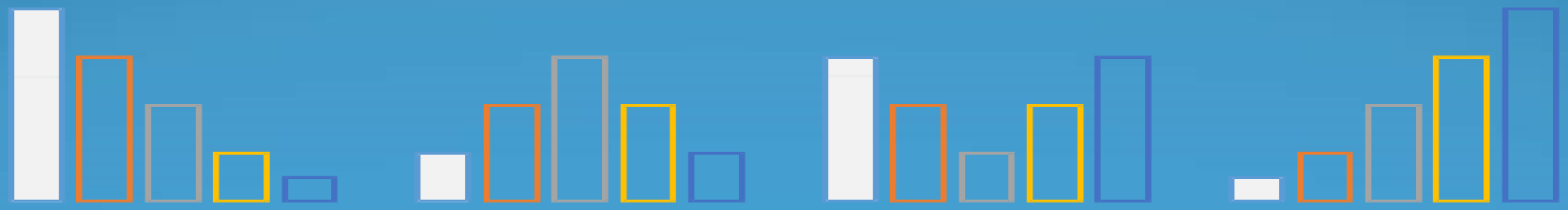
函数和代码的复用



函数的定义

微实例：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>):
    '''注释'''
    <函数体>
    return <返回值列表>
```

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

```
>>>
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；

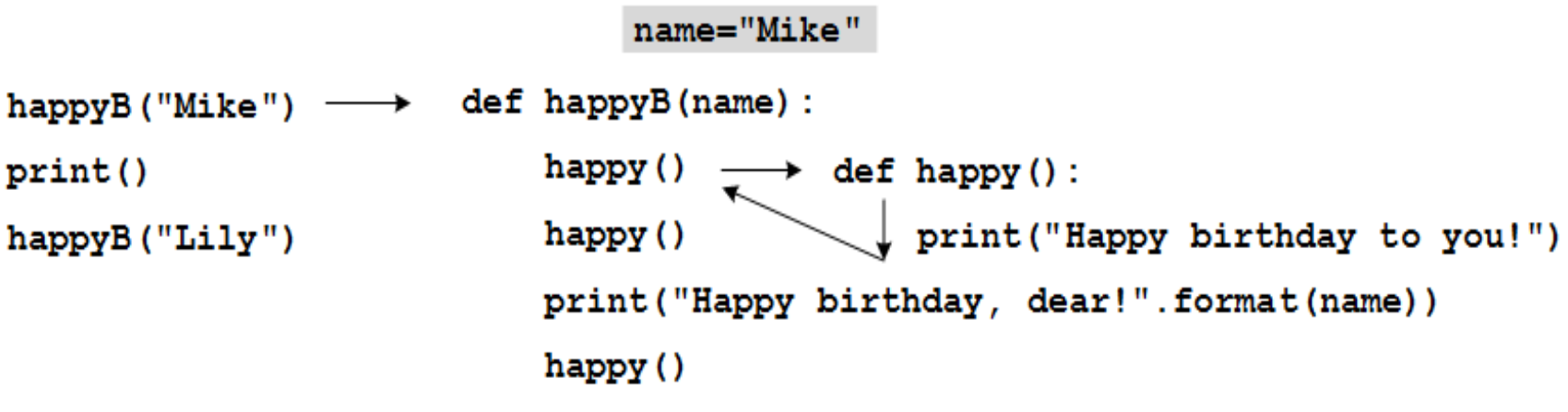
（4）函数调用结束给出返回值，程序回到调用前的暂停处继续执行。

函数调用的过程

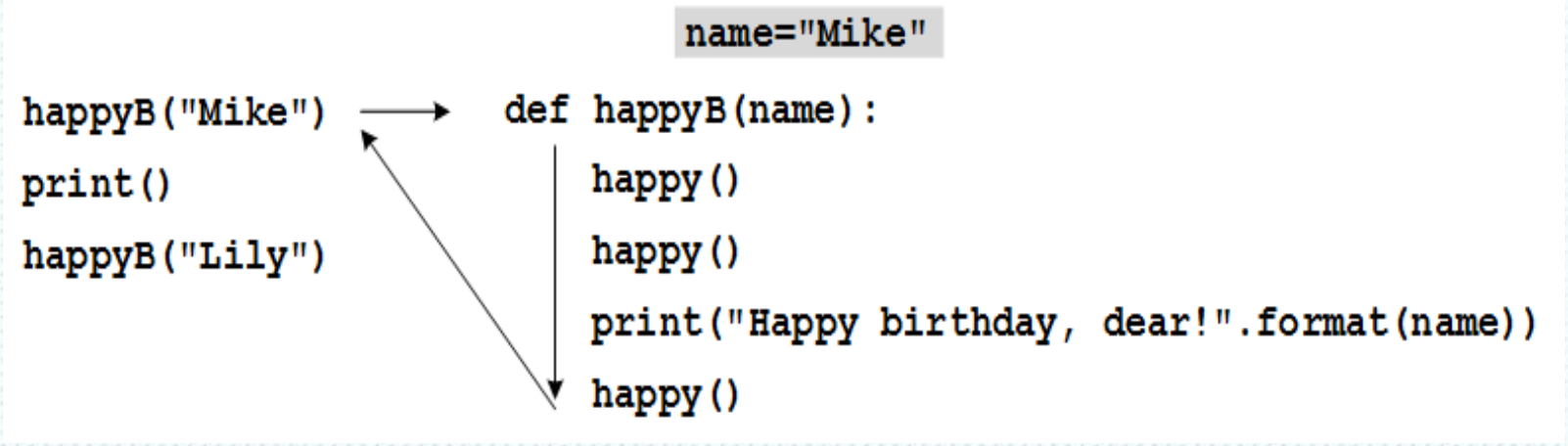
```
                                    name="Mike"
happyB("Mike")  ⟶   def happyB(name):
print()                 happy()
happyB("Lily")          happy()
                        print("Happy birthday, dear!".format(name))
                        happy()
```

微实例5.1中happyB()的被调用过程

函数调用的过程


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

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，其中一个是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，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。

默认值参数必须出现在函数参数列表的**最右端**，且任何一个默认值参数右边不能有非默认值参数。

```
>>>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

```
>>>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：

•对于**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。

案例精选

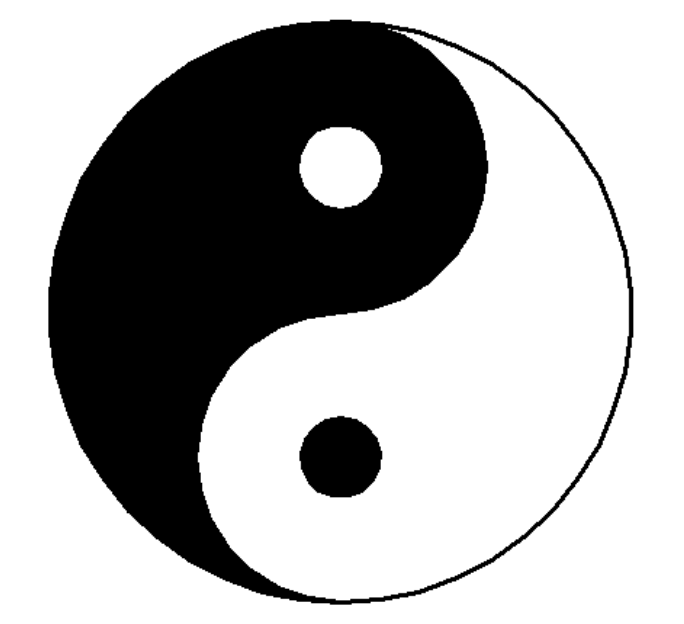
- 例：编写函数模拟猜数游戏。系统随机产生一个数，玩家最多可以猜5次，系统会根据玩家的猜测进行提示，玩家则可以根据系统的提示对下一次的猜测进行适当调整。

案例精选

```
from random import randint

def guess(maxValue=100, maxTimes=5):
    value = randint(1,maxValue)     #随机生成一个整数
    for i in range(maxTimes):
        prompt = 'Start to GUESS:' if i==0 else 'Guess again:'
        #使用异常处理结构，防止输入不是数字的情况
        try:
            x = int(input(prompt))
        except:
            print('Must input an integer between 1 and ', maxValue)
        else:
            if x == value:                  #猜对了
                print('Congratulations!')
                break
            elif x > value:
                print('Too big')
            else:
                print('Too little')
    else:                                   #次数用完还没猜对，游戏结束，提示正确答案
        print('Game over. FAIL.')
        print('The value is ', value)
```

小练习: 使用turtle绘制阴阳图形



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：

- 类datetime.date：可以表示年、月、日等
- 类datetime.time：可以表示小时、分钟、秒、毫秒等
- 类datetime.datetime：日期和时间表示的类，其很多功能覆盖date和time类
- 类datetime.timedelta：时间间隔有关的类
- 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：

from datetime import datetime

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
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

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

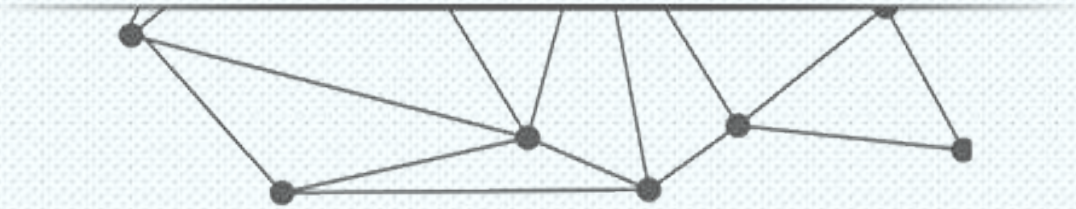
格式化字符串	日期/时间	值范围和实例
%Y	年份	0001~9999，例如：1900
%m	月份	01~12，例如：10
%B	月名	January~December，例如：April
%b	月名缩写	Jan~Dec，例如：Apr
%d	日期	01～31，例如：25
%A	星期	Monday~Sunday，例如：Wednesday
%a	星期缩写	Mon~Sun，例如：Wed
%H	小时（24h制）	00～23，例如：12
%I	小时（12h制）	01～12，例如：7
%p	上/下午	AM, PM，例如：PM
%M	分钟	00～59，例如：26
%S	秒	00～59，例如：26

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

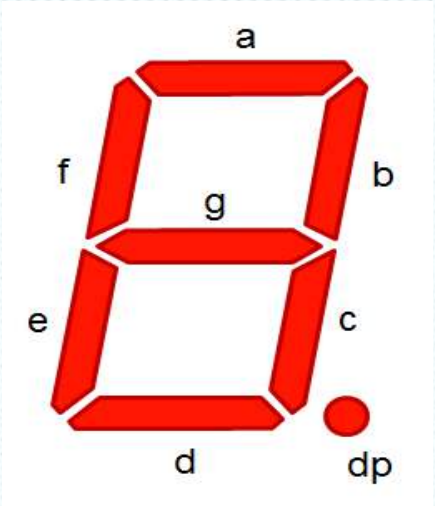
```
>>>from datetime import datetime
>>>now = datetime.now()
>>>now.strftime("%Y-%m-%d")
'2016-09-20'
>>>now.strftime("%A, %d. %B %Y %I:%M%p")
'Tuesday, 20. September 2016 01:53PM'
>>>print("今天是{0:%Y}年{0:%m}月{0:%d}日".format(now))
今天是2016年09月20日
```

七段数码管绘制



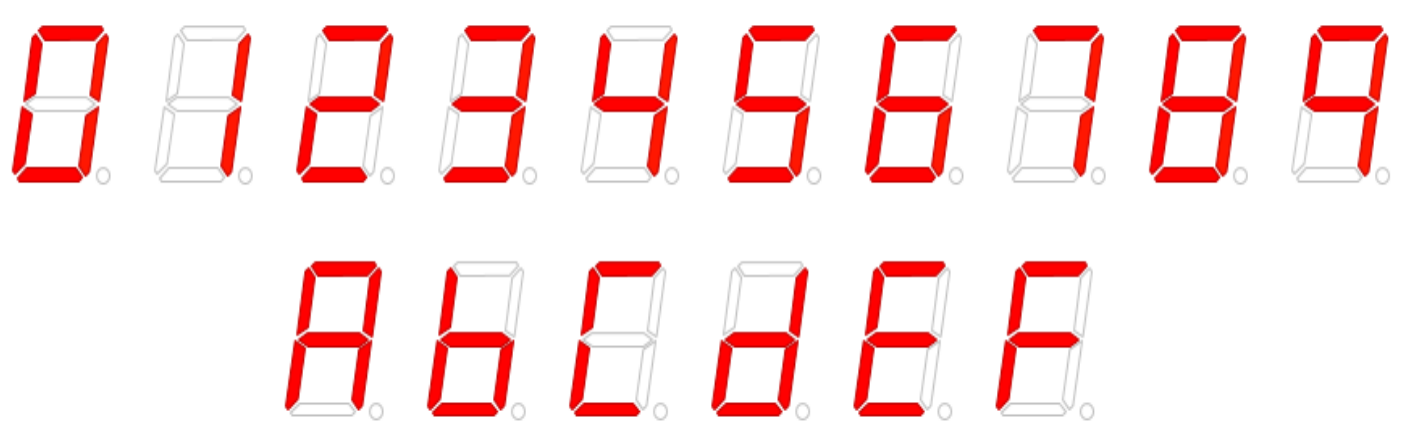
七段数码管绘制

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。