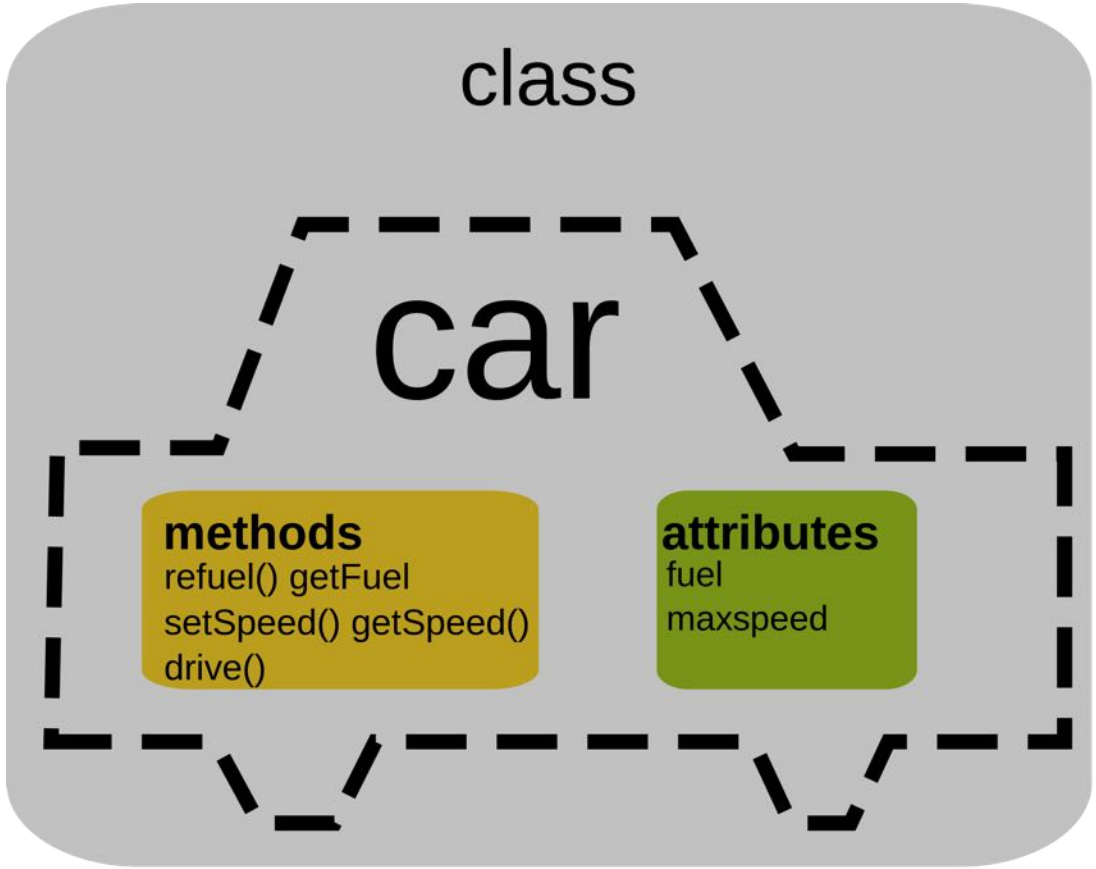
Python语言基础与应用05

北京大学 陈斌

2019.07.08

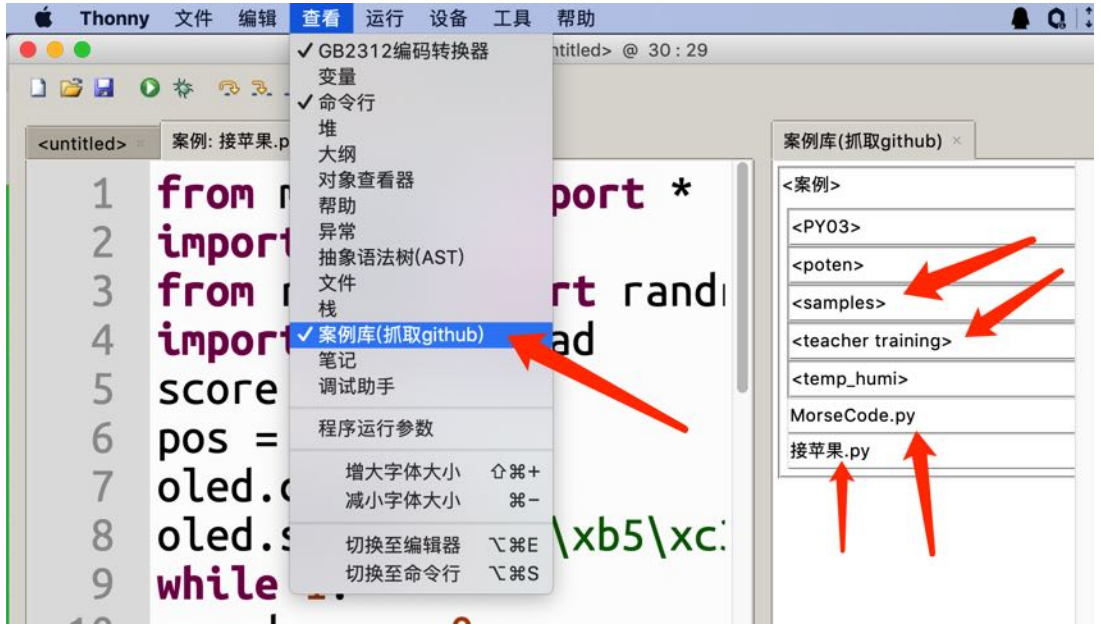
目录

- micro:bit案例分析
- 控制流程结构对比
- 函数的参数：定义与调用
- 面向对象
- 特殊方法



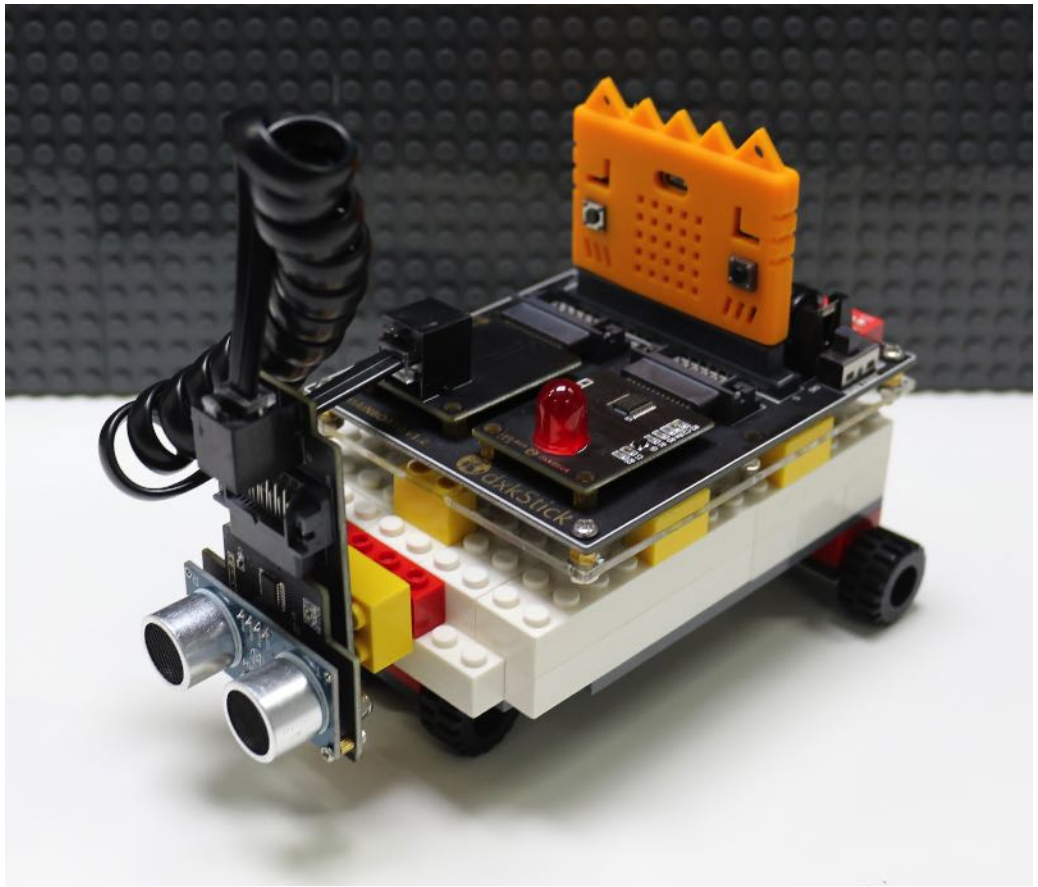
micro:bit案例分析

- 从github可以看到许多案例
- Thonny有一个案例库侧栏
  - 查看->案例库
  - 直接从github加载案例
- 加载后直接点击写入代码即可
- 按reset运行



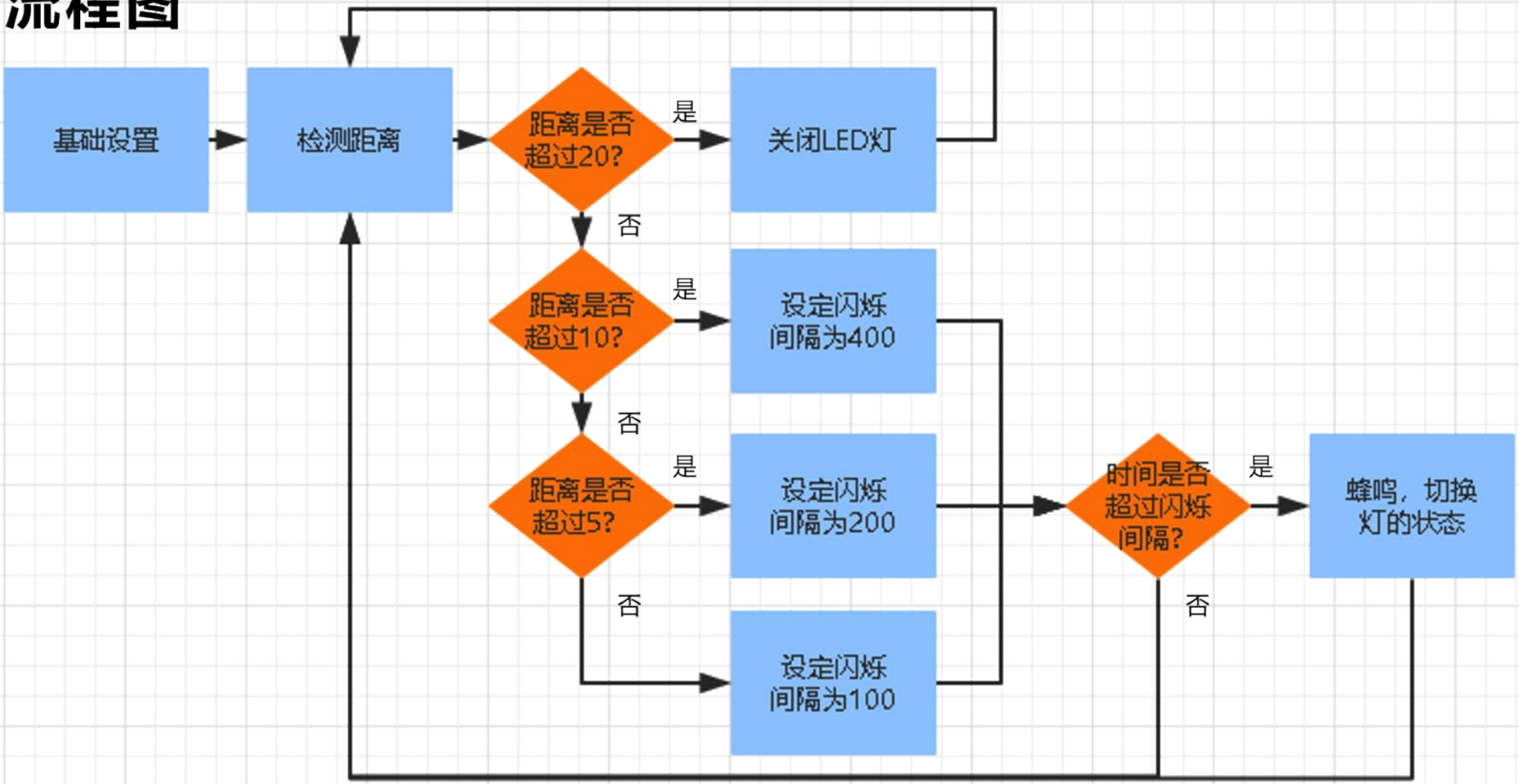
倒车雷达

- 模拟倒车雷达功能
- 通过超声测距模块测量与障碍物的距离
- 并根据距离远近发出警告：
  - 不同间隔的蜂鸣声
  - 不同间隔的LED灯闪烁提示。
- 20厘米以外不报警
- 20厘米以内距离越近，声音越急促，闪烁越频繁



倒车雷达

流程图


基础设置
检测距离
距离是否超过20?
是
关闭LED灯
否
距离是否超过10?
是
设定闪烁间隔为400
否
距离是否超过5?
是
设定闪烁间隔为200
否
设定闪烁间隔为100
时间是否超过闪烁间隔?
是
蜂鸣，切换灯的状态
否

倒车雷达代码分析：基础设置

```
# 倒车雷达

# 简介:
# 模拟倒车雷达功能，通过超声测距模块测量与障碍物的距离
# 并根据距离远近发出不同频率的蜂鸣声与LED灯闪烁提示。

# 硬件模块:
# micro:bit×1；主板×1；延长插槽×1
# 模块×2：超声测距、LED灯泡

from microbit import *
import music
import ultrasonic, led   # 模块控制库

# 初始化LED灯闪烁系统
light_on = 0
ltimer = 0
led.off()

# 初始化测距记录系统
dist = ultrasonic.value()
update_timer = 0
```

- 通过import加载必要的模块
  - microbit, music, ultrasonic, led
- 警报的循环体控制
  - ltimer 控制声响/闪烁的间隔
  - 记录LED灯的点亮状态
- 测距的循环体控制
  - update_timer 控制测距时间间隔

倒车雷达代码分析：主循环

- 测距离的循环体控制
- 每50次循环检测一次距离，记录在dist变量中

```
24      while True:
25          # 每隔一段时间读取一次距离示数
26          update_timer += 1
27          if update_timer > 50:
28              update_timer -= 50
29              dd = ultrasonic.value()
30              if dd != None:   # 仅在读取示数成功时更新距离记录
31                  dist = dd
32                  update_timer -= 50
```

倒车雷达代码分析：主循环

- 报警的循环体控制
- 大于20cm不响应
- 小于20cm确定间隔
  - check
- ltimer与check判断
  - 实现了不同报警间隔

```
34      # 在获取到距离读数后进入响应系统
35      if dist != None:
36          if dist > 20:   # 在该距离以上不响应
37              if light_on:
38                  light_on = 0
39                  led.off()
40          else:
41              # 由距离确定闪烁频率
42              check = 400 if dist > 10 else 200 if dist > 5 else 100
43
44              # 更新闪烁计时
45              ltimer += 1
46              if ltimer > check:
47                  ltimer -= check
48
49                  # 蜂鸣
50                  music.pitch(1000, 50, wait=0)
51
52                  # 切换LED灯状态
53                  if light_on:
54                      light_on = 0
55                      led.off()
56                  else:
57                      light_on = 1
58                      led.on()
```

接苹果代码分析

- 初始化导入模块
  - microbit, music, random, oled, joypad
  - score得分，pos用户的位置

```
1  from microbit import *
2  import music
3  from random import randrange
4  import oled,joypad
5  score = 0
6  pos = 2
7  oled.clear()
8  oled.show(0,0,b'\xb5\xc3\xb7\xd6\xa3\xba0')# 得分: 0
```

接苹果代码分析

- 苹果dropx/y
- 随机的dropx
- 获取游戏按键
  - 3-左
  - 4-右
- 绘制苹果和用户
  - set_pixel
- 判断接到
- 未接到结束

```
while 1:
    dropy = 0
    dropx = randrange(5)
    while dropy<=4:
        keys=joypad.keys()
        if keys[3]:
            pos=max(0,pos-1)
        elif keys[4]:
            pos=min(4,pos+1)
        display.clear()
        display.set_pixel(dropx,dropy,9)
        display.set_pixel(pos,4,9)
        dropy+=1
        sleep(max(200,500-5*score))
    if dropx==pos:
        score+=1
        music.play('C7:1',wait=False)
        oled.show(0,0,b'\xb5\xc3\xb7\xd6\xa3\xba%d'%score)# 得分：{score}
    else:
        break
oled.show(2,0,b'\xd3\xce\xcf\xb7\xbd\xe1\xca\xf8')# 游戏结束
music.play(music.POWER_DOWN)
```

声控小灯

- 初始化导入模块
  - microbit
  - mic，led
- 设置固定变量
  - 声音强度
  - 亮灯的时间长度
- 设置计时变量
  - 点亮时长
  - 循环计数tloop
  - 音量过程变量

```
1   # 声控小灯
2
3   # 简介:
4   # 麦克风模块会持续读取声音，当读数高于一定值时会使小灯点亮一段时间。
5
6   # 硬件模块:
7   # micro:bit×1; 主板×1
8   # 模块×2: 麦克风、LED灯泡
9
10  from microbit import *
11  import mic, led   # 模块控制库
12
13  thr = 60   # 声音强度阈值
14  dur = 50   # 亮灯时间
15
16  # 初始化LED灯闪烁系统
17  led_on = 0
18  led.off()
19
20  # 初始化计时变量
21  t_led_delay = 0
22  tloop = 0
23  volumn = 0
```

声控小灯

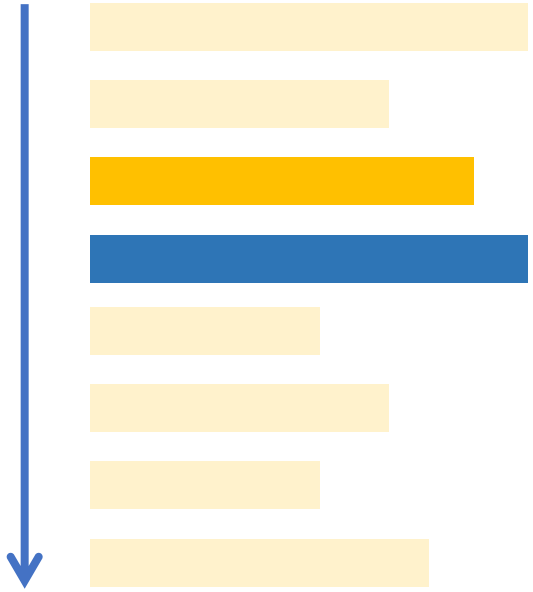
- 每10次循环记录一次音量
- 每30次循环检测是否点亮
  - 达到条件则设置点亮时长
- 可以点亮那么开LED
- 时长变量复位则关闭LED

```
25  while True:
26      tloop += 1
27
28      # 每10帧取一次音量，保留最大值
29      if tloop % 10:
30          tmp = mic.value()
31          if tmp != None:
32              volumn = max(volumn, tmp)
33
34      # 每30帧更新一次，在声音足够大时刷新LED灯点亮时间
35      if tloop > 30:
36          tloop -= 30
37          if volumn > thr:
38              t_led_delay = dur
39          volumn = 0
40
41      # 更新LED灯显示状态
42      t_led_delay -= 1
43      if t_led_delay > 0:
44          if not led_on:
45              led_on = 1
46              led.on()
47      else:
48          if led_on:
49              led_on = 0
50              led.off()
```

micro:bit-micropython编程技巧

- 不要用复杂对象，很占内存
- 定期进行垃圾回收
  - import gc
  - gc.collect()
- 调试传感器，熟悉其最短访问时间
- 为硬件增加sleep恢复时间
  - 要有，也不要太长（10ms）
- 硬件模块的延迟问题
- 在一个循环中轮询和处理事件
  - 对于多个不同间隔的事件采用不同的计数器来分别计时
  - 动态调整间隔
  - 安排好轮询不同硬件的次序
  - 背景音乐和动画的处理
    - display（wait）
    - music（loop，wait）
- 注意USB串口写入会受内置程序运行的影响
  - 按reset然后点写入，多试几次

控制流程结构对比



顺序结构

条件分支结构

循环结构

控制流语句：条件if

- 条件语句

```
if <逻辑条件>:
    <语句块>
elif <逻辑条件>: #可以多个elif
    <语句块>
else: #仅1个
    <语句块>
```

- 各种类型中某些值会自动被转换为False，其它值则是True：
  - None, 0, 0.0, '',
  - [], (), {}, set()

```
>>> a = 12
>>> if a > 10:
        print ("Great!")
elif a > 6:
        print ("Middle!")
else:
        print ("Low!")


Great!
```

控制流语句：while循环

- 条件循环while

```
while <逻辑条件>:
    <语句块>
    break #跳出循环
    continue #略过余下循环语句
    <语句块>
else: #条件不满足退出循环，则执行
    <语句块>
```

- else中可以判断循环是否遭遇了break

```
>>> n = 5
>>> while n > 0:
        n = n - 1
        if n < 2:
            break
        print (n)

4
3
2
```

```
>>> n = 5
>>> while n > 0:
        n = n - 1
        if n < 2:
            continue
        print (n)
else:
        print ('END!')


4
3
2
END!
```

控制流语句：for循环

- 迭代循环for：

```
for <变量> in <可迭代对象>:
    <语句块>
    break #跳出循环
    continue #略过余下循环语句
else: #迭代完毕，则执行
    <语句块>
```

- 可迭代对象有很多类型
  - 象字符串、列表、元组、字典、集合等
  - 也可以有后面提到的生成器、迭代器等

```
>>> for n in range(5):
        print (n)

0
1
2
3
4
```

```
>>> alist = ['a', 123, True]
>>> for v in alist:
        print (v)

a
123
True
```

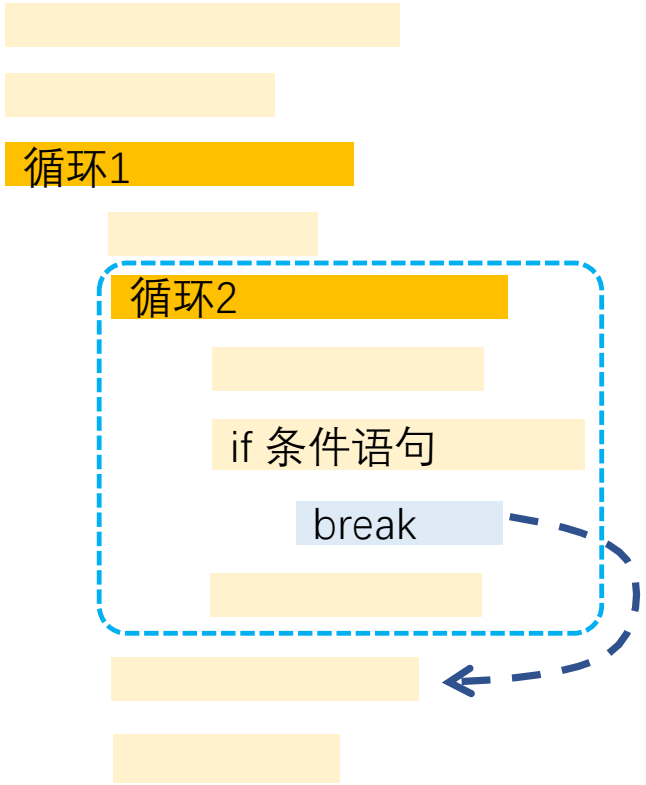
```
>>> adic = {'name':'Tom', 'age':18, 'gender':'Male'}
>>> for k in adic:
        print (k, adic[k])

name Tom
age 18
gender Male
```

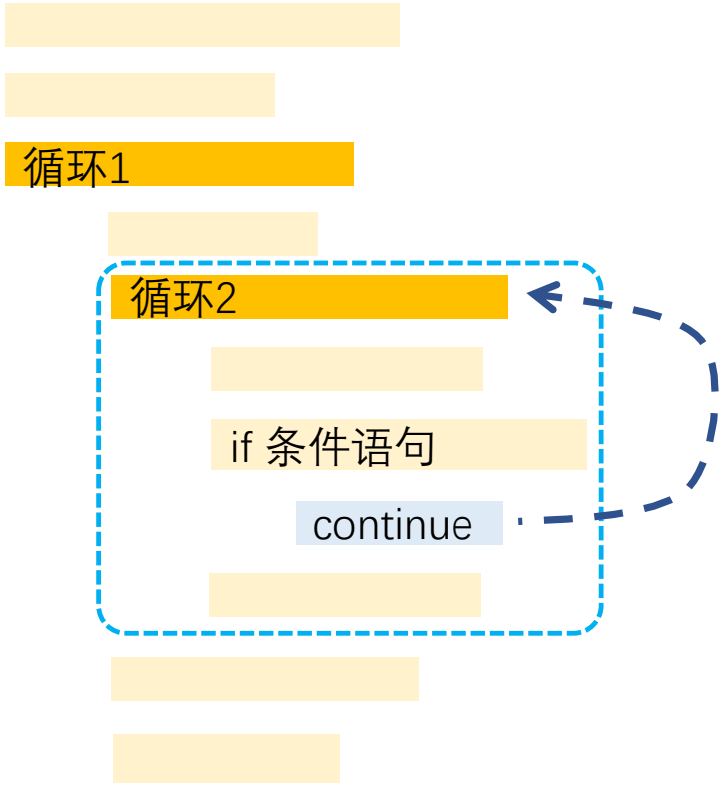
```
>>> for k, v in adic.items():
        print (k, v)

name Tom
age 18
gender Male
```

循环语句中的循环体控制

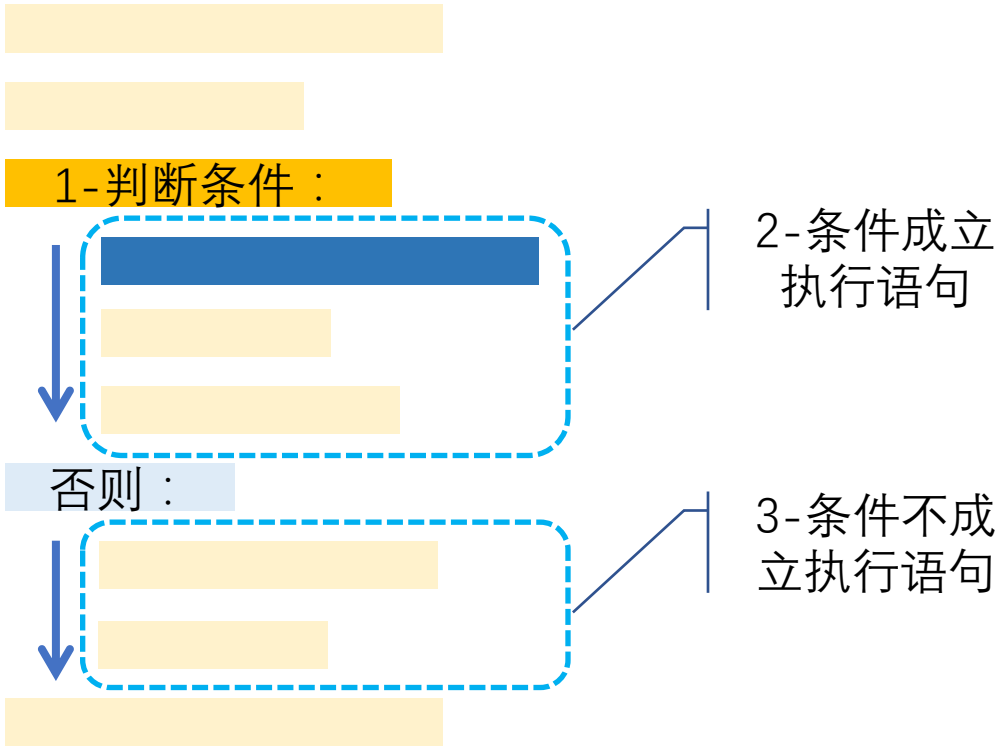


break语句跳出最近一层循环



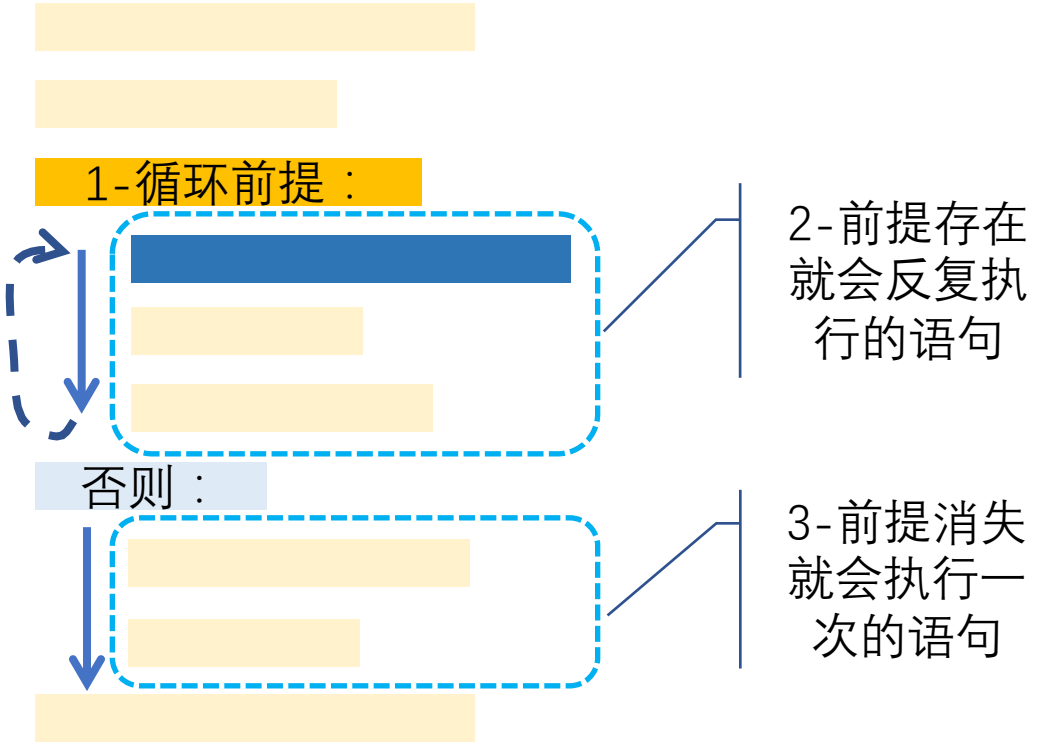
continue语句忽略同一层循环剩余语句

循环语句中的else



条件分支结构扩展要素1

可以用于判断是否break强制退出循环



循环结构扩展要素

上机练习：画一个圆

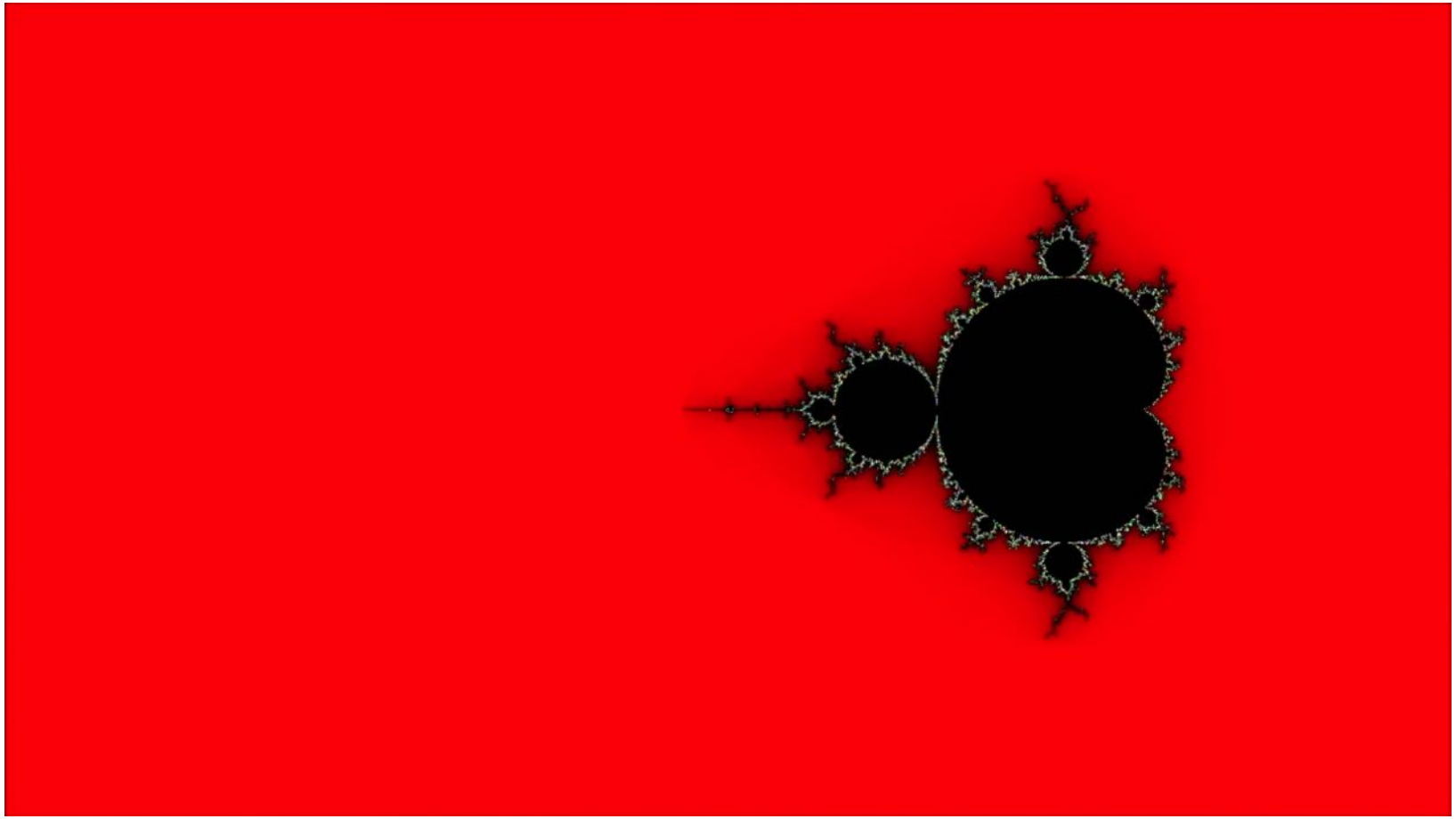
- 字符构成的圆
- 输入n
- 画出一个由#字符构成的圆，半径为n
- 背景是字符“+”
- 采用圆方程

- 海龟作图的圆（不用circle函数）
- 输入n
- 画出一个由红色点构成的圆，半径为n
- 背景是蓝色点构成
- 采用圆方程

上机练习：曼德勃罗集合（Mandelbrot Set）

- 用海龟作图（100*100）
- 对平面上的每个点（复数）c=x+yi进行检验：
  - 从z=0开始，$f(z)=z^2+c$
  - 序列0, f(0), f(f(0)), f(f(f(0)))....
- 如果序列收敛在有限区域，那么c属于集合（设为黑色），否则不属于集合（保持背景白色）
  - 一般在最大迭代次数max_repeats内，如果这个复数点没有超出逃逸圆（半径为2），则属于集合

奇幻的旅程



定义函数的参数：固定参数／可变参数

- 定义函数时，参数可以有两种；
- 一种是在参数表中写明参数名key的参数，固定了顺序和数量
  - def func(key1, key2, key3…):
  - def func(key1, key2=value2…):
- 一种是定义时还不知道会有多少参数传入的可变参数
  - def func(*args): #不带key的多个参数
  - def func(**kwargs): #key=val形式的多个参数

```
16  def func_test(key1, key2, key3=23):
17      print("k1=%s,k2=%s,k3=%s" % (key1, key2, key3))
18
19
20  print("====func_test")
21  # 没有传入key3，用了缺省值
22  func_test('v1', 'v2')
23  # 传入了key3
24  func_test('ab', 'cd', 768)
25  # 使用参数名称就可以不管顺序
26  func_test(key2='KK', key1='K')
```

```
====func_test
k1=v1,k2=v2,k3=23
k1=ab,k2=cd,k3=768
k1=K,k2=KK,k3=23
```

定义函数的参数：固定参数／可变参数

```
29  # 可以随意传入0个或多个无名参数
30  def func_test2(*args):
31      for arg, i in zip(args, range(len(args))):
32          print("arg%d=%s" % (i, arg))
33
34
35  print("====func_test2")
36  func_test2(12, 34, 'abcd', True)
```

```
====func_test2
arg0=12
arg1=34
arg2=abcd
arg3=True
```

```
39  # 可以随意传入0个或多个带名参数
40  def func_test3(**kwargs):
41      for key, val in kwargs.items():
42          print("%s=%s" % (key, val))
43
44
45  print("====func_test3")
46  func_test3(myname="Tom", sep="comma", age=23)
```

```
====func_test3
sep=comma
age=23
myname=Tom
```

调用函数的参数：位置参数／关键字参数

- 调用函数的时候，可以传进两种参数；
- 一种是没有名字的位置参数
  - func(arg1, arg2, arg3…)
  - 会按照前后顺序对应到函数参数
- 一种是带key的关键字参数
  - func(key1=arg1, key2=arg2…)
  - 由于指定了key，可不按顺序对应
- 如果混用，所有位置参数必须在前，关键字参数必须在后

```
16  def func_test(key1, key2, key3=23):
17      print("k1=%s,k2=%s,k3=%s" % (key1, key2, key3))
18
19
20  print("====func_test")
21  # 没有传入key3，用了缺省值
22  func_test('v1', 'v2')
23  # 传入了key3
24  func_test('ab', 'cd', 768)
25  # 使用参数名称就可以不管顺序
26  func_test(key2='KK', key1='K')
```

```
====func_test
k1=v1,k2=v2,k3=23
k1=ab,k2=cd,k3=768
k1=K,k2=KK,k3=23
```

函数小技巧：map()函数

- 有时候，需要对列表中每个元素做一个相同的处理，得到新列表
  - 例如所有数据乘以3
  - 例如所有字符串转换为整数
  - 例如两个列表对应值相加
- map(func, list1, list2….)
  - 函数func有几个参数，后面跟几个列表

```
num = [10, 20, 40, 80, 160]
lst = [2, 4, 6, 8, 10]
def mul3(a):
    return a * 3

print (list( map(mul3, num) ))

def atob(a, b):
    return a + 1.0/b

print (list( map(atob, num, lst) ))
```

```
[30, 60, 120, 240, 480]
[10.5, 20.25, 40.166666666666664, 80.125, 160.1]
```

函数小技巧：匿名函数lambda

- 有时候，函数只用一次，其名称也就不重要，可以无需费神去def一个
- Lambda表达式可以返回一个匿名函数
  - lambda <参数表>:<表达式>

```
num = [10, 20, 40, 80, 160]
lst = [2, 4, 6, 8, 10]
def mul3(a):
    return a * 3

print (list( map(mul3, num) ))

def atob(a, b):
    return a + 1.0/b

print (list( map(atob, num, lst) ))

print (list( map(lambda a:a * 3, num)))
print (list( map(lambda a,b:a+1.0/b, num, lst)))
```

上机练习：函数定义

- 水仙花数判定：创建一个函数，接受一个参数n(n>=100)，判断这个数是否为水仙花数
  - 即满足如果这个数为m位数，则每个位上的数字的m次幂之和等于它本身,例如1^3 + 5^3 + 3^3 = 153，1^4+6^4+3^4+4^4=1634），返回True或者False。
- 创建一个函数，接受一个参数max(max>=1000)，调用上题编写的判断函数，求100到max之间的水仙花数。
- 创建一个函数minus，可以接受1个或者2个数值参数，如果2个参数，返回差；如果1个参数返回负数
  - 如minus(4, 2)是2
  - minus(4)是-4
- 创建一个函数avg，可以接受不确定个数的数值参数，返回这些参数的平均值。
  - 如avg(12, 34, 45, 44, 32)
  - avg(1, 2, 3)这样

上机练习：map函数

- 编写程序
- 输入以空格隔开的一些正整数
- 输出这些正整数占总和的比重分别是多少，保留小数点后2位
- 如：输入1 2 3
- 输出0.17 0.33 0.50

- 编写程序
- 输入一些空格隔开的英文单词
- 输出这些单词的分行居中排列
  - 最长的单词两头没有空格
- 如：输入hello world see i
- 输出：
  - hello
  - world
  - see
  - i

面向对象：什么是对象？

- Python中的所有事物都是以对象形式存在
  - 从简单的数值类型，到复杂的代码模块，都是对象。
- 对象以id作为标识，既包含数据（属性），也包含代码（方法）
  - 赋值语句给予对象以名称，对象可以有多个名称（变量引用），但只有一个id
  - 同一类（class）的对象具有相同的属性和方法，但属性值和id不同
- 对象实现了属性和方法的封装，是一种数据抽象机制

```
>>> id(1)
4297537952
>>> type(1)
<class 'int'>
>>> dir(1)
['__abs__', '__a
r__', '__dir__',
_floordiv__', '_
t__', '__hash__'
_lshift__', '__l
r__', '__pos__',
 '__reduce_ex__'
l__', '__ror__',
', '__rtruediv
, '__
ugate
```

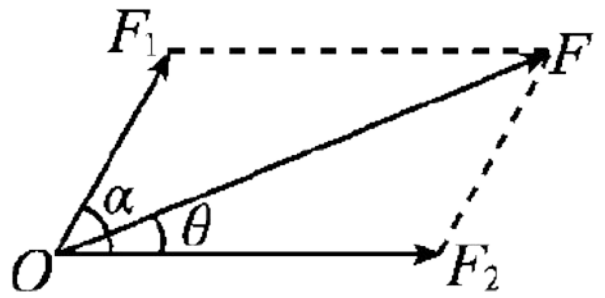
```
>>> id('a')
4300773280
>>> type('a')
<class 'str'>
>>> dir('a')
['__add__', '__class__', '__contains__', '__
_eq__', '__format__', '__ge__', '__getattrib
__', '__gt__', '__hash__', '__init__', '__it
__mod__', '__mul__', '__ne__', '__new__',
epr__', '__rmod__', '__rmul__', '__setattr__
```

```
>>> abs(-1)
1
>>> id(abs)
4298931872
>>> type(abs)
<class 'builtin_function_or_method'>
>>> dir(abs)
['__call__', '__class__', '__delattr__', '__
at__', '__ge__', '__getattribute__', '__gt__
 '__lt__', '__module__', '__name__', '__ne__
uce__', '__reduce_ex__', '__repr__', '__self
_str__', '__subclasshook__', '__text_signatu
```

面向对象：类的定义与调用

- 类是对象的模版，封装了对应现实实体的性质和行为
- 定义类：class语句；
  - class <类名>:
  - def __init__(self, <参数表>):
  - def <方法名>(self, <参数表>):
- 调用类：<类名>（<参数>）
  - 调用类会创建一个对象，（注意括号！）
  - obj = <类名> (<参数表>)
  - 返回一个对象实例，
  - 类方法中的self指这个对象实例！

```
1  class Force:  # 力
2      def __init__(self, x, y):  # x,y方向分量
3          self.fx, self.fy = x, y
4
5      def show(self):  # 打印出力的值
6          print("Force<%s,%s>" % (self.fx, self.fy))
7
8      def add(self, force2):  # 与另一个力合成
9          x = self.fx + force2.fx
10         y = self.fy + force2.fy
11         return Force(x, y)
12
13
14 # 生成一个力对象
15 f1 = Force(0, 1)
16 f1.show()
17
18 # 生成另一个力对象
19 f2 = Force(3, 4)
20 # 合成为新的力
21 f3 = f1.add(f2)
22 f3.show()
```




```
Force<0,1>
Force<3,5>
```

对象属性和方法的引用

- 通过<对象名>.<属性名>的形式引用，可以跟一般的变量一样用在赋值语句和表达式中
- Python语言动态的特征，使得对象可以随时增加或者删除属性或者方法
  - 也必须先赋值再引用

```
44  print(f3.fx, f3.fy)
45  f3.fz = 3.4
46  print(f3.fz)
47  del f3.fz
```

```
0.0 4.5
3.4
```

类定义中的特殊方法

- 在类定义中实现一些特殊方法，可方便地使用python一些内置操作
  - 所有特殊方法以两个下划线开始结束
  - __str__(self)：自动转换为字符串
  - __repr__(self)：转换为“正式”字符串
  - __add__(self, other)：使用+操作符
  - __mul__(self, other)：使用*操作符
  - __eq__(self, other)：使用==操作符
- 其它特殊方法参见课程网站
  - http://gis4g.pku.edu.cn/python-magic-method/

```
13      __add__ = add
14
15      def __str__(self):
16          return "F<%s,%s>" % (self.fx, self.fy)
17
18      def __mul__(self, n):
19          x, y = self.fx * n, self.fy * n
20          return Force(x, y)
21
22      def __eq__(self, force2):
23          return (self.fx == force2.fx) and \
24                 (self.fy == force2.fy)
```

```
37  # 操作符使用
38  f3 = f1 + f2
39  print("Fadd=%s" % (f3,))
40  f3 = f1 * 4.5
41  print("Fmul=%s" % (f3,))
42  print("%s==%s? -> %s" % (f1, f2, f1 == f2))
```

```
Fadd=F<3,5>
Fmul=F<0.0,4.5>
F<0,1>==F<3,4>? -> False
```

自定义对象的排序

- Python列表类型的sort方法和内置排序函数sorted()
  - 每种数据类型可以定义特殊方法def __lt__(self, y)
  - 返回True视为比y“小”，排在前，而返回False视为比y“大”，排在后
  - 任何自定义类都可以使用x<y这样的比较，只要类中定义了特殊方法__lt__
- 例子：Student
  - 姓名，成绩
- 按照成绩排序
  - 由高到低
- 用内置sort

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

    # 内置sort函数只引用 < 比较符来判断前后
    def __lt__(self, other):
        # 成绩比other高的，排在他前面
        return self.grade > other.grade

    # Student的易读字符串表示
    def __str__(self):
        return "(%s,%d)" % (self.name, self.grade)

    # Student的正式字符串表示，我们让它跟易读表示相同
    __repr__ = __str__
```

Python可扩展的“大小”比较及排序

- 我们构造一个Python列表
- 在列表中加入Student对象
- 直接调用列表的sort方法
- 可以看到已经根据__lt__定义排序
- 直接检验Student对象的大小
  - <
- 另外可以定义其它比较符
  - __gt__等

```
# 构造一个Python List对象
s = list()

# 添加Student对象到List中
s.append(Student("Jack", 80))
s.append(Student("Jane", 75))
s.append(Student("Smith", 82))
s.append(Student("Cook", 90))
s.append(Student("Tom", 70))
print("Original:", s)

# 对List进行排序，注意这是内置sort方法
s.sort()

# 查看结果，已经按照成绩排好序
print("Sorted:", s)
```

```
============= RESTART: /Users/chenbin/Documents/homework/stu.py =
Original: [(Jack,80), (Jane,75), (Smith,82), (Cook,90), (Tom,70)]
Sorted: [(Cook,90), (Smith,82), (Jack,80), (Jane,75), (Tom,70)]
>>> s[0]<s[1]
True
>>>
```

Python可扩展的“大小”比较及排序

- 我们可以把__lt__方法重新定义，改为比较姓名
- 这样sort方法就能按照姓名来排序

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

    # 内置sort函数只引用 < 比较符来判断前后
    def __lt__(self, other):
        # 姓名字母顺序在前，就排在他前面
        return self.name < other.name

    # Student的易读字符串表示
    def __str__(self):
        return "(%s,%d)" % (self.name, self.grade)

    # Student的正式字符串表示，我们让它跟易读表示相同
    __repr__ = __str__
```

```
============== RESTART: /Users/chenbin/Documents/homework/stu2.py =
Original: [(Jack,80), (Jane,75), (Smith,82), (Cook,90), (Tom,70)]
Sorted: [(Cook,90), (Jack,80), (Jane,75), (Smith,82), (Tom,70)]
>>> s[0]<s[1]
True
>>>
```

类的继承机制：代码复用

- 如果两个类具有“一般-特殊”的逻辑关系，那么特殊类就可以作为一般类的“子类”来定义，从“父类”继承属性和方法
  - class <子类名>(<父类名>):
  - def <重定义方法>(self,…):
- 子类对象可以调用父类方法，除非这个方法在子类中重新定义了（覆盖override）

类继承例子

```
71  gcar=GasCar("BMW")
72  gcar.fill_fuel(50.0)
73  gcar.run(200.0)
```

```
75  ecar=ElecCar("Tesla")
76  ecar.fill_fuel(60.0)
77  ecar.run(200.0)
```

```
BMW: run 200 miles!
Tesla: fuel out!
```

```
45  class Car:
46      def __init__(self, name):
47          self.name = name
48          self.remain_mile = 0
49
50      def fill_fuel(self, miles):  # 加燃料里程
51          self.remain_mile = miles
52
53      def run(self, miles):  # 跑miles英里
54          print (self.name, end=': ')
55          if self.remain_mile >= miles:
56              self.remain_mile -= miles
57              print("run %d miles!" % (miles,))
58          else:
59              print("fuel out!")
60
61
62  class GasCar(Car):
63      def fill_fuel(self, gas):  # 加汽油gas升
64          self.remain_mile = gas * 6.0  # 每升跑6英里
65
66
67  class ElecCar(Car):
68      def fill_fuel(self, power):  # 充电power度
69          self.remain_mile = power * 3.0  # 每度电3英里
```

子类与父类

- 子类可以添加父类中没有的方法和属性
- 如果子类同名方法覆盖了父类的方法，仍然还可以调用父类的方法

```
class GasCar(Car):
    def __init__(self, name, capacity):  # 名称和排量
        super().__init__(name)  # 父类初始化方法，只有名称
        self.capacity = capacity  # 增加了排量属性
```

关于self

- 在类定义中，所有方法的首个参数一般都是self
- self实际上代表对象实例
  - <对象>.<方法>(<参数>)
- 等价于：
  - <类>.<方法>(<对象>, <参数>)
- 这里的对象就是self了
- 如右图Line81和82

```
79  gcar = GasCar("BMW")
80  gcar.fill_fuel(50.0)
81  gcar.run(200.0)
82  GasCar.run(gcar, 200.0)
```

上机练习

- 创建一个类People
  - 包含属性name, city
  - 可以转换为字符串形式（__str__）
  - 包含方法moveto(self, newcity)
  - 可以按照city排序
  - 创建4个人对象，放到列表进行排序
- 创建一个类Teacher
  - 是People的子类，新增属性school
  - moveto方法改为newschool
  - 按照school排序
  - 创建4个教师对象，放到列表进行排序

- 创建一个mylist类，继承自内置数据类型list（列表）
  - 增加一个方法“累乘”product
    - def product(self):
    - 返回所有数据项的乘积。
  - 将list的+法改为两个mylist对应相加
    - def __add__(self, other):
    - 返回对应相加的mylist
  - 将list的*法改为两个mylist对应相乘再求和
    - def __mul__(self, other):
    - 返回对应相乘再求和的数值

【H8】打印杨辉三角形

- 编写程序
- 输入正整数n
- 打印输出杨辉三角形的前n行
- 如右图是n=6的情况
- 如果能输出为对称的三角形就更好了！

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```