程序设计基础
——函数

林大 经管学院 瞿华

函数

一. 函数参数与变量

二. 对象方法

三. 函数对象

四. Map、Filter与Reduce

五. 返回函数的函数*

六. 异常处理

函数

一. **函数参数与变量**

二. 对象方法

三. 函数对象

四. Map、Filter与Reduce

五. 返回函数的函数*

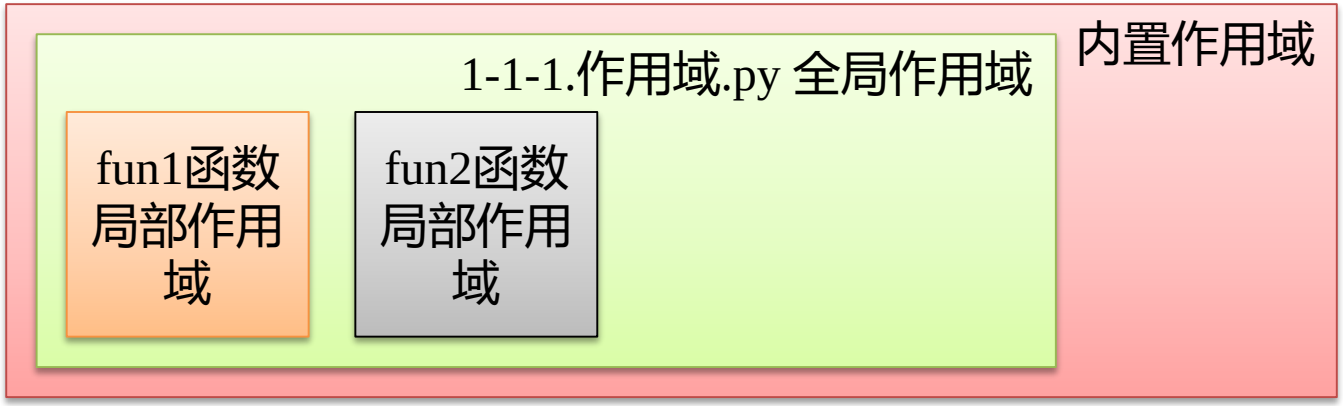
六. 异常处理

1.1 作用域（1）

- 在介绍模块化与函数时，我们提到过，函数是**相对独立**的功能单位。
- 我们可以把函数想象成一个功能黑盒子，给它必要的输入，它运行后就能返回相应的结果
- 独立性要求，函数在运行时不会对外界造成不必要的影响：
  - 比如，不小心改变了外界变量的指向（python中变量就是指向某个对象的标签）
- 因此，在包括python在内的大多数程序设计语言中，都有**作用域（Scope）**的概念。
- 在Python官方教程中对作用域定义如下：作用域就是Python程序中变量可以被直接访问到(directly accessible)的文字区域(textual area)。

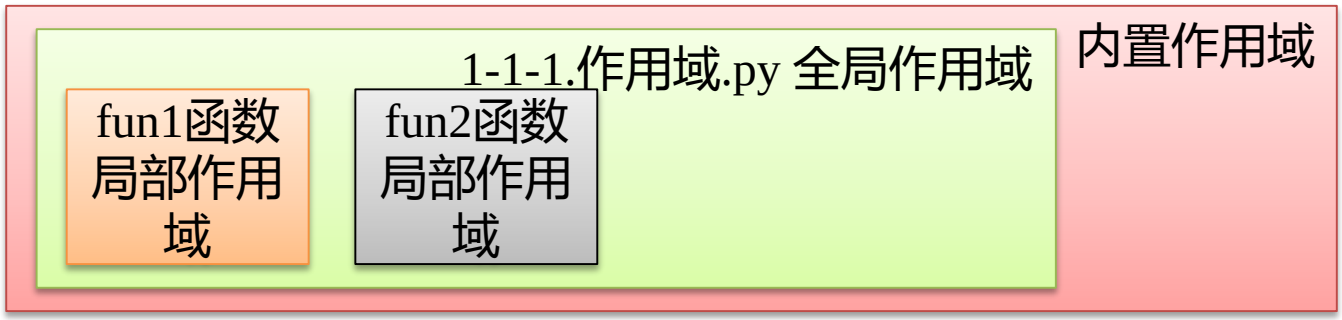
1.1 作用域（2）

- 在python中，每个模块和函数都有自己的作用域。
- 不考虑函数嵌套（在函数的内部定义另外一个函数）的情况，我们可以简单的认为，python程序有三个作用域：
  - 局部作用域（local）：某个函数内部的作用域。局部作用域中的变量称为**局部变量**。
  - 全局作用域（global）：整个模块（程序）的作用域。全局作用域中的变量称为**全局变量**。
  - 内置作用域（built-in）：python所有内置函数和变量的作用域。



1.1 作用域（3）

- 作用域访问的规则：
  - 局部作用域被包含在全局作用域中。
  - 在某个作用域中定义的变量，只能在这个作用域中被访问到。
  - 程序在访问变量时，从最内层的作用域开始逐层查找。也就是说，如果最内层的作用域中没找到该变量，python会到包含其的外层作用域中继续查找。直到最后外层作用域也找不到为止。
  - 如果没有特别指明，变量定义在其被定义的语句所在的最内层的作用域中。

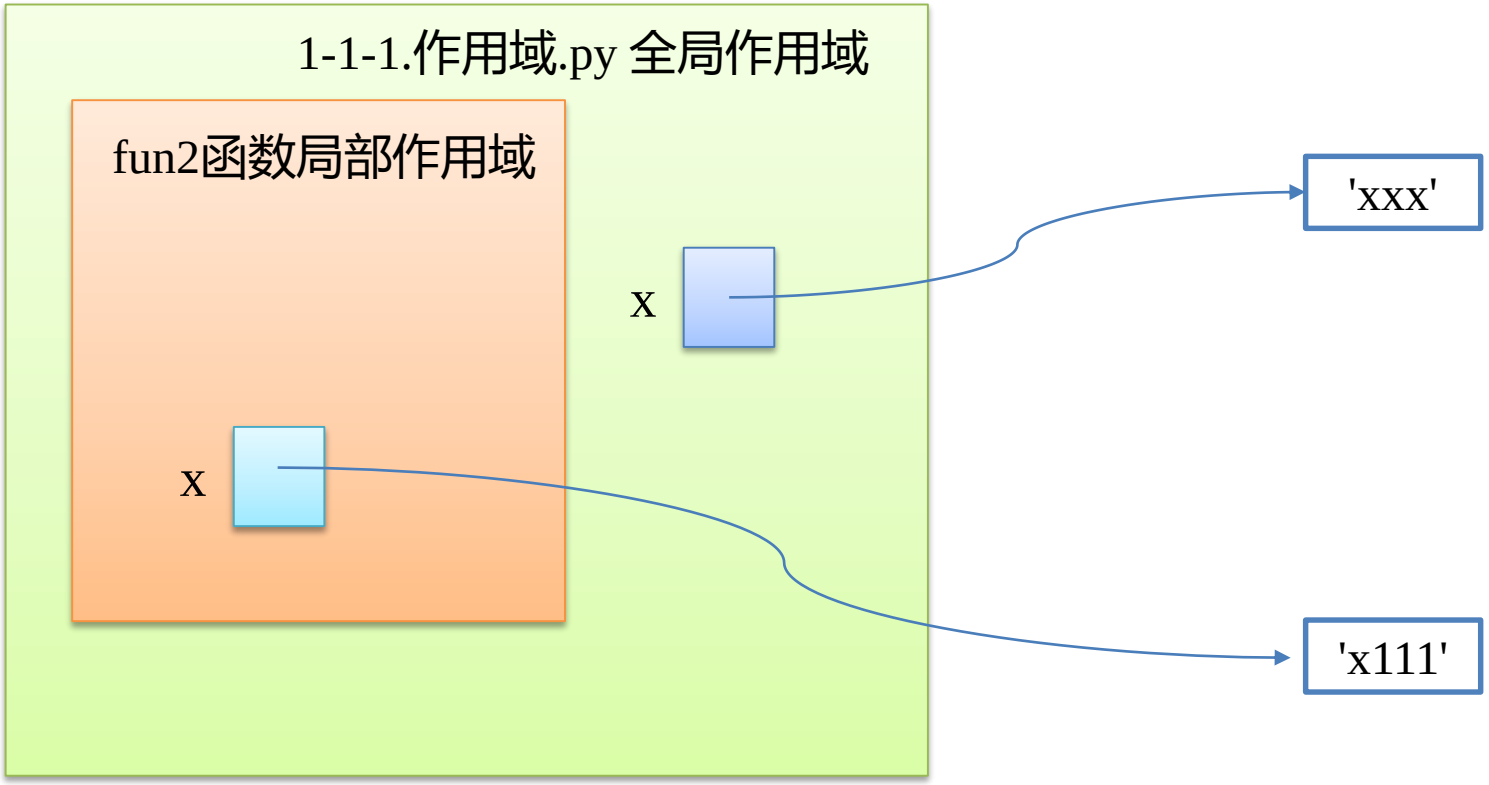


1.1 作用域（4）

- 在python中，可以用locals()和globals()函数分别获取当前的局部作用域和全局作用域中的所有变量。
- 观看程序1-1-1.作用域.py及其运行结果，思考：
  - 函数fun1()中的x和y是哪儿来的?
  - 为什么fun2()函数运行时，x和y输出的结果是"x222"和"y222"，而不是"xxx"和"yyy"?
  - 为什么fun1()和fun2() 运行完之后，x和y输出的结果是"xxx"和"yyy"，而不是"x333"和"y333"?
- 程序1-1-1.作用域.py的运行结果告诉我们：局部作用域和全局作用域的同名变量实际上是两个互不影响的不同变量（只是恰好重名而已）。
  - 因此，作用域保证了函数中的局部变量不会影响函数外的变量，从而实现了函数的独立性。

1.2 函数参数与对象可变性

- 作用域保证了函数的独立性。在函数中改变一个局部变量的指向，不会影响函数外同名变量的指向：



1.2.1 形式参数与实际参数（1）

- 大部分函数都有参数，为其提供输入或者说初始条件
  - 函数定义中的参数我们称为**“形式参数”（formal parameters）**
- 在要运行函数时，给函数的各个参数提供的实际值我们称为**“实际参数”（actual parameters）**

```
def calc_triangle_area(a,b,c):
    """
    使用海伦公式计算三角形面积
    a、b、c为三角形边长
    :return: 三角形面积
    """
    p=(a+b+c)/2
    return math.sqrt(p*(p-a)*(p-b)*(p-c))
area=calc_triangle_area(3,4,5)
print(f"面积等于{area}")
```

a，b，c三个参数就是形式参数（函数定义中的参数）

3，4，5三个参数就是实际参数（运行函数时实际给的参数）

1.2.1 形式参数与实际参数（2）

```
def calc_triangle_area(a,b,c):
    """
    使用海伦公式计算三角形面积
    a、b、c为三角形边长
    :return: 三角形面积
    """
    p=(a+b+c)/2
    print(locals())
    return math.sqrt(p*(p-a)*(p-b)*(p-c))
area=calc_triangle_area(3,4,5)
print(f"面积等于{area}")
```

猜一猜，print(locals())的输出结果会是什么?

1.2.1 形式参数与实际参数（3）

- 从上例中我们可以看出:
  - 形式参数实际上是函数内部的局部变量
  - 在函数运行时，会先将实际参数赋给形式参数

```
def swap(a,b):
    a,b = b,a

a,b=3,5
swap(a,b)
print(f"a={a}, b={b}")
```

上例中swap()函数试图通过赋值的方式交换变量a和b的值。

程序运行的结果是什么？a和b的值被成功交换了吗？为什么?

1.2.2 形参与对象可变性

- 上例告诉我们，在函数中无法通过重新赋值的形式，改变实际参数指向哪个对象。
- 但如果实际参数是可变对象，那我们可以通过形式参数来修改实际参数所指向的对象的内容。
- 我们无法让张三变成李四，但是我们可以给他修改发型，做个整容……
- 因为列表、字典和集合（set）都是可变对象，所以我们在函数中对这些形参的改变都会反映到实参中。

```
def add_elem(lst,ele):
    lst.append(ele)

my_lst=[1,2,3]
print(my_lst)
add_elem(my_lst,5)
print(my_lst)
```

注意，我们没有对形参lst重新赋值，而是通过其方法append()向它增加元素

1.2.2 形参与对象可变性（2）

- 关于形参与对象可变性的总结：
  - 无法通过形参来给实参重新赋值（改变其指向哪个对象）
  - 如果实参变量指向一个可变对象，那么因为形参和实参指向同一个对象，因此通过形参对对象所作的修改，也会通过实参反映出来
- **常用的可变对象：列表、字典、集合(set)等**
- **常用不可变对象：整数、浮点数、布尔值、字符串等**

1.3 函数与全局变量（1）

- 在1.1.节中，我们看到了，在函数中不能对全局变量进行重新赋值：
  - 只会在函数作用域中建立一个和全局变量同名的局部变量
  - 从保证函数的独立性角度出发，这样当然是合理的
- 但如果我们确实需要在函数中改变全局变量的值怎么办?
  - python提供的global关键字可以实现该功能

1.3 函数与全局变量（2）

```
def do_local():
    x = "x local"

def do_global():
    global x
    x = "x global"

x = "test"
do_local()
print("局部赋值后:", x)
do_global()
print("全局赋值后:", x)
```

1-3-1.函数中修改全局变量.py

1.4 参数匹配

- 在python中，可以用两种方式来匹配给函数的实际参数和形式参数：
  - 位置形式：这个是最基本的提供参数的形式，python根据参数的位置，来确定形参和实参的对应关系
  - 关键字形式：在提供实参时，在实参前加上“形参名=”。python根据所给的形参名来确定形参和实参的对应关系。

```
def fun1(a,b):
    print(f"a={a} b={b}")
```

函数定义。其有两个形式参数a和b

```
fun1(5,3)
```

根据参数的位置，python知道5给a，3给b

```
fun1(b=5,a=3)
```

根据形式参数的名称，python知道5给b，3给a

函数

一. 函数参数与变量

二. **对象方法**

三. 函数对象

四. Map、Filter与Reduce

五. 返回函数的函数*

六. 异常处理

二、对象方法

- 很多时候，函数和数据是紧密关联的
- 例2-1：模拟一个完全弹性的球，在一个二维平面框内的运动。（见2-1.ball.py）
- 分析：这是一个模拟问题，我们首先要搞清楚需要哪些量来完成模拟：
  - 因为是在二维平面上，所以所有的位置坐标和矢量都需要x和y两个分量
  - 小球：球心的位置（x和y坐标）、球的半径，球的速度矢量（x和y分量）
  - 框：因为框是一个两边分别和x和y轴平行的矩形，所以只需要框左上角（x和y坐标）和右下角位置（x和y坐标）即可

2.1 模拟小球运动（1）

```
class World:
    def __init__(self, left, top, right,
bottom):
        self.left = left  # 左上角x坐标
        self.top = top  # 左上角y坐标
        self.right = right  # 右下角x坐标
        self.bottom = bottom  # 右下角坐标
```

边框类

因为边框构成了小球所处的世界，
所以我们直接称其为world

2.1 模拟小球运动（2）

```
class Ball:
    def __init__(self,world,cx,cy,r,vx,vy):
        self.world = world # 球所在的边框（世界）
        self.cx=cx # 球心x坐标
        self.cy=cy # 球心y坐标
        self.r = r # 球心半径
        self.vx=vx # 速度x分量
        self.vy=vy # 速度y分量
```

球类

为了后面的处理方便，球需要知道自己所在的边框（以便进行碰撞检测）

2.1 模拟小球运动（3）

- 为了模拟方便，我们假设一个单位时间t（t的具体数值无所谓），这样就可以用循环来模拟时间的流逝：
  - 每循环一次，时间向前推进t。
- 因此，我们只需要模拟每次循环，整个系统（框和小球）的变化即可：
  - 框永远固定不动
  - 在未和边框发生碰撞时，小球始终保持原有的速度和方向移动
  - 如果发生垂直碰撞（和底部、顶部边框相撞），小球在水平方向上的速度分量不变，在垂直方向的速度分量反转（改变正负号）
  - 如果发生水平碰撞（和左侧、右侧边框相撞），小球在水平方向上的速度分量反转（改变正负号），在垂直方向的速度分量不变

2.2 对象方法

- 显然，模拟的核心就在于计算球的运动和碰撞。
- 因为运动和水平碰撞检测、垂直碰撞检测都是相对完整独立的功能，我们将其都包装成函数。
- 又因为无论是运动，还是碰撞检测都和球密切关联，所以我们将其作为球的**对象方法**。

```
class Ball:
    def move(self):
        new_x = self.cx+self.vx
        new_y = self.cy+self.vy
        if self.v_collide():
            self.vy = - self.vy
        if self.h_collide():
            self.vx = - self.vx
        self.cx += self.vx
        self.cy += self.vy
```

python中对象方法的第一个参数永远是self，表示对象本身

首先，我们计算出按原运动方向和速度前进，球的新位置

然后，我们判断球的新位置是否会和边框发生碰撞。如果会，则相应改变运动方向（碰撞）

最后按照新的运动速度和方向，计算出球的新位置

2.2.1 模拟小球运动（5）

```
class Ball:
    def v_collide(self):
        return self.cy < self.world.top + self.r \
               or self.cy > self.world.bottom - self.r

    def h_collide(self):
        return self.cx < self.world.left + self.r \
               or self.cx > self.world.right - self.r
```

垂直和水平碰撞检测方法

注意，在屏幕坐标系下，屏幕左上角坐标是(0,0),x轴正向向左，y轴正向向下（y轴方向和一般的平面直角坐标系相反）

简单起见，我们实际是检测球的外接正方形是否和边框相撞

2.2.1 模拟小球运动（6）

```
while is_run():
    ball.move()
    if delay_jfps(200):
        clear()
        set_color("black")
        rect(20,20,380,380)
        set_color("red")
        set_fill_color("red")
        fill_circle(ball.cx,ball.cy,ball.r)
```

模拟主循环

首先调用ball.move()方法，计算球的新位置

然后，用clear()函数擦除上次绘制的所有图形

接下来，用rect()函数绘制边框

最后，用fill_circle()函数绘制小球

2.3 对象方法的调用

```
class Ball:
    def move(self):
        ……


ball =Ball(……)
ball.move()
```

例2-1中Ball类的move方法定义

以及如何调用ball对象的move()方法。（ball对象是Ball类的一个实例）

注意，在move方法的定义中包含一个形式参数self

而在下面调用ball对象move方法的语句ball.move()中，并没有提供实际参数。

显然，在执行该语句时，python自动将ball对象作为参数提供给了move()方法。

```
Ball.move(ball)
```

实际上，我们可以用上面这种方式来调用ball对象的move方法，其效果和ball.move()是一样的。

可见，在python中，对象方法本质上和一般的函数没有区别。只不过用专门的语法进行了包装而已。

函数

一. 函数参数与变量

二. 对象方法

三. 函数对象

四. Map、Filter与Reduce

五. 返回函数的函数*

六. 异常处理

三、函数对象

- 在python中，一切编程元素都是对象，函数也不例外。
- 我们定义一个名为fun1的函数，实际上是定义了一个函数对象，同时定义了一个指向该对象的变量fun1

```
def fun1():
    print("this is fun1")

print(fun1)
fun1=34
print(fun1)
```

3-1-1.函数对象.py

3.1.1 函数赋值给变量

- 既然函数是一个对象，那么和其他对象一样，可以赋值给变量，也可以作为参数提供给别的函数，或者作为返回值被函数返回：

```
def fun1():
    print("this is fun1")

x = fun1
print("运行x所指的函数：")
x()
```

3-1-2.函数对象赋值.py

注意：要运行函数，在函数名后要加括号；
只是要访问函数对象，函数名后面不能有括号。

3.1.1 函数做参数

```
def fun1():
    print("this is fun1")

def fun2(f):
    print("this is fun2")
    f()

print("运行fun2:")
fun2(fun1)
```

3-1-3.函数对象做参数.py

注意：要运行函数，在函数名后要加括号；
只是要访问函数对象，函数名后面不能有括号。

3.1.1 函数做返回值

```
def fun1():
    def fun2():
        print("this if fun2")
    return fun2

print("运行fun1并将返回值赋给x：")
x=fun1()
print("运行x所指的函数对象：")
x()
```

3-1-4.函数对象做返回值.py

注意：要运行函数，在函数名后要加括号；
只是要访问函数对象，函数名后面不能有括号。

3.2 高阶函数及其应用

- 在程序设计中，这种以别的函数为参数或者返回值的函数，我们称为高阶函数（Higher-order Functions）
- 通过引入高阶函数，可以极大的简化我们的程序。
- 例3-2-1：已知2-2.成绩.csv中包含某系所有同学的学号、姓名、班级和语文、数学、英语考试成绩。请编程分别找出语文、数学和英语考试分数最高的同学。
- 分析：我们之前已经学习过如何使用迭代访问，寻找某一门分数最高的同学。因此我们可以这样：
  - 用列表保存所有同学的成绩信息（需要定义成绩信息类型）
  - 用迭代访问的方法找出语文最高分；再用迭代访问的方法找出数学最高分；最后用迭代的方法找出英语最高分。

```
class Score:
    def __init__(self,id,name,
clazz, math,literacy,english):
        self.id = id
        self.name = name
        self.clazz = clazz
        self.math = math
        self.literacy=literacy
        self.english = english
```

```
def find_max_math(scores):
    max = scores[0]
    for score in scores:
        if score.math > max.math:
            max = score
    return max
```

```
def find_max_literacy(scores):
    max = scores[0]
    for score in scores:
        if score.literacy > max.literacy:
            max = score
    return max
```

```
def find_max_english(scores):
    max = scores[0]
    for score in scores:
        if score.english > max.english:
            max = score
    return max
```

2-2-1.求各科最高分.py

3.2.1 高阶函数示例一（2）

- 在上例中，三个求数学、语文和英语最高分的函数唯一的区别只在于如何比较已找到的最高分同学信息和当前访问的学生信息元素之间的大小。
- 很自然的，如果我们把找最高分函数中的比较大小抽象成一个函数参数，那么只需要一个找最高分函数就可以找出不同课程的最高分。

```
def find_max(scores,cmp_key):
    max = scores[0]
    for score in scores:
        if cmp_key(score)>cmp_key(max):
            max = score
    return max
```

注意，find_max()的第二个参数cmp_key是一个函数参数。
其作用是返回给定成绩记录对象要进行比较大小的属性值（如果要比较数学成绩，就应该返回对象的数学成绩属性）

3.2.1 高阶函数示例一（3）

```
def find_max(scores,cmp_key):
    max = scores[0]
    for score in scores:
        if cmp_key(score)>cmp_key(max):
            max = score
    return max
```

```
def key_literacy(score):
    return score.literacy
```

```
def key_english(score):
    return score.english
```

```
def key_math(score):
    return score.math
```

注意：在提供函数参数时，函数名后不能加括号！

```
scores = read_csv(filename)
max_math = find_max(scores,key_math)
max_literacy = find_max(scores,key_literacy)
max_english = find_max(scores,key_english)
```

3.2.2 python内置函数

- 事实上，python及其标准库提供了大多数常用算法的实现函数，我们直接使用即可。
- 例3-2-3：已知2-2.成绩.csv中包含某系所有同学的学号、姓名、班级和语文、数学、英语考试成绩。请编程使用python内置的max函数和min()函数，分别找出语文、数学和英语考试分数最高的同学。

```
def key_literacy(score):
    return score.literacy
```

```
def key_math(score):
    return score.math
```

```
def key_english(score):
    return score.english
```

```
max_math = max(scores, key = key_math)
max_literacy = max(scores, key = key_literacy)
max_english = max(scores, key = key_english)
```

求最大值

注意这里使用了关键字形式的参数匹配

3.2.2 python内置函数（2）

- 例3-2-3：已知2-2.成绩.csv中包含某系所有同学的学号、姓名、班级和语文、数学、英语考试成绩。请编程使用python内置的max函数和min()函数，分别找出语文、数学和英语考试分数最高的同学。（续）

```
def key_literacy(score):
    return score.literacy
```

```
def key_math(score):
    return score.math
```

```
def key_english(score):
    return score.english
```

```
min_math = min(scores,
key=key_math)
min_literacy = min(scores,
key=key_literacy)
min_english = min(scores,
key=key_english)
```

求最小值

注意这里使用了关键字形式的参数匹配

3.2.2 python内置函数（3）

- 例3-2-4：已知2-2.成绩.csv中包含某系所有同学的学号、姓名、班级和语文、数学、英语考试成绩。请编程使用python内置的sorted()函数，分别按照语文、数学和英语考试成绩对学生进行排序。

```
def key_literacy(score):
    return score.literacy
```

```
def key_math(score):
    return score.math
```

```
def key_english(score):
    return score.english
```

```
scores =
sorted(scores,key=key_math,reverse
=True)
```

按数学成绩从高到低排序
reverse参数为True表示从大到小排序，为False表示从小到大排序

注意这里使用了关键字形式的参数匹配

函数

一. 函数参数与变量

二. 对象方法

三. 函数对象

四. Map、Filter与Reduce

五. 返回函数的函数*

六. 异常处理

4.1 迭代访问与Map、Filter和Reduce

- 迭代访问是到目前为止我们用的最多，也是最基本的（大量）数据处理方法。
- 总结起来，我们通过迭代访问实现的功能基本都可以归入三大类中：
  - **映射（Map）**：将列表中的各个元素逐一变换成另外一种形式。例：已知每个学生的语数英成绩，求每个同学的总分。
  - **过滤（Filter）**：找出列表中所有符合特定条件的元素。例：找出所有不及格的同学。
  - **化简（Reduce）**：将列表中所有元素逐一归并成计算形成唯一结果。例如：对数列中所有元素求和、求最大最小值
- 因此，我们可以将这三种类型的迭代访问编写成三个函数，然后直接使用这三个函数而不需要再写for循环。

4.1.1 基本实现

```
def map(fn,lst):
    new_lst = []
    for elem in lst:
        new_elem = fn(elem)
        new_lst.append(new_elem)
    return new_lst
```

map函数

```
def reduce(fn,lst,initial):
    result = initial
    for elem in lst:
        result = fn(result,elem)
    return result
```

reduce函数

```
def filter(fn,lst):
    new_lst = []
    for elem in lst:
        if fn(elem):
            new_lst.append(elem)
    return new_lst
```

filter函数

4.1.2 基本用法

```
def double(n):
    return n*2

def is_odd(n):
    return n%2==1

def add(a,b):
    return a+b

def hello(name):
    return f"{name},你好"
```

```
lst_a=list(range(10))
print(lst_a)
lst_b=['张三','李四','王五']

lst_c=map(double,lst_a)
print(lst_c)

lst_c = map(hello,lst_b)
print(lst_c)

lst_c = filter(is_odd,lst_a)
print(lst_c)

result = reduce(add,lst_a,0)
print(result)
```

4.1.3 python中的Map、Filter和Reduce

- 实际上，python内置了map()和filter()函数
  - python在标准库的functools包中还提供了reduce()函数。
  - 因为求和、求最大最小值是最常见的化简操作，因此在大多数情况下我们不需要使用reduce()函数，直接使用python内置的sum()、max()和min()等函数即可
- 由于python的map()和filter()函数返回的是只能迭代访问一次的迭代器对象（iterator）而不是列表，因此：
  - 如果我们需要列表形式的结果，或者需要对结果进行多次迭代访问的话，还需要**使用list()函数将结果转化为列表**。
- 此外：
  - python的operator包提供了各种运算符（加减乘除等）对应的函数
  - python的itertools包提供了在已有可迭代对象基础上生成新可迭代对象的一系列方法

4.1.4 python内置map说明（1）

```
import operator as op
n=op.add(3,5)
lst_a = list(range(10))
lst_b = list(range(10))
lst_c = map(op.add,lst_a,lst_b)
```

可以对多个列表做map，如果进行map的函数需要多个参数的话

```
import math
lst_c = map(math.sin,lst_c)
print("iteraction 1:")
for elem in lst_c:
    print(elem)
print("iteraction 2:")
for elem in lst_c:
    print(elem)
```

map返回的迭代器对象只能迭代一次

4.1.4 python内置map说明（2）*

```
from fractions import Fraction
f = Fraction(1,2)
lst_a = list(range(10))
lst_b = [10]*10
lst_c = map(Fraction,lst_a,lst_b)
```

类构造方法也可以map

```
f2 = Fraction(1,3)
f3 = f.__add__(f2)
lst_d = map(f.__add__,lst_c)
```

类普通方法也可以map

事实上，不止map，其他所有高阶函数都可以将类方法当作函数参数来使用

或者说，python中类（对象）方法和函数本质上没有区别，任何需要用函数的地方都可以用类（对象）方法

4.1.4 python内置map说明（3）*

```
s1=str.capitalize(s)
lst_a = ['hello','word','name','this','game']
lst_c = map(str.capitalize,lst_a)
```

像普通函数一样使用类方法

4.1.5 python内置reduce举例

```
import operator as op
lst_a = list(range(1,11))
a=reduce(op.add,lst_a,0)
```

```
lst_b = ['张三','李四','王五']
b=reduce(op.add,lst_b,'')
```

不仅仅可以对数进行reduce运算

```
lst_a=['张三','李四','王五','张三','王五','王五','
张三','李四']
def count(name_map,name):
    n=name_map.get(name,0)
    name_map[name]=n+1
    return name_map
name_map = reduce(count,lst_a,{})
```

统计不同字符串出现的次数

reduce运算接受另一种形式的函数：
以对象a和对象b为参数，返回新的对象a

4.2 综合举例

- 本节中我们使用map()、filter()函数和sum()函数来实现一些基本的功能
- 例4-2-1：求所有小于等于n的质数的平方和
- 用filter map reduce的框架来分析：
  - 初始列表：数列1到n（用range函数)
  - filter：判断是否质数
  - map：求平方
  - reduce：求和，直接用sum函数即可

程序见4-2-1.求质数平方和.py

4.2.2综合实例二（1）

- 例4-2-2：已知3-2.成绩.csv中包含某系所有同学的学号、姓名、班级和语文、数学、英语考试成绩。请使用map、filter和reduce求出电商18班同学的数学平均分。
- 分析：使用map、filter和reduce来编写程序，最核心的是把求解过程拆分成map、filter和reduce的基本运算。本例可以这样做：
  1. 找出电商18班的所有同学，组成新可迭代对象lst1（filter运算）
  2. 求出lst1中每位同学的数学成绩，组成可迭代对象lst2（map运算）
  3. 对lst2中所有元素求总和（reduce运算，直接用sum()函数即可）
  4. 求lst2中元素数量（先将lst2转化为列表，然后用len()函数求元素数量）
  5. 平均成绩=总和/元素数量

注：3、4、5步也可以改用statistics包中的mean函数来求平均值
见4-2-2-1.求某班数学平均分-使用mean函数.py

4.2.2 综合实例二（2）

```
def score_to_math(score):
    return score.math
```

求同学的数学成绩

```
def filter_by_class(score):
    return score.clazz == '电商18'
```

判断是否在电商18班

4-2-2.求某班数学平均分.py

注：也可以使用statistics模块中的mean来求平均值

主程序

```
lst1=filter(filter_by_class,scores)
lst2=list(map(score_to_math,lst1))
total = sum(lst2)
count = len(lst2)
average = total / count
print(f"电商18班{count}名同学数学平均分为{average:.2f}")
```

注意：函数参数在前，要迭代访问的列表或可迭代对象参数在后。

因为len()要求参数为列表类型，所以用list()函数将map()的结果转化为列表

如果只求和不求平均值的话，这里可以不用转化为列表。

4.3 大数据与Map、Filter和Reduce

- 使用map/filter/reduce高阶函数和直接使用for循环来进行迭代访问，在本质上是相同的。
  - 其实在python里，map/reduce高阶函数比for循环更慢
- 但是在大数据时代，高阶函数形式的迭代访问体现出了独特的优势：更适合并行处理
- 简单的说，并行处理就把原来一台电脑或者一块CPU处理的工作，分给多台电脑或者CPU来同时进行。
  - 因为map/filter/reduce函数中每次迭代访问都是相对独立，而且与顺序关系无关的，因此，可以直接简单的将要遍历的列表划分成多个子列表，然后分给多个处理器同时对这些子列表进行迭代访问，化简出结果。最后再对各处理器上的化简结果进行合并，形成最终结果。
- 见4-3.并行mapfilterreduce.py

函数

一. 函数参数与变量

二. 对象方法

三. 函数对象

四. Map、Filter与Reduce

五. 返回函数的函数*

六. 异常处理

五 返回函数的函数

- 函数除了可以用作其他函数的参数外，也可以用作其他函数的返回值。
- 例5-1-1：已知2-2.成绩.csv中包含某系所有同学的学号、姓名、班级和语文、数学、英语考试成绩。请求出指定班级所有同学三科总分的平均分。
- 分析：可以使用MapFilter形式来完成。map求总分不难，最大的问题在于用于filter筛选的条件——班级——需要在程序运行时由用户输入。
  - 我们在例3-2-1里提供给filter函数的过滤函数filter_by_class，班级是直接以字面量的形式固化在代码里的。
  - 如何需要修改该函数，才能使其既满足作为filter函数参数的需要（只能有score一个参数），又能让班级成为一个可变的条件呢?

```
def filter_by_class(score):
    return score.clazz == '电商18'
```

求三科总分函数

```
def score_to_total(score):
    return score.math+score.english+score.literacy
```

五 返回函数的函数（2）

- 最简单的方法是将班级作为一个全局变量clazz：

```
def filter_by_class(score):
    return score.clazz == clazz
```

判断是否在指定的班级
注意clazz是一个全局变量

```
clazz = dlg.get_string("请输入班级名称")
lst1=filter(filter_by_class,scores)
lst2=list(map(score_to_total,lst1))
total = sum(lst2)
count = len(lst2)
average = total / count
```

4-1-1.求某班平均分-全局变量实现.py

但是在函数中引入全局变量，破坏了模块化的独立性原则，所以应该尽量避免使用！

五 返回函数的函数（3）

- 我们可以使用返回函数的高阶函数来解决该问题：

```
def generate_filter_by_class(clazz):
    def filter(score):
        return score.clazz == clazz
    return filter
```

根据所给的clazz参数，返回一个判断学生是否在该班级内的函数

```
clazz = dlg.get_string("请输入班级名称")
filter_by_class = generate_filter_by_class(clazz)
lst1=filter(filter_by_class,scores)
lst2=list(map(score_to_total,lst1))
total = sum(lst2)
count = len(lst2)
average = total / count
```

我们先用generate_filter_by_class函数，生成需要的filter_by_class函数，再将其作为参数提供给filter函数进行过滤。

4-1-2.求某班平均分-高阶函数实现.py

五 返回函数的函数（4）

- 自己写高阶函数看上去有些别扭（因为不习惯）
- 使用python标准库functools包中的partial函数，我们可以用更简单的方式来生成想要的函数：

```
def filter_by_class_base(score, clazz):
    return score.clazz == clazz
```

1.先定义一个两参数的普通过滤函数

```
my_clazz = dlg.get_string("请输入班级名称")
filter_by_class = partial(filter_by_class_base, clazz = my_clazz)
lst1=filter(filter_by_class,scores)
lst2=list(map(score_to_total,lst1))
```

2.使用functools包中的partial函数，在原函数的基础上，通过固定指定的参数，生成想要的新函数：本例中，固定filter_by_class_base的clazz参数的值为my_clazz因为原函数有两个参数score和clazz，固定clazz之后，得到的新函数就只有一个参数score了。

函数

一. 函数参数与变量

二. 对象方法

三. 函数对象

四. Map、Filter与Reduce

五. 返回函数的函数*

六. 异常处理

六 异常处理

- 在正常情况下，函数通过返回值的形式返回计算结果
- 但在某些情况下，函数可能无法正常完成计算。如何通知调用它的程序，处理失败？
- 现代程序设计语言中，大多数采用异常（Exception）机制来报告和处理影响程序正常进行的问题。
- 在Python中，异常就是一种特殊类型的对象。
- 在程序的任意位置，可以通过raise语句来抛出（throw）一个异常对象。

```
print(a)
```

如果变量a被赋值，运行
上面的语句会产生NameError

```
raise NameError('a is not defined')
```

手动生成一个NameError异常对象并抛出

6.1 用异常来报告函数错误

- 异常最基本的用法就是报告会导致函数无法正常工作的错误。
- 在第一章《结构化程序设计基础》中的“防卫式编程”一节中，我们已用异常来报告参数错误：

```
def calc_sales_price(price):
    if price<0:
        raise ValueError("price should not < 0!")
    sales=0.9*price
    return sales
```

Chap01Basics　6-2.防卫式编程-打9折.py

raise会中止所在函数的执行，返回到调用它的上一级函数（或主程序）；

如果上一级函数未设置异常捕获(catch)，那么会继续中止这一级函数的执行，继续返回上上一级；

这个过程会一直重复，直到异常被捕获，或者整个程序中止执行为止。

5.2 捕获异常

```
try:
    price=float(dlg.get_string("请输入价格："))
    sales_price=calc_sales_price(price)
    dlg.show_message(f"打折后价格为{sales_price}")
except ValueError as e:
    print("出现错误",e)
finally:
    print("我总会被执行")
```

6-2.捕获异常.py

使用try … except 来捕获异常

try里面是要正常执行的代码段

except是如果try代码段执行出现指定的异常，进行的处理

finally是不论是否出现异常，最后都会被执行的代码

编程思路总结

- 在循环的编程思路总结中，我们提到，通过找规律发现重复性的步骤，并用循环实现，是编程的基本能力
- 更进一步的，找出常用或者类似的问题解决步骤，封装成函数，以减少重复开发，是提升编程工作效率的重要途径。
- 本章所学习的高阶函数（以函数为参数或者返回值），可以帮助我们最大程度的提高函数的通用性。（比如排序、求最大值等）
- Map、Filter和Reduce给我们提供了迭代访问列表（或任意可迭代对象）以解决问题的另外一种思路。在大数据分析处理中，经常会见到运用这种思路编写的处理程序。
- 将方法（函数）和属性（相关数据）整合到一个程序元素（对象）中，是目前的主流程序设计方法——面向对象程序设计的基本思路。我们要逐步习惯和适应这种编程思路和语法。

本章重点

- 变量的作用域
- 函数做参数及其应用
- Map、Filter和Reduce及其应用
- 异常处理的基本方法（以及防卫式编程）
- python中对象方法的声明语法

本章练习

- 见练习《第四章 函数》