程序设计基础
——数据结构与算法入门

林大 经管学院 瞿华

数据结构与算法入门

一. 列表

二. 使用对象自定义数据类型

三. 文件输入输出

四. 迭代访问及其应用

五. 查找与排序

六. 字典与集合

数据结构与算法入门

一. **列表**

二. 使用对象自定义数据类型

三. 文件输入输出

四. 迭代访问及其应用

五. 查找与排序

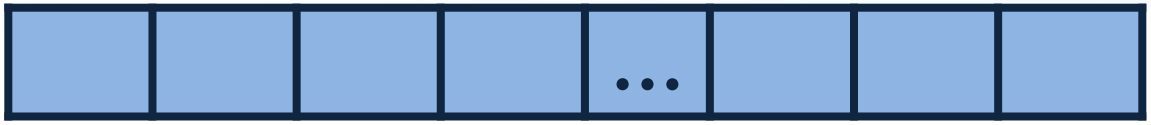
六. 字典与集合

一、列表

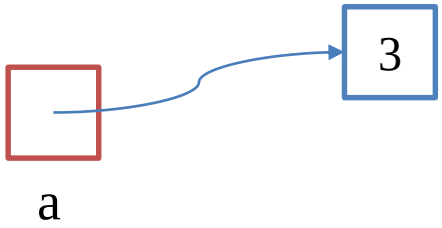
- 回忆：如何使用内存存取数据？
- 但是：如果程序需要排序比较全校所有学生的成绩，该怎么办？
- 现实中的程序往往需要处理大量的数据。针对程序要解决的不同性质的问题：
  - 应如何在内存中组织和存取这些数据？
  - 程序如何高效的处理这些数据以解决问题？
- **数据结构**（data structure）：（**大量**）数据在内存中的存放和组织形式。
- 对于C、Java这类需要声明数据类型的语言来说，最基本的数据结构是**数组（Array）**：
  - 将一组相同类型的数据顺序依次存放在内存中，就构成了一个数组。

一、列表（2）

- 数组：



- python中也提供了数组，但是由于数组在定义时必须确定每个格子的大小，而我们知道python中的变量所占空间大小是不确定的。因此在python中直接使用数组是非常不方便的。
- 在python中，最基本的数据结构是**列表(List)**。
- 列表可以看成是在python变量基础上的扩展。
- 还记得python的变量吗：不直接存储数据，而是存放数据的引用（reference），相当于一个指向数据的标签

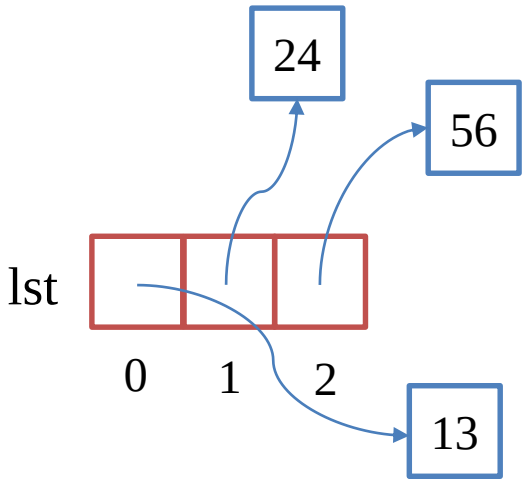


python变量图示

一、列表（3）

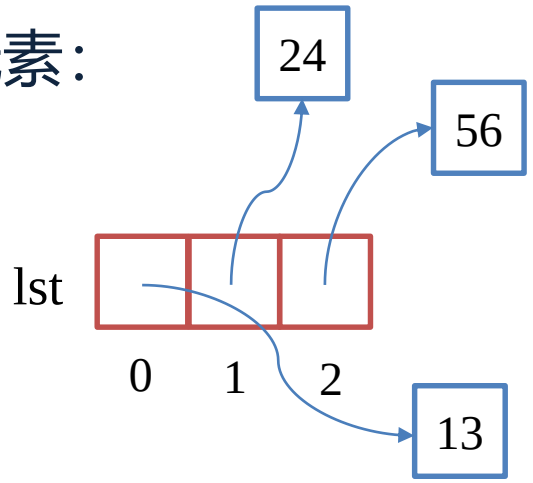
- 列表可以看成是把一组数据的引用（标签）依次存放在一起。
  - 引用（标签）存放在了一起
  - 数据并没有存放在一起!
- 列表中的每个引用（标签）称为一个**元素（element）**。
- 下例中定义了一个名为lst的列表，其中包含三个元素（标签），分别指向数据13，24和56。

```
lst=[13,24,56]
```



1.1 元素与下标

- 列表中的每个引用（标签）称为一个元素（element）。
  - 我们经常将列表中每个引用所指向的数据也称为元素。
  - 在大多数情况下我们不严格区分引用和其指向的数据。
- 元素在列表中的位置或者序号称为下标（index）。
  - 在大多数程序设计语言中，数组或者列表的下标从0开始，python也是如此。
- 使用列表名和下标可以唯一的确定一个元素：
  - 在本例中，lst[0]就是13
  - lst[2]就是56

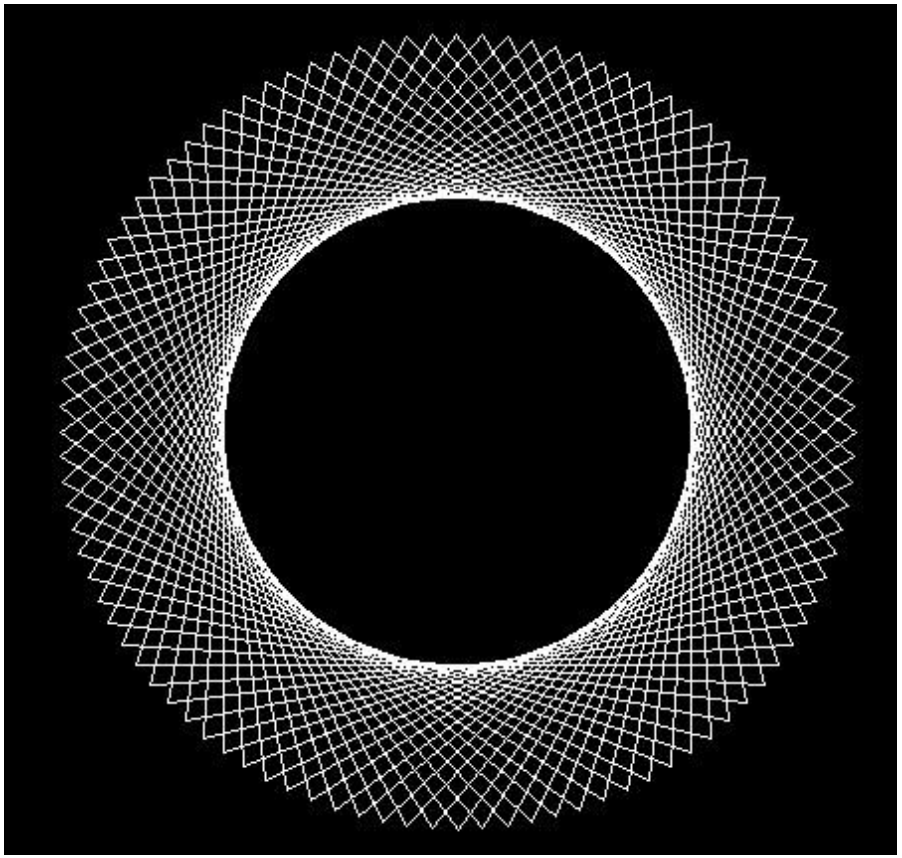


1.2 列表基本操作

操作	说明
`lst=[]`	*建立一个空列表*
`lst=[12,57,42,36,81]`	*建立一个包含5个元素的列表*
`lst=[12]*5`	*建立一个包含5个12的列表*
`lst.append(23)`	*向列表末尾放入新元素23*
`len(lst)`	*了解列表中元素的数量*
`x=lst[4]`	*获取列表中下标为4的元素*
`lst[4]=76`	*将列表中下标为4的元素赋为76（指向数据76）*
`print(lst)`	*输出整个列表的内容*
`del lst[2]`	删除列表中下标为2的元素，列表中后面的元素前移。
`lst.insert(0,23)`	*在列表的开头插入新元素23*
`lst.clear()`	*删除列表中所有元素*
`lst.pop()`	返回并删除列表的最后一个元素

1.3 列表应用举例

- 例1-3-1：绘制包络圆



1.3.1 包络圆（2）

- 运行1-3-1.包络圆.py，仔细观察，可知其绘制过程是这样的：
  - 预先确定一个圆（定位圆）的圆心和半径
  - 在定位圆的圆周上找出100个点，这些点将定位圆的圆周均分为100段。
  - 用线段依次连接第1个点和第31个点、第2个点和第32个点、……
  - 就得到了我们想要的图形。

1.3.1 包络圆（3）

- 程序可以这样实现：
  - 首先，计算出100个圆周点的坐标，保存在列表lst_x和lst_y中。如何计算？
  - 迭代访问这100个圆周点，每次用线段连接当前点（第i个点）和对应的点（第i+30个点）。
  - 第90个点对应的点的下标应该是多少？如何计算其下标？

1.3.1 包络圆（4）

- 程序设计分析:
  - 输入：无
  - 初始条件：定位圆圆心位置center_x,center_y，定位圆半径radius，圆周等分点个数n，连接点间隔数m
  - 输出：包络圆图形
  - 步骤:
    1. 初始化绘图环境
    2. 计算各等分点坐标：根据圆心和半径，利用圆心角，确定各等分点坐标(x,y)，依次分别保存到列表lst_x和lst_y中
    3. 绘制包络圆：依次绘制组成包络圆的各条线段
    - ✓ 显然，第2、3两步都包含重复的动作，各自可以用一个循环来实现

1.3.1 包络圆（5）

- 计算各等分点坐标 循环设计:
  - 用到的变量及初始值：定位圆圆心位置center_x,center_y=0,0，定位圆半径radius=290，圆周等分点个数n=100，各点x坐标列表lst_x=[]，各点y坐标列表lst_y=[]，圆等分角step=360/n，当前等分点坐标x,y，当前等分点对应圆心角angle
  - 循环处理步骤（循环体）：
    1. 求当前等分点对应圆心角angle
    2. 根据angle和radius，计算等分点坐标x,y
    3. 将x,y分别添加到列表lst_x和lst_y的末尾
  - 循环条件：循环n次

1.3.1 包络圆（6）

- 绘制各线段 循环设计:
  - 用到的变量及初始值：圆周等分点个数n=100，各点x坐标列表lst_x=[]，各点y坐标列表lst_y=[]，连接点间隔数m
  - 循环处理步骤（循环体）：
    1. 求点i对应的间隔点下标j：如果j大于等于n，该怎么办?
    2. 从lst_x和lst_y中取出下标为i和j的点坐标
    3. 使用line()函数绘制线段
  - 循环条件：循环n次

1.3.1 包络圆（7）

```
import math

from easygraphics import *

init_graph(800,600)
#将0,0从屏幕左上角移动到屏幕正中
translate(400,300)
#翻转y轴方向使其朝上
set_flip_y(True)
set_render_mode(RenderMode.RENDER_MANUAL)
set_background_color("black")
set_color("white")
```

1-3-1.包络圆.py(1)

初始化绘图环境

1.3.1 包络圆（7）

```
center_x,center_y=0,0
radius = 290
n=100
m=30
step = math.pi * 2 / n
lst_x = []
lst_y = []
for i in range(n):
    angle = i * step
    x = center_x+radius * math.cos(angle)
    y = center_y+radius * math.sin(angle)
    lst_x.append(x)
    lst_y.append(y)
```

计算各等分点坐标

1-3-1.包络圆.py(2)

1.3.1 包络圆（7）

```
for i in range(n):
    j=(i+m) % n
    x1=lst_x[i]
    y1=lst_y[i]
    x2=lst_x[j]
    y2=lst_y[j]
    line(x1,y1,x2,y2)
    delay(100)

pause()
close_graph()
```

1-3-1.包络圆.py(3)

绘制包络圆各线段

1.3.2 约瑟夫问题

- 例1-3-2: 约瑟夫是一个聪明的犹太数学家。在罗马人占领乔塔帕特后，他和他的好朋友和其他39个犹太人一起逃到了一个山洞中。大家决定宁死也不要被罗马人抓到，于是决定了一个自杀方式，41个人排成一个圆圈，由第一个人开始报数，每报到第三个人，这个人就自杀，然后由下一个人重新开始报数，……直到所有人都自杀身亡为止。
- 但是，约瑟夫和他的朋友并不想自杀，于是，他快速计算出了最后一个和倒数第二个自杀的位置，并和他的朋友占到了相应的位置，从而成功逃脱了死亡。
- 请问，约瑟夫和他的朋友到底站到了哪个位置呢?

1.3.2 约瑟夫问题（2）

- 自然的想法是用1到41对每个人编号，然后用循环模拟这个过程：
  - 每一次循环模拟一个人报数
- 但是，随着报数的进展，后面报数时必须跳过已经死亡的人，这会导致循环体内的处理复杂化，因此，我们可以改为这样循环模拟：
  - 用1到41对每个位置编号
  - 每次循环，尝试让这个位置上的人报数（如果这个位置上的人已死，就跳过（continue)这次循环）
  - 每当报数的人满3个，就让当前报数的人自杀
  - 为了便于判断各个人是否已自杀，我们引入一个列表alive，alive[i]为True表示第i个位置上的人活着，为False表示第i个位置上的人已死

1.3.2 约瑟夫问题（3）

- 程序设计分析：
  - 输入：无
  - 初始条件：总人数n=41
  - 输出：死亡序列dead_lst 内容为3、6、9、……
  - 分析：
    - ✓使用循环模拟报数过程，直到所有人都死亡为止

1.3.2 约瑟夫问题（4）

- 模拟报数循环分析：
  - 用到的变量及其初始值：n=41，alive=[True]*(n+1)，当前报数序号i=0，当前已报数人数count=0，死亡序列dead_lst=[]
  - 循环处理步骤（循环体）：
    - 计算当前报数序号i（等于上次序号加1.如果大于n，则从1开始)
    - 判断第i个人是否已死(not alive[i]为True)，是则跳过本次循环(continue)
    - 已报数人数count加一
    - 如果count == 3，则当前人自杀: alive[i]=False，将当前人的序号加入死亡序列：dead_lst.append(i)，重置count=0
  - 循环条件：一直循环，直到死亡序列dead_lst中的人数等于n。适合用while循环。

1.3.2 约瑟夫问题（5）

```
n=41
alive=[True]*(n+1)
dead_lst=[]
count = 0
i=0
while len(dead_lst)<n:
    i+=1
    if i>n:
        i=1
    if not alive[i]:
        continue
    count+=1
    if count == 3:
        alive[i]=False
        dead_lst.append(i)
        count = 0
```

1-3-2.约瑟夫问题.py

数据结构与算法入门

一. 列表

二. 使用对象自定义数据类型

三. 文件输入输出

四. 迭代访问及其应用

五. 查找与排序

六. 字典与集合

二、使用对象自定义数据类型

- 假设我们要编写一个学生成绩管理系统。对于每一个学生，我们都需要记录他的学号、姓名、班级、成绩等信息。用什么数据结构合适呢?
- 如果能将属于同一个学生的信息，放到同一个数据单元中，是最符合逻辑，也最便于进行删除、排序等操作的。
- 在绝大多数程序设计语言中，都提供了定义由多个数据项组成一个数据单元（自定义数据类型）的功能。
- 在python中，更进一步，不但可以将属于同一实体的多个数据项放在一个数据单元中，还可以把属于这个实体的功能也放在数据单元中。这就构成了对象（object）。

二、使用对象自定义数据类型

- 我们在使用对象来存储和表示现实中的实体信息前（如张三、李四、王五……各位同学的学生信息），必须先让python知道这种对象到底应该包含哪些数据项（属性）。
- 因此，我们必须首先定义这些对象所属的类（class）。

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

以上我们定义了名为Student的类。该类有3个属性，分别是id、name和score

2.1 类和对象的基本用法

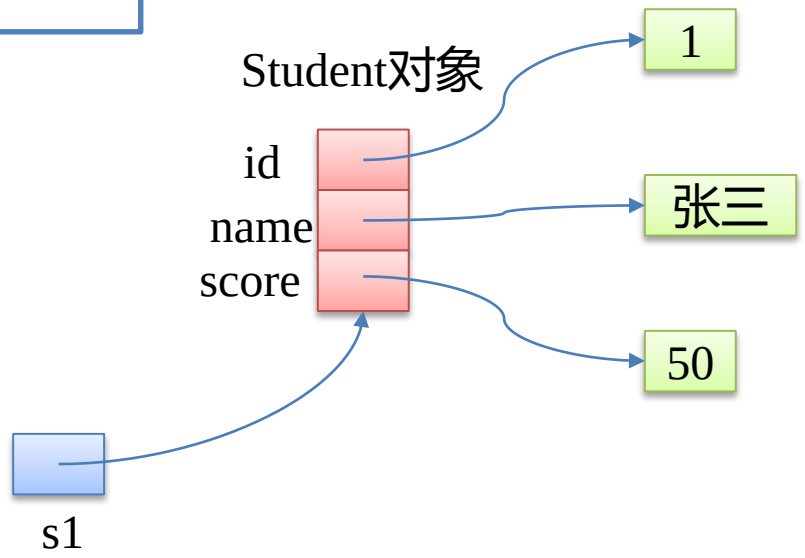
语句	说明
s1=Student(1,'张三',80)	创建一个Student对象，其id为1,name为张三，score为80
s1.score = 70	将s1对象的score属性赋为70
str=s1.name	将s1对象的name属性赋给str变量
s2 = s1	将s1对象赋给s2

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

2.1.1 对象创建与初始化

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

```
s1 = Student(1,'张三',50)
```

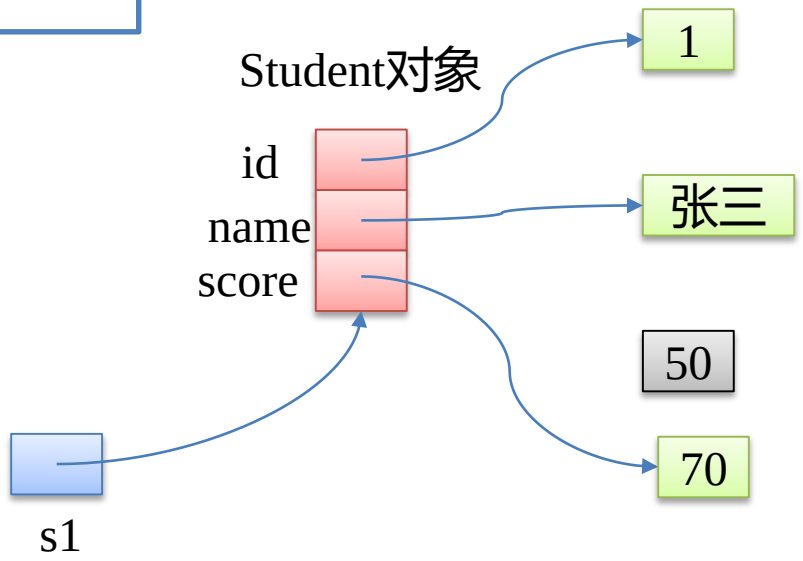


- 以上语句创建了一个新的Student对象，并让变量s1指向该对象。
- 在创建某个类的对象的时候，会自动执行该类的__init__()方法。
- 该方法也称为类的构造方法（constructor）。

2.1.2 对象属性访问

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

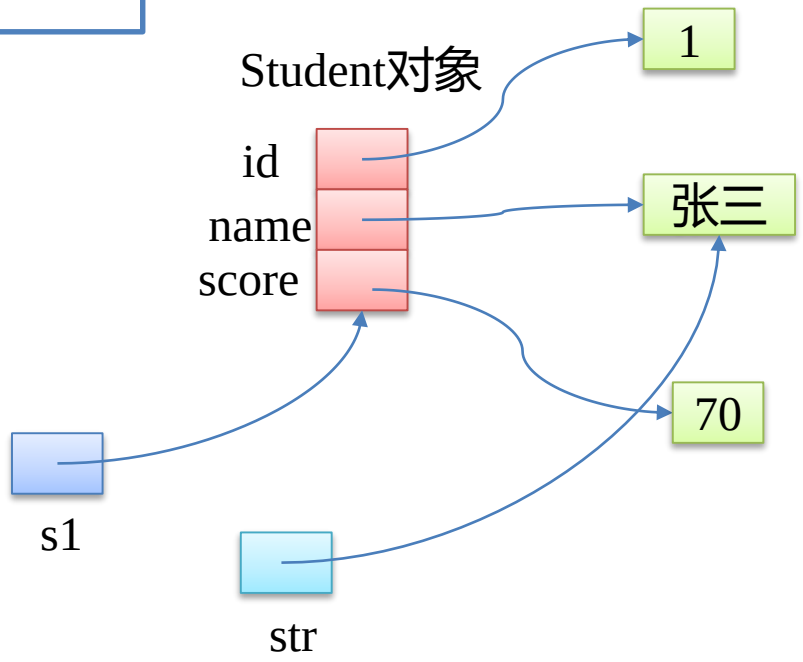
```
s1.score = 70
```



2.1.2 对象属性访问（2）

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

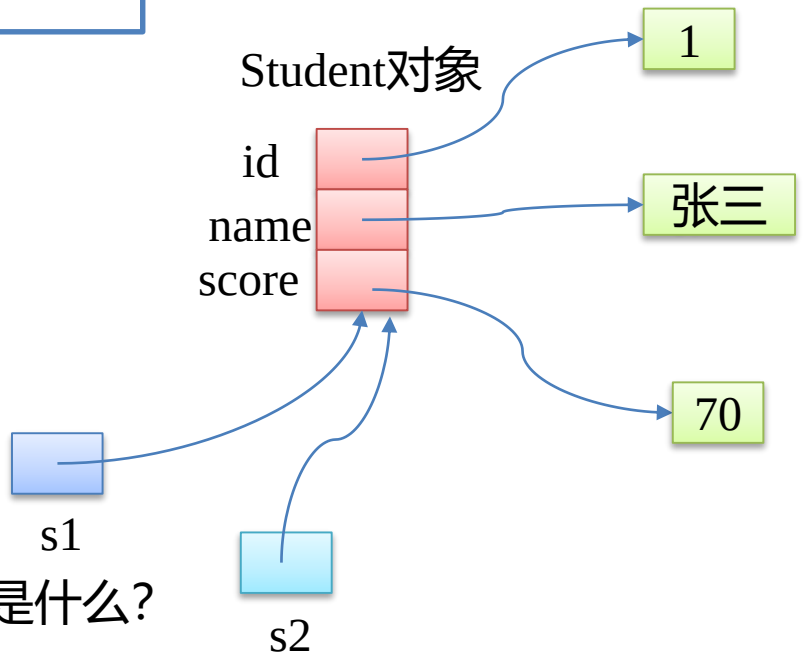
```
str = s1.name
```



2.1.3 对象赋值

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

```
s2 = s1
```



猜一猜：执行下面的语句后，s2.name是什么？

s1.name = '王五'

2.2 可变对象与不可变对象

- 从2.1.3对象赋值中我们可以看到，由于在python中，变量只是指向数据（对象）的标签。所以，如果多个变量指向同一个对象，当对象的属性值发生变化时，通过这些变量访问都会得到变化后的属性值。
- 例如，我们声明一个二维平面上的矢量类，创建对象v1，赋给v2，然后让v1的长度增加1倍：

```
class Vector:
    def __init__(self,x,y):
        self.x = x
        self.y = y
```

```
v1 = Vector(1,1)
v2 = v1
v1.x *= 2
v1.y *= 2
```

我们会发现，v2的长度也同时增加了1倍！
（因为v1和v2实际上是同一个矢量对象）

2.2 可变对象与不可变对象（2）

- 很多时候，我们希望赋值之后，一个变量属性的变化不会影响另外一个变量。这在python中，通过不可变对象（immutable object）来实现。
- 所谓不可变对象，就是初始化后，其属性就不能再被改变的对象。新的值只能通过创建另外一个对象来进行。
- 例如，整数、实数、字符串还有我们之前用过的fractions包中的Fraction类都是不可变对象。
- 我们可以用is运算符来判断两个变量是否指向同一个对象。
- 课时所限，我们不讨论如何定义不可变类。但我们需要搞清楚一个类是否为不可变类。
- 在使用可变类定义对象时，我们必须注意变量之间的影响。

2.2 可变对象与不可变对象（3）

```
from fractions import Fraction

f1=Fraction(1,2)
f2=f1
print(f1,f2)
print(f1 is f2)
print(f1.numerator ,f1.denominator)
# f1.numerator = 3
f3=Fraction(1,3)
f1+=f3
print(f1,f2)
print(f1 is f2)
```

不可变对象示例。
注意被注释掉的f1.numerator =3，该语句运行会报错“AttributeError: can't set attribute”
另外注意两次print(f1 is f2)的结果

2.3 对象列表

- 和普通类型的数据一样，我们可以用列表来保存一组对象。
- 例2-3：读入并显示某便利店某天的所有销售流水信息（每条销售信息包括商品名、单价、购买数量）。
- 数据结构（数据类型）设计：销售信息类，包含三个属性：商品名、单价、购买数量
  - 多条销售信息存在一个列表sales中

```
class Sale:
    def __init__(self,name,price,quantity):
        self.name = name
        self.price = price
        self.quantity = quantity
```

2.3 对象列表（2）

- 分析（续）：
  - 输入：多条销售信息，保存到sale_lst中
    - 如何读入多条信息？用循环，每次读入一条信息。
    - 如何读入一条信息？一条信息包含多项，该如何读入？
    - 怎么知道已经读完所有信息？（何时结束读入）
  - 输出：流水信息。用何种方式输出？
    - 显然，也用循环，每次循环输出一条信息
  - 在本例中我们采用已学的方法中最简单的来进行输入输出：
    - 输入：input()函数，每个数据项一次input()。
    - 输出：使用print()。每条信息一行，每个数据项用制表符（\t）隔开。

2.3 对象列表（3）

- 输入循环设计：
  - 用到的变量及其初始值：sales=[]
  - 循环处理步骤：
    1. 读入商品名称
    2. 读入单价
    3. 读入购买数量
    4. 创建销售信息对象，加入sales列表中。
  - 循环结束条件：一直循环，直到读入的商品名为空为止。用带break的white True循环最合适。

2.3 对象列表（4）

- 输出循环设计：
  - 用到的变量：sales列表
  - 循环处理步骤：
    1. 输出一条销售信息
  - 循环结束条件：sales列表中有几条销售信息（有几个对象），就循环几次。用for循环最合适。
    - ✓ range()函数的参数应该是多少？

```
from decimal import Decimal

class Sale:
    def __init__(self,name,price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

sales = []
while True:
    name = input("请输入商品名称:")
    if name == "":
        break
    price = Decimal(input("请输入商品价格:"))
    quantity = int(input("请输入购买数量:"))
    sale = Sale(name,price, quantity)
    sales.append(sale)
```

2-3.销售信息输入输出.py(1)

说明：因为价格涉及到钱，为了避免二进制浮点数的舍入误差，我们使用Decimal（十进制浮点数）对象来保存价格

2.3 对象列表（6）

```
print("名称\t单价\t数量")
for i in range(len(sales)):
    sale = sales[i]
    print(f"{sale.name}\t{sale.price}\t{sale.quantity}")
```

2-3.销售信息输入输出.py(2)

注意：len(sales)的作用

数据结构与算法入门

一. 列表

二. 使用对象自定义数据类型

三. 文件输入输出

四. 迭代访问及其应用

五. 查找与排序

六. 字典与集合

三、文件输入输出

- 上例中，我们在程序中用input()函数获取用户手工输入的销售信息。
  - 对程序作者而言，是用了最简单的方式
  - 对于用户而言，并不好用
- 更常见的作法，是提供给用户一套完整的录入、查询和修改的图形界面。
- 折中的办法是，用户预先用系统自带的编辑器把数据编辑好，保存成文件（file）。程序运行时从指定的文件中读取数据。
- 本节简要介绍如何从文件中读取和写入数据。

3.1 获取文件名

- 程序要读取文件，首先必须知道要读取的是哪个文件。
- 最简单的方式是直接把文件名作为字面量（前提条件）写在程序中。下例在变量filename中保存了文件路径（D盘的test.txt文件）：
  - filename = "d:\\test.txt"
- 在easygraphics.dialog包中提供了get_open_file_name()函数，可以打开一个对话框让用户选择，然后返回用户选择的文件路径。

```
import easygraphics.dialog as dlg
filename = dlg.get_open_file_name("选择要打开的文件
",dlg.FileFilter.TxtFiles)
if filename == '':
    print("未选择文件")
    exit(-1)
print(filename)
```

如果用户直接关闭了对话框或者选择了取消（cancle），会返回空字符串。

exit()函数的作用是结束（退出）程序执行。

dlg.FileFilter.TxtFiles参数表示我们要打开的是txt后缀的文本文件

3.1 获取文件名（2）

- 类似的，在easygraphics.dialog包中还有一个get_save_file_name()函数，用于让用户输入要写入的文件路径。

```
import easygraphics.dialog as dlg
filename = dlg.get_save_file_name("选择要打开的文件"
,dlg.FileFilter.TxtFiles)
if filename == '':
    print("未选择文件")
    exit(-1)
print(filename)
```

如果用户直接关闭了对话框或者选择了取消（cancle），会返回空字符串。

exit()函数的作用是结束（退出）程序执行。

dlg.FileFilter.TxtFiles参数表示我们要打开的是txt后缀的文本文件

3.2 读写文本文件

- 计算机中的数据文件可以分为两大类：
  - 文本文件
  - 二进制文件
- 在文本文件中，所有的信息都以其字符形式来保存：
  - 例如，我们要在文本文件中保存数字315，实际保存的是'3'、'1'和'5'的字符编码。如果用ASCII码，就是51、49、53。
- 用非文本格式来保存的文件就是二进制文件。
- 因为文本文件中，一切都是字符，因此我们可以直接用文字编辑软件（如记事本、Word等）来打开和编辑文本文件。
- 二进制文件比文本文件更紧凑（节省空间），而文本文件更方便用户处理。
  - 由于现在存储设备很便宜，所以大家更倾向于用文本文件（或者压缩过的文本文件）来保存各种数据信息

3.2 读写文本文件（2）

- 文件的读写其实就是三步：
  1. 打开要读（或者写）的文件
  2. 读取或者写入打开的文件
  3. 关闭打开的文件
- 我们通过例子来看文本文件的读写
- 例3-2：从文件中：
  - 读入显示某便利店某天的所有销售流水信息（每条销售信息包括商品名、单价、购买数量）。
    - ✓ 每条销售信息三行，每个数据项各一行
  - 将销售信息写入另一个文件，每行一条销售记录，行内各数据项之间以制表符(\t)分隔

3.2.1 读取文本文件（1）

```
# 读取文件
filename = dlg.get_open_file_name("选择要打开的文件"
,dlg.FileFilter.TxtFiles)
if filename == '':
    print("未选择文件")
    exit(-1)

sales = []
file=open(filename,mode="r",encoding="UTF-8")
while True:
    name = file.readline().strip()
    if name == "":
        break
    price = Decimal(file.readline().strip())
    quantity = int(file.readline().strip())
    sale = Sale(name,price,quantity)
    sales.append(sale)
file.close()
```

3-1.读写文本文件.py （文件读取部分）

3.2.1 读取文本文件（2）

- 关于Open函数的打开模式(mode参数）：
  - "r"表示以只读方式打开文本格式文件
  - "w"表示以覆写方式打开文本格式文件：如果文件不存在，则创建一个新文件；如果文件已存在，则清除原有文件的全部内容，重新写入。
  - 其他模式请自行搜索或者参考python相关文档
- 关于open函数的encoding参数:
  - 说明文件所使用的字符编码
  - 如果是用windows自带的文本编辑器创建的文本文件，缺省的编码是"GBK"，应使用`open(filename,mode="r",encoding="GBK")`来打开
  - Pycharm的左下角会显示当前打开文件所用的字符编码

3.2.1 读取文本文件（3）

- 关于readline()方法：从文件中读取一行并返回。
  - 如果已经到了文件末尾，则返回空字符串("")
  - 返回的内容中包括一行结尾的换行符
- 关于strip()函数：返回去除指定字符串两端的空白字符（包括制表符、换行符等）后得到的新字符串
  - 注意：字符串是不可变对象
- 读取完成后，别忘了用close() 方法关闭文件。
- 最后说明：本例中为了处理方便，未采用规范的逐行读取（一次循环只从文件中读取一行内容）方法。

3.2.2 写入文本文件（1）

```
# 写入文件
filename = dlg.get_save_file_name("要保存到哪个文件"
,dlg.FileFilter.TxtFiles)
if filename == '':
    print("未选择文件")
    exit(-1)


file=open(filename,mode="w",encoding="UTF-8")
for i in range(len(sales)):
    sale = sales[i]
    file.write(f"{sale.name}\t{sale.price}\t{sale.quantity}\n")
file.close()
```

3-1.读写文本文件.py （文件写入部分）

3.2.2 写入文本文件（2）

- write()方法向文本文件中写入内容。
- 注意：write()方法和print()函数不同，缺省不会自动在写入内容的后面加换行符，因此在写程序时必须自行在字符串的末尾放一个"\n"。
- 写完后记得用close()函数关闭文件。

3.3 读写csv格式数据文件

- csv格式（Comma-Separated Values，https://tools.ietf.org/html/rfc4180.html）是一种常用的文本文件格式，用于保存数据。
  - 其最大的优点在于多种数据处理软件，如excel、spss都直接支持这种格式。
- 其格式很简单：
  - 每条数据一行
  - 第一行可以是标题（而非数据）
  - 组成一条完整数据的各数据项之间由逗号隔开。
  - 如果某数据项本身的内容包含逗号，则需要用双引号将其括起来。
- python标准库中的csv包直接提供了读写csv格式文件的功能
- 例：从3-2.销售信息.csv中读取销售信息，然后将其写入另一个csv文件。

```
filename = dlg.get_open_file_name("选择要打开的文件",
dlg.FileFilter.CSVFiles)
if filename == '':
    print("未选择文件")
    exit(-1)
sales = []
with open(filename,mode="r",encoding="UTF-8") as file:
    reader = csv.reader(file)
    next(reader) # 跳过csv第一行（标题行）
    for row in reader:
        name = row[0]
        price = Decimal(row[1])
        quantity = int(row[2])
        sale = Sale(name,price,quantity)
        sales.append(sale)
```

3-2.csv读写.py （读取csv部分）

dlg.FileFilter.CSVFiles参数表示我们要打开的是csv后缀的文件

close()去哪儿了？：with语句会在代码段执行完后，自动关闭打开的文件

csv.reader在打开的文件file基础上，返回一个csv迭代器

用for循环逐个读取csv中的每一行。（读取出的每一行row是一个列表）

3.3.2 写入csv格式数据文件

```
filename = dlg.get_save_file_name("要保存到哪个文件
",dlg.FileFilter.CSVFiles)
if filename == '':
    print("未选择文件")
    exit(-1)

with open(filename,mode="w",encoding="UTF-8") as file:
    file.write(f"商品名称,单价,数量\n")
    for i in range(len(sales)):
        sale = sales[i]

file.write(f"{sale.name},{sale.price},{sale.quantity}\n")
```

3-2.csv读写.py （写入csv部分）

dlg.FileFilter.CSVFiles参数表示我们要打开的是csv后缀的文件

close()去哪儿了？：with语句会在代码段执行完后，自动关闭打开的文件

因为csv文件格式很简单，我们不用csv包的帮助，直接按照其格式逐行写入即可

3.4 更复杂数据的存储

- csv文件适合用于保存数据项是基本类型（数字和字符串）的数据，对于数据项本身也是复杂类型的数据，就不合适了。
- 例如：班级，其学生属性是一个列表
- 这类复杂数据可以用json、xml格式的数据文件保存，或者保存到数据库中。课时所限，有兴趣的同学可以自学。

数据结构与算法入门

一. 列表

二. 使用对象自定义数据类型

三. 文件输入输出

四. 迭代访问及其应用

五. 查找与排序

六. 字典与集合

4.1 迭代访问

- 从之前的例子中我们看到，对于列表而言，最基本的操作就是用循环来逐一访问其中的每一个元素。
- **迭代访问**（Iterate over the list）：依次访问列表、数组等基本数据结构中的每一个元素
- 之前我们在迭代访问列表时，都是通过range()生成0，1，2的整数序列，然后用其作为下标来访问对应的列表元素
- 实际上，在python中可以用更简单的形式来迭代访问列表。

回忆：1.3.1 可迭代对象

- for循环的基本形式就是：

```
for 变量 in 可迭代对象:
    代码段（循环体）
```

- 我们可以念做“用变量依次循环可迭代对象里的每一项”
- 出现在for语句中的变量称为“**循环变量**”
- 其中，**可迭代对象(iteratable object)**是一个存放数据的容器，可以通过迭代（iteration）的形式来访问其中的每一项数据
- 如果一个变量x中存放的是可迭代对象，那我们就可以使用iter()函数来得到它的迭代器（iterator），并使用迭代器来依次访问该可迭代对象中存放的数据。

4.1.1 列表迭代访问

- 因为在python中，列表就是一个可迭代对象，所以可以直接用for循环对其进行迭代：

```
for 元素变量 in 列表:
    代码段（循环体）
```

4.2 列表与统计

- 例4-2-1：用程序从文件4-2-1.score.csv中读取班上所有同学的学号、姓名和成绩，保存在列表scores中，然后找出该班成绩最好的同学并显示（简单起见，不考虑有多位同学并列第一的情况）。
- 分析：该题是典型的求最大值的问题。
- 练习：用文本编辑器打开4-2-1.score.csv文件，找出其中的最高分。你是如何找出来的?
- 一般的做法是这样的：
  - 记住当前找到的最高分。一开始可以把第一位同学的成绩当作最高分
  - 逐个查看每个同学的成绩，如果比之前找到的最高分更高，就把他/她的成绩作为已找到的最高分
  - 全部同学的成绩都看完后，找到的最高分就是所有人里的最高分

4.2.1 找最大值（1）

- 首先，设计相关数据类型，选择合适的数据结构。通过查看4-2-1.score.csv文件，我们知道每条学生信息包含三个数据项姓名、学号和成绩。据此我们设计Score类型如下：

```
class Score:
    def __init__(self,id,name,score):
        self.id = id
        self.name = name
        self.score = score
```

Score类定义

4.2.1 找最大值（2）

- 程序分析：
  - 用到的数据类型：Score类型，包含三个属性姓名、学号和成绩。Score列表scores。
  - 输入：从csv文件中读入，保存到scores列表中。
  - 输出：找到的最高分同学信息max_score。
  - 主要处理步骤：
    - ✓ 使用for循环迭代访问scores列表，找出成绩最高的同学
  - 在这里我们顺便复习一下函数的内容：把找最高分的功能做成一个函数find_max_score()，该函数从所给的参数scores列表中找出最高分返回。

4.2.1 找最大值（3）

- 循环分析：
  - 循环用到的变量及初值：scores列表，找到的最高分同学信息 max_score=scores[0]
  - 循环处理步骤：
    - ✓ 如果当前同学成绩score.score > 已找到的最高成绩 max_score.score，则更新已找到的最高成绩max_score = score
  - 循环条件：用score变量迭代访问scores列表中的每个元素。

4.2.1 找最大值（4）

```
def find_max_score(scores):
    """
    寻找并返回scores列表中成绩最高的学生信息

    :param scores: scores列表
    :return: 最高分同学信息
    """
    max_score = scores[0]
    for score in scores:
        if score.score > max_score.score:
            max_score = score
    return max_score
```

4-2-1.找最高分.py (find_max_score函数定义)

4.2.1 找最大值（5）

```
import easygraphics.dialog as dlg
import csv
from decimal import Decimal
```

4-2-1.找最高分.py (开头import部分)

```
# 读取文件
filename = dlg.get_open_file_name("选择要打开的文件",
dlg.FileFilter.CSVFiles)
if filename == '':
    print("未选择文件")
    exit(-1)
scores = read_csv_file(filename)
dlg.show_objects(scores)
max_score=find_max_score(scores)
dlg.show_message(f"获得最高分的同学：{max_score.id}
{max_score.name},成绩为{max_score.score}")
```

4-2-1.找最高分.py (主程序部分)

dlg.show_objects()函数以表格形式显示对象列表

4.2.1 找最大值（6）

```
def read_csv_file(filename):
    """
    读取指定的csv文件，保存并返回scores列表
    :param filename: 要读取的csv文件名
    :return: scores列表
    """
    scores = []
    with open(filename, mode="r", encoding="UTF-8") as file:
        reader = csv.reader(file)
        next(reader)  # 跳过csv第一行（标题行）
        for row in reader:
            id = row[0]
            name = row[1]
            score = Decimal(row[2])
            score = Score(id,name,score)
            scores.append(score)
    return scores
```

4-2-1.找最高分.py (read_csv_file函数定义)

4.2.2 求和（1）

- 例4-2-2：用程序从文件4-2-2.销售信息.csv中读取某超市某时间段内的所有销售信息（包括商品名、单价和数量）。保存在列表sales中。计算并显示该超市在该时间段内的总销售额。
- 求和（和求平均值）是统计计算中最基本的操作。
- 原理也很简单：通过迭代访问，把所有元素的指定属性逐一累加起来即可。
- 首先，数据类型设计和数据结构选择通过查看4-2-2.销售信息.csv文件，我们知道每条销售信息包含三个数据项名称、单价和数量。据此我们设计Sale类型如下：

```
class Sale:
    def __init__(self,name,price,quantity):
        self.name = name
        self.price = price
        self.quantity = quantity
```

Sale类定义

4.2.2 求和（2）

- 程序分析：
  - 数据结构（类型）：销售信息Sale类，sales列表
  - 读入：从csv文件中读入
  - 输出：销售总额total
  - 核心处理步骤：
    - 迭代访问sales列表，累加计算销售总额total
    - 本例中需要额外的处理的是，在销售信息中没有直接提供销售金额属性，需要计算得到:
      - 销售金额=单价*数量

4.2.2 求和（3）

- 迭代访问（循环）分析：
  - 循环中用到的变量及初始值：sales，total=0
  - 循环处理步骤:
    - ✓计算本条记录的销售额: amount = s.price * s.quantity
    - ✓累计销售额：total += amount
  - 循环条件：用变量s迭代访问sales中每个元素

4.2.2 求和（4）

```
def calc_sales_total(sales):
    """
    计算销售总额

    :param sales: 销售记录列表
    :return: 销售总额
    """
    total = 0
    for s in sales:
        amount = s.price * s.quantity
        total += amount
    return total
```

4-2-2.求销售总额.py (计算销售总额函数)

4.2.3 分段计数（1）

- 例4-2-3：用程序从文件4-2-1.score.csv中读取班上所有同学的学号、姓名和成绩，保存在列表scores中。统计0-10（不含）、10-20（不含）、……、80-90（不含）、90-100（不含）各分数段的人数（不考虑考100分的情况）。
- 首先，数据类型设计和数据结构选择。通过查看4-2-1.score.csv文件，我们知道每条学生信息包含三个数据项姓名、学号和成绩。据此我们设计Score类型如下：

```
class Score:
    def __init__(self,id,name,score):
        self.id = id
        self.name = name
        self.score = score
```

Score类定义

4.2.3 分段计数（2）

- 然后思考：如何统计0-10分数段的人数?
  - 用num0表示该分数段的人数。初值为0
  - 迭代访问scores列表，如果当前元素的分数小于10，num0就累加1，否则继续下一次循环。（每找到一个满足条件的num0就加一）
  - 可以在for循环内用if来进行条件判断和相应的处理
- 显然，更多的分数段，我们可以加入变量num1、num2、……、num9，以及相应if语句来进行判断和处理。

4.2.3 分段计数（3）

- 既然num0、num1……num9的作用类似，我们可以把它们放到一个列表nums中：
  - nums[0]记录0-10分数段人数
  - nums[1]记录10-20分数段人数
  - nums[2]记录20-30分数段人数
  - ……
  - nums[9]记录90-100分数段人数
  - 看看，nums中元素的下标，和对应的分数段之间是否存在规律？

4.2.3 分段计数（4）

```
def count_scores(scores):
    """
    分段计数

    :param scores: scores列表
    :return: 分段计数结果列表
    """
    nums = [0]*10
    for s in scores:
        if 0<=s.score<10:
            nums[0]+=1
        elif 10<=s.score<20:
            nums[1]+=1
        ……

        elif 90<=s.score<100:
            nums[9]+=1
    return nums
```

基本做法

nums = [0]*10的含义是，
创建一个包含10个0的列表，
赋给nums变量

4.2.3 分段计数（5）

```
def count_scores(scores):
    nums = [0]*10
    for s in scores:
        c=int(s.score // 10)
        if 0==c:
            nums[c]+=1
        elif 1==c:
            nums[c]+=1
        elif 2==c:
            nums[c]+=1
         ……
        elif 9==c:
            nums[c]+=1
    return nums
```

引入中间变量c

4.2.3 分段计数（6）

```
def count_scores(scores):
    """
    分段计数

    :param scores: scores列表
    :return: 分段计数结果列表
    """
    nums = [0]*10
    for s in scores:
        c=int(s.score // 10)
        nums[c]+=1
    return nums
```

进一步简化

4-2-3.分段计数.py(计数部分)

寻找规律以简化处理是程序员的本能……

4.3 分类统计（1）

- 在例4-2-2中，我们计算了商店所有商品的销售总额。
- 在现实中，管理者不仅需要知道总的销售额，也需要关心某种特定商品的销售情况。
- 例4-3-1：已知文件4-3.sales.csv中保存着某商店当天的所有销售记录。请编程读入所有销售记录保存到sales列表中，计算并显示方便面的销售总额。
- 首先，数据类型设计和数据结构选择。通过查看4-3.sales.csv文件，我们知道每条销售信息包含三个数据项商品名称、单价和数量。据此我们设计Sale类型如下：

```
class Sale:
    def __init__(self,name,price,quantity):
        self.name = name
        self.price = price
        self.quantity = quantity
```

Sale类定义

4.3 分类统计（2）

- 程序分析:
  - 数据结构：销售记录Sale类，sales列表
  - 输入：4-3.sales.csv文件，读入到sales列表中
  - 输出：方便面的销售总额total
  - 基本思路和例4-2-2一致,只是在迭代访问时,需要判断当前销售记录对应的商品是不是"方便面",只有是的时候才累计,不是就不累计.

4.3 分类统计（3）

- 循环分析:
  - 循环中用到的变量与初始值：sales、销售总额total=0，要统计的商品名称goods_name
  - 循环处理步骤:
    1. 如果当前销售记录的商品名称s.name等于goods_name，则:
       1. 计算销售额amount=s.price * s.quantity
       2. 累计销售总额total+=amount
  - 循环条件：用变量s迭代访问列表sales

4.3 分类统计（4）

```
def calc_sub_total(sales, goods_name):
    """
    统计指定商品的销售额
    :param sales: 销售记录列表
    :param goods_name: 要统计的商品名
    :return: 指定商品的销售额
    """
    total = 0
    for s in sales:
        if s.name == goods_name:
            amount = s.price * s.quantity
            total += amount
    return total
```

4-3-1.求方便面销售额.py(统计部分)

4.4 筛选子集（1）

- 例4-4：文件4-4.浦发银行.csv中保存浦发银行股票从2014年到2019年2月的信息。请编程将所有信息读入保存到stocks列表中，然后找出并显示所有2014年的信息
- 这是个典型的筛选（filter）问题：从已知集合寻找所有满足特定条件的元素，组成一个新的集合
  - 注：一般程序设计或者数据结构中所说的集合（Collection）不是值数学意义上的集合（Set），其中的元素是可以重复的。
- 首先，数据类型射界和数据结构选择。我们查看4-4.浦发银行.csv文件的内容：
  - 包含“日期,股票代码,名称,收盘价,最高价,最低价,开盘价,前收盘,涨跌额,涨跌幅,换手率,成交量,成交金额,总市值,流通市值”等字段。
  - 其中股票代码和名称字段的内容是重复的，没必要保留
- 由此可以设计出相应的Stock类型：

4.4 筛选子集（2）

```
class Stock:
    def __init__(self, pdate, pclose, high, low, popen,
last_price, change, change_percent, turnover_rate, volume,
amount, cap, tradable_cap):
        self.pdate = pdate
        self.pclose = pclose
        self.high = high
        self.low = low
        self.popen = popen
        self.last_price = last_price
        self.change = change
        self.change_percent = change_percent
        self.turnover_rate = turnover_rate
        self.volume = volume
        self.amount = amount
        self.cap = cap
        self.tradable_cap = tradable_cap
```

Stock类定义

4.4 筛选子集（3）

- 程序分析：
  - 数据结构：Stock类、stocks列表
  - 输入：从csv文件中读入股票信息到stocks列表
  - 输出：用easygraphics.dialog中的show_objects()函数显示筛选结果filtered列表
  - 处理方法：迭代访问stocks中的所有元素，查看其是否满足条件（年份等于2014），如果满足，就加入filtered列表。
- 知识点:日期处理
  - python标准库的datetime包中提供了date类，可以进行相关的日期处理。

```
from datetime import date

pdate = date.fromisoformat(row[0])

if pdate.year == 2014:
```

导入date类

将'2014-2-14'形式的字符串转换为对应的日期

判断日期中的年份是否为2014

4.4 筛选子集（4）

- python还提供了其他的日期和时间类，相关使用方法直接在搜索引擎中搜索“python 字符串转日期”和“python 日期处理”即可
- 循环处理:
  - 循环中用到的变量和初始值：stocks列表，筛选结果列表filtered=[]，筛选年份year
  - 循环步骤:
    - 如果当前股票信息s的日期年份等于给定的筛选年份year，则将其加入filtered数组
  - 循环条件：用变量s迭代访问stocks列表

4.4 筛选子集（5）

```
def filter_by_year(stocks,year):
    """
    筛选出指定年份的股票信息

    :param stocks: 股票信息列表
    :param year: 年份
    :return: 指定年份的股票信息列表
    """
    result = []
    for s in stocks:
        if s.pdate.year == year:
            result.append(s)
    return result
```

4-4.股票筛选.py(筛选函数)

4.4 筛选子集（6）

```
props =
'pdate,pclose,high,low,popen,last_price,change,change_percent,turn
over_rate,volume,amount,cap,tradable_cap'.split(',')
```

s.split(sep)方法的作用：以sep为分隔符，将字符串s分割成多个字符串。返回由这些字符串组成的列表。
上例中的语句运行完后，props变量对应的值就是列表['pdate','pclose',….,'tradeable_cap']

```
dlg.show_objects(stocks,fields=props,field_names=prop_names)
```

show_objects()函数中fields参数和field_names参数的作用:
fields参数: 显示对象的哪些属性（以及显示的顺序）
field_names: 表格标题中各对象属性的名称

数据结构与算法入门

一. 列表

二. 使用对象自定义数据类型

三. 文件输入输出

四. 迭代访问及其应用

五. 查找与排序

六. 字典与集合

5.1 查找

- 例5-1-1：已知文件5-1.scores.csv中保存着班上所有同学的学号、姓名和成绩信息。请编程实现：读入所有同学的信息，并根据用户输入的学号x查询其姓名和成绩。
- 这种在某个数据结构中寻找满足特定条件的元素的操作，就是**查找**（search）。
  - 最基本的查找条件形式就是：元素的特定属性和给定值相等
  - 用来查找和比较的给定值就叫做**关键字**（key）
- 最直接的查找方法就是，使用迭代访问来进行查找，即：
  - 用循环将s中的每一个元素的学号和x进行比较，看看是否相等。
- 这种通过迭代访问每个元素来查找的方法称为**顺序查找**（sequential search）。

5.1.1 顺序查找（1）

- 数据类型设计和数据结构选择。查看数据文件内容，设计相关数据结构(略)
- 程序分析：相关数据结构、输入、输出、主要处理方法（略）

```
class Score:
    def __init__(self, id, name, score):
        self.id = id
        self.name = name
        self.score = score
```

Score类型定义

5.1.1 顺序查找（1）

- 我们把按照id在学生信息列表中进行查找做成一个函数:
  - 输入参数：待查找的学生信息列表，待查找的id
  - 输出：找到的学生信息在列表中的下标。如果没找到，返回-1
  - 处理方法：迭代访问列表中每个元素，检查其id是否和待查找的id相等。如果相等，就返回其下标。如果迭代访问全部结束都未找到，则返回-1
- 迭代访问查找循环分析：循环中用到的变量及其初值、循环步骤、循环条件（略）
  - 注意：因为我们要返回元素的下标，因此不能直接用元素形式（因为这种形式下得不到元素的下标），而要用下标形式进行迭代访问

5.1.1 顺序查找（2）

```
def find_by_id(scores, id_key):
    """
    在scores列表中查找id等于id_key的元素

    :param scores: scores列表
    :param id_key: 要查找的id关键字
    :return: 找到的元素在scores列表中的下标。如果没有找到，返回-1
    """
    for i in range(len(scores)):
        s = scores[i]
        if s.id == id_key:
            return i
    return -1
```

5-1-1.顺序查找.py(按id查找函数)

注意：本例中使用下标进行迭代访问

5.1.1 顺序查找（3）

```
id = int(dlg.get_string("请输入要查找的id"))
i = find_by_id(scores,id)
if i==-1:
    dlg.show_message(f"找不到id为{id}的学生")
else:
    s=scores[i]
    dlg.show_message(f"id: {s.id} 姓名: {s.name} 成绩
: {s.score}")
```

5-1-1.顺序查找.py(主程序部分)

注意：本例中Score类的id属性（见本例源程序中的read_csv()函数，ppt上略）和id变量的类型都是整数（后面的例子中会用到）

因为有找不到的可能（列表中无此元素），在完成查找后，要判断是否找到了。

5.1.1 顺序查找(4)

- 思考：
  - 如果班上没有同学的学号为x，程序一共会比较多少次？
  - 如果班上有同学的学号为x，程序最多会比较多少次？
- 当数据结构中数据量很大时，顺序查找是一个非常低效的算法。
- 例如，警察需要验证身份证的真假。即根据身份证号码，查找对应的人员信息。
  - 全国有多少人？
  - 如果每次比较操作耗时1微秒(百万分之一秒)，而且该身份证号是假的，那么查找最终要花多少时间才能结束？

5.1.2 二分法查找（1）

- 如果数组中的**数据已按照大小顺序排列好**，那么就可以采用**二分查找法**（binary search，也称为折半查找）来进行查找。
- 二分法的原理很简单。看个例子：猜数游戏。
  - 老师在心中想一个1-100之间的整数。请同学用最少的询问次数来找出这个数。
  - 同学是如何找到这个数的?

5.1.2 二分法查找（2）

- 同理，二分查找法的算法如下：
  - 输入：已按照从小到大顺序排好序的列表，待查找的关键字
  - 输出：列表中与关键字相等的元素的下标
  - 步骤：

1. 计算列表最中间位置元素的下标；
2. 比较列表最中间位置元素与关键字，如果两者相等，则查找成功；
3. 否则利用中间位置的下标记录将列表分成前、后两个子列表和中间元素三部分，如果中间元素大于关键字，则进一步查找前一子列表，否则进一步查找后一子列表。
4. 重复以上过程，直到找到满足条件的元素，即查找成功；或直到子列表中不存在元素为止，此时查找不成功。

5.1.2 二分法查找（3）

- 练习：已知列表
  a=[34,48,56,60,72,75,80,87,90,95,100]。
  - 用二分查找法查找56是否在列表中，步骤是怎样的?
  - 用二分查找法查找92是否在列表中，步骤是怎样的?
  - 用二分查找法查找100是否在列表中，步骤是怎样的?

5.1.2 二分法查找（4）

- 例5-1-2：已知文件5-1.scores.csv中保存着班上所有同学的学号、姓名和成绩信息，且所有信息已经按照学号从小到大的顺序排好。请编程实现：读入所有同学的信息，并根据用户输入的学号id，使用二分法查询其姓名和成绩。
- 数据类型设计和数据结构选择：查看数据文件内容，设计相关数据结构(略)
- 程序分析：相关数据结构、输入、输出、主要处理方法（略）

```
class Score:
    def __init__(self, id, name, score):
        self.id = id
        self.name = name
        self.score = score
```

Score类型定义

5.1.2 二分法查找（5）

- 我们把按照id在学生信息列表中进行二分法查找做成一个函数:
  - 输入参数：待查找的学生信息列表，待查找的id
  - 输出：找到的学生信息在列表中的下标。如果没找到，返回-1
  - 处理方法：略
- 注意：因为进行二分法查找时要进行大小比较，所以本例中id参数、Score类的id属性都应该是int类型。在输入处理时要注意。

```
def read_csv(filename):
    ……
        for row in reader:
            id = int(row[0])
            ……
            s=Score(id,name,score)
    ……
```

读入id时要转换为int类型

- 循环分析：
  - 循环中用到的变量：待查找列表scores，要查找的id关键字id_key，在列表中查找范围的开始下标start=0, 结束下标end=len(scores)-1， 中间元素下标mid，中间元素s
  - 循环处理步骤：
    - ✓ 计算中间元素下标mid=(start+end)//2
    - ✓ 获取中间元素s=scores[mid]
    - ✓ 判断中间元素s的id属性和id_key的大小关系:
      - ✦ 相等：找到了，函数返回s的下标：mid
      - ✦ s.id小于id_key：在列表后半部分找：start=mid+1
      - ✦ s.id大于id_key：在列表前半部分找：end = mid -1
  - 循环条件：start和end组成的查找范围中有元素存在。即start<=end
- 循环结束之后，仍未找到，函数返回-1

```python
def find_by_id(scores, id_key):
    """
    用二分法在scores列表中查找id等于id_key的元素

    :param scores: scores列表
    :param id_key: 要查找的id关键字
    :return: 找到的元素在scores列表中的下标。如果没有找到，返回-1
    """
    start,end = 0,len(scores)-1
    while start<=end:
        mid = (start+end)//2
        s=scores[mid]
        if s.id == id_key:
            return mid
        elif s.id < id_key: #查找后一半
            start = mid+1
        else: #查找前一半
            end = mid - 1
    return -1
```

5-1-2.二分法查找.py(二分查找函数)

5.1.2 二分法查找（8）

- 思考：
  - 如果要找的元素不在列表中，那么需要比较几次?
  - 如果输入的列表中元素是按照从大到小顺序排列的，应如何使用二分法查找?
  - 如果输入的列表中元素没有按照顺序排列，会怎样?

5.2 索引（1）

- 如果一个列表中有n个元素，那么平均下来做一次顺序查找需要比较$n/2$次，二分法查找需要$log_2 n$次。在n很大时，二分法的效率要远远高于顺序查找（例如，$log_2 1000000 \approx 20$）。
- 但基本的二分法要求数据是按照要查找的字段完全排好序的，这在很多时候难以实现。（比如，可能既需要对姓名字段进行查找，也需要对学号字段查找。每个字段的排序结果不一样）
- 解决方案是，在原有的列表基础上，增加对应的专用数据结构：**索引（Index）**。
  - 索引是为了加速对数据行的检索而创建的数据结构。
  - 建立索引的检索（查找）索引目标属性称为索引字段（Index Field）。例如，我们建立一个按照学号查询学生成绩用的索引，那么学号就是该索引的索引字段。
- 索引有很多种形式，本节我们只介绍最简单的索引形式。

补充说明：属性和字段

- 对于组成各复杂数据类型的数据项，在不同的方法中有不同称呼：
  - 在面向对象程序设计中，一般称为为**属性**（attribute，python或java等语言）或者**成员变量**（member variable，C++等语言）
  - 在数据库设计或者数据分析中，一般称其为**字段**（Column或者Field）
- 遇到的时候我们知道这几个词的意思基本是相同的即可

5.2 索引（2）

- 例5-2：文件5-2.成绩.csv中保存着某专业所有同学的学号、姓名和考试成绩。
  - 已知：所有学生的学号在1-1000之间，无重复。
  - 请编程读入所有同学的成绩信息，建立对应的索引列表，然后使用索引来查找指定学号的同学信息。
- 首先，数据类型设计和数据结构选择。
  - 根据题目要求，设计Score类型如下：

```
class Score:
    def __init__(self, id, name, score):
        self.id = id
        self.name = name
        self.score = score
```

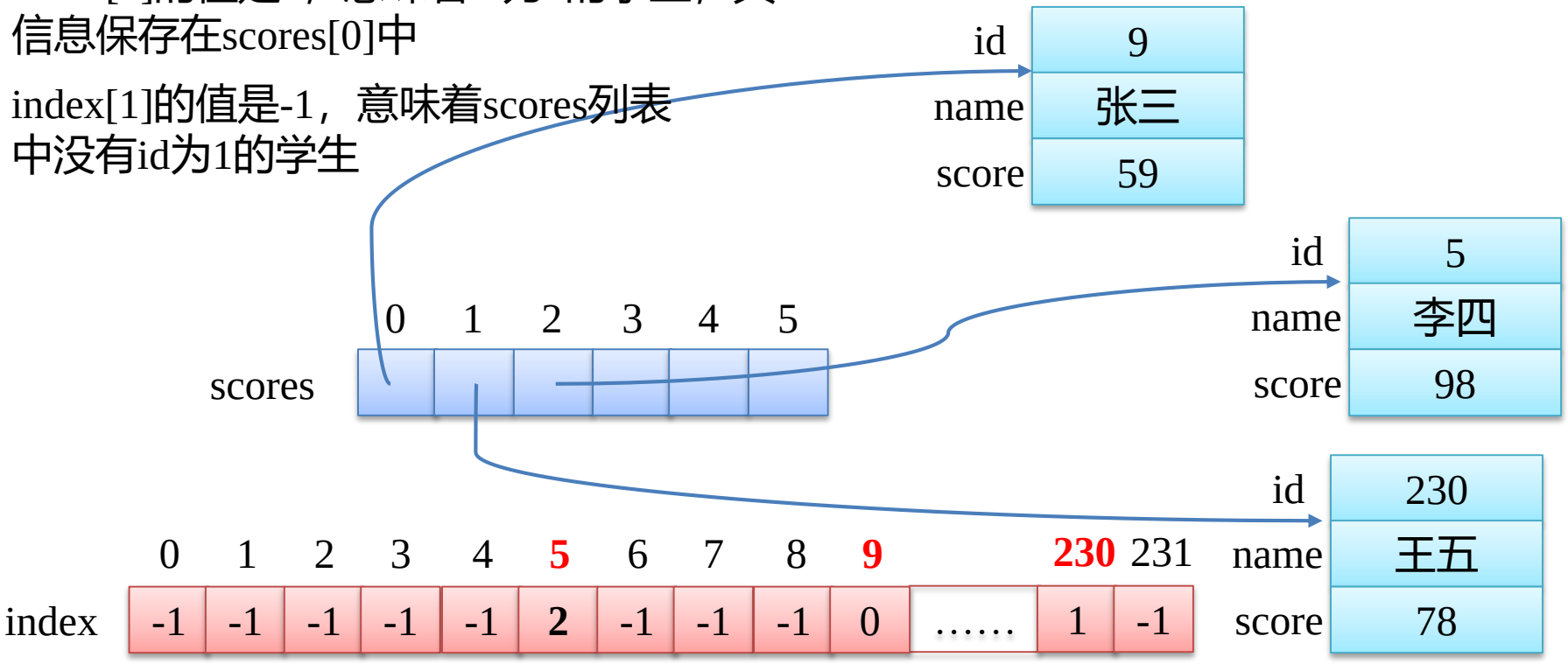
Score类型定义

5.2 索引（3）

- 索引的原理：用空间换时间
  - 直接用成绩对象的成绩做其在索引列表index中的下标
  - index中保存成绩在scores列表中的下标

index[9]的值是0，意味着id为9的学生，其信息保存在scores[0]中

index[1]的值是-1，意味着scores列表中没有id为1的学生



5.2 索引（4）

- 程序分析：
  - 输入分析：略
  - 输出分析：略
  - 处理：
    - ✓根据scores列表，建立对应的学号索引列表index
    - ✓使用index索引列表进行查找

5.2.1 建立索引（1）

- 可以这样建立索引：
  1. 建立一个大小等于学号取值范围中整数个数的列表index，所有元素都初始化为-1
  2. 用下标i迭代访问scores列表：
     1. 取当前学生元素s=scores[i]
     2. 设置对应的索引：索引中下标为当前学生学号(s.id)的元素，其值为该元素在scores列表中的下标（i）：index[s.id]=i
  3. 完成迭代访问后，索引index就建好了

5.2.1 建立索引（2）

```
def create_index(scores):
    """
    创建index索引列表
    :param scores: 要索引的scores列表
    :return: index索引列表
    """
    index = [-1]*(MAX_ID+1)
    for i in range(len(scores)):
        s=scores[i]
        index[s.id] = i
    return index
```

5-2.索引与查找.py （索引建立函数）

注意：因为需要用到元素的下标，所以使用下标形式进行迭代访问

5.2.2 建立索引（2）

```
MAX_ID = 1000
def create_index(scores):
    """
    创建index索引列表
    :param scores: 要索引的scores列表
    :return: index索引列表
    """
    index = [-1]*(MAX_ID+1)
    for i in range(len(scores)):
        s=scores[i]
        index[s.id] = i
    return index
```

5-2.索引与查找.py （索引建立函数）

注意：因为需要用到元素的下标，所以使用下标形式进行迭代访问

5.2.2 使用索引查找

- 有了索引，查找非常简单：直接找索引中学号对应的元素即可：

```
def find_using_index(index,id_key):
    """
    使用index索引列表，查找学号等于id_key在scores列表中的下标
    :param index: 索引列表
    :param id_key: 要查找的学号
    :return: 学生在scores列表中的下标。-1表示列表中无此学生。
    """
    return index[id_key]
```

5-2.索引与查找.py （查找函数）

5.2.3 进一步扩展*

- 在上例中索引的基础上，我们可以进一步扩展:
  - 已知学号取值范围为[a,b]。如果b非常大怎么办?
    - 只要b-a不太大，我们只需要在索引时对学号先做个减a的运算，再索引即可
  - 如果b-a也很大，怎么办?
    - 可以通过某种数学运算，对学号的取值范围进行压缩
  - 对于非整数类型的数据，如字符串，怎么索引?
    - 可以用某种方法先将字符串转换成一个整数，然后再索引。这个转换称为**散列**或者**哈希**（hashing）。转换得到的结果称为哈希值
    - python内置的**hash()**函数可以将计算任意数据的哈希值。

5.2.3 进一步扩展*（2）

- 在上例中索引的基础上，我们可以进一步扩展（续）：
  - 如果哈希或者压缩后的字段值有重复，怎么办?
    - 首先，需要选择一个好的哈希函数，其应该保证不同的数据，哈希后得到相同哈希值的可能性尽量小。（python的hash()函数能做到这一点）
    - 我们可以对索引列表中的元素做改进，使其在同一个哈希值对应的元素中可以存放多个索引。
    - 假设index是学生姓名索引，并且'aaa'和'bbb'的哈希值都是13，那么index[13]中将同时存放姓名为'aaa'和'bbb'的学生在scores数组中的下标。
  - 这种利用哈希建立索引的数据结构，就是哈希表（Hash table）
- 有兴趣的同学自行查看5-2-1.姓名索引.py

5.3 排序

- 例5-3：已知列表lst中保存着班上所有同学的考试分数（只有分数，无其他信息，如[61,78,49,32,95]）。现在希望能够将成绩按照从大到小的顺序排列，该如何处理?
- 本题是一个典型的**排序**(Sort)问题：如何将一组原本无序的数据，按照从大到小（或从小到大）的顺序排列起来?
- 可以利用我们之前学的找数组中最大值的算法来进行排序：
  - 找出lst[0]..lst[n-1]中最大元素，把它放到lst[0]。为了不丢失原来lst[0]的值，将原lst[0]值放到最大元素原来的位置t上，即对换lst[0]和lst[t]。（lst[0]现在是最大的元素了）
  - 找出lst[1]..lst[n-1]中最大元素的下标t，对换lst[0]和lst[t]。（lst[1]现在是第2大的元素了）
  - 依次从2、3...开始重复上述步骤，一直到n-1，我们就可以得到一个排好序数组!

5.3.1 选择排序（1）

- 这就是所谓的**选择排序(Selection sort)。**
- 练习：已知数组lst=[15,20,4,20,76,58,95,32,63,60]，用选择排序法实现从大到小排列的过程是怎样的?
- 程序分析：输入、初始条件和常量、输出、核心处理方法（略）
- 使用迭代访问找lst[start]到lst[end-1]中的最大值：
  - 循环中用到的变量及其初始值：列表lst，开始元素下标start，比最后一个元素下标多一的变量end，最大元素下标max=start
  - 循环处理步骤：
    - ✓ 如果当前元素lst[i]大于之前找到的最大值lst[max]，说明lst[i]是到目前为止最大的元素，应该更新max：max=i
  - 循环条件：i从start循环到end-1，每次加一

5.3.1 选择排序（2）

```
def find_max(lst,start,end):
    """
    找从lst[start] 到 lst[end-1]之中的最大值下标
    :return: 最大值在lst中的下标
    """
    max = start
    for i in range(start,end):
        if lst[i] > lst[max]:
            max = i
    return max
```

5-3-1.选择排序1.py (找最大值下标函数)

5.3.1 选择排序（3）

- 排序循环：
  - 循环中用到的变量及其初始值：列表lst，列表中的元素个数n，从lst[i]到lst[n-1]中的最大值下标t
  - 循环处理步骤：
    - ✓ 找lst[i]到lst[n-1]之中的最大值下标t
    - ✓ 把最大值lst[t]换到前面：交换lst[i]和lst[t]
  - 循环条件：i从0循环到n-1，每次加一

5.3.1 选择排序（4）

```
def select_sort(lst):
    """
    选择排序

    :param lst: 要排序的列表
    """
    n = len(lst)
    for i in range(n):
        t = find_max(lst,i,n)
        lst[i],lst[t]=lst[t],lst[i]
```

5-3-1.选择排序1.py (选择排序函数)

5.3.1 选择排序（5）

```
def select_sort(lst):
    """
    选择排序

    :param lst: 要排序的列表
    """
    n = len(lst)
    for i in range(n):
        t = i
        for j in range(i, n):
            if lst[j] > lst[t]:
                t = j
        lst[i],lst[t]=lst[t],lst[i]
```

找lst[i]到lst[n-1]中的最大值下标t

5-3-1.选择排序2.py (选择排序函数)

将找最大值函数合并进选择排序
形成双重循环

5.3.1 选择排序（6）

- 上例中我们对一个整数列表进行了排序。在实际应用中，我们很少会对单一的整数列表排序。更多的，是对各种对象列表数组，按照某个（或某几个）字段进行排序。
- 例5-3-2：已知文件5-2.成绩.csv中保存着某专业所有同学的学号、姓名和考试成绩。请编写程序读入同学们的成绩，保存到scores数组中，按照成绩从高到底的顺序进行排列，然后输出。
- 分析：
  - 本例和例5-3-1采用相同的排序方法，唯一的区别就在于比较两个元素的方法不同：上例是直接比较两个元素，而本例需要比较两个元素的数学成绩字段。

5.3.1 选择排序（7）

```
def find_max(scores,start,end):
    """
    找从scores[start]到scores[end-1]中，成绩最高的元素下标
    """
    max = start
    for i in range(start,end):
        if scores[i].score > scores[max].score:
            max = i
    return max
```

5-3-2.对象列表选择排序.py (找最大值函数)

5.3.1 选择排序（8）

```
def select_sort(scores):
    """
    选择排序
    :param scores: 待排序列表
    """
    n = len(scores)
    for i in range(n):
        t = find_max(scores,i,n)
        scores[i],scores[t] = scores[t],scores[i]
```

5-3-2.对象列表选择排序.py (选择排序函数)

数据结构与算法入门

一. 列表

二. 使用对象自定义数据类型

三. 文件输入输出

四. 迭代访问及其应用

五. 查找与排序

六. 字典与集合

6.1 字典

- 在5.2.3中我们介绍了索引和哈希表。实际上，由于索引和查找在程序处理中的应用非常广泛，python直接内置了哈希表的实现，即字典（Dict）。
- 字典就是利用哈希和索引的原理，实现高效存储的一种数据结构。

6.1.1 字典基本用法

语句	作用
`scores = {'张三':95, '李四':78, '王五': 82}`	创建包含三个元素的字典
`scores ={}`	创建空字典
`scores['郑六']=100`	在字典中增加关键字'郑六'及其对应的值100
`scores['王五']=59`	将'王五'对应的值修改为59
`s=scores['王五']`	查找关键字'王五'对应的值
`s=scores.get('孙七',-1)`	在字典中查找'孙七'对应的值。如果找不到，返回-1
`'王五' in scores`	判断字典中是否存在关键字'王五'

`for key in scores:`
`  value = scores[key]`
`  print(f"scores[{key}]={value}")`
	遍历访问字典中的所有关键字及其对应的值

6.1.1 字典基本用法（2）

- 关于字典，在使用时需要注意以下几点:
  - 关键字不能重复。在scores字典里不能有两个王五。
    - 因为人名存在重复的可能，所以一般不用人名做关键字
    - 关键字重复时，python不会提示错误，只会用后加入的值替换之前的值
  - 关键字必须是可哈希的（hashable）对象。常见的整数、浮点数和字符串都是可哈希的，可以用做关键字。
    - 所有的可变对象（见2.2节）都是不可哈希的。列表和字典都是可变对象，因此不能用作关键字。

6.1.2 字典应用举例

- 例6-1-2：已知文件5-1.scores.csv中保存着班上所有同学的学号、姓名和成绩信息。请编程实现：读入所有同学的信息，并根据用户输入的学号x查询其姓名和成绩。
- 之前我们用列表来保存学生信息，然后分别用顺序查找和二分查找解决了该问题。现在我们改用字典保存学生信息。
- 数据类型设计和数据结构选择：Score类型设计略。学生信息保存在字典中（关键字为学生的学号，值为Score类型的对象）

```
class Score:
    def __init__(self, id, name, score):
        self.id = id
        self.name = name
        self.score = score
```

Score类型定义

6.1.2 字典应用举例（2）

- 程序设计分析: 输入输出略
  - 核心处理步骤:
    1. 读取并建立学生信息字典
    2. 使用字典进行查询

6.1.2 字典应用举例（3）

```
def read_csv(filename):
    """
    从csv中读取学生信息，并以学号为关键字存入字典中
    :param filename: csv文件名
    :return: 以学号为关键字的学生信息字典
    """
    scores = {}
    with open(filename,mode="r",encoding="UTF-8") as file:
        reader = csv.reader(file)
        next(reader)
        for row in reader:
            id = row[0]
            name = row[1]
            score = Decimal(row[2])
            s=Score(id,name,score)
            scores[id]=s
    return scores
```

注意：如何向字典中加入新条目

3-1-2.字典查询.py (读取csv并生成学生信息字典)

6.1.2 字典应用举例（4）

```
id=dlg.get_string("请输入要找的学号")
if id in scores:
    s = scores[id]
    dlg.show_message(f"id: {s.id} 姓名: {s.name} 成绩: {s.score}")
else:
    dlg.show_message(f"找不到id为{id}的学生")
```

3-1-2.字典查询.py (查询部分)

6.2 集合

- 前面说过，在程序设计中，我们一般所说的集合（Collection）和数学中的集合（Set）并不完全一样。
  - Set中的元素不能重复，Collection中的元素可以重复
- 但有时我们也会需要用到符合数学中集合定义的数据结构。我们可以用列表来模拟这种集合，但实际上python中已经内置了专门的set对象，我们直接使用即可。

语句	说明
`s=set()`	建立一个空集合
`s.add('张三')`	向集合s中添加元素'王五' 重复添加的元素会被忽略
`'王五' in s`	判断s中是否包含元素'王五'
`s.discard('王五')`	从集合s中删除元素'王五'
`for elem in s:` `  print(elem)`	遍历集合s中所有元素

6.2.1 关于set的说明

- set和字典一样，内部使用哈希索引原理。因此：
  - 判断set中是否包含一个元素（即查找一个元素）的效率非常高
  - 和字典的关键字一样，set中的元素必须是可哈希的(hashable)
    - ✓ 大部分不可变对象，包括字符串、整数、浮点数等，都是可哈希的
    - ✓ 所有的可变对象，如列表、字典等，都是不可哈希的。

6.2.2 集合应用举例（1）

- 例6-2-2：已知文件6-2.sales.csv中保存着某商店当天的所有销售记录。请编程读入所有销售记录保存到sales列表中，然后找出并显示当天都销售了哪几种商品。
- 数据类型设计与数据结构选择：
  - 销售记录类型：Sales类设计（略）。因为没有特殊要求，我们选择使用最简单的列表来存储所有销售记录。
  - 商品名称：我们需要一个数据结构来保存出现在销售记录中商品名称。由于同样的商品名称只需要保存一个，所以最合适的数据结构是set。

```
class Sale:
    def __init__(self,name,price,quantity):
        self.name = name
        self.price = price
        self.quantity = quantity
```

Sale类定义

6.2.2 集合应用举例（2）

- 程序设计分析：输入、输出略
  - 核心处理步骤：
    1. 从csv文件中读取销售信息，保存到sales列表中
    2. 创建空集合names
    3. 迭代访问sales中所有元素，将其名称加入集合names
    4. 最后names中的所有元素就是题目中要求的商品名称

6.2.2 集合应用举例（3）

- 迭代访问sales循环分析:
  - 循环中用到的变量及其初始值：销售记录sales列表、商品名称集合names=set()
  - 处理步骤:
    - ✓将当前销售记录s的name属性加入names集合
  - 循环条件:
    - ✓用变量sale迭代访问sales列表中所有元素

6.2.2 集合应用举例（4）

```
def gather_names(sales):
    """
    汇总销售记录中的商品名

    :param sales: 销售记录列表
    :return: 商品名集合
    """
    names = set()
    for sale in sales:
        names.add(sale.name)
    return names
```

6-2-2.统计分类.py (汇总分类名称函数)

6.2.2 集合应用举例（5）

```
filename = dlg.get_open_file_name("请选择csv文件:
",dlg.FileFilter.CSVFiles)
if filename == "":
    print("未选择文件")
    exit(-1)
sales = read_csv(filename)

names = gather_names(sales)
print(names)
```

6-2-2.统计分类.py (主程序部分)

6.3 算法与数据结构

- **数据结构**（data structure）：（**大量**）数据在内存中的存放和组织形式。
  - 数组是最基本的一种数据结构。啥是数组？
- 同一种实体可以用不同的数据结构来表示。
  - 例如，集合可以用普通数组来表示，也可以用标识数组来表示。
- 同一个问题，由于采用的数据结构不同，其适合的解决方法也不同：
  - 例如，如何从集合中删除或增加元素？

6.3 算法与数据结构（2）

- 这些针对特定问题而提出的，由一系列明确的执行步骤而组成的解决方法，就是所谓的**算法(algrithms)**。
- 结构化程序设计思想认为：

  程序=数据（结构）+算法
- 显然，算法与数据结构是密切相关的。
- 不同的数据结构，处理相同的问题时用的算法不同
- 不同的算法，对数据结构的要求不同：
  - 二分查找要求列表中的数据满足特定的顺序要求
- 不同的算法，其处理效率也是不同的：
  - 二分查找的效率远远高于顺序查找

6.3 算法与数据结构（3）

- 学习程序设计，归根结底就是系统的学习：
  1. 已知一个算法，如何用程序来实现它？
  2. 常见问题的计算机表示方法及其处理方法
  3. 对于未知的问题，如何将其转化为已知的问题以便用已知数据结构和算法来解决？
  4. 对于未知的问题，提出新的表示方法（数据结构）或解决方法（算法）。
- 本章介绍了一部分最基本的算法。在《数据结构》课程中将进一步介绍更多的算法。

编程思路总结

- 在用程序（或者模块、函数）解决一个问题前，首先需要考虑的是：在程序（或者模块、函数）中，如何存储和表示问题相关的数据（信息）？
- 对于复杂结构的信息，我们需要自己设计相应的数据类型（类）
- 为了后续处理的方便，我们还需要选择合适的数据结构（列表，字典，集合或者其他的数据结构？）
- 然后我们再继续分析算法，包括：
  - 输入和其他初始条件
  - 输出
  - 处理步骤
- **对于列表而言，最基本的算法是用循环进行迭代访问，必须熟练掌握。**

本章重点

- 类的定义方法及其使用
- 列表及其迭代访问的用法
- 顺序查找和二分查找
- 选择排序法
- 字典及其使用方法

本章练习

- 见练习《第三章 数据结构与算法》