程序设计基础
——结构化程序设计基础

林大 经管学院 瞿华

结构化程序设计基础

一. 标识符与变量与数据类型

二. 运算符、表达式与赋值

三. 输入与输出

四. 关系运算、逻辑运算与if语句

五. 字符串初步

六. 函数与模块初步

结构化程序设计基础

一. **标识符与变量与数据类型**

二. 运算符、表达式与赋值

三. 输入与输出

四. 关系运算、逻辑运算与if语句

五. 字符串初步

六. 函数与模块初步

1.1 标识符

- **标识符(identifier)：指用来标识某个实体的一个符号**
  - **就是程序开发中用到的名字。**
- 在Python语言中，标识符主要用于对程序中的各种基本元素一一如变量、函数、类等一一进行命名。
- **在Python 2中，标识符只能由字母、数字和下划线组成，且首字符必须为字母或下划线。**
- 例如：
  - 合法标识符 firstname， name， age，_totale，student_num，FirstName。
  - 非法标识符 student.num，990810birth，$yuan 。
- 在Python 3中，在标识符中也可以使用中文汉字字符，但在实际程序中很少有人这么用。

1.1 标识符

- Python和C、Java等语言一样，标识符具有**大小写敏感性**（Case sensitivity）。
  - 即同一个字母的大写和小写会被认为是不同的两个符号。
  - 例如：firstname与FirstName是两个不同的标识符。
- 为了让程序更好理解，在给程序中的变量等元素起名时，最好能遵循一定的规则，即**“命名法”**
  - 最有名的是“匈牙利命名法”（据称是Microsoft的传奇人物，匈牙利人Chartes Simonyi发明的）
  - 基本原则：变量名=属性/类型+对象描述
  - 例如：nFileSize，sFileName
- 命名法没有优劣之分，只要在同一个公司、项目中遵守相同的规则即可。

1.1.1 Python代码规范

- 为了提升代码的可读性，降低软件开发的成本，在编写代码时应遵守一定的规范，如Python代码规范，Google代码规范等。
  - 重要的是合作者内部采用统一的规范
- 本课程采用Python代码规范
  - 原文见https://www.python.org/dev/peps/pep-0008/
  - 中文翻译请参考 https://www.cnblogs.com/hanweiblog/p/9345616.html

1.2 关键字

- **关键字(Key Word):在Python语言中具有特殊含义的单词，用户不能将它们用作标识符。**
- 常用的关键字：

  if else for while True False 等
- 现在不用刻意去记这些关键字，随着我们逐步的学习自然会知道
- 注意：虽然IF和if在Python语言看来是两个不同的单词，但我们也应该**避免使用这种容易和关键字搞混的单词作为标识符。**

1.3 变量

- 程序在运行时,必然会产生很多中间的运算结果
- 这些中间结果都记在什么地方呢?
- **寄存器（Register）**：寄存器是CPU内的组成部分。寄存器是有限存贮容量的高速存贮部件。
  - x86 CPU有8个通用寄存器
- **内存(Memory)**：CPU能直接通过数字编号（内存地址）寻址的存储空间。
  - 由于寄存器数量非常有限，所以大部分数据还是要放在内存中
- 例如：汇编语言(下面所有的数字均为16进制)：
  - mov ax,100 //把数100放到寄存器ax中
  - add ax, [200] //把地址为200的内存中的数与寄存器ax中的数相加，结果存放到ax中
  - mov [300],ax //把ax中的数存放到地址为300的内存中


免费寄物柜
Free Locker
寄包
若您的包规格在10x15cm以上请寄包。
取包
1. 现金、手机及贵重物品，价值200元以上请自行保管，丢失概不负责。
2. 勿向他人展示密码，保管好自己随身贵重物品。
3. 公共场所，谨防盗窃。
顾客寄包须知
寄包
取包
请您将小纸条放进垃圾篮
谢谢合作

1.3 变量

- 显然，无论直接用地址来访问内存，还是直接操作寄存器来存放数据，都是比较麻烦的。我们需要一个更方便的工具，这就是“变量”。
- **变量(variable)：一个有名字的存储单元。**
  - **我们不用考虑这个存储单元到底在寄存器中还是在内存中，具体的地址到底是多少，直接用它的名字来访问。**
  - **因为其中保存的值是可以被程序修改的（可变的），因此称为变量。**

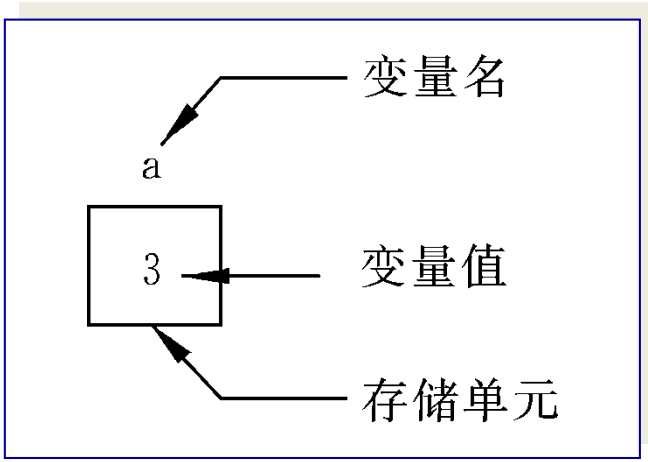
1.3.1 C语言中的变量

- **在C或者Java等语言中：**
  - **变量需要先声明，再使用**
  - **名字和存储单元的对应关系是固定不变的。**
- 例如：我们定义一个名为a的变量，并将其值设为3。

```
int a=3;
```

  - 也可以分成两步来写：

```
int a;
a=3;
```



1.3.1 C语言中的变量

- 如果我们在C语言的程序中定义了7个变量a,b,c,d,e,f,g，那么内存中就会是这样。

a b c d e f g

- 很显然，C语言必须得知道变量a占用多少内存空间（a格子右的边界到哪里），才能够去为下一个b变量分配对应的格子。
- C语句int a;中的int是变量类型，其作用之一就是告诉C的编译器，a到底要占多大空间。
- C中的变量一旦声明之后，其格子就固定了。以后再对其赋值，就会用新值直接覆盖格子中旧的内容。

1.3.1 C语言中的变量

- 给你一个格子，你在里面最多能写下多少位数字?
- 由于在C语言中，变量一旦声明之后，其对应的内存格子大小固定，而固定大小的格子其能存放的数字位数是有限的，其能存放的最大（最小）数字也就是定值。
- 如果试图向变量中存放一个更大（或者更小）的数字，超出的位数无处可存就会导致出错，这就是所谓的**整数变量溢出问题**（Integer Overflow）

```
int main() {
    int a,b,c;
    a=123456789;
    b=987654321;
    c=a+b;
    printf("c=%d\n",c);
    c=c*10;
    printf("c=%d\n",c);
    return 0;
}
```

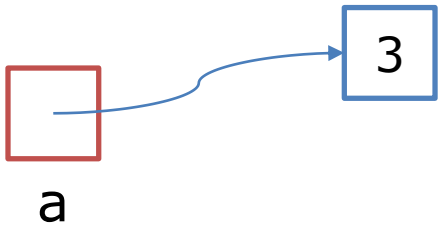
在32位C编译器下，第2次乘法计算c变量会溢出
(1-1.整数溢出.cpp)

1.3.2 Python语言中的变量

- **python为了解决整数溢出等问题，在普通变量的处理上与C是不一样的：**
  - **根据赋值的内容，来决定其在内存中存放的格子大小**
  - **每次赋值时，重新为新赋值内容找一个新的格子**
  - **变量中存放的是值所在格子的位置**
- 因此，Python中的变量可以这样表示：

```
a=3
```

python代码



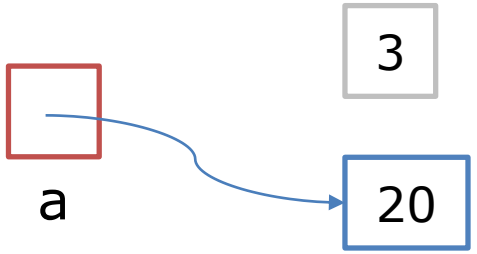
python变量图示

1.3.2 Python语言中的变量

- 重新赋值的图示：

```
a=3
a=20
```

python代码



python变量图示

Python会定期回收不再使用的旧格子

Python中的变量更像是一个标签！

1.3.3 两种变量处理方式的比较

- C和Python对变量的处理方式不同，哪个更好?
- Python变量的优点:
  - 使用方便
    - ✓ 不用声明类型
    - ✓ 不用担心溢出
- Python变量的缺点:
  - 处理速度慢（见《绪论》3.2节）
  - 消耗更多的内存空间
- 没有免费的午餐!

1.3.4 关于变量名

- Python代码规范中建议：
  - 变量名全小写，用下划线连接各单词，如
    - color = WHITE
    - this_is_a_variable = 1
  - 常量（程序运行时不变量的量）全大写，用下划线连接各单词，如PI、WHITE、BLACK等。
  - 避免使用"l","o"等容易看错的单个字母作为变量名或者变量名的一部分，如is_l, name_o等。

1.4 数据类型

- Python中的数据有不同的类型：
  - 各自的存储格式和支持的操作不同
- 例如：
  - 浮点数：31.4，使用类似科学记数法($3.14*10^1$)的二进制浮点格式来保存
  - 字符串：不支持加减法，但支持连接和重复（见第5节）
- Python中内置了若干数据类型，用户也可以根据需要定义自己的数据类型。
- Python在保存数据时，会同时保存其类型
- 可以用type()函数查看一个变量中的值的类型

1.4 数据类型

- Python中的数据有不同的类型：
  - 各自的存储格式和支持的操作不同
- 例如：
  - 浮点数：31.4，使用类似科学记数法($3.14*10^1$)的二进制浮点格式来保存
  - 字符串：不支持加减法，但支持连接和重复（见第5节）
- Python中内置了若干数据类型，用户也可以根据需要定义自己的数据类型。
- Python在保存数据时，会同时保存其类型
- 可以用type()函数查看一个变量中的值的类型

1.4 Python数据类型

- 和大多数语言一样，Python中内置了下列4种基本的数据类型：
  - 整型数字（int）：表示整数
  - 浮点型数字(float)：表示带小数的数字
  - 逻辑型/布尔型(bool)：表示逻辑真和假
  - 字符串(str)：表示文字内容

```
x=3
type(x)
x=3.14
type(x)
x=True
type(x)
x='abc'
type(x)
```

在交互模式下逐行运行上面的预计，观看type()函数的运行结果

1.4 Python数据类型

- 和大多数语言一样，Python中内置了下列4种基本的数据类型：
  - 整型数字（int）：表示整数
  - 浮点型数字(float)：表示带小数的数字
  - 布尔型(bool)：表示逻辑真和假
  - 字符串(str)：表示文字内容

```
x=3
type(x)
x=3.14
type(x)
x=True
type(x)
x='abc'
type(x)
```

在交互模式下逐行运行上面的预计，观看type()函数的运行结果

1.4.1 浮点数与舍入误差

- 浮点（实型）类型主要用于科学计算和表示实数等整型无法表示的数据。
- 浮点类型得名于其采用的（二进制）浮点数据表示法，浮点表示法与我们熟悉的十进制科学计数法类似。
  - 使用科学计数法法，我们可以把任意的十进制实数表示为$a*10^b$的形式
  - 同理，使用浮点表示法可以把任意的二进制实数表示为$a*2^b$的形式。
- **使用浮点数时，最容易出现的问题是舍入精度问题。**
  - 特别的，很多用10进制表示时小数位数有限的数，用二进制来表示时小数位数会增加、甚至成为无限循环小数！

1.4.1 浮点数与舍入误差

- 例如：0.1表示为2进制：

  0.0001100110011001100110011001100110011001100110011001100110
  011001100...

- 因此你觉得可能不存在舍入精度问题的数字，恰恰就可能会存在舍入误差！（舍入误差.py）

```
x=0
for i in range(10):
    x=x+0.1
    print(x)
```

舍入误差.py

导弹

案例：爱国者导弹

- 浮点运算的不精确性能够产生灾难性的后果。
- 1991年2月25日，在海湾战争期间，沙特阿拉伯的达摩地区设置的美国爱国者导弹，未能成功拦截一枚伊拉克发射的飞毛腿导弹。该枚飞毛腿导弹击中了美国的一个兵营，并造成28名士兵死亡。
- 事后，美国GAO（Goverment Accountability Office）对拦截失败的原因做了详细的分析，最终发现原因是爱国者导弹控制软件在使用浮点数计算时产生的累积误差：
- （一）在爱国者导弹中内置了一个计时单元，该单元使用整数来表示时间。在计算时，需要将其乘上0.1转换为浮点数来表示实际的时间。由于0.1是个二进制循环小数，所以转换后会有大约$9.5 * 10^{-8}$的误差。
- （二）该误差会随着导弹系统的连续运行而不断累积。如果一套爱国者导弹系统连续运转100小时，拦截目标的速度为1676米/秒，位置计算的误差就会有573米！

结构化程序设计基础

一. 标识符与变量与数据类型

二. 运算符、表达式与语句

三. 输入与输出

四. 关系运算、逻辑运算与if语句

五. 字符串初步

六. 函数与模块初步

二、运算符、表达式与赋值

- **运算符**（operator）就是用抽象符号（而不是标识符或关键字）定义的对数据的运算或者操作。
- 例如，在Python语言中，要表示两个数10和5相加：
  - 不用 10 add 5
  - 而是10 + 5
- **运算符及其操作的对象组成了表达式**(Expression）。
- 表达式的计算结果称为表达式的值。

2.1 无运算符表达式

- 在Python语言中，变量也是表达式。
- 变量表达式的值就是变量中保存的值：
  - 已知变量a的值是5
  - 则表达式 a 的值就是5。
- 括号表达式：用圆括号括起来的表达式
  - （ 表达式 ）

2.2 算术运算和算数表达式

- **算术运算符**是Python语言中最常用的运算符之一，其实就是我们小学时学的加减乘除等算术运算。
- 基本的算术运算符：+ （**加法运算符**，或**正值运算符**。如：3 + 5、 + 3）
  - - （**减法运算符**，或**负值运算符**。如：5 - 2、 - 3）
  - * （**乘法运算符**。如：3 * 5）
  - / （**除法运算符**。如：5 / 3 的结果为1.67）
  - ** （乘幂运算符。5**2的结果为25）
  - // （整除运算符。10/3的结果为整数3）
  - % （**模运算符**，或称**求余运算符**。如:10%3的值为1，即10除3的余数为1）。

2.3 算数运算次序与优先级

- 与数学中的运算一样，Python语言中的运算也是有次序和优先级的。
- 例如，在算术运算中，乘除的优先级高于加减运算。
  - 在算术运算中，运算的进行次序是**先左后右**
  - 3+5*4的结果是多少?
- 同样，在Python语言中也可以使用括号”(“”)”来改变运算的次序或优先级别。
- 例如:
  - (3+5)*4的结果是多少?

2.3 算数运算次序与优先级

- **注意，在Python语言中，除了()之外，其他类型的括号都另有用途，不能用在算术表达式中！**
- **例如：**

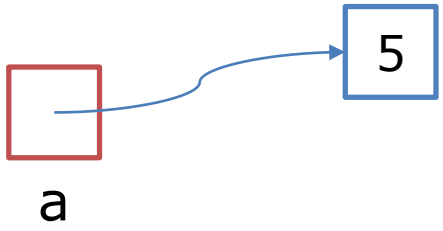
  3*((3+4)*5+6)

  注意**不能**写成3*[(3+4)*5+6]

2.4 赋值(1)

- 赋值（Assignment）语句的作用是将一个数据赋给一个变量，其符号是“=”。
- 赋值语句的左边必须是一个变量，右边可以是任意的表达式等。
- 如果被赋值的变量之前不存在，则Python自动创建一个新的变量
- 在python中，赋值的作用是让变量指向被抚育的值（对象object），因此也称为绑定(bind)。
- 例如：

```
a=5    #右边是字面量表达式
```

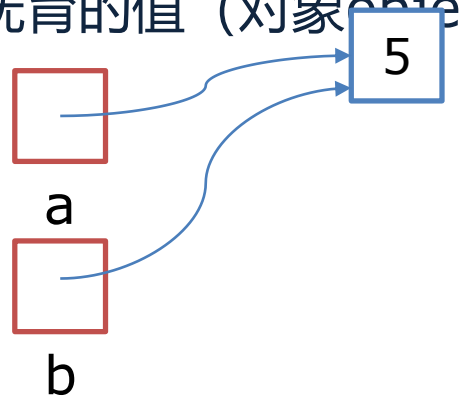


将a绑定为5

2.4 赋值(2)

- 赋值（Assignment）语句的作用是将一个数据赋给一个变量，其符号是“=”。
- 赋值语句的左边必须是一个变量，右边可以是任意的表达式等。
- 如果被赋值的变量之前不存在，则Python自动创建一个新的变量
- 在python中，赋值的作用是让变量指向被抚育的值（对象object），因此也称为绑定(bind)。
- 例如：

```
a=5    #右边是字面量表达式
b=a    #右边是变量表达式
```

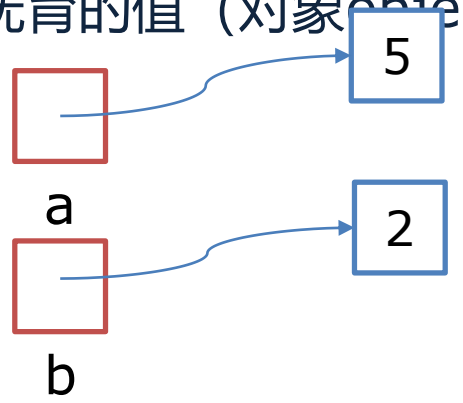


a和b都被绑定到5

2.4 赋值(3)

- 赋值（Assignment）语句的作用是将一个数据赋给一个变量，其符号是“=”。
- 赋值语句的左边必须是一个变量，右边可以是任意的表达式等。
- 如果被赋值的变量之前不存在，则Python自动创建一个新的变量
- 在python中，赋值的作用是让变量指向被抚育的值（对象object），因此也称为绑定(bind)。
- 例如：

```
a=5    #右边是字面量表达式
b=a    #右边是变量表达式
b=(5+7)/6   #右边是算数表达式
```

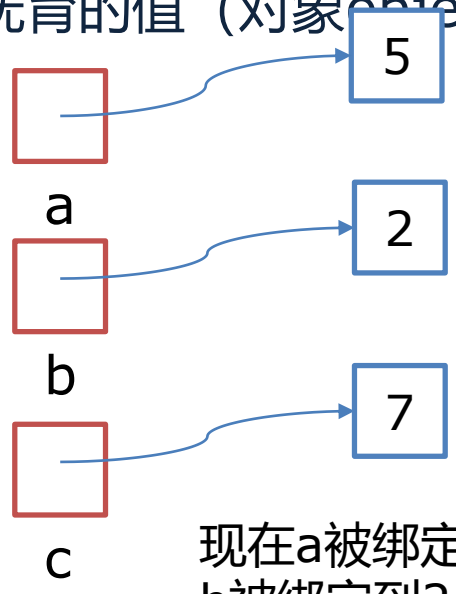


现在a被绑定到5
b被绑定到2

2.4 赋值(4)

- 赋值（Assignment）语句的作用是将一个数据赋给一个变量，其符号是“=”。
- 赋值语句的左边必须是一个变量，右边可以是任意的表达式等。
- 如果被赋值的变量之前不存在，则Python自动创建一个新的变量
- 在python中，赋值的作用是让变量指向被抚育的值（对象object），因此也称为绑定(bind)。
- 例如：

```
a=5    #右边是字面量表达式
b=a    #右边是变量表达式
b=(5+7)/6    #右边是算数表达式
c=a+b    #右边是算数表达式
```



现在a被绑定到5
b被绑定到2
c被绑定到7

2.4 赋值(5)

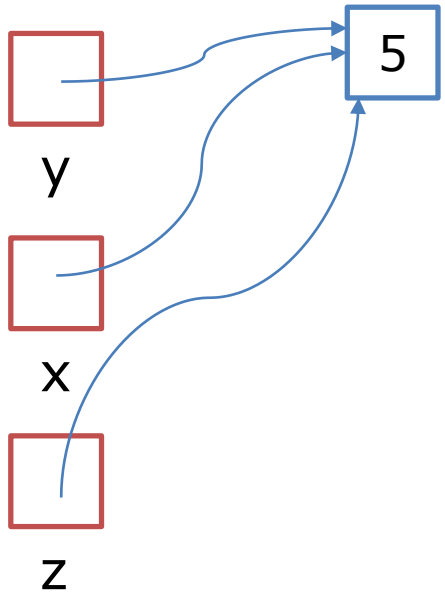
- 例：交换两个变量的值
  - 已知a、b两个变量。如何让a和b互换所指向的内容?
  - 类比：有两个杯子，一个装咖啡，一个装牛奶。如何交换这两个杯子中的饮料?
- 直接a=b;b=a;行不行？为什么?
- 应该怎么办?
  - 需要引入一个中间变量t:

    t=a;a=b;b=t;

注：python中回车换行和;都可以用来分开两个语句

2.4.1 连续赋值

- 有时我们需要给几个变量赋相同的值。Python提供了简化的方法
- 例如：以下的代码将z、x、y三个变量都赋值为3
  - z=x=y=3



2.4.2 复合赋值运算符(1)

- 前面提到过，我们在给变量起名时应尽可能起有意义的名字，所以我们在实际程序中经常会写这样的语句：

  sumOfScores=sumOfScores+score

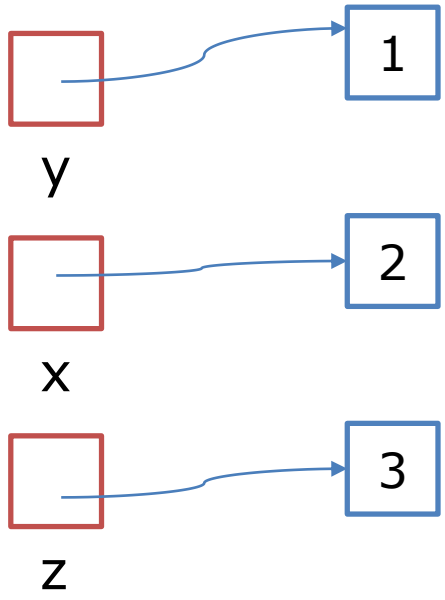
- 这样的语句内容无论是输入还是阅读显然都是比较繁琐的，能否简化一点呢?

2.4.2 复合赋值运算符(2)

- **复合赋值运算就是将算术运算和赋值运算结合在一起而构成的运算。**例如：
  - a+=b相当于a=a+b, sumOfScores+=score相当于sumOfScores=sumOfScores+score
  - a-=7相当于a=a-7
  - a%=3相当于a=a%3，a//=3相当于a=a//3
  - a*=4相当于a=a*4
  - b/=5相当于b=b/5
  - x**=2相当于x=x**2
  - 等等
- 合理使用复合赋值运算可以简化程序的输入和表示

2.4.3 列表赋值

- 除了连续赋值，在Python中还可以同时给多个变量赋不同的值，称为列表赋值
- 例如：以下的代码将z、x、y三个变量分别赋值为1，2，3
  - x,y,z=1,2,3



注意：赋值号两边列表中的元素个数必须相同

2.4.3 列表赋值

- 使用列表赋值语法，我们可以直接这样来交换a、b所绑定的值：
  - a,b=b,a

结构化程序设计基础

一. 标识符与变量与数据类型

二. 运算符、表达式与语句

三. 输入与输出

四. 关系运算、逻辑运算与if语句

五. 字符串初步

六. 函数初步

3 输入与输出

- 任何一个有意义的程序都需要通过某种方式**将信息**（如运算结果等）**表现给用户**，即**输出**。
- 很多程序也需要某种手段来让用户给出运算的起始条件，或对计算过程进行控制。这种**程序从用户处获取信息的手段就是输入**。
- 因此，在继续学习其他知识前，我们需要先了解一下Python语言中最基本的输入和输出方式。
- 其实就是Python内置的两个函数：
  - input()
  - print()

3.1 屏幕输出print

- print的作用是将内容输出到标准输出（一般是屏幕窗口）中。
- 基本用法
  - print(表达式1,表达式2,……)
- 例如：
  - print(2*3)
  - print(x)
  - print(x,y,z)
  - print(x**2)
  - print("你好！")

3.2 键盘输入input

- **input函数的作用：**从标准输入读入用户输入的内容（字符串）保存到指定的变量中。
- **基本用法**:

  变量=input("输入提示语")
- 例如:
  - s=input("请输入s:")

3.3 类型转换

- 在1.4节中介绍过，不同数据类型支持的操作不同
- input函数读入的数据是字符串型，不支持算术运算
- 可以使用下列类型转换函数来转换数据类型：
  - int()：将字符串转换为整数
  - float()：将字符串转换为浮点数
- 如果我们希望读入整数或者浮点数，需要使用类型转换函数进行类型转换：
  - 输入整数：x=int(input("请输入整数"))
  - 输入浮点数：x=float(input("请输入浮点数"))

3.4 对话框输入输出*

- 通过第三方库EasyGraphics，我们可以在程序中以对话框和提示框的形式进行输入输出
  - easygraphics安装见"A1.Python开发环境安装"第3.2节
  - 在程序中首先需要导入easygraphics：
    - ✓ import easygraphics.dialog as dlg

*:本节自学

3.4.1对话框输入*

```
import easygraphics.dialog as dlg

s=dlg.get_string("请输入s","这是标题")
print(s)
```

输入单条数据（字符串）

```
import easygraphics.dialog as dlg

values=dlg.get_many_strings("请输入： ",labels=["输入项1","
输入项2","输入项3"])
print(values[0],values[1],values[2])
```

输入多条数据（字符串），返回的是一个dict

3.4.1对话框输入(2)*

```
import easygraphics.dialog as dlg

c=dlg.get_yes_or_no("你是张三吗？")
print(c)
```

让用户选择Yes/No，选Yes返回True，选No返回False

```
import easygraphics.dialog as dlg

c=dlg.get_choice("选一个吧","这是标题",choices=["选择1","选择2","选择3"])
print(c)
```

让用户选择，返回选择的内容（单选）

3.4.1对话框输入(3)*

```
import easygraphics.dialog as dlg

choices=dlg.get_list_of_choices("多选几个",choices=["选择1","选择2","选择3","选择4","选择5"])
print(choices)
```

让用户选择，返回选择的内容（多选）
返回的是一个列表（见第3章）

3.4.2对话框输出*

```
import easygraphics.dialog as dlg

dlg.show_message("这是我的内容","这是标题")
```

输出内容

```
import easygraphics.dialog as dlg

s="张三"
dlg.show_message(s+",你好！ ","打招呼")
```

输出内容（字符串表达式）

结构化程序设计基础

一. 标识符与变量与数据类型

二. 运算符、表达式与语句

三. 输入与输出

四. 关系运算、逻辑运算与if语句

五. 字符串初步

六. 函数与模块初步

4.1 比较运算

- 数字之间除了算术运算外，还有一类重要的运算，如：
  - a>3
  - Salary<3500
  - isGirl==1
- 这类判断两个数字间大小或相等关系的运算在Python语言中称为**比较运算**（Comparisons）。
- **比较运算符就是对两个值进行比较的运算符**
- **比较表达式由比较运算符和两个值组成**

4.1 比较运算(2)

- **Python语言提供了六种比较运算符：**

  ① < (小于)　② <= (小于等于)

  ③ > (大于)　④ >= (大于等于)

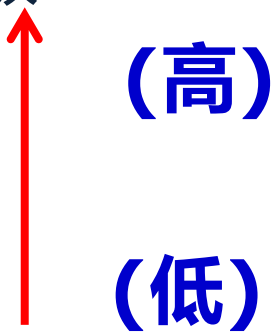
  ⑤ **==　(等于)**　⑥ **!=　(不等于)**

4.1 比较运算(3)

- 比较运算的结果：
  - 如果两个值的实际大小关系和比较表达式的形式一致，则运算结果为True（即条件判断成立）
  - 如果两个值的实际大小关系和比较表达式的形式不一致，则运算结果为“假”（即条件判断不成立）
- 例如：在python交互环境下执行下列语句，查看结果：
  - 5>3
  - 5<3
  - 5==4
- 思考：a>b的结果是多少?

4.1 比较运算(4)

- 比较、算术、赋值的优先级：



- 例：

c>a+b　　等效于　c>(a+b)

a=b>c　　等效于　a=(b>c)

4.1.1 链式比较运算

- 在Python中，可以这样：
  - 判断a是否介于3和10之间（开区间）：3<a<10
  - 判断a是否介于3和10之间（闭区间）：3<=a<=10
- 注意：**这个语法在C和Java中不适用！**

4.2 if语句与选择结构

- 在现实中处理问题时，往往需要对情况进行进行判断，不同的情况采用不同的处理方式。
- 例如：上课进教室
  - 到教室门口
  - 判断：已经上课了吗?
    - 是：从后门悄悄进入
    - 不是：从前门正常进入

4.2 if语句与选择结构(2)

- 其他的一些例子：
  - 如果星期天天气好，就去爬香山
  - 如果今天不想上课，而且老师上课不点名，那么就跷课
  - 如果饭卡里钱多于20块，就买麻辣香锅；否则就买稀饭馒头咸菜
  - 如果今天是女朋友生日，那么就去买一束花
- 类似的，我们在程序中也经常需要判断处理，这是通过**选择结构**来实现的。

4.2 if语句与选择结构（3）

- Python语言中通常用if语句来实现选择，其语法为：

```
if 条件表达式:
    代码段1
else:
    代码段2
```

  - 如果条件表达式计算结果为True，则执行代码段2；如果结果为False，则执行代码段2
  - else和代码段2可以省略，即条件表达式结果为True，则执行代码段2；否则不做处理继续执行if下面的语句。
- 注意：python使用缩进来标记代码段的开始和结束
  - **缩进完全相同**的语句处于同一个代码段！
  - 必须使用**英文空格**（半角空格）或者制表符（不推荐）来缩进
  - 大多数IDE会自动把tab键输入的制表符转换为4个英文空格缩进

4.2 if语句与选择结构（4）

- 使用python运行下面两段代码，比较运行结果：

```
x=10
if x>5:
    print("天气不错")
else:
    print("天气不好")
    print("不好")
```

```
x=10
if x>5:
    print("天气不错")
else:
    print("天气不好")
print("不好")
```

4.2 if语句与选择结构(5)

- 例4-1：女生节食堂加餐
  - 女生节某食堂促销，给每个来打饭的女同学免费送一个鸡腿。
  - 编写程序，询问来打饭同学的性别，根据回答提示是否送鸡腿。
- 程序设计的基本套路就从回答这几个问题开始：
  1. 程序需要哪些输入？格式是怎样的？用什么方式输入？输入的数据如何存放（用哪些变量）？
  2. 程序应该得到哪些结果？以什么样的方式输出？
  3. 如何从输入得到想要的结果？需要哪些步骤？是否需要变量来保存中间步骤的结果？

作为编程基础课，我们首先重视编程思路的锻炼！

4.2 if语句与选择结构(6)

- 示范思路：

1. 程序需要哪些输入？格式是怎样的？用什么方式输入？输入的数据如何存放（用哪些变量）？

需要输入用户的性别；可以用逻辑值True表示是女生，False表示不是；可以用对话框提示用户选择是否女生；存放到变量is_girl中。

2. 程序应该得到哪些结果？以什么样的方式输出？

是否给鸡腿。可以用对话框来提示用户是否给鸡腿。

3. 如何从输入得到想要的结果？需要哪些步骤？是否需要变量来保存中间步骤的结果？

用if语句对is_girl变量中的值进行判断即可

作为编程基础课，我们首先重视编程思路的锻炼！

4.2 if语句与选择结构(7)

```
import easygraphics.dialog as dlg

is_girl=dlg.get_yes_or_no("你是女生吗？")
if is_girl:
    dlg.show_message("免费送你个鸡腿~~")
dlg.show_message("继续打饭")
```

4-1.女生节.py

4.2 if语句与选择结构(8)

- 例4-2：火车票价查询。编程，输入用户的身高和车票的全票票价，根据下列规则计算票价：
  - 身高在1.5以上的全价
  - 身高小于等于1.5米的半价
  - 身高小于等于1.3米的免票
- 描述一下你的编程思路？
  1. 程序需要哪些输入？格式是怎样的？用什么方式输入？输入的数据如何存放（用哪些变量）？
  2. 程序应该得到哪些结果？以什么样的方式输出？
  3. 如何从输入得到想要的结果？需要哪些步骤？是否需要变量来保存中间步骤的结果？

4.2 if语句与选择结构(9)

- 参考编程思路：
  1. 输入
     1. 需要读入身高和全票票价。身高单位用米，读入一个浮点数；票价单位是元，同样读入一个浮点数；
     2. 使用对话框读入身高和票价。由于easygraphics对话框读入的是字符串，所以需要将读入的数据分别转换成浮点数（见3.3节）
     3. 使用变量height和full_price来保存读入的数据
  2. 结果和输出
     1. 结果是票价，保存在变量price中。
     2. 使用对话框输出票价计算结果
  3. 如何处理：
     1. 使用if语句，根据height的值所处的区间，用对应的公式来计算票价

4.2 if语句与选择结构(10)

```
import easygraphics.dialog as dlg

values = dlg.get_many_strings("请输入", labels=["身高(米)", "
全票票价"])
height = float(values[0])
full_price = float(values[1])
if height <= 1.3:
    price = 0
else:
    if height <= 1.5:
        price = full_price / 2
    else:
        price = full_price

dlg.show_message("最终票价为:" + str(price))
```

4-2.票价计算.py

4.2 if语句与选择结构(11)

```
import easygraphics.dialog as dlg

values = dlg.get_many_strings("请输入", labels=["身高(米)",
"全票票价"])
height = float(values[0])
full_price = float(values[1])
if height <= 1.3:
    price = 0
elif height <= 1.5:
    price = full_price / 2
else:
    price = full_price

print("最终票价为:" + str(price))
```

4-2.票价计算2.py

注意elif 相当于 else + if

4.3 逻辑值与数

- 由于历史原因（为了兼容C语言的一些习惯），在Python中：
  - 当需要逻辑值时，可以提供一个数：
    - **结果为非0，相当于**True： if 5:
    - 结果为0,相当于False: if 0:
  - 当需要数时，可以提供一个逻辑值：
    - True相当于1 ：True+1　True+True
    - False相当于0 ：2-False
- 为了避免混乱，我们最好不要在自己的python程序中使用这个特性

4.4 逻辑运算

- 很多时候我们需要对多个条件判断的结果进行汇总，得到最终的判断结果。例如：
  - 如果明天早上天气好而且我心情好，就去上自习
  - 如果明天早上天气不好或者我心情不好，就不去上自习
  - a介于3和10之间（a大于3并且a小于10）
- 这种运算在计算机和数学中称为逻辑运算或者布尔运算（Boolean Operations）
- Python语言中使用**逻辑运算符**来将多个条件判断组合在一起。

4.4 逻辑运算

- 3种逻辑运算符：

  and（逻辑与） or（逻辑或） not（逻辑非）

- **逻辑表达式就是用逻辑运算符将比较表达式或其他逻辑量连接起来的式子。**例：
  - **a>3 and a<10**
  - **x>0 or y>0**
  - **not a>3**
  - **not True**

4.4.1 逻辑与

- 逻辑运算的作用可以用真值表来。
- 假设a和b分别是两个逻辑变量，对于and有

a	b	a and b
True	True	True
True	False	False
False	True	False
False	False	False

- 即只有a和b同时为真，a and b才为真；只要a或b其中之一为假，a and b就为假
- and相当于语言中的“而且”

4.4.2 逻辑或

- 假设a和b分别是两个逻辑变量，对于or有

a	b	a or b
True	True	True
True	False	True
False	True	True
False	False	False

- 即只要a或b之一为真，a or b就为真；只有当a和b全为假时，a or b才为假
- or相当于语言中的“或者”

4.4.3 逻辑非

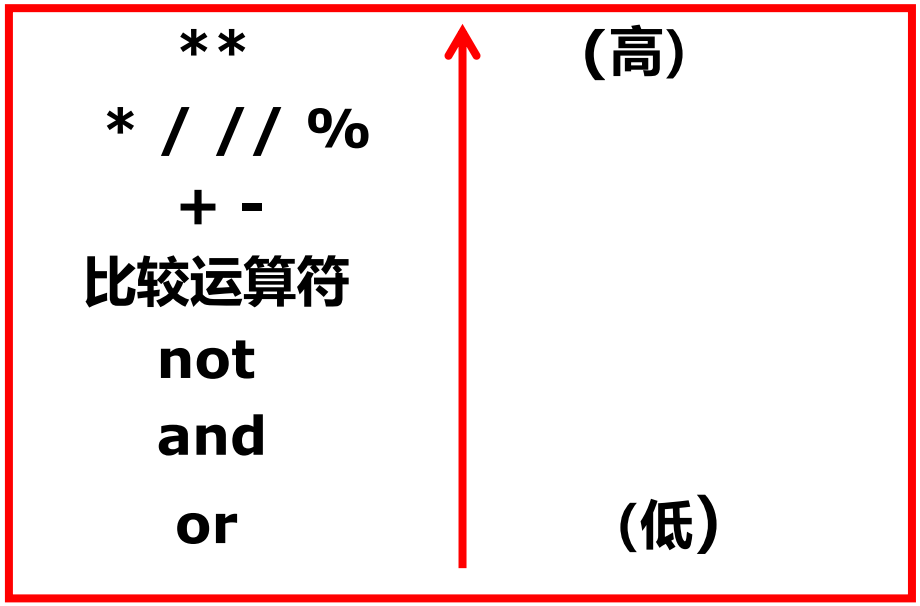
- 假设a是一个逻辑变量，对于not有

a	not a
真T	假F
假F	真T

- 即not a正好和a相反，a为真，！a就为假；a为假，!a就为真
- 例： !(x>3)等价于x<=3
- !相当于语言中的 “非”或者“不是”
- 练习：阅读1-4.logic.c，预测其输出

4.4.4 逻辑运算优先次序

- 与其他运算符的优先次序



- **为了避免错误，尽量用括号表明计算次序而不是依赖运算符本身的优先级！**

4.4.5 练习

- 说出下列表达式何时为真，何时为假?
  - (a>5) or (b>7)
  - (x!=3) and (y==4)
  - not is_girl or (age<20)

4.4.5 练习

- 如何用C表达式表达下列条件判断?
  - x小于等于10
  - a是负数
  - a介于6（包含）和10（不包含）之间
  - x是大于10的偶数

结构化程序设计基础

一. 标识符与变量与数据类型

二. 运算符、表达式与语句

三. 输入与输出

四. 关系运算、逻辑运算与if语句

五. 字符串初步

六. 函数与模块初步

5.1 字符串与字符

- 计算机中文字内容表示和处理的基本单元是字符串。
- 字符串（String）就是由若干个字符(Character）顺次排列起来的文字内容：
  - "今天天气不错"
  - '3+4*5'
  - "1235423"
  - "She's beautiful!"
  - 'He said:"Goodbye!"'
- 为了和其他编程元素区分，用"或者'把字符串内容包裹起来。

5.2 字符

- 在Python中没有专门的字符类型（C和Java有），用单个字符的字符串来表示单个字符：
  - 'a','c','哈','.',','…
  - 特殊字符：'\n'（回车),' '(空格),'\''(单引号),'\\'（单个\，因为\'是单引号）等
- 在计算机中，一切都是数或者说0和1（二进制只有0和1），因此计算机内部用数字编号来表示字符。
- 每个字符都对应一个数字编号，计算机看到这个编号就知道是哪个字符：
  - 例如，在常用的ASCII编码下，65对应'A',66对应'B'，97对应'a'，48对应'0'，32对应' '(空格)
- 注意：**数字0和字符'0'对于计算机来说是不一样的！**

5.2.1 字符编码

- 这种文字符号对应的数字编号就称为**字符编码（Character Code）**。
- 字符和字符编码之间的对应关系（编码方式,Encoding）完全是人为规定的。
- 但是为了计算机之间、程序之间能够相互理解对方的信息，显然有必要使用统一的编码方式。
- 由使用同一规则编码的所有字符及其对应的代码组成了一个**字符集(Character Set)**。
- ASCII是目前最常用的英文字符编码集，其中，一共包括127个常用英文和数字字符。
- 大多数非英文字符集都是ASCII的超集，即其中前127个字符及其编码与ASCII完全相同。

5.2.2 GBK编码*

- ASCII编码表中未包括中文，因此要表示和处理中文，必须使用其他字符集。
- 现行的常用简体中文字符集主要有两个，即GBK和UTF-8。
- 中文Windows使用的GBK编码来源于GB2312。前面的GB代表“国标”。
  - 为了统一汉字在计算机中的标识，中国国家标准总局在1980年以GB2312号国家标准的形式规定了基本汉字的编码。这就是GB2312字符集，其中共包含7445个字符，其中6763个常用汉字。
  - 由于GB2312中包含的汉字太少，特别是不包含一些少见的姓名文字、繁体字和日文、韩文，造成使用上的不变。
  - 于是微软在GB2312的基础上自行进行了扩充，并用于Windows 95系统，这就是GBK。该字符集并非国家标准，但是被国内业界广泛采用。GBK共包含21886个字符，其中常用汉字14240个。

5.2.3 UTF-8编码*

- UTF-8是互联网应用较多的一种中文编码。
- 由于历史原因，各非英语国家都采用自己的一套互不兼容的字符编码标准。
- 进入互联网时代后，这种文字标准互不兼容的现状严重阻碍了各国网民在互联网上的交流。
  - 例如，我用GBK编码给日本的网友发信息，他看见的可能就是一堆乱码。
- 各自不同的编码标准也导致开发可供多种语言使用的程序非常困难。

5.2.3 UTF-8编码*(2)

- 因此，成立了Unicode学会，并致力于一种能够表示世界上所有语言的编码的开发。
  - 1994年正式公布第一版，目前是第6版。
  - Unicode中实际上包含多种编码方案，其中最常用的是即**UTF-8**。
  - 例子:查看百度、新浪和林大主页的字符编码
  - python 3内部使用utf8编码保存和处理字符串
  - 从python 3开始，程序中如果没有明确声明，则认为程序中所有的字符串都是utf-8编码。

#要抓取的股票代码
code="sh601006"
#抓取网页(json格式)
response = request.urlopen('http://hq.sinajs.cn/list='+code)
raw_data=response.read().decode('gbk')

5.2.4 中文编码*

- 从外部信息源读写数据时，需要告诉python使用哪个字符集，否则可能会出现乱码问题。

```
from urllib import request
import easygraphics.dialog as dlg


#要抓取的股票代码
code="sh601006"
#抓取网页(json格式)
response =
request.urlopen('http://hq.sinajs.cn/list='+code)
raw_data=response.read().decode('gbk')
```

查询股票信息.py（局部）
试一试，如果把标红语句中的'gbk'去掉或者改成'utf-8'，会怎样?

5.2.5 字符与编码转换

- 在python中可以：
  - 使用ord()函数获取单个字符对应的字符编码
  - 使用chr()函数获取编码对应的字符
- 练习：在交互模式下查看下列结果：

```
ord('a')
ord('b')
ord('0')
ord('A')
ord('.')
ord(',')
ord('\n')
ord(' ')
ord('中')
```

```
chr(97)
chr(98)
chr(48)
chr(65)
chr(46)
chr(44)
chr(10)
chr(32)
chr(20013)
```

5.3 基本字符串操作

- Python提供了一系列常用的字符串操作，其构成了文字处理的基础：
  - 赋值：s='You are good!'
  - 字符串连接：'abc'+'123';
  - 字符串重复：'abc'*3
  - 获取字符串长度（字符个数）：len(s)
  - 判断前缀：s.startswith('You')
  - 判断后缀：s.endswith('od!')
  - 查找:s.find('are')
  - 替换：s.replace('are','is')
- 在交互模式下运行上述语句，查看结果

语句	说明
s='You are good!'	赋值
s='abc'+'123' s1=s+'456'	字符串连接
s='abc'*3 s1=s*2	字符串重复
len(s)	获取字符串长度（字符个数）
s.startswith('You')	判断s是否以'You'为前缀
s.endswith('od!')	判断s是否以'od!'为后缀
s.find('are')	查找'are'在s中出现的位置
s.replace('are','is')	返回用'is'替换s中所有的'are'所得到的新字符串
s.strip() s.lstrip() s.rstrip()	返回去除s两端（左侧/右侧）的空白字符（包括回车、制表符）得到的新字符串
s.upper() s.lower()	返回将s中所有字符全变为大写（小写）所得到的新字符串

5.3.1 字符串类型转换

- 输入和输出时经常需要将字符串转换成其他类型，或反之：
  - print("你的个人所得税是：" +str(tax))
  - x=float(input("请输入身高(米):"))
  - y=int(input("你有几只宠物:"))

5.4 格式运算

- 在向用户显示计算结果时，我们经常需要按照预定的格式向用户显示计算结果：
  - “您的月收入是XXX，个人所得税是XXX"
  - "考号：XXX,身份证号：XXXX，姓名：XXX，成绩：XXX “
- 可以使用连接运算"+"来连接预定义的文字和计算出来的内容，但是当预定的格式内容较多，要填写的内容复杂时，这样会很繁琐。
- Python提供了多种格式处理的方法：
  - "%"运算："您的月收入是%d,个人所得税是%d"%(income,tax)
  - "format"函数："您的月收入是{0},个人所得税是{1}".formt(income,tax)
  - 格式化字面量： f"您的月收入是{income},个人所得税是{tax}"

5.4.1 %运算

- %运算又称为类printf格式，其语法来源于C语言中的printf函数。
- %前是格式字符串，后面是用（）括起来填写的内容（当只有一个值要填写时，可以省略括号）
- 格式字符串中用下列占位符表示内容要以什么格式输出：
  - %d （十进制）整数
  - %f （十进制）浮点数
  - %s 字符串

```
name="张三"
income=19000
rate=0.3
s="%s您好，你的收入是%d元，应缴%.2f元"
print(s%(name,income,income*rate))
print("%s您好，你的收入是%d元，应缴
%.2f元"%(name,income,income*rate))
```

%实例，%.2f表示浮点数四舍五入到小数点后2位

当占位符格式不对时会出错，所以不建议新手使用！

5.4.2 format函数

- format函数提供了更加安全的格式化途径。
- 格式化字符串中使用'{0}','{1}'来表示填写的内容。
  - 其中的数字表示要填写的内容在format参数中的位置。

```
name="张三"
income=19000
rate=0.3
s="{0}您好，你的收入是{1}元，应缴{2}元"

print(s.format(name,income,income*rate))
print("{0}您好，你的收入是{1}元，应缴{2}元
".format(name,income,income*rate))
```

format实例

5.4.3 格式化字面量

- 格式化字面量（Formatted String Literals）是python 3.6版本新加入的功能。可以直接在字符串中嵌入和计算表达式的值
  - 在"前加f，表示这是个格式化字符串字面量
  - 用'{'和'}'括起要插入的表达式

```
name="张三"
income=19000
rate=0.3

print(f"{name}您好，你的收入是{income}元，应缴{income*rate}元")
```

format实例

结构化程序设计基础

一. 标识符与变量与数据类型

二. 运算符、表达式与赋值

三. 输入与输出

四. 关系运算、逻辑运算与if语句

五. 字符串初步

六. 函数与模块初步

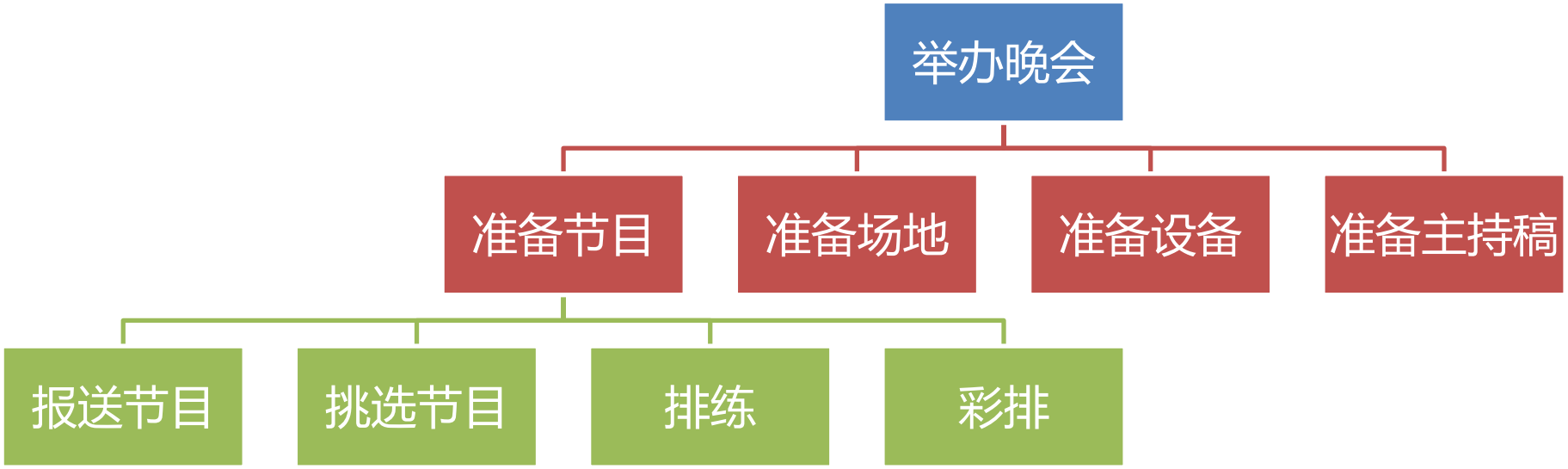
6.1.1 代码重用与模块化

- 在设计复杂软件时，经常需要重复使用相同的功能。
- 例：教务管理系统
  - 对学生按学号排序
  - 对学生成绩排序
  - 对老师按照姓名排序等
- 用复制/粘贴实现相同的功能——不好！
  - 维护、修改麻烦：需要处处修改
- 代码重用（Code Reuse）：把基本的功能编写成相对独立的程序单位，以便在程序中重复利用，也称为代码复用。
- 代码重用要求我们把重复使用的功能写成相对独立的程序单位。

6.1.2 自顶向下与模块化

- 在现实中，当需要解决复杂问题时，我们的基本方法是分治法（分而治之，Divide and Conquer）
- 所谓分治法，就是：
  - 把大问题分解成（相对独立的）小问题
  - 逐个解决小问题，然后最终解决原始问题
- 在解决小问题时，同样可以应用分治法，对问题进一步的分解。
- 例如：举办新年晚会
  - 分解成：准备节目、准备场地、准备设备、准备主持稿等
  - 准备节目又可以进一步分解成报送节目、挑选节目、排练、彩排等

6.1.2 自顶向下与模块化（2）



这样问题的分解结果就像是一颗自上而下生长出来的树，因而称为“自顶向下”(Top-Down)

6.1.2 自顶向下与模块化（3）

- 显然，自顶而下的写程序方法也自然要求我们把每个小问题的解决步骤写成一个个相对独立的程序单位。
- 这种先把程序分解成一个个相对独立的小单位（零件），然后组装在一起的写程序哲学就是所谓的模块化。
- 在Python中，最基本的相对独立的程序（功能）单位就是函数（function）
- 注意：
  - function在英语中也有“功能”的含义。因此当初翻译成“功能单位"也许更加贴切些。

6.2 函数

- 在Python中，函数就是一个最基本的功能单位。
- 每个函数的基本作用就是：根据给定的前提条件(输入或者参数parameters），进行处理，最后返回处理的结果（输出）
- 因此，我们使用函数的基本形式是：
  - x=函数名(输入参数)　#函数有返回值（如len())
  - 函数名(输入参数)　　#函数没有返回值（如print())
- 当函数很多时，函数名重复或者说冲突就成了问题。
- 为了解决这个问题，Python引入了更高一层的组织单位:模块（Module)
  - 一个模块中可以包含多个函数
  - 不同模块下的同名函数互不冲突

6.2.1 导入命令

- 除了print()、input()、int()、float()、str()、len()等十多个内置函数(built-in functions) 外，python自带的大多数标准库函数都放在模块中。
- 要使用模块中的函数，需要在程序中先进行导入。
- 导入可以有两种形式：
  - 导入模块
  - 导入模块中的特定函数

6.2.1.1 导入模块

```
import easygraphics.dialog

easygraphics.dialog.show_message("hello!")
```

导入easygraphics.dialog模块，调用其中的show_message函数

```
import easygraphics.dialog as dlg

dlg.show_message("hello!")
```

导入easygraphics.dialog模块，将其在程序中命名为dlg，
并调用其中的show_message函数

6.2.1.2 导入特定函数

```
from easygraphics.dialog import show_message

show_message("hello!")
```

导入easygraphics.dialog模块中的show_message函数，然后调用

```
from easygraphics.dialog import *

show_message("hello!")
```

导入easygraphics.dialog模块中的所有函数，
然后调用show_message函数

注意：这种方法容易造成函数名冲突，因此应慎用

6.3 自定义函数

- 除了使用别人已经定义的函数外，我们也需要能够定义（define）自己的函数。
- python中函数定义的基本形式如下：

```
def 函数名(参数列表):
    函数体
```

- 其中：
  - 函数名：函数的名称，其命名规则见1.1节。通常使用（英文）动词短语
  - 参数列表：逐一列出函数接收的所有输入参数。参数间用","隔开
  - 函数体：由1条或多条语句组成。与if一样，用完全相同的缩进来表示函数体包含哪些语句。

6.3.1 自定义函数实例1

- 例6-1：超市的收银POS系统需要有打折促销的功能。请编写一个函数，能够计算商品9折后的价格。
- 和写程序类似，写函数的第一步是想清楚下列问题（请同学们回答）：
  1. 需要哪些参数（输入数据或信息）？其格式是什么样的?
  2. 是否需要返回处理结果？应返回什么形式的结果?
  3. 应该叫什么名字?
  4. 如何进行处理?

6.3.1 自定义函数实例1(2)

- 示范思路：
  1. 需要哪些参数？
     ① 一个输入：未打折的价格（原价），格式为浮点数(float)
  2. 是否需要返回处理结果？应返回什么形式的结果？
     ① 需要，应返回打折后的价格（浮点数）
  3. 如何命名？
     ① `cals_sales_price()`
  4. 如何进行处理？将原价乘以0.9即可

6.3.1 自定义函数实例1(3)

```
import easygraphics.dialog as dlg


def cals_sales_price(price):
    sales = 0.9 * price
    return sales


price = int(dlg.get_string("请输入原始价格:"))
sales_price = cals_sales_price(price)
dlg.show_message(f"打折后价格为{sales_price}")
```

6-1.打9折函数.py

注意**return**语句，其作用是结束函数处理，并返回处理的结果

6.3.2 防卫式编程

- 想一想，如果输入的价格是负数，结果会怎样？
  - 错误的原始或者中间结果会一路传导到最终的结果处。想找到错误的根源会很繁琐！
- 因此，如果程序比较复杂，或者多人协作开发时，应在每个函数的开始处判断输入的参数是否正确！
- 这种编程习惯我们称为“**防卫式编程**”（Defensive Programming）
  - 过马路时，即使是绿灯，也要先看清楚两边有没有闯红灯的车再过！
  - 写程序也一样，要尽量把错误抵挡在自己的代码之外！

6.3.2 防卫式编程（2）

```
import easygraphics.dialog as dlg

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

price=-20
sales_price=calc_sales_price(price)
dlg.show_message(f"打折后价格为{sales_price}")
```

6-2.防卫式编程-打9折函数.py

注意**raise**语句，其作用是结束函数处理并抛出一个异常（Exception）
python收到异常后会停止程序的执行，并提示错误

6.3.3 docstring

- 为了让用户更好的使用你的函数或者程序，通常我们应该为函数和程序编写对应的说明书
- 如果程序代码和说明书分别写在不同的地方，随着后续维护的进行，二者逐渐就会不一致：
  - 改代码时忘了改说明书！
- 因此，最好能将说明和代码写在一起！
- python的docstring提供了这个功能

6.3.3 docstring(2)

```
def calc_sales_price(price):
    """
    计算9折后的价格

    :param price: 折前价格
    :return:  折后价格
    """
    if price < 0:
        raise ValueError("price should not < 0!")
    sales = 0.9 * price
    return sales
```

完整程序见 6-3.docstring-打9折函数.py

6.3.4 同时返回多值

- 在python中，可以同时返回多个值
- 例：某电商网站，规定满200可以打95折，满500可以打9折。请编写一个函数，根据购买的商品总金额，计算并返回折扣数和折扣后的应付款金额。
- 思考并回答：
  - 该函数应该有哪些输入？（题目中已经给明）
  - 该函数是否应有返回值？应该返回什么？（题目中已经给明）
  - 如何进行处理/计算?
  - 给函数取什么名字?

6.3.4 同时返回多值(2)

- 该函数的输入：商品总金额
- 该函数的输出：（两个）折扣数、折后金额
- 处理步骤：
  1. 使用if语句来决定折扣数
  2. 折后金额=折扣数*商品总金额
  3. 返回折扣数和折后金额

6.3.4 同时返回多值(3)

```
def calc_discount(total):
    """
    计算折扣数和折后金额

    :param total: 购物金额
    :return: 折扣数,折后金额
    """
    if total<0:
        raise ValueError('total should not < 0!')
    if total>=500:
        discountRate=0.9
    elif total>=200:
        discountRate=0.95
    amount=total*discountRate
    return discountRate,amount
```

计算折扣函数（紫色部分为docstring，棕色部分为防卫式编程代码）
完整程序见"6-4.折扣计算.py"

6.3.5 关于函数名

- Python代码规范中建议：
  - 函数名全小写，用下划线连接各单词，如
    - is_odd()
    - get_area()
  - 避免使用"l","o"等容易看错的单个字母作为函数名或者函数名的一部分，如is_l, name_o等。
- 补充建议：使用动词或者动词短语来命名函数，避免使用名词或形容词短语，如：
  - is_valid()
  - get_area()

本章重点

- 变量的概念和使用
- 算术运算和表达式的使用
- 赋值的使用
- print()和input()函数的使用
- 关系运算和逻辑运算
- if语句
- 什么是字符编码和字符集。基本字符串运算
- 编写自定义函数
- 自顶向下的问题解决方法；模块化程序设计；
- 什么是防卫式编程， docstring

本章练习

- 见练习《第一章 结构化程序设计基础》
- 至少应完成前5道练习
- 有余力的同学可以完成全部8道练习