# 5.程序控制语句

## 5.1.运算符

非顺序式的程序控制，往往需要根据一定的条件，决定程序运行的路线。因此，我们首先来认识一下运算符。

### 5.1.1.比较运算

比较运算符有>、<、>=、<=、==、!=，返回值为布尔类型

a = 10

b = 8

print(a > b) *# 大于*

print(a < b) *# 小于*

print(a >= b) *# 大于等于*

print(a <= b) *# 小于等于*

print(a == b) *# 等于*

print(a != b) *# 不等于*

True

False

True

False

False

True

非空判断

ls = [1]

**if** ls: *# 数据结构不为空、变量不为0、None、False 则条件成立*

print("非空")

**else**:

print("空的")

非空

### 5.1.2.逻辑运算

与、或、非

a = 10

b = 8

c = 12

print((a > b) **and** (b > c)) *# 与*

print((a > b) **or** (b > c)) *# 或*

print(**not**(a > b)) *# 非*

False

True

False

复合逻辑运算的优先级  
 非 > 与 > 或

print(**True** **or** **True** **and** **False**)

True

print((**True** **or** **True**) **and** **False**)

False

### 5.1.3.存在运算

元素 in 列表/字符串

cars = ["BYD", "BMW", "AUDI", "TOYOTA"]

print("BMW" **in** cars)

print("BENZ" **in** cars)

True

False

元素 not in 列表/字符串

print("BMW" **not** **in** cars)

print("BENZ" **not** **in** cars)

False

True

## 5.2.分支结构-if语句

### 5.2.1.单分支

if 条件：  
 缩进的代码块

age = 8

**if** age > 7:

print("孩子，你该上学啦！")

孩子，你该上学啦！

### 5.2.2.二分支

if 条件：  
 缩进的代码块  
 else：  
 缩进的代码块

age = 6

**if** age > 7:

print("孩子，你该上学啦！")

**else**:

print("再玩两年泥巴！")

再玩两年泥巴！

### 5.2.3.多分支

if 条件：  
 缩进的代码块  
 elif 条件：  
 缩进的代码块  
 elif 条件：  
 缩进的代码块  
 ...  
 else：  
 缩进的代码块

不管多少分支，最后只执行一个分支

age = 38

**if** age < 7:

print("再玩两年泥巴")

**elif** age < 13:

print("孩子，你该上小学啦")

**elif** age < 16:

print("孩子，你该上初中了")

**elif** age < 19:

print("孩子，你该上高中了")

**elif** age < 23:

print("大学生活快乐")

**elif** age < 60:

print("辛苦了，各行各业的工作者们")

**else**: *# 有时为了清楚，也可以写成elif age >= 60:*

print("享受退休生活吧")

辛苦了，各行各业的工作者们

### 5.2.4.嵌套语句

题目：年满18周岁，在非公共场合方可抽烟，判断某种情形下是否可以抽烟

age = eval(input("请输入年龄"))

**if** age > 18:

is\_public\_place = bool(eval(input("公共场合请输入1，非公共场合请输入0")))

print(is\_public\_place)

**if** **not** is\_public\_place:

print("可以抽烟")

**else**:

print("禁止抽烟")

**else**:

print("禁止抽烟")

请输入年龄16

禁止抽烟

## 5.3.循环语句

### 5.3.1.for 循环

**格式：**

for 元素 in 可迭代对象：  
   执行语句

**执行过程：**

从可迭代对象中，依次取出每一个元素，并进行相应的操作

1、直接迭代——列表[ ]、元组( )、集合{ }、字符串" "

graduates = ("李雷", "韩梅梅", "Jim")

**for** graduate **in** graduates:

print("Congratulations, "+graduate)

Congratulations, 李雷

Congratulations, 韩梅梅

Congratulations, Jim

2、变换迭代——字典

students = {201901: '小明', 201902: '小红', 201903: '小强'}

**for** k, v **in** students.items():

print(k, v)

**for** student **in** students.keys():

print(student)

201901 小明

201902 小红

201903 小强

201901

201902

201903

3、range()对象

res=[]

**for** i **in** range(10000):

res.append(i\*\*2)

print(res[:5])

print(res[-1])

[0, 1, 4, 9, 16]

99980001

res=[]

**for** i **in** range(1,10,2):

res.append(i\*\*2)

print(res)

[1, 9, 25, 49, 81]

**循环控制：break 和 continue**

break 结束整个循环

product\_scores = [89, 90, 99, 70, 67, 78, 85, 92, 77, 82] *# 1组10个产品的性能评分*

*# 如果低于75分的超过1个，则该组产品不合格*

i = 0

**for** score **in** product\_scores:

**if** score < 75:

i += 1

**if** i == 2:

print("产品抽检不合格")

**break**

产品抽检不合格

continue 结束本次循环

product\_scores = [89, 90, 99, 70, 67, 78, 85, 92, 77, 82] *# 1组10个产品的性能评分*

*# 如果低于75分,输出警示*

print(len(product\_scores))

**for** i **in** range(len(product\_scores)):

**if** product\_scores[i] >= 75:

**continue**

print("第**{0}**个产品，分数为**{1}**，不合格".format(i, product\_scores[i]))

10

第3个产品，分数为70，不合格

第4个产品，分数为67，不合格

**for 与 else的配合**

如果for 循环全部执行完毕，没有被break中止，则运行else块

product\_scores = [89, 90, 99, 70, 67, 78, 85, 92, 77, 82] *# 1组10个产品的性能评分*

*# 如果低于75分的超过1个，则该组产品不合格*

i = 0

**for** score **in** product\_scores:

**if** score < 75:

i+=1

**if** i == 2:

print("产品抽检不合格")

**break**

**else**:

print("产品抽检合格")

产品抽检不合格

### 5.3.2.while 循环

**为什么要用while 循环**

代码可能需要重复执行，可是又不知道具体要执行多少次

**while循环的格式**

while 判断条件：  
   执行语句  
 条件为真，执行语句  
 条件为假，while 循环结束

猜数字游戏：

albert\_age = 18

guess = int(input(">>:"))

**while** guess != albert\_age:

**if** guess > albert\_age :

print("猜的太大了，往小里试试...")

**elif** guess < albert\_age :

print("猜的太小了，往大里试试...")

guess = int(input(">>:"))

print("恭喜你，猜对了...")

**while与标示**

可以立一个布尔类型的标示，通过在while中的判断进行终止

albert\_age = 18

flag = **True** *# 布尔类型*

**while** flag:

guess = int(input(">>:"))

**if** guess > albert\_age :

print("猜的太大了，往小里试试...")

**elif** guess < albert\_age :

print("猜的太小了，往大里试试...")

**else**:

print("恭喜你，猜对了...")

flag = **False** *# 当诉求得到满足，就让风向变一下*

**while 与循环控制 break、continue**

break跳出循环，continue跳过本次循环

albert\_age = 18

**while** **True**:

guess = int(input(">>:"))

**if** guess > albert\_age :

print("猜的太大了，往小里试试...")

**elif** guess < albert\_age :

print("猜的太小了，往大里试试...")

**else**:

print("恭喜你，猜对了...")

**break** *# 当诉求得到满足，就跳出循环*

*# 输出10以内的奇数*

i = 0

**while** i < 10:

i += 1

**if** i % 2 == 0:

**continue** *# 跳出本次循环，进入下一次循环*

print(i)

1

3

5

7

9

**while与else**

如果while 循环全部执行完毕，没有被break中止，则运行else块

count = 0

**while** count <= 5 :

count += 1

print("Loop",count)

**else**:

print("循环正常执行完啦")

Loop 1

Loop 2

Loop 3

Loop 4

Loop 5

Loop 6

循环正常执行完啦

**while两个例子**

删除列表中的特定值

pets = ["dog", "cat", "dog", "pig", "goldfish", "rabbit", "cat"]

**while** "cat" **in** pets:

pets.remove("cat")

pets

['dog', 'dog', 'pig', 'goldfish', 'rabbit']

将未读书籍列表中书名分别输出后，存入已读书籍列表

not\_read = ["红楼梦", "水浒传", "三国演义", "西游记"]

have\_read = []

**while** not\_read: *# not\_read非空，循环继续，否则中止*

book = not\_read.pop()

have\_read.append(book)

print("我已经读过《**{}**》了".format(book))

print(not\_read)

print(have\_read)

我已经读过《西游记》了

我已经读过《三国演义》了

我已经读过《水浒传》了

我已经读过《红楼梦》了

[]

['西游记', '三国演义', '水浒传', '红楼梦']

## 5.4.控制语句注意问题

**尽可能减少多层嵌套**

可读性差，容易把人搞疯掉

**if** 条件:

执行语句

**if** 条件:

执行语句

**if**...

**避免死循环**

条件一直成立，循环永无止境

*# while True:*

*# print("天地之渺渺，时间之无限")*

**封装过于复杂的判断条件**

如果条件判断里的表达式过于复杂，出现了太多的 not/and/or等，导致可读性大打折扣。

考虑将条件封装为函数

a, b, c, d, e = 10, 8, 6, 2, 0

**if** (a > b) **and** (c >d) **and** (**not** e):

print("我已经晕鸟")

*# 进行封装*

numbers = (10, 8, 6, 2, 0)

**def** judge(num):

a, b, c, d, e = num

x = a > b

y = c > d

z = **not** e

**return** x **and** y **and** z

**if** judge(numbers):

print("就是这个feel，biu倍儿爽")

就是这个feel，biu倍儿爽

## 5.5.作业练习

运算符：

1、以下条件测试的结果分别是什么：

\* d = {}

if d:

print("非空")

else:

print("空的")

\* True or False and False

\* foods = ["bread", "fish", "potato"]

"breads" in foods

"tomato" in foods

分支语句：

2、试编写一个程序，实现如下功能：

从黑箱中摸到一个球（无需实现，直接手动赋值），按照球的不同颜色分配不同的奖金。

\*如果摸到黑球，获得奖金0元；

\*如果摸到白球，获得奖金10元；

\*如果摸到蓝球，获得奖金20元；

\*如果摸到粉球，获得奖金30元；

\*如果摸到紫球，获得奖金50元；

\*如果摸到彩球，获得奖金100元；

遍历循环-for

3、用两层嵌套for循环，打印输出九九乘法表。

条件循环-while

4、创建一个while循环，通过input，动态输入高三.二班学生学号、姓名和性别。

\*以学号为键，姓名和性别为值，将学生信息存储在字典students\_of\_grade3\_class2中。

\*当通过input 输入字母“Q”或“q”时，程序结束。

\*程序结束后，对学生信息进行遍历输出。

注意问题：

5、以下程序能否正常执行，如果不能，请说明原因：

i = 0

while i <= 5:

print(i)

6、为什么我们不希望嵌套层数过多？

7、如果条件测试过于复杂，通常会如何处理？

答案：

1. \* 空的

\* True

\* False

False

2.

**import** **random** **as** **ran**

ball = ["白球", "黑球", "蓝球", "粉球", "紫球", "彩球"].pop(ran.randrange(6))

print(ball)

money = 0

**if** ball == "黑球":

money = 0

**elif** ball == "白球":

money = 10

**elif** ball == "蓝球":

money = 20

**elif** ball == "粉球":

money = 30

**elif** ball == "紫球":

money = 50

**elif** ball == "彩球":

money = 100

print(money)

3.

**for** i **in** range(1, 10):

**for** j **in** range(1, i + 1):

print("**{}** × **{}** = **{}**".format(j, i, i \* j), end=" ")

print()

1 × 1 = 1

1 × 2 = 2 2 × 2 = 4

1 × 3 = 3 2 × 3 = 6 3 × 3 = 9

1 × 4 = 4 2 × 4 = 8 3 × 4 = 12 4 × 4 = 16

1 × 5 = 5 2 × 5 = 10 3 × 5 = 15 4 × 5 = 20 5 × 5 = 25

1 × 6 = 6 2 × 6 = 12 3 × 6 = 18 4 × 6 = 24 5 × 6 = 30 6 × 6 = 36

1 × 7 = 7 2 × 7 = 14 3 × 7 = 21 4 × 7 = 28 5 × 7 = 35 6 × 7 = 42 7 × 7 = 49

1 × 8 = 8 2 × 8 = 16 3 × 8 = 24 4 × 8 = 32 5 × 8 = 40 6 × 8 = 48 7 × 8 = 56 8 × 8 = 64

1 × 9 = 9 2 × 9 = 18 3 × 9 = 27 4 × 9 = 36 5 × 9 = 45 6 × 9 = 54 7 × 9 = 63 8 × 9 = 72 9 × 9 = 81

4.

students\_of\_grade3\_class2 = {}

**while** **True**:

id, name, sex = input("请输入高三二班学生学号、姓名、性别：").split(" ")

students\_of\_grade3\_class2[id] = (name, sex)

**if** id **in** ["Q", "q"]:

**break**

**for** k, v **in** students\_of\_grade3\_class2.items():

print(k, v)

请输入高三二班学生学号、姓名、性别：1 小明 男

请输入高三二班学生学号、姓名、性别：2 小张 男

请输入高三二班学生学号、姓名、性别：q q q

1 ('小明', '男')

2 ('小张', '男')

1. 不能，死循环
2. 影响性能和可读性
3. 封装为函数