

# Python流程控制

# 流程控制

- 数据是被加工处理的原材料,而**处理过程**要用**流程控制**结构来描述
  - 类比:烹调=食材+**烹制过程**
    - 烹制过程:先炒再煮;如果淡了则加盐;反复翻炒**5**分钟;  
...
- 常见的流程控制结构
  - 顺序,跳转,分支,循环,子程序等
- 好的流程:结构清晰,**易理解,易验证,易维护**

---

# 提纲

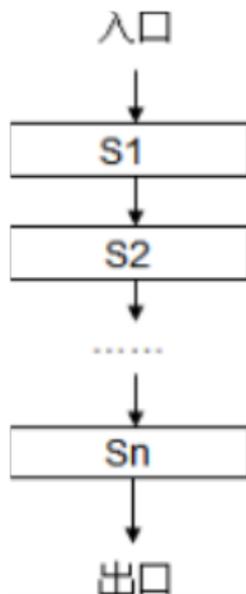
- 顺序
- 分支
  - **if/else**
- 循环
  - **for**
  - **while**

# 提纲

- 顺序
- 分支
  - **if/else**
- 循环
  - **for**
  - **while**

# 顺序控制结构

- 按语句的**自然先后**顺序执行



# 编程实例

· 温度转换程序:

· 华氏转换成摄氏

```
f = input("Temperature in degrees Farenheit: ")
```

```
c = (f-32) * 5.0 / 9
```

```
print "Temperature in degrees Celsius:",c
```

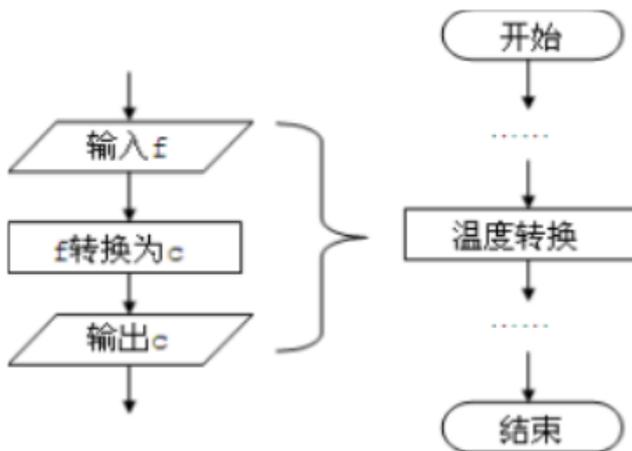
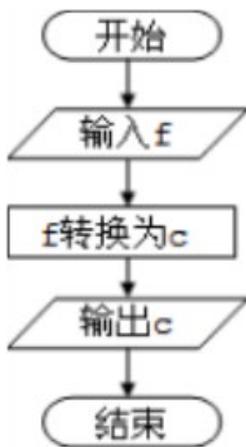
```
Temperature in degrees Farenheit: 89
```

```
Temperature in degrees Celsius: 31.6666666667
```

# 编程实例

流程图:用标准化的图形符号来表示程序步骤

流程图中的步骤可以是不同抽象级的



# 提纲

- 顺序
- 分支
  - **if/else**
- 循环
  - **for**
  - **while**

# 代码缩进

- 代码块通过缩进对齐表达代码逻辑而不是使用大括号，因为有了额外的字符，程序的可读性更高。
  - 一是简洁，
  - 二是可读性好

```
if expression:  
    if_suite  
else:  
    else_suite
```

# If语句

**if** 表达式1:

表达式为真需要执行的语句段

**elif** 表达式2:

表达式1为假，表达式2为真执行的语句段

**else:**

全部为假

注: **elif**和**else**都可省略，**elif**可有多个

# 分支控制结构

- 可以选择不同的执行路径

- 单分支结构

**if** <条件>:

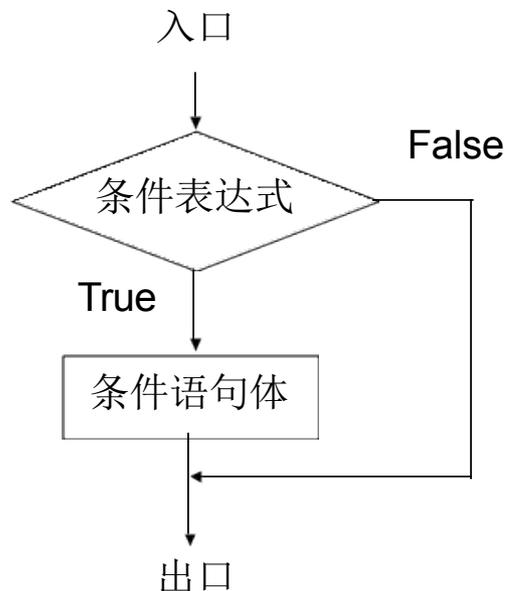
    <语句体>

- <条件>: 布尔表达式

- <语句体>: 语句序列.

- 左边需要“缩进”四个空格

- 语义: 计算<条件>的真假. 若为**真**, 则执行<语句体>, 并把控制转向下一条语句; 若为**假**, 则直接把控制转向下一条语句.



# 布尔表达式

- <条件>是一个布尔表达式
  - 结果为布尔值**True**或**False**
- 简单布尔表达式:
  - <表达式1> <关系运算符> <表达式2>
  - 关系运算: <, <=, ==, >=, >, !=
    - 数值比较
    - 字符串比较: 按字符序
      - 字符序由编码(ASCII等)决定. 如:大写字母在小写字母前.
    - 列表,元组的比较

# 布尔表达式

- 复杂布尔表达式:

<布尔表达式1><布尔运算><布尔表达式2>

- 布尔运算:**and, or, not**

<布尔表达式1> **and** <布尔表达式2>

<布尔表达式1> **or** <布尔表达式2>

**not** <布尔表达式>

# and的定义

- **and**表示“并且”:

**$P$  and  $Q$** 为真如果 **$P$** 和 **$Q$** 都为真

- 真值表见右

· 例

>>> **(3 > 2) and (2 > 1)**

**True**

>>> **(3 > 2) and (2 > 3)**

**False**

<b><math>P</math></b>	<b><math>Q</math></b>	<b><math>P</math> and <math>Q</math></b>
<b>F</b>	<b>F</b>	<b>F</b>
<b>F</b>	<b>T</b>	<b>F</b>
<b>T</b>	<b>F</b>	<b>F</b>
<b>T</b>	<b>T</b>	<b>T</b>

# or的定义

- **or**表示“或者”:

**$P$  or  $Q$** 为假如果 **$P$** 和 **$Q$** 都为假

- 真值表见右

- 与日常用语中互斥的“或”不同!

- 例

```
>>> (3 > 2) or (3 <= 2)
```

```
True
```

```
>>> (2 > 3) or (2 > 4)
```

```
False
```

<i>P</i>	<i>Q</i>	<i>P or Q</i>
<b>F</b>	<b>F</b>	<b>F</b>
<b>F</b>	<b>T</b>	<b>T</b>
<b>T</b>	<b>F</b>	<b>T</b>
<b>T</b>	<b>T</b>	<b>T</b>

# not的定义

- **not**表示“否定”:

**not  $P$** 为真如果 $P$ 为假.

- 真值表见右

- 例

```
>>> not 3 > 2
```

```
False
```

```
>>> not not 3 > 2
```

```
True
```

$P$	<b>not <math>P</math></b>
<b>T</b>	<b>F</b>
<b>F</b>	<b>T</b>

# 布尔运算符的优先级

- **not**最高, **and**次之, **or**最低
  - **Q: a or not b and c**何意?
  - **A: a or ((not b) and c)**
  - 最好**使用括号!**

# 例：一局乒乓球比赛的结束

- 双方任何人先得**11**分

`a == 11 or b == 11`

- 更准确的：一方至少要多**2**分才胜

`(a >= 11 and a - b >= 2) or`

`(b >= 11 and b - a >= 2)`

或者写成

`(a >= 11 or b >= 11) and abs(a - b) >= 2`

# 编程实例

· 温度转换程序:

· 增加热浪告警功能

```
f = input("Temperature in degrees  
Fahrenheit: ")  
c = (f - 32) * 5.0 / 9  
print "Temperature in degrees Celsius:", c  
if c > 35:  
    print "Warning: Heat Wave!"
```

# 编程实例

· 温度转换程序:

· 增加热浪和寒潮告警功能

```
f = input("Temperature in degrees Farenheit: ")
c = (f - 32) * 5.0 / 9
print "Temperature in degrees Celsius:", c
if c >= 35:
    print "Warning: Heat Wave!"
if c <= -6:
    print "Warning: Cold Wave!"
```

# 两路分支结构

语法

```
if <条件>:  
    <if-语句体>
```

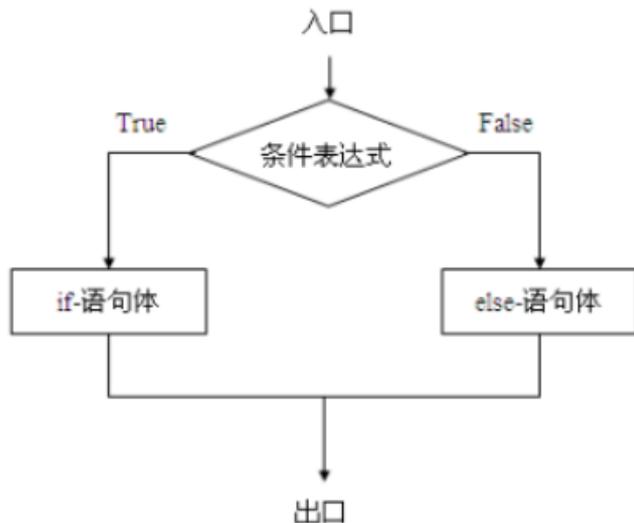
```
else:
```

```
    <else-语句体>
```

**if**和**else**是非此即彼的关系。

下列语句对吗?

```
if c >= 35:  
    print "Warning: Heat Wave!"  
else:  
    print "Warning: Cold Wave!"
```



# 多路分支:嵌套if-else

· **if**语句可以嵌套

· 多重嵌套不好

· 难读

· 代码松散

```
if c >= 35:
```

```
    print "Warning: Heat wave!"
```

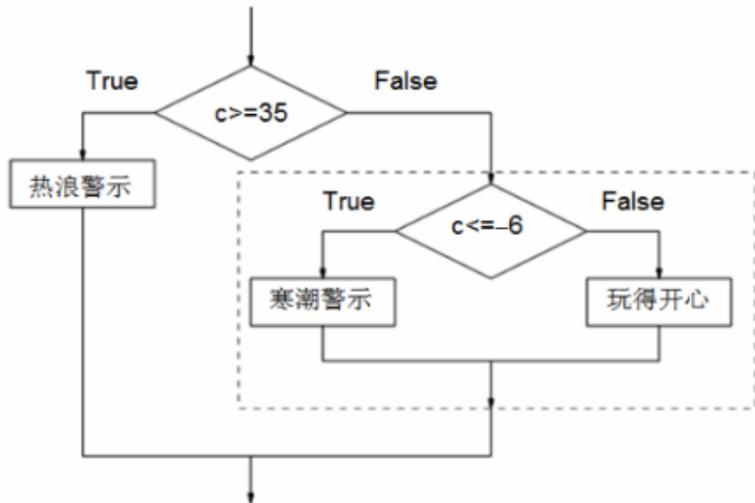
```
else:
```

```
    if c <= -6:
```

```
        print "Warning: Cold Wave!"
```

```
    else:
```

```
        print "Have fun!"
```



# 多路分支:if-elif-else结构

## 语法

```
if <条件1>:  
    <情形1语句体>  
elif <条件2>:  
    <情形2语句体>  
...  
elif <条件n>  
    <情形 n语句体 >  
else <其他情形语句体>
```

语义:找到**第一个为真的条件并执行**对应语句序列,控制转向下一条语句;若无,则执行**else**下的语句序列,控制转向下一条语句.

# 编程实例

## · 温度转换程序

```
f = input("Temperature in degrees Farenheit: ") c =  
(f - 32) * 5.0 / 9  
print "Temperature in degrees Celsius:", c  
if c >= 35:  
    print "Warning: Heat Wave!"  
elif c <= -6:  
    print "Warning: Cold Wave!"  
else:  
    print "Have fun!"
```

# 编程实例

```
Please input your score:98  
Congratulations, you passed the examine.  
Done
```

输入成绩，判断是否通过考试

```
pass_level = 60;  
score = int(raw_input("Please input your score:"))  
if score >= pass_level:  
    print 'Congratulations, you passed the examine.' #  
New block starts here  
else:  
    print 'No, you failed to pass the eaamine.'  
  
print 'Done'  
# This last statement is always executed, after the if  
statement is executed
```

# 编程实例

根据考试成绩将成绩分为**A,B,C,D**四档

```
score = int(raw_input("Please input your score:"))
level = 'D'
if score >= 80:
    level = 'A'
elif score >= 70:
    level = 'B'
elif score >= 60:
    level = 'C'
print 'Your Score:', score, "Your level", level
```

Please input your score:77  
Your Score: 77 Your level B

# 编程实例

输入  $x, y$ ，判断属于第几象限

```
请输入X:1  
请输入Y:1  
(1,1)在第一象限
```

- 提示:
1. 使用三个判断，可使用判断的嵌套
  2. `print`语句可使用格式化字符串，如  
`print "%d*%d=%d"%(i,j,i*j)`
  2. 在程序首行加入`#encoding=utf-8`可使用汉字

# 编程实例

```
37 vx = int(raw_input("请输入X:"))
38 vy = int(raw_input("请输入Y:"))
39 level = 'D'
40 if vx>=0 and vy>=0:
41     print "(%d,%d)在第一象限"%(vx,vy)
42 elif vx>=0 and vy<0:
43     print "(%d,%d)在第二象限"%(vx,vy)
44 elif vx<0 and vy<0:
45     print "(%d,%d)在第二象限"%(vx,vy)
46 else:
47     print "(%d,%d)在第二象限"%(vx,vy)
```

请输入X:1

请输入Y:1

(1,1)在第一象限

# 程序运行错误的处理

- 程序编译正确,但在运行时发生错误。
  - 例如:**a/b**语法没错,但运行时万一**b=0**,就会出错
  - 又如:**输入数据的类型和个数不对,列表索引越界,等等**
- 编程时如果没有考虑运行错误,程序就很容易运行崩溃,非正常结束。
- 好的程序应该是**健壮的**。

# 编程实例

- 求一元二次方程根:

```
import math
```

```
a, b, c = input("Enter (a, b, c): ")
```

```
discRoot = math.sqrt(b*b - 4*a*c)
```

```
root1 = (-b + discRoot) / (2*a) root2 =  
(-b - discRoot) / (2*a)
```

```
print "The solutions are: ", root1, root2
```

- 运行程序,输入1,2,3
- 程序崩溃!

# 提高健壮性:使用错误检测代码

- 错误检测代码:利用**if**判断是否发生了某种运行错误.

```
do_sth()
```

```
if some-error:
```

```
    do_sth_else()
```

# 编程实例

· 解方程程序的改进: Enter (a, b, c): 1,2,3  
The equation has no real roots!

```
import math
a, b, c = input("Enter (a, b, c): ")
discrim = b * b - 4 * a * c
if discrim >= 0:
    discRoot = math.sqrt(discrim)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print "The solutions are:", root1, root2
else:
    print "The equation has no real roots!"
```

# 提高健壮性:利用函数返回码

- 函数中有**检测代码**,执行正常与否可利用返回值作为**标志码**.
- 调用者无条件调用函数,并检测返回值.
  - 例如,为了解决**sqrt**函数的问题,设计**robustSqrt()**:

```
def robustSqrt(x):  
    if x < 0:  
        return -1  
    else:  
        return math.sqrt(x)
```

- 则程序中可以这样检测

```
if robustSqrt(b*b - 4*a*c) < 0:...
```



# 异常处理

- 能否既健壮,又不破坏原来算法的清晰?

- **异常处理机制**

- 程序运行时如果出错则"抛出"一个"异常";
- 程序员能编写代码"捕获"并处理异常;
- 可使程序不因运行错误而崩溃,尽量使用户不受意外结果的困扰.



# Python的缺省异常处理

- 程序运行出错时,抛出的异常被**Python**系统自动处理
  - 基本就是中止程序的执行并显示一些错误信息.

```
>>> a = "Hello"
```

```
>>> print a[5]
```

```
Traceback (most recent call last):      File  
    "<stdin>", line 1, in <module>
```

```
IndexError: string index out of range
```

# 程序员自定义异常处理

- **Python**提供**try-except**语句,可用来自定义异常处理代码.

```
>>> a = "Hello"
```

```
>>> try:
```

```
    print a[5]
```

```
except IndexError:
```

```
    print "Index wrong!"
```

**Index wrong!**

# 异常处理机制的优点

- 既保持核心算法的清晰,又能提高程序的健壮性.

算法清晰  
但不健壮:

```
doStep1()  
doStep2() doStep3()  
()
```

健壮但算法不清晰:

```
if doStep1() == ERROR:  
    错误处理代码1  
elif doStep2() == ERROR:  
    错误处理代码2  
elif doStep3() == ERROR:  
    错误处理代码3
```

算法清晰且健壮:

try:

```
doStep1()  
doStep2() doStep3()  
()
```

except ERROR:

错误处理代码

# 异常处理语句

- 可以对不同类型的错误分别指定处理代码

**try:**

    <正常程序体>

**except** <错误类型1>:

    <异常处理程序1>

...

**except** <错误类型n>:

    <异常处理程序n>

**except:**

    <其他异常的处理程序>

# 编程实例

- 解方程程序的改进:用异常处理语句来捕获 **math.sqrt**的溢出错误

```
import math
try:
    a, b, c = input("Enter (a, b, c): ")
    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print "The solutions are:", root1, root2
except ValueError:
    print "The equation has no real roots!"
```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：<https://d.book118.com/355233014342011233>