

数据情况的检查、逻辑判断

■ 输入来自程序外部

- 输入可能不符合程序的需要

- 应在实际计算之前，检查程序所获得的输入

 - 输入正确时完成计算

 - 输入数据不正确时输出错误信息

} 要求程序能够进行判断，并分情况处理

■ 比较数据对象、检查条件既是做逻辑判断

- 逻辑判断的结果也是数据，用对象表示

 - Python 用逻辑类型 **bool** 表示

 - **bool** 类型只包含两个对象：**True** (真/关系成立)，**False** (假/关系不成立)

比较运算符、关系表达式

■ 比较运算符：判断数值之间关系的运算符

==	!=	<	<=	>	>=
等于	不等于	小于	小于等于	大于	大于等于

- 所有比较运算符的优先级相同，低于加减法运算符

■ 关系表达式

- 语法：由一个比较运算符和两个数值表达式构成

例：5 != 0、x + y > x**2

- 语义：判断两个数值之间的某种关系是否成立 (如成立则值为 **True**，否则值为 **False**)
- 做不同数值类型之间比较之前，先转换到相同类型

比较运算符、关系表达式

■ 比较运算符：判断数值之间关系的运算符

==	!=	<	<=	>	>=
等于	不等于	小于	小于等于	大于	大于等于

□ 所有比较运算符的优先级相同，低于加减法运算符


■ 关系表达式

```
>>> type(True)
<class 'bool'>
>>> 1 + 1 == 2
True
```

```
>>> x, y = 2**3, 3**2
>>> x > y
False
>>> 0.1 + 0.2 == 0.3
False
```

逻辑运算符

- 分别以关键字作为运算符，用于描述组合判断 (优先级由高至低排列)

否定运算	 not	not A 为真，当 A 为假
与运算	and	A and B 为真，当 A 和 B 都为真
或运算	or	A or B 为真，当 A 和 B 中至少一个为真

其中，**A** 和 **B** 是结果为逻辑值或者可作为逻辑值使用的表达式

- 逻辑运算符的优先级低于比较运算符

```
>>> (1 + 1 == 2) or (3 > 4)
```

```
True
```

```
>>> not True
```

```
False
```

```
>>> not (3 > 4)
```


```
True
```

```
>>> not 2.1**2 > 5.25 or 4.1 < 6.3 and 2.3*3.2 > 7
```

```
True
```

逻辑运算符

- 分别以关键字作为运算符，用于描述组合判断 (优先级由高至低排列)

否定运算	 not	not A 为真，当 A 为假
与运算	and	A and B 为真，当 A 和 B 都为真
或运算	or	A or B 为真，当 A 和 B 中至少一个为真

其中，**A** 和 **B** 是结果为逻辑值或者可作为逻辑值使用的表达式

- 逻辑运算符的优先级低于比较运算符

- **Python** 允许任意连续使用关系运算，

□ **a < b <= c** 相当于 **a < b and b <= c**

□ **a < b >= c** 相当于 **a < b and b >= c**

```
>>> 7 > 3 >= 3
True
>>> 12 < 23 < 22
False
>>> m, n = 4, 8
>>> 1 <= m < n <= 10
True
```

条件语句 (if 语句)

- **if 语句**：一种实现流程控制的程序结构，基于逻辑判断的结果执行不同操作

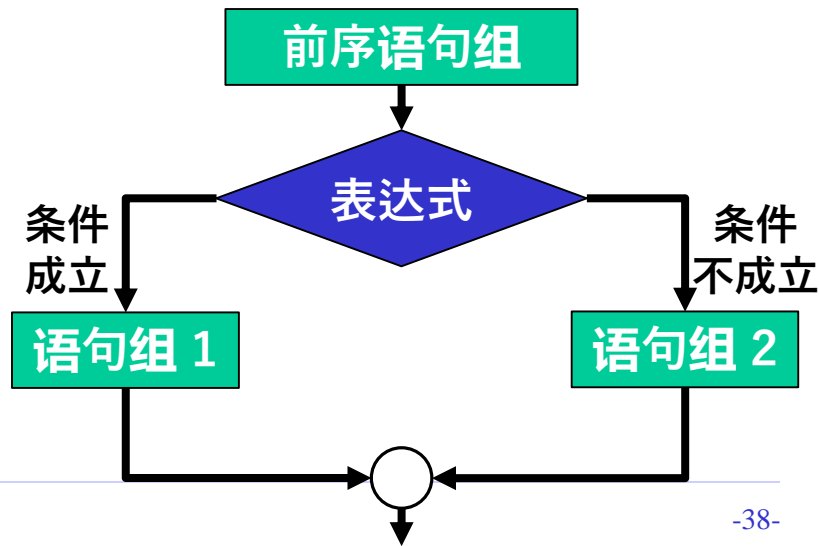
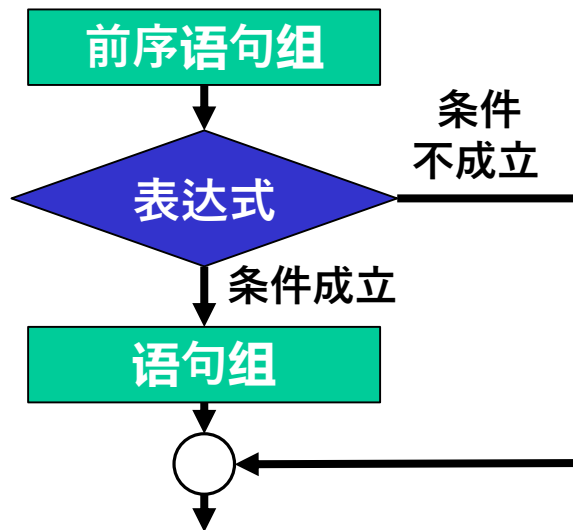
- 基本形式1：语法和语义

if 表达式：
语句组

- 基本形式2：语法和语义

if 表达式：
语句组 1
else:
语句组 2

- 实例：增加输入数据检查的三角形面积计算程序



if 语句和程序格式

- **if 段 / else 段**：从关键字 **if / else** 到相应语句组结束
 - 段的**头部**：**if** 表达式 : / **else** :
 - 段的**体**：语句组 (手册里称为 **suite**)
 - 执行语句组即是顺序地执行其中的 (一条或者多头) 语句
 - 如果语句组只有一条语句，可写在头部同一行 (不提倡)
- **Python** 严格规定复杂结构/组合结构的书写格式
 - 程序格式有语义作用 (区别于很多其他语言)
 - 同一层次的成分必须**垂直对齐**，下一层次的成分**统一缩进**
 - 语句组**总是**下一层次的成分
- 建议：直接使用 **IDE** 的默认对齐方式
 - 不要混用 **Tab** 和 **4** 空格缩进

if 语句

■ if 语句的一般形式 (多分支 if 语句)

if 表达式 :
 语句组 1
elif 表达式:
 语句组 2
... ..
else:
 语句组 n

细节:

1. 一个 **if** 段
2. 最后是可选的 **else** 段
3. 两者之间可以有任意多个 **elif** 段

□ 语句组里可有 if 语句 (即嵌套的 if 语句) 或其他程序结构

○ 对于嵌套的 if 语句, 代码缩进决定了 if 和 else 的配对

■ 实例: 交互式程序, 求一元二次方程的根, 方程系数由用户输入

逻辑类型和逻辑值

- 逻辑类型 **bool** 只有两个值 **True** 和 **False**
 - 关系表达式以及 **in**, **not in**, **is**, **is not** 运算符的值都是逻辑值
 - 主要用在 **if**、**while** 语句和条件表达式里，用做条件判断
- 逻辑值可以直接作为数据 (函数返回值、存入变量、作为复合数据对象的元素...)
 - 也可参与各种运算 (但并不常见，也不提倡)

```
>>> int(True)
1
>>> int(False)
0
>>> 5 * True
5
>>> 7 - False
7
```

```
>>> True / 4
0.25
>>> -True
-1
>>> True > -5
True
>>> False <= 0
True
```

逻辑判断

■ Python 允许将其他内置类型的对象用于逻辑判断和逻辑运算

□ 各种类型的下列值都表示假

- False;
- None;
- 各种数值类型里的零，包括：0、0.0、0.0+0.0j；
- 各种空序列，包括 ""、[]、()；
- 空字典{}，空集 set() 和 frozenset()。

□ 程序中，在要求逻辑值的位置，如果出现这些类型的其他值都作为真

例：

```
if not lst or lst1:  
    .....
```

```
>>> bool(0)  
False  
>>> bool(999)  
True
```

and/or/not 与短路求值

■ and / or / not 的确切语义：

- ❑ **a and b**: 首先求值表达式 **a**，如果 **a** 的值为假，则以 **a** 的值作为整个表达式的结果；否则，求值表达式 **b**，并以 **b** 的值作为整个表达式的结果
- ❑ **a or b**: 首先求值表达式 **a**，如果 **a** 的值为真，则以 **a** 的值作为整个表达式的结果；否则，求值表达式 **b** 并以 **b** 的值作为整个表达式的结果
- ❑ **not a**: 求值表达式 **a**，值为假得到 **True**，否则得到 **False**

■ 注意：

- ❑ 运算对象不一定是逻辑值，可以是其他能表示真假的对象
- ❑ **and/or** 表达式的结果不一定是 **True**、**False**，也可能是其他类型的对象

```
>>> 1 and 2
2
>>> 0 or 'b'
'b'
>>> 1 and []
[]
>>> not ''
True
```

and/or 与短路求值

■ 短路求值规则

- **ex_1 and ex_2 and ... and ex_n** 顺序求值 ex_1, ex_2, \dots, ex_n
 - 一旦某个 ex_i 求出的值是假，整个 **and** 表达式的值就是 ex_i 的值，余下的 ex 不再求值
- **ex_1 or ex_2 or ... or ex_n** 顺序求值 ex_1, ex_2, \dots, ex_n
 - 一旦某个 ex_i 求出的值是真，整个 **or** 表达式的值就是 ex_i 的值，余下的 ex 不再求值

■ 例:

```
if x > 0 and y/x > 1:  
    ... x ... y ...
```

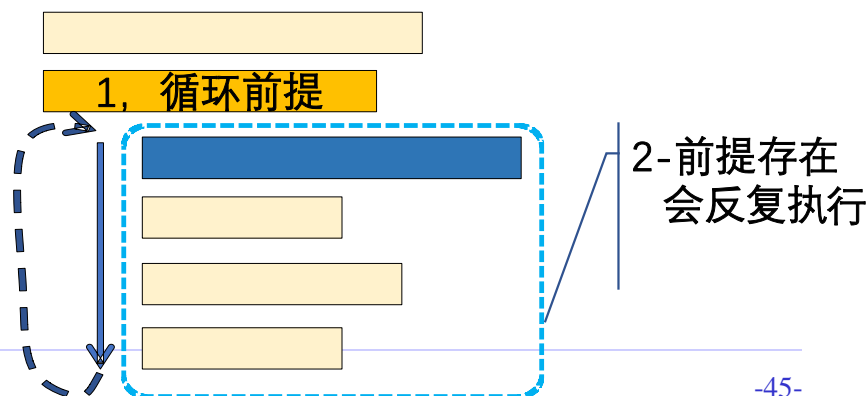
不会发生除零错误

```
if lst and lst[0] > 1:  
    ... ..
```

不会对空表取元素值

程序的控制

- 已介绍两种控制 (和相关执行方式)
 - 顺序结构 (语句的顺序排列, 描述**顺序执行**)
 - 分支结构 (if 语句, 描述根据条件**选择执行**)
- **直线型程序**: 只包含顺序结构的程序
 - **Python** 系统顺序地执行每条语句一次, 只有一条执行路径
- **分支程序**: 只包含 if 复合语句的程序
 - 程序里的每条基本语句最多执行一次
 - 处理不同数据时执行的语句序列 (执行路径) 可能不同, 但可以列举出程序里所有可能的执行路径
- **重复计算**是最基本的复杂计算, 各种高级编程语言都提供专门的结构来描述重复计算



循环语句，for 语句

■ 循环语句

- 实现代码的重复执行，执行中可能多次执行其成分语句组
- Python 提供两种循环语句：**for** 语句和 **while** 语句

■ **for 语句**：根据一个 **循环控制器** 来实现重复执行

□ 语法 (简单形式)

for 变量 in 可迭代对象： # **循环头部**：描述循环前提条件
语句组 # **循环体**

- **可迭代对象 (iterable)**：描述值的序列 (如字符串、表等)
- 在循环体中，可以使用**头部中的变量 (循环变量)**
- 语义：让**循环变量**按顺序地取 (即，遍历) “值序列” 中每一个值，并且**对每个值执行语句组一次**

for 循环语句

- 调用内置函数 **range** 得到一个描述“等差整数序列”的可迭代对象，有三种调用形式：

- **range(n)** 得到整数序列 0, 1, ..., n-1

- $n \leq 0$ 时，得到空序列，for 直接结束

- **range(m, n)** 得到整数序列 m, m+1, ..., n-1

- $n \geq m$ 时，得到空序列，for 直接结束 (左闭右开规则)

- **range(m, n, d)** 得到整数序列 m, m+d, m+2*d, ... (小于 n)

- d 为负整数时，序列值逐项减小但大于 n

注：参数
m, n, d
都是整数

- 实例1：等差整数序列求和

注：循环体里也可以写：

s += n # 原地更新

类似有 **-=**, ***=**, **/=**, **//=**, **%=**, ****=**

```
a = int(input("Start from: "))
b = int(input("End at: "))
c = int(input("Step: "))

s = 0
for n in range(a, b + 1, c): #
    s = s + n

print("The sum is", s)
```

for 循环程序实例

■ 实例2：求阶乘 (累积变量)

```
prod = 1
for i in range(2, n + 1):
    prod *= i

print('The factorial of', n, 'is', prod)
```

■ 实例3：通过再一层循环，做三次阶乘计算

- 循环可以**多层嵌套**
- 注意：过多层的嵌套会增加理解程序的难度
- 通过调试运行 / pythontutor.com 观察循环语句的执行情况

while 循环语句

- **while** 用表达式 (其结果为逻辑值或可作为逻辑值使用) 描述循环条件，用于描述各种复杂循环

while 表达式:
语句组

循环头部
循环体

- 执行方式 (语义): 首先求值表达式 (循环条件)
 1. 条件为真时，执行循环体，而后重复执行整个 **while** 语句
 2. 条件为假时，**while** 语句结束
- 用 **for** 语句写的循环都可以改用 **while** 语句 (反之则不然)
 - 需引入循环变量并做适当的增量操作 (例，**while** 循环求阶乘)

```
prod = 1          # 累积变量
i = 2             # 循环变量
while i <= n:
    prod = prod * i
    i = i + 1
```

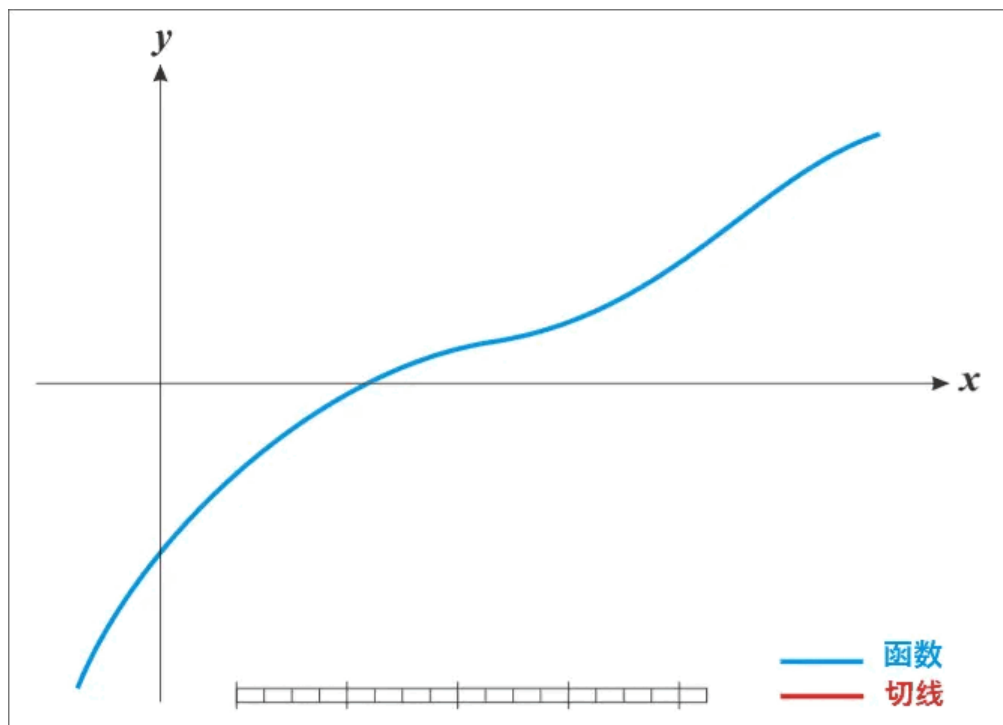
比较两种循环语句

- 控制重复执行 (和结束) 的方式不同
 - **for** 语句基于由可迭代对象确定的值序列
 - **while** 语句基于逻辑条件
- 如果两者都能用, 用 **for** 语句写的程序代码通常更清晰简单
 - 对于比较规范的循环、控制比较简单的循环、事先可确定次数的循环, 提倡使用 **for** 语句
- 如果事先不能确定循环方式或次数, 则需要用 **while** 语句
 - 实际需求: 由用户控制实际循环的次数, 如用户提供特定的输入时循环终止等
 - 实例 1: 交互式阶乘计算器
 - 不断接受用户输入的正整数, 计算并输出相应的阶乘
 - 直至用户输入一个负数 (循环控制条件)

while 循环实现迭代计算

- 实例 2：写程序计算非负实数 s 的平方根 (的近似值)
 - 数学定义： s 的平方根是满足 $x*x = s$ 的非负数 x
 - 数值计算方法：[牛顿迭代法](#)

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



while 循环实现迭代计算

■ 实例 2：写程序计算非负实数 s 的平方根 (的近似值)

□ 迭代计算规则 / 过程：

- 1) 任取非负实数值 x (迭代的初值)
- 2) 如 $x * x$ 等于 s ，则计算结束， x 即是 s 的平方根
- 3) 否则，令 $x = (x + s/x)/2$ ，转回步骤 2)

□ 分析

- 用表达式 $x * x \neq s$ 控制结束，出现什么情况？
 - **while** 可能出现**无穷 (死) 循环**，在 **IDLE** 用 **ctrl-c** 中断执行
- 根据允许误差判断结束，求出的是近似值：如 $x * x$ 足够接近 s ，则接受 x 作为 s 的平方根的近似值
 - 实际计算时间由程序所允许误差决定

循环的控制

- 新需求：在执行循环体的过程中决定终止或控制循环
- 两个循环控制语句：**只能用在循环体里，控制所在的最内层循环**
 - **break** 语句：使当前循环立即终止 (继续执行之后的语句)
 - 形式： **break**
 - **continue** 语句：结束循环体本次执行，回到循环头部 (继续)
 - 形式： **continue**
- 例：用 **break** 语句改造阶乘计算器

```
while True:      # 永真的循环条件
    n = int(input("Factorial for (-1 to stop): "))
    if n < 0: break

    prod = 1
    for i in range(2, n + 1): prod = prod * i

    print("The factorial of", n, "is", prod)
```

循环控制语句的扩充

■ while / for 语句：可以带一个 **else** 段/子句

- 如果有 **else** 段，循环正常完成 (即没有被 **break** 语句中断) 后执行 **else** 段的语句组，而后整个 **while / for** 语句结束
- 如果循环通过 **break** 语句中止，**else** 段的语句组不被执行

■ 例：

```
# 设 lst 是一个整数表
for x in lst:
    if x % 2 == 0:
        break
```

```
else:    # else 段
```

```
    print(f'There is no even number in {lst}.')
```

注意区别：循环语句的 **else** 段语句组 vs. 循环之后的语句