

# 第三章 流程控制





# 主要语句

语句通常是一行一条语句。

如一行中有多条语句，则用分号（;）分开（不建议），

如语句太长要跨行时，可以用续行符（\）跨行表示一个语句。

■ **输入输出语句（已讲）**

■ 条件判断

■ 赋值语句

■ 分支语句

■ 循环语句



# 赋值语句





# 赋值形式

- 在Python中，**不需要事先声明变量名及其类型**，直接赋值即可创建各种类型的对象变量。适用于Python任意类型的对象。
- 基本形式是“**变量 = 值**”的形式

单变量赋值

```
>>>x = 1  
>>>y = 2  
>>>k = x + y  
>>>print(k)  
3
```

序列赋值

```
>>>x,y = 4,8  
>>>print(x,y)  
4 8  
>>>a,b = "34"  
>>>a  
'3'  
>>>b  
'4'
```

不等长序列赋值

```
>>>i,*j = "123"  
>>>i  
'1'  
>>>j  
['2', '3']
```

复合赋值

```
>>>j = 5  
>>>j *= 3 + 1  
>>>j  
20
```





## 例题：交换a, b值

输入两个整数a和b，并将两个数交换后输出

分析：第一步，输入两个数

第二步，交换两个数

第三步，输出两个数

关键点：如何交换两个数？



采用序列赋值方式，仅需要一行代码

```
a = int(input())  
b = int(input())  
print(a,b)  
a,b = b,a #a和b交换  
print(a,b)
```



### 课后任务：

探索在Python中，为什么可以通过 $x, y = y, x$ 这样的语句来实现两个变量的值互换？





# 例：输入三角形的三边长度3, 4, 5，求这个三角形的面积

```
import math #引入数学库
```

```
a = int(input())
```

```
b = int(input())
```

```
c = int(input())
```

```
p = (a + b + c) / 2
```

```
area = math.sqrt(p * (p - a) * (p - b) * (p - c))
```

```
print("三角角形的边长:{:3d},{:3d}, {:3d}".format(a,b,c))
```

```
print("三角角形的面积:{:8.3f}".format(area))
```

$$p = \frac{a+b+c}{2} \quad s = \sqrt{p(p-a)(p-b)(p-c)}$$

```
1 import math  
2 a,b,c = map(int,input().split())  
3 s = (a+b+c)/2  
4 area = math.sqrt(s*(s-a)*(s-b)*(s-c))  
5 print(f"三角角形的边长: {a},{b},{c}",end=' ')  
6 print(f"三角角形的面积: {area:.2f}")
```

① 三角角形的边长: 3,4,5 三角角形的面积: 6.00

② 三角角形的边长: 3, 4, 5  
三角角形的面积: 6.000



# 条件判断





# 布尔表达式

- 条件判断是程序的“决策大脑”。
- 采用布尔表达式。
- 作为条件判断时，最终结果是True或False，程序根据结果执行相应的流程。是编写分支语句、循环和逻辑判断的基础，是编程中不可或缺的组成部分。





# 关系运算

关系运算是用于比较两个值的关系（如大小、相等性），该运算会返回一个布尔值（True 或 False）。主要包括以下六种：

运算符	描述	示例	结果
<code>==</code>	判断两个操作数是否相等	<code>3 == 2</code>	<code>False</code>
<code>!=</code>	判断两个操作数是否不相等	<code>3 != 2</code>	<code>True</code>
<code>&gt;</code>	判断左操作数是否大于右操作数	<code>3 &gt; 2</code>	<code>True</code>
<code>&lt;</code>	判断左操作数是否小于右操作数	<code>3 &lt; 2</code>	<code>False</code>
<code>&gt;=</code>	判断左操作数是否大于或等于右操作数	<code>3 &gt;= 2</code>	<code>True</code>
<code>&lt;=</code>	判断左操作数是否小于或等于右操作数	<code>3 &lt;= 2</code>	<code>False</code>

数据类型要求：类型一致

比较规则：数字型、**字符串**、列表、元组

链式比较：`a < b < c`，其运算等价于 `a < b and b < c`





# 逻辑运算符

逻辑量1	逻辑量2	结果
False	False	False
False	True	False
True	False	False
True	True	True
and 运算		

逻辑量1	结果
False	True
True	False
not 运算	

逻辑量1	逻辑量2	结果
False	False	False
False	True	True
True	False	True
True	True	True
or 运算		





# 合理使用逻辑运算符

## ■ 掌握运算符的优先级：

- not (非) > and (与) > or (或)
- 可以使用括号明确

## ■ 利用短路原则：

- and运算符：当第一个操作数为假时，第一个操作数的值就是表达式的结果，不计算第二个操作数。
- or运算符：当第一个操作数为真时，第一个操作数的值就是表达式的结果，不计算第二个操作数。

## ■ 利用德摩根定律：

- $\neg(A \wedge B)$  等价于  $\neg A \vee \neg B$
- $\neg(A \vee B)$  等价于  $\neg A \wedge \neg B$





# 逻辑运算符实例

闰年的条件：年份能被4整除，且年份不能被100整除，或者年份能被400整除。

```
(year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
```

```
>>> 3>5 and a>3      #注意，此时并没有定义变量a
```

```
False
```

```
>>> 3>5 or a>3      #3>5的值为False，所以需要计算后面的表达式
```

```
NameError: name 'a' is not defined
```

```
>>> 3<5 or a>3      #3<5的值为True，不需要计算后面的表达式
```

```
True
```

```
name = input("Enter your name: ") or "Guest"  
print(f"Welcome, {name}!")
```





# 德摩根定律应用示例：

登录验证条件优化

原始需求：当用户同时满足以下条件时允许登录：

1. 用户名不为空
2. 密码长度  $\geq 8$

原始代码：

```
if not (username != "" and len(password) >= 8):  
    print("登录失败")
```

应用德摩根定律转换： $\neg(A \wedge B)$  等价于  $\neg A \vee \neg B$ ：

```
if username == "" or len(password) < 8:  
    print("登录失败")
```

效果：逻辑等价但更易读，直接描述触发失败的条件。





# 成员运算和身份运算

## ■ 成员运算符：

用于检查某个值是否存在于容器中，如字符串、列表、元组、集合或字典等。

运算符	功能	示例	结果
<b>in</b>	判断元素是否存在于容器中（返回 True/False）	2 in [1,2,3]	True
<b>not in</b>	判断元素是否不在容器中（返回 True/False）	'ab' in 'bac'	False

## ■ 身份运算符：

比较两个对象的身份，通常就是指内存地址，从而判断它们是否是同一对象。

运算符	功能
<b>is</b>	用于判断两个对象是否是同一对象（即它们在内存中的地址是否相同）。如果两个对象是同一个对象，则返回 True，否则返回 False。
<b>is not</b>	用于判断两个对象是否不是同一对象。如果两个对象不是同一个对象，则返回 True，否则返回 False。



# 分支语句





# 分支语句

## ■ 分支语句的三种格式：

单分支语句	二分支语句	连缀的if-elif-else
<pre>if 条件：     语句块1</pre>	<pre>if 条件：     语句块1 else:     语句块2</pre>	<pre>if 条件1：     语句块1 elif 条件2:     语句块2 ... elif 条件n:     语句块n else:     语句块 n+1</pre>



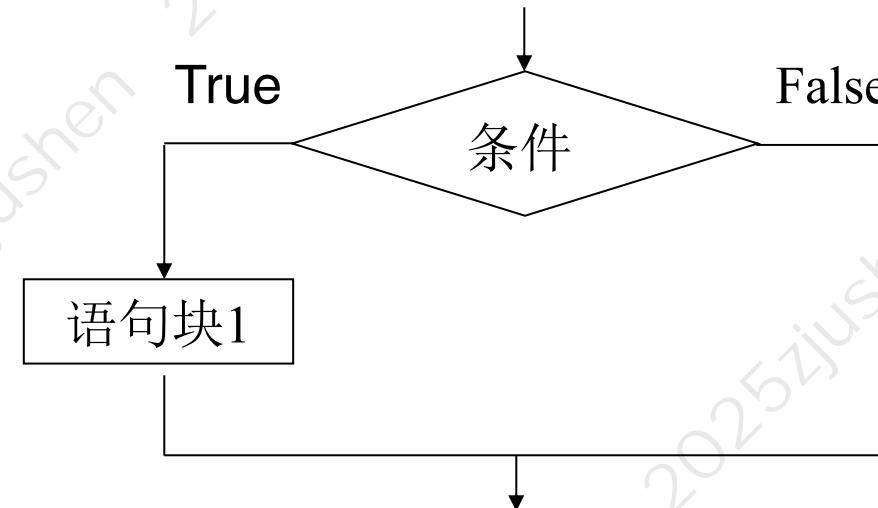


# 单分支语句

- 格式：

`if 条件 :`

`语句块1 #分支语句块,书写时必须缩进。`





# 单分支语句

- 一个基本的分支语句由一个关键字if开头，跟上一个表示条件的逻辑表达式，然后是一个冒号:。
- 从下一行开始，所有缩进了的语句就是当条件成立（逻辑表达式计算的结果为True）的时候要执行的语句。
- 如果条件不成立，就跳过这些语句不执行，而继续下面的其他语句。

```
x = int(input())
y=z=0
if x>20:
    y = 100      # 书写缩进，当x>20时执行
    z = 200      # 书写缩进，当x>20时执行
print(y+z) # if语句后续的语句
```

```
x = int(input())
y=z=0
if x>20:
    y = 100
    z = 200
print(y+z)
```





# 二分支语句

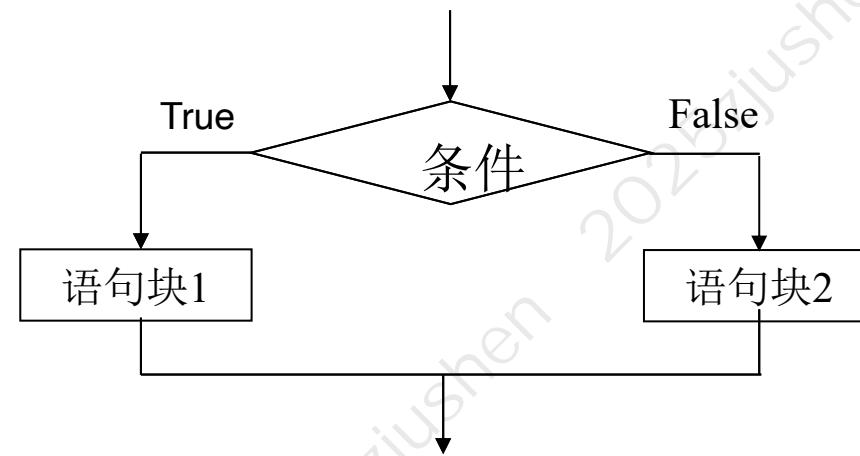
## ● 格式：

**if** 条件:

语句块1 #分支语句块, 条件成立时执行。

**else:**

语句块2 #分支语句块, 条件不成立时执行。





# 例：比较2个数的大小

```
x,y = map(int, input().split())
```

```
if x > y:
```

```
    maxn = x
```

语句块1（条件成立时执行）

```
else:
```

```
    maxn = y
```

语句块2（条件不成立时执行）

```
print(maxn)
```





# 例题：分段函数

本题目要求计算下列分段函数  $f(x)$  的值：

$$y = f(x) = \begin{cases} \frac{1}{x} & x \neq 0 \\ 0 & x = 0 \end{cases}$$

输入格式：

输入在一行中给出实数  $x$ 。

输出格式：

在一行中按 “ $f(x) = result$ ” 的格式输出，其中  $x$  与  $result$  都保留一位小数。

```
File Edit Format Run Options Window Help
x = float(input())
if x == 0:
    y = 0
else:
    y = 1/x
print(f'f({x:.1f}) = {y:.1f}')
```

- 注意冒号
- 一定要注意缩进，并对齐
- 判断是否相等必须是两个等号





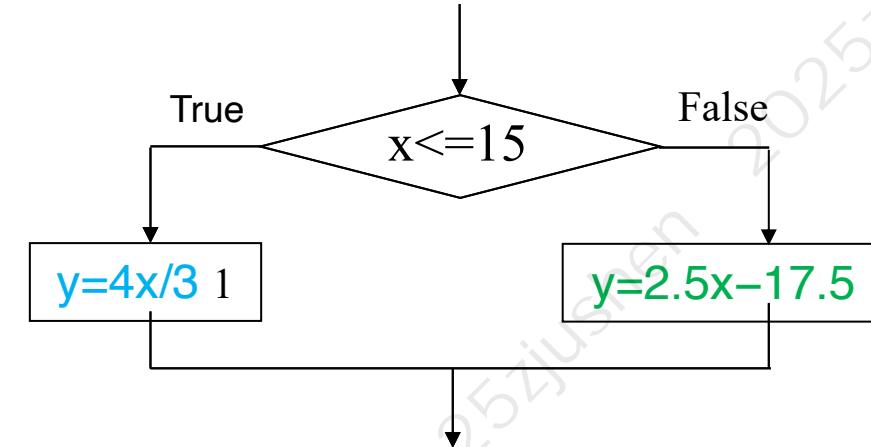
## 例题：计算阶梯水费

为鼓励居民节约用水，自来水公司采取按用水量阶梯式计价的办法，居民应交水费 $y$ （元）与月用水量 $x$ （吨）相关：

当 $x$ 不超过15吨时， $y=4x/3$ ；

超过后， $y=2.5x-17.5$ ；

输出结果小数部分保留2位。请编写程序实现水费的计算。



```
1 x = float(input())
2 if x <= 15:
3     y = 4*x/3
4 else:
5     y = 2.5*x-17.5
6 print(f"{y:.2f}")
```





## 课堂练习：求分段函数

输入x，求分段函数  $f(x)$  的值，保留2位小数。

$$f(x) = \begin{cases} e^x & x \leq 1 \\ x^2 - 1 & x > 1 \end{cases}$$



时间到

要解决的问题：

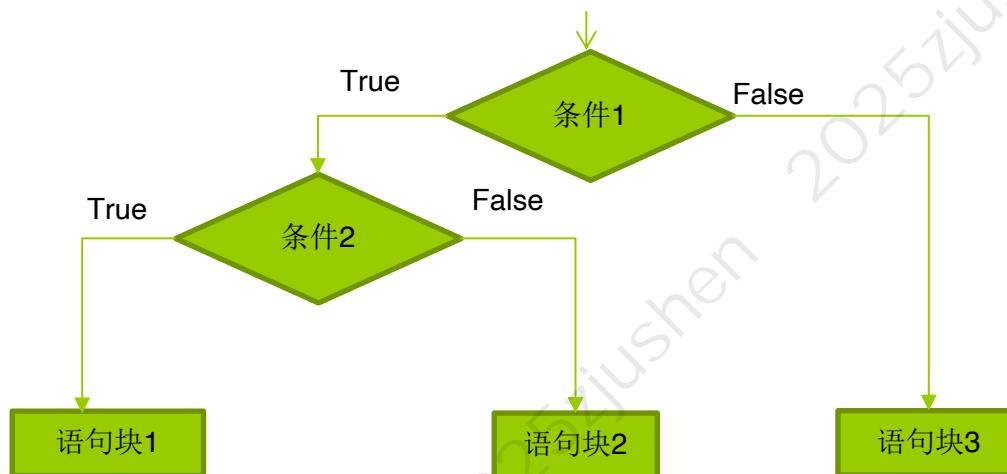
- 输入
- 计算分段函数
- 求 $e^x$
- 输出，并保留2位小数



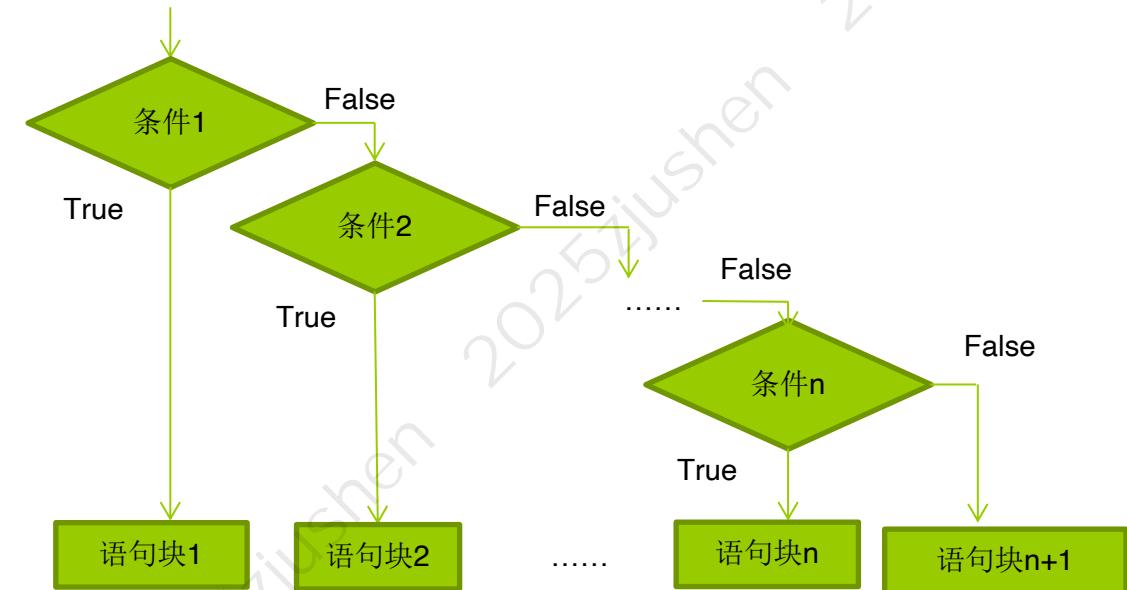


# 多分支语句

## ■ 嵌套的分支语句



## ■ 连缀的if-elif-else



课后任务：

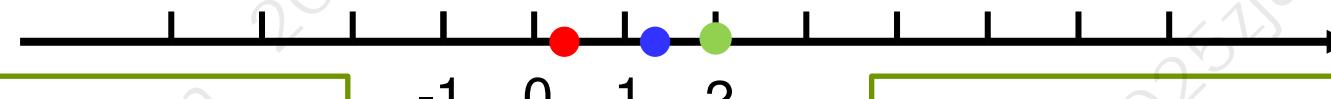
了解 `match case` 使用方式





# 嵌套的分支语句

- 分支语句（块）中包含另一个if语句，这种情况称为分支语句的嵌套
- 在嵌套if语句里，最重要的问题是else的匹配。else总是根据它自己所处的缩进和同列的那个if匹配。



```
if x<2:  
    if x<1:  
        y = x + 1  
    else:  
        y = x + 2
```

书写缩进

```
if x<2:  
    if x<1:  
        y = x + 1  
    else:  
        y = x + 2
```





## 小练习：

游戏根据得分和时间两个值进入不同阶段

■ S级评价：玩家累计得分  $\geq 90$  分

- 奖励：解锁「星核共振技能」
- 隐藏成就：若同时满足 通关耗时  $< 100$  秒，额外获得「时空穿梭者徽章」

■ A级评价：玩家得分在  $60 \leq \text{得分} < 90$  分 区间

- 奖励：获得「基础能源晶体」

■ 未通过认证：玩家得分  $< 60$  分

- 惩罚：被强制传送到关卡起点





# 连缀的if-elif-else

```
if 条件1:  
    语句块1  
elif 条件2:  
    语句块2  
...  
elif 条件n:  
    语句块n  
else:  
    语句块 n+1
```

#其中的if, elif和else必须在同一列对齐。





## 例：分段函数

■ 多分段函数在数学中也是常见的，比如下面的这个函数：

$$f(x) = \begin{cases} -1; & x < 0 \\ 0; & x = 0 \\ 2x; & x > 0 \end{cases}$$



如何改写成嵌套的if语句形式

```
x = int(input())
```

```
if x < 0:
```

```
    f = -1
```

```
elif x == 0:
```

```
    f = 0
```

```
else:
```

```
    f = 2 * x
```

```
print(f)
```

2次条件判断，实现3个分支。





# 条件表达式

- 条件表达式类似if-else语句，是用来直接得到值。
- 条件表达式是三元的，需要三个值：
  - 条件满足时的值
  - 条件
  - 条件不满足时的值

如：`y = 10 if x > 20 else 30`   当x大于20时，条件表达式值为10，否则为30

使用场景：

- 简单二选一赋值
- 保持代码简洁的表达式

不建议在多嵌套的场景使用



# 循环语句





# 循环语句类型

循环语句指用于重复执行某段代码，是控制流程的重要组成部分

- 自动化处理：实现批量数据操作的核心机制。
- 迭代优化：例如在AI模型训练中，循环结构是不可或缺的基础支撑。
- 资源控制：有效防止无限循环，避免系统资源的无谓消耗。

条件控制型：`while`语句

根据一个逻辑条件（循环条件），在条件成立时执行循环体语句，不成立时结束循环。

遍历型：`for`语句

从序列（或其他可迭代对象）中每次取一个元素，然后执行循环体，直到全部元素取完。



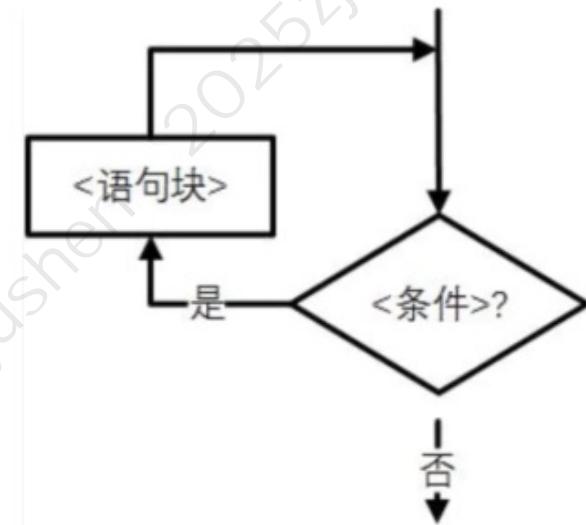


# while循环要点

- 执行while语句（不带else子句）的流程：

1. 判断条件是否成立
2. 如果条件成立，则执行语句块1
3. 回到第1步
4. 如果条件不成立，结束

while 条件：  
    循环体



- 循环语句书写必须缩进
- 在循环体内部，应该有改变循环条件的语句，以控制循环的次数，避免产生无限循环（死循环）。

条件循环





# 输入一组非负整数，求其中的偶数和

```
t=int(input())
s=0
while t>0:
    if t%2==0:
        s=s+t
    t=int(input())
print(f"偶数和为{s}")
```

循环条件

语句块





# 例：计算存几年可以本息合计达1500元

某客户在银行中存入1000元人民币，年利率为3%，假设该客户每年的利息都不取出，计算至少存几年本息合计达1500元。

本息计算公式： $a = p(1 + r)^y$

其中：a表示本息合计，p是第一次存入本金，r表示年利率，y表示年数。

```
y=1  
a=p*(1+r)**y  
while a<1500:  
    y=y+1  
    a=p*(1+r)**y  
print(f'存{y}年本息合计{a:.2f}')
```

存14年本息合计1512.59

不能漏掉，否则会陷入死循环





# break语句

## ■ 前测式循环：

先检查循环条件，再决定是否执行循环体。

使用while语句实现

## ■ 后测式循环：

先执行循环体，再检查条件。

可以通过while True配合break语句模拟

## ■ break语句的作用：跳出所在的循环





# break语句:猜数游戏

```
c.py - C:/Users/rshen/Desktop/c.py (3.8.1)
File Edit Format Run Options Window Help

import random
number = random.randint(0, 100)
count = 0
while True:      # 循环条件是逻辑常量True, 意味着无限循环
    a = int(input('输入你猜的数:'))
    count += 1
    if a == number:
        break      # 跳出当前循环, 执行while后面的语句。
    elif a>number:
        print('你猜的大了')
    else:
        print('你猜的小了')
print('猜中了! 你用了{}次!'.format(count))

Ln: 14 Col: 0
```





## continue语句

**continue语句作用：**跳过当前循环的剩余部分本次循环，进入到下一次循环。

在循环结构中，当我们希望在某些特定条件下跳过某些操作时，就可以使用**continue**语句，避免不必要的计算或操作。





# Continue语句:计算偶数的平均数

要求: 输入一系列的整数, 最后输入-1表示输入结束, 然后程序计算出这些数字中的偶数的平均数, 输出输入的数字中的偶数的个数和偶数的平均数。

```
*求和问题.py - E:/2020网上教学/python/课件/第六周/代码/求和问题.py (3.8.1)*
File Edit Format Run Options Window Help

sum = 0
count = 0
while True:
    number = int(input())
    if number == -1:
        break
    if number % 2 == 1:
        continue      # 如果是奇数的话, 跳过后面的循环语句
    sum += number
    count += 1
average = sum / count
print(average)

Ln: 13 Col: 0
```





# else子句

## ■ 执行流程:

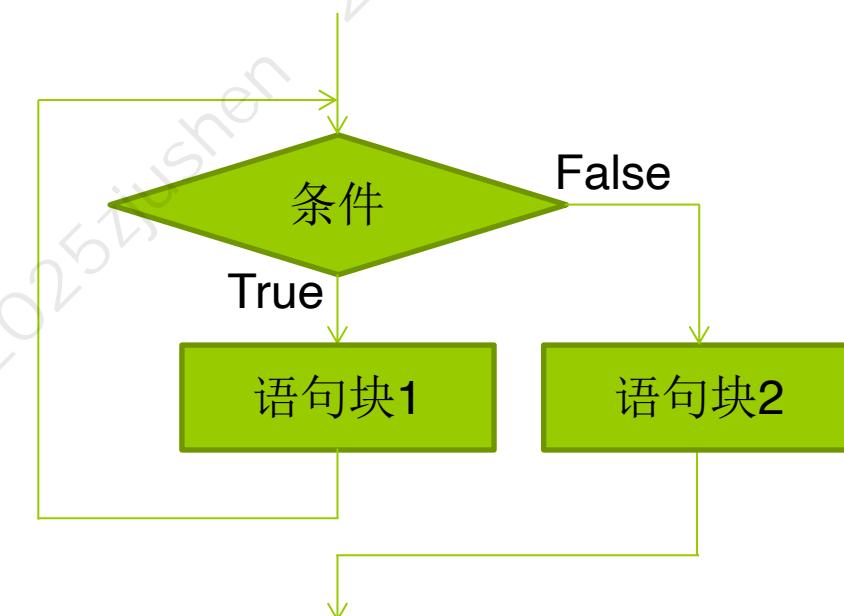
当循环正常执行结束后，else子句被触发，

如果在循环过程中执行了break语句，else子句则不会被执行。

## ■ 功能：

帮助减少代码中使用额外的标志变量来判断循环的退出条件

```
while 条件:  
    语句块1  
else:  
    语句块2
```





# else子句：素数判断

输入一个大于等于2的正整数，判断是否为素数。

- 例: num=6

i	2	3	4	5
num%i	0	0	2	1

num=5

i	2	3	4
num%i	1	2	1

```
num=int(input())
a=num//2
i=2
while i<=a:
    if num % i == 0:
        print("不是素数")
        break      # 跳出当前循环，包括else子句。
    i=i+1
else:
    print("是素数")
```

```
num = int(input())
a = num//2
i = 2
flag = 1
while i<=a:
    if num % i == 0:
        print("不是素数")
        flag = 0
        break
    i = i + 1
if flag:
    print("是素数")
```

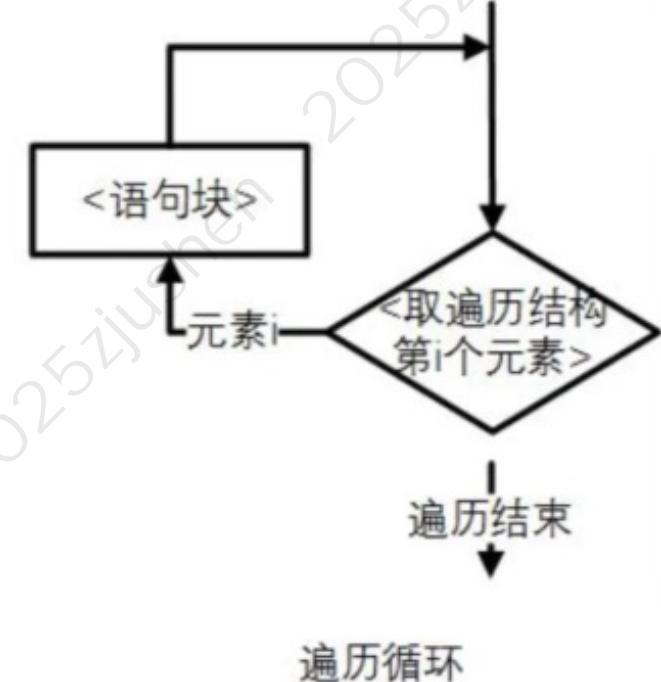


# for语句

**for 变量 in 可迭代对象:**  
    **循环体**

**变量：**先被赋给**可迭代对象**（可以是字符串、range对象、列表等等）的第一个值，并执行语句块。然后变量被赋给可迭代对象中的第二个值，再次执行语句块。该过程一直继续，直到穷尽这个可迭代对象中的所有元素。

**循环体部分：**语句块缩进表示它是属于for代码块





# range

## range(start, stop, step)

start: 计数从start开始。默认是从0开始。例如range (5) 等价于range (0, 5)

stop: 计数到stop结束, 但不包括 stop。例如: range (0, 5) 是0, 1, 2, 3, 4。不包括5

step: 步长, 默认为1。例如: range (0, 5) 等价于range(0, 5, 1)

注意: 三个值都是整型, 取数是左闭右开



range 是 (rangeobject) Python 内置的类型, 它可以认为是一个不可变的数字序列, 通常用在 for 循环。





## range对象实例

```
>>> list(range(10))      # 从 0 开始到 10, 不包括10  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

```
>>> list(range(1,11))    # 从 1 开始到11  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> list(range(0, 30, 4)) # 步长为 4  
[0, 4, 8, 12, 16, 20, 24, 28]
```

```
>>> list(range(0, -10, -1)) # 步长为负数  
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```





```
for i in range(1,6):  
    print(i)
```

1  
2  
3  
4  
5

for variable in range (<计数值>):

语句块

```
for i in range(1,6):  
    print('*')
```

\*  
\*  
\*  
\*  
\*

```
for _ in range(1,6):  
    print('*')
```





# 例：计算存款复利

某客户在银行中存入1000人民币，年利率为3%，假设该客户每年的利息都不取出，计算10年间每年年底账户的本息合计是多少？

本息计算公式： $a = p(1 + r)^y$

其中：a表示本息合计，p是第一次存入本金，r表示年利率，y表示年数。

```
p=1000
r=0.03
for y in range(1,11):
    a=p*(1+r)**y
    print(f'{y:>2}{a:>10.2f}')
```

1	1030.00
2	1060.90
3	1092.73
4	1125.51
5	1159.27
6	1194.05
7	1229.87
8	1266.77
9	1304.77
10	1343.92





# 思考：如何选用for 或 while

建议：

- 如果是可迭代对象，一般选用for循环，更简洁。
- 如果是根据某种状态决定是否进行循环，一般选用while循环，更容易表达。





# 如何选用for 或 while

计算数列 $1+2+\dots+n$ ，当和大于或等于1000停止，输出此时是数列的第几项及总和

while 循环：

```
s=0  
i=0  
while s<1000:  
    i=i+1  
    s=s+i  
print(i,s)
```

for 循环：

```
s=0  
for i in range(1,100):  
    s=s+i  
    if s>=1000:  
        break  
    print(i,s)
```



# 练习题





# 求和问题

编写 Python 代码，计算 0~1000 之间的整数，所有个位数字是 3 的数的和。

思考：

1. 求和问题，循环实现，选 **for** or **while** ?
2. 怎么找出个位数是3的数字？

```
s = 0
for i in range(1,1001):
    if i % 10 == 3:
        s = s + i
print(f'结果为:{s}')
```

```
s = 0
for i in range(1,1001):
    if str(i)[-1] == '3':
        s = s + i
print(f'结果为:{s}')
```

```
s = sum(i for i in range(1,1001) if i % 10 == 3)
print(f'结果为:{s}')
```





# 求解最大公约数

在一行里输入两个正整数，求解它们的最大公约数，并输出。

任务分析：

1. 输入
2. 求解最大公约数
3. 输出





# 最大公约数（算法一）

逐一验证算法：

- 第一步：比较A和B这两个数：A设置为较大的数，B为较小的数；
- 第二步：i从B到1循环执行第三步；
- 第三步：如果i能够同时整除A和B，则最大公约数就是i，程序结束；否则重复进行第二步、第三步。

```
for i in range(b, 0, -1):
    if a%i==0 and b%i==0:
        print(i)
        break
```





# 最大公约数（算法二）

辗转相除算法：

- 第一步：比较A和B这两个数：A设置为较大的数，B为较小的数；
- 第二步：A除以B，得到余数C；
- 第三步：如果C等于0，则最大公约数就是B，程序结束；否则将B赋值给A，C赋值给B；
- 重复进行第二步、第三步。

```
c = a % b  
while c > 0:  
    a, b = b, c  
    c = a % b
```





# 进制转换

输入一个正整数，将其转换成三进制输出。 ➔ 进一步思考，转换成16进制如何实现？

思考：

联想一下十进制到二进制的转换过程，设计十进制到三进制的算法

```
n = int(input())
s = ''
while n>0:
    s = str(n%3)+s
    n = n//3
print(s)
```





# 温控程序

编写程序实现让一个机器人持续检测环境温度。当它连续3次检测到温度>30时，会停止加热，并报警。

思考：

1. 通过不断输入温度值（假设每次输入为一个浮点数）模拟传感器采集数据。
2. 若当前温度>30，记录一次“超标”；否则清零之前的超标计数。
3. 当连续超标次数达到3次时，输出温度超标，报警！并结束程序。
4. 选用for or while？

```
flag = 0
while flag < 3:
    temp = float(input())
    if temp > 30:
        flag += 1
    else:
        flag = 0
print(f"当前连续超标次数: {flag}")
print("温度超标, 报警! ")
```





# 安全检测（循环嵌套）

一个机器人需要检查一个 $5 \times 5$ 的方形区域（坐标范围： $0 \leq x < 5$ ,  $0 \leq y < 5$ , 坐标为整型数据）。若某个坐标点满足  $x + y \leq 4$ , 则认为该点是安全的。编写程序实现以下功能：

1. 用嵌套循环遍历所有坐标点。
2. 对每个坐标点，判断是否安全。
3. 安全的点输出(x,y) 安全，否则输出(x,y) 危险。

4	.	.	.	.	.
3	.	.	.	.	.
2	.	.	.	.	.
1	.	.	.	.	.
0	.	.	.	.	.
0	1	2	3	4	

```
for x in range(5):
    for y in range(5):
        if x + y <= 4:
            print(f'({x},{y})安全')
        else:
            print(f'({x},{y})不安全')
```





# 求[m, n]之间的素数和

```
# 从标准输入读取一行，这行包含两个以空格分隔的整数，然后将它们转换为整数并赋值给m和n
m, n = map(int, input().split())
# 如果m等于1，将m的值改为2，因为1不是素数
if m == 1:
    m = 2
# 初始化一个变量s来存储素数的和，初始值为0
s = 0
# 遍历从m到n（包括n）的每一个整数i
for i in range(m, n + 1):
    # 对于每一个整数i，检查它是否是素数
    # 通过检查i是否能被2到sqrt(i)之间的任何整数整除来实现
    # 如果i能被整除，则它不是素数，使用break跳出内层循环
    for j in range(2, int(i ** 0.5) + 1):
        if i % j == 0:
            break
    # 如果内层循环正常结束（即没有通过break跳出），说明i是素数
    # 此时，将i的值加到s上
    else:
        s = s + i
# 打印出s的值，即m到n之间所有素数的和
print(s)
```



**END**

