Python脚本语言

第六周 Python程序流程控制

陈世峰 岭南师范学院信息工程学院
chenshifeng@lingnan.edu.cn/13234075348

学习目标

1 了解程序流程的基本概念，掌握程序流程控制的3种结构

2 掌握if选择控制语句，并能熟练使用

3 掌握for、while循环控制语句，并能熟练使用

4 掌握else、break、continue流程控制语句的使用方法

5 掌握简单的数学问题求解方法，如质数的判断、阶乘求解等

引言

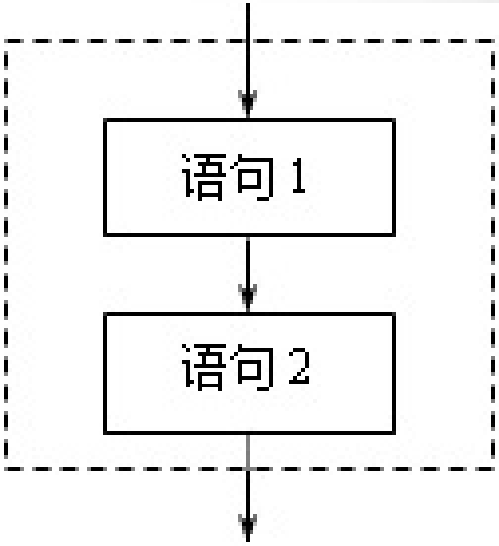
程序语言一般有三类基本程序结构语句，

- **顺序结构语句**
- **分支结构语句**
- **循环结构语句**

再加上一些方便程序编写的其它语句，一个实际工程项目的编程问题就有了语句基础了。只要解题思路清楚、解题步骤正确，就能编写出解题程序。

顺序结构

右图是一个顺序结构的流程图，它有一个入口、一个出口，依次执行语句1和语句2。



一般情况下，实现程序顺序结构的语句主要是赋值语句和内置的输入函数（input()）和输出函数（print()）。这些语句可以完成输入、计算、输出的基本功能。

顺序结构举例

例4.1：圆面积和周长的计算。

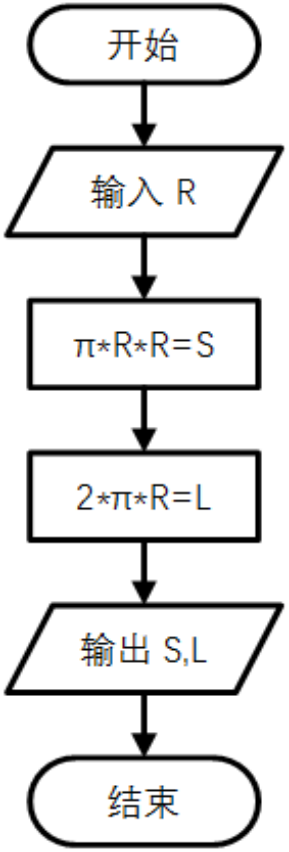
输入： 圆半径R

处理：

圆面积：S = π*R*R

圆周长：L = 2*π*R

输出： 圆面积S、周长L



问题IPO描述

```
R = int(input("请输入圆半径:"))
S = 3.1415*R*R
L = 2*3.1415*R
print("面积和周长:",S,L)
```

Python代码描述

分支结构

分支结构，就是按照给定条件有选择地执行程序中的语句。

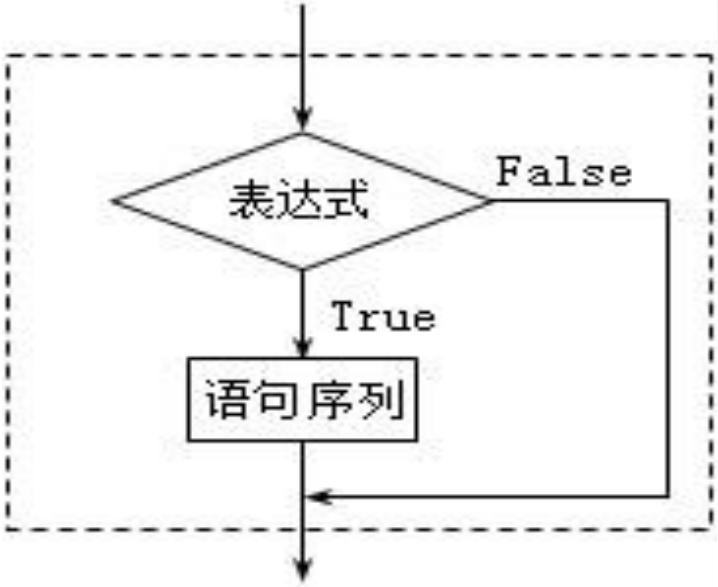
在Python语言中，实现程序分支结构的语句有：

if语句（单分支）

if...else语句（双分支）

if...elif语句（多分支）

if语句（单分支）



语法格式：

```
if <表达式>:
    <语句序列>
```

其中：

（1）表达式是任意的数值、字符、关系或逻辑表达式，或用其它数据类型表示的表达式。它表示条件，以True表示真，False表示假。

（2）<语句序列>称为if语句的内嵌语句以缩进方式表达，编辑器也会提示程序员开始书写内嵌语句的位置，如果不再缩进，表示内嵌语句在上一行就写完了。

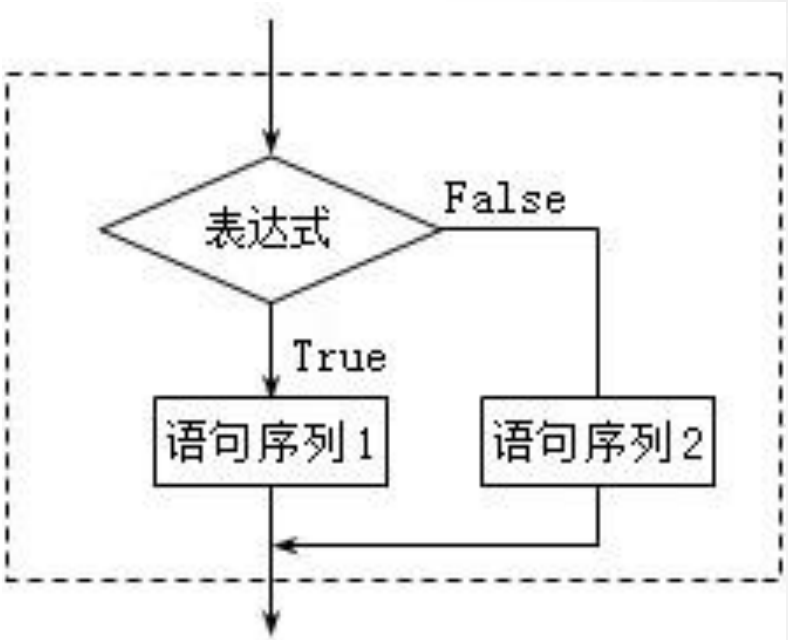
执行顺序见流程图。

例4.2：输入两个整数a和b，按从小到大的顺序输出这两个数（从这个例子开始，写出完整的Python程序代码）。

分析：若a>b，则将a、b交换，否则不交换。

```
#  分支结构举例
a = eval(input("a="))
b = eval(input("b="))
if a>b:
    a,b=b,a
print(a,b)
```

if...else语句（双分支）



语法格式：

```
if <表达式>:
    <语句序列1>
else:
    <语句序列2>
```

执行顺序是：首先计算表达式的值，若<表达式>的值为True，则执行<语句序列1>，否则执行<语句序列2>。if...else语句的流程图如下。

例4.3：输入一个年份year，判断是否为闰年。

分析：闰年的条件为：（1） 能被4整除但不能被100整除；
（2）能被400整除。
用逻辑表达式表示为
(year%4==0 and year%100 !=0) or (year%400==0)

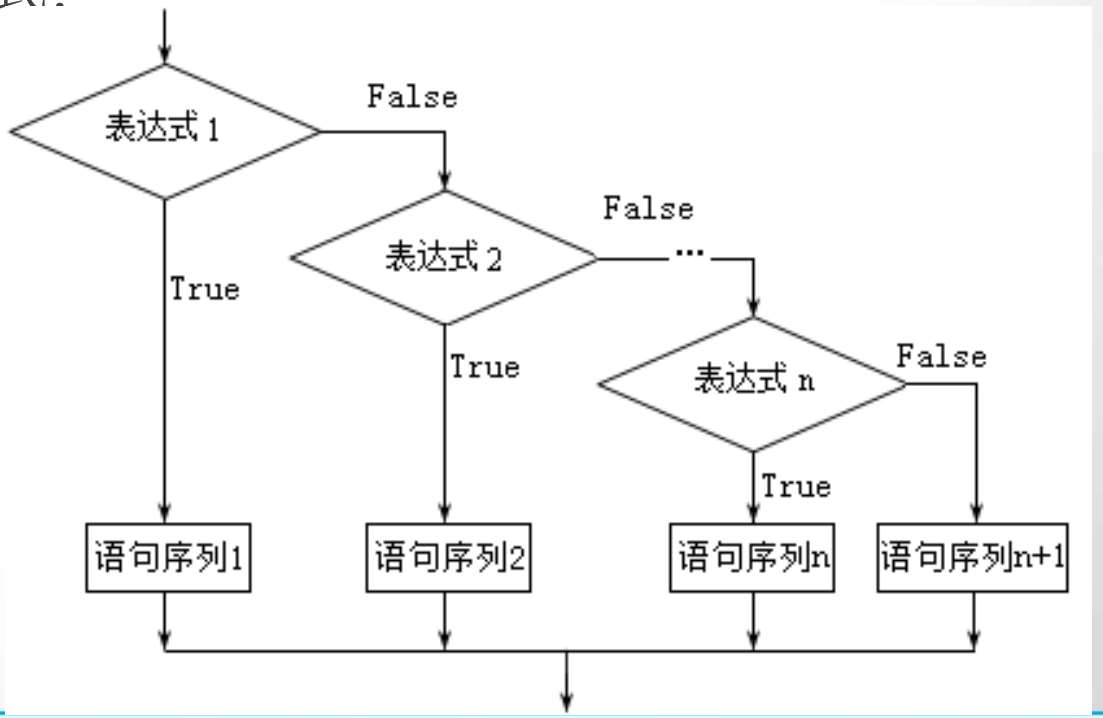
```
year = eval(input())   # 可用int()函数
if (year%4==0 and year%100 !=0) or (year%400==0):
    print(year, ": 闰年")
else:
  print(year, ": 非闰年")
```

if...elif语句（多分支）

双分支结构只能根据条件的True和False决定处理两个分支中的一支。当实际处理的问题有多种条件时，就要用到多分支结构。

if...elif语句的语法格式：

```
if <表达式1>:
    <语句序列1>
elif <表达式2>:
    <语句序列2>
    ...
elif <表达式n>:
    <语句序列n>
else:
    <语句序列n+1>
```



注意：

（1）不管有几个分支，程序执行了一个分支以后，其余分支不再执行。

（2）当多分支中有多个表达式同时满足条件，则只执行第一条与之匹配的语句。

例4.4 成绩从百分制变换成等级制

将百分制分数mark转换为五级制（优、良、中、及格、不及格）的评定等级grade

$$成绩等级=\begin{cases}优 & mark \geq 90\\ 良 & 80 \leq mark < 90\\ 中 & 70 \leq mark < 80\\ 及格 & 60 \leq mark < 70\\ 不及格 & mark < 60\end{cases}$$

方法一：

```
mark = int(input("请输入分数： "))
if (mark >= 90): grade = "优"
elif (mark >= 80): grade = "良"
elif (mark >= 70): grade = "中"
elif (mark >= 60): grade = "及格"
else: grade = "不及格"
```

分支结构的嵌套形式

如果if语句和if... else语句中的内嵌的语句序列又是一个if语句或if... else语句，则称这种形式为if语句（或if... else语句）的嵌套形式。

```
if <表达式1>:
    if <表达式2>:
        <语句序列1>
    else:
        <语句序列2>
[else:
    if <表达式3>:
        <语句序列3>
    else:
        <语句序列4>]
```

实际上，用if语句（或if... else语句）的嵌套形式完全可以代替if...elif语句。但从程序结构上讲，后者更清晰。所以，程序语言中的某些语句只是为了方便程序员写程序，不一定是必要的。

例4.5 使用嵌套的选择结构实现百分制成绩到等级制的转换

将百分制分数score转换为五级制（A、B、C、D、E）的评定等级degree

```
degree = 'DCBAAE'
if score > 100 or score < 0:
    return 'wrong score. must between 0 and 100.'
else:
    index = (score - 60)//10
    if index >= 0:
        return degree[index]
    else:
        return degree[-1]
```

循环结构

循环结构是一种让指定的代码块重复执行的有效机制，Python可以使用循环使得在满足“预设条件”下，可以重复执行一段语句块。

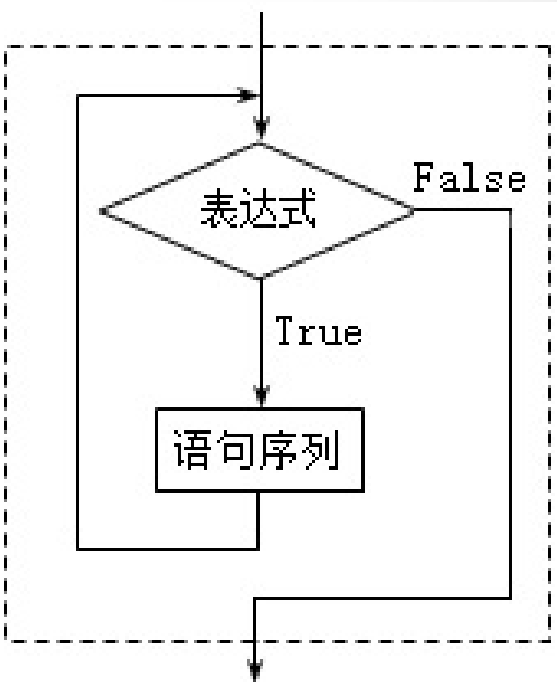
构造循环结构有两个要素，

一是循环体，即重复执行的语句和代码，

另一个是循环条件，即重复执行代码所要满足的条件。

为了能够适应不同场合的需求，Python用while和for关键字来构造两种不同的循环结构，即表达两种不同形式的循环条件。

while语句



while语句用于实现当型循环结构，其特点是**先判断，后执行**

语法格式：

```
While <表达式> :
    <语句序列>
```

其中：

（1）**<表达式>称为循环条件，可以是任何合法的表达式**，其值为True、False，它用于控制循环是否继续进行。

（2）**<语句序列>称为循环体**，它是要被重复执行的代码行。

执行顺序是：首先判断<表达式>的值，若为True，则执行循环体<语句序列>，继而再判断<表达式>，直至<表达式>的值为False时退出循环 。

例4.6 求自然数1～100之和。

即计算sum=1+2+3+...+100。

分析：这是一个累加求和的问题，循环结构的算法是，定义两个int变量，i表示加数，其初值为1；sum表示和，其初值为0。首先将sum和i相加，然后i增1，再与sum相加并存入sum，直到i大于100为止。

```
# 100以内整数的累加和
i = 1
sum = 0
while i<= 100 :
    sum +=i      #等价于sum=sum+i
    i+=1

print("sum= ", sum)
```

程序的运行结果如下：

sum=5050

注意：

（1）在循环体中应该有改变循环条件表达式值的语句，否则将会造成无限循环（死循环）。

（2）该循环结构是先判断后执行循环体，因此，若<表达式>的值一开始就为False，则循环体一次也不执行，直接退出循环。

（3）要留心边界值（循环次数）。在设置循环条件时，要仔细分析边界值，以免多执行一次或少执行一次。

例4.7：求出满足不等式

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}\geqslant 8$$

的最小n值。

分析：此不等式的左边是一个和式，该和式中的数据项个数是未知的，也正是要求出的。

```
# ex-5
i = 0
s = 0
while s<8:
    i+=1
    s+=1/i

print("n= ", i)
```

程序的运行结果如下：

n=1674

for 语句

for <变量> in <可迭代容器>：

　　<语句序列>

其中，<变量>可以扩展为变量表，变量与变量之间用“，”分开。**<可迭代容器>可以是序列、迭代器或其它支持迭代的对象**。

　　执行顺序：<变量>取遍<可迭代容器>中的每一个值。每取一个值，如果这个值在<可迭代容器>中，执行<语句序列>，返回，再取下一个值，再判断，再执行，…，直到遍历完成或发生异常退出循环。

例4.8 使用序列迭代：

```
>>> s = ["XYZ", "Hello", "ABC", "Python"]
>>> for i in s :
...  print(i)
...
XYZ
Hello
ABC
Python
```

例4.9 使用数字对象迭代：

```
>>> x = range(5)
>>> for i in x :
...  print(i, x[i])
...
0 0
1 1
2 2
3 3
4 4
```

range()函数

range()函数的一般格式有两种：

range(end)和range(start,end[,step])，

前一种是默认初始值为0，只要指出终点值。后一种格式可指出两个参数（起点、终点）或三个参数（起点、终点、步长）。

两种格式不可合并。range()函数调用后的结果如下：

```
>>> range(10)
range(0, 10)          # range对象
要想看到表中元素，只能这样：
>>> x=range(10000)
>>> x[9999]
9999
```

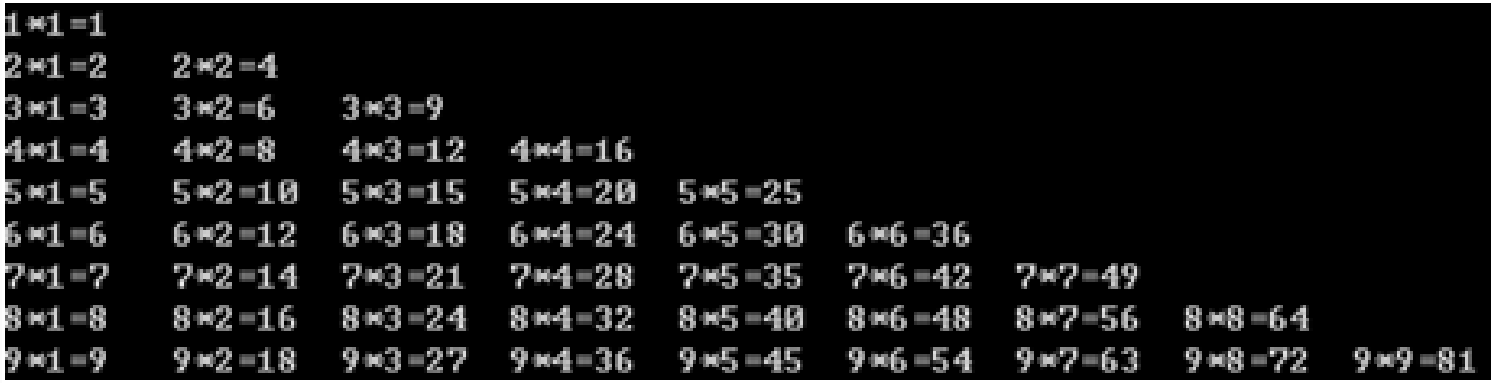
range()函数返回一个range对象，其实是一个列表。

循环结构的嵌套

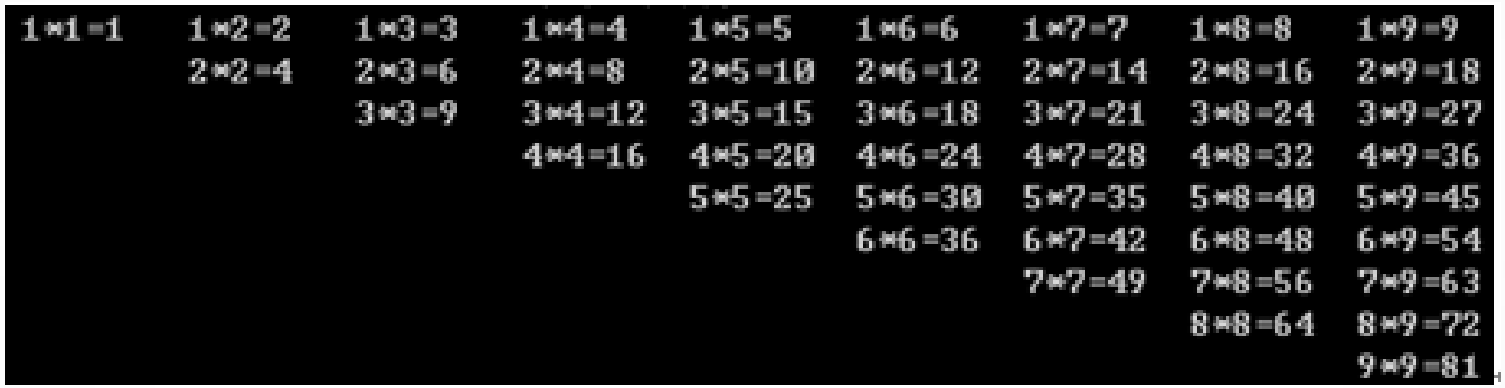
多重循环又称为循环嵌套，是指在某个循环语句的循环体内还可以包含有循环语句。在实际应用中，两种循环语句不仅可以自身嵌套，还可以相互嵌套，嵌套的层数没有限制，呈现出多种复杂形式。在嵌套时，要注意在一个循环体内包含另一个完整的循环结构。

例4. 10 打印九九乘法表

要求输出三角形的九九乘法表




```
for j in range(1,10):
    for i in range(1,j+1):
        print("{}*{}={:>2}". format(j, i, j*i), end=" ")
    print()
```


1×1=1 1×2=2 1×3=3 1×4=4 1×5=5 1×6=6 1×7=7 1×8=8 1×9=9
2×2=4 2×3=6 2×4=8 2×5=10 2×6=12 2×7=14 2×8=16 2×9=18
3×3=9 3×4=12 3×5=15 3×6=18 3×7=21 3×8=24 3×9=27
4×4=16 4×5=20 4×6=24 4×7=28 4×8=32 4×9=36
5×5=25 5×6=30 5×7=35 5×8=40 5×9=45
6×6=36 6×7=42 6×8=48 6×9=54
7×7=49 7×8=56 7×9=63
8×8=64 8×9=72
9×9=81

for循环与while循环区别与联系

- Python提供了两种基本的循环结构语句——while语句、for语句。
- while循环一般用于循环次数难以提前确定的情况，也可以用于循环次数确定的情况。
- for循环一般用于循环次数可以提前确定的情况，尤其是用于枚举序列或迭代对象中的元素。
- 一般优先考虑使用for循环。
- 相同或不同的循环结构之间都可以互相嵌套，实现更为复杂的逻辑。

其他控制关键字

pass 语句可以算作顺序语句，可用在任何地方。break、continue、else语句可以算作特别的顺序语句，只是它们用的地方特别。它们要与分支、循环语句合作使用。

pass语句

pass 语句表示不做任何事情，可以用在任何地方。

```
pass                # 什么事也没做，一个空语句

if True：           # 在if语句中，if语句满足条件时，也不做事
    pass

while 1：           # 无穷循环，循环体没有任何动作
    pass
```

break 语句

break语句用在循环语句（迭代）中，结束当前的循环（迭代）跳转到循环语句的下一条。

break语句常常与if语句联合，满足某条件时退出循环（迭代）。

例4.11 求200以内能被17整除的最大正整数

```
for i in range(200, 0, -1):
    if i % 17 == 0:
        print(i)
        break
```

continue语句

continue语句用在循环语句（迭代）中，忽略循环体内continue语句后面的语句，回到下一次循环（迭代）。

例4.12： 请写出下列程序的运行结果。

```
s = 0
for i in range(1,11) :
    if i%2 == 0 :
        continue
    if i%10 == 7 :
        break
    s = s+i
print("s= ",s)
```

程序的运行结果如下：s= 9

else语句

else语句还可以在while语句或for语句中使用。else语句（块）写在while语句或for语句尾部，当循环语句或迭代语句正常退出（达到循环终点、或迭代完所有元素）时，执行else语句下面的语句序列（else语句下面可以写多个子句）。这意味着break语句也会跳过else语句。

这个附加的else语句加得好，让程序员不必考虑循环退出后，循环是正常退出还是中途退出。

例4.13 判断一个正整数是否为质数

分析：质数大于等于2 不能被它本身和1以外的数整除的数

在一般领域，对正整数n，如果用2到$\sqrt{n}$之间的所有整数去除，均无法整除，则n为质数。

```
x=eval(input())
i=2
isPrime=true
while i<=int(x**0.5):
    if x%i==0:
        isPrime=false
        break;
    i+=1
if isPrime:
    print(x, " : 质数")
else:
    print(x, " : 非质数")
```

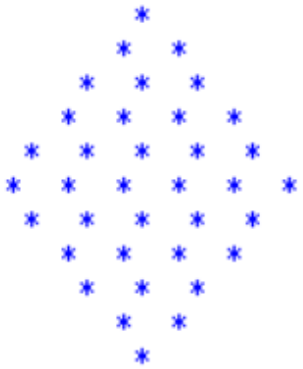
```
x = eval(input("输入一个数： "))
i = 2
while i<=int(x**0.5):
    if x%i ==0 :
        print(x, " : 非质数")
        break
    i = i+1
else :
    print(x, " : 质数")
```

例4.14 计算小于100的最大素数。

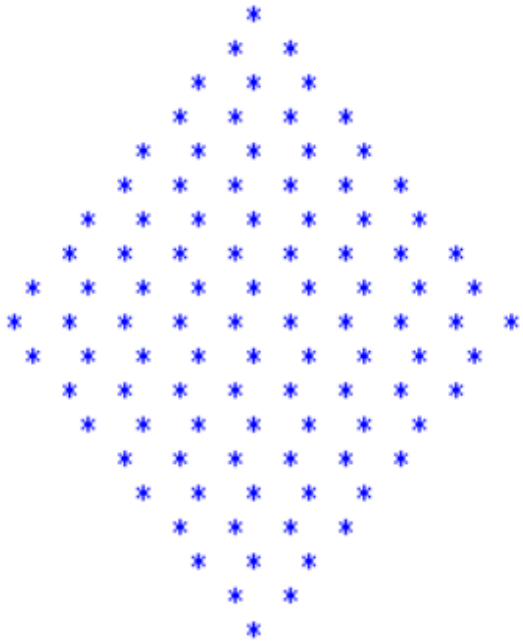
```
for n in range(100, 1, -1):
    for i in range(2, n**0.5):
        if n%i == 0:
            break
    else:
            print(n)
            break
```

- 例4. 18 编写程序，输出星号组成的菱形。

```
n=int(input())
for i in range(n):
    print((' * '*i).center(n*3))
for i in range(n, 0, -1):
    print((' * '*i).center(n*3))
```



n=6 的运行效果



n=10 的运行效果

本章小结

- 顺序结构是最基本的程序结构，在顺序结构中的程序按照从上往下的顺序一条一条的被执行，分支结构则是在顺序结构的程序中加入了判断和选择的成分，在Python中这样的控制成分包括if、else和elif，使用分支结构的好处是我们可以让程序根据某些条件的成立与否，进行不同的选择，从而执行不同的语句块，最终实现不同的功能，更好地满足用户的需求。

本章小结

- 循环结构与分支结构不同，其目的主要是消灭程序代码中连续的重复内容，让程序更加简洁，从而提升程序的可读性，通常使用while和for语句来完成循环结构的控制成分，
- while语句是一种判断型循环控制语句，通常在循环的起始位置会设置一个循环条件，只有当循环条件被打破时，循环才会终止，
- 而for循环则与之不同，for循环是一种遍历型循环，也就是说在循环的起始位置我们需要设置一个遍历范围或者需要遍历的数据集合，在for循环的执行过程，它会将该范围或者集合中的数据带入到循环体中挨个执行一遍，直到所有的数据都尝试过为止。

本章小结

- 在程序的流程控制过程中，还有几个特别重要的关键字需要关注，分别是else、pass、break和continue，这些关键字在程序的流程控制中起着举足轻重的作用，希望读者牢牢掌握，并加以实践。
- 以上内容是构成Python程序的根本。使用这三种结构来构造我们的程序，你便可以解决各种各样的问题。


输入理想的程序
输出快乐的人生