第4章 流程控制语句

AI程序设计课程组

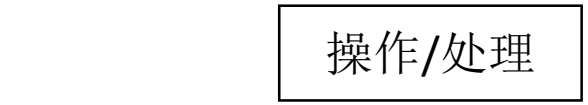
目录页

流程图

程序的流程图

程序流程图用一系列图形、流程线和文字说明描述程序的基本操作和控制流程，它是程序分析和过程描述的最基本方式。

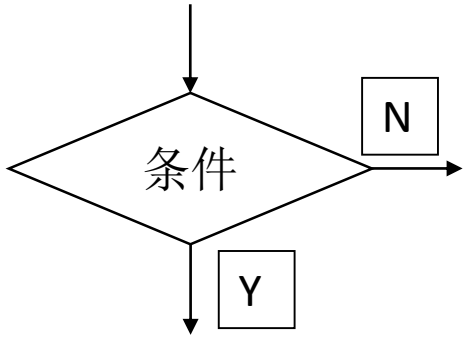
开始符/结束符：表示本段算法的开始或结束。



处理框：框中指出要处理的内容。通常有一个入口和一个出口。

输入符/输出符：表示算法的输入或输出。

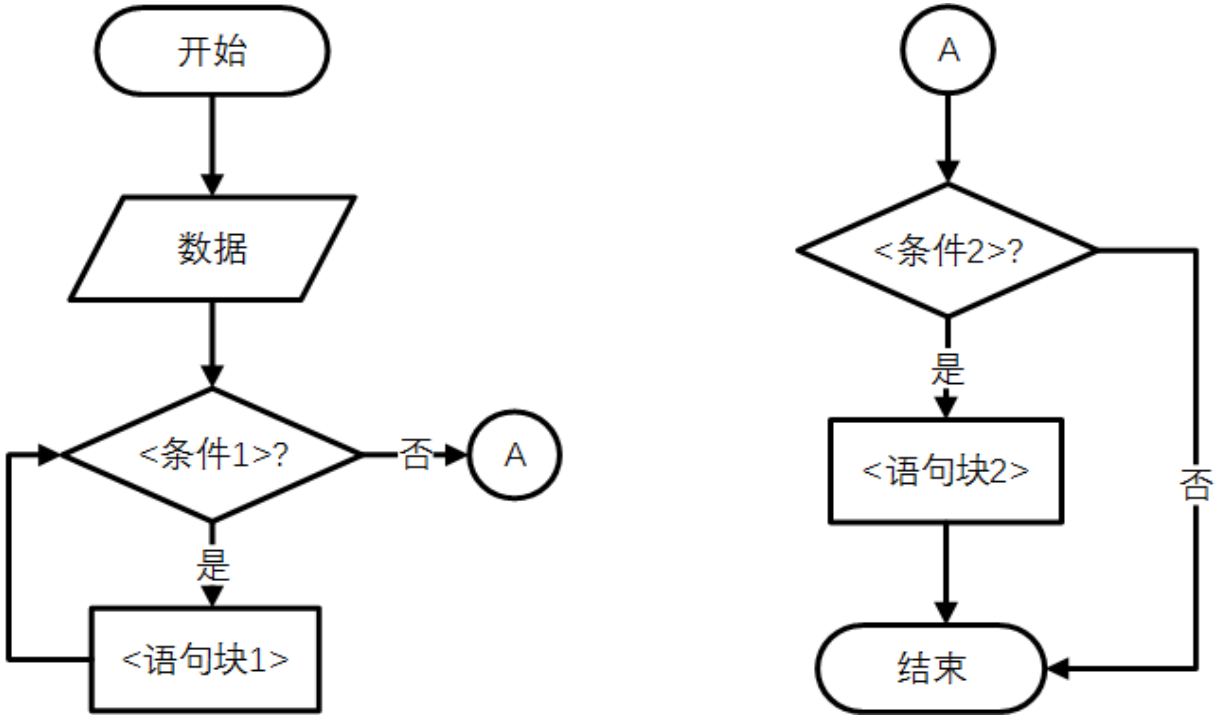
流程线：有向线段，指出流程控制方向。



判断框：表示分支情况。

程序的流程图

- 程序流程图示例：由连接点A连接的一个程序



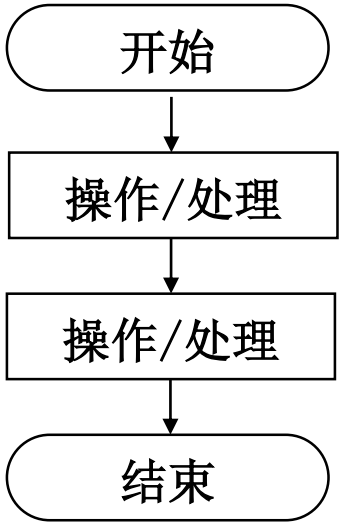
顺序结构

程序的基本结构

- 顺序结构是程序的基础，但单一的顺序结构不可能解决所有问题。
- 程序由三种基本结构组成：
  - 顺序结构
  - 选择结构
  - 循环结构
- 这些基本结构都有一个入口和一个出口。任何程序都由这三种基本结构组合而成

顺序结构

- 顺序结构只要按照解决问题的顺序写出相应的语句就行，它的执行顺序是自上而下，依次执行。
- 顺序结构是程序的基础，但单一的顺序结构不可能解决所有问题。



顺序结构

- 对于一个计算问题，可以用IPO描述、流程图描述或者直接以Python代码方式描述

输入：圆半径R

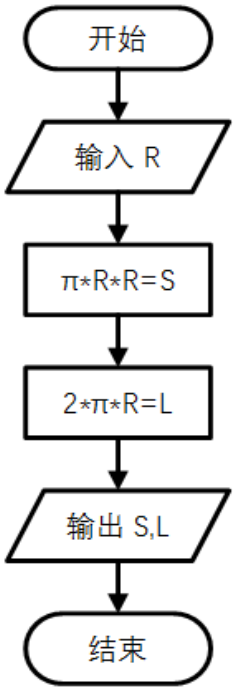
处理：

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（Input-Process-Output）描述



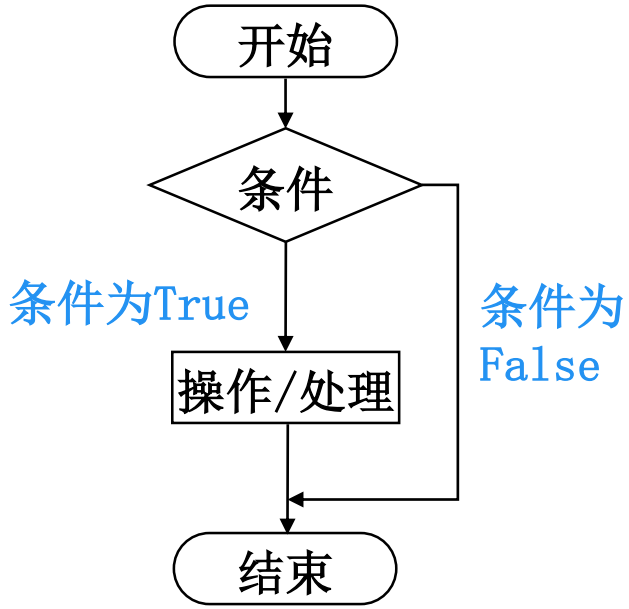

```
1  R = eval(input("请输入圆半径:"))
2  S = 3.1415*R*R
3  L = 2*3.1415*R
4  print("面积和周长:",S,L)
```

Python代码描述

选择结构

选择结构

- 选择结构用于判断给定的条件，根据判断的结果来决定执行的代码块。
- if语句是最简单的条件判断语句，它可以控制程序的执行流程。




```
if <判断条件>:
    <语句块>
```

■ 实例：PM 2.5空气质量提醒

输入：接收外部输入PM2.5值

处理：

if PM2.5值 >= 75，打印空气污染警告

if 35 <= PM2.5值 < 75，打印空气污染警告

if PM2.5值 < 35，打印空气质量优，建议户外运动

输出：打印空气质量提醒

```
1 PM = eval(input("请输入PM2.5数值: "))
2 if 0<= PM < 35:
3     print("空气优质，快去户外运动!")
4 if 35 <= PM <75:
5     print("空气良好，适度户外活动！")
6 if 75 <= PM:
7     print("空气污染，请小心！")
```

布尔值

- 对于计算机而言，布尔值 True 和 False 就表示真和假，除了 True 、 False 是比较显式的真和假, 而在Python中以下值都会被看作是假（False）：

```
False None 0 "" () [] {}
```

- if-else语法格式，缩进决定代码作用域

```
if <判断条件>:
        <语句块1>
else:
        <语句块2>
```

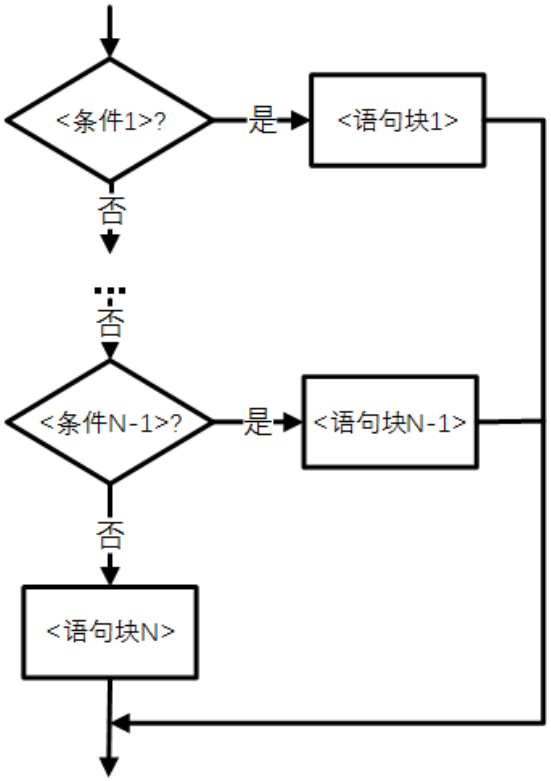
■ 实例：PM 2.5空气质量提醒

```
1  PM = eval(input("请输入PM2.5数值: "))
2  if PM >= 75:
3      print("空气存在污染，请小心！")
4  else:
5      print("空气没有污染，可以开展户外运动!")
```

选择结构

- **Python依次评估寻找第一个结果为True的条件，执行该条件下的语句块，同时结束后跳过整个if-elif-else结构，执行后面的语句。如果没有任何条件成立，else下面的语句块被执行。else子句是可选的。**

```
if <判断条件1>:
        <语句块1>
elif <判断条件2>:
        <语句块2>
...
else:
        <语句块N>
```



案例

■PM 2.5空气质量提醒

```
PM = eval(input("请输入PM2.5数值: "))
if 0<= PM < 35:
    print("空气优质，快去户外运动!")
elif PM <75:
    print("空气良好，适度户外活动！ ")
else:
    print("空气污染，请小心！ ")
```

■考试成绩判定

```
#if条件判断语句案例：判断考试成绩等级
score=int(input('请输入考试成绩：'))
grade=''
if score>=90:
    grade='优秀'
elif score>=75:
    grade='良好'
elif score>=60:
    grade='合格'
else:
    grade='不及格'
print(grade)
```

多条件判断

- 如果判断需要多个条件需同时判断时，可以使用 or （或），表示两个条件有一个成立时判断条件成功；使用 and （与）时，表示只有两个条件同时成立的情况下，判断条件才成功。

```
#if 多条件判断语句案例
num = 8
# 判断值是否在0~5或者10~15之间
if (num >= 0 and num <= 5) or (num >= 10 and num <= 15):
    print('hello')
else:
    print('undefine')
```

```
undefine
```

选择结构—三元表达式

- **if-else结构还有一种更简洁的表达方式，适合通过判断返回特定值**

<表达式1> if <条件> else <表达式2>

```
1 PM = eval(input("请输入PM2.5数值: "))
2 print("空气{}污染!".format("存在" if PM >= 75 else "没有"))
```

```
>>>count = 2
>>>count if count!=0 else "不存在"
2
>>>count = 0
>>>count if count!=0 else "不存在"
"不存在"
```

循环结构

循环结构

- 根据循环执行次数的确定性，循环可以分为确定次数循环和非确定次数循环。
- 确定次数循环指循环体对循环次数有明确的定义。循环次数采用遍历结构中元素个数来体现。
- **Python通过保留字for实现确定次数循环：**

```
for <循环变量> in <遍历结构>:
    <语句块>
```

循环结构——for循环

- **Python通过保留字for实现“遍历循环”：**

```
for <循环变量> in <遍历结构>:
    <语句块>
```

- **遍历结构可以是字符串、文件、组合数据类型或range()函数**

循环N次

```
for i in range(N):
    <语句块>
```

遍历文件fi的每一行

```
for line in fi:
    <语句块>
```

遍历字符串

```
for c in s:
    <语句块>
```

遍历列表

```
for item in lista:
    <语句块>
```

循环结构——for循环

- **range(): 考虑到我们使用的数值范围经常变化，Python提供了一个内置range()函数，它可以生成一个数字序列。**

语法格式

```
for i in range(start,stop,step):
    执行循环语句
```

程序在执行for循环时：

- 循环计时器变量i被设置为start；
- 执行循环语句；
- i递增
- 每设置一个新值都会执行一次循环
- 当i等于end时，循环结束。

循环结构——for循环

- for循环还有一种扩展模式

```
for <循环变量> in <遍历结构>:
    <语句块1>
else:
    <语句块2>
```

- 当for循环正常执行之后，程序会继续执行else语句中内容。

  else语句只在循环正常执行之后才执行并结束，

- 因此，可以在<语句块2>中放置判断循环执行情况的语句。

循环结构——for循环

[69]:
```
#for 循环使用案例
#for 循环遍历列表
classesinfo=['AI Programming','Machine Learning','AI Hardware']
for classname in classesinfo:
    print('课程信息为：{classname:}'.format(classname=classname))
```

```
课程信息为：AI Programming
课程信息为：Machine Learning
课程信息为：AI Hardware
```

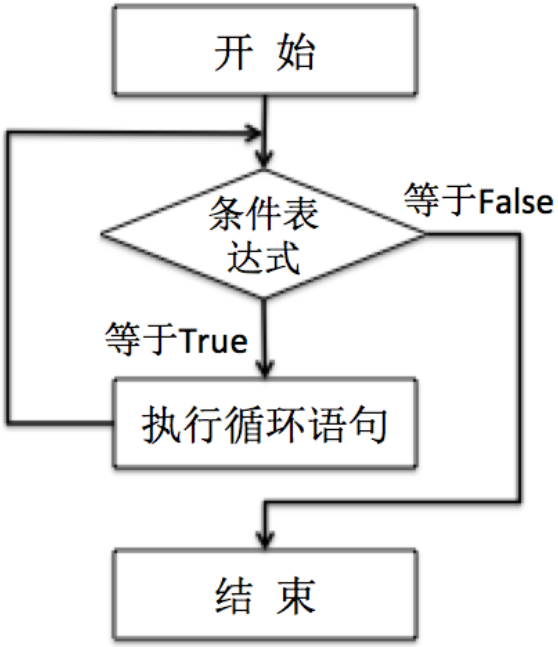
```
#for 循环使用案例
#通过序列索引迭代
classesinfo=['AI Programming','Machine Learning','AI Hardware']
for index in range(len(classesinfo)):
    print('课程信息为：{}'.format(classesinfo[index]))
```

```
课程信息为：AI Programming
课程信息为：Machine Learning
课程信息为：AI Hardware
```

循环结构——for循环

- 非确定次数循环（无限循环）直保持循环操作直到特定循环条件不被满足才结束，不需要提前知道确定循环次数。
- Python通过保留字while实现无限循环，使用方法如下：

```
while <条件>:
        <语句块>
```



While循环语句

- Python 编程中 while 语句用于循环执行程序，即在某条件下，循环执行某段程序，以处理需要重复处理的相同任务。其基本形式为：

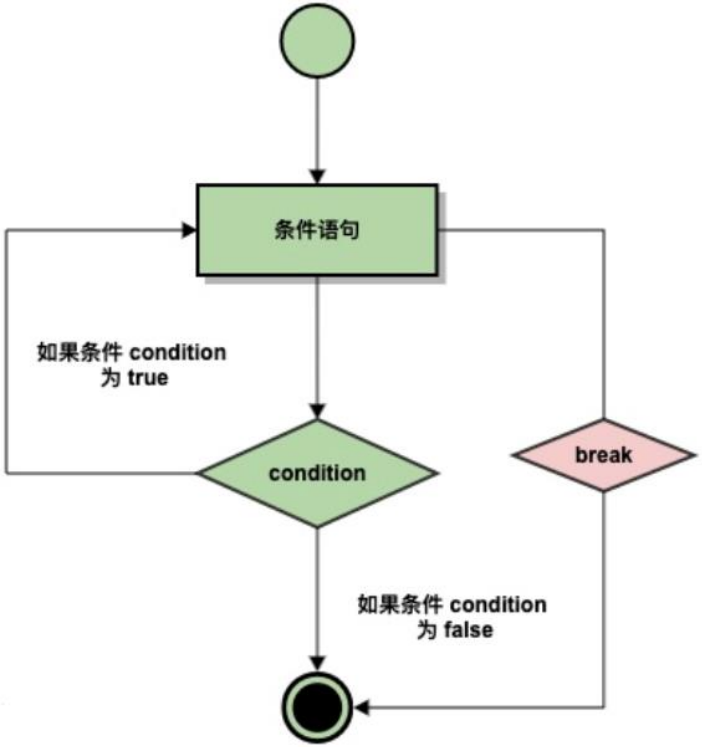
```
while 判断条件(condition)：
    执行语句(statements)……
```

In [56]:
```
#while语句示例
count = 0
while (count < 9):
   print('The count is:', count)
   count = count + 1
print("Good bye!")
```

```
The count is: 0
The count is: 1
```

break语句

- break语句用来终止循环语句，即循环条件没有达到False条件或者序列还没被完全递归完，也会停止执行循环语句。

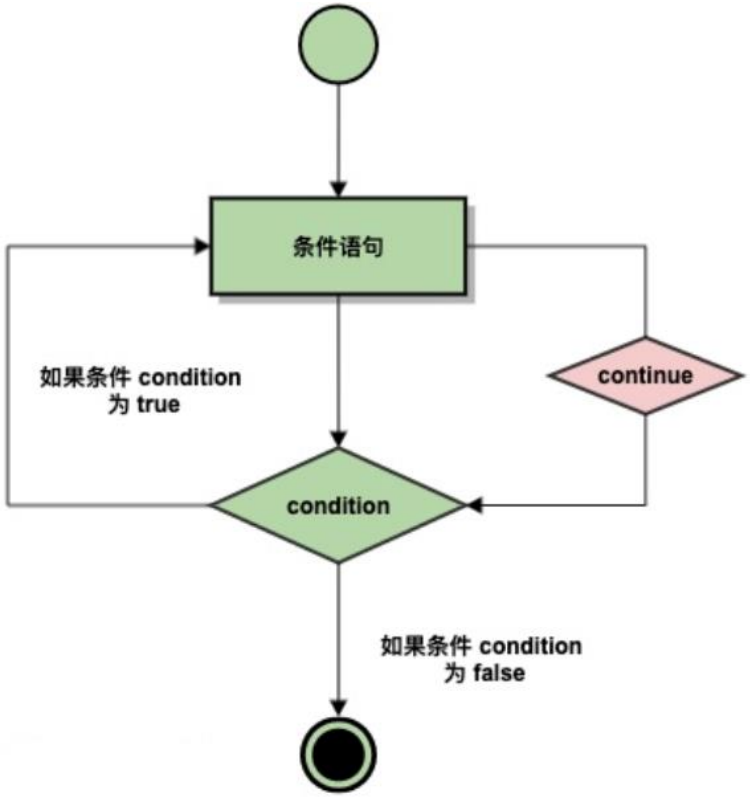



```
#break 的用法
i = 1
while 1:                    # 循环条件为1必定成立
    print(i,end=',')                # 输出1~10
    i += 1
    if i > 10:          # 当i大于10时跳出循环
        break
```

```
1,2,3,4,5,6,7,8,9,10,
```

continue语句

- continue 语句跳出本次循环，而break跳出整个循环。continue 语句用来告诉Python跳过当前循环的剩余语句，然后继续进行下一轮循环。




```
# continue 的用法
i = 1
while i < 10:
    i += 1
    if i%2 != 0:      # 非双数时跳过输出
        continue
    print('i的值为: {}'.format(i))
```

```
i的值为: 2
i的值为: 4
i的值为: 6
i的值为: 8
i的值为: 10
```

While循环语句

- 在 python 中，while … else 在循环条件为 false 时执行 else 语句块：

In [61]:
```
#while…else用法
count = 0
while count < 5:
   print(count, " is  less than 5")
   count = count + 1
else:
   print (count, " is not less than 5")
```

```
0  is  less than 5
1  is  less than 5
2  is  less than 5
3  is  less than 5
4  is  less than 5
5  is not less than 5
```

循环嵌套

- 在一个循环体里嵌入另一个循环

```
for iterating_var in sequence:
    for iterating_var in sequence:
        statements(s)
    statements(s)
```

```
while expression:
    while expression:
        statement(s)
    statement(s)
```

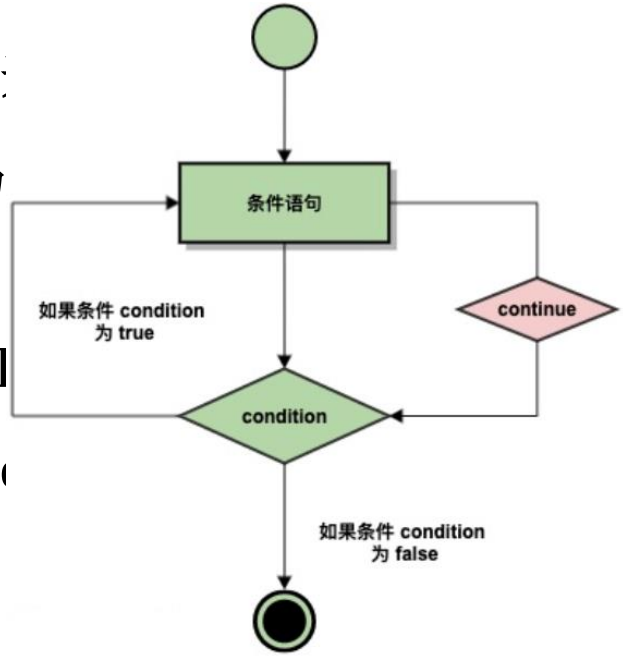
循环保留字：break和continue

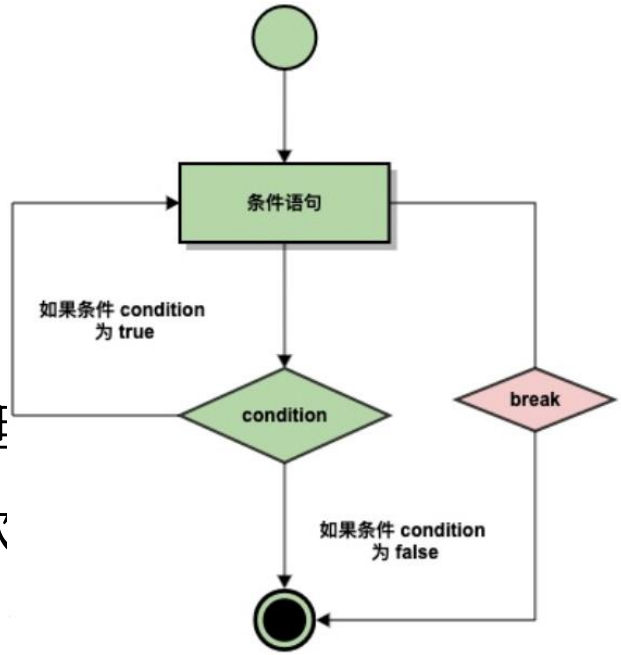
- 循环结构有两个辅助保留字：**break和continue**，它们用来辅助控制循环执行
- **break用来跳出最内层for或while循环，脱离该循环后程序从循环后代码继续续执行**

其中，break语句跳出了最内层for循环，但仍然继续执行外层循环。每个break语句只有能力跳出当前层次循环。

循环保留字：break和continue

- 循环结构有两个辅助保留字：**break和continue**，它们用来辅助控制循环执行
- **break用**[illegible]**环，**[illegible]**环后代码**[illegible]





其中，**break**[illegible]**不，但**[illegible]

环。每个br[illegible]**层次**

循环保留字：break和continue

- **continue**用来结束当前当次循环，即跳出循环体中下面尚未执行的语句，但不跳出当前循环。
- 对于**while**循环，继续求解循环条件。而对于**for**循环，程序流程接着遍历循环列表

```
for s in "PYTHON":
    if s=="T":
        continue
    print(s, end="")
```

```
PYHON
```

```
for s in "PYTHON":
    if s=="T":
        break
    print(s, end="")
```

```
PY
```

循环保留字：break和continue

- for循环和while循环中都存在一个else扩展用法。
- else中的语句块只在一种条件下执行，即for循环正常遍历了所有内容，没有因为break或return而退出。
- continue保留字对else没有影响。看下面两个例子

```
for s in "PYTHON":
    if s=="T":
        continue
    print(s, end="")
else:
    print("正常退出")
```

PYHON正常退出

```
for s in "PYTHON":
    if s=="T":
        break
    print(s, end="")
else:
    print("正常退出")
```

PY

列表推导式

- list是使用python过程中是一个非常常用的数据结构，使用列表推导式可以让循环在列表内完成。

```
#列表推导式
listdemo1=[i+2 for i in range(5)]
listdemo2=[i+2 for i in range(5) if i!=2]
print(listdemo1)
print(listdemo2)
```

```
[2, 3, 4, 5, 6]
[2, 3, 5, 6]
```

字典推导式

- 字典推导式多用于需要元素有一一对应关系时，比如前面谈到当变量是字符型时，需要将字符转换为一一对应的数值型。

```
#字典推导式
classesinfo=['AI Programming','Machine Learning','AI Hardware']
dictdemo1={i:classinfo for i, classinfo in enumerate(classesinfo)}
dictdemo2={value:key for key,value in dictdemo1.items()}
print(dictdemo1)
print(dictdemo2)
```

```
{0: 'AI Programming', 1: 'Machine Learning', 2: 'AI Hardware'}
{'AI Programming': 0, 'Machine Learning': 1, 'AI Hardware': 2}
```

案例：计算π的值

- π（圆周率）是一个无理数，即无限不循环小数。精确求解圆周率π是几何学、物理学和很多工程学科的关键。
- 对π的精确求解曾经是数学历史上一直难以解决的问题之一，因为π无法用任何精确公式表示，在电子计算机出现以前，π只能通过一些近似公式的求解得到，直到1948年，人类才以人工计算方式得到π的808位精确小数。

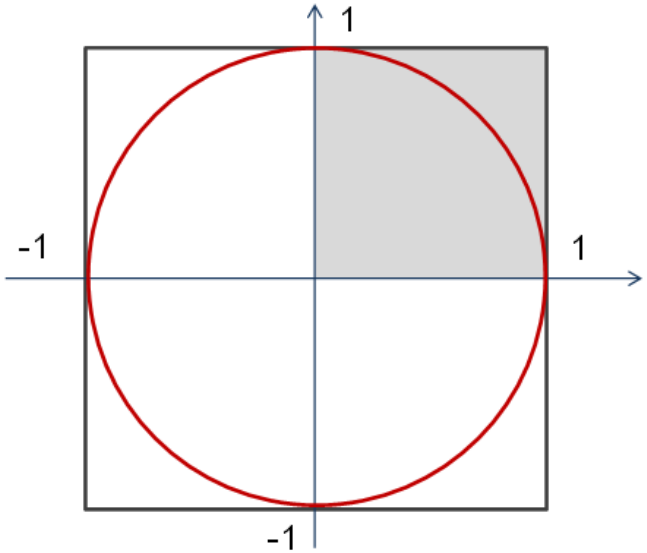
案例：计算π的值

- 随着计算机的出现，数学家找到了另类求解π的另类方法：蒙特卡罗（Monte Carlo）方法，又称随机抽样或统计试验方法。当所要求解的问题是某种事件出现的概率，或者是某个随机变量的期望值时，它们可以通过某种“试验”的方法，得到这种事件出现的频率，或者这个随机变数的平均值，并用它们作为问题的解。这就是蒙特卡罗方法的基本思想。

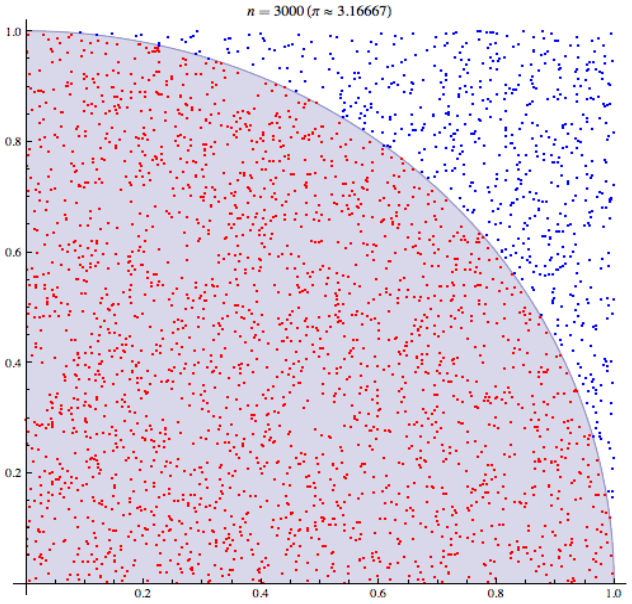
案例：计算π的值

- 应用蒙特卡罗方法求解π的基本步骤如下：
  - 随机向单位正方形和圆结构，抛洒大量“飞镖”点
  - 计算每个点到圆心的距离从而判断该点在圆内或者圆外
  - 用圆内的点数除以总点数就是π/4值。
- 随机点数量越大，越充分覆盖整个图形，计算得到的π值越精确。实际上，这个方法的思想是利用离散点值表示图形的面积，通过面积比例来求解π值。

案例：计算π的值



计算π使用的正方形和圆结构



计算π使用的1/4区域和抛点过程

案例：计算π的值


```
from random import random
from math import sqrt
import time

DARTS = 10000
hits = 0.0
t0 = time.time()
for i in range(1, DARTS+1):
    x, y = random(), random()
    dist = sqrt(x ** 2 + y ** 2)
    if dist <= 1.0:
        hits = hits + 1
pi = 4 * (hits/DARTS)
print("Pi值是{}.".format(pi))
print("运行时间是: {:5.5}s".format(time.time()-t0))
```

```
Pi值是3.1108.
运行时间是: 0.0043936s
```

案例：计算π的值

DARTS	π	运行时间
2^{10}	3.109375	0.011s
2^{11}	3.138671	0.012s
2^{12}	3.150390	0.014s
2^{13}	3.143554	0.018s
2^{14}	3.141357	0.030s
2^{15}	3.147827	0.049s
2^{16}	3.141967	0.116s
2^{18}	3.144577	0.363s
2^{20}	3.1426696777	1.255s
2^{25}	3.1416978836	40.13s

不同抛点数产生的精度和运行时间

总结

- 流程图与程序结构
- 顺序结构，选择结构，循环结构