程序设计（Python）

循环结构

主讲：数据与目标工程学院 胡瑞娟 副教授

知识回顾

基于军事通信信息案例，我们需要**根据不同的通信内容做出不同的处理**。比如：根据状态，

（1）如果是紧急通信，要立即上报；

（2）如果是紧急通信，要立即上报；如果是常规通信，按正常流程处理。

（3）如果是紧急通信，要立即上报；如果是常规通信，10分钟内处理；如果是待处理通信，30分钟内处理；其他情况，标记为待核实，暂不处理。

分支结构
（选择）

知识回顾

【猜数字游戏】

如果猜小了，提示“低了”；猜大了，提示“高了”，猜中，提示“恭喜你猜对了！”。

表示要猜和被猜的数 → 两个数比较（多分支结构） → 得出结论

```
请猜一个1-100数字：50
高了
游戏结束，不玩啦^_^
```

```
请猜一个1-100数字：40
低了
游戏结束，不玩啦^_^
```

思考：IPO的流程，I环节需要准备哪些数据，P环

【找同学黑板上写代码】

```
#猜一次
guess = 49
temp = eval(input("请猜一个1-100数字："))
if guess > temp:
        print("低了")
elif guess < temp:
        print("高了")
else:
        print("恭喜猜对了！")
print("游戏结束，不玩啦^_^")
```

案例引入

【猜数字游戏】

（1）只猜1次：如果猜小了，提示“低了”；猜大了，提示“高了”，猜中，提示“恭喜你猜对了！”。

（2）限定次数n：最多只能猜n次，超过n次，提示“机会用完了！”

（3）不限定次数，直到猜中为止，每次提示“低了”、“低了”、“恭喜你猜对了！”。

（4）复活版：你猜的次数用完了之后，你可以选择使用复活机会（比如最多2次），复活之后玩家的猜测次数重置，游戏继续。

思考重点：对猜1次的动作的重复

循环结构

案例引入

【案例背景】无人机目标识别，假设你是一名无人机操作员，正在执行一项侦察任务。任务目标是确认一个可疑区域内的敌方坦克数量。根据情报，坦克数量在1到100之间。由于战场环境的复杂性（如伪装、遮挡等），无人机需要多次侦察才能获得准确数字。

模拟过程：

（1）紧急侦察模式，侦察1次：每次侦察会传回一个估计数字：系统会根据真实数字给出反馈：“侦察结果偏低”、“侦察结果偏高”或“目标确认！”。

（2）标准侦察模式，限定次数n：由于无人机每次侦察都有被击落的风险，所以任务需要在有限次数内完成。

（3）持续侦察模式，不限定次数：直到确认目标为止，模拟不惜代价获取情报 。

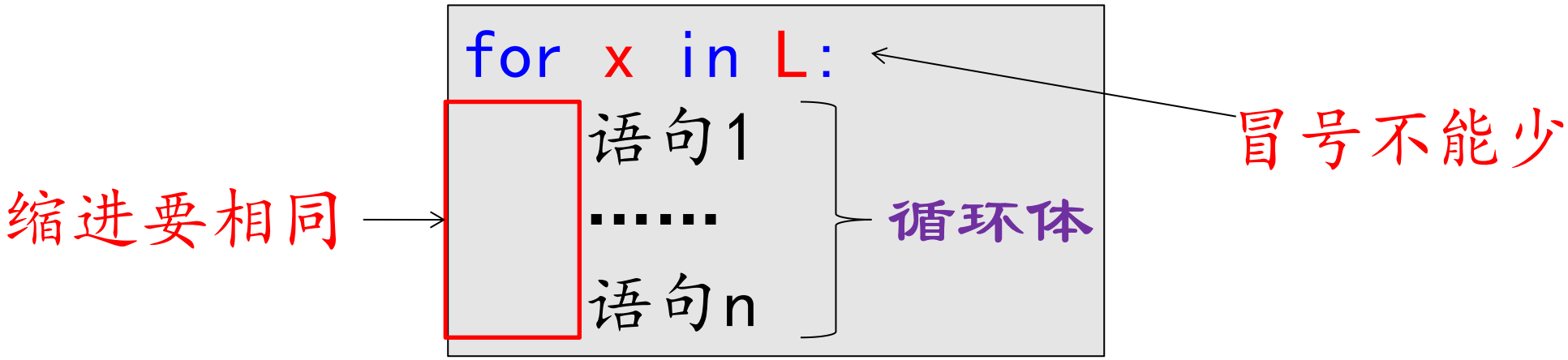
（4）备用方案侦察模式，复活版：无人机有备用机或重新部署的机会，当无人机被击落（次数用尽）时，调用备用无人机（复活）继续侦察，重置侦察次数。

循环结构

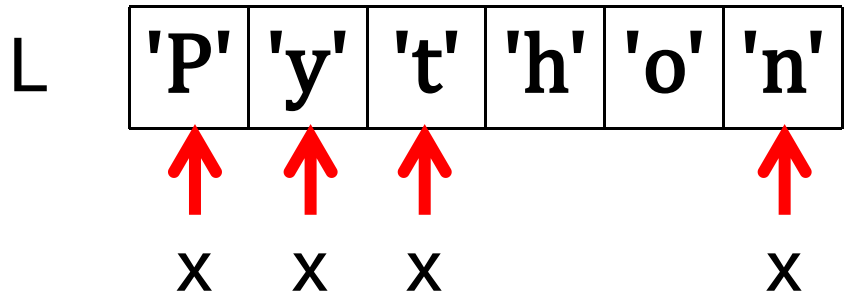
01

for循环

- for循环（遍历循环）
  - x为变量，L为组合数据（遍历结构）
  - 将L中的元素依次取出赋给x，每取一个元素，执行一次循环体



1. for循环



P
y
t
……

```
L = 'Python'
for x in L:
    print(x)
```

```
print(x,end='') #python
```

range函数：产生一系列整数

range (start, end, step)		
Start:起始值（包含）	End:终值（不包含）	Step：步长（不能为0）

range(1,101,1) ⇩ [1,2,3.....,100]

range (start, end)		
Start:起始值（包含）	End:终值（不包含）	省略，默认步长为1

range(1,101) ⇩ [1,2,3.....,100]

range (end)		
省略，默认起始值为0	End:终值（不包含）	省略，默认步长为1

range(101) [0,1,2,3.....,100]

1. for循环

- >>> range(1, 11)
- >>> range(0, 30, 5)
- >>> range(0, 10, 3)
- >>> range(0, -10, -1)
- >>> range(0)
- >>> range(1, 0)

```
for x in range(1,11):
    print(x)
```

0，-1，-2……，-9

空

空

生成以下数字序列

写出range()？

- 2, 4, 6, 8, ..., 100

```
range(2, 101, 2)
```

- 0, 1, 2, 3, ..., 99

```
range(100)
```

- 50, 51, 52, ..., 100

```
range(50, 101)
```

- 100, 99, 98, ..., 1

```
range(100, 0, -1)
```

【思考】求任意5个数的积。

顺序结构

```
f=1
n1=eval(input())
f1=f*n1
n2=eval(input())
f2=f1*n2
n3=eval(input())
f3=f2*n3
……
print(f5)
```

重复操作部分

如何高效表示?

```
n=eval(input())
f=f*n
```

循环5遍

循环结构

```
f=1 #初始值
for i in range( 5 ):
    n=eval(input())
    f=f*n
print(f)
```

【举一反三】计算1-100之间所有7的倍数的和。


```
sum=0
for i in range(1,101):
    if i%7==0:
        sum=sum+i
print(sum)
```

```
sum=0
for i in range(7,101,7):
        sum=sum+i
print(sum)
```

【举一反三】计算1-100之间所有7的倍数的和。

```
sum=0
for i in range(1,101):
    if i%7==0:
        sum=sum+i
print(sum)
```

属于“累加器”类型问题。

基本方法：

（1）在进入累加前先给累加器赋初值（一般为0）；

（2）用循环语句实现累加；

（3）循环体语句的设计。

累加器当前值=累加器原值+循环变量当前值

【举一反三】对比三个程序各输出多少个数?

```
sum=0
for i in range(1,101):
    if i%3==0:
        sum=sum+i
        print(sum)
```

```
sum=0
for i in range(1,101):
    if i%3==0:
        sum=sum+i
    print(sum)
```

```
sum=0
for i in range(1,101):
    if i%3==0:
        sum=sum+i
print(sum)
```

【举一反三】求Fibonacci数列：1, 1, 2, 3, 5, 8, ...的前30项。

分析：

f1=1　　(n=1)

f2=1　　(n=2)

fn=fn-1+fn-2　　(n≥3)　　“递推法”

- f1--第一个数　f2--第二个数　f3--第三个数

  f1=1; f2=1; f3=f1+f2;

- 以后只要改变f1,f2的值，即可求出下一个数.

  f1=f2;f2=f3; f3=f1+f2;

【举一反三】求Fibonacci数列：1, 1, 2, 3, 5, 8, ...的前10项。

```
f1,f2=1,1
print(f"第 1项：{f1:10d}")
print(f"第 2项：{f1:10d}")
for i in range(3,11):
    f3=f1+f2
    f1=f2
    f2=f3
    print(f"第{i:2d}项：{f2:10d}")
```

```
第 1项：         1
第 2项：         1
第 3项：         2
第 4项：         3
第 5项：         5
第 6项：         8
第 7项：        13
第 8项：        21
第 9项：        34
第10项：        55
```

【案例分析】

(2) 限定次数n：最多只能猜n次 (假设n=4)

#random:用于生成随机数的标准库

解决思路

```
import random  as r
guess=r.randint(1,100)      #产生随机数
重复执行4次:
    temp=eval(input("请输入数字(1-100):"))
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    else:
        print("猜对了")
```

random库（模块）

- 常用的函数：

函数	含义
seed(a=None)	初始化随机数种子，默认值为当前系统时间
random()	生成一个[0,1.0]之间的随机浮点数
uniform(a,b)	生成一个a到b之间的随机浮点数
randint(a,b)	生成一个a到b之间的随机整数
choice(<list>)	从列表中随机返回一个元素
shuffle(<list>)	将列表中元素随机打乱
sample(<list>,k)	从指定列表中随机获取k个元素

设置相同的随机数种子，
生成相同的随机数

```
>>> import random
>>> random.seed(5)
>>> random.random()
0.6229016948897019
>>> random.seed(5)
>>> random.random()
0.6229016948897019
```

- 产生的随机数也是在特定条件下产生的确定值， 即**伪随机数**

【案例分析】

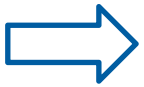
（2）限定次数n：最多只能猜n次，超过n次，提示“机会用完了！”（假设n=4)

编码实现1

分支结构相对for循环是缩进的

```
import random  as r
guess=r.randint(1,100)
for i in range(4):
    temp=eval(input("请输入数字(1-100):"))
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    else:
        print("猜对了")
```

条件循环（n<=4）



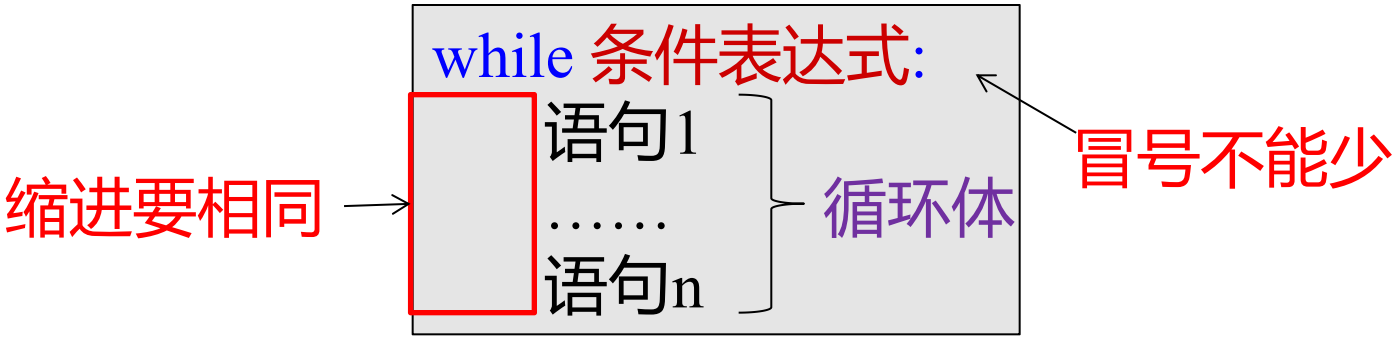
while

02

while循环

(1)条件循环

- **重复执行语句1~语句n，直到条件表达式不再成立**



条件表达式与if中的规定相同

2.while循环

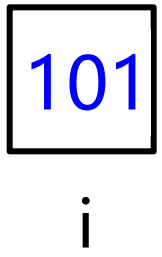
【示例】依次打印学号1~100，最后打印'end'。

循环变量

循环条件

循环变量更新

```
i=1
while i<=100:
    print(i)
    i=i+1
print('end')
```



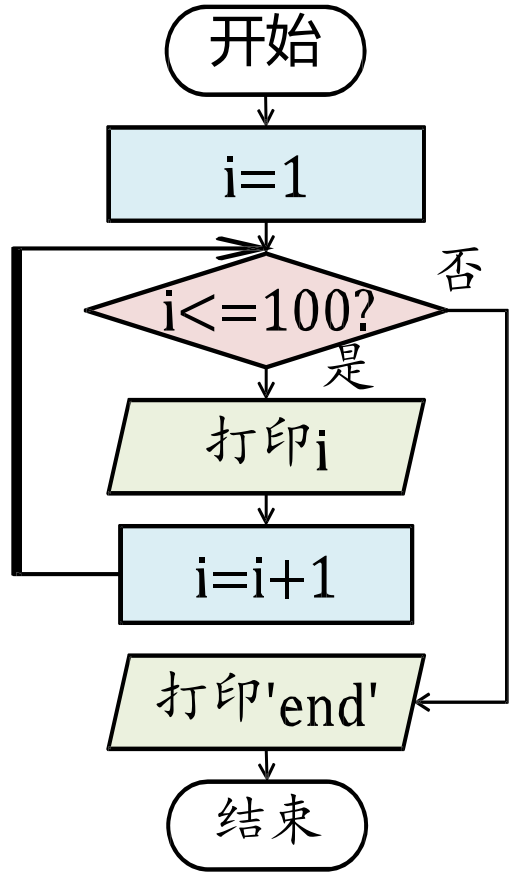
1

2

……

100

end



(2)注意事项

```
1
end
2
end
…
100
end
```

```
i=1
while i<=100:
    print(i)
    i=i+1
print('end')
```

循环体范围

需要重复执行的语句才放入循环体

```
i=1
while i<=100:
    print(i)

print('end')
```

变化量

注意检查某些变量在循环体中的变化是否与预期相符

```
1
1
1
1
1
1
…
```

(2)注意事项

```
2
3
4
。
。
。
101
end
```

```
i=1
while i<=100:
    i=i+1
    print(i)
print('end')
```

第1次执行
注意检查循环体的
第1次执行过程
是否和预期相符

```
i=1
while i<100:
    print(i)
    i=i+1
print('end')
```

最后1次执行
注意检查循环体的
最后1次执行过程
是否和预期相符

```
1
2
3
。
。
。
99
end
```

【举一反三】计算1+3+5+...+997+999

=250000

变量s：存放计算结果

变量i：变化量

```
i=1
s=0
while i<1000:
    s=s+i
    i=i+2
print(s)
```

或

```
i=1
s=0
while i<1000:
    if i%2==1:
        s=s+i
    i=i+1
print(s)
```

【举一反三】计算1*4*7*10*.....*100

变量s：存放计算结果　　变量i：变化量

```
i=1
s=1
while i<101:
    s*=i
    i+=3
print(s)
```

【案例分析】-使用while循环

（2）限定次数n：最多只能猜n次 (假设n=4)

```
for i in range(4):
    temp=eval(input("请输入数字(1-100):"))
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    else:
        print("猜对了")
```

```
i=0
while i<4:
    temp=eval(input("请输入数字"))
    i=i+1
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    else:
        print("猜对了")
        break
```

1 2 3

【思考】如果第3次猜对了，还需要猜第4次吗？

分析：两种循环结构中循环变量i和循环体语句有哪些区别？

重点：**while循环**中要让循环的条件表达式越来越接近False,**在循环体中要改变循环变量的值**，**避免死循环**

03

循环结构控制

3. 循环结构控制

- 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 和 continue

```
>>> for c in "PYTHON" :
        if c == "T" :
            continue
        print(c)
```

结束本次循环

```
P
Y
H
O
N
```

少输出T

```
>>> for c in "PYTHON" :
        if c == "T" :
            break
        print(c)
```

结束当前整个循环

```
P
Y
```

到T就停止循环了

3. 循环结构控制

- break跳出并结束当前整个循环（嵌套循环时只有能力跳出当前层次循环），执行循环后的语句
- continue结束当次循环（当前层当次循环），继续执行后续次数循环
- break和continue可以与for和while循环搭配使用

3. 循环结构控制

【习题】

#只输出字母

```
s="123abcdef"
for c in s:
    if "0"<=c<="9":
        continue
    print(c)
```

运行结果：

```
a
b
c
d
e
f
```

```
for c in "123abcde":
    if 'a'<=s<='z':
        break
    print(c)
```

运行结果为（）

1
2
3

【习题】

```
for i in range(1, 101):
    if i % 15 == 0:
        print(f"{i} 是15的倍数，特殊处理")
        break
    elif i % 5 == 0:
        print(f"{i} 是5的倍数")
        continue
    elif i % 3 == 0:
        print(f"{i} 是3的倍数")
    else:
        print(f"{i} 不是3、5的倍数")
```

3. 循环结构控制

【案例分析】

（2）限定次数n：最多只能猜n次，超过n次，提示“机会用完了！”（假设n=4)

```
import random  as r
guess=r.randint(1,100)
for i in range(4):
    temp=eval(input("请输入数字(1-100):"))
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    else:
        print("猜对了")
        break
```

怎么实现输出“机会用完了”？

```
import random  as r
guess=r.randint(1,100)
for i in range(4):
    temp=eval(input("请输入数字(1-100):"))
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    else:
        print("猜对了")
        break
    if i>=3:
        print(“机会用完了”)
```

循环与else

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

- 当循环没有被break语句退出时，执行else语句块
- else语句块作为"正常"完成循环的奖励

循环与else

问题：程序的运行结果？

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

PYHON正常退出

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

PY

【案例分析】

（2）限定次数n：最多只能猜n次，超过n次，提示“机会用完了！”（假设n=4)

```
import random  as r
guess=r.randint(1,100)
for i in range(4):
    temp=eval(input("请输入数字(1-100):"))
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    else:
        print("猜对了")
        break
else:
    print(“机会用完了！”)
```

互斥

3. 循环结构控制

```
for i in range(3):
    if i == 1:
        break
    print(i)
else:
    print("循环结束")
```

A 0

B 0 1

C 0 1 2

D 0 2

【案例分析】

（3）不限定次数，直到猜中为止。

满足特定条件就重复执行

解决思路

```
temp=eval(input("请输入数字(1-100):"))
if guess>temp:
    print("小了")
elif guess<temp:
    print("大了")
else:
    print("猜对了")
```

【案例分析】

（3）不限定次数，直到猜中为止。

思路和方法1

```
guess=49
一直重复
    temp=eval(input("请输入数字(1-100):"))
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    else:
        print("猜对了")
        猜对之后就退出
```

什么语句能一直为True呢？

【案例分析】

（3）不限定次数，直到猜中为止。

思路和方法1

```
guess=49
while True:
    temp=eval(input("请输入数字(1-100):"))
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    else:
        print("猜对了")
        break
```

while True表示一直循环

结束循环

3. 循环结构控制

- while True 的使用场景非常广泛，包括**用户输入监听**、服务器主循环、**实时监控**、游戏开发、任务调度等。
- 合理使用 while True 可以让程序更加灵活和高效，但需要注意**避免死循环**和资源浪费。 搭配break使用，否则程序陷入死循环。

```
import time
while True:
    temperature = read_temperature_sensor()
    if temperature > 30:
        print("温度过高！")
    time.sleep(1)  # 每隔1秒检查一次
```

【案例分析】

（3）不限定次数，直到猜中为止。

思路和方法2

guess=49
输入要猜的数
只要两个数不相等

先比较大小
再提示出入要猜的数

相等就输出“恭喜，猜对了”

```
guess=49
temp=eval(input("请输入数字:"))
while temp!=guess:
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    temp=eval(input("请输入数字:"))
else:
    print(“恭喜，猜对了”)
```

【拓展】在原功能的基础上，计算并输出猜的次数。

```
guess=49
count=0 #设置次数的变量并初始化为0
for i in range(4):
    temp=eval(input("请输入数字(1-100):"))
    count=count+1
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    else:
        print("猜对了")
        break
else:
    print( “机会用完了！ ” )
print(count)
```

限定次数

```
guess=49
count=0
while  True:
    temp=eval(input("请输入数字(1-100):"))
    count=count+1
    if guess>temp:
        print("小了")
    elif guess<temp:
        print("大了")
    else:
        print("猜对了")
        break
print(count)
```

不限定次数

【案例分析】

(4) 复活版，又给2次机会（一共三轮），如何实现?

```
guess = 49
# 外层循环控制游戏轮次
for round in range(1, 4):  # 进行3轮游戏
    count = 0
    print(f"\n=== 第{round}轮游戏开始 ===")
    # 内层循环控制单轮猜测
    while True:
        temp = eval(input("请输入数字(1-100):"))
        count = count + 1
        if guess > temp:
            print("小了")
        elif guess < temp:
            print("大了")
        else:
            print("猜对了！ ")
            break
    print(f"本轮共猜了{count}次")
print("\n=== 游戏结束，共完成3轮 ===")
```

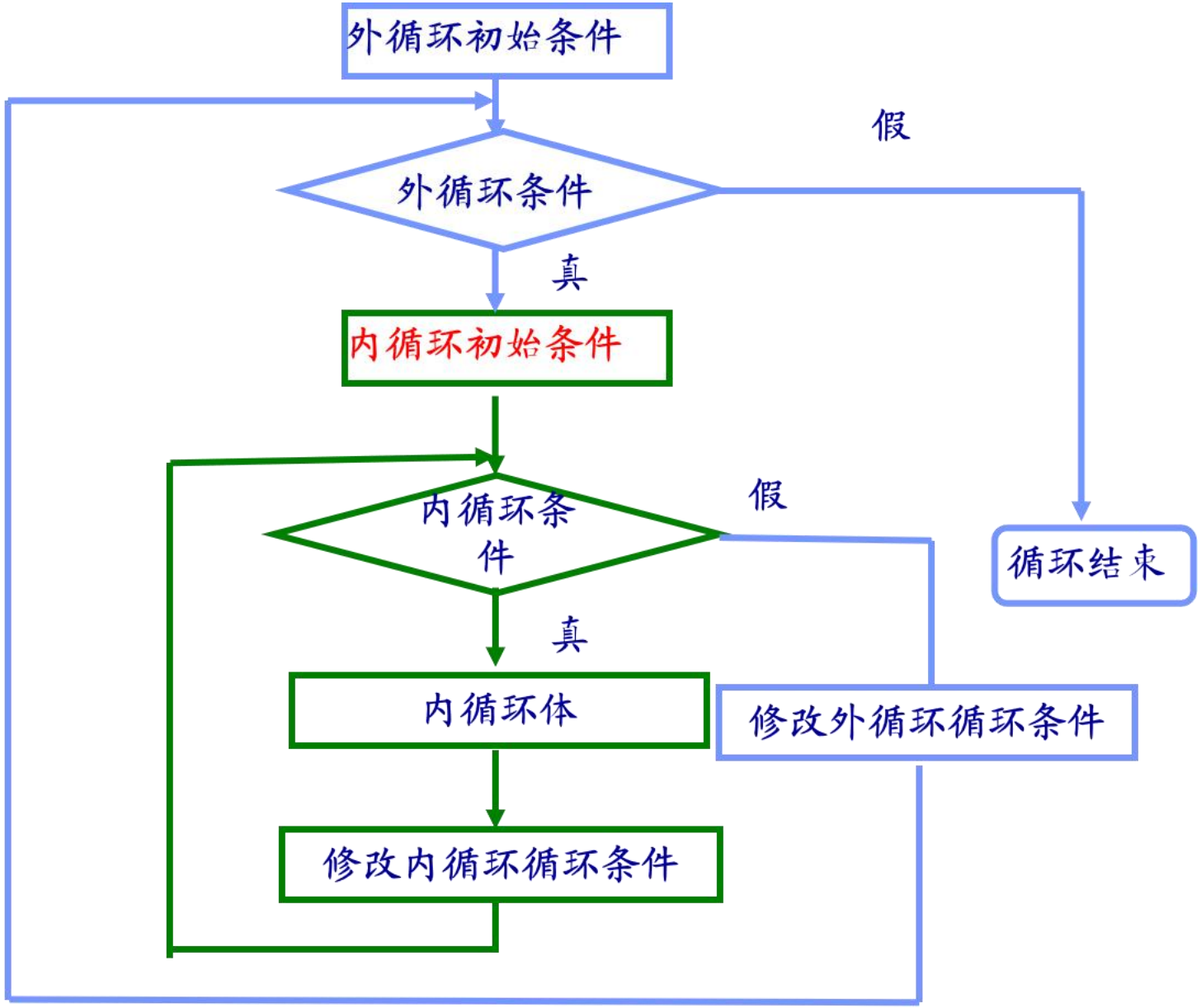
循环的嵌套

循环的嵌套

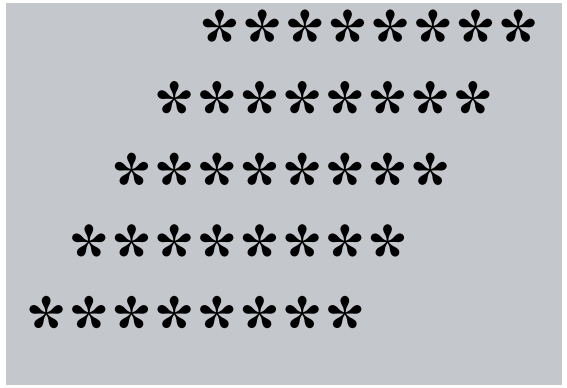
- 在循环体语句中又包含有另一个完整的循环结构的形式，称为循环的嵌套。
- 嵌套在循环体内的循环体称为内循环，外面的循环称为外循环。

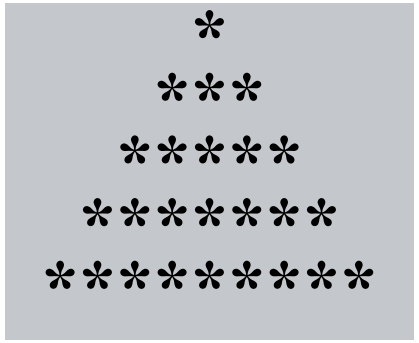
循环的嵌套

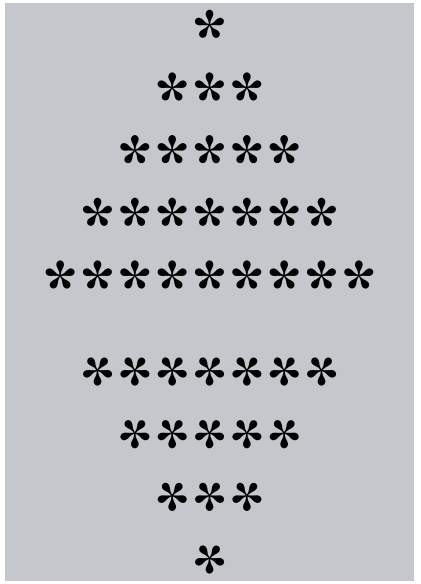
二重循环嵌套
结构执行流程



思考







循环的嵌套

```
        *********
      *********
    *********
  *********
*********
```

```
for row in range(1, 6):
    for col in range(1, 6 - row):
        print(" ", end="")
    for col in range(1, 9):
        print("*", end="")
    print()
```

1. 找出循环体
2. 循环体次数界定
3. 循环主体框架的确定
4. 内外循环控制变量的关系

总结

- 从单次识别到多轮确认，从基础循环到智能优化——追求卓越的技术精神

  我们在编程中要发扬精益求精、反复调试的工匠精神。

- 从简单猜数到复杂军事应用——技术创新

  我们要树立“科技是核心战斗力”的思想，用创新编程助力强军兴军目标。

- 人生如循环，在反复锤炼中突破自我，每一次坚持都是向梦想的靠近。

作业

1.编写程序实现无人机目标识别（课堂案例）。

2.反序数就是将整数的数字倒过来形成的整数。例如，1234的反序数是4321。设N是一个四位数，它的9倍恰好是其反序数，编程计算并输出N的值。

下课并不代表思考的终止
期待我们下次的思想碰撞