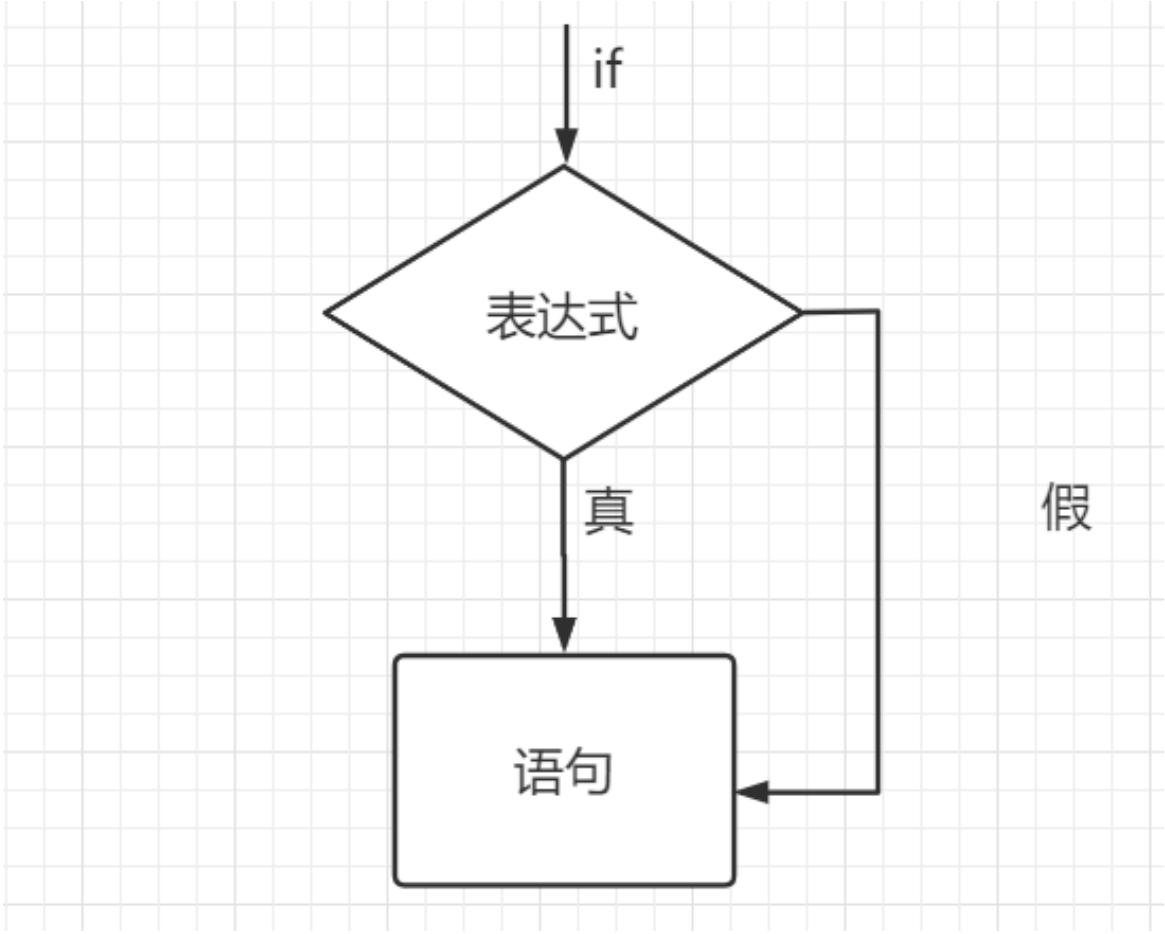
流程控制

条件判断

- 顺序结构的程序虽然能解决计算、输出等问题，但不能做判断再选择。对于要先做判断再选择的问题就要使用分支结构。
- 单分支语句
  - 语法：

```
if 表达式:
    代码块
```

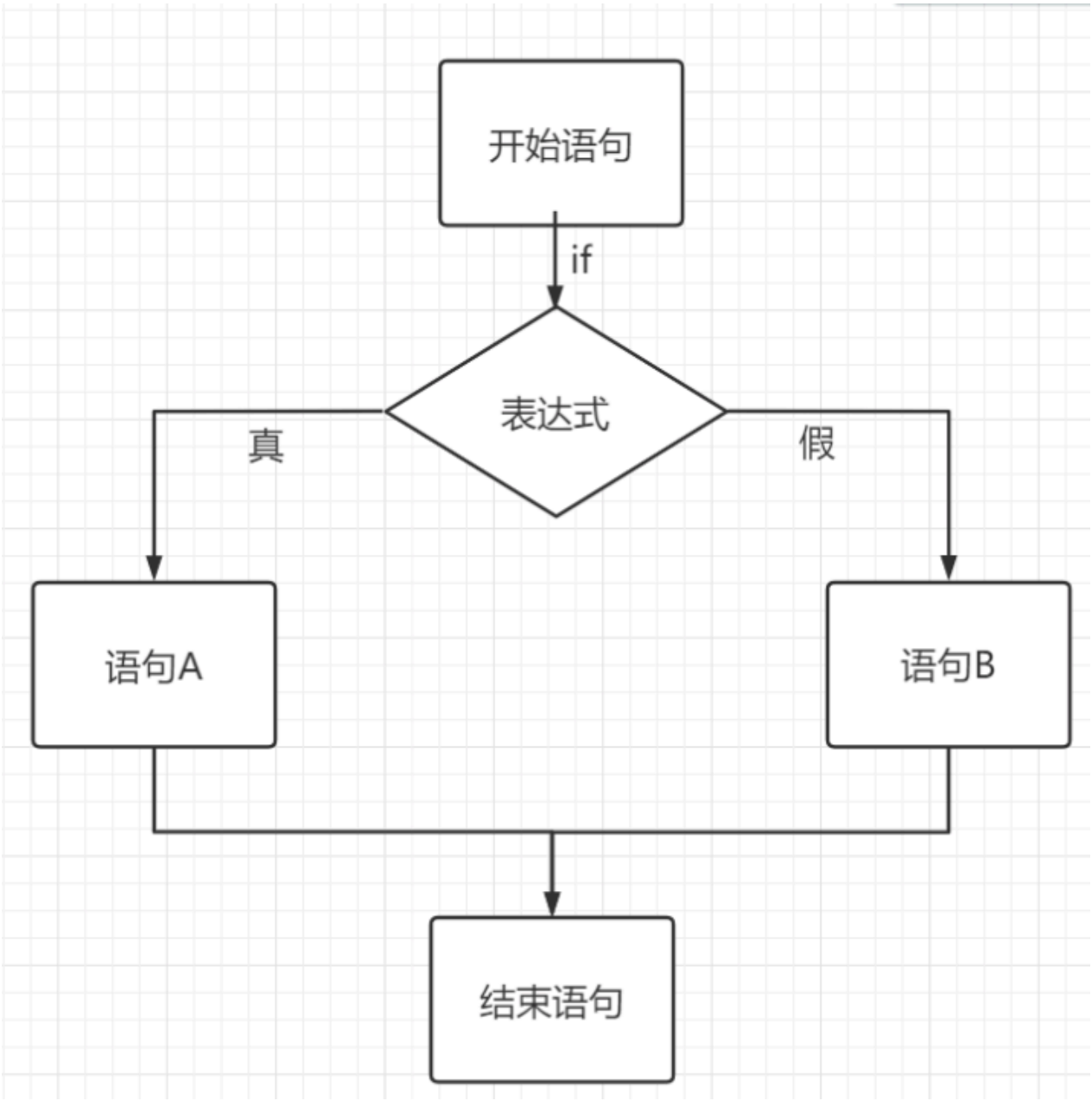
  - > 说明：
    >
    > 1、“表达式”可以是一个单一的值或者复杂语句，形式不限，但解释器最后会通过`bool`获取一个`true`或者`false`的布尔值
    >
    > 2、“代码块”由：与具由相同缩进标识的若干条语句组成（一般是四个缩进）。



- 双分支语句
  - 双分支语句顾名思义，二条分支二选一执行！语法格式：

```
if 表达式:
    代码块 1
else:
    代码块 2
```

  -

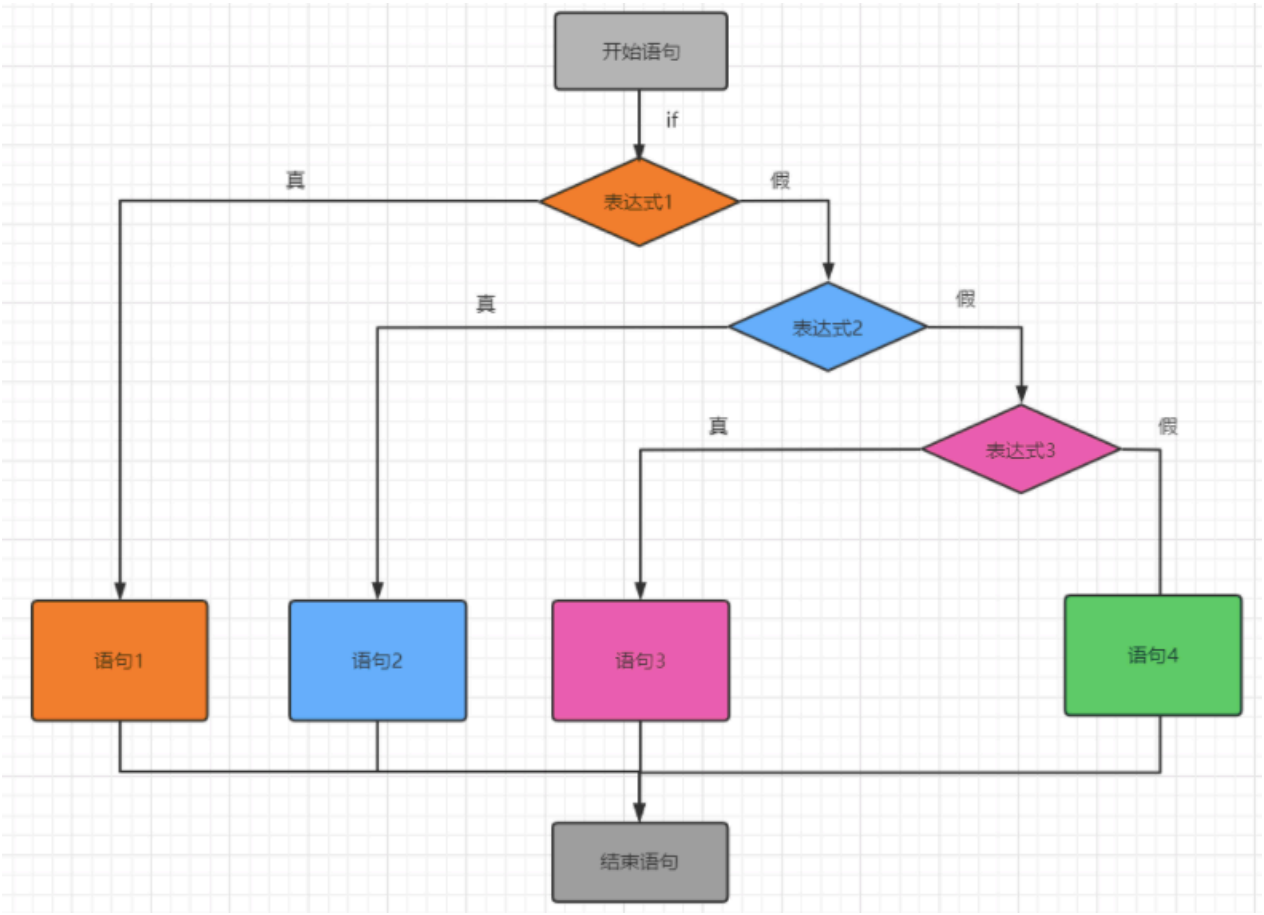

开始语句
if
表达式
真
假
语句A
语句B
结束语句

```
#if(如果)-else(否则)
#场景：如果明天下雨，我们就去钓鱼，否则就在家睡觉
# if 明天下雨---条件：
#      钓鱼
# else:
#      回家睡觉

num = int(input('enter a num:'))
if num > 10:
    print('num大于10')
else:
    print('num小于等于10')
```

- 多分支语句
  - 多分支即从比双分支更多的分支选择一支执行。语法格式：

```
if 表达式 1:
    代码块 1
elif 表达式 2:
    代码块 2
elif 表达式 3:
    代码块 3
...# 其它elif语句
else:
    代码块 n
```

- 

- 案例:猜数字游戏

```
num = 9
gus = input('enter a num:')
gus = int(gus)
if gus > 9:
    print('您猜的数字大了')
elif gus == 9:
    print('您猜对了')
else:
    print('您猜的数字小了')
```

循环控制语句

在不少实际问题中有许多具有规律性的重复操作，因此在程序中就需要重复执行某些语句。一组被重复执行的语句称之为循环体，能否继续重复，决定循环的终止条件。

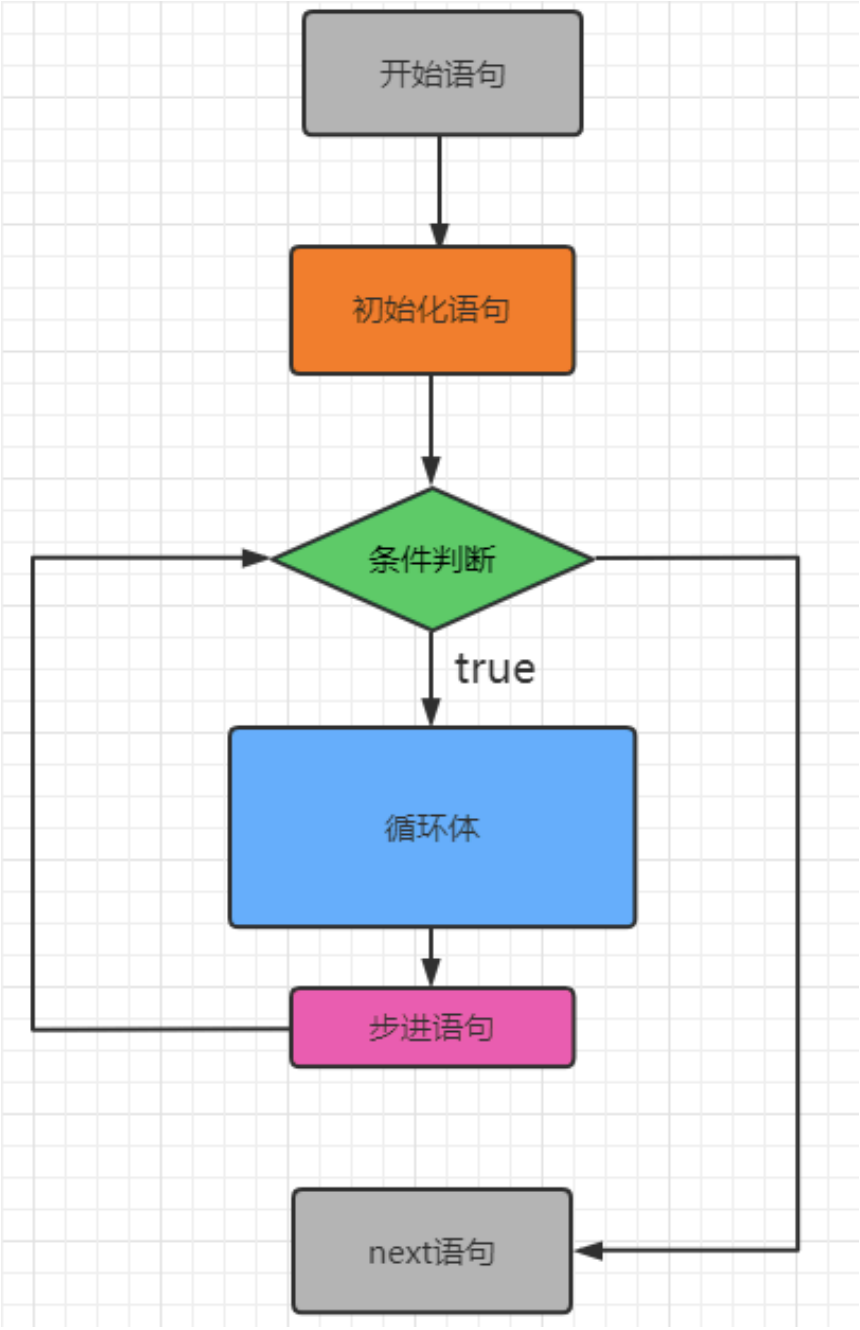
Python语言中的循环语句支持 while循环（条件循环）和for循环（遍历循环）。

while循环

- 语法：

```
while 表达式:
    循环体
```

-



- 案例：循环打印十遍"hello world"

```
i = 0
while i < 10: #while关键字后面的代码是条件
    #下面两行代码：循环体（用来被重复执行）
    print('i love you!')
    i = i + 1
    #执行步骤：1.判断条件是否成立，如果条件成立  2.执行循环体  3.判断条件  4.执行循环体....n.条件不成立，结束循环
```

- 案例：求解1-10累加和

```
自己设计实现一下
```

for循环

- 语法格式

```
for 迭代变量 in 字符串|列表|元组|字典|集合:
    代码块
```

- 格式中，迭代变量用于存放从序列类型变量中读取出来的元素，所以一般不会在循环中对迭代变量手动赋值；代码块指的是具有相同缩进格式的多行代码（和 while 一样），由于和循环结构联用，因此代码块又称为循环体。

```
for i in "hello world":
    print(i)

for name in ["张三",'李四',"王五"]:
    print(name)

for i in range(10): # range函数
    print(i)
```

死循环

- 永远都不会结束的循环。
- 但有我们是会需要无限循环。例如Web服务器响应客户端的实时请求。

循环嵌套

- f判断可以嵌套，while和for当然也可以嵌套。但是建议大家不要嵌套3层以上，那样的效率会很低。

退出循环

如果想提前结束循环（在不满足结束条件的情况下结束循环），可以使用break或continue关键字。

- break
  - 通常情况下的循环要么执行出错，要么死循环，要么就

只能老老实实等它把所有的元素循环一遍才能退出。如果想在循环过程中退出循环，怎么办？用break语句！

- break只能用于循环体内。其效果是直接结束并退出当前循环，剩下的未循环的工作全部被忽略和取消。注意当前两个字，Python的break只能退出一层循环，对于多层嵌套循环，不能全部退出。

```
for i in [1,2,3,4,5]:
    if i == 3:
        break  #强制结束循环
    print('i love you')
```

- Continue
  - 与break不同，continue语句用于跳过当前循环的剩余部分代码，直接开始下一轮循环。它不会退出和终止循环，只是提前结束当前轮次的循环。同样的，continue语句只能用在循环内。

```
for i in [1,2,3,4,5]:
    if i == 3:
        continue  #结束当次循环，开始下一次新的循环
    print('i love you')
```

案例1：实现完整的猜数字游戏

- 规则如下：
  - 可以不间断的进行猜数字游戏环节，找到猜对了，结束程序，猜不对，可以不断的进行游戏，并且需要提示用户猜打了还是猜小了。
  - 拓展功能实现：
    - 1.最后需要统计出，用户猜了多少次才猜对。
    - 2.每一个用户的初始分数为100，每猜错一次扣5分，最后程序结束，统计用户的得分

```
'''
    规则如下：

- 可以不间断的进行猜数字游戏环节，找到猜对了，结束程序，猜不对，可以不断的进行游戏，并且需要提示用户猜打了还是猜小了。
- 拓展功能实现：
  - 1.最后需要统计出，用户猜了多少次才猜对。
  - 2.每一个用户的初始分数为100，每猜错一次扣5分，最后程序结束，统计用户的得分
'''
score = 100 #用户初始的分值
count = 0 #用来记录猜的次数
#写出大致的框架，然后补充细节
```

```
guess_num = 15  #提供了一个让用户猜的数字
#让用户去猜（实现和用户进行交互）
#input就是一个工具：获取用户从键盘上录入数据
while 1: #死循环：没有结束的循环
    count += 1 #count = count + 1,使得猜的次数增加一个1
    num = input('enter you guess num:')
    num = int(num) #类型转换，将字符串的num转成数据类型的num
    #判断：guess_num和num是否一样
    if guess_num == num:#猜对了
        print('猜对了，您一共猜的次数为：',count)
        print('最终得分为:',score)
        break #结束循环
    elif guess_num < num:#猜大了
        print('猜大了')
        score -= 5
    else:#猜小了
        print('猜小了') #至此一局游戏结束
        score -= 5
```

案例2：算法实现

- 规则如下：有三个变量a,b,c,其值均为自然数，其中a2+b2=c2且a+b+c=1000,求出满足要求a，b，c的所有组合。

```
for a in range(1001): #a的取值就是0-1000
    for b in range(1001):#b的取值就是0-1000
        c = 1000 - a - b #得到了c的取值
        if a + b + c == 1000 and a**2 + b**2
== c**2:
            print(a,b,c)
```

案例3：算法实现

- 找寻出列表中存储元素的最大值，且将其移动到列表最后的位置

```
alist = [2,8,61,7,6]
#如何找寻出列表中最大元素的值？需要两两元素比较
for i in range(len(alist)-1):#range(4)==>
[0,1,2,3]
    if alist[i] > alist[i+1]:
        #两个元素交换位置
        alist[i],alist[i+1] =
alist[i+1],alist[i]
print(alist)
```