

程序设计基础 ——循环

林大 经管学院 瞿华

循环

- 一. 基本循环
- 二. 应用：迭代计算
- 三. 应用：计算机仿真
- 四. 应用：统计模拟
- 五. 多重循环
- 六. 应用：AI与穷举法

循环

- 一. **基本循环**
- 二. 应用：迭代计算
- 三. 应用：计算机模拟
- 四. 多重循环
- 五. 应用：AI与穷举法

一、基本循环

- ❖ 我们都听说这个笑话：
 - 一个小孩学写字，学会“一、二、三”以后就觉得都会不用在学了。然后有天他爸爸让他写个“万”字，他写了一天也没写完。。
- ❖ 我们之前所写的程序，无法发挥计算机的速度快的优势，因为：
 - 逐行执行程序中的语句
 - 我们写多少行语句，计算机就执行多少处理！
- ❖ 如何能够解决这个问题呢？
- ❖ 类比，小孩子做了错事要惩罚他写字，但他会写的字很少，怎么办？
 - 每个字罚写100遍！

1.1 循环概述 (2)

- ❖ 事实上，在现实中也经常会进行重复性的处理，例如：
 - 每天进行体育锻炼、每周来上课
 - 在商店挑选衣服：逐一试穿每一件选中的衣服
 - 计算某个商店当日的总营业额：将当日发生的每一笔交易的金额逐一累加
 -
- ❖ 同样，我们可以让计算机来反复执行一段代码。
 - **实际上，充分发挥计算机处理能力的关键就是找出让其重复进行的步骤！**
- ❖ 循环（Loop）结构：在一定条件下反复执行某段代码的程序结构。

1.1 循环概述 (3)

- ❖ 显然，循环结构中应该包含这些信息：
 - 应重复执行哪些操作？
 - 重复到何时结束，或者应重复多少次？
- ❖ 循环结构的基本形式就是：重复执行一段语句，直到某种条件判断失败（或成立）。
 - 重复执行的这段语句称为**循环体**。告诉计算机“应该重复做什么”
 - 控制重复执行的条件判断称为**循环条件**。告诉计算机“在什么情况下重复”或“什么情况下结束重复”。

1.2 while循环

- ❖ while循环是最基本的循环，我们来看一个例子。
- ❖ 例子1-2-1：使用海龟作图绘制五角星。

```
from easygraphics.turtle import *  
import math  
  
create_world(600, 400)  
n=0  
while n<5:  
    forward(200)  
    left_turn(144)  
    n+=1  
pause()  
close_world()
```

1-2-1.五角星.py

1.2 while循环(2)

❖ while是常用的循环语句之一。其形式为：

while 逻辑表达式：

代码段(suite)

❖ 这里的逻辑表达式就是**循环条件**，代码段就是**循环体**。

❖ 念作：“**当……成立时，反复执行下面的语句**”，即：

(1) 判断一下循环条件的值为True吗？

(2.1) 为False，则while语句执行结束；

(2.2) 为True，则执行循环体中的语句。返回 (1) 。

1.2 while循环(3)

❖ 因此，上个程序中循环的执行过程就是：

1. n小于5吗？ 是的 (n为0)。海龟前进200， 左转144度， n加1；
2. n小于5吗？ 是的 (n为0)。海龟前进200， 左转144度， n加1；
3. n小于5吗？ 是的 (n为0)。海龟前进200， 左转144度， n加1；
4. n小于5吗？ 是的 (n为0)。海龟前进200， 左转144度， n加1；
5. n小于5吗？ 是的 (n为0)。海龟前进200， 左转144度， n加1；
6. n小于5吗？ 不是 (n为5)。循环结束

1.2 while循环(4)

- ❖ 例子1-2-2：统计某位同学到目前位置已修的总学分。
- ❖ 分析：
 - 什么是“已修的总学分”？如何计算？需要哪些信息才能计算？
- ❖ 基本思路：
 - 逐一输入到目前为止考核通过的每一门课程的学分
 - 将所有这些课程的学分加到一起，就是已修的总学分。
- ❖ 怎么输入？
 - 从用户友好的角度考虑，我们可以用对话框。但是因为要多次反复输入学分，多次显示和关闭对话框对用户来说会比较麻烦。所以我们采用最简单的input()输入

1.2 while循环(5)

- ❖ 例子1-2-2：统计某位同学到目前位置已修的总学分。
(续一)
- ❖ 怎么输出：
 - 既然使用()输入，那么也使用print()输出即可。
- ❖ 问题：
 - 怎么反复输入和累加学分？
 - 可以用循环，但程序怎么知道所有课程的学分都已经输入了？
 - ✓ 可以规定，如果用户输入的课程学分小于等于0，表示输入结束。
- ❖ 输入和计算过程中是否需要变量？
 - 变量credit，用于保存本次循环中输入的课程学分
 - 变量total，用于累计总的学分
 - 变量count，用于累计总的课程门数

1.2 while循环(6)

```
total=0
count=0
credit=float(input(f"请输入第{count+1}门课的学分:"))
while credit>0:
    total += credit
    count+=1
    credit=float(input(f"请输入第{count+1}门课的学分:"))

print(f"一共{count}门课，总学分为{round(total,1)}")
```

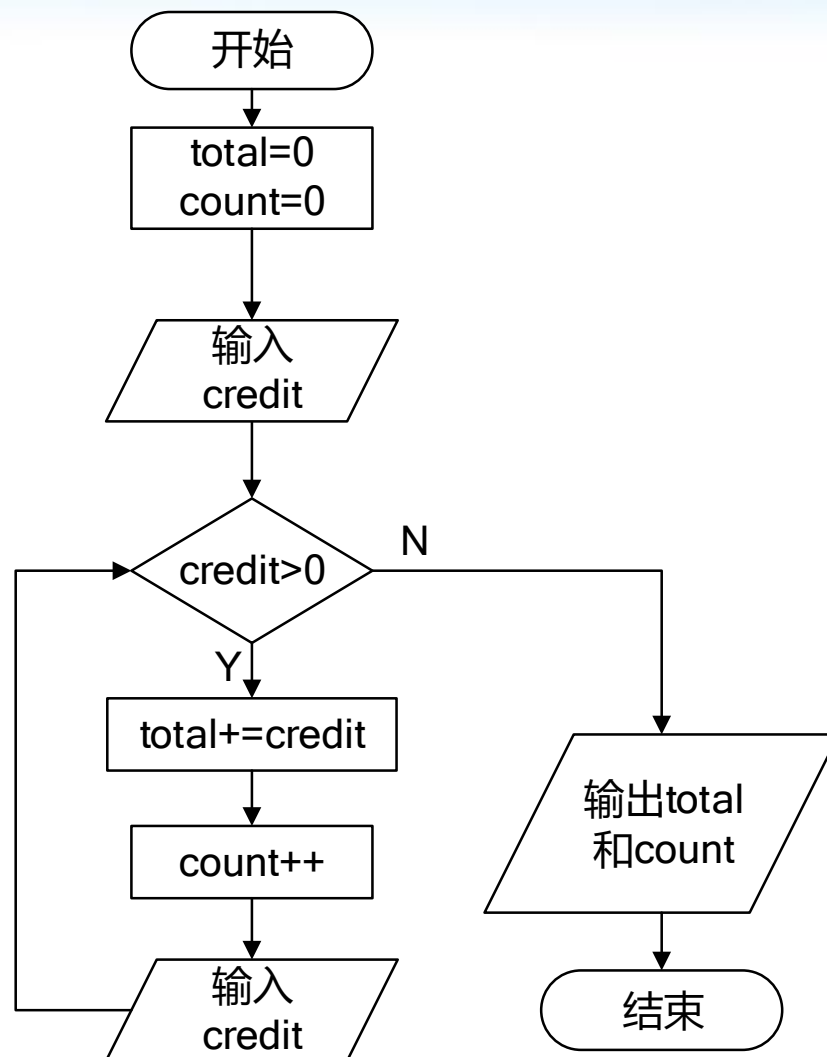
1-2-2.总学分.py

1.2 while循环(7)

❖ 程序逻辑流程图：

❖ 思考：

- 如果第一次输入的credit为0, 那么循环体是否会被执行?



1.2 while循环(8)

❖ 例1-2-3： 看下面的程序，预测其结果。

➤ 注： `fractions.Fraction(1,i)` 是一个对象，表示分数 $1/i$ （ i 分之1）

```
import fractions

n=int(input("请输入n:"))
i=1
total=0
while i<=n:
    total=total+fractions.Fraction(1,i)
    i+=1
print(f"结果是{total}")
```

1-2-3.while练习.py

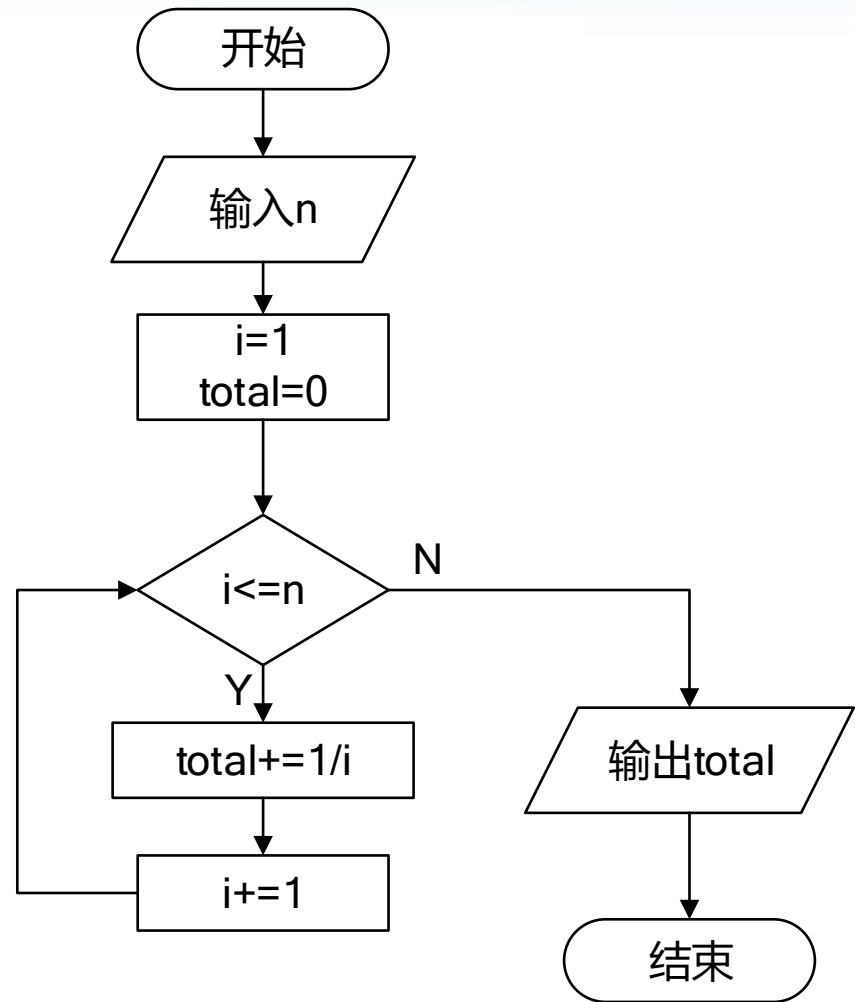
1.2 while循环(9)

n=5时，程序执行过程中关键变量的值

时点	i	total
第1次while判断前	1	0
第2次while判断前	2	1
第3次while判断前	3	$1+1/2$
第4次while判断前	4	$1+1/2+1/3$
第5次while判断前	5	$1+1/2+1/3+1/4$
第6次while判断前	6	$1+1/2+1/3+1/4+1/5$

1.2 while循环(10)

- ❖ 程序逻辑流程图：
- ❖ 思考：
 - 如果循环体最后忘了写 $i+=1$ ；
会怎样？
- ❖ **死循环**是使用循环时最容易出现错误！
- ❖ 如果程序长时间不出结果，
就要检查是否存在死循环。



趣题： 买啤酒

- ❖ 学习了while循环，我们就可以开始解决一些问题了
- ❖ 例1-2-4. 已知：
 - 一瓶啤酒 2 元钱
 - 4 个瓶盖可换 1 瓶啤酒
 - 2 个空瓶可换 1 瓶啤酒
- ❖ 求： 10 元钱总共可以喝到几瓶啤酒
- ❖ 显然，这个问题的处理实际上就是不停的循环：
 - 买酒或用之前剩下的瓶盖和空瓶换酒
 - 把手上的酒都喝光（留下瓶盖和空瓶）
- ❖ 因此我们可以用while循环来编写程序求解。

趣题： 买啤酒

❖ 分析：

- 输入：由于题目条件是确定的，所以不需要用户输入，直接在程序开头用变量表示初始条件即可。
- 输出：简单起见用print()即可
- 程序的核心就是while循环求解。循环应怎么写呢？
- **写循环，我们首先应该考虑循环中要用到哪些变量。**显然，可以有下面的几个变量：
 - ✓ 手上现有的啤酒数量:beer
 - ✓ 手上现有的瓶盖数量:lid
 - ✓ 手上现有的酒瓶数量:bottle
 - ✓ 一共已喝掉的酒数量: total
- **然后，我们要考虑循环开始前各变量的初始状态。**开始的时候有10元钱，显然这10元钱都可以换成酒。因此简单起见，我们直接beer=5。剩下的lid, bottle和total显然都应该是0。

趣题： 买啤酒

❖ 分析（续一）：

- 第三，我们考虑循环体的写法。刚才我们已经分析过，每次处理有两大步，即换酒和喝酒。我们来分别处理：
- 换酒(注意顺序)：
 - ✓ 计算本轮啤酒数=上轮剩下啤酒数+上轮剩下瓶盖数//4+上轮剩下酒瓶数//2,即 $beer = beer + lid // 4 + bottle // 2$ (瓶盖和酒瓶分别换酒)
 - ✓ 计算换剩下的瓶盖数: $lid = lid \% 4$
 - ✓ 计算换剩下的酒瓶数: $bottle = bottle \% 2$

趣题： 买啤酒

❖ 分析（续二）：

➤ 喝酒（注意处理顺序）：

- ✓ 计算酒喝光后剩下的瓶盖数： $lid = lid + beer$
- ✓ 计算酒喝光后剩下的酒瓶数： $bottle = bottle + beer$
- ✓ 计算已喝光的酒数： $total += beer;$
- ✓ **把酒喝光**： $beer = 0$

❖ 最后，我们考虑循环结束条件：

- 最显然的条件就是无法再换酒。即 $lid < 4$ and $bottle < 2$ 。但这样会导致一开始循环就不执行。（为什么？想想初始条件）
- 如何通过修改初始条件，让程序能够完成计算？
- 怎么把循环结束条件改成while中的循环条件（循环继续进行）？

❖ 程序见1-4.beer.py

1.2 while循环(8)

```
beer=0
total = 5
bottle = 5
lid = 5
while lid >= 4 or bottle >=2:
    # 换酒
    beer = beer + lid //4 + bottle //2
    lid = lid % 4
    bottle = bottle % 2
    # 喝酒
    total = total + beer
    lid = lid + beer
    bottle = bottle + beer
    beer = 0
print(f"总共喝了{total}瓶，剩瓶盖{lid}个，剩酒瓶{bottle}个")
```

1.3 for循环

- ❖ 在很多时候，我们需要：
 - 直接通过指定循环的次数
 - 或者对某个序列中的对象逐一进行循环。例如，依次对自然数序列1, 2, 3, 4, 5中的数字逐一循环
- ❖ 在这两种情况下，我们可以用for循环。
 - 显然，前一种情况可以用后一种情况来实现。需要循环n次，我们对0,1,2,3,...,n-1逐一进行循环即可（恰好循环了n次）。
- ❖ 例1-3-1：阅读程序1-3-1.数列.py，并观察其结果。

```
n=int(input("请输入n:"))  
for i in range(n):  
    print(i)
```

1-3-1.数列.py

可以看到，每行依次输入了0, 1, 2, 3,...,n-1，共输出了n个数。

1.3.1 可迭代对象

❖ for循环的基本形式就是：

```
for 变量 in 可迭代对象:  
    代码段（循环体）
```

- ❖ 我们可以念做 “用变量依次循环可迭代对象里的每一项”
- ❖ 出现在for语句中的变量称为 “**循环变量**”
- ❖ 其中，**可迭代对象(iteratable object)**是一个存放数据的容器，可以通过迭代（iteration）的形式来访问其中的每一项数据
- ❖ 如果一个变量x中存放的是可迭代对象，那我们就可以使用iter()函数来得到它的迭代器（iterator），并使用迭代器来依次访问该可迭代对象中存放的数据。

1.3.1 可迭代对象 (2)

❖ 我们可以用`list()`函数将可迭代对象转化为列表（后面章节会讲），然后查看其内容。

❖ 试一试：在交互式环境中输入如下代码，查看结果*：

```
list(range(5))
```

❖ 可见，`range(n)`的作用就是生成一个由`0,1,2,...,n-1`，一共`n`个整数构成的数列（可迭代对象）。

❖ `range()`函数还可以有下面的形式：

- `range(n,m)`：得到数列`n,n+1,n+2,...,m-1`。
- `range(n,m,s)`:(`s`大于0) 得到数列：`n,n+s,n+2*s,...`(数列最后一个数小于`m`)
- `range(n,m,-s)`:(`s`大于0) 得到数列：`n,n-s,n-2*s,...`(数列最后一个数大于`m`)

*注：在交互式环境下会直接显示每行语句的执行结果（返回值），在程序中需要使用`print()`输出。

1.3.1 可迭代对象 (3)

- ❖ 练习：使用`range()`语句生成如下数列：
 - 1,2,3,4,5
 - 1,2,3,...,n
 - 2,4,6,8,10
 - n,n-1,n-2,...,1
 - n-1,n-2,...,0
- ❖ 刚开始学python时搞错`range()`函数所产生数列的起、止是正常现象。在写程序时可以随时打开一个交互式环境，实验和确认`range()`的用法即可。
 - 可以方便的在交互式环境中进行尝试，正是python最大的优点之一！

1.3.2 for工作原理*

- ❖ 例1-3-2 在交互式环境中逐行输入并执行下面这段代码，查看结果：

```
x=range(5)
iterator = iter(x)
print(next(iterator))
print(next(iterator))
print(next(iterator))
print(next(iterator))
print(next(iterator))
print(next(iterator))
```

我们可以看到，第一次运行`next(iterator)`，返回的是0；第二次返回的是1；第三次是2；。。。最后一次运行，报`StopIteration`异常。

for循环工作原理：生成可迭代对象的迭代器，每次循环前，用`next(迭代器)`取出可迭代对象中的下一项值。如果取值时遇到`StopIteration`异常，则循环结束。

1.3.3 控制循环次数

- ❖ 我们已经看到，`range()`的作用是产生一个数列，而`for`语句就是通过从前到后逐个访问数列中的每一项，来控制循环。
- ❖ 因此，对于下面的循环：

```
for i in range(n):  
    代码段（循环体）
```

- ❖ 我们通常这么念："i从0循环到n-1，每次加1"
- ❖ **for语句+range()循环最基本的用途就是控制循环次数。**
- ❖ 例1-3-3：海龟作图绘制多边形。
 - 编写一个海龟作图函数，绘制边长为size的正n边形。

1.3.3 控制循环次数(2)

❖ 例1-3-3：海龟作图绘制多边形。

➤ 编写一个海龟作图函数，绘制边长为size的正n边形。

❖ 函数分析：

➤ 输入：

✓ 两个参数——边数n，边长size

➤ 输出：

✓ 直接在屏幕上绘制相应的图形。无返回值。

➤ 处理：直接使用循环完成绘制

❖ 主循环分析：

➤ 循环中用到的各变量及其循环前初始状态：只需要参数n和size

➤ 每次循环的处理：用fd()向前画一条长度为n的线段，然后旋转 $180 - ((n - 2) * 180 / n)$ 度。

➤ 循环条件：一共循环n次

1.3.3 控制循环次数(3)

```
from easygraphics.turtle import *
from easygraphics.dialog import *

def polygon(n,size):
    for i in range(n):
        fd(size)
        lt(180-(n-2)*180/n)

create_world(800,600)
results=get_many_strings(labels=['n=','size='])
n=int(results[0])
size=int(results[1])
polygon(n,size)
pause()
close_world()
```

1-3-3.polygon.py

1.3.4 生成与应用数列

- ❖ for语句+range()循环的另一个主要用途是生成和利用数列(序列)。
- ❖ 例1-3-4: 编写函数, 已知自然数 n , 计算下面式子 $1 - 1/2 + 1/3 - 1/4 + \dots + (-1)^{n-1}n$ 的值
- ❖ 函数分析:
 - 输入:
 - ✓ 一个参数: 自然数 n
 - 输出:
 - ✓ 一个返回值: 式子的值total。
 - 处理:
 - ✓ 这是一个典型的数列求和计算, 可以使用循环来完成

1.3.4 生成与应用数列(2)

❖ 主循环分析:

- 循环中用到的各变量及其循环前初始状态:
 - ✓ 数列的和total, 循环前初值为0
 - ✓ 数列项item, 循环前不需要初值
- 每次循环的处理:
 - ✓ 根据循环变量i的值, 得到数列对应的第i项 ($1, -1/2, 1/3, \dots$), 赋给item
 - ✓ 将item累加到total中
- 循环条件:用循环变量i依次访问数列 $1, 2, 3, \dots, n$

1.3.4 生成与应用数列(3)

n=5时，循环执行过程中关键变量的值

时点	i	item	total
循环开始前			0
第1次循环结束前	1	1	1
第2次循环结束前	2	-1/2	1-1/2
第3次循环结束前	3	1/3	1-1/2+1/3
第4次循环结束前	4	-1/4	1-1/2+1/3-1/4
第5次循环结束前	5	1/5	1-1/2+1/3-1/4+1/5

1.3.4 生成与应用数列(4)

```
import fractions

def calc(n):
    total = 0
    for i in range(1, n + 1):
        item = fractions.Fraction((-1) ** (i - 1), i)
        total = total + item
    return total

n=int(input("请输入n:"))
print(f"结果是{calc(n)}")
```

1-3-4.数列求和.py

注意range的两个参数! 为什么是1和n+1?

1.4 break语句

- ❖ 除了正常的循环结束外，我们可以在循环内部，使用break语句强制结束并退出循环。利用这一点，我们可以简化循环条件或者结构。
- ❖ 例1-4-1：宿舍6位同学一起下馆子，采用这样的方式来筹集饭费：
 - 从宿舍老大开始，按照年龄从大到小的顺序依次输入每人愿意掏的金额，当总数达到300元时就结束，统计此时掏钱的人数，总金额，以及平均每人可以花的钱数
- ❖ 程序分析：
 - 输入：需要输入每个人的饭费。因为题目中没有说明，简单起见我们可以规定饭费为整数，单位为元，采用input()输入。可以采用6个变量分别保存每个人的饭费，但是我们在分析循环的时候会看到，其实只需要一个变量amount，保存本次循环输入的饭费即可；
 - 输出：掏钱人数count，总金额total，平均每人饭费average
 - 处理：可以采用循环来逐个输入每个人的饭费，并进行统计；

1.4 break语句(1)

❖ 循环分析：

➤ 循环用到的变量和初始值：

- ✓ 循环中需要统计掏钱人数count，总金额total。他们的初值均为0

➤ 循环处理：

- ✓ 读取第i位同学所掏的金额amount
- ✓ 累计掏钱人数count，总金额total

➤ 循环结束条件：

- ✓ 所有6个学生都掏钱了，或者总金额达到300。

❖ 按照上面的分析，如果使用while循环，循环条件中将包含一个用and连接的复合逻辑表达式（注意循环条件和循环结束条件是相反的）。

1.4 break语句(2)

- ❖ 如果循环条件中包含的条件很多很复杂时，会容易出错。我们可以在循环体中结合使用if语句和break语句来简化循环条件判断。
- ❖ 修改后的循环分析：
 - 循环用到的变量和初始值：
 - ✓ 循环中需要统计掏钱人数count，总金额total。他们的初值均为0
 - 循环处理：
 - ✓ 读取第i位同学所掏的金额amount
 - ✓ 累计掏钱人数count，总金额total
 - ✓ 如果总金额达到300，则结束循环 (break)
 - 循环条件：
 - ✓ i从1循环到6(6位同学)

1.4 break语句(3)

```
count = 0
total = 0
for i in range(1, 7):
    amount = int(input(f"请输入第{i}位同学的饭费: "))
    total += amount
    count += 1
    if total >= 300:
        break

average = round(total / count, 2)
print(f"共{i}位同学掏钱了，一共{total}元，人均{average}元")
```

1-4-1.宿舍聚餐.py

注意：range的第二个参数为什么是7？

1.4 break语句(4)

- ❖ 例子1-4-2：使用break语句简化[例1-2-2](#)。
- ❖ 在例1-2-2中，我们为了能够判断是否结束处理，在循环中读入学分，在循环开始前也读入了一次学分。
- ❖ 我们可以这样来修改循环：
 - 将循环条件修改为True
 - 在循环中读入学分后，判断是否小于等于0，是则退出循环(break)

1.4 break语句(5)

```
total=0
count=0
while True:
    credit=float(input(f"请输入第{count+1}门课的学分:"))
    if credit <= 0 :
        break
    total += credit
    count+=1

print(f"一共{count}门课，总学分为{round(total,1)}")
```

1-4-2.总学分break.py

1.5 continue语句

- ❖ 例1-5-1：编写函数，计算1..n中所有不能被5整除的数字的和。
- ❖ 分析：
 - 输入参数：
 - ✓ 一个参数：整数n
 - 输出：
 - ✓ 一个输出：求和结果total
 - 处理方法：
 - ✓ 使用循环从1开始到n，逐个累加符合条件的数

1.5 continue语句

❖ 循环分析：

- 循环中用的变量和初始值：
 - ✓ 求和结果total，初始值为0
- 每次循环处理（循环体）：
 - ✓ 判断i是否能被5整除，如果不能，则累加到total上
- 循环条件：
 - ✓ 用i从1循环到n

❖ 按照这个思路，我们是在循环中满足特定条件时才进行某种处理（把符合的选出来）；

❖ 有时反过来，改为如果满足特定时才不进行某种处理（把不符合的剔除掉），会更符合一般人的思维习惯。这时就可以使用continue语句。

1.5 continue语句

- ❖ continue语句的作用：跳出本次循环处理，继续下一次循环
- ❖ 使用continue修改的循环思路分析：
 - 循环中用的变量和初始值：
 - ✓ 求和结果total，初始值为0
 - 每次循环处理（循环体）：
 - ✓ 判断i是否能被5整除，如果能，则跳出本次循环(continue)
 - ✓ 将i累加到total上
 - 循环条件：
 - ✓ 用i从1循环到n

1.5 continue语句

```
def calc_total(n):  
    total = 0  
    for i in range(1,n+1):  
        if i % 5 == 0:  
            continue  
        total += i  
    return total  
  
n=int(input("请输入n:"))  
total = calc_total(n)  
print(f"计算结果为{total}")
```

1-5-1.求和continue.py

1.5.1 continue与break比较

❖ 总结：break与continue的异同。

➤ **break语句**的作用：

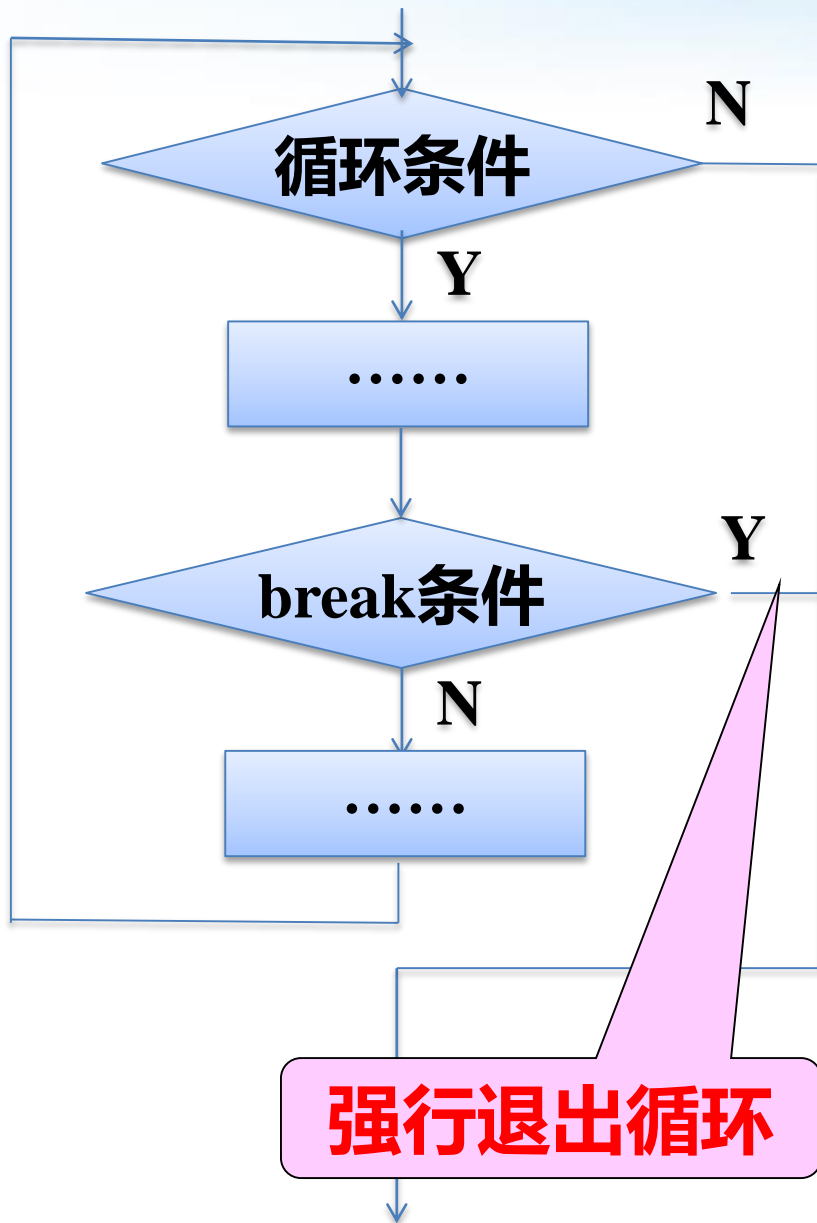
✓ 跳出当前的循环体。

✓ 即提前结束当前循环的执行，接着执行循环下面的语句。

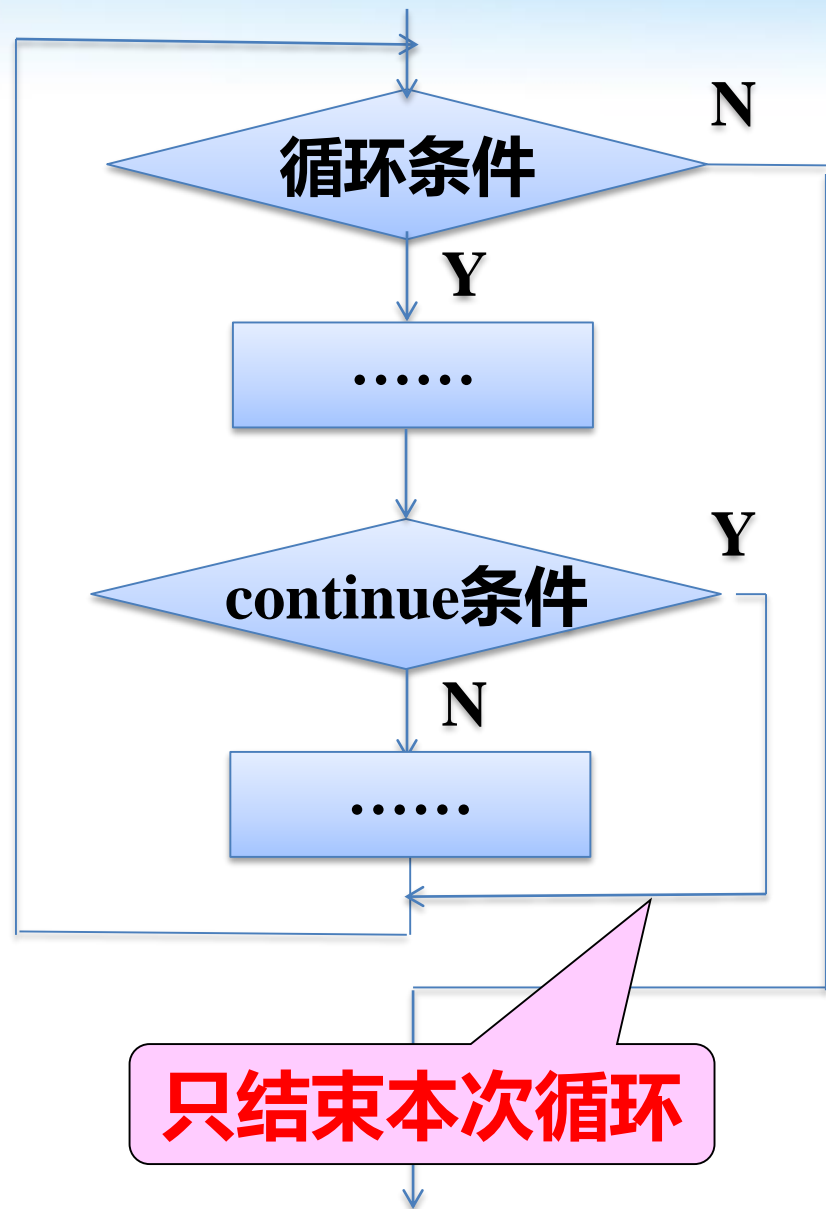
➤ 有时候不需要结束整个循环，而是**结束本次循环体执行步骤，进行下一次循环。**

✓ 这是**continue语句**的作用。

break语句



continue语句



循环

- 一. 基本循环
- 二. **应用：迭代计算**
- 三. 应用：计算机模拟
- 四. 多重循环
- 五. 应用：AI与穷举法

2.1 退休金计算

- ❖ 例2-1-1: 张三25岁开始工作, 每年存1万作为退休基金。已知存款年利率为2.5%, 等40年后他65岁退休时, 退休基金账户中有多少钱?
- ❖ 分析:
 - 输入: 常量 年利率 $rate=0.025$, 退休时存款年数 $n=40$, 每年存入金额 $deposit=10000$
 - 输出: 退休时账户余额 $balance$
 - 处理方法: 使用循环来模拟每年的存款动作, 逐年计算累计余额。

2.1 退休金计算 (2)

❖ 循环处理分析：

- 循环中用到的变量及初始值：
 - ✓ 账户余额balance=0, 当年利息interest (不需要初始值)
- 每次循环处理 (循环体) :
 - ✓ 计算当年利息interest: 余额*年利率
 - ✓ 更新余额balance: 在原有余额的基础上, 累加当年利息和新存入金额
- 循环条件:
 - ✓ 循环n次

2.1 退休金计算 (3)

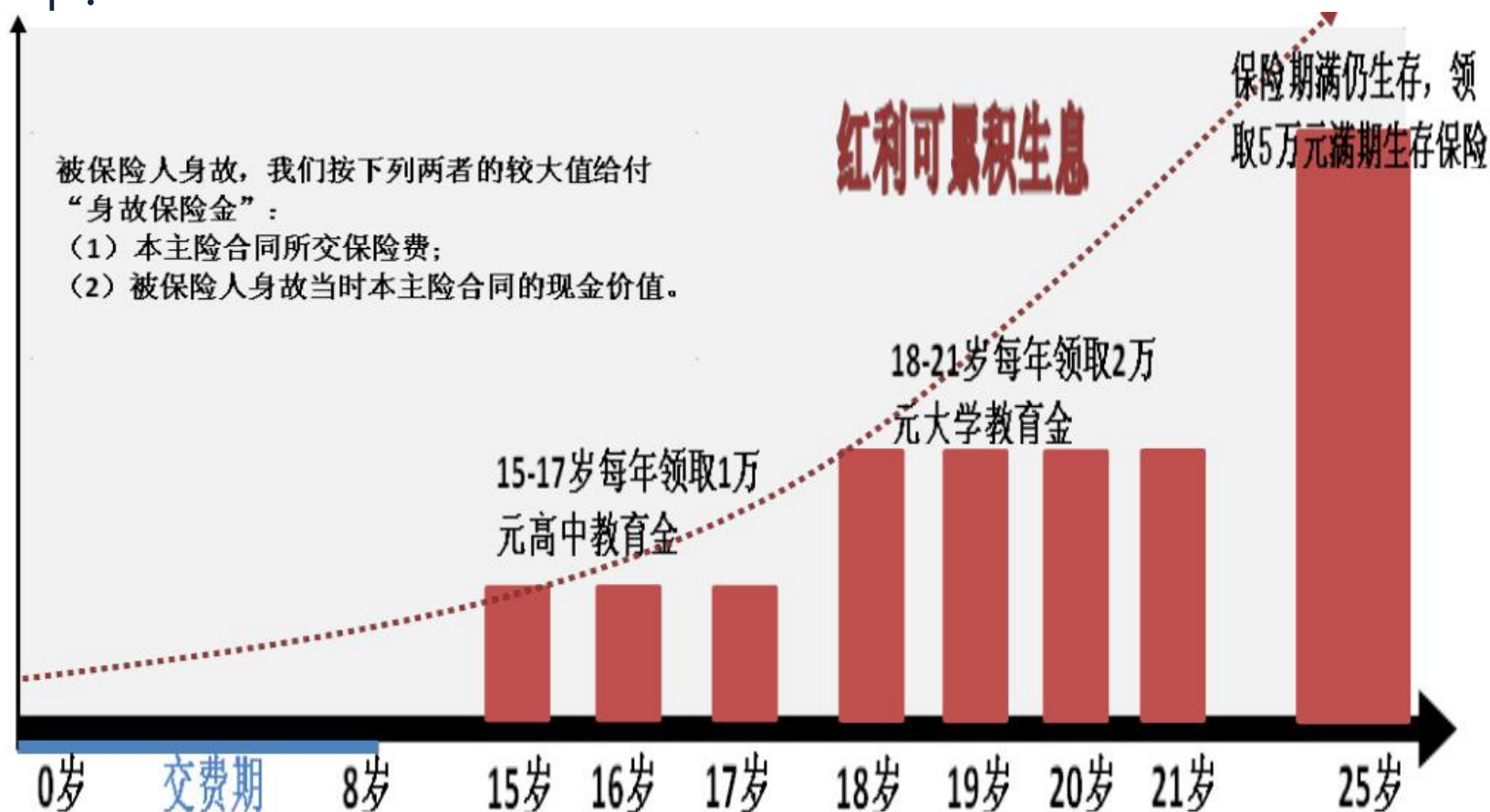
```
rate = 0.025
deposit = 10000
balance = 0
n=40
for i in range(n):
    interest = balance * rate
    balance = balance + interest + deposit
    balance = round(balance,2)
    print(f"第{n}年账户余额为{balance}")

print(f"{n}年后，账户余额为{balance}")
```

2-1-1.养老金.py

2.2 保险计算

- ❖ 例2-2-1:0岁男孩，投保平安鸿运英才少儿两全保险（分红型），基本保险金额50000元，0-8岁每年交保险费15350元。之后领取时间及金额如下：



如果未投保,而是按照同样的方案在银行存取款,假设年利为2%,和保险相比哪种更划算?

2.2 保险计算(2)

- ❖ 分析:我们可以假设在银行开立一个存款账户, 然后按照保险中所述的方式从0岁开始存钱到8岁,然后从15岁开始取钱, 到25岁为止, 看看最后账户里还剩多少钱。
- ❖ 如果最后账户的余额大于零, 那么存款就比买保险更合算。
- ❖ 程序设计分析:
 - 输入:
 - ✓ 常量利率 $rate=0.02$
 - 输出:
 - ✓ 最后的账户余额 $balance$
 - 处理:
 - ✓ 用循环逐年模拟存取款和利息累积过程。
 - 规定所有金额变量的单位为元。

2.2 保险计算(3)

❖ 循环设计分析:

- 循环用到的变量和循环前初始值:
 - ✓ 账户余额balance=15350 (第0年存款), 利息interest, 当年存取款额deposit (大于0为存, 小于0为取)
- 每次循环处理 (循环体) :
 - ✓ 计算当年利息interest
 - ✓ 判断当年存取款金额deposit
 - ✓ 计算当年余额balance
- 循环条件:
 - ✓ 表示年龄的变量age从1循环到25

```
rate = 0.02
balance = 15350
for age in range(1,26):
    interest = balance * rate
    if 0<=age<=8:
        deposit = 15350
    elif 15<=age<=17:
        deposit = -10000
    elif 18<=age<=21:
        deposit = -20000
    elif age == 25:
        deposit = -50000
    else:
        deposit=0
    balance += deposit + interest
    balance = round(balance,2)
    print(f"{age}岁时 存款{deposit} 账户余额为{balance}")

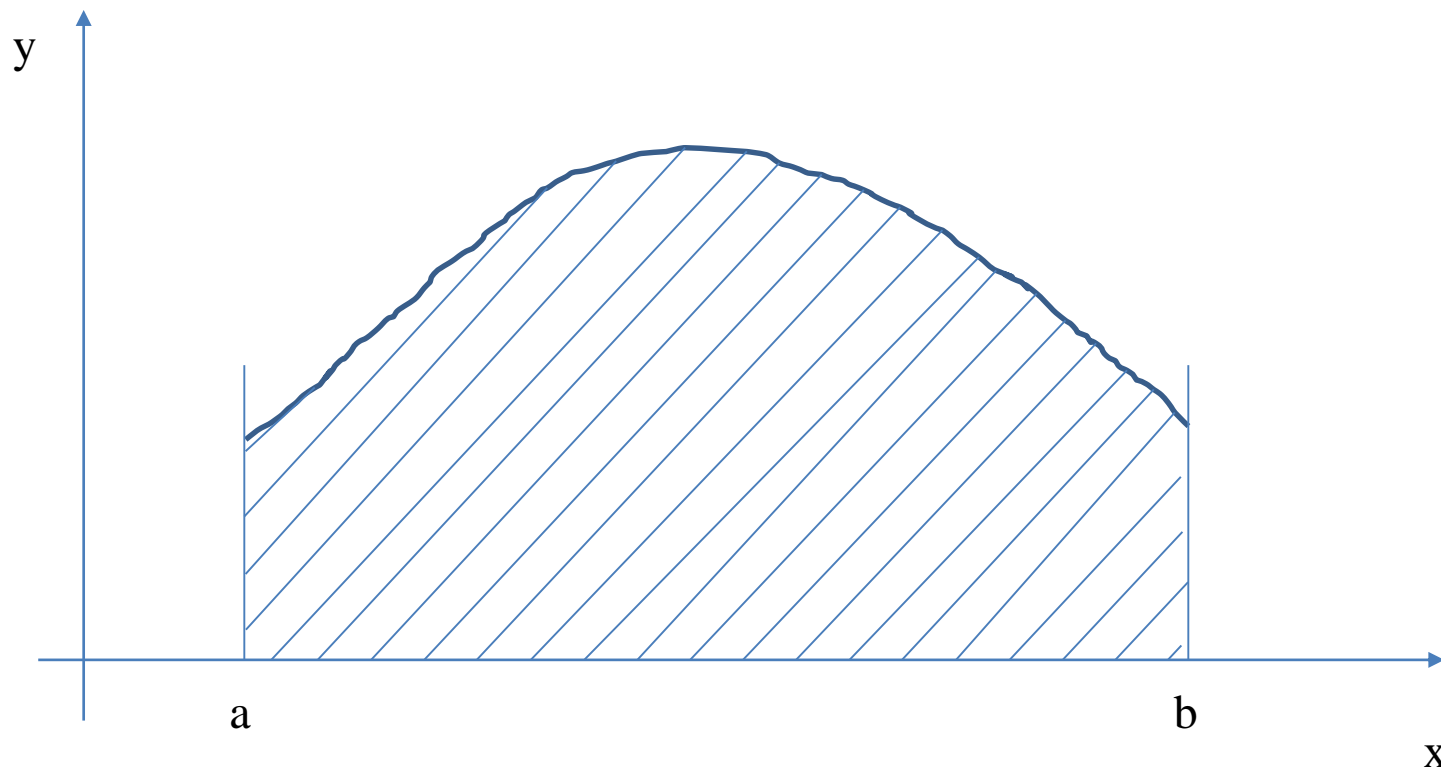
print(f"{age}岁时, 账户余额为{balance}")
```

2.3 数值计算

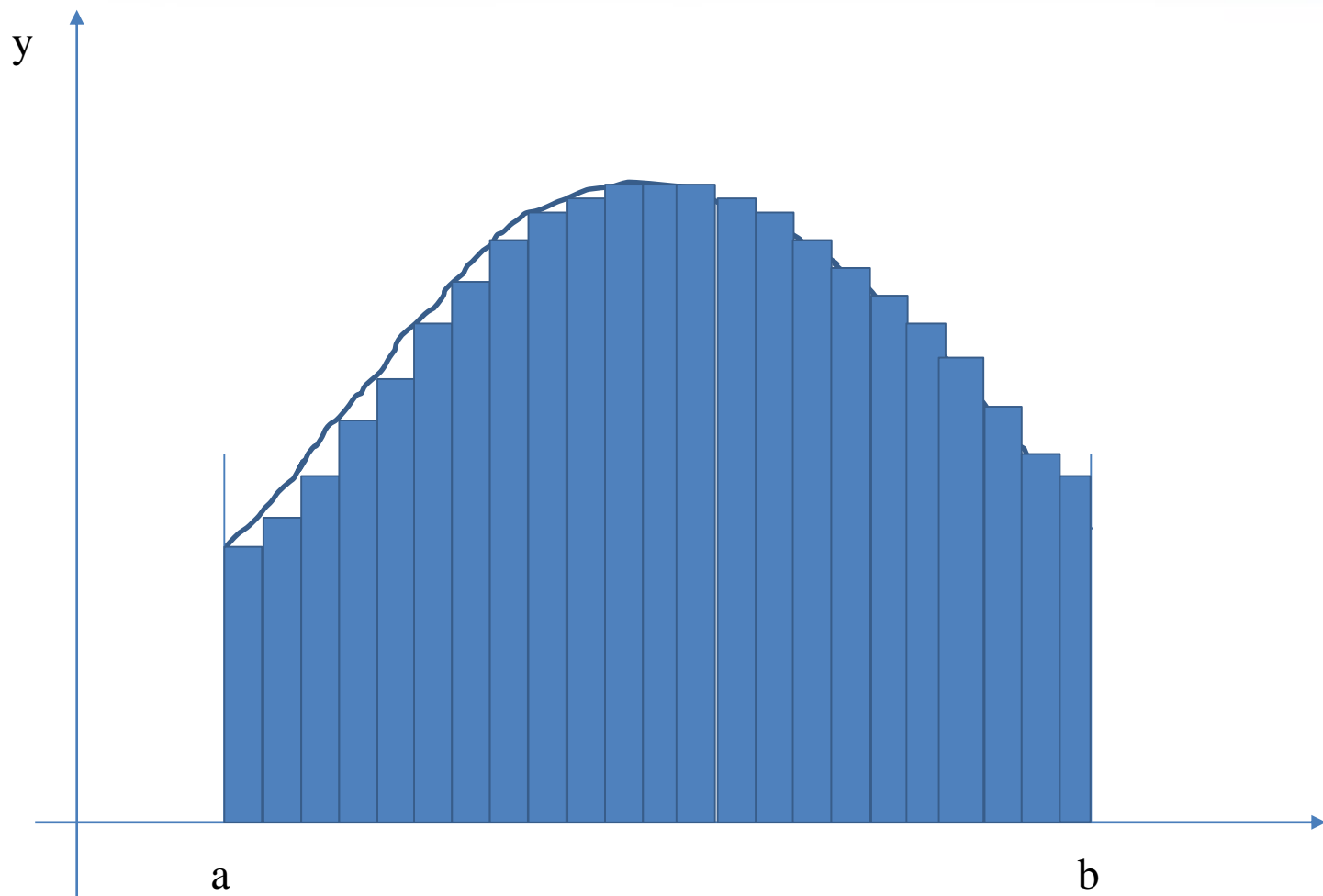
- ❖ 现实中，很多数学问题是没办法求出精确解的。
- ❖ 例： $\sqrt{2}$ ，这是一个无限非循环的实数
- ❖ 例： $\int_0^x \frac{t^3}{e^t - 1} dt$ ，该积分没有解析解
- ❖ 对于这些数学问题，我们只能寻求一定精度要求下的近似解
- ❖ 在过去，求近似解是一件计算量非常大的工作
 - 有了计算机，就好办了
- ❖ **数值计算**：使用数字计算机求数学问题近似解的方法与过程
 - 数值计算是计算机最基本的应用领域之一

2.3.1 近似求积分

❖ 对于定积分 $\int_a^b f(x)dx$ ，实际是求平面直角坐标系下 $f(x)$, $x=a$, $x=b$ 以及 x 轴围成的面积。



2.3.1 近似求积分(2)



可以用一系列小矩形的面积的和来近似 $\int_a^b f(x)dx$ x

2.3.1 近似求积分(3)

- ❖ 例2-3-1：求 $\int_0^{\frac{\pi}{2}} \sin(x)dx$
- ❖ 我们可以编写一个函数integration，求任意连续函数 $f(x)$ 在区间 $[a,b]$ 上的定积分。
- ❖ 函数设计分析：
 - 输入：3个参数，函数 f ，区间起始点 a ，区间结束点 b ，每个小矩形的底边边长step(各区间分段长度)
 - 输出：对应的定积分近似值计算结果result
 - 处理方法：使用循环，逐一求出每个小矩形的面积，累加到result上即可。

2.3.1 近似求积分(4)

❖ 循环设计分析:

- 循环中用到的变量及初始值:
 - ✓ 计算结果 $result=0$, 小矩形左下角端点横坐标 x , 小矩形面积 $area$
- 每次循环处理 (循环体) :
 - ✓ 计算小矩形面积: $area=f(x)*step$
 - ✓ 累加结果: $result+=area$
 - ✓ 更新 x 坐标: $x=x+step$
- 循环条件: x 小于 b

2.3.1 近似求积分(5)

```
import math

def integration(f, a, b, step):
    result = 0
    x = a
    while x < b:
        area = step * f(x)
        result += area
        x += step
    return result

result = integration(math.sin, 0, math.pi/2, 0.00001)

print(f"result : 0.8f")
```

2-3-1.定积分.py

注意怎么把一个函数作为参数传给另一个函数

如果想计算 $[0, \pi]$ 上 $\cos(x)$ 的积分, 该怎么做? $3x^2+4$ 的积分呢?

2.3.2 迭代法求平方根

- ❖ 例2-3-2: 对于任意非负实数 a , 可以用下面的牛顿迭代公式来逐渐逼近其平方根 \sqrt{a}

$$x_k = \frac{1}{2} \left(x_{k-1} + \frac{a}{x_{k-1}} \right)$$

- ❖ 通常取 $x_0=a$;当 $x_{k+1}-x_k$ 的绝对值小于需要的精度时, 结束计算

- ❖ 例: 求 $\sqrt{2}$

$$x_0 = 2$$

$$x_1 = \frac{1}{2} \left(x_0 + \frac{2}{x_0} \right) = 1.5$$

$$x_2 = \frac{1}{2} \left(x_1 + \frac{2}{x_1} \right) = 1.416667$$

$$x_3 = \frac{1}{2} \left(x_2 + \frac{2}{x_2} \right) = 1.414216$$

$$x_4 = \frac{1}{2} \left(x_3 + \frac{2}{x_3} \right) = 1.414214$$

.....

2.3.2 迭代法求平方根(2)

- ❖ 因此，我们总结编程任务是——编写函数`sqrt(a)`计算 a 的平方根
- ❖ 函数分析：
 - 输入：两个参数，即实数 a 和要求的精度 ϵ
 - 输出： a 的平方根 root
 - 处理：使用迭代法逐次计算迭代数列的第 k 项 x_k

2.3.2 迭代法求平方根(3)

❖ 循环分析：

- 循环中用到的变量及其循环前初始值：
 - ✓ 迭代次数 $k=0$,第 k 次迭代的结果 $x_k=a$ （因为此时 $k=0$ ）。第 $k-1$ 次迭代的结果 x_{k-1} 。
- 每次循环处理（循环体）：
 - ✓ 更新迭代次数 k ： $k+=1$
 - ✓ 因为开始了新一次循环，所以 x_k 中的值已经是上一次迭代的了。用它来更新 x_{k-1} ： $x_{k-1}=x_k$
 - ✓ 用迭代公式计算新的 x_k ： $x_k=(x_{k-1}+a/x_{k-1})/2$
- 循环结束条件：
 - ✓ x_k 和 x_{k-1} 之差的绝对值小于要求的精度 ϵ

❖ 显然，使用带break的while循环最合适。

```
import easygraphics.dialog as gui
import math

def sqrt(a, epsilon):
    k=0
    xk = a
    while True:
        k+=1
        xk1 =xk
        xk = (xk1+a/xk1)/2
        if math.fabs(xk-xk1)<epsilon:
            break
    root=xk
    return xk

ep=0.0001
a=float(gui.get_string("请输入a:"))
root = sqrt(a,ep)
gui.show_message(f"{a}的平方根是{root}")
```

2.3.3 牛顿法求方程的根

- ❖ 例2-3-2实际上是牛顿迭代法的特例。
- ❖ 牛顿法 (Newton-Raphson method) 是一种求方程 $f(x)=0$ 的近似解的重要方法。
- ❖ 若函数 $f(x)$ 在区间 $[a,b]$ 上二次可导, 则可以将 $f(x)$ 在 $[a,b]$ 上的一点 x_0 附近作泰勒展开:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(\xi)(x - x_0)^2}{2}$$

- ❖ 略去二次项, 得到

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

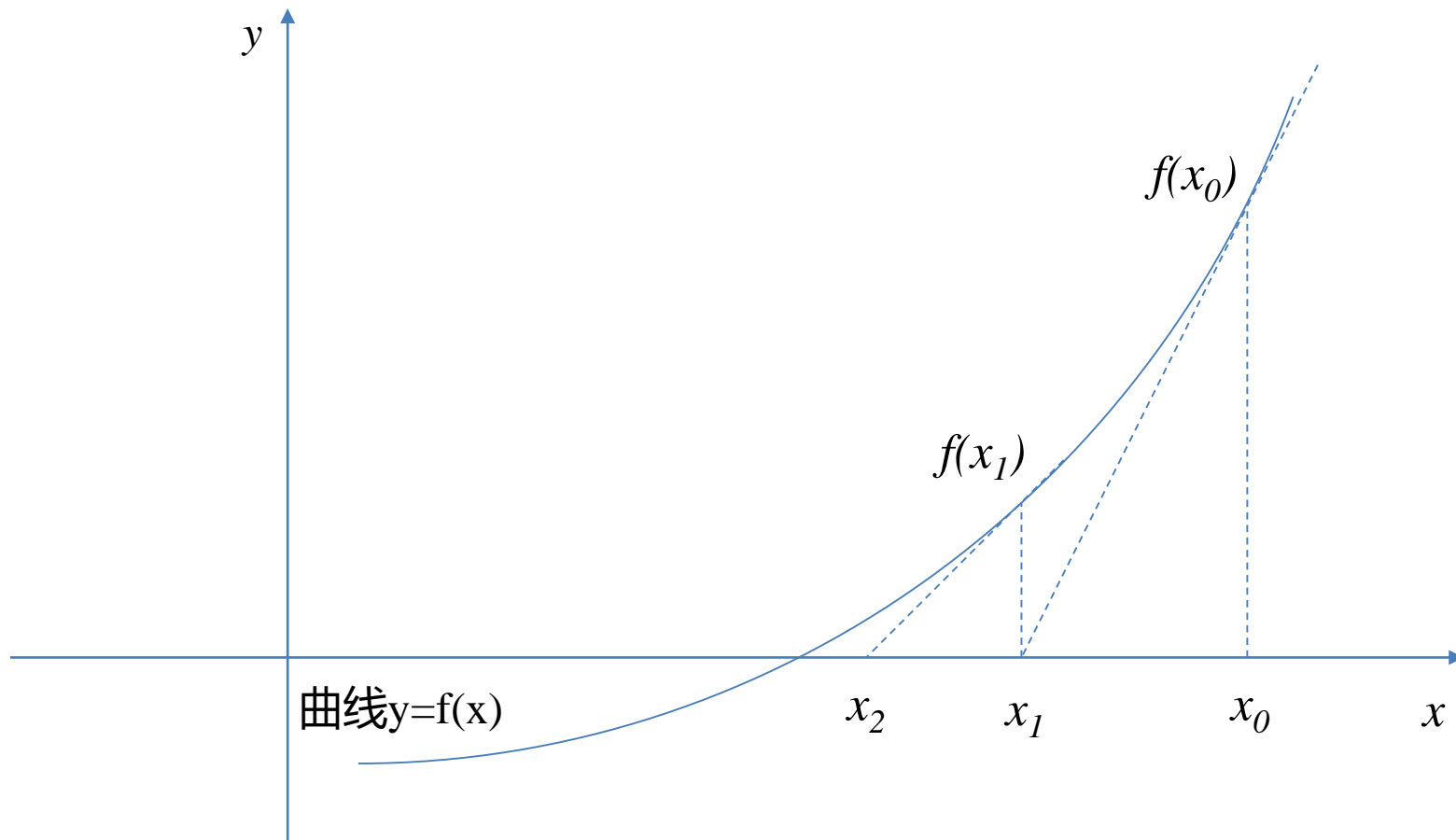
- ❖ 代入 $f(x)=0$, 可得

$$0 \approx f(x_0) + f'(x_0)(x - x_0) \quad \text{即} \quad x \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$

- ❖ 由此可得到求方程 $f(x)=0$ 的根的迭代公式:

$$x_k \approx x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}$$

2.3.3 牛顿法求方程的根(2)



Newton-Raphson法图示

2.3.3 牛顿法求方程的根(3)

- ❖ 在使用牛顿法求根时，需要预先估计迭代的起点 x_0 。
 - 通常可以使用函数图形或根据经验判断
- ❖ 例2-3-3：求方程 $2x^3 - 4x^2 + 3x - 6 = 0$ 在 $x=1.5$ 附近的根。
- ❖ 分析：
 - $f(x) = 2x^3 - 4x^2 + 3x - 6$
 - $f'(x) = 6x^2 - 8x + 3$
 - 因此，牛顿迭代公式为：
$$x_k \approx x_{k-1} - \frac{2x_{k-1}^3 - 4x_{k-1}^2 + 3x_{k-1} - 6}{6x_{k-1}^2 - 8x_{k-1} + 3}$$
 - 使用该公式迭代到 $x_{k+1} - x_k$ 的绝对值小于0.000000000001为止

2.3.3 牛顿法求方程的根(4)

- ❖ 因此，我们总结编程任务是——编写函数`sqrt(a)`计算 a 的平方根
- ❖ 函数分析：
 - 输入：两个参数，即实数 a 和要求的精度 ϵ
 - 输出： a 的平方根 root
 - 处理：使用迭代法逐次计算迭代数列的第 k 项 x_k

2.3.2 迭代法求平方根(3)

❖ 循环分析：

➤ 循环中用到的变量及其循环前初始值：

- ✓ 迭代次数 $k=0$,第 k 次迭代的结果 $x_k=a$ （因为此时 $k=0$ ）。第 $k-1$ 次迭代的结果 x_{k-1} 。

➤ 每次循环处理（循环体）：

- ✓ 更新迭代次数 k ： $k+=1$
- ✓ 因为开始了新一次循环，所以 x_k 中的值已经是上一次迭代的了。用它来更新 x_{k-1} ： $x_{k-1}=x_k$
- ✓ 用迭代公式计算新的 x_k ： $x_k=(x_{k-1}+a/x_{k-1})/2$

➤ 循环结束条件：

- ✓ x_k 和 x_{k-1} 之差的绝对值小于要求的精度 ϵ

❖ 显然，使用带break的while循环最合适。

```
import math

def f(x):
    return 2 * x ** 3 - 4 * x ** 2 + 3 * x - 6

def f_derivative(x):
    return 6 * x ** 2 - 8 * x + 3

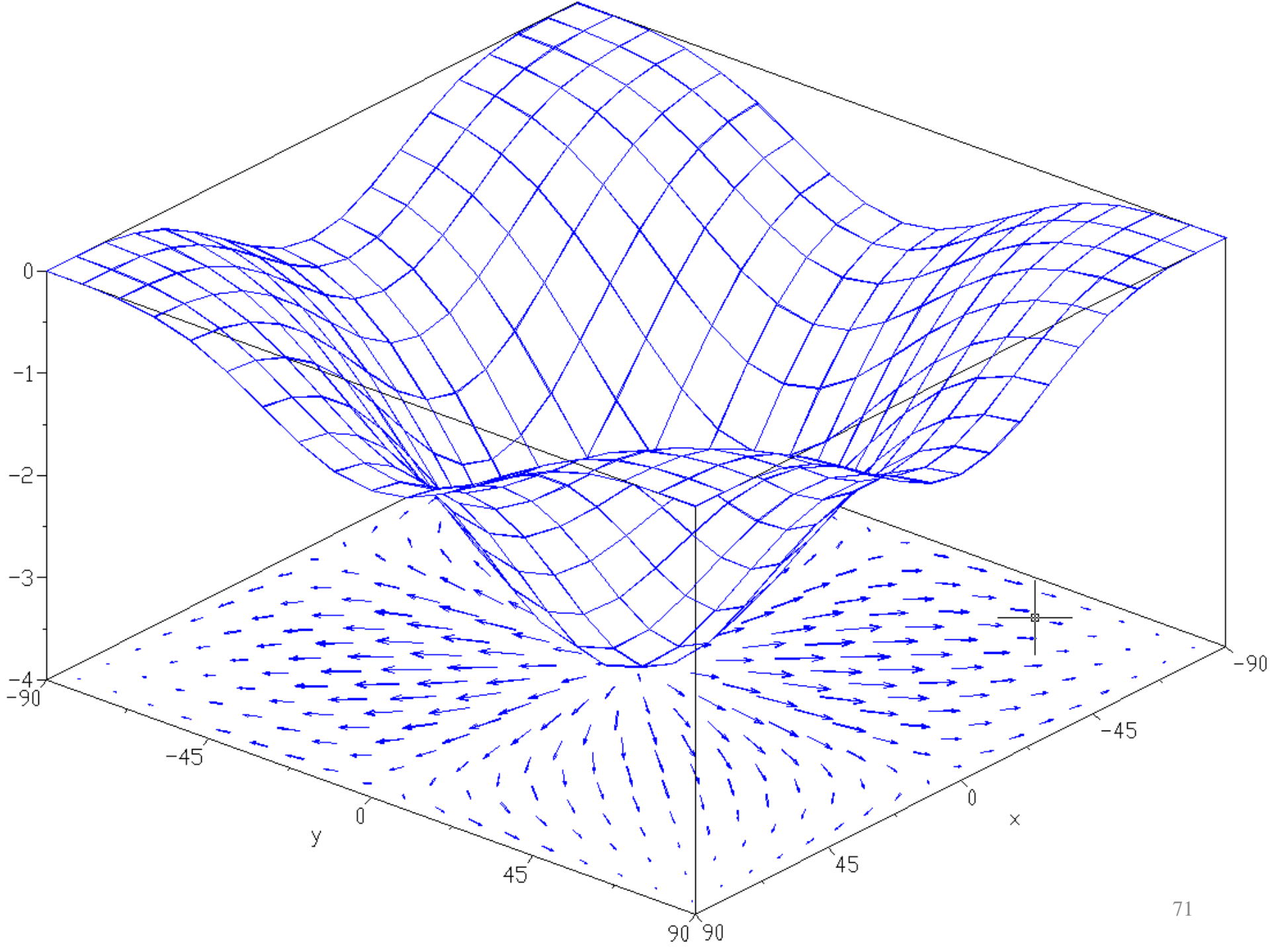
ep = 0.00000001
xk = 1.5
xk1 = 1.5
while True:
    xk1 = xk
    xk = xk1 - f(xk1) / f_derivative(xk1)
    print(xk1, xk)
    if math.fabs(xk - xk1) < ep:
        break
print(xk)
```

2.3.4 求函数极值

- ❖ 求最值（极值）问题是现实中常见的问题
 - 可以利用极值条件（导数等于0）来求解，但现实中很多问题用解析方法无法直接求得结果
 - 多采用**梯度下降法** (*Gradient Descent*) 的求解
 - ✓ 梯度：以函数的各个偏导数为分量的矢量
 - ✓ 二元函数 $z=f(x, y)$ 在点 (x_0, y_0) 处的梯度为：

$$\nabla f(x_0, y_0) = \left(\frac{\partial f(x_0, y_0)}{\partial x_0}, \frac{\partial f(x_0, y_0)}{\partial y_0} \right)$$

梯度 $\nabla f(x_0, y_0)$ 是函数在点 (x_0, y_0) 处增加最快的方向



2.3.4 求函数极值(2)

- ❖ 很自然的，可以从函数上任意一点出发求极小值：
 - 求出该点的梯度
 - 沿着梯度的反方向(函数值减少最快的方向) 前进一小段距离找下一个点
 - 重复以上两步。当连续两次迭代的函数值的差别非常小时，我们可以认为已经到达了极值点
 - 每次前进的距离（步长）很关键，太小了迭代太慢，太大了可能找不到极值点

2.3.4 求函数极值(3)

- ❖ 例5-2-4：求函数 $f(x, y)=x^2+y^2$ 的最小值
- ❖ 程序分析：
 - 输入：无
 - 输出：最小值点 (x_{min}, y_{min}) 和函数的最小值 $f(x_{min}, y_{min})$
 - 方法：使用循环迭代实现梯度下降法

2.3.4 求函数极值(4)

❖ 循环处理分析：

➤ 用到的变量及其初始值：

- ✓ 当前点坐标值 x_k 和 y_k （其初始值为 x_0 和 y_0 (随机产生)）、函数值 f_k ；
上一次点坐标 x_{k1} 和 y_{k1} 、函数值 f_{k1} 、梯度 x 分量 $grad_x$, y 分量 $grad_y$ ；
阈值 ϵ , 步长系数 t

➤ 每次循环处理（循环体）：

- ✓ 用上一次迭代的坐标值和函数值更新 x_{k1} 、 y_{k1} 和 z_{k1}
- ✓ 求出点 (x_{k1}, y_{k1}) 处的梯度($grad_x=2*x_{k1}, grad_y=2*y_{k1}$)
- ✓ 沿梯度方向下降：求出本次迭代点坐标 $(x_k, y_k)=(x_{k1}-t*grad_x, y_{k1}-t*grad_y)$
- ✓ 求出本次迭代点处的坐标值 $z_k=x_k**2+y_k**2$

➤ 循环结束条件：

- ✓ 当两次迭代点间的函数值 f_k 和 f_{k1} 之差的绝对值小于预定的阈值 ϵ 时，迭代结束

```

import math
import random

random.seed()
# random.uniform(a,b)产生一个[a,b]区间上服从均匀分布的随机浮点数
x0=random.uniform(-10,10)
y0=random.uniform(-10,10)

ep = 0.00000001
xk,yk = x0, y0
zk = xk ** 2 + yk ** 2
t = 0.001
while True:
    xk1,yk1 = xk,yk
    zk1 = zk
    grad_x,grad_y = 2*xk1,2*yk1
    xk=xk1-t*grad_x
    yk=yk1-t*grad_y
    zk = xk ** 2 + yk ** 2
    if math.fabs(zk - zk1) < ep:
        break
print(f"f({xk :.8f},{yk :.8f})={zk :.8f}")

```

2.3.4 求函数极值(5)

- ❖ 梯度下降方法在最优化和人工智能领域应用非常广泛。
- ❖ 例如最近非常火热的围棋软件AlphaGo，采用的深度网络（一种人工神经网络），其网络的训练就是采用梯度下降方法来进行的。

循环

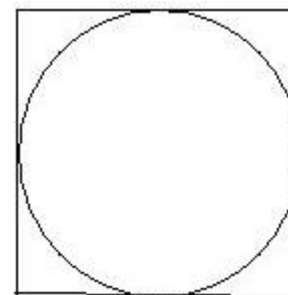
- 一. 基本循环
- 二. 应用：迭代计算
- 三. **应用：计算机模拟**
- 四. 多重循环
- 五. 应用：AI与穷举法

三、计算机统计模拟

- ❖ 现实中很多问题存在随机性，难以直接用解析方法求出结果
- ❖ 利用计算机运算速度快的特点，可以在短时间内用计算机进行大量的模拟试验，并进一步利用统计学原理汇总得到结果。
- ❖ python标准库中的random模块提供了基本的随机数功能，使用时注意：
 - 使用import random导入random模块
 - 在程序开始时调用random.seed()初始化随机数发生器

3.1 蒙特卡罗法求圆周率

- ❖ 摩纳哥的Monte Carlo是世界著名的赌城
- ❖ 冯·诺伊曼和M.乌拉姆提出蒙特卡罗方法，用电子计算机实现统计模拟或抽样，以获得问题的近似解。
- ❖ 1777年，法国数学家布丰提出用投针实验的方法求圆周率，这被认为是蒙特卡罗法的起源。
- ❖ 投针法求圆周率的原理：
 - 已知正方形面积，在正方形中作一内切圆。只要能知道圆与正方形的面积之比，就可以求出圆的面积。
 - 由圆面积公式即可求出圆周率



3.1 蒙特卡罗法求圆周率

- ❖ 分析：如何求圆与正方形的面积比？
 - 我们可以用一把枪向正方形上射击若干次。如果子弹打到正方形每个位置上的概率密度都是相同的，那么子弹打到内接圆里面的概率正好就是圆的面积与正方形面积之比。
 - 按照大数定律，当射击的次数足够多时，那么最终打到圆里面的子弹的频率应该趋近于概率。
- ❖ 因此，我们可以用计算机模拟射击过程，求出落在内接圆中的子弹数量与总射击次数的比值，就可以近似得到内接圆的面积。

3.1 蒙特卡罗法求圆周率(2)

❖ 模拟实验分析:

- 靶子区域设计：单位正方形边长1，内接单位圆半径 $r=0.5$ ，正方形中心和圆心坐标 $(cx,cy)=(0,0)$ 。正方形左上角坐标 $(-0.5,0.5)$ ，右下角坐标 $(0.5,-0.5)$
- 如何模拟一次射击：随机产生子弹落点位置 (x,y) 。根据题设， x 和 y 均服从 $[-0.5,0.5]$ 上的均匀分布。
- 如何判断子弹是否落在园内？判断点 (x,y) 到圆心 (cx,cy) 的距离是否小于等于半径即可。

3.1 蒙特卡罗法求圆周率(3)

❖ 程序分析：

- 输入：实验次数 n （在程序开头处定义的常量，不需要用户输入）
- 输出：求得的圆周率 π
- 处理：进行 n 次重复射击实验
 - ✓ 显然，可以用循环实现重复实验，每次循环做一次模拟射击实验。

3.1 蒙特卡罗法求圆周率(4)

❖ 循环分析：

- 循环中用到的变量及其初始值：
 - ✓ 落到圆内的实验次数 $\text{count}=0$ ，本次实验的子弹落点位置坐标 (x,y) ，圆心位置 $\text{cx},\text{cy}=(0,0)$ ，圆半径 $r=0.5$
- 每次循环中的处理步骤：
 - ✓ 随机产生落点位置坐标 x 和 y
 - ✓ 判断点 (x,y) 是否落在圆内。是则 count 加一。
- 循环条件：
 - ✓ 循环 n 次。显然用 for 循环最合适。

3.1 蒙特卡罗法求圆周率(5)

```
import random
import math

random.seed()
cx,cy=0,0
r=0.5
n=500000
count = 0
for i in range(n):
    x=random.uniform(-0.5,0.5)
    y=random.uniform(-0.5,0.5)
    if math.hypot(x-cx,y-cy)<=r:
        count +=1

area = count / n
pi = area / r / r
print(f"pi = {pi : .4f}")
```

3.2 产品盈利估计

- ❖ 例3-2：已知某新产品一年后上市，其盈利同时受产品销量、市场价格、利率影响。已知：
 - 项目总成本为200,000元。
 - 产品销量服从均值为30000，方差为10000的正态分布；
 - 市场价格服从均值为6元，方差为1元正态分布；
 - 利率服从均值为10%元，方差为2%的正态分布；
 - 求该项目亏损的可能性有多大？
- ❖ 模拟分析：
 - 可以使用蒙特卡洛法，每次随机生成销量、价格和利率，用下式计算出项目盈利： $\text{盈利} = \text{销量} \times \text{价格} / (1 + \text{利率}) - \text{成本}$
 - 重复实验多次，统计其中亏损的次数占总次数的比例即可

3.2 产品盈利估计(2)

❖ 程序分析：

➤ 输入：无

➤ 初始条件（常量）：

✓ 项目总成本 cost ，产品销量、市场价格和利率的随机参数（均值和方差），实验次数 n

➤ 输出：

✓ 亏损的次数占总次数的比例

➤ 处理步骤：

✓ 使用循环进行重复模拟实验

3.2 产品盈利估计(3)

❖ 循环分析：

➤ 循环中用到的变量及初始值：

- ✓ 本次实验销量sale, 本次实验价格price, 本次实验利率rate
- ✓ 本次实验利润profit, 亏损实验次数count=0

➤ 循环处理步骤：

- ✓ 随机产生本次实验销量、价格和利率
- ✓ 计算本次实验利润
- ✓ 判断本次实验是否亏损, 累计亏损实验次数

➤ 循环结束条件：

- ✓ 循环n次。显然用for循环最合适。

3.2 产品盈利估计(4)

```
import random

def get_normal(mean,dev):
    """
    产生一个非负，服从指定正态分布的随机数

    :param mean: 正态分布的均值
    :param dev: 正态分布的标准差
    :return: 产生的随机数
    """
    if mean<=0 or dev<=0 :
        raise RuntimeError("Mean and dev must be positive
! ")
    result = random.normalvariate(mean,dev)
    return max(0,result)
```

3-2.profit.py 产品盈利估计

思考：max函数的作用是返回所给参数中最大的那个。这里有什么用？

3.2 产品盈利估计(5)

```
random.seed()
cost = 200000
sale_mean, sale_dev = 30000, 10000
price_mean, price_dev = 6, 1
rate_mean, rate_dev = 0.1, 0.02
n = 500000
count = 0
for i in range(n):
    sale = get_normal(sale_mean, sale_dev)
    price = get_normal(price_mean, price_dev)
    rate = get_normal(rate_mean, rate_dev)
    profit = sale * price / (1 + rate) - cost
    if profit <= 0:
        count += 1

print(f"亏损的比例为{count/n * 100: .2f}%")
```

3.3 租车业务分析

- ❖ 例3-3：某公司打算开展A和B两地间的自行车租用业务。可以从A租车骑到B归还，或则从B租车骑到A归还。假设：
 - 每1小时内有0.3的可能有人从A租车到B；有0.5的可能有人从B租车到A。
 - 简单起见假设能够在1小时时间段内从A到B或从B到A的
 - 一开始A，B各有100辆车。
 - 如果用户租不到车（无车可借），则不等待立即离开
- ❖ 该公司希望知道，在10000小时的运营时间内，顾客租不到车（无车可借）的可能性有多大？

3.3 租车业务分析 (2)

❖ 分析：

- 我们可以以1小时为一个时间段（一次循环），来模拟租车的运营过程

❖ 在每个时间段内：

- 判断是否有人租车从A到B：产生一个 $[0,1]$ 上均匀分布的随机数，如果小于等于0.3，则有人租车；否则无人租车
- 如果有人租车从A到B，那么：
 - ✓ A地租车总用户数量加一
 - ✓ 如果A地有车，则A地车数量减一，B地车数量加一
 - ✓ 否则，则A地租不到车的用户数量加一
- 类似的，判断是否有人租车从B到A，并进行相应处理

3.3 租车业务分析 (3)

❖ 程序设计分析:

- 输入：无
- 初始条件（常量）：A地租车可能性=0.3，B地租车可能性=0.5，A地起始车辆数量=100，B地起始车辆数量=100，模拟时间段数量 $n=10000$
- 输出：租不到车的用户比例（想租但未租到的用户数/来租车的用户总数）
- 处理步骤：使用循环按前面的分析进行模拟，每次循环相当于一个小时的时间段

3.3 租车业务分析 (4)

❖ 循环处理分析:

- 循环中用到的变量: A地有人要租车标志want_a, B地有人要租车want_b, A地车数量num_a=100, B地车数量num_b=100, A地想租车总人数customer_a=0, 未租到车人数no_bike_count_a=0; B地想租车总人数customer_b=0, 未租到车人数no_bike_count_b=0
- 循环处理步骤:
 - ✓ 产生随机数, 判断A地是否有人租车。如有人租车, 则按前述分析进行相应处理;
 - ✓ 产生随机数, 判断B地是否有人租车。如有人租车, 则按前述分析进行相应处理;
- 循环条件: 循环n次 (重复n个时间段)。显然, 这里用for循环最合适。

```
import random

random.seed()
n=10000
prob_a,prob_b = 0.3,0.5
num_a,num_b = 100,100
customer_a,customer_b = 0,0
no_bike_count_a,no_bike_count_b=0,0
for i in range(n):
    want_a = random.uniform(0,1) <= 0.3
    want_b = random.uniform(0,1) <= 0.5
    if want_a:
        customer_a += 1
        if num_a > 0:
            num_a -= 1
            num_b += 1
        else:
            no_bike_count_a += 1
```

3.3 租车业务分析 (6)

```
if want_b:
    customer_b += 1
    if num_b > 0:
        num_b -= 1
        num_a += 1
    else:
        no_bike_count_b += 1
```

```
ratio_a = no_bike_count_a / customer_a * 100
ratio_b = no_bike_count_b / customer_b * 100
print(f"A地想租车人数{customer_a},未租到人数\n{no_bike_count_a}, 未租到比例{ratio_a:0.2f}%")
print(f"B地想租车人数{customer_b},未租到人数\n{no_bike_count_b}, 未租到比例{ratio_b:0.2f}%")
```

3-3.rent.py (续)

循环

- 一. 基本循环
- 二. 应用：迭代计算
- 三. 应用：计算机模拟
- 四. **多重循环**
- 五. 应用：AI与穷举法

四、多重循环

- ❖ 我们在现实中，经常会遇到重复中嵌套重复的情况：
 - 一箱10盒，每盒8块月饼
 - 每周锻炼7天，每天锻炼200次
- ❖ 在程序中，我们同样可以在一个循环的循环体内包含另外一个循环，组成所谓的多重循环。

4.1 两重循环

- ❖ 在一个循环的循环体中包含（**嵌套**）另一个循环，就是两重循环。
- ❖ 例4-1：输出一个n行m列的由*组成的矩形。
 - 例如， $n=3, m=5$ 时，输出为：

4.1 两重循环（1）

❖ 分析：

- 输入：无
- 初始条件：每行*号个数 m ；行数 n
- 输出： $n*m$ 个*号组成的矩形
- 处理步骤：我们可以把问题分成两个重复问题：
 - ✓ 问题1：如何连续输出 m 个*字符？
 - ✓ 问题2：如何输出 n 行上述 m 个字符？
 - ✓ 显然这两个问题都是重复性工作，可以用循环来实现
 - ✓ 因此，我们用两重循环实现

4.1 两重循环 (2)

❖ 外层循环 (第一层循环) 分析:

- 循环中用到的变量: 无
- 循环处理步骤:
 - ✓ 输出连续的 m 个*号 (内层循环)
 - ✓ 另起一行 (输出回车)
- 循环条件: 循环 n 次 (n 行)

4.1 两重循环 (3)

❖ 内层循环 (第二层循环) 分析:

- 循环中用到的变量: 无
- 循环处理步骤:
 - ✓ 输出1个*号
- 循环条件: 循环 m 次 (m 列, m 个*号)

4.1 两重循环 (4)

```
n=5
m=10
for i in range(n):
    for j in range(m):
        print("*",end="")
    print()
```

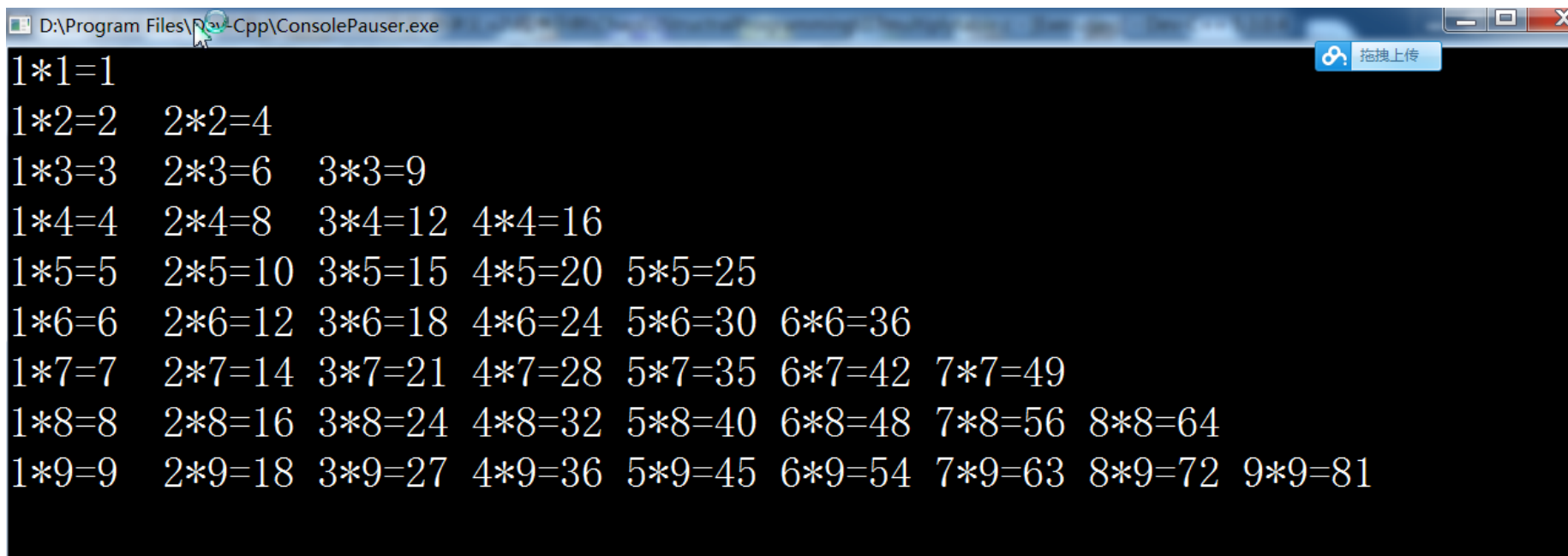
4-1.rectangle.py

`print("*",end="")`中, `end=""`的作用是在输出一个*号后, 不换行。
下一个`print()`会继续在同一行输出。如果把`end=""`去掉, 运行结果会怎样?

`print()`的作用是另起一行。把这条语句去掉, 结果会怎样?
这条语句一共运行了几次?

4.2 乘法表

❖ 例4-2：编写程序，输出下图所示的乘法表（见下一页提示）



The screenshot shows a Windows console window titled "D:\Program Files\New-Cpp\ConsolePauser.exe". The console displays a 9x9 multiplication table. The first row contains "1*1=1". The second row contains "1*2=2" and "2*2=4". The third row contains "1*3=3", "2*3=6", and "3*3=9". The fourth row contains "1*4=4", "2*4=8", "3*4=12", and "4*4=16". The fifth row contains "1*5=5", "2*5=10", "3*5=15", "4*5=20", and "5*5=25". The sixth row contains "1*6=6", "2*6=12", "3*6=18", "4*6=24", "5*6=30", and "6*6=36". The seventh row contains "1*7=7", "2*7=14", "3*7=21", "4*7=28", "5*7=35", "6*7=42", and "7*7=49". The eighth row contains "1*8=8", "2*8=16", "3*8=24", "4*8=32", "5*8=40", "6*8=48", "7*8=56", and "8*8=64". The ninth row contains "1*9=9", "2*9=18", "3*9=27", "4*9=36", "5*9=45", "6*9=54", "7*9=63", "8*9=72", and "9*9=81".

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

4.2 乘法表 (2)

❖ 分析：

- 输入：无
- 初始条件：9行9列乘法表
- 输出：乘法表
- 处理方法：
 - ✓ 可以利用双重循环实现。外层循环控制有多少行，内层循环控制每行有哪些内容（有多少列）。
 - ✓ 找规律：每个乘法式的第一个乘数和行数、列数有何关系？第二个乘数呢？
 - ✓ 第 i 行应该有几列（几个乘法式？）

4.2 乘法表 (3)

❖ 外层循环分析:

- 循环中用到的变量: 行数 i (循环变量)
- 循环处理步骤:
 - ✓ 输出一行内容 (使用内层循环)
 - ✓ 另起一行 (输出回车)
- 循环条件: 循环9次

4.2 乘法表 (4)

❖ 内层循环分析:

- 循环中用到的变量: 行数 i (外层循环变量) 和列数 j (内层循环变量)
- 循环处理步骤:
 - ✓ 输出一个乘法式
- 循环条件: 循环 i 次

4.2 乘法表 (5)

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

4-2.乘法表.py

注意外层循环中我们使用了range(1,10)。i会从几循环到几？一共循环几次？

注意内层循环中我们使用了range(1,i+1)。j会从几循环到几？一共循环几次？

4.3 多重循环跳转

- ❖ 当在多重循环内部遇上了break/continue，退出/继续的到底是哪一重循环呢？
- ❖ 例4-3：求200以内的质数（素数）。
- ❖ 啥是质数：
 - 只能被1和它自己整除的正整数。（但1不是质数）
- ❖ 根据质数定义，我们可以用循环来判断一个整数n是否是质数。
 - 从2到n-1逐个尝试是否能整除n
 - 进一步可以简化为从2到几？
- ❖ 因此本例可以用两重循环求解：
 - 外层循环从2循环到200，即逐个尝试2到200之间的所有整数
 - 内层循环负责判断当前整数是否是质数

4.3 多重循环跳转 (2)

❖ 外层循环分析：

- 循环中用到的变量：整数 n （循环变量）
- 循环处理步骤：
 - ✓ 使用循环判断 n 是否为质数（内层循环）
 - ✓ 如果 n 为质数，则输出 n
- 循环条件： n 从2循环到200

❖ 在每次内层循环结束后，我们需要知道判断的结果（ n 是否为质数）。最简单的办法是用一个变量`is_prime`来保存判断结果。

4.3 多重循环跳转 (3)

❖ 内层循环分析:

- 循环中用到的变量及其初值: 整数 n (外层循环变量), 尝试整除因子 $factor$ (内层循环变量), $is_prime=True$
- 循环处理步骤:
 - ✓ 使用 $factor$ 尝试整除 n
 - ✓ 如果能整除, is_prime 设为 $False$; $break$ 退出循环 (知道不是质数, 就没必要再继续试了)
- 循环条件: $factor$ 从2循环到 $n-1$

4.3 多重循环跳转 (4)

```
for n in range(2,201):  
    is_prime = True  
    for factor in range(2,n):  
        if n % factor == 0:  
            is_prime = False  
            break  
    if is_prime:  
        print(n)
```

4-3.找质数.py

注意外层循环中我们使用了range(2,201)。n会从几循环到几？一共循环几次？

注意内层循环中我们使用了range(2,n)。factor会从几循环到几？一共循环几次？

is_prime = True放在外层循环开始前（程序第一行）行不行？为什么？

4.4 矿工逃生

- ❖ 例4-4：一大群矿工身陷在有三个门的矿井之中。
- ❖ 已知：
 - 如果某人选择进入第1个门，行进两小时后，他将安全逃生；
 - 如果选择进入第2个门，行进三小时后，他将返回原先的矿井；
 - 如果选择进入第3个门，行进五小时后，他又将返回原先的矿井原地
- ❖ 如果每个矿工每次都等可能的选择任意一个门，那么，平均每个矿工安全逃生所需时间是多少？
- ❖ 分析：
 - 我们可以用程序模拟每个矿工逃生的过程，记录下他逃生所需的时间，求所有矿工逃生时间的平均值即可

4.4 矿工逃生 (2)

- ❖ 如何模拟某个矿工逃生的过程？
- ❖ 显然，这是个不断重复的过程（循环）：
 - 选门
 - 如果选对了门，循环结束
 - 如果选错了门，其逃生时间增加，返回原矿井继续循环
- ❖ 也就是说，每次实验是一个循环（内层循环）。
- ❖ 因为我们还需要用循环来做多次实验求平均值（外层循环），因此总共需要两重循环。

3.2 矿工逃生 (3)

❖ 程序分析:

- 输入：无
- 初始常量 (条件): 实验次数 n
- 输出：平均逃生时间average
- 处理步骤:
 - ✓ 双重循环做 n 次实验，求出总的逃生时间
 - ✓ 平均逃生时间=总逃生时间/实验次数

4.4 矿工逃生 (4)

❖ 外层循环设计：

- 循环中用到的变量及其初值：总的矿工逃生时间 `total_time=0`，本次实验矿工逃生过程 `escape_time`
- 循环处理步骤（循环体）：
 - ✓ 模拟一次矿工逃生过程，得到其逃生时间 `escape_time`
 - ✓ 累计总逃生时间： `total_time+=escape_time`
- 循环条件：循环 `n` 次（实验 `n` 次）。显然用 `for` 循环最合适。

4.4 矿工逃生 (5)

❖ 内层循环设计：

- 循环中用到的变量及其初值：本次实验矿工逃生过程escape_time=0
- 循环处理步骤（循环体）：
 - ✓ 按照前面分析中的步骤，模拟一次矿工逃生过程
- 循环条件：一直循环直到矿工选对门为止。
- 这里最合适（简单）的循环方式是使用带break退出的while True循环。

```
import random

total_time = 0
n=500000

for i in range(n):
    escape_time = 0
    while True:
        door = random.randint(1,3)
        if door == 1:
            escape_time += 2
            break
        elif door == 2:
            escape_time += 3
        elif door == 3:
            escape_time += 5
    total_time += escape_time

average = total_time / n
print(f"平均逃生时间为{average: .2f}小时")
```

循环

- 一. 基本循环
- 二. 应用：迭代计算
- 三. 应用：计算机模拟
- 四. 多重循环
- 五. **应用：AI与穷举法**

五、穷举法

❖ 思考：

- 假设你和你的GF/BF在一个从没去过的商场里约会&逛街。
- 你想要上洗手间。你知道每个商场里肯定有洗手间，但是不知道在哪里。
- 你怎么办？

❖ 最笨但是一定有效的办法：

- 把商场里的每条路、每个门、角落都走一遍，直到找到洗手间为止。

❖ 这种把所有可能的解决方案都找出来——尝试的方法，就是**穷举法**。

例3 汽车厂生产计划

汽车厂生产三种类型的汽车，已知各类型每辆车对钢材、劳动时间的需求，利润及工厂每月的现有量。

	小型	中型	大型	现有量
钢材（吨）	1.5	3	5	600
劳动时间（小时）	280	250	400	60000
利润（万元）	2	3	4	

- 制订月生产计划，使工厂的利润最大。
- 如果生产某一类型汽车，则至少要生产80辆，那么最优的生产计划应作何改变？

汽车厂生产计划



模型建立

设每月生产小、中、大型汽车的数量分别为 x_1, x_2, x_3

	小型	中型	大型	现有量
钢材	1.5	3	5	600
时间	280	250	400	60000
利润	2	3	4	

$$\text{Max } z = 2x_1 + 3x_2 + 4x_3$$

$$\text{s.t. } 1.5x_1 + 3x_2 + 5x_3 \leq 600$$

$$280x_1 + 250x_2 + 400x_3 \leq 60000$$

$$x_1, x_2, x_3 \geq 0$$

整数规划模型

五、穷举法

- ❖ 例4-3（求200以内质数）就是一个最简单的穷举法。
 - 把所有可能整除 n 的整数都拿出来试一试，看看是不是真能整除 n ？
- ❖ 穷举法也是计算机科学一个重要的分支——**人工智能**的基本方法。
 - 人工智能：用计算机模拟人的智能来**解决复杂问题**的科学。
- ❖ 如何用程序实现穷举法？
 - 穷举法本质是逐一检查所有的可能，当然最直接的做法就是用循环来解决。

5.1 乘法原理

- ❖ 现实中的问题中往往包含多种因素，穷举就需要把各因素的所有排列组合情况都找出来。
 - 因此，穷举法的基础是排列与组合。
- ❖ 用程序实现穷举法的基础就是如何找出所有的排列组合。
- ❖ 例5-1：已知某班有10名男生，20名女生。现该班需要选取一对男女生搭档参加学院交际舞大赛。请编程求出所有可能的搭档方案。
- ❖ 分析：这是一个典型的排列组合问题，用乘法原理可知一共有 $10 \times 20 = 200$ 种不同的搭档方案。问题是如何列举出这200种方案。

5.1 乘法原理 (2)

❖ 可以这样做：

- 首先固定选取第1号男生，依次与第1、2、3...20号女生搭配，这是20种方案。
- 然后固定选取第2号男生，依次与第1、2、3...20号女生搭配，这又是20种方案
-
- 最后固定选取第10号男生，依次与第1、2、3...20号女生搭配，这是最终的20种方案。

❖ 仔细观察上述处理步骤，可以发现存在两层重复处理环节：

- 外层重复：依次选取1、2.....10号男生，与女生搭配
- 内层重复：固定选取第b号男生，依次与第1、2、3...20号女生搭配

❖ 自然的，我们可以用两重for循环来实现上述处理，见5-1.找搭档.c

5.1 乘法原理 (3)

```
count=0
for b in range(1,11):
    for g in range(1,21):
        count+=1
        print(f"第{count}号搭配方案: {b}号男生和{g}号女生")

print(f"总共有{count}种搭配方案")
```

5-1.找搭档.py

提示一：多重循环就相当于排列组合中的乘法原理

提示二：在使用程序来处理现实中的多个同类实体时，通常需要先对实体进行**编号**

提示三：range()的起止设置

5.2 排列与组合

- ❖ 两个因素的排列组合需要用两重循环来实现，那么更多的因素呢？
- ❖ 例5-2：要求甲、乙、丙三名同学分别加入社团。每人需要从足球、外语、天文、心理这4个社团中选一个参加。每人只能参加一个社团，每个社团允许参加的人数不限。找出所有可能的加入方案。
 - 如果有 n 个社团呢？
- ❖ 分析：
 - 可以用 $1\dots n$ 来代表每一个社团
 - 可以用变量 i, j, k 来分别保存甲、乙、丙参加的社团
 - 此问题实际是一个从 $1\dots n$ 中取三个数的**有重复排列问题**。（3因素排列）
 - 显然，每个人的选择可以用一个从1到 n 的循环来表示。
 - 将三个循环嵌套在一起，形成三重循环，就可以求出全部排列。

5.2 排列与组合 (2)

```
count=0
n=4
for i in range(1,n+1):
    for j in range(1,n+1):
        for k in range(1,n+1):
            count += 1
            print(f"第{count}种方案: 甲加入{i}号社团, 乙加入{j}号社团, 丙加入{k}号社团")

print(f"一共{count}种方案")
```

5-2.社团.py

提示一：在使用程序来处理现实中的多个同类实体时，通常需要先对实体进行**编号**

提示二：range()的起止设置

5.2 排列与组合 (3)

- ❖ 例5-3：甲、乙、丙三名同学当课代表。每人需要从高数、C语言、线代、英语这4门课中选一门课程来担任课代表。每人只能担任一门课代表。找出所有可能的课代表选择方案。
 - 如果有 n 门课呢？
- ❖ 分析：
 - 此问题实际是一个从 $1..n$ 中取三个数的**无重复排列**问题。
 - 还可以三重循环，只需要用if排除所有有重复数字的选择就行了！
- ❖ 思考：如果本次循环发现重复，应该使用break退出循环，还是用continue跳过本次循环？

5.2 排列与组合 (4)

```
count=0
n=4
for i in range(1,n+1):
    for j in range(1,n+1):
        if i==j:
            continue
        for k in range(1,n+1):
            if i==k or j==k:
                continue
            count += 1
            print(f"第{count}种方案: 甲担任{i}号课代表, 乙担任{j}号课代表, 丙担任{k}号课代表")

print(f"一共{count}种方案")
```

提示一: range()的起止设置

提示一: if和continue的使用

5-3.课代表.py

5.2 排列与组合 (5)

- ❖ 如何将程序改为求 $1..n$ 中取三个不同数的所有组合？(无重复组合)
 - $(1,2,3)(1,3,2) \dots (3,2,1)$ 都是同一种组合，我们只需要 $(1,2,3)$ 即可。
 - 同理 $(1,2,4)(1,4,2) \dots (4,2,1)$ 只需要 $(1,2,4)$
 - $(2,3,4) \dots (4,3,2)$ 只需要 $(2,3,4)$
 - 规律：后面的数总比前面的数大
 - 如何实现？
- ❖ 思考：如何用程序求 $1..n$ 中任取三个数的所有组合（允许重复）？

5.2 排列与组合 (6)

```
count=0
n=4
for i in range(1,n+1):
    for j in range(i+1,n+1):
        for k in range(j+1,n+1):
            count += 1
            print(f"第{count}种组合方案: {i}, {j}, {k}")

print(f"一共{count}种方案")
```

提示一: range()的起止设置

5-4.无重复组合.py

5.3 穷举法应用 (1)

汽车厂生产计划



模型建立

设每月生产小、中、大型汽车的数量分别为 x_1, x_2, x_3

	小型	中型	大型	现有量
钢材	1.5	3	5	600
时间	280	250	400	60000
利润	2	3	4	

$$\text{Max } z = 2x_1 + 3x_2 + 4x_3$$

$$\text{s.t. } 1.5x_1 + 3x_2 + 5x_3 \leq 600$$

$$280x_1 + 250x_2 + 400x_3 \leq 60000$$

$$x_1, x_2, x_3 \geq 0$$

整数规划模型

只需要把 x_1, x_2 和 x_3 的所有可能排列方案都试一遍即可!

```

max_profit = 0
max_x1,max_x2,max_x3 = 0,0,0

for x1 in range(0,401):
    for x2 in range(0,201):
        if 1.5 * x1 + 3 * x2 > 600:
            break
        if 280 * x1 + 250 * x2 > 60000:
            break
        for x3 in range(0,121):
            if 1.5*x1+3*x2+5*x3>600:
                break
            if 280*x1+250*x2+400*x3>60000:
                break
            profit = 2*x1+3*x2+4*x3
            if profit > max_profit:
                max_profit = profit
                max_x1,max_x2,max_x3 = x1,x2,x3

```

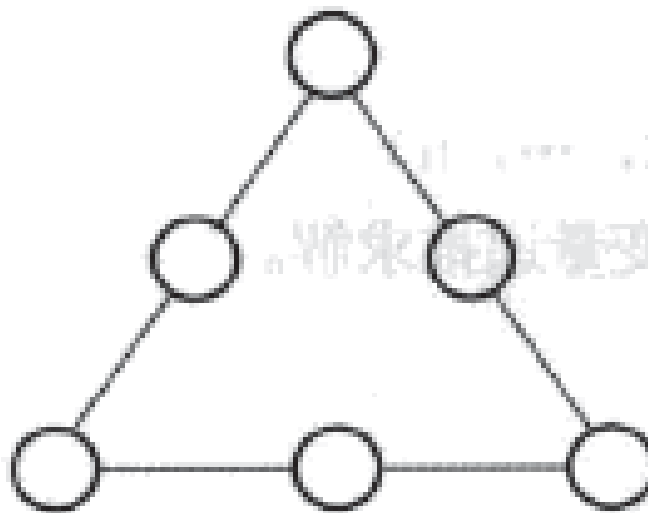
提示一：range()的起止设置。为什么结束不用600？

5-5-1.整数规划.py

思考：为什么这里的if用break？可不可以用continue？

5.3 穷举法应用 (3)

- ❖ 例5-5-2: 填写6数字三角形。即将1-6的6个数字分别填入下面的圆圈中, 使三边的和都相等。



5.3 穷举法应用 (4)

- ❖ 分析：这显然可以用穷举法来求解。
- ❖ 第一步，找出所有的填写方案，实质是一个求1..6的全排列问题。
- ❖ 可以用五重循环来解决该问题。（为啥不是六重循环？）
- ❖ 对六个圆圈分别编号为b1至b6如下：

b3
b2 b4
b1 b6 b5

5.3 穷举法应用 (5)

❖ b_1 、 b_2 、.....、 b_6 之间的关系如下：

➤ b_1 到 b_6 应各不相同

➤ $b_6 = 21 - b_1 - b_2 - b_3$

$-b_4 - b_5$

b_3

b_2

b_4

b_1

b_6

b_5

❖ 正确的解应该满足的条件：

➤ $b_1 + b_2 + b_3 = b_3 + b_4 + b_5 =$

$b_5 + b_6 + b_1$


```

count = 0
for b1 in range(1,7):
    for b2 in range(1,7):
        if b1 == b2:
            continue
        for b3 in range(1,7):
            if b3 == b1 or b3 == b2:
                continue
            for b4 in range(1,7):
                if b4 == b1 or b4 == b2 or b4 == b3:
                    continue
                for b5 in range(1,7):
                    if b5==b4 or b5==b3 or b5==b2 or b5==b1:
                        continue
                    b6 = 21-b1-b2-b3-b4-b5
                    if b1+b2+b3==b3+b4+b5 and b3+b4+b5==b5+b6+b1:
                        count+=1
                        print(f"      {b3}")
                        print(f"    {b2}      {b4}")
                        print(f"{b1}      {b6}      {b5}")

print(f"共有{count}种填法")

```

思考：为什么这里必须用continue而不是break？

5-5-2.三角填数.py

5.3 穷举法应用 (7)

❖ 用循环实现穷举法的基本步骤:

1. 找出所有可能的解

- ① 判断是组合问题还是排列问题
- ② 决定参与组合/排列的因素数量, 构造对应的多重循环

2. 按照问题的条件筛选正确的解

- ① 在最内层循环中对找出的可能解进行进一步判断。
- ② 如果是正确解, 则输出或进一步处理。
 - I. 如果只需要找到一个正确解, 可以结束寻找
 - II. 如果需要找到所有正确解, 则继续寻找
- ③ 如果不是正确解, 继续寻找

5.3 穷举法应用 (8)

- ❖ 例5-5-3：五位选手参赛前，预测比赛结果如下：
- A选手说：B第二，我第三；
 - B选手说：我第二，E第四；
 - C选手说：我第一，D第二；
 - D选手说：C最后，我第三；
 - E选手说：我第四，A第一。
 - 结果每人只猜对了一个。请求出比赛的实际名次情况。

5.3 穷举法应用 (9)

❖ 分析：

- 显然，求比赛的所有可能名次是一个1..5的全排列问题。需要几重循环？
- 如何将五位选手的预测转换为逻辑表达式？
- 如何表达每人只猜对了一个？

❖ 见5-5-3.比赛名次.py

```
for a in range(1, 6):  
    for b in range(1, 6):  
        if b == a :  
            continue  
        for c in range(1, 6):  
            if c == b or c == a:  
                continue  
            for d in range(1, 6):  
                if d==c or d==b or d==a:  
                    continue  
                e=15-a-b-c-d  
                if b==2 and a==3:  
                    continue  
                if b!=2 and a!=3:  
                    continue  
                if b==2 and e==4:  
                    continue  
                if b!=2 and e!=4:  
                    continue
```

思考：为什么这里必须用continue而不是break？

5-5-3.比赛名词.py(1)
141

```
if c==1 and d==2:
    continue
if c!=1 and d!=2:
    continue
if c==5 and d==3:
    continue
if c!=5 and d!=3:
    continue
if e==4 and a==1:
    continue
if e!=4 and a!=1:
    continue
print(f"a={a}, b={b}, c={c}, d={d}, e={e}")
```

思考：为什么这里必须用continue而不是break？

5-5-2.三角填数.py(2)

六、结构化编码

- ❖ 在程序设计的早期，程序设计者可以使用goto语句实现程序执行路径的任意改变。
 - 见程序6-1.unstructured.c，尝试预测其输出？
- ❖ 用这种可以随意改变执行路径的思想设计出来的程序很难被人理解和维护，也容易导致程序缺陷的产生。
- ❖ 因此提出了结构化编码的概念，即：
- ❖ **结构化编码**：只使用**顺序**、**分支**和**循环**三种结构来编写程序的设计方法。

六、结构化编码

- ❖ 使用结构化编码来设计的程序，只能有三种执行路径的变化：
 - 不执行某段代码（选择结构）
 - 重复执行某段代码(循环结构)
 - 从某段代码内部中跳出（break、continue、return和raise）

编程思路总结

- ❖ 设计程序时最基本的思路就是.....
- ❖ 找规律：找出那些需要重复执行的步骤！ 例如：
 - 输入：
 - ✓ 要求用户（连续）输入多组相同格式的数据
 - ✓ 等待用户进行某种操作（如点击鼠标），然后执行相应的处理
 - 输出：
 - ✓ 按照固定的格式（如表格）输出多组计算结果
 - 运算：
 - ✓ 累加
 - ✓ 逐一比较判断
 - ✓
- ❖ 重复的步骤就可以写成循环！

编程思路总结

❖ 结合上一章中讲过的自顶向下和模块化思想，我们写程序的基本思路就是这样的：

1. 如果问题比较复杂，先按照自顶向下思想，看看能不能把问题分解为较小的子问题或者解决步骤？然后逐个解决这些子问题或者解决步骤。
2. 如果子问题依然比较复杂，继续分解。如此层层分解，直到问题不需要再分解可以直接解决为止。
3. 不管在解决问题的哪一层级，看看是否需要连续重复的步骤？有就可以用循环来处理。
4. 看看各部分中有没有相近或者类似的问题处理步骤？如果有，看看能不能做成一个统一的函数或者模块，以减少代码重复。（需要对编程比较熟练）

❖ **找规律**用循环解决问题是整个程序设计中最基本的技能。必须反复练习熟练掌握！

本章重点

- ❖ while循环的语法及其应用
- ❖ for循环的语法及其应用
- ❖ range()函数的用法
- ❖ break和continue的用法
- ❖ 多重循环的语法及其应用
- ❖ 什么是穷举法；使用多重循环解决简单排列组合问题
- ❖ 什么是蒙特卡洛方法。能够编程实现使用蒙特卡洛模拟解决简单随机问题。

本章练习

- ❖ 见练习 《第二章 结构化程序设计基础》
- ❖ 至少应完成练习1-7, 9-12共11道练习。
- ❖ 有余力的同学可以完成全部练习