

```
672
673 #习题册58页80题
674 li=[1,2,3,4,5,6,7,8,9,10]
675 for i in li:
676     if i%2==0:
677         continue
678     elif i==9:
679         break
680     print(i)
```

控制台

```
1
3
5
7
程序运行结束
```

```
681
682 #习题册58页81题
683 def f(a,b):
684     return a+b
685
686 print(f(1,2))
687
```

控制台

```
3
程序运行结束
```

```
690 def find_factors(n):
691     因数列表=[]
692     for i in range(1,n+1):
693         if n%i==0:
694             因数列表.append(i)
695     return 因数列表
696 print(find_factors(12))
697
```

控制台

```
[1, 2, 3, 4, 6, 12]
程序运行结束
```

```
699 #习题册59页83题
700 def fibonacci(n):
701     if n==1:
702         return 0
703     elif n==2:
704         return 1
705     else:#斐波那契数列的第n项，是前两项之和
706         return fibonacci(n-2)+fibonacci(n-1)
707 for 项数 in range(1,7):
708     print(f'斐波那契数列第{项数}项是%d'%fibonacci(项数))
709 #书上说fibonacci(6)应返回8，是错的
```

控制台

```
斐波那契数列第1项是0
斐波那契数列第2项是1
斐波那契数列第3项是1
斐波那契数列第4项是2
斐波那契数列第5项是3
斐波那契数列第6项是5
程序运行结束
```

```

712 #习题册59页83题, 解法2
713 def fibonacci(n):
714     a=0
715     b=1
716     if n==1:
717         return a
718     elif n==2:
719         return b
720     else:
721         for i in range(n-2):
722             c=a+b
723             a=b
724             b=c
725         return c
726 for 项数 in range(1,7):
727     print(f'斐波那契数列第{项数}项是%d'%fibonacci(项数))
728

```

控制台

```

斐波那契数列第1项是0
斐波那契数列第2项是1
斐波那契数列第3项是1
斐波那契数列第4项是2
斐波那契数列第5项是3
斐波那契数列第6项是5
程序运行结束

```

```

731 #习题册59页84题, 铺垫
732 #自定义一个函数, 返回一个正整数的真因子
733 def 真因子(n):
734     L=[]
735     for i in range(1,n):
736         if n%i==0:
737             L.append(i)
738     return L
739 print('10的真因子为: ',真因子([10]))
740

```

控制台

```

10的真因子为: [1, 2, 5]
程序运行结束

```

```
743
744 #习题册59页84题，铺垫，可以用sum()函数和真因子()函数
745 # 计算一个正整数的真因子之和
746 print('10的真因子之和为：',sum(真因子(10)))
747
```

控制台

```
10的真因子之和为： 8
程序运行结束
```

```
750 #习题册59页84题，
751 def 真因子(n):
752     L=[]
753     for i in range(1,n):
754         if n%i==0:
755             L.append(i)
756     return L
757
758 def is_perfect_number(num):
759     if sum(真因子(num))==num:
760         return True
761     else:
762         return False
763
764 print(is_perfect_number(6))
```

控制台

```
True
程序运行结束
```

```

767 #习题册59页84题，解法2，仅使用一个自定义函数
768 def is_perfect_number(num):
769     L=[]
770     for i in range(1,num):
771         if num%i==0:
772             L.append(i)
773     if sum(L)==num:
774         return True
775     else:
776         return False
777
778 print(is_perfect_number(6))

```

控制台

True  
程序运行结束

```

782 #60页85题，铺垫，自定义一个函数，能够返回
783 #一个正整数的每一位的数字的列表
784 #例如一个正整数123，它的每一位数字是[1,2,3]
785 def 每位数字(n):
786     L=[]
787     while n>0:
788         L.insert(0,n%10)#将n的个位数插到列表开头
789         n//=10#将n缩小为原来的十分之一，向下取整
790         #相当于n=n//10的写法
791     return L
792 print(每位数字(123))#验证一下，函数是否能返回123的每位数字

```

控制台

[1, 2, 3]  
程序运行结束

```
781 #60页85题, 铺垫, 一个正整数的位数, 是每位数字的列表的长度
782 def 每位数字(n):
783     L=[]
784     while n>0:
785         L.insert(0,n%10)
786         n//=10
787     return L
788 位数=len(每位数字(123))
789 print(123,'的位数是',位数)
```

控制台

```
123 的位数是 3
程序运行结束
```

```
781 #60页85题, 设一个正整数num是n位数,
782 #用自定义函数, 返回num的每位数字的n次方之和
783 def 每位数字(n):
784     L=[]
785     while n>0:
786         L.insert(0,n%10)
787         n//=10
788     return L
789 def 各位数字的n次方之和(num):
790     L=每位数字(num)
791     n=len(L)
792     和=0
793     for i in L:
794         和+=i**n
795     return 和
796 print(各位数字的n次方之和(123))
```

控制台

```
36
程序运行结束
```

```
802 #60页85题，完整解法，一，用3个自定义函数
803 def 每位数字(n):
804     L=[]
805     while n>0:
806         L.insert(0,n%10)
807         n//=10
808     return L
809 def 各位数字的n次方之和(num):
810     L=每位数字(num)
811     n=len(L)
812     和=0
813     for i in L:
814         和+=i**n
815     return 和
816 def is_armstrong_number(num):
817     if 各位数字的n次方之和(num)==num:
818         return True
819     else:
820         return False
821 print(is_armstrong_number(153))
```

控制台

```
True
程序运行结束
```

```

825 #60页85题，完整解法二，用1个自定义函数
826 def is_armstrong_number(num):
827     t=num#预先复制一份num的值，用t标记
828     #得到num的每位数字，放进列表，并记录num最终是几位数
829     L=[]
830     位数=0
831     while num>0:
832         L.append(num%10)
833         位数+=1
834         num//=10
835     和=0#计算每位数字的位数次幂之和
836     for i in L:
837         和+=i**位数
838     return 和==t#若和==t返回True，否则返回False
839     #在while循环中,num本身发生了变化，t表示最初的num
840
841 print(is_armstrong_number(153))

```

控制台

True  
程序运行结束

60页86题，知识铺垫，牛顿迭代法计算2的平方根

要不要我给你一个最简单的算例（比如求 $\sqrt{2}$ ），

好，我们就用求 $\sqrt{2}$ 来演示牛顿迭代法，超直观。

#### 目标

求方程

$$f(x) = x^2 - 2 = 0$$

的正根，也就是 $\sqrt{2} \approx 1.4142\dots$

#### 1. 写出迭代公式

$$f'(x) = 2x$$

牛顿公式：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



代入得:

$$x_{n+1} = \frac{1}{2} \left( x_n + \frac{2}{x_n} \right)$$

## 2. 随便给个初值

取  $x_0 = 1$

## 3. 开始迭代

$$1. x_1 = \frac{1}{2} (1 + 2/1) = 1.5$$

$$2. x_2 = \frac{1}{2} (1.5 + 2/1.5) \approx 1.4167$$

$$3. x_3 = \frac{1}{2} (1.4167 + 2/1.4167) \approx 1.4142$$

```
844
845 #60页86题, 铺垫: 套用牛顿迭代法计算正整数n的平方根的公式
846 #做一个自定义函数sqrt(n), 用来返回n的平方根
847 def sqrt(n):#求n的平方根
848     x1=1#设n的平方根的初始值为1
849     x2=(1/2)*(x1+n/x1)#将x1代入牛顿迭代法公式, 计算的到新的平方根x2
850     while x1!=x2:#重复判断, 前一次的平方根和后一次的平方根是否相等
851         x1=x2#如果不相等, 前一次的平方根x1设为后一次的平方根x2
852         x2=(1/2)*(x1+n/x1)#后一次的平方根x2重新计算
853     return x2#返回x2 (与返回x1等效)
854
855 print(sqrt(2))#输出根号2
856
```

控制台

```
1.414213562373095
程序运行结束
```

```

858
859 #60页86题, 完整解法
860 def sqrt(n):
861     x1=1
862     x2=(1/2)*(x1+n/x1)
863     while x1!=x2:
864         x1=x2
865         x2=(1/2)*(x1+n/x1)
866     s='%.3f'%x2#用x2这个小数, 生成一个3位小数格式的字符串
867     return float(s)#将字符串类型转为浮点数类型, 并作为函数值, 返回
868
869 print(sqrt(2))
870

```

控制台

1.414  
程序运行结束

```

871
872 #60页86题, 精简代码解法
873 def sqrt(n):
874     x=1
875     while x!=(1/2)*(x+n/x):#重复判断正整数n的当前平方根—x是否等于, 迭代计算之后的平方根—(1/2)*(x+n/x)
876         x=(1/2)*(x+n/x)
877     s='%.3f'%x
878     return float(s)
879
880 print(sqrt(2))

```

控制台

1.414  
程序运行结束

60页87题, 铺垫知识:

**$C(n,k)$** : 从  $n$  个元素中选  $k$  个的不重复选法总数 (不排序)。

### 1. 核心公式

$$C(n, k) = \frac{n!}{k! \cdot (n-k)!}$$

- 规定:  $C(n, 0) = 1, C(n, n) = 1$
- $k > n$  或  $k < 0$  时, 结果 = 0

```
884 #60页87题, 铺垫:
885 #组合数的计算公式依赖阶乘, 可以先自定义一个f(n)函数, 返回n的阶乘
886 def f(n):
887     乘积=1
888     for i in range(2,n+1):
889         乘积*=i
890     return 乘积
891 print(f(5))#验证一下, f(5)是5的阶乘(5!)
```

控制台

```
120
程序运行结束
```

```
895 #60页87题, :
896 def f(n):
897     乘积=1
898     for i in range(2,n+1):
899         乘积*=i
900     return 乘积
901 #根据已经定义好的, 计算阶乘的公式, 完成自定义函数combination(n,k)的定义
902 #combination(n,k)是用来计算从n个不同元素中选k个元素进行组合时, 所有组合的个数
903 def combination(n,k):
904     return f(n)//(f(k)*f(n-k))
905 print(combination(5,2))#验证一下, combination(5,2)为10
```

控制台

```
10
程序运行结束
```

```
909
910 #60页87题, 用标准模块math模块中的阶乘函数factorial做
911 from math import factorial as f
912 def combination(n,k):
913     return f(n)//(f(k)*f(n-k))
914 print(combination(5,2))
```

控制台

```
10
程序运行结束
```

```
917
918 #60页87题, 用标准模块math模块中的组合数函数comb做
919 from math import comb as c
920 def combination(n,k):
921     return c(n,k)
922 print(combination(5,2))
```

控制台

10  
程序运行结束

```
925 #61页88题, 铺垫: 需要一个自定义函数f(n)判断一个整数n是否为质数
926 # 如果n是质数, f(n)返回True
927 # 如果n不是质数, f(n)返回False
928 - def f(n):
929 -     if n<2:
930         return False#小于2的整数不是质数 (返回False)
931 -     else:
932 -         for i in range(2,n):
933 -             if n%i==0:#如果在2到n-1的区间发现了n的因数
934                 return False#则n不是质数 (返回False)
935         return True#如果上述循环, 每次的if条件都不成立
936         #说明没有在2到n-1的区间发现n的因数
937         #说明n是质数, (返回True)
938 #验证一下f(n)能否判断出0到100的所有质数:
939 质数列表=[]
940 - for i in range(101):
941 -     if f(i):
942         质数列表.append(i)
943 print(质数列表)
```

控制台

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]  
程序运行结束

```

944 #61页88题, 铺垫: 判断一个大于2的正整数n是不是质数
945 #没必要验证2到n-1中是否存在n的因数
946 #只需要验证2到 n的平方根中是否存在n的因数即可
947 #对于自定义函数f(n)可以作优化, 少验证一些整数
948
949 def f(n):
950     if n<2:
951         return False
952     else:
953         n的平方根=n**0.5#或写为 n**(1/2)
954         m=int(n的平方根)+1
955         for i in range(2,m):
956             if n%i==0:
957                 return False
958         return True
959
960 质数列表=[]
961 for i in range(101):
962     if f(i):
963         质数列表.append(i)
964 print(质数列表)

```

控制台

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
程序运行结束
```

```

968 #61页88题, 铺垫: 判断一个大于2的正整数n是不是质数
969 #小于2的整数不是质数
970 #2是质数
971 #大于2的偶数不是质数
972 #对于大于2的奇数n, 只需要验证3到 n的平方根中, 的奇数中, 是否存在n的因数即可
973 #对于自定义函数f(n)可以作优化, 少验证一些整数
974
975 def f(n):
976     if n<2:
977         return False
978     elif n==2:
979         return True
980     elif n%2==0:
981         return False
982     else:
983         m=int(n**0.5)+1
984         for i in range(3,m):
985             if n%i==0:
986                 return False
987         return True
988
989 质数列表=[]
990 for i in range(101):
991     if f(i):
992         质数列表.append(i)
993 print(质数列表)

```

控制台

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
程序运行结束
```

```

993 | #61页88题, 对于自定义函数is_prime_and_factors(num)
994 | # 可以先用f(num)判断num是否为质数
995 | def f(n):
996 |     if n<2:
997 |         return False
998 |     elif n==2:
999 |         return True
1000 |     elif n%2==0:
1001 |         return False
1002 |     else:
1003 |         m=int(n**0.5)+1
1004 |         for i in range(3,m):
1005 |             if n%i==0:
1006 |                 return False
1007 |         return True
1008 |

```

```

1009 | def is_prime_and_factors(num):
1010 |     if f(num):
1011 |         return True,[]#也可以写作return (True,[])
1012 |         #如果返回多个数据, 用英文逗号隔开, 默认是把这些分散的数据放进元组, 一起返回
1013 |     else:
1014 |         #根据题目的意思, 如果num不是质数, 则对num作质因数分解, 把质因数分解的结果放进列表
1015 |         #举例来说, 若num是12, 因为对12进行质因数分解的结果是 2×2×3,
1016 |         #所以12的质因数分解列表为[2,2,3]
1017 |         L=[]#先准备一个空列表, 以便于发现num的质因数时, 把质因数放进去
1018 |         #用循环判断出num所有的因数:
1019 |         m=1#m是num起始的因数
1020 |         while num>1:#如果发现m是num的因数, 那么将num更新为num除以m的商, 所以num会不断缩小, 当num缩小到1的时候, 退出循环
1021 |             while num%m==0 and f(m):#如果发现m是num的因数, 并且m也是质数, 就把m放进列表(即m是num的质因数)
1022 |                 L.append(m)
1023 |                 num//=m#将num缩小为原来的m分之1, 然后再次判断m还是不是num的质因数
1024 |             m+=1
1025 |         return False,L
1026 | print(is_prime_and_factors(12))
1027 |

```

控制台

```

(False, [2, 2, 3])
程序运行结束

```

□

```

1026 | print(is_prime_and_factors(13))
1027 |

```

控制台

```

(True, [])
程序运行结束

```