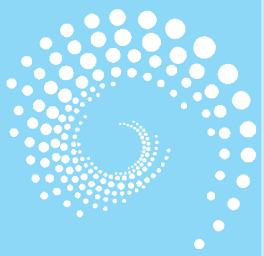
第3章

基本语句

CHAPTER 3



在编程语言中，基本语句是程序逻辑表达与功能实现的基础单元。Python作为一门广泛应用于数据科学、人工智能等领域的高级语言，其基本语句承担着数据处理、流程控制等核心功能。掌握Python的基本语句不仅是学习编程的第一步，更是深入理解算法和高级编程的基础。

本章通过基本语句的学习与实践，帮助读者将问题求解思路转化为可执行的计算机指令，从而实现程序功能。

3.1 算法描述与流程图

计算机解决问题必须按照一定的方法和步骤循序渐进地进行。对于求解的同一问题，可以有不同的方法和步骤，这些方法与步骤在运行效率、占用内存资源上可能存在很大区别。因此，如何将复杂问题简单化，简单问题流程化，是程序设计之前需要重点研究的工作。算法就是解决某个特定问题的方法和步骤，计算机科学就是研究算法的科学，本节将从算法的概念与描述方法两方面对算法进行阐述。

3.1.1 算法的基本概念

瑞士计算机科学家Niklaus Writh提出了“程序＝算法＋数据结构”公式，该公式揭示了程序的两大核心要素：一是对数据的组织与描述(数据结构)，二是对数据的操作方法(算法)。该公式强调了算法效率与数据结构选择在程序设计中的重要性。算法应该具备以下特点。

(1) 有穷性：一个算法的步骤必须是有限的。

(2) 确定性：算法的每个步骤必须具有确切的定义，无二义性。

(3) 可行性：算法能够转换为程序语言，且能够有效地执行。

(4) 输入性：算法必须有初始量。

(5) 输出性：算法有一个或多个输出。

算法是描述某一问题求解的有限步骤，必须要能在有限的时间内完成，对相同的输入有相同的输出，侧重于逻辑框架，关注解决问题的思路。程序设计则是将算法转化为具体的编程语言实现，着重于代码层面的执行过程。

3.1.2　算法的描述方法

有很多工具描述算法，比如自然语言、伪代码、N-S 流程图、传统流程图，本书使用自然语言和传统流程图(以下简称流程图)来描述算法。

1. 使用自然语言描述算法

自然语言描述，是指使用人们日常使用的语言(如中文、英文)来描述算法。其优点是通俗易懂，不需要具备专业的编程知识就能理解，适合相对简单问题的算法描述。

【例 3.1】 判断正整数 n(n>1)是否为质数。

【求解思路】

对 n 的因子进行穷举搜索。用自然语言描述如下。

(1) 从 2 开始搜索 n 可能的因子：k=2。

(2) 如果 k<n，转(3)；否则，n 是质数，转(5)。

(3) 如果 n 能被 k 整除，则 n 不是质数，转(5)。

(4) k 增加 1，转(2)。

(5) 结束。

2. 使用流程图描述算法

在流程图中，使用各种图形符号(图 3.1)并结合文字说明来表示算法的逻辑结构和执行流程。其优点是直观形象，能够清晰地展示算法的整体框架和各个步骤之间的关系，便于理解和分析算法的流程。

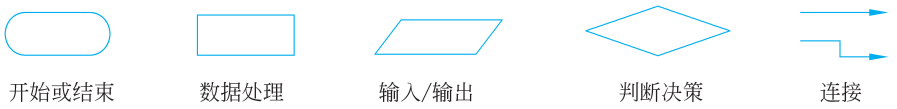


图 3.1　流程图的常用图形符号

【例 3.2】 用流程图描述例 3.1 的算法，如图 3.2 所示。

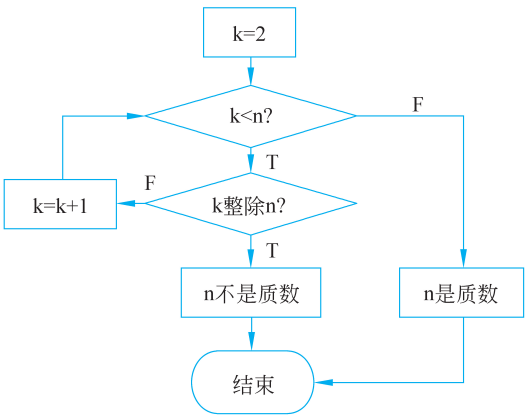


图 3.2　判断 n 是否为质数的流程图

3.2 if 条件语句的使用

Python 程序默认按顺序执行(顺序结构),但可通过条件判断改变程序流程。其中,if 语句作为基础条件判断工具,用于在条件满足时执行对应代码块,否则跳过并继续执行后续语句,从而实现程序流程的控制。

3.2.1 if 条件语句的基本语法

if 条件语句有三种形式的结构:单分支、双分支和多分支。

1. 单分支 if 语句

(1) 语法格式:

```
if 条件:
    语句块
```

(2) 执行流程。

如果条件成立,则执行语句块;否则,直接跳出 if 语句,如图 3.3 所示。

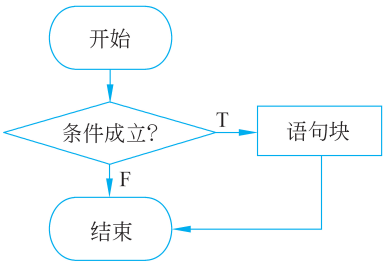


图 3.3 单分支 if 语句的执行流程

【特别提示】

(1) 由一条或多条语句组成的语句,称为语句块。同一个语句块的语句必须左对齐。

(2) 语句块不能与 if 语句左对齐,必须右缩进(默认 4 个空格)。

(3) 当语句块比较简短时,也可不换行。例如,if a>b: b=a。

(4) 在 Python 中,冒号(:)是重要的语法符号,用于标识代码块的开始。编写代码时,如果输入冒号后按回车键,会自动换行且右缩进。

【例 3.3】 输入两个整数,输出它们的较大值。

```
s=input("输入两个整数:")
a,b=eval(s)                    #将整数串转换为两个整数
if b>a:
    a=b                        #仅当条件 b>a 成立时执行
print(f"较大值={a}")
```

【练一练 3.1】 如果不能改变 a 和 b 的值,如何修改以上代码?

2. 双分支 if-else 语句

(1) 语法格式。

```
if 条件:
   语句块 1
else:
   语句块 2
```

（2）执行流程。

如果条件成立，则执行语句块 1；否则，执行语句块 2，如图 3.4 所示。

【例 3.4】　输入两个整数，输出它们的较大值。

```
s=input("输入两个整数:")
a,b=eval(s)                                    #将整数串转换为两个整数
if a>b:
    print(f"较大值={a}")                        #当条件 a>b 成立时执行
else:
    print(f"较大值={b}")                        #当条件 a>b 不成立时执行
```

【练一练 3.2】　修改本例代码，将较大值保存到另一变量 c 中，最后输出 c 的值。

3. 多分支 if-elif-else 语句

（1）语法格式。

```
if 条件 1:
    语句块 1
elif 条件 2:
    语句块 2
else:
    语句块 3
```

（2）执行流程。

如果条件 1 成立，则执行语句块 1；否则，如果条件 2 成立，则执行语句块 2；否则，执行语句块 3。执行流程如图 3.5 所示。

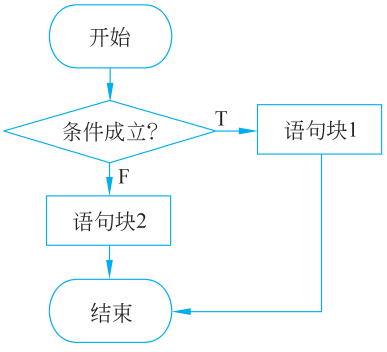


图 3.4　if-else 语句的执行流程

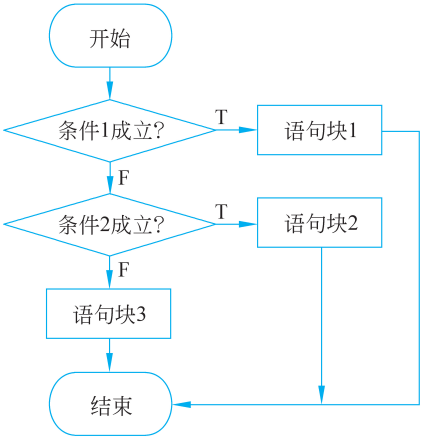


图 3.5　if-elif-else 语句的执行流程

【特别提示】

（1）语句块不能与 if、elif 和 else 关键字左对齐，必须右缩进（默认 4 个空格）。

（2）如果语句块简短，也可以不换行。

（3）一个 if 语句中可以使用多条 elif 子句，但最多只能搭配一条 else 子句。

（4）else 子句必须写在 elif 子句之后，且可缺省 else 子句。

【例 3.5】　已知 x＞0 时 y 的值为 2x；x＝0 时 y 的值为 1；x＜0 时 y 的值为 6＋x。输入 x 的值，输出 y 的值。

```
x=float(input("输入 x 的值:"))
if x>0:y=2*x
elif x==0:y=1
else:y=6+x
print(f"x={x}时 y={y}")
```

【练一练 3.3】 给定满足 x>0、x<0 和 x==0 条件下的 x 值,计算表达式(x>0)*2*x+(x<0)*(6+x)+(x==0)的值。请分析比较此运算表达式与 if 条件语句实现分段函数时的运算效率与代码简洁性,并探讨在实际编程中如何平衡代码执行效率与可读性之间的关系。

3.2.2 if 表达式

if 表达式的语法格式:

```
A if B else C
```

执行流程:如果 B 成立,返回表达式 A 的值;否则,返回表达式 C 的值。

【例 3.6】 输入成绩,60 分及以上输出“及格”,否则,输出“不及格”。

```
cj=input("输入成绩(0-100 上的整数):")
cj=int(cj)
pd="及格" if cj>=60 else "不及格"
print(f"成绩{cj},评定为{pd}")
```

【练一练 3.4】 使用 if 语句输出成绩评定结果。

【特别提示】

(1) if 表达式不同于 if 语句,if 表达式有值,通常用于赋值语句中;if 语句的子句不一定是赋值语句,分支处理更灵活。

(2) if 表达式完全可以利用 if 语句实现,但多分支、较复杂的 if 语句难以转换为等效的 if 表达式。

(3) if 表达式中也可以使用 if 表达式(称为嵌套)。

【例 3.7】 成绩评定。

```
cj=input("输入成绩(0~100 的整数):")
cj=int(cj)
pd="D" if cj<60 else ("C" if cj<75 else ("B" if cj<90 else "A"))
print(f"成绩{cj},评定为{pd}")
```

【练一练 3.5】 使用表达式 chr(65+(cj<60)+(cj<75)+(cj<90))输出成绩评定结果。

3.2.3 if 条件语句的嵌套使用

if 语句的嵌套是指在 if 语句的子句中又使用了 if 语句。

【例 3.8】 输入三个整数,输出它们中的最大数。

```
s=input("输入三个整数:")
a,b,c=eval(s)
if a>b:
    if a>c:print("最大数是:",a)
    else:print("最大数是:",c)
elif b>c:print("最大数是:",b)
else:print("最大数是:",c)
```

【练一练 3.6】　使用 if 语句嵌套，输出三个整数中的最小数。

3.3　match 条件语句

在 Python 3.10 版中，引入了 match 语句，提供了一种更结构化、更简洁的方式来进行模式匹配和条件分支处理。

3.3.1　match 条件语句的基本语法

1. match 语句基本语法

语法格式：

```
match  A:
    case B1:
       C1
    case B2:
       C2
    …
    case Bn:
       Cn
```

说明：

(1) A 为表达式。

(2) B1、B2、……、Bn 为模式，可以是值、变量或通配符，通配符的使用参阅 3.3.2 节。

(3) C1、C2、……、Cn 为语句块。

(4) 至少需要一条 case 子句。

2. 执行流程

根据 A 的值从上而下逐一匹配 case 子句。如果存在匹配，则执行对应的语句块；否则，不执行任何操作。执行流程如图 3.6 所示。

【例 3.9】　成绩评定。

```
s=input("输入成绩:");s=int(s)
match (s<60)+(s<80):
    case 0:print("优秀")
    case 1:print("及格")
    case 2:print("未通过")
```

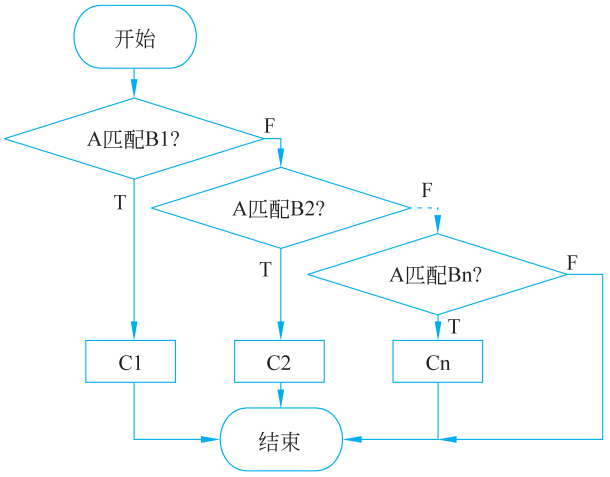


图 3.6 match 语句的执行流程

3.3.2 match 条件语句的高级特性

1. 使用通配符

在 match 条件语句中,通配符在模式匹配中发挥着重要作用,可用于匹配任意值或者一组值,匹配更加灵活。主要通配符如下。

(1) _(单下画线)。这是最基础的通配符,可以匹配任意单个值。“case _”通常用于不满足任何匹配条件时的处理。

【例 3.10】

```
month=input("输入月份:")
match month:
    case '3'|'4'|'5':print(f"{month}月在春季")
    case '6'|'7'|'8':print(f"{month}月在夏季")
    case '9'|'10'|'11':print(f"{month}月在秋季")
    case _:print(f"{month}月在冬季")
```

(2) * _(星号加单下画线)。用于匹配任意数量(包含零个)的值序列,通常用在序列模式匹配中,会将匹配到的元素作为一个序列忽略掉。

【例 3.11】

```
nums=[1, 2, 3, 4, 5]
match nums:
    case [first, * _, last]:print(f"first={first} last={last}")
```

(3) **__(双星号加双下画线)。用于字典模式匹配,它可以匹配字典中剩余的任意键值对。

【例 3.12】

```
d={'name': 'WPing', 'age': 20,'SN':2401}
match d:
    case {'name': name, **__}:print(f"name={name}")
```

（4）*变量名和**变量名。“*变量名”用于捕获序列中剩余的元素并将其作为一个列表绑定到指定变量；“**变量名”用于捕获字典中剩余的键值对，并将其作为一个字典绑定到指定变量。

【例 3.13】

```
nums=[1, 2, 3, 4, 5]
match nums:
    case [first, *rest]:print(f"first={first},rest={rest}")
d={'name': 'WPing', 'age': 20,'SN':2401}
match d:
    case {'name': name, **others}:
        print(f"name={name},others={others}")
```

【代码解释】　在序列匹配中，*rest 捕获了列表中除第一个元素之外的所有元素，rest 为列表；**others 捕获了字典中除 'name' 键之外的所有键值对，others 为字典。

2. 多模式组合匹配

可以使用“|”符号将多个模式组合在一起，只要满足其中一个模式即为匹配成功。

【例 3.14】

```
s=input("输入成绩:")
s=int(s)//10
match s:
    case 6 | 7 | 8:print("及格")
    case 9 | 10:print("优秀")
    case _:print("未通过")
```

3. 带守卫的模式匹配

守卫是在 case 语句中使用 if 子句，用于添加额外的条件判断，让匹配更加灵活。只有当模式匹配且守卫条件为真时，才会执行对应的代码块。

【例 3.15】

```
s=input("输入成绩:")
s=int(s)
match s:
    case x if x<60:print("未通过")
    case x if x<90:print("及格")
    case _:print("优秀")
```

4. 嵌套模式的匹配

在 match 语句中，可以使用嵌套的模式来匹配复杂的数据结构，如嵌套的列表、元组或字典。

【例 3.16】

```
data=('person', {'name': 'Alice', 'age': 25})
match data:
    case ('person', {'name': name, 'age': age}):
        print(f"{name}的年龄是{age}")
```

【代码解释】　外层模式先匹配元组，确定第一个元素为'person'后，内层模式再匹配字

典,提取出键'name'的键值给变量 name,提取出键'age'的键值给变量 age。

3.4 while 循环语句

3.4.1 while 循环语句的基本语法

while 循环语句是实现循环结构的重要语句之一,主要用于在满足特定条件的情况下重复执行代码块,直到条件不满足或强行终止为止。

1. 语法格式

```
while 条件:
    语句块(循环体)
```

2. 执行流程

当条件为 True 时,执行语句块,直到条件为 False,终止循环。执行流程如图 3.7 所示。

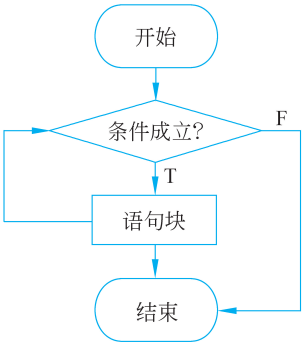


图 3.7 while 循环语句的执行流程

【特别提示】

(1) 如果"语句块"较短,也可以不换行,直接写在冒号之后。

(2) 应在"语句块"中改变循环的条件,以避免因循环条件始终为真而导致无限循环(死循环)。

(3) 调试代码时,一旦出现死循环,可按 Ctrl+C 组合键中断。

【例 3.17】 计算和值:1+2+…+100。

```
i,s=1,0
while i<=100:
    s+=i
    i+=1
print("1+2+...+100=",s)
```

【练一练 3.7】 利用公式$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}\cdots$计算 π 的近似值(保留到小数点后 6 位)。

3.4.2 使用 break 和 continue 转向语句

1. break 语句

在 while 语句的循环体中,可以使用 break 语句强行终止循环,它通常与 if 语句搭配使用。

【例 3.18】 设 s=1+2+…+n,n 为正整数。求满足 s>1200 的最小值 s。

```
s,n=0,1
while True:
    s+=n
```

```
    if s>1200:break                    #当条件成立,退出循环
    n+=1
print(s)
```

【练一练 3.8】　不使用 break 语句求解例 3.18。

2. continue 语句

在 while 语句的循环体中,可以使用 continue 语句结束本次循环,直接跳转到下轮次循环。它通常与 if 语句搭配使用。

【例 3.19】　计算不能被 6 整除的所有 4 位正整数的和。

```
s=0;k=1000
while k<10000:
#当 k 是 6 的倍数时,执行 k+=1 和 continue 语句,转去检查 k<10000 条件是否成立
    if k%6==0:k+=1;continue
    s+=k;k+=1
print(f"不能被 6 整除的所有 4 位正整数的和为{s}")
```

【练一练 3.9】　不使用 continue 语句求解例 3.19。

3.4.3　使用 else 子句

在 Python 中,else 子句不仅可以与 if 语句搭配使用,也可以与 while 语句搭配使用。当循环条件不满足时,执行 else 子句的语句块。

【例 3.20】　质数判断。

```
n=int(input("输入一个整数:"))
if n<2:print(f'{n}不是质数')
else:
    k=2
    while k<n:
        if n%k==0:print(f'{n}不是质数');break
        k+=1
    else: print(f'{n}是质数')
```

【练一练 3.10】　不使用 else 子句求解上例问题。提示:可以利用一个变量 f 标记判断结果。其初值为 1(表示 n 是质数)。当 n 不是质数时(即 n%k==0 时),执行 f=0 并退出循环。循环结束后,最后根据 f 的值来确定输出结果。

3.5　for 循环语句

3.5.1　for 循环语句的基本语法

for 循环语句用于遍历可迭代对象,即按照一定的顺序访问可迭代对象的每个元素,且每个元素只被访问一次。

1. 语法格式

```
for 循环变量 in 可迭代对象:
    语句块(循环体)
```

2. 执行流程

对可迭代对象的每一元素执行一次语句块,直到遍历完成,如图 3.8 所示。

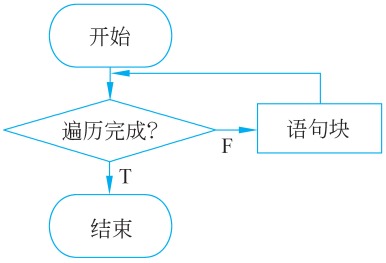


图 3.8 for 语句的执行流程

【例 3.21】 用 for 语句计算和值 1+2+…+100。

```
s=0
for i in range(1,101):s+=i
print("1+2+...+100=",s)
```

【代码解释】

(1) range()函数是 Python 内置函数,其语法格式为 range(a,b,c),返回一个以 a 为首项、b 为终止项(不包含该项)、公差为 c 的等差数列。该等差序列称为 range 对象,是可迭代对象,a、b、c 均为整数。特别地,range(n)返回的是首项为 0、终止项为 n-1、公差为 1 的等差数列;range(a,b)返回的是首项为 a、终止项为 b-1、公差为 1 的等差数列。

(2) 语句"for i in range(1,101): s+=i"的执行过程:循环变量 i 依次取值 1,2,…,100。对 i 的每一次取值,均执行语句"s+=i"。

【练一练 3.11】 利用 range(2,101,2)计算 2+4+6+…+100,即求 100 以内的偶数之和。

【特别提示】 在 for 语句中也可使用:

(1) continue 语句,表示终止本轮循环,直接跳转到下一轮循环。

(2) break 语句,表示强行终止循环。

(3) else 子句,表示遍历完成后需执行的语句。

【例 3.22】 判断序列中是否存在不能被 3 整除、但能被 7 整除的整数。

```
a=eval(input("输入整数序列:"))
for k in a:
    if k%3==0:continue
    if k%7==0:print(f"不能被 3 整除、能被 7 整除的数是:{k}");break
else:print("不能被 3 整除、能被 7 整除的数:不存在")
```

3.5.2 循环语句的嵌套使用

循环语句的嵌套是指在 for 或 while 语句的循环中还可以使用 for 或 while 语句。

【例 3.23】　计算 1!＋2!＋…＋10!。

```
s=0
for i in range(1,11):                    #外循环:计算 i 的阶乘并加到 s
    t=1                                  #i 的阶乘的初值为 1
    for j in range(1,i+1):t*=j           #内循环:计算 i 的阶乘
    s+=t
print(f"1!+2!+...+10!={s}")
```

【练一练 3.12】　将上述代码的外循环或内循环改为 while 语句。

【特别提示】

(1) 循环嵌套时,外层循环中可以包含一个或多个内层循环语句。

(2) 理论上循环结构的嵌套层数不受限制,但实际应用中嵌套层次一般不超过 3 层,以保证程序的可读性。

(3) 在例 3.23 的代码中,内循环均重复计算 j!,没有利用 j!＝(j－1)! ＊j,可如下优化:

```
s,t=0,1
for i in range(1,11):
    t*=i
    s+=t
print(f"1!+2!+...+10!={s}")
```

3.5.3　利用推导式创建数据对象

在 Python 中,推导式是一种简洁且高效的创建可迭代对象的方式。它可以根据已有的可迭代对象,通过特定的规则快速生成新的可迭代对象。

1. 利用推导式创建列表

语法格式:

```
[A for B in C if D]
```

说明:根据可迭代对象 C 的每一满足条件 D 的元素 B 生成列表元素 A,其中 B 会遍历 C。这里的 if 子句可缺省。

【例 3.24】

```
a=[x*x+1 for x in range(5)]
print(a)                                 #输出:[1,2,5,10,17]
b=[x for x in a if x%5>0]                #从列表 a 中筛选出不能被 5 整除的元素
print(b)                                 #输出:[1,2,17]
```

2. 利用推导式创建字典

语法格式:

```
{A:B for C in D if E}
```

说明:对可迭代对象 D(元素必须是含 2 个数据的对象)的每一满足条件 E 的元素 C,生成字典的键值对(A 为键,B 为键值)。这里的 if 子句可缺省。

【例 3.25】

```
a=[["A",20],["B",18],["C",21]]
d={x:y for x,y in a}                                    #d={'A': 20, 'B': 18,'C':21}
d={x[0]:x[1] for x in a if x[1]%2==0}                   #d= {'A': 20, 'B': 18}
```

3. 利用推导式创建集合

语法格式:

```
{A for B in C if D}
```

说明:B 遍历可迭代对象 C,如果 B 满足条件 D 则生成集合的元素 A。推导式会自动删除重复的元素。A 必须是不可变对象。这里的 if 子句可缺省。

【例 3.26】

```
a={x**2 for x in range(10)}                             #包含 0 到 9 的平方的集合
b={x for x in range(30) if x%3+x%5 ==1}                 #b={6,10,21,25}
```

【特别提示】

(1) 利用推导式不能创建元组。推导式(x for x in range(10) if x%5>0)创建的是 Generator 对象(生成器对象)。生成器是一种特殊的可迭代对象,它不会一次性生成所有元素,而是在需要时逐个生成,以节省内存。

(2) 推导式可以嵌套使用,即一个推导式内部包含另一个推导式。推导式的嵌套虽然可以使代码简洁,但过度使用可能会降低代码的可读性,因此应根据实际情况合理运用。

【例 3.27】

```
a=[[(i,j) for j in range(3)] for i in range(2)]
print(a)                          #输出:[[(0,0),(0,1),(0,2)],[(1,0),(1,1),(1,2)]]
```

【代码解释】 外层循环 for i in range(2) 控制生成 2 个子列表。对于每次外层循环,内层循环[(i,j) for j in range(3)] 生成子列表。执行过程如下。

(1) i=0。

(2) [(i,j) for j in range(3)]生成子列表[(0,0),(0,1),(0,2)]。

(3) i=1。

(4) [(i,j) for j in range(3)]生成子列表[(1,0),(1,1),(1,2)]。

3.5.4 while 语句与 for 语句的比较与应用选择

1. 语法比较

for 语句主要用于遍历可迭代对象(如列表、元组、字符串、字典等)。

while 语句主要用于基于条件判断来控制循环的执行,只要条件表达式的值为 True,就会一直执行循环体。

相同点:均可使用 continue 语句、break 语句和 else 子句。

2. 特点比较

for 语句简洁,循环次数由可迭代对象的元素个数决定,循环次数是确定的,不需要管理

计数器。

while 语句的循环条件可以是任意表达式，非常灵活，可以根据不同的条件动态控制循环的执行，适用于不确定循环次数的情况。while 语句需要在循环体中更新条件表达式中的变量，否则可能会导致无限循环。

3. 应用选择

(1) 使用 for 语句的场景。

遍历可迭代对象：当需要遍历列表、元组、字符串等可迭代对象中的每个元素时，使用 for 语句更方便快捷。

固定次数的循环：如果循环的次数是已知的，可以使用 range()函数结合 for 语句来实现固定次数的循环。

(2) 使用 while 语句的场景。

不确定循环次数：当无法确定循环的次数，而是根据某个条件来决定是否继续循环时，使用 while 语句更为合适。

动态条件控制：当循环的条件需要根据循环体中的操作动态变化时，使用 while 语句。在实际编程中，需要根据具体的需求和场景来选择使用 for 语句还是 while 语句，有时候也可以结合使用来实现更复杂的应用逻辑。

3.6　异常处理语句

3.6.1　异常处理的基本概念

异常(Exception)指的是在程序执行期间出现的错误或意外事件，它会中断程序的正常执行流程，甚至导致程序崩溃。

1. 异常的产生

异常的产生(触发异常)可能来自以下几个方面。

(1) 用户输入错误。用户输入的数据类型与程序的要求不符。

(2) 文件操作问题。尝试打开不存在的文件，或者对文件无读/写权限。

(3) 网络问题。在进行网络请求时，可能会遇到网络连接失败、超时等问题。

(4) 程序逻辑错误。例如，用一个数除以零，就会引发异常。

(5) 在代码中，当设置的条件满足时主动抛出异常。

2. 常见的异常类型

当触发异常时，Python 会自动创建异常对象。这些对象常见的类型如下。

(1) SyntaxError：语法错误，通常是由于代码书写不符合 Python 语法规则导致的，比如缺少冒号、括号不匹配等。

(2) TypeError：类型错误，当操作或函数应用于不适当类型的对象时引发。例如，对整数和字符串进行加法运算。

(3) ValueError：值错误。例如，int("abc")会触发 ValueError 异常，因为"abc"无法转换为整数。

(4) ZeroDivisionError：除零错误。用一个数除以零会触发该异常。

(5) IndexError：索引错误。当序列对象的索引超过有效范围时触发该异常。

(6) NameError：标识符未定义错误。例如,使用了未定义的变量或函数时会触发该异常。

(7) Exception：是所有内置异常类的基类。

3. 异常处理

如果程序中出现异常但没有进行处理,程序通常会终止执行,并输出错误信息。为了避免程序因异常而意外终止,提高程序的健壮性,需要对异常进行处理。通常使用 try 语句来捕获,并处理异常。

3.6.2 使用 try-except 语句处理异常

1. 语法格式

```
try:
    语句块 A
except:
    语句块 B
```

2. 执行流程

执行语句块 A,如果有语句触发异常,则跳转到语句块 B 继续执行。

【例 3.28】

```
while True:
    try:
        n=int(input("请输入一个整数:"))          #输入数据不能转换为整数时触发异常
        break                                    #上一语句未触发异常,退出循环
    except:continue                              #异常处理:继续执行下轮循环
```

【代码解释】 当输入的不是整数时,要求重新输入。

【特别提示】

except 子句之后可以指定异常的类型。未指定时捕获所有异常,这时 try 可以匹配多个 except 子句。

【例 3.29】

```
try:x=a/b
except NameError:print("变量 a 或 b 未定义")
except TypeError:print("变量 a 或 b 的数据类型错误")
except Exception as e:print(e)
```

【代码解释】 except 子句之后指定异常类型时,可以通过 as e 来获得异常的详细信息并保存到变量 e。

3.6.3 使用 try-except-finally 语句处理异常

1. 语法格式

```
try:
    语句块 A
```

```
except:
    语句块 B
else:
    语句块 C
finally:
    语句块 D
```

2. 执行流程

执行语句块 A，如果有语句触发异常，则跳转到语句块 B 继续执行；否则执行语句块 C。无论语句块 A 是否触发异常，均将执行语句块 D。

【特别提示】

(1) else 子句和 finally 子句可以缺省。如果 try 与 finally 搭配使用，没有 except 子句，则触发的异常未处理。

(2) try 或 except 语句块中包含 break、continue 或者 return 语句时，finally 中的语句也会被执行，return 语句在第 4 章介绍。

【例 3.30】 输入运算表达式，计算结果。若表达式错误，则给出错误提示，并重新输入。

```
while True:
    s=input("输入表达式:")
    try:v=eval(s)
    except:print("表达式错误!");continue
    else:print(f"{s}={v}");break
    finally:print(s)
```

【代码解释】 v=eval(s)用于计算输入的表达式的值。如果触发异常，执行 except 后的语句，会显示“表达式错误!”，执行 finally 后的语句，显示输入的表达式，又因为有 continue 语句，会继续显示“输入表达式:”。如果未触发异常，会执行 else 后的语句，显示表达式和表达式的运算结果，也会执行 finally 后的语句，显示输入的表达式，并执行 break 语句，终止循环。

3.6.4　使用 assert 或 raise 语句抛出异常

1. 使用 assert 语句抛出异常

语法格式：

```
assert 表达式[,信息]
```

参数说明：表达式的值会自动转换为逻辑值(True 或 False)。转换后的值为 False 时触发 AssertionError 异常。这里的“信息”通常是描述异常信息的字符串。

语句作用：如果表达式的值为假则触发 AssertionError 异常；否则，则不执行任何操作。

【特别提示】

assert 语句通常在 try 语句中使用；否则，抛出异常后会导致程序终止。

【例 3.31】 输入控制。

```
s=input("请输入一个数:")
try:
```

```
    x=int(s)                                    #s 不能转换为整数时会触发异常
    assert 0<=x<=100,"未输入[0,100]的整数!"      #x 不是[0,100]的整数时触发异常
except Exception as e:print(e)                  #处理异常:输出异常信息
else:print(x)                                   #未触发异常:输出 x 的值
```

【练一练 3.13】 如何控制当输入的数据不是[0,100]的整数时,要求用户重新输入?

2. 使用 raise 语句抛出异常

语法格式:

```
raise a
```

参数说明:a 指定抛出的异常类型。

语句作用:抛出异常,并创建该异常类的实例(可传递描述异常信息的字符串)。raise 语句通常结合 if 语句使用。

【例 3.32】 求解一元二次方程 $ax^2+bx+c=0$。输入无效或无解时触发异常。

```
try:
    a,b,c=eval(input("输入一元二次方程系数 a,b,c:"))
    if a==0:raise ValueError("二次项的系数不能为 0!")
    c=b*b-4*a*c
    if c<0:raise ValueError("方程无解!")
    x1=(-b+c**0.5)/(2*a)
    x2=(-b-c**0.5)/(2*a)
    print(f"x1={x1},x2={x2}")
except Exception as e:print(e)
```

【练一练 3.14】 修改代码,用 assert 语句抛出异常。

3.7 综合应用案例

案例 3:完数判断

【问题描述】

正整数 n 的全部真因子(小于 n 的正整数 k 如果能够整除 n,则称 k 为 n 的真因子)之和等于 n,则称 n 为完数。输入 n,若 n 是完数则输出 n 的真因子组成的和式;否则输出 None。

程序执行效果:

```
输入 1 个正整数:28
输出:28=1+2+4+7+14
```

【案例分析】

对于正整数 n,如果 n<6,则 n 不是完数。

当 n≥6 时,可在[1,n-1]搜索 n 的真因子,求出全部真因子的和值 s 及和式 t。如果 s=n,则 n 是完数,输出 t;否则 n 不是完数。算法可描述如下(执行流程见图 3.9)。

(1) 初始化真因子的和值 s=1 及和式 t=f"{n}=1"。

(2) 从 2 开始搜索 n 的真因子：i=2。

(3) 如果 i<n,转至步骤(4);否则转至步骤(6)。

(4) 如果 n 能被 i 整除,则更新 s=s+i,t=t+"+"+str(i)。

(5) i=i+1(搜索下一个可能的因子),转至步骤(3)。

(6) 如果 s=n,则 n 是完数,输出和;否则输出 None。

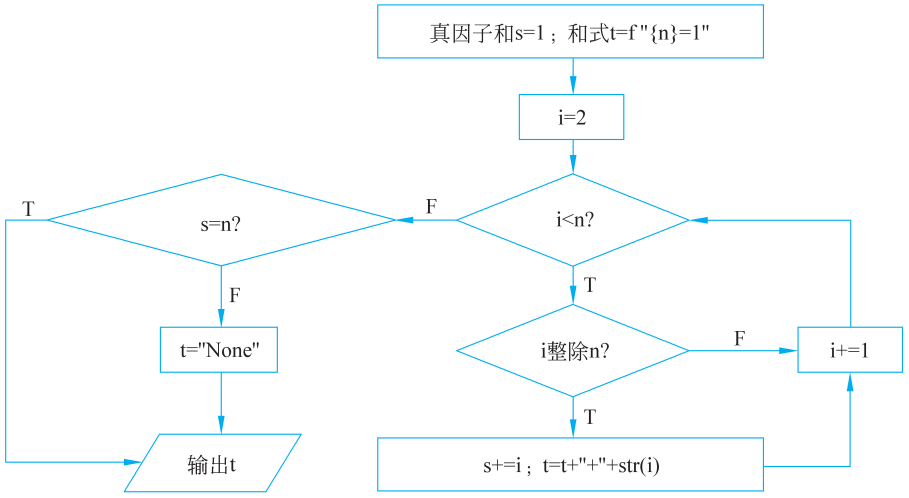


图 3.9　完数判断流程

【参考代码】

```
try:
    n=int(input("输入 1 个正整数:"))
    assert n>5
except:print("None")
else:
    s=1;t=f"{n}=1";i=2
    while i<n:
        if n%i==0:s+=i;t=f"{t}+{i}"
        i+=1
    if s!=n:t="None"
    print(t)
```

【拓展训练】

(1) 找出[1,1000]的全部完数。

(2) 设 k 是 n 的真因子,则 n÷k 也是 n 的真因子。因此,只须在满足 k×k≤n 的 k 中搜索 n 的真因子即可。请按照该思路优化代码。注意：当 k 是 n 的真因子时,n÷k 可能与 k 相等。

(3) 使用列表递推式求出 n 的全部真因子。

案例 4：质数搜索

【问题描述】

输入正整数 m、n(m<n),输出[m,n]质数的个数。程序执行效果：

```
输入 2 个正整数:3,1000
[3,1000]上共有 167 个质数
```

【案例分析】

1. 质数的判断

如果正整数 k 只能被 1 和 k 两个正整数整除,则称 k 为质数。因此,如果正整数 k 在区间[2,k-1]上无约数,则 k 是质数;否则不是质数。

2. 质数的搜索与统计

基本思路:利用循环,对[m,n]的每一个整数 k 逐一判断,如果 k 是质数,则质数个数增加 1。

【参考代码】

```
m,n=eval(input("输入两个正整数:"))
k=m;s=0                          #k 为[m,n]的第一个整数。s 为质数个数
while k<=n:                      #k 依次取[m,n]的每一个整数
    a=2                          #[2,k-1]k 的第一个可能的约数
    while a<k:                   #在[2,k-1]搜索 k 的约数 a
        if k%a==0:break          #a 是 k 的约数时终止内循环
        a+=1                     #搜索 k 的下一个可能的约数
    else:s+=k>1                  #内循环完成,k>1 时 k 是质数
    k+=1
print(f"[{m},{n}]共有{s}个质数")
```

【拓展训练】

(1) 在代码中,“s+=k>1”不能改为“s+=1”,试分析其中的原因。

提示:“s+=k>1”改为“s+=1”后,当 k=1 时会产生误判。

(2) 正奇数 k 为质数的充要条件是:k 在[3,t]无奇约数,其中 t 为 k 的算术平方根。

这是因为,如果 x 是 k 的约数,则 k÷x 也是 k 的约数。因此,如果能够找到 k 的较小的一个约数 x,则它的另一个约数就是 y=k÷x。由于 x≤y,故有 x×x≤k,即 x≤t。如果这个较小的 x 不存在,则 k 一定是质数。

习题三

一、单选题

1. 可以终止循环的语句是(　　)。

A. continue　　B. break　　C. else　　D. match

2. 下列关于循环 for 语句的说法中,错误的是(　　)。

A. 可以使用 break 终止 for 循环　　B. for 语句不能与 while 语句嵌套使用

C. for 语句可以遍历字符串　　D. for 语句可以遍历集合

3. else 子句不可以搭配的语句是(　　)。

A. match　　B. while　　C. if　　D. try

4. 设 a=70,则'C' if a<60 else ('B' if a<80 else 'A')的值是(　　)。

A. 'A'　　B. 'B'　　C. 'C'　　D. False

5. 语句 assert 0<=a<=9 的作用是(　　)。

A. 在 a>0 且 a<9 时触发异常　　B. 在 a>0 或 a<9 时触发异常

C. 在 a 是[0,9]的数值时触发异常　　D. 在 a 不是[0,9]的数值时触发异常

6. 在 match 语句中,模式 * _的作用是(　　)。

A. 匹配序列中任意数量的连续元素,并且会将它们绑定到变量

B. 匹配序列中的一个元素,并且会将它们绑定到变量

C. 匹配序列中的一个元素,不会将它们绑定到变量

D. 匹配序列中任意数量的连续元素,不会将它们绑定到变量

7. try 语句中,finally 子句的执行条件是(　　)。

A. try 语句中触发异常　　B. try 语句中未触发异常

C. 未使用 except 子句　　D. 无论是否触发异常,均会执行

8. list(range(5))的值是(　　)。

A. [1,2,3,4]　　B. [1,2,3,4,5]　　C. [0,1,2,3,4]　　D. [0,1,2,3,4,5]

9. 使用未定义的变量时,触发的异常是(　　)。

A. TypeError　　B. ValueError　　C. NameError　　D. IndexError

10. 在 match 语句中,多模式匹配使用的符号是(　　)。

A. |　　B. &　　C. or　　D. #

二、编程题

1. 输入两个正整数 m、n,输出满足 m+(m+1)+…+(m+k)>n 的最小整数 k。要求:

(1) m 是[10,200]的整数;n 是[1000,9999]的整数。

(2) 输入无效数据时重新输入。

(3) 程序执行效果:

```
输入两个正整数:123,4567
输出:k=32
```

2. 使用 Eratosthenes 筛法统计[3,n]的质数个数。n 为输入数据,是[100,10000]的整数。

提示:Eratosthenes 筛法执行流程如下。

(1) 把所有数由小到大排序,得到一个序列 a。设质数个数 s=0。

(2) 序列 a[0]是质数,s=s+1。

(3) 在序列 a 中删除能被 a[0]整除的所有数。

(4) 如果序列 a 为空,则结束,输出 s。

(5) 转至步骤(2)。

3. 如果正整数 k 是 k^2 的尾数,则称 k 是同构数。例如,$25^2=625$,所以 25 是同构数。输入两个正整数 m、n(m<n),输出[m,n]同构数的个数。如果输入数据无效,输出 0。

程序执行效果:

```
输入两个正整数(m,n):1,100
输出:[1,100]共有 5 个同构数。
```