第 3 章

程序基本流程控制

程序从主体上说都是顺序执行的，例如，前面章节中的程序都是按照语句的先后顺序依次执行的。但现实世界中的逻辑处理会更加复杂，因此在多数情况下，需要让程序在总体顺序执行的基础上，根据所要实现的功能选择执行一些语句而不执行另外一些语句，或者反复执行某些语句。程序设计时，通常有顺序结构、选择结构和循环结构三种基本结构。

本章学习编程中常用的选择结构和循环结构，从而实现较为复杂的程序逻辑。

3.1 选择结构语句

选择结构又称为分支结构，根据判断条件表达式是否成立(True 或 False)决定下一步选择执行特定的代码。

在 Python 语言中，条件语句使用关键字 if、elif、else 来表示，基本语法格式如下。

```
if 条件表达式 1:
    if 语句块 1
[elif 条件表达式 2:
    elif 语句块 2
else:
    else 语句块 3]
```

其中，冒号是语句块开始标记，方括号内为可选项。

在 Python 中，条件表达式的值只要不是 False、0(或 0.0、0j 等)、""、()、[]、{}、空值(None)、空对象，Python 解释器均认为其与 True 等价。也就是说，所有 Python 合法表达式(算术表达式、关系表达式、逻辑表达式等，包括单个常量、变量或函数)都可以作为条件表达式。

选择结构分为单分支结构、双分支结构、多分支结构、嵌套分支结构等多种形式。

3.1.1 单分支结构

单分支结构的语法格式如下。

```
if 条件表达式:
    语句块
```

功能：单分支结构中只有一个条件。如果条件表达式的值为 True，则表示条件满足，执行语句块；否则不执行语句块。一个语句块中可以包含多条语句。

Python 程序是依靠代码块的缩进体现代码之间的逻辑关系的。行尾的冒号表示缩进的开始，缩进结束就表示一个代码块结束了。整个 if 结构就是一个复合语句。

同一级别的语句块的缩进量必须相同。

【例 3-1】 输入一个人的身高和体重，然后根据 BMI 指数判断肥胖程度或健康程度。

实现代码如下。

```
height = float(input("请输入您的身高(单位为 m):"))
weight = float(input("请输入您的体重(单位为 kg):"))
bmi=weight/(height * height)  #用于计算 BMI 指数,公式为"体重(kg)/身高的平方(m²)"
print("您的 BMI 指数为:"+str(round(bmi,2)))     #输出 BMI 指数
#判断肥胖程度或健康程度
if bmi<18.5:
    print("您的体重过轻。")
if bmi>=18.5 and bmi<24.9:
    print("正常范围,注意保持。")
if bmi>=24.9 and bmi<29.9:
    print("您的体重过重。")
if bmi>=29.9:
    print("肥胖!")
```

运行结果如下。

```
请输入您的身高(单位为 m):1.7
请输入您的体重(单位为 kg):85
您的 BMI 指数为:29.41
您的体重过重。
```

3.1.2　双分支结构

双分支结构的语法格式如下。

```
if 条件表达式:
    语句块 1
else:
    语句块 2
```

功能：双分支结构可以表示两个条件。如果条件表达式的值为 True，则执行语句块 1；否则执行语句块 2。

【例 3-2】 询问你的年龄，如果年龄大于或等于 18 岁，输出“恭喜！你成年了。”，如果小于 18 岁，输出“要年满 18 岁才成年，你还差 x 岁。”。

实现代码如下。

```
age = int(input("你的年龄是:"))
if age >= 18:
    print("恭喜!你成年了。")
else:
    diff = str(18 - age)
    print("要年满 18 岁才成年,你还差 " + diff + " 岁。")
```

运行结果如下。

```
第一种情况:
你的年龄是:20
恭喜!你成年了。
第二种情况:
你的年龄是:15
要年满 18 岁才成年,你还差 3 岁。
```

注意：一个 if 语句最多只能有一个 else 子句，且 else 子句必须是整条语句的最后一个子句，else 没有条件。其中 else 不能单独使用，它必须与保留字 if 一起使用。

在程序中使用 if…else 语句时，如果出现 if 语句多于 else 语句的情况，那么该 else 语句将会根据缩进确定该 else 语句属于哪个 if 语句。

3.1.3 多分支结构

多分支结构的语法格式如下。

```
if 条件表达式 1:
    if 语句块 1
elif 条件表达式 2:
    elif 语句块 2
elif 条件表达式 3:
    elif 语句块 3
[else:
    else 语句块 4]
```

功能：使用 if…elif…else 语句时，条件表达式可以是一个单纯的布尔值或变量，也可以是比较表达式或逻辑表达式，如果条件表达式为真，执行语句；如果条件表达式为假，则跳过该语句，进行下一个 elif 的判断；只有在所有条件表达式都为假的情况下，才会执行 else 中的语句。

【例 3-3】 编写程序，判断工作年龄是否合法，我国的合法工作年龄是 18～60 岁，即如果年龄小于 18 岁的情况为童工，不合法；如果年龄为 18～60 岁的是合法工龄；大于 60 岁是法定退休年龄。

实现代码如下。

```
age = int(input('请输入您的年龄:'))
if age < 18:
    print(f'您的年龄是{age},童工一枚')
elif (age >= 18) and (age <= 60):
    print(f'您的年龄是{age},合法工龄')
elif age > 60:
    print(f'您的年龄是{age},可以退休')
```

运行结果如下。

```
请输入您的年龄:20
您的年龄是 20,合法工龄
```

需要注意的是,if 和 elif 都需要判断条件表达式的真假,而 else 不需要判断;另外,elif 和 else 都必须与 if 一起使用,不能单独使用。

3.1.4　嵌套分支结构

多分支结构也可以使用嵌套分支结构实现。语法格式如下。

```
if 条件表达式 1:
    语句块 1
    if 条件表达式:
        语句块 2
    else:
        语句块 3
else:
    if 条件表达式 4:
        语句块 4
```

上述格式中,外层的 if 块中嵌套了一个 if…else 结构,外层的 else 块中嵌套了一个 if 结构。

【例 3-4】 判断是否为酒后驾车。

国家质量监督检验检疫局发布的《车辆驾驶人员血液、呼气酒精含量阈值与检验》中规定:车辆驾驶人员血液中的酒精含量小于 20mg/100ml 不构成饮酒驾驶行为;酒精含量大于或等于 20mg/100m 而小于 80mg/100ml 为饮酒驾车;酒精含量大于或等于 80mg/100ml 为醉酒驾车。

要求使用嵌套的 if 语句,实现根据输入的酒精含量值判断是否为酒后驾车的功能。

实现代码如下。

```
degree = int(input("请输入每 100 毫升血液的酒精含量:"))
if degree <20:
```

```
        print("您还不构成饮酒驾驶行为,可以开车,请注意安全。")
    else:
        if degree <80:
            print("已经达到酒后驾驶标准,请不要开车。")
        else:
            print("已经达到醉酒驾驶标准,千万不要开车。")
```

运行结果如下。

```
请输入每 100 毫升血液的酒精含量:25
已经达到酒后驾驶标准,请不要开车。
```

注意：代码的逻辑级别是通过代码的缩进量控制的，同一级别的语句块的缩进量必须相同。

3.2 循环结构语句

循环结构是指满足一定条件的情况下，重复执行特定代码块的一种编码结构。其中，被重复执行的代码块称为循环体，判断是否继续执行的条件称为循环终止条件。

Python 中常见的循环语句是 while 语句和 for 语句两种格式。

while 循环与 for 循环的思路类似，区别在于 while 循环需要通过条件来实现逻辑控制，而不能像 for 循环那样直接读取序列对象。

3.2.1 while 循环

while 语句通过条件表达式建立循环。

while 循环可实现无限循环，即永远执行。无限循环的本质是死循环，仅在特定场景下使用，因此应该在循环中设计退出机制。

while 循环的语法格式如下。

```
while 条件表达式:
    循环体
```

当条件表达式的值为 True 时，执行循环体的语句，循环体中可以包含多条语句，这些语句都会被重复执行。while 语句中必须有改变循环条件的语句(也就是有把循环条件改变为 False 的代码)，否则会进入死循环。

【例 3-5】 模拟取款机密码输入。一般在取款机上取款时需要输入 6 位银行卡密码，接下来我们模拟一个简单的取款机(只有一位密码，默认密码为 0)，每次要求用户输入一位数字密码，并对如下 3 种情况进行判断：如果密码正确输出“密码输入正确，正进入系统!”；如果输入错误，输出“密码输入错误，您已经输错 * 次”；如果密码连续输入错误 6 次，输出“您的卡将被锁死，请与发卡行联系!”。

实现代码如下。

```
password=0
i = 1
while i < 7:
    num = input("请输入一位数字密码:")
    num = int(num)
    if num == password:
        print("密码输入正确,正进入系统!")
        i=7
    else:
        print("密码输入错误,您已经输错",i,"次")
    i += 1
if i == 7:
    print("您的卡将被锁死,请与发卡行联系!")
```

运行结果如下。

```
请输入一位数字密码:6
密码输入错误,您已经输错 1 次
请输入一位数字密码:2
密码输入错误,您已经输错 2 次
请输入一位数字密码:0
密码输入正确,正进入系统!
```

3.2.2　for 循环

for 循环是一个依次重复执行的循环。通常适用于枚举或遍历序列,以及要迭代对象中的元素。由于可迭代对象每次返回一个元素,因而适用于循环。Python 包括以下几种可迭代对象:①序列(sequence),例如字符串(str)、列表(list)、元组(tuple)等;②字典(dict);③文件对象;④迭代器对象(iterator);⑤生成器函数(generator)。

迭代器是一个对象,表示可迭代的数据集合,包括方法__iter__ ()和__next__(),可以实现迭代功能。生成器是一个函数,使用 yield 语句,每次产生一个值,也可以用于循环迭代。

for 循环的语法格式如下。

```
for 迭代变量 in 序列或迭代对象:
    循环体
```

其中,迭代变量用于保存读取出的值;对象为要遍历或迭代的对象,该对象可以是任意有序的序列对象,循环体为一组被重复执行的语句。

for 语句依次从序列或可迭代对象中取出一个元素并赋值给变量,然后执行循环体代码,直到序列或可迭代对象为空。

使用 for 语句处理列表时，程序会自动迭代列表对象，不需要定义和控制循环变量，代码更简洁。

【例 3-6】 计算 1＋2＋3＋4＋…＋100 的结果。

实现代码如下。

```
print("计算 1+2+3+4+…+100 的结果为:")
result=0
for i in range(1,101,1):
    result += i
print(result)
```

运行结果如下。

```
计算 1+2+3+4+…+100 的结果为:
5050
```

本例中的 range()函数属于 Python 内置的函数，返回一个可迭代对象，语法格式如下。

```
range(start, end, step)
```

参数说明如下。

- start 是指定计数的起始值，可以省略，若省略默认值为 0。
- end 为指定计数的结束值(但不含该值)，不可缺省。
- step 是指定计数的步长，即两个数之间的间隔，可以省略，若省略默认值为 1。

这个函数的功能是，产生以 start 为起点，以 end 为终点(不包括 end)，以 step 为步长的整型列表对象。这里的 3 个参数可以是正整数、负整数或者 0。

在使用 range()函数时，如果只有一个参数，那么表示指定的是 end；如果有两个参数，则表示指定的是 start 和 end；当 3 个参数都存在时，最后一个参数才表示步长。

3.2.3 循环嵌套

在 Python 中，允许在一个循环体中嵌入另一个循环，这称为循环嵌套。

在 Python 中，for 循环和 while 循环都可以进行循环嵌套。

【例 3-7】 使用循环嵌套实现打印九九乘法表。

实现代码如下。

```
for i in range(1, 10):                          #输出 9 行
    for j in range(1, i + 1):                   #输出与行数相等的列
        print(str(j) + "×" + str(i) + "=" + str(i * j) + "\t", end='')
    print('')                                   #换行
```

运行结果如下。

```
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
```

上述示例使用了双层 for 循环，第 1 个循环可以看成是对乘法表行数的控制，同时也是每一个乘法公式的第 2 个因数；第 2 个循环控制乘法表的列数，列数的最大值应该等于行数，因此第 2 个循环的条件应该是在第 1 个循环的基础上建立的。

在循环嵌套中，如果外层循环执行 n 次，内层循环执行 m 次，则整个循环需要执行 n×m 次。

【例 3-8】 猜数字游戏。每一个数字可以连续猜 6 次，每人可以连续猜 3 个数字。

实现代码如下。

```
from random import randint
for i in range(4):                          #控制竞猜的轮次
    print('*** 猜第{0}个数 ***'.format(i+1))
    x = randint(0,100)
    for j in range(6):                      #控制一轮的竞猜次数
        guess = int(input("请输入 1 到 100 的数: "))
        if guess == x:
            print('恭喜你,猜对了!')
            break
        elif guess > x:
            print('很遗憾,太大了!')
        else:
            print('很遗憾,太小了!')
    print('第{0}次竞猜结束'.format(i+1))
```

运行结果如下。

```
*** 猜第 1 个数 ***
请输入 1 到 100 的数: 23
很遗憾,太小了!
请输入 1 到 100 的数: 87
很遗憾,太小了!
```

```
请输入1到100的数：99
恭喜你,猜对了!
第1次竞猜结束
*** 猜第2个数 ***
请输入1到100的数:
```

3.3 break、continue与else语句

在循环结构中,还可以使用break、continue和else等语句控制循环过程或处理循环结束后的工作。

在循环过程中,有时可能需要提前跳出循环,或者跳过本次循环的剩余语句以提前进行下一轮循环,在这种情况下,可以在循环体中使用break语句或continue语句。如果存在多重循环,则break语句只能跳出其所属的那层循环。break语句和continue语句通常与if语句配合使用。

1. break语句

break语句可以终止当前的循环,包括while和for在内的所有控制语句。

break语句的语法比较简单,只需要在相应的while或for语句中加入即可。break语句一般与if语句搭配使用,表示在某种条件下,跳出循环。如果使用嵌套循环,break语句将跳出最内层的循环。

【例3-9】 输入一个整数,判断是否为素数。素数是只能被1和自身整除的数,例如对于9,要判断9能否被2～8的数整除：如果能,说明不是素数;如果都不能,说明是素数。

实现代码如下。

```
number = int(input("请输入整数:"))     #9:2~8
if number < 2:
    print("不是素数")
else:
    for i in range(2, number):
        if number % i == 0:
            print("不是素数")
            break                       #如果有结论了,就不需要再与后面的数字计算了
    else:
        print("是素数")
```

运行结果如下。

```
请输入整数:9
不是素数
```

2. continue 语句

continue 语句的作用没有 break 语句那么强大，它只能终止本次循环而提前进入到下一次循环中。

continue 语句的语法比较简单，只需要在相应的 while 或 for 语句中加入即可。continue 语句一般与 if 语句搭配使用，表示在某种条件下，跳过当前循环的剩余语句，然后继续进行下一轮循环。如果使用嵌套循环，continue 语句将只跳过最内层循环中的剩余语句。

【例 3-10】 设计一个验证用户密码程序，用户只有三次输入机会，不过如果用户输入的内容中包含“*”，则不计算在内。

实现代码如下。

```
count = 3
password = '123'

while count:
    passwd = input('请输入密码:')
    if passwd == password:
        print('密码正确,进入程序......')
        break
    elif '*' in passwd:
        print('密码中不能含有"*"号!您还有',count,'次机会!',end=' ')
        continue
    else:
        print('密码输入错误!您还有',count-1,'次机会!',end=' ')
    count -= 1
```

运行结果如下。

```
请输入密码:666
密码输入错误!您还有 2 次机会! 请输入密码:125
密码输入错误!您还有 1 次机会! 请输入密码:123
密码正确,进入程序......
```

3. else 语句

while 语句和 for 语句的后边还可以带有 else 语句，用于处理循环结束后的“收尾”工作。

else 语句的语法格式如下。

格式 1：

```
while 条件表达式:
    循环体
```

```
else:
    else子句代码块
```

格式2：

```
for 迭代变量 in 序列或迭代对象:
    循环体
else:
    else子句代码块
```

else子句是可选的。如果有else子句，则当循环因为条件表达式不成立或序列遍历完毕而自然结束时，就会执行else子句的代码。如果有else子句，但循环因为执行了break语句而使得循环提前结束的，则不会执行else子句的代码。

【例3-11】 编写程序，随机产生骰子的一面（数字1～6），给用户三次猜测机会，程序给出猜测提示（偏大或偏小）。如果某次猜测正确，则提示正确并中断循环；如果三次均猜错，则提示机会用完。

分析：使用随机函数产生随机整数，设置循环初值为1，循环次数为3，在循环体中输入猜测并进行判断，如果密码正确则使用break语句中断当前循环。

实现代码如下。

```
import random
point=random.randint(1,6)
count=1
while count<=3:
    guess=int(input("请输入您的猜测:"))
    if guess>point:
        print("您的猜测偏大。")
    elif guess<point:
        print("您的猜测偏小。")
    else:
        print("恭喜您猜对了!")
        break
    count=count+1
else:
    print("很遗憾,三次全猜错了!")
```

运行结果如下。

```
请输入您的猜测:23
您的猜测偏大。
请输入您的猜测:1
您的猜测偏小。
```

```
请输入您的猜测:3
您的猜测偏小。
很遗憾,三次全猜错了!
```

3.4　pass 语 句

在 Python 中还有一个 pass 语句,表示空语句,它将不做任何事情,一般起到占位作用,用于保持程序结构的完整性。有时候程序需要占一个位置,放置一条语句,但又不希望这条语句做任何事情,此时就可以通过 pass 语句来实现。使用 pass 语句比使用注释更加优雅。

【例 3-12】 应用 for 循环输出 1～20 之间(不包括 20)的偶数,当不是偶数时,使用 pass 语句占个位置,方便以后对不是偶数的数进行处理。

实现代码如下。

```
for i in range(1, 20):
    if i % 2 == 0 :
        print(i,end=' ')
    else:
        pass
```

运行结果如下。

```
2 4 6 8 10 12 14 16 18
```

3.5　程序的错误与异常处理

3.5.1　程序的错误与处理

Python 程序的错误通常可以分为三种类型,即语法错误、运行时错误和逻辑错误。

1. 语法错误

Python 程序的语法错误是指其源代码中拼写语法错误,这些错误导致 Python 编译器无法把 Python 源代码转换为字节码,故又称为编译错误。程序中包含语法错误时,编译器将显示 SyntaxError 错误信息。

通过分析编译器抛出的运行时错误信息,仔细分析相关位置的代码,可以定位并修改程序错误。

2. 运行时错误

Python 程序的运行时错误是在解释执行过程中产生的错误。例如,如果程序中没有导入相关的模块(例如,import random)时,解释器将在运行时抛出 NameError 错误信

息;如果程序中包括零除运算,解释器将在运行时抛出 ZeroDivisionError 错误信息;如果程序中试图打开不存在的文件,解释器将在运行时抛出 FileNotFoundError 错误信息。

同样,通过分析解释器抛出的运行时错误信息,仔细分析相关位置的代码,可以定位并修改程序错误。

3. 逻辑错误

Python 程序的逻辑错误可以使程序执行(程序运行本身不报错),但运行结果不正确。对于逻辑错误,Python 解释器无能为力,需要编程人员根据结果来判断和修改。

3.5.2 程序的异常与处理

Python 语言采用结构化的异常处理机制。

在程序运行过程中,如果出现错误,Python 解释器会创建一个异常对象,并抛给系统运行时(runtime)处理,即程序终止正常执行流程,转而执行异常处理流程。

在某种特殊条件下,代码中也可以创建一个异常对象,并通过 raise 语句,抛给系统运行时处理。异常对象是异常类的对象实例。Python 异常类均派生于 BaseException。常见的异常包括 NameError、SyntaxError、AttributeError、TypeError、ValueError、ZeroDivisionError、IndexErroror、KeyError 等。在应用程序开发过程中,有时候需要定义特定于应用程序的异常类,表示应用程序的一些错误类型。

当程序中异常引发后,Python 虚拟机通过调用堆栈查找相应的异常捕获程序。通过 try 语句来定义代码块,以运行可能抛出异常的代码;通过 except 语句,可以捕获特定的异常并执行相应的处理;通过 finally 语句,可以保证即使产生异常(处理失败),也可以在事后清理资源等。

try…except…else…finally 语法格式如下。

```
try:
    可能产生异常的语句
except Exception1:                          #捕获异常 Exception1
    发生异常时执行的语句
except (Exception2, Exception3):            #捕获异常 Exception2、Exception3
    发生异常时执行的语句
except Exception4 as e:                     #捕获异常 Exception4,实例为 e
    发生异常时执行的语句
except:                                     #捕获其他所有异常
    发生异常时执行的语句
else:                                       #无异常
    无异常时执行的语句
finally:                                    #不管发生异常与否,保证执行
    不管发生异常与否,保证执行的语句
```

【例 3-13】 通过两个数相除,演示 try…except…else…finally 的应用示例。

实现代码如下。

```
try:
    num1 = int(input('请输第一个数字:'))
    num2 = int(input('请输入第二个数:'))
    if num1 <= 10:
        raise Exception('输入的值太小了,有可能不够除!')
    result=num1/num2
    print('计算结果:', result)
except ZeroDivisionError:
    print('出错了,除数不能为零!!!')
except ValueError as e:
    print('输入错误,只能是整数:', e)
else:
    print('计算完成...')
finally:
    print('程序运行结束。')
```

运行结果如下。

```
请输第一个数字:15
请输入第二个数:0
出错了,除数不能为零!!!
程序运行结束
```

由上述例子可以看出,在 Python 中,提供了 try…except 语句捕获并处理异常。在使用时,把可能产生异常的代码放在 try 语句块中,把处理结果放在 except 语句块中,这样当 try 语句块中的代码出现错误时,就会执行 except 语句块中的代码,如果 try 语句块中的代码没有错误,那么 except 语句块将不会被执行。

本章小结

本章主要介绍了程序三种基本结构以及辅助控制语句等内容,主要涉及如下知识点。

(1) 程序从主体上说都是顺序的,一条语句执行完之后会自动执行下一条语句。但在很多情况下,还需要在总体顺序执行的基础上,根据程序要实现的功能选择一些语句执行或者反复执行某些语句,此时需要使用选择结构或循环结构。程序设计时,通常有顺序结构、选择结构和循环结构三种基本结构。

(2) 选择结构使用 if 语句,根据条件表达式是否成立决定下一步的执行语句。

(3) 循环结构使用 while 语句或 for 语句,在一定条件下重复执行某段程序。while 语句通过条件表达式建立循环;当条件表达式的值为 True 时执行循环体语句。for 语句通过遍历序列或可迭代对象建立循环。

(4) 在循环结构中,使用 break 语句可以跳出其所属层次的循环体;使用 continue 语句可以跳过本次循环的剩余语句,然后继续进行下一轮循环。

(5) 循环结构的最后可以带有 else 语句,用来处理循环结束后的工作。如果是因为执行了 break 语句而提前结束循环,则不会执行 else 语句。

(6) 选择结构和循环结构可以嵌套使用。Python 语言通过缩进体现代码的逻辑关系,同一个语句块必须保证相同的缩进量。

(7) Python 程序的错误通常可以分为三种类型,即语法错误、运行时错误和逻辑错误。

(8) Python 语言采用结构化的异常处理机制。当程序中引发异常后,Python 虚拟机通过调用堆栈查找相应的异常捕获程序。通过 try 语句来定义代码块,以运行可能抛出异常的代码;通过 except 语句,可以捕获特定的异常并执行相应的处理;通过 finally 语句,可以保证即使产生异常(处理失败),也可以在事后清理资源等。

思考与练习

1. 请简述 for 循环和 while 循环的执行过程。

2. 请简述跳转语句 break 和 continue 的区别是什么?

3. 请简述 pass 语句的作用。

4. 什么叫异常? 简述 Python 的异常处理机制。

5. 编写公交车票计票。

购买公交车票的规定如下:

乘 1～3 站,1 元/位;乘 4～8 站,2 元/位;乘 8 站以上,3 元/位。

输入乘坐人数(pernum)和乘坐站数(stanum),计算购买公交车票需要的总金额,并将计算结果输出。

注意:如果乘坐人数和乘坐站数为 0 或负数,输出 error。

6. 编程实现判断奇偶数。

输入一个整数,判断它是奇数还是偶数。如果是奇数,输出 odd;如果是偶数,输出 even。

7. 编程实现提取字符串中的数字组成整数。

输入一个字符串,将这个字符串中的所有数字字符('0'...'9')提取出来,将其转换为一个整数,并将转换后的整数输出。

8. 编程实现寻找 100～999 内的水仙花数。

水仙花数(Narcissistic number)又称为超完全数字不变数,水仙花数是指一个 3 位数,它的每个位上的数字的 3 次幂之和等于它本身(例如,$1^3+5^3+3^3=153$)。找出所有的水仙花数。

提示:三位数在 100～999 内,关键是将其个位数字、十位数字和百位数字解出来。

9. 编程实现判断是否通过科目一考试。

在进行机动车驾驶人考试时,首先进行的是科目一考试。该科目考试为上机答题,满分 100 分,通过分数为 90 分。请编写程序,判断考生是否通过考试。要求考生输入自己的分数,系统进行判断,如果分数大于或等于 90 分,则提示“考试合格!”;否则提示“考试

不合格”。

10. 编程实现求三角形的周长并判断其是何种三角形。

在数学中三角形是由同一平面内不在同一直线上的三条线段首尾顺次连接所组成的封闭图形。如果设置三条线段(也称三条边)分别为 a、b、c,那么它将有以下规律:

- 三条边的关系为 a+b>c、a+c>b 且 b+c>a。
- 周长(p)就是三条线段的和,即 p=a+b+c。
- 构成等边三角形的条件为 a=b=c。
- 构成等腰三角形的条件为 a=b、a=c 或 b=c。
- 构成直角三角形的条件为 $a^2+b^2=c^2$、$a^2+c^2=b^2$ 或 $b^2+c^2=a^2$。

请根据以上规律编写一个 Python 程序,实现以下功能:

输入三角形的三条边长,求三角形的周长;若不能构成三角形,则输出提示。

根据用户输入的三角形的三条边长判定是何种三角形(一般三角形、正三角形、等腰三角形、直角三角形)。