1 条件语句

1.1 if语句

```
if expression:
  expr_true_suite
```

1. if语句的 `expr_true_suite` 代码块只有当条件表达式 `expression` 结果为真时才执行，否则将继续执行紧跟在该代码块后面的语句。
2. 单个if语句中的 `expression` 条件表达式可以通过布尔操作符 `and` ， `or` 和 `not` 实现多重条件判断

【例子】

```
if 2 > 1 and not 2 > 3:
  print('Correct Judgement!')

# Correct Judgement!
```

1.2 if-else语句

```
if expression:
  expr_true_suite
else
  expr_false_suite
```

Python提供与if搭配使用的else，如果if语句的条件表达式结果布尔值为假，那么程序将执行else语句后的代码。

【例子】

```
temp = input("猜□猜□姐姐想的是哪个数字？")
guess = int(temp) # input 函数将接收的任何数据类型都默认为 str。
if guess == 666:
  print("你太了解□姐姐的□思了！")
  print("哼，猜对也没有奖励！")
else:
  print("猜错了，□姐姐现在□□想的是666！")
print("游戏结束，不玩□啦！")
```

if语句支持嵌套，即在一个if语句中嵌入另一个if语句，从而构成不同层次的选择结构。

Python使用**缩进**而不是大括号来标记代码块边界，因此要特别注意else的悬挂问题。

1.3 if-elif-else语句

elif语句即为else if，用来检查多个表达式是否为真，并在为真时执行特定代码块中的代码。

【例子】

```
temp = input('请输□成绩:')
source = int(temp)
if 100 >= source >= 90:
  print('A')
elif 90 > source >= 80:
  print('B')
elif 80 > source >= 60:
  print('C')
elif 60 > source >= 0:
  print('D')
else:
  print('输□错误！')
```

1.4 assert关键词

当关键词 `assert` （断言）后面的条件为False时，程序自动崩溃并抛出 `Assertion Error` 的异常。

【例子】

```
my_list = ['lsgogroup']
my_list.pop(0)
assert len(my_list) > 0

# AssertionError
```

在进行单元测试时，可以用来在程序中置入检查点，只有条件为True才能让程序正常工作。

2 循环语句

2.1 while循环

```
while 布尔表达式:
  代码块
```

布尔表达式为真或非零非空时执行代码块，假或0或空时不执行。

2.2 while-else循环

```
while 布尔表达式:
  代码块
else:
  代码块
```

当while循环正常执行完的情况下，执行else输出，如果while循环中执行了跳出循环的语句，比如说break，将不执行else代码块的内容。

【例1】

```
count = 0
while count < 5:
  print("%d is less than 5" % count)
  count = count + 1
else:
  print("%d is not less than 5" % count)

# 0 is less than 5
# 1 is less than 5
# 2 is less than 5
# 3 is less than 5
# 4 is less than 5
# 5 is not less than 5
```

【例2】

```
count = 0
while count < 5:
  print("%d is count = 6 less than 5" % count)
  break
else:
  print("%d is not less than 5" % count)

# 0 is less than 5
```

2.3 for循环

for循环是迭代循环，在Python中相当于一个通用的序列迭代器，可以遍历任何有序

```
for 迭代变量 in 可迭代对象:
  代码块
```

每次循环，迭代变量被设置为可迭代对象的当前元素，提供给代码块使用。

【例子】

```
for i in 'ILoveLSGO':
  print(i, end=' ') # 不换□输出

# I L o v e L S G O
```

2.4 for-else循环

当for循环正常执行完的情况下，执行else输出，如果for循环中执行了跳出循环的语句，比如说break，将不执行else代码块的内容，和while-else一样。

2.5 range() 函数

```
range([start,] stop[, step=1])
```

1. 这个BIF（Built-in function）有三个参数，其中用中括号括起来的两个表示这两个参数是可选的。
2. step = 1 表示第三个参数的默认值是1。
3. range() 这个BIF的作用是生成一个从start参数的值开始到stop参数到值结束的数字序列，该序列包含start的值但不包含stop

的值。（左闭右开）

【例1】

```
for i in range(2, 9): # 不包含9
  print(i)

# 2
# 3
# 4
# 5
# 6
# 7
# 8
```

【例2】

```
for i in range(1, 10, 2):
  print(i)

# 1
# 3
# 5
# 7
# 9
```

2.6 enumerate()函数

```
enumerate(sequence, [start=0])
```

1. sequence – 一个序列、迭代器或其他支持迭代对象。
2. start – 下标起始位置
3. 返回enumerate(枚举)对象

【例1】

```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
lst = list(enumerate(seasons))
print(lst)
# [(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
lst = list(enumerate(seasons, start=1)) # 下标从 1 开始
print(lst)
# [(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

【例2】enumerate() 与for循环的结合使用

```
for i, a in enumerate(A):
  do something with a
```

用enumerate(A) 不仅返回了A中的元素，还顺便给该元素一个索引值（默认从0开始）。此外，用enumerate(A, j)还可以确定索引起始值为j。

```
languages = ['Python', 'R', 'Matlab', 'C++']
for language in languages:
  print('I love', language)
print('Done!')

# I love Python
# I love R
# I love Matlab
# I love C++
# Done!

for i, language in enumerate(languages, 2):
  print(i, 'I love', language)
print('Done!')

# 2 I love Python
# 3 I love R
# 4 I love Matlab
# 5 I love C++
# Done!
```

2.7 break语句

break语句可以跳出当前所在层的循环。

2.8 continue语句

continue语句终止本轮循环并开始下一轮循环。

2.9 pass语句

pass语句的意思就是“不做任何事”，如果你在需要有语句的地方不写任何语句，那么解释器会提示出错，而pass语句就是用来解决这些问题的。

pass是空语句，不做任何操作，只起到占位的作用，目的是为了保持程序结构的完整性。

3 推导式

3.1 列表推导式

```
[ expr for value in collection [if condition] ]
```

【例子】

```
x = [-4, -2, 0, 2, 4]
y = [a * 2 for a in x]
print(y)
# [-8, -4, 0, 4, 8]

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

a = [(i, j) for i in range(0, 3) if i < 1 for j in range(0, 3) if j > 1]
print(a)
# [(0, 2)]
```

3.2 元组推导式

```
( expr for value in collection [if condition] )
```

【例子】

```
a = (x for x in range(10))
print(a)

# <generator object <genexpr> at 0x0000025BE511CC48>

print(tuple(a))

# (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

3.3 字典推导式

```
{ key_expr: value_expr for value in collection [if condition] }
```

【例子】

```
b = {i: i % 2 == 0 for i in range(10) if i % 3 == 0}
print(b)
# {0: True, 3: False, 6: True, 9: False}
```

3.4 集合推导式

```
{ expr for value in collection [if condition] }
```

```
c = {i for i in [1, 2, 3, 4, 5, 5, 6, 4, 3, 2, 1]}
print(c)
# {1, 2, 3, 4, 5, 6}
```

4 练习题

1、编写一个Python程序来查找那些既可以被7整除又可以被5整除的数字，介于1500和2700之间。

```
for i in range(1500, 2701):
  if i % 7 == 0 and i % 5 == 0:
    print(i)
```

2、龟兔赛跑游戏

题目描述：

话说这个世界上有各种各样的兔子和乌龟，但是研究发现，所有的兔子和乌龟都有一个共同的特点——喜欢赛跑。于是世界上各个角落都不断在发生着乌龟和兔子的比赛，小华对此很感兴趣，于是决定研究不同兔 子和乌龟的赛跑。他发现，兔子虽然跑比乌龟快，但它们有众所周知的毛病——骄傲且懒惰，于是在与乌龟的比赛中，一旦任一秒结束后兔子发现自己领先t米或以 上，它们就会停下来休息s秒。对于不同的兔子，t，s的数值是不同的，但是所有的乌龟却是一致——它们不到终点决不停止。

然而有些比赛相当漫长，全程观看会耗费大量时间，而小华发现只要在每场比赛开始后记录下兔子和乌龟的数据——兔子的速度v1（表示每秒兔子能跑v1 米），乌龟的速度v2，以及兔子对应的t，s值，以及赛道的长度l——就能预测出比赛的结果。但是小华很懒，不想通过手工计算推测出比赛的结果，于是他找 到了你——清华大学计算机系的高才生——请求帮助，请你写一个程序，对于输入的一场比赛的数据v1，v2，t，s，l，预测该场比赛的结果。

输入:

输入只有一行，包含用空格隔开的五个正整数v1，v2，t，s，l，其中(v1,v2<=100;t<=300;s<=10;l<=10000且为v1,v2的公倍数)

输出:

输出包含两行，第一行输出比赛结果——一个大写字母“T”或“R”或“D”，分别表示乌龟获胜，兔子获胜，或者两者同时到达终点。

第二行输出一个正整数，表示获胜者（或者双方同时）到达终点所耗费的时间（秒数）。

样例输入：

10 5 5 2 20

样例输出

D
4

```
v1,v2,t,s,l = map(int, input().split())
if v1<=100 and v2<=100 and t<=300 and s<=10 and l<=10000 and l%v1==0 and l%v2==0:
    s1,s2,i = 0,0,0
    while s1<l and s2<l:
        s1,s2,i=v1+s1,v2+s2,i+1
        if s1==l or s2==l:
            break
        elif s1-s2>=t:
            s2,i=s2+v2*s,i+s
    if s1>s2:
      print('R')
    if s1==s2:
      print('D')
    if s1<s2:
      print('T')
    print(i)
```