sem4

第4次习题课讲义¶

本次讲解作业2的难点，复习分支相关理解误区，讲解循环例题，并现场进行作业3的试做和答疑。

分支复习¶

基本语法：关键字¶

Python 使用 `if`、`elif` 和 `else` 三个关键字来进行条件判断。

本文中我们称判断一个互斥条件的部分构成一个基本的**条件分支语句块**，比如接下来这个示例：

In [1]:

```
x = 10
if x > 0:
    print("x   ")
elif x == 0:
    print("x  ")
else:
    print("x   ")
```

```
x
```

当然，如果将elif视为else和if的耦合，那它也似乎成为了两个基本语句块的嵌套，但是这两种在逻辑上是等价

In [2]:

```
#
x = 10
if x > 0:
    print("x   ")
else:
    if x == 0:
        print("x  ")
    else:
        print("x   ")
```

```
x
```

一个独立/互斥的条件判断写在同一个语句块里，减少对自己的误导。上述不推荐版本中就有两层缩进，就将其

判断条件：bool值¶

if分支的目标是判断一个条件是否成立，本质是判断语句中的**bool值**的真假（等同于中学数学中命题的真假）

使用以下基本运算符之后，一个表达式的值就成为bool值：

```
==  #
!=  #
>   #
<   #
>=  #
<=  #
```

In [3]:

```
a = 1
print("a == 1:", a == 1)
print("a != 2:", a != 2)
print("a > 2:", a > 2)
b = a == 1
print("b = a == 1:", b)
```

```
a == 1: True
a != 2: True
a > 2: False
b = a == 1: True
```

Python 里，以下值会被视为 False：

```
False, None, 0, 0.0, "", [], {}, set(), range(0)
```

In [4]:

```
print(False)
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(range(0)))
```

```
False
False
False
False
False
```

python也支持逻辑学中的与（^）、或（∨）、非（¬）运算，分别使用and、or和not。

In [5]:

```
a = 1
b = a == 1
c = bool(0)
print("b:", b)
print("c:", c)
print("b and c:", b and c)
print("b or c:", b or c)
print("not c:", not c)
```

```
b: True
c: False
b and c: False
b or c: True
not c: True
```

分支的并列和嵌套¶

值得注意的是，在 同一个 `if-elif-else` 语句块中，只会执行**第一个满足条件**的分支，其余分支将被跳过。因适用于 **多个互斥条件** 的情况，而 不能用于**同时满足多个独立条件** 的判断。

比如这个例子，10不可能同时为正数、零和负数，所以称为互斥的。这种情况就适合用**一个分支语句块**来做：

In [6]:

```
x = 10
if x > 0:
    print("x   ")
elif x == 0:
    print("x  ")
else:
    print("x   ")
```

```
x
```

如下是一个错误示例（仅执行第一个满足条件的分支）：

In [7]:

```
#
x = 10
y = 5

if x > 0:
    print("x   ")
elif y > 0:
    print("y   ")
```

```
x
```

尽管 `y > 0` 也是 True，但由于 `x > 0` 先被满足，程序不会执行 `elif y > 0` 这一分支。

如果需要对 **多个独立的条件**（一件事情的不同角度） 进行判断，可以使用以下几种方法：

1. 并列多个 `if` 语句¶ 适用于多个独立条件都需要执行的情况。

x > 0 和 y > 0 都是 独立事件，都需要执行。

In [8]:

```
x = 10
y = 5

if x > 0:
    print("x   ")
else:
    print("x   ")
if y > 0:
    print("y   ")
else:
    print("y   ")
```

```
x
y
```

2. 嵌套 `if` 语句¶ 适用于在一个条件满足的基础上再细分的情况

需要 **在一个条件成立的基础上** 继续细分判断。在有多个独立条件的情况下，它实际上改变了条件之间的优先关

比如以下示例中，x就被视为了一个更重要的独立变量，如果x不是正数，接下来的判断就不做了：

In [9]:

```
x = 10
y = -5

if x > 0:
    print("x   ")
    if y > 0:
        print("y   ")
```

```
x
```

如果需要建立和原先等同的逻辑，需要重复低优先级的条件：

In [10]:

```
x = 10
y = -5
```

```
if x > 0:
    print("x   ")
    if y > 0:
        print("y   ")
    else:
        print("y   ")
else:
    print("x   ")
    if y > 0:
        print("y   ")
    else:
        print("y    ")
```

```
x
y
```

3. bool值穷举¶ 适用场景：当不同变量的组合情况有限且复杂时，可以用bool值穷举 来明确所有可能的情况。

In [11]:

```
x = 10
y = -5

if x > 0 and y > 0:
    print("x   y   ")
elif x > 0 and y <= 0:
    print("x   y   ")
elif x <= 0 and y > 0:
    print("x    y   ")
else:
    print("x   y    ")
```

```
x   y
```

理论上，这个方法是一定可行的，它等同于给一个命题的组合绘制真值表，每一个条件都是独立的。

x > 0	y > 0	结果
True	True	x 和 y 都是正数
True	False	x 是正数，y 不是正数
False	True	x 不是正数，y 是正数
False	False	x 和 y 都不是正数

综合题讲解：作业2第五题¶

这道题综合运用了独立条件的并列、组合，以及bool值的特性。

描述¶

在小说《沙丘》的世界中，”李桑–阿尔·盖布”(Lisan al Gaib)是弗雷曼人预言中的救世主。假如为了验证
阿尔·盖布”，需要通过三个测试:

1. 沙丘测试:在沙丘上行走10公里,如果速度超过15公里/小时（>15）,则通过测试；
2. 骑行测试:驾驭一只成年沙虫,如果骑行时间超过30分钟（>30）,则通过测试；
3. 毒水测试:饮用一剂”生命之水”,如果昏迷时间在5h内,则通过测试，超过5h（>5）或死亡则不通过。

请你编写一个程序,根据一个人在这三个测试中的表现,判断他是否有可能是”李桑–
阿尔·盖布”。

我们注意到，这三个条件实际上都是**独立**而非互斥的。对应到前文的讲述，我们至少不能只用一个if分支语句块

```
if speed > 15:
    #  1
if ride > 30:
    #  2
if 0 <= water <= 5:
    #  3
```

输入¶

三行,每行包含一个整数,分别表示

1. 在沙丘测试中的速度(公里/小时)
2. 骑行测试的持续时间(分钟)
3. 毒水测试的昏迷时间(小时),如果昏迷时间为–1表示死亡。

输入是OJ对你的品质保证，它说是整数就一定是整数。输入部分的代码很简单：

```
speed = int(input())
ride = int(input())
water = int(input())
```

输出¶

一行字符串,表示对这个人是否是”李桑–阿尔·盖布”的判断：

- 如果服用“生命之水”后**死亡**，一定不是”李桑–阿尔·盖布”，输出“NO”；
- 如果**三个**测试都通过，则他很有可能是”李桑–阿尔·盖布”，输出”YES“；
- 如果只通过了**两个**测试，则他有一定可能是”李桑–阿尔·盖布”，但还需要更多的证据，输出“MAYB
- 如果只通过了**一个或零个**测试，则他不太可能是”李桑–
  阿尔·盖布”，输出“NO”。

本题的解题要点是，我们发现虽然有四个命题，但其实是两个独立的判断，**死亡与否**，和**通过测试的数量**；同

前面的讲解中提到，有三种方法。有优先级的条件，比较典型的可以使用分支嵌套。第一层分支判断高优条件

我们假定服用生命之水之后的生存时间是`water`，通过测试的数量是`count`，输出部分的代码使用嵌套方式就可

```
if water == -1:
    print("NO")
else:
    if count == 3:
        print("YES")
    elif count == 2:
        print("MAYBE")
    else:
        print("NO")
```

或者

```
if water == -1:
    print("NO")
elif count == 3:
    print("YES")
elif count == 2:
    print("MAYBE")
else:
    print("NO")
```

如果要写成并列分支：

```
if water == -1:
    print("NO")
if count == 3:
    print("YES")
elif count == 2:
    print("MAYBE")
else:
    print("NO")
```

这样的写法可能会在输出NO之后再输出一个根据count产生的结果。两种解决办法：

```
if water == -1:
    print("NO")
    exit(0)
if count == 3:
    print("YES")
elif count == 2:
    print("MAYBE")
else:
    print("NO")
```

或者通过改变count复用NO分支：

```
if water == -1:
    count = 0
if count == 3:
    print("YES")
```

```
elif count == 2:
    print("MAYBE")
else:
    print("NO")
```

推荐掌握上述两种常用写法就足够了。

这里补充一个使用bool值穷举的一个示例（感谢2300017463，2100017436_刘子涵、2200017747(Echo)也使

In [12]:

```
# 2300017463
# s=int(input())
# t=int(input())
# r=int(input())
s = 30
t = 30
r = 0
if r==-1:
    print("NO")
elif s>15 and t>30 and r<=5:
    print("YES")
elif s>15 and t>30 and r>5:
    print("MAYBE")
elif s>15 and t<=30 and r<=5:
    print("MAYBE")
elif s<=15 and t>30 and r<=5:
    print("MAYBE")
else:
    print("NO")
```

```
MAYBE
```

保存bool值的优化：

In [13]:

```
# s=int(input())
# t=int(input())
# r=int(input())
s = 30
t = 30
r = 0

if r==-1:
    print("NO")
else:
    a = s>15
    b = t>30
    c = r<=5
```

```
    if a and b and c:
        print("YES")
    elif a and b or b and c or c and a:
        print("MAYBE")
    else:
        print("NO")
```

```
MAYBE
```

如何计数：bool值应用¶ 输入输出的部分解决了，现在还有一个关键的条件，就是通过的测试数量还没有求解

原先分析出来的框架：

```
if speed > 15:
    #  1
if ride > 30:
    #  2
if 0 <= water <= 5:
    #  3
```

方法1：直接计数

几个并列的分支，每个都独立计数，这样就能求出通过了几个测试。

```
count = 0
if speed > 15:
    count += 1
if ride > 30:
    count += 1
if 0 <= water <= 5:
    count +=1
```

方法2：加和

上面的方法，没有运用到bool值True和int值1之间的转换关系，比如我们可以看到True可以等效为1，实现计数

回顾一下bool值的性质：

In [14]:

```
a = 1
b = a > 0
print(b)
print(int(True))
print(int(False))
```

```
True
1
0
```

In [15]:

```
#  2300014564
# a = int(input())
# b = int(input())
# c = int(input())
a = 30
b = 30
c = 0
if c == -1:
    print("NO")
else:
    tests = (a > 15) + (b > 30) + (c < 5)
    if tests == 3:
        print("YES")
    elif tests == 2:
        print("MAYBE")
    else:
        print("NO")
```

```
MAYBE
```

作业2提示¶

由于作业2还有一周截止，只能完整放出第五题的答案，前四题在本部分给予完整的思路讲解。

如何上手做一道简单题？输入，操作，输出。

这其实和用户增长的流程一样，前期用户分析，中期执行，后期复盘优化:D

我们接下来实践一下这个思路。

1 是否赴约¶

描述¶ 晶晶的朋友贝贝约晶晶下周一起去看电影，但晶晶每周的1,3,5有课必须上课，请帮晶晶判断她能否接受
YES；如果不能则输出 NO。

输入¶ 输入有一行，贝贝邀请晶晶去看展览的日期，用数字1到7表示从星期一到星期日。

输出¶ 输出有一行，如果晶晶可以接受贝贝的邀请，输出YES，否则，输出NO。注意YES和NO都是大写字母

分析¶ 首先确定，只有一个独立条件，且条件各个选项都是互斥的，只使用一个块能解决。

- 输入：一个整数
- 三个条件：当天等于1，等于3，**或者**等于5。可以用三个bool值or连接，也可以全部elif。
- 输出：符合条件输出YES，否则NO

2 车牌限号¶

描述¶ 今天某市交通管制，**车牌尾号为奇数**的车才能上路。问给定的车牌号今天是否能上路。

输入¶ 一行。一个由字母和数字构成的长度为1到10的字符串，代表车牌号，**车牌号最后一位是数字**。

输出¶ 今天可以上路，输出“YES”，否则输出“NO”。注意字母均为大写。

同样是只有一个独立条件，一个块解决。

- 输入：一个字符串
- 判断：尾号（`s[-1]`）为奇数
- 输出：符合条件输出YES，否则NO

3 判断是否构成三角形¶

描述¶ 给定三个正整数，分别表示三条线段的长度，判断这三条线段能否构成一个三角形。

输入¶ 输入共一行，包含**三个正整数**，分别表示三条线段的长度，数与数之间以**一个空格分开**。（三条边的长
1000010000）

输出¶ 如果能构成三角形，则输出 1 ，否则输出 0。

这道题的输入和判断需要联合思考，相对复杂一点。

- 输入：一行内输入，使用`input().split()`，取到之后转换成int
- 逻辑判断：理想情况是两个短边相加和长边比较，但是题目没有保证顺序。所以需要排序或者轮流。
- 输出：简单输出。

4 商品折扣计算¶

描述¶ 某电商平台推出阶梯折扣活动，根据订单金额给予不同优惠：

订单金额低于500元，无折扣 500元到2000元之间，超出500元的部分打9折
2000元到5000元之间，超出2000元的部分打8折 5000元以上，超出5000元的部分打6折
输入订单金额，计算用户实际支付金额（单位：元），要求使用f-string直接保留两位小数

输入¶ 一行。一个大于等于0的浮点数（包含整数和小数部分），表示订单金额。

输出¶ 一行。保留两位小数的实际支付金额。

本题简单输入，简单输出，复杂全在逻辑

- 输入：一个简单浮点数
- 输出：一个简单浮点数

如果设old为原价，ans为目标答案，计算的表达式如下：

```
ans = 0
if old <= 500:
    ans = old
elif 500 < old <= 2000:
    ans = 500 + (old - 500) * 0.9
elif 2000 < old <= 5000:
    ans = 500 + 1500 * 0.9 + (old - 2000) * 0.8
elif old > 5000:
    ans = 500 + 1500 * 0.9 + 3000 * 0.8 + (old - 5000) * 0.6
```

循环要点整理和作业3提示¶

Python中，常用的循环结构主要有两种：for循环和while循环。

for循环适合遍历一个已知长度的数据结构，比如通过range()生成的数字序列或字符串中的每个字符；这种迭代

while循环则更灵活，它根据运行时的条件来判断是否继续执行，非常适合那些终止条件不固定、依赖动态计算

for–range¶

range是一个python中的生成器，用于表示一个区间的概念。

range可以在for循环中使用，也是最常见的用法：

- `range(stop)`：生成从 0 到 stop–1 的整数序列。
- `range(start, stop)`：生成从 start 到 stop–1 的整数序列。
- `range(start, stop, step)`：生成从 start 开始，每次增加 step，直到达到但不超过 stop 的整数序列。

In [16]:

```
for i in range(1, 10, 2):
    print(i, end=' ')
print()
for i in range(10, 1, -1):
    print(i, end=' ')
```

```
1 3 5 7 9
10 9 8 7 6 5 4 3 2
```

拓展知识：生成器具有惰性计算的特点

In [17]:

```
import sys

r = range(10**6)  #      0   999999   range
l = list(r)       #   range
```

```
print(f"Size of range: {sys.getsizeof(r)} bytes")  #
print(f"Size of list: {sys.getsizeof(l)} bytes")  #

r[9999]
print(f"Size of range: {sys.getsizeof(r)} bytes")  #
```

```
Size of range: 48 bytes
Size of list: 8000056 bytes
Size of range: 48 bytes
```

例题：自由练习030

给出两个不相等的正整数，求它们之间（包括边界）的奇数之和。

输入：两个正整数。 输出：区间中奇数的求和。

In [18]:

```
# a = int(input())
# b = int(input())
a, b = 9, 2

if a > b:
    a, b = b, a
ans = 0
for i in range(a, b+1):
    if i % 2:
        ans += i

print(ans)
```

```
24
```

for–str¶

字符串在之前的课程中已经使用过多次，使用如下语句：

```
for c in str1:
    #
```

例题：自由练习031

输入一行字符，统计出其中数字字符的个数。

输入：一行字符串，总长度不超过255。 输出：输出为1行，输出字符串里面数字字符的个数。

In [19]:

```
# 2100017436_
# str1 = input()
str1 = "abc1234"
```

```
numN = 0
for x in str1:
    if x in "1234567890": # isdigit(x)
        numN += 1
print(numN)
```

```
4
```

二重循环¶

循环可以嵌套。

例题：自由练习034 求阶乘的和

给定正整数n，求不大于n的正整数的阶乘的和，（即求1!+2!+3!+...+n!）

输入 一个正整数n（1 < n < 12）。 输出 不大于n的正整数的阶乘的和阶乘的和。

In [20]:

```
# 2100017436
# n = int(input())
n = 5
s = 0
for x in range(1,n+1):
    a = 1
    for x1 in range(1,x+1):
        a *= x1
    s += a
print(s)
```

```
153
```

优化：

In [21]:

```
# n = int(input())
n = 5
s = 0
a = 1
for x in range(1,n+1):
    a *= x
    s += a
print(s)
```

```
153
```

while循环¶

while循环是一种比for循环更加基本的循环，只要条件满足，就不断循环，条件不满足时退出循环。

例题：自由练习038 数字黑洞2178

数学中有一个有趣的现象，给一个4位数，按照一个简单的规则，总能得到2178或0。

这个规则是：将数字颠倒并与原数字相减，得到一个差（要求用大的减小的，或者说取绝对值）。

例如：输入1960，运算过程如下：

```
1269   1    1269    1960-0691 = 1269
8352   2    8352   9621-1269 = 8352
5814    3
1629
7632
5265
360   7    360    5625-5265=360
297   8    360   360-063 = 297
495
99
0   11    0
```

又例如：输入 9801，共运算3次，分别得到8712，6534，2178。得到2178也结束。

编写程序，输入一个4位数，输出得到2178或0的过程共运算了几次。

输入：一个4位正整数 输出：一个大于等于0的整数，它是得到2178或0的过程的运算次数

In [22]:

```
# n = int(input())
n = 1960
count = 0
if n == 0 or n == 2178:
    print(count)
else:
    while n != 0 and n!= 2178:
        reverse = str(n)
        reverse = reverse[::-1]
        reverse = int(reverse)
        difference = abs(n - reverse)
        count += 1
        n = difference
    print(count)
```

```
11
```