General

Infinite loops

Take care to avoid infinite loops. An infinite loop is a loop that never stops executing: in most of the cases it concerns a `while`-loop where the statements inside the loop never take care to make the `while`-condition `False` after some time. As an example, take a look at the following code snippet

```
>>> i = 0
>>> a = 0
>>> while i < 4:
...     a += 1
```

Because the statement `a += 1` will never cause the value of the variable `i` to become larger than or equal to 4, the condition `i < 4` will evaluate to `True` forever.

Tip: If you work with PyCharm, you know that a program that was started is still running from the red square to the left of the Console or top right in the navigation bar. If you click the red square, you force the program to stop.

Premature abortion of loops

Use the statements `break` and `continue` to abort a loop before it has come to completion. However, these statements are generally considered bad programming style. So you'd better avoid them at all cost, as they will cost you points when used during an evaluation or an exam.

One situation where you may want premature abortion of a loop to occur, is when you want to find a solution by trying all possible cases, and stop as soon as one solution has been found. Instead of using `break` or `continue` in this case, it is better to use an additional Boolean variable that indicates whether the solution has already been found.

```
>>> found = False
>>> while not found:
...     if <solution found>:  #solution found represents a condition
...         found = True
...
```

As soon as the solution has been found (represented here by the fact that the condition *solution found* evaluates to `True`), the variable `found` is assigned the value `True`. As a result, the `while`-loop ends the next time the `while`-condition is evaluated after the current iteration.

New York Times

Read multiple lines from input

If you know in advance how many lines must be read from input, you may use the following strategy

```
>>> lines = 4
>>> for _ in range(lines):
...     line = input()
...     # process the line
...
```

If the number of lines that must be read from input is not known in advance, but you know for example that the last line is an empty line, you may use the following strategy

```
>>> line = input()
>>> while line:
...     # process the line
```

```
...     line = input()
...
```

Specific information

You always need to print the 'sign' of the increment between consecutive numbers. This can be done by using the *format specifier* `print(f"{x:+d}")` that automatically displays the sign (positive or negative) of integer `x` (`int`).

Canvascrack

Counting starts at zero

Computer scientists by default start counting from zero, not from one as mathematicians would. Python follows this tradition in many of its design choices. As an example, the built-in function `range` generates a sequences of successive integers that starts at zero, if you only pass a single argument to the function. Here's how you count to 5

```
>>> for i in range(6):
...     print(i)
...
0
1
2
3
4
5
```

If you want counting to start at another value, you can pass this value as an extra argument to the `range` function.

```
>>> for i in range(1, 6):
...     print(i)
...
1
2
3
4
5
```

However, it is considered a more *pythonic* solution to write the above as

```
>>> for i in range(5):
...     print(i + 1)
...
1
2
3
4
5
```

Thoughts that count

Specific information

To solve this problem, you can iterate all possible amounts of white roses. With this number you can calculate the corresponding number of red and blue roses. When the sum of the number of red roses and the number of blue roses satisfies the condition, you print the result.