Payslip

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()
...
```

Conan the Bacterium

Specific information

The first experiment

We can translate the problem to a mathematical formula. Lets call the number of bacteria after t seconds z_t. From the description we deduce that the number of bacteria in the testtube can be modelled as:

$$\begin{aligned} z_0 &= 1 \\ z_{t+1} &= az_t + b \end{aligned}$$

In

Pythonically speakign, this can be implemented with a loop. For each iteration of this loop `z = a * z + b` should be executed. Decide for yourself if a `for`-loop or `while`-loop is more appropriate.

The second experiment

In the second experiment a very similar formula can be found:

$$\begin{aligned} z_0 &= t \\ z_{t+1} &= az_t + b \end{aligned}$$

There are only two differences: the initial value z_0 and the loop should be terminated if the number of bacteria is more than in the first test tube. Again, decide if a `for`-loop or `while`-loop is more appropriate.

Heat Wave

Premature abortion of loops

In Python you can use the statements `break` and `continue` to abort a loop before it has come to completion. In general, however, these statements are considered bad programming style.

One situation where you may want a premature abortion of a loop occurs 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 or not 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`-conditions is evaluated after the current iteration.

Specific information

In this assignment it is a good idea to count the number of successive **summer days** that have been observed in the passed period, and how many **tropical days** have been observed in the passed period of successive summer days.

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 initial 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 Eclipse, you known that a program that was started is still running if you observe a red square in the top menu of the Console. If you click the red square, you force the program to stop.

Counting starts at zero

Computer scientists by default start counting from zero, not from one, and Python follows this tradition in many of its design decisions. 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 in Python

```
>>> 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
```