Loops

Exercise 1 — While loop

1. (a) What do the variables V and W contain after this sequence of instructions? Answer this question by showing a step-by-step evaluation of the 4 variables.
   (b) Same question if the initial values of A and B in lines 1 and 2 are $A = 17; B = 5$? If they are $A = 12; B = 3$?
   (c) Can you guess what happens for the general case $A = m; B = n$?

Algorithm 1.

Variables:
A and B are two positive integers.
V and W are two integers.

Instructions of the algorithm:
1 $A \leftarrow 5$
2 $B \leftarrow 12$
3 $W \leftarrow 0$
4 $V \leftarrow A$
5 While $(V \geq B)$
6 $\quad V \leftarrow V - B$
7 $\quad W \leftarrow W + 1$
8 End While

2. The specification of the algorithm makes it mandatory for A and B to be positive integers. However, there is no `Python` type for positive integers. It is therefore not directly possible to translate this algorithm into a `Python` program ; in `Python`, A, B, V and W would be 4 integers. Add, between lines 2 and 3, a test to check that A and B are positive numbers, and halt the program if one of them is not.

Exercise 2 — For loop

What does the variable X contain after this sequence of instructions?

Algorithm 2.

Variables:
N and I are two positive integers.
X is an integer.

Instructions of the algorithm:
1 $N \leftarrow$ Input("Enter a positive integer: ")
2 $X \leftarrow 0$
3 For I from 1 to N
4 $\quad X \leftarrow X + I \times I$
5 End For

Exercise 3 — Palindromes

A palindrome is a sequence of letters that can be read the same way backward as forward. Usually, spaces are omitted.

Example: "kayak", "Bob", "Never odd or even", "Esope reste et se repose" (French), "Νίψον ἀνομήματα, μὴ μόναν ὄψιν" (Greek)...

Our goal is to write a program that recognizes if a string given as input is, or is not, a palindrome. It is relatively easy to write this program as long as we look at the `Python` documentation:

“

str.**lower**()
Return a copy of the string with all the cased characters converted to lowercase.

[...]

str.**replace**(*old*, *new*[, *count*])

Return a copy of the string with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.

[...]

s[i:j:k][1]

slice of s from i to j with step k

https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

”

1. To have the string s be transformed backwards, use a slice of s with step -1, without specifying start and end indexes (it will thus slice the full string).

2. “Bob” transformed backwards by the previous question is “boB”. If we ask `Python` to compare those two strings[2], it will tell us they are different. Use the `lower` method before slicing to overcome this problem.

3. “Never odd or even” transformed by the previous question is “never odd or even”, and its transformation backwards is “neve ro ddo reven”. If we ask `Python` to compare those two strings, it will tell us they are different. We will remove spaces before slicing to overcome this problem: find a way to use the `replace` method to do this.

4. Deduce from the previous questions a sequence of instructions that asks the user to enter a string, then tests if it is a palindrome, and finally outputs a message to tell the user whether it is a palindrome or not.

5. You might wonder what this exercise has to do with loops. We have used neither the `while` nor the `for` loop. In fact, the three operations we did on strings (slicing, putting to lower case, removing the spaces) all use a loop which is hidden to us because we use directly the method, without seeing its implementation. It is interesting to re-implement those methods “by hand”, so please finish the `Python` example below.

```
1 string=input("Enter a string: ")
2 new_string=""
3 for c in string:                    #Loop over the characters in string
4     if
5         new_string = new_string + c
6 print("Your string without spaces is: " + new_string)
```

Listing 1: Unfinished implementation of a space removal algorithm.

[1] This slice operator is an extension of the slice operator that only takes the start and end indexes, as we have seen in Work n°4, 2. 4. Make sure you finish section 2 of Work 4 before using this more powerful slice operator!

[2] Testing if two strings are equals is what we have seen in Work n°4, 1. 17 (4^{th} line). Make sure you finish section 1 of Work 4!