Imperative...

Exercise 1 (From Caml to Algo)

CAML

What does the following "program" do?

```
1
2  let test n =
3    (n >= 100) && (n < 1000) ;;
4
5
6
7  let sum_digits n =
8    let (a, b, c) =
9        (n / 100, (n/10) mod 10, n mod 10)
10   in
11     a + b + c ;;
12
13
14
15 let product_digits n =
16   let (a, b, c) =
17       (n / 100, (n/10) mod 10, n mod 10)
18   in
19     a * b * c ;;
20
21
22
23 let abs n =
24   if n > 0 then
25     n
26   else
27     -n ;;
28
29
30
31 let rec loop n =
32   let n = abs n  in
33     if sum_digits n = product_digits n
     then
34       n
35     else
36       loop (n+1) ;;
```

ALGO

```
function test (integer n) : boolean
begin
   return ( (n >= 100) and (n < 1000) )
end

function sum_digits (integer n) : integer
  variables
      integer a, b, c
begin
   a ← n div 100
   b ← (n div 10) mod 10
   c ← n mod 10
   return (a + b + c)
end

function product_digits (integer n) : integer
  variables
     integer a, b, c
begin
   a ← n div 100
   b ← (n div 10) mod 10
   c ← n mod 10
   return (a * b * c)
end

function abs (integer n) : integer
begin
   if n > 0 then
      return n
   else
      return -n
   end if
end

function loop (integer n) : integer
begin
   n ← abs(n)
   if sum_digits(n) = product_digits(n) then
      return n
   else
      return loop(n+1)
   end if
end
```

Python

Exercise 2 (Maximum)

The following algorithm reads three integer values and displays the maximum of the three.

```
function max3(integer x, y, z) : integer
    variables
        integer    m
begin
    if (x > y) and (x > z) then
        m ← x
    else
        if (y > x) and (y > z) then
            m ← y
        else
            if (z > x) and (z > y) then
                m ← z
            end if
        end if
    end if
    return m
end

variables
    integer    a, b, c

begin      /* main algorithm */
    read (a)
    read (b)
    read (c)
    write (max3 (a, b, c))
end
```

Correct (if necessary) and simplify the algorithm: there must be the fewer tests possible. Then translate it in Python.

To write (display), use ***print****:*

```
>>> print("test")
test
>>> print(42)
42
>>> print("test", 42)
test 42
```

To read, use ***input****:*

```
>>> x = int(input("your integer here: "))
your integer here: 42
>>> x
42
```

Exercise 3 (The day after)

A date is defined by three integers for the year, the month and the day.

Write a procedure `tomorrow(d, m, y)` that displays, given a date, the date of the day after. Furthermore, the procedure has to test whether the given date is valid (ex: there is no February 30^{th}).

Indication: a year is leap if it is a multiple of 4 and it is not a multiple of 100 except if it is a multiple of 400.

Application examples:

```
>>> tomorrow(31, 2, 2021)
non-valid
>>> tomorrow(31, 10, 2021)
The day after: 1 / 11 / 2021
>>> tomorrow(29, 2, 2022)
non-valid
>>> tomorrow(29, 2, 2020)
The day after: 1 / 3 / 2020
```

Exercise 4 (List to 9)

Given a 2-digit positive integer AB such that A and B are different. For example $AB = 19$.

- Reverse the 2 digits to obtain 91.
- Subtract 19 from 91 to obtain 91-19=72.

This process is repeated with 72 (to obtain 45 = 72 - 27). A last repetition gives 9 = 54 - 45. This list is called "list to 9" because it always ends with the number 9.

If the two digits are equals, the list to 9 is not defined.

Write a function `list_to_9(n)` that displays the list to 9 of a positive number with 2 different digits and returns the number of elements of that list (0 if the list is not defined).

Application examples:

```
>>> list_to_9(19)
19
72
45
9
4 # return value of list_to_9(19)
>>> list_to_9(22)
no list to 9!
0 # return value of list_to_9(22)
>>> list_to_9(123)
123 is not a 2-digit positive integer
0 # return value of list_to_9(123)
```

Exercise 5 (Bonus: Sequence)

Let the following sequence be:

line 0 :	1	is 1 "1"
line 1 :	11	is 2 "1"
line 2 :	21	is 1 "2" followed by 1 "1"
line 3 :	1211	is 1 "1" followed by 1 "2" followed by 2 "1"
line 4 :	111221	...
line 5 :	312211	
line 6 :	13112221	

...

Write a function `sequence` that returns the n^{th} line of this sequence, as an **integer**.

Loops, string of characters and lists cannot be used!

Application examples:

```
>>> sequence(5)
312211
>>> sequence(0)
1
```