Binary II

We continue our exploration of the world of 1's and 0's.

Activity 1 (Palindromes)**.**

Goal: find palindromes in the binary and decimal numeral systems.

In English a palindrome is a word (or a sentence) that can be read in both directions, for example "**RADAR**" or "**A MAN, A PLAN, A CANAL: PANAMA**". In this activity, a ***palindrome*** will be a list, which has the same elements when browsing from left to right or right to left.
Examples:

- `[1,0,1,0,1]` is a palindrome (in the binary numeral system),
- `[2,9,4,4,9,2]` is a palindrome (in the decimal numeral system).

1. Program an `is_palindrome(mylist)` function that tests if a list is a palindrome or not.
   Hints. You can compare the items at index i and $p-1-i$ or use `list(reversed(mylist))`.
2. We are looking for integers n such that their binary value is a palindrome. For example, the binary notation of $n = 27$ is the palindrome `[1,1,0,1,1]`. This is the tenth integer n that has this property.
   What is the thousandth integer $n \geqslant 0$ whose binary value is a palindrome?
3. What is the thousandth integer $n \geqslant 0$ whose decimal value is a palindrome?
   For example, the digits of $n = 909$ in the decimal system, form the palindrome `[9,0,9]`. This is the hundredth integer n that has this property.
4. An integer n is a ***bi-palindrome*** if its binary notation *and* its decimal notation are palindromes. For example $n = 585$ has a decimal value which is a palindrome and so is its binary value `[1,0,0,1,0,0,1,0,0,1]`. This is the tenth integer n that has this property.
   What is the twentieth integer $n \geqslant 0$ that is a bi-palindrome?

Lesson 1 (Logical operations)**.**
We consider that 0 represents "False" and 1 represents "True".

- With the logical operation "OR", the result is true as soon as at least one of the two terms is true. Case by case:
  - 0 OR 0 = 0
  - 0 OR 1 = 1
  - 1 OR 0 = 1
  - 1 OR 1 = 1
- With the logical operation "AND", the result is true only when both terms are true. Case by case:

  - 0 AND 0 = 0
  - 0 AND 1 = 0
  - 1 AND 0 = 0
  - 1 AND 1 = 1
- The logical operation “NOT”, inverts true and false values:
  - NOT 0 = 1
  - NOT 1 = 0
- For numbers in binary notation, these operations range from bit to bit, i.e. digit by digit (starting with the digits on the right) as one would with adding (without carrying).
  For example:

	1.0.1.1.0			1.0.0.1.0
AND	1.1.0.1.0		OR	0.0.1.1.0
	1.0.0.1.0			1.0.1.1.0

  If the two systems do not have the same number of bits, we add non-significant 0’s on the left (as shown in the example of 1.0.0.1.0 OR 1.1.0 in the figure on the right).

Activity 2 (Logical operations)**.**

Goal: program the main logical operations.

1. (a) Program a `NOT()` function which corresponds to the negation for a given list. For example, `NOT([1,1,0,1])` returns `[0,0,1,0]`.
   (b) Program an `OReq()` function which corresponds to “OR” with two lists of equal length. For example, given `mylist1 = [1,0,1,0,1,0,1]` and `mylist2 = [1,0,0,1,0,0,1]`, the function returns `[1,0,1,1,1,0,1]`.
   (c) Do the same for `ANDeq()` for two lists having the same length.
2. Write a `zero_padding(mylist,p)` function that adds zeros at the beginning of the list to get a list of length `p`. Example: if `mylist = [1,0,1,1]` and `p = 8`, then the function returns `[0,0,0,0,1,0,1,1]`.
3. Write two functions `OR()` and `AND()` which correspond to the logical operations, but with two lists that do not necessarily have the same length.
   Example:
   - `mylist1 = [1,1,1,0,1]` and `mylist2 = [1,1,0]`,
   - it should be considered that `mylist2` is equivalent to the list `mylist2bis = [0,0,1,1,0]` of the same length as `mylist1`,
   - so `OR(mylist1,mylist2)` returns `[1,1,1,1,1]`,
   - then `AND(mylist1,mylist2)` returns `[0,0,1,0,0]` (or `[1,0,0]` depending on your choice).

   Hints. You can use the content of your functions `OReq` and `ANDeq`, or you can first add zeros to the shortest list.

Activity 3 (De Morgan's laws)**.**

Goal: generate all possible lists of 0*'s and* 1*'s to check a proposition.*

1. **First method: use binary notation.**
   We want to generate all possible lists of 0's and 1's of a given size p. Here's how to do it:
   - We define an integer n, that ranges from 0 to $2^p - 1$.
   - For each of these integers n, we calculate its binary value (in the form of a list).
   - We add (if necessary) 0 at the beginning of the list, in order to get a list of length p.

   Program this method into a function.
   Example: $n = 36$ in binary notation is `[1,0,0,1,0,0]`. If you want a list of $p = 8$ bits, you should add two 0: `[0,0,1,0,0,1,0,0]`.
2. **Second method (optional): a recursive algorithm.**
   We again want to generate all the possible lists of 0's and 1's of a given size. We adopt the following procedure: if we know how to find all the lists of size $p-1$, then to obtain all the lists of size p, we just have to add one 0 at the beginning of each list of size $p-1$, then to start again by adding a 1 at the beginning of each list of size $p-1$.
   For example, there are 4 lists of length 2: `[0, 0]`, `[0, 1]`, `[1, 0]`, `[1, 1]`. I get the 8 lists of length 3:
   - 4 lists by adding 0 at the front: `[0, 0, 0]`, `[0, 0, 1]`, `[0, 1, 0]`, `[0, 1, 1]`,
   - 4 lists by adding 1 at the front: `[1, 0, 0]`, `[1, 0, 1]`, `[1, 1, 0]`, `[1, 1, 1]`.

   This gives the following algorithm, which is known as a recursive algorithm (because the function calls itself).

   Algorithm.
   Use: `every_binary_number(p)`
   Input: an integer `p` > 0
   Output: the list of all possible lists of 0's and 1's of length `p`
   - If `p` $= 1$ return the list `[ [0], [1] ]`.
   - If `p` $\geqslant 2$, then:
     - get all lists of size `p-1` by the call `every_binary_number(p-1)`
     - for each item in this list, build two new items:
       - on the one hand add `0` at the beginning of this element;
       - on the other hand add `1` at the beginning of this element;
       - then add these two items to the list of lists of size `p`.
   - Return the list of all the lists with a size `p`.
3. **De Morgan's laws.**
   De Morgan's laws state that for booleans (true/false) or bits (1/0), we always have these properties:

   $$\text{NOT}(b_1 \text{ OR } b_2) = \text{NOT}(b_1) \text{ AND } \text{NOT}(b_2) \qquad \text{NOT}(b_1 \text{ AND } b_2) = \text{NOT}(b_1) \text{ OR } \text{NOT}(b_2).$$

   Experimentally check that these equations are still true for any list ℓ_1 and ℓ_2 of exactly 8 bits.