Binary I

The computers transform all data into numbers and manipulate only those numbers. These numbers are stored in the form of lists of 0's and 1's. It's the binary numeral system of numbers. To better understand this binary numeral system, you must first need to understand the decimal numeral system better.

Lesson 1 (Base 10 system)**.**
We usually denote integers with the decimal numeral system (based on 10). For example, 70 685 is $7 \times 10\,000 + 0 \times 1\,000 + 6 \times 100 + 8 \times 10 + 5 \times 1$:

7	0	6	8	5
10000	1000	100	10	1
10^4	10^3	10^2	10^1	10^0

(we can see that 5 is the number of units, 8 the number of tens, 6 the number of hundreds…).
It is necessary to understand the powers of 10. We denote $10 \times 10 \times \cdots \times 10$ (with k factors) by 10^k.

d_{p-1}	d_{p-2}	$\cdots$	d_i	$\cdots$	d_2	d_1	d_0
10^{p-1}	10^{p-2}	$\cdots$	10^i	$\cdots$	10^2	10^1	10^0

We calculate the integer corresponding to the digits $[d_{p-1}, d_{p-2}, \ldots, d_2, d_1, d_0]$ by the formula :

$$n = d_{p-1} \times 10^{p-1} + d_{p-2} \times 10^{p-2} + \cdots + d_i \times 10^i + \cdots + d_2 \times 10^2 + d_1 \times 10^1 + d_0 \times 10^0$$

Activity 1 (From decimal system to integer)**.**

Goal: from the decimal notation, find the integer.

Write a function `decimal_to_integer(list_decimal)` which takes a list representing the decimal notation and calculates the corresponding integer.

`decimal_to_integer()`

Use: `decimal_to_integer(list_decimal)`
Input: a list of numbers between 0 and 9
Output: the integer whose decimal notation is the list

Example: if the input is `[1,2,3,4]`, the output is `1234`.

Hints. It is necessary to sum up elements of type:

$$d_i \times 10^i \quad \text{for } 0 \leqslant i < p$$

where p is the length of the list and d_i is the digit at index i *counting from the end* (i.e. from right to left). To manage the fact that the index i used for the power of 10 does not correspond to the index in the list, there are two solutions:

- understand that $d_i = \texttt{mylist}[p-1-i]$ where `mylist` is the list of p digits,
- or start by reversing `mylist`.

Lesson 2 (Binary).

- **Power of 2.** We denote 2^k for $2 \times 2 \times \cdots \times 2$ (with k factors). For example, $2^3 = 2 \times 2 \times 2 = 8$.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

- **Binary system: an example.**
  Integers can be represented using a binary numeral system, i.e. a notation where the only numbers are 0 or 1. In binary, the digits are called ***bits***. For example, 1.0.1.1.0.0.1 (pronounce the numbers one by one) is the binary numeral system of the integer 89. How to do you do this calculation? The same way we did using the base 10, but instead using the powers of 2.

1	0	1	1	0	0	1
64	32	16	8	4	2	1
2^6	2^5	2^4	2^3	2^2	2^1	2^0

  So the system 1.0.1.1.0.0.1 represents the integer:

$$1 \times 64 + 0 \times 32 + 1 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 64 + 16 + 8 + 1 = 89.$$

- **Binary system: formula.**

b_{p-1}	b_{p-2}	$\cdots$	b_i	$\cdots$	b_2	b_1	b_0
2^{p-1}	2^{p-2}	$\cdots$	2^i	$\cdots$	2^2	2^1	2^0

  We can calculate the integer corresponding to the bits $[b_{p-1}, b_{p-2}, \ldots, b_2, b_1, b_0]$ as a sum of terms $b_i \times 2^i$, by the formula :

$$n = b_{p-1} \times 2^{p-1} + b_{p-2} \times 2^{p-2} + \cdots + b_i \times 2^i + \cdots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

- `Python` **and the binary.** `Python` accepts the binary decimal system directly as long as we use the prefix "`0b`". Examples:
  - with `x = 0b11010`, then `print(x)` displays 26,
  - with `y = 0b11111`, then `print(y)` displays 31,
  - and `print(x+y)` displays 57.

Activity 2 (From binary numeral system to integer).

Goal: convert binary values to integers.

1. Calculate integers whose binary numeral system is given below. You can do it by hand or get help from `Python`. For example 1.0.0.1.1 is equal to $2^4 + 2^1 + 2^0 = 19$ as confirmed by the command `0b10011` which returns 19.
   - 1.1, 1.0.1, 1.0.0.1, 1.1.1.1
   - 1.0.0.0.0, 1.0.1.0.1, 1.1.1.1.1
   - 1.0.1.1.0.0, 1.0.0.0.1.1

- 1.1.1.0.0.1.0.1

2. Write a function `binary_to_integer(list_binary)` which takes in a list representing a binary system value and calculates the corresponding integer.

`binary_to_integer()`

Use: `binary_to_integer(list_binary)`
Input: a list of bits, 0 and 1
Output: the integer whose binary numeral system is the list

Examples:
- input `[1,1,0]`, output 6
- input `[1,1,0,1,1,1]`, output 55
- input `[1,1,0,1,0,0,1,1,0,1,1,1]`, output 3383

Hints. This time it is necessary to sum up elements of type:

$$b_i \times 2^i \quad \text{for } 0 \leqslant i < p$$

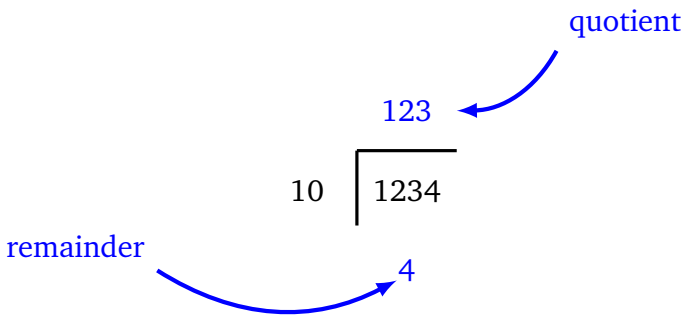
where p is the length of the list and b_i is the bit (0 or 1) at index i of the list counting from the end.

3. Here is an algorithm that does the same conversion: it allows the calculation of the integer from the binary notation, but it has the advantage of never directly calculating powers of 2. Program this algorithm into a `binary_to_integer_bis()` function that has the same characteristics as the previous function.

Algorithm.
Input: `mylist` : a list of 1's and 0's
Output: the associated binary number
- Initialize a variable n to 0.
- For each element b from `mylist` :
  - if b is 0, then make $n \leftarrow 2n$,
  - if b is 1, then make $n \leftarrow 2n+1$.
- The result is the value of n.

Lesson 3 (Base 10 system (again))**.**
Of course, when you see the number 1234 you can immediately find the list of its digits $[1,2,3,4]$. But how to do it in general from an integer n?

1. We will need to use the Euclidean division by 10:
   - calculate the ***remainder*** left over after dividing by 10 using `n % 10`, we also call this ***n modulo 10***
   - calculate `n // 10` or the ***quotient*** of this division.



2. The Python commands are simply `n % 10` and `n // 10`.
   Example: `1234 % 10` is equal to 4; `1234 // 10` is equal to 123.
3. - The digit of the units of n is obtained using modulo 10: it is $n\%10$. For example $1234\%10 = 4$.
   - The tens figure is obtained from the quotient of n divided by 10 and then taking modulo 10: it is therefore $(n//10)\%10$. Example: $1234//10 = 123$, then we have $123\%10 = 3$; the figure of tens of 1234 is indeed 3.
   - For the figure of hundreds, we divide n by 100, then take modulo 10. Example $1234//100 = 12$; 2 is the figure of the units of 12 and it is the figure of the hundreds of 1234. Note: dividing by 100 means dividing by 10, then again by 10.
   - For the figure of thousands we calculate the quotient of the division by 1000 then we take the figure of the units...

Activity 3 (Find the digits of an integer)**.**

Goal: decompose an integer into a list of its digits (based on 10*).*

Program the function `integer_to_decimal()` using the following algorithm.

`integer_to_decimal()`

Use: `integer_to_decimal(n)`
Input: a positive integer
Output: the list of its digits

Example: if the input is 1234, the output is `[1,2,3,4]`.

Algorithm.
Input: an integer $n > 0$
Output: the list of its digits

- Start from an empty list.
- As long as n is not zero:
  - add $n\%10$ at the beginning of the list,
  - set $n \leftarrow n//10$.
- The result is the desired list.

Lesson 4 (Binary system with `Python`)**.**
`Python` calculates the binary value of an integer very well using the `bin()` function.

`python : bin()`

Use: `bin(n)`
Input: an integer
Output: the binary numeral system of n in the form of a string starting with `'0b'`

Example:
- `bin(37)` returns `'0b100101'`
- `bin(139)` returns `'0b10001011'`

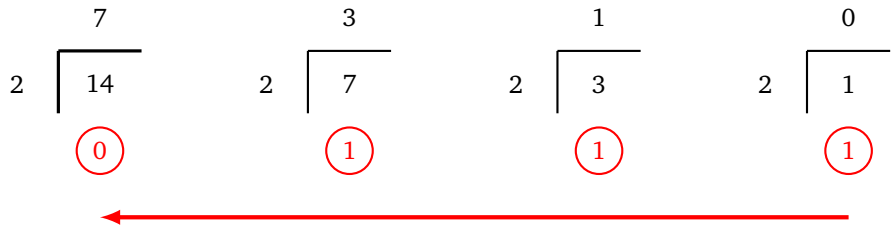
Lesson 5 (Binary system calculation)**.**
Calculating the binary value of an integer n, is the same as calculating the decimal value except replace the divisions by 10 with a divisions by 2.
So we need:
- $n\%2$: the remainder of the Euclidean division of n by 2 (also called n modulo 2); the remainder is either 0 or 1.
- and $n//2$: the quotient of this division.

Note that the remainder $n\%2$ is either 0 (when n is even) or 1 (when n is odd).
Here is a general method to calculate the binary value of an integer:
- We start from the integer whose binary value we want.
- A series of Euclidean divisions by 2 are performed:
  - for each division, you get a remainder 0 or 1;
  - we get a quotient that we divide again by 2... We stop when this quotient is zero.
- The binary system is read as the sequence of the remainders, but starting from the last one.

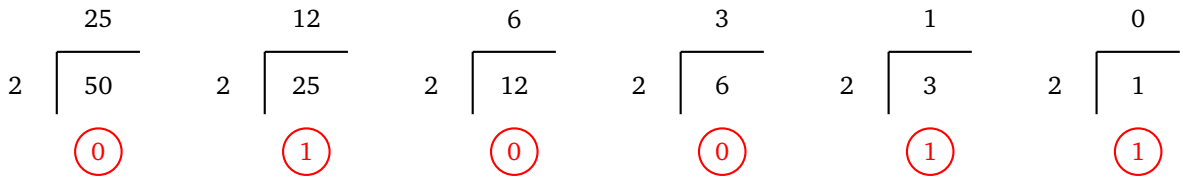
Example.
Calculation of the binary value of 14.

- We divide 14 by 2, the quotient is 7, the remainder is 0.
- We divide 7 (the previous quotient) by 2: the new quotient is 3, the new remainder is 1.
- We divide 3 by 2: quotient 1, remainder 1.
- We divide 1 by 2: quotient 0, remainder 1.
- It's over (the last quotient is zero).
- The successive remainders are 0, 1, 1, 1. we read the binary numeral system backwards, it is 1.1.1.0.

The divisions are done from left to right, but the remainders are read from right to left.



Example.
Binary value of 50.



The successive remainders are 0, 1, 0, 0, 1, 1, so the binary notation of 50 is 1.1.0.0.1.0.

Activity 4.

Goal: find the binary value of an integer.

1. Calculate the binary value of the following integers by hand. Check your results using the `Python bin()` function.
   - 13, 18, 29, 31,
   - 44, 48, 63, 64,
   - 100, 135, 239, 1023.
2. Program the `integer_to_binary()` function using the following algorithm.

Algorithm.
Input: an integer $n > 0$
Output: its binary value in the form of a list

- Start from an empty list.
- As long as n is not zero:
  - add $n\%2$ at the beginning of the list,
  - set $n \leftarrow n//2$.
- The result is the desired list.

`integer_to_binary()`

Use: `integer_to_binary(n)`
Input: a positive integer
Output: its binary notation in the form of a list

Example: if the input is 204, the output is `[1,1,0,0,1,1,0,0]`.

Make sure your functions are working properly:

- start with any integer n,
- calculate its binary value,
- calculate the integer associated with this binary notation,
- you should find the starting integer!