Information and Communication Technologies — S6 — Work n°6

Functions

It is useful to define functions and use them in some algorithms, as we saw with Reeborg. This avoids copying and pasting lines of code, when we want to use the same operations at different locations in our program. For example, it was really useful to teach Reeborg how to turn right, as in Listing 1, and call it every time we need it, instead of writing three `turn_left()` each time.

```
def turn_right():
    repeat 3:
        turn_left()
```

Listing 1: Sample code for a right turn with Reeborg.

In a cook book, it is convenient to describe only once how to make a "béchamel" sauce, and to refer to this description every time a recipe needs this sauce, instead of rewriting the recipe for the sauce each time!

A function can take inputs or not, and can return values, or not (if it does not return any value, we say that it produces only "side effects"). For example, a function that adds two integers can be written as:

```
function addition($a$ : integer ; $b$ : integer) : integer
    return $a + b$
```

It is important to note the type of the input variables, to understand what can be done with those input variables. It is also important to specify the type of the function, which is the type of the values returned by the function. Some programming languages automatically type the input variables and the functions, some ask the programmer to specify the types.

To define a function in `Python`, we write the keyword `def` followed by the name we want to give to the function, followed by the name of the input variables, between parentheses. Then, we indent the body of the function, which is a sequence of operations that the function will compute, and we finish by the keyword `return` followed by the value the function must return.

Example: a `Python` program that implements the addition function defined before is given in Listing 2.

```
def addition(a, b):
    return a + b
```

Listing 2: Sample code for the addition of two integers.

Remark: this function can also be called with two strings as input, and will then return the concatenation of those two strings (*i.e.*, `addition("Hel", "lo.")` will return `"Hello."`. `Python` does not ask us to type input variables, and will only generate an error upon calling this function if it is impossible to use the built-in "+" operation (this operation is thus ambiguous!).

Exercise 1 — The minimum

1. Write a function "min2numbers" that takes two numbers as input and returns the minimum of them.
2. Write a function "min3numbers" that takes three numbers as input and returns the minimum of them:
   (a) Using comparisons, but not the function "min2numbers".
   (b) Using "min2numbers", but without using comparisons.

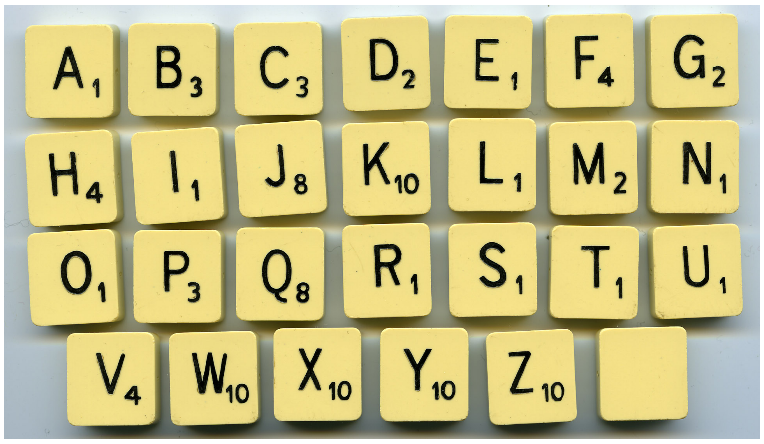
3. Implement these functions in `Python`.
   Remark: of course, in `Python` those functions already exist: it is the function `min`.

Exercise 2 — Strings

For this exercise, you can start directly in `Python`. It is useful to remind the `Python` functions seen in the Work n°4. You can use them, but not other `Python` functions that manipulate strings.

For each question, some examples are given, but the functions must of course give correct results on other inputs!

1. Write a function “accumulate” that takes as inputs a string s and an integer n, and returns a string containing n times the string s, separated by spaces.
   - the call `accumulate("ATGC", 4)` must return `"ATGC ATGC ATGC ATGC"`.
2. Write a function “correct_sentence” that takes a string s as input and return a boolean: it will return `True` if and only if s starts with an upper case letter, then contains only spaces or lower case letters, then ends with a point.
   - the call `correct_sentence("Hel lo.")` must return `True`
   - the call `correct_sentence("School3.")` must return `False`
3. Write a function “points” that takes as input a string s (for this question, we can assume that any s given as input only contains letters and spaces), and return the number of points this string would give if played in a Scrabble® game, without bonus. The points of each letter are those of a French version, given by the following picture:



   - the call `points("ABCDEF")` must return `14`
   - the call `points("ZX")` must return `20`

Exercise 3 — Back to Reeborg’s World

If you’ve finished the previous exercises, please complete the three worlds Harvest 1, Harvest 2 and Harvest 3 (`https://reeborg.ca/reeborg.html`, then if those worlds are not available, choose “Other worlds” and click on “Go to introduction Reeborg”). You can also finish the “Home” and “Around” worlds before if you did not finish them previously.