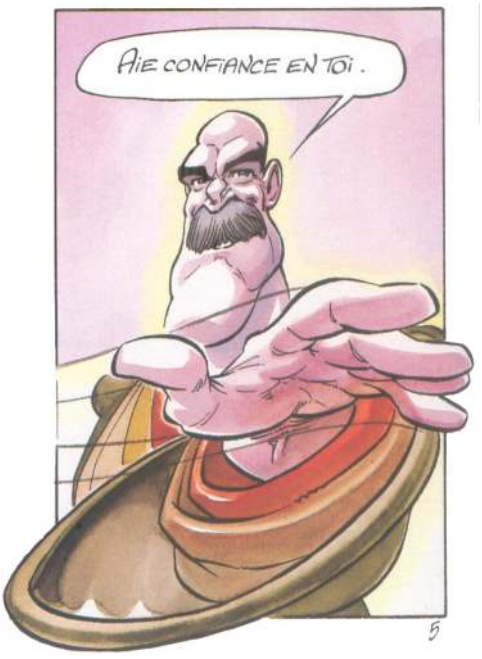
Algorithmics
Final Exam #1 (P1)

Undergraduate 1^{st} year S1#
EPITA

20 Juin 2019

Instructions (read it) :

- ☐ You must answer on **the answer sheets provided.**
  - No other sheet will be picked up. Keep your rough drafts.
  - Answer within the provided space. **Answers outside will not be marked**: Use your drafts!
  - Do not separate the sheets unless they can be re-stapled before handing in.
  - Penciled answers will not be marked.
- ☐ The presentation is negatively marked, which means that you are marked out of 20 points and the presentation points (maximum of 2) are taken off this grade.
- ☐ **Code:**
  - All code must be written in the language `Python` (no C, CAML, ALGO or anything else).
  - **Any `Python` code not indented will not be marked.**
  - All that you need (types, routines) is indicated in the **appendix** (last page)!
- ☐ Duration : 2h



Exercise 1 (Stack or queue? – *2 points*)

Values A, B, C, D, E and F are inserted, in this order, into an empty linear data structure. Indicate, for each output order given on the answer sheets, whether the structure in question may be: a stack, a queue (it can be both), or neither (neither a stack nor a queue).

Exercise 2 (Searching algorithms - *3 points*)

In this exercise, we focus on the *cost* of a positive search in a list sorted in increasing order (strictly).
Reminder: the *cost* of a searching algorithm is defined by its number of element comparisons

Let a list of n elements following an arithmetic progression of 1^{st} term u_0 and common difference $r > 0$.
For example with $u_0 = 0$, $r = 2$ and $n = 10$, the list is $\lambda = \{0, 2, 4, 6, 8, 10, 12, 14, 16, 18\}$.
Fill out the table for each question, with $n = 20$ and $n = 100$.

- Give a value for which the search costs 1. (e.g. u_5)
- Give **one** value for which the cost of the search is maximum (several answers possible). What is this cost ? (an integer)

Exercise 3 (Sorted – *3 points*)

Write the function `is_sorted(L)` that checks if the elements of the input list L are sorted in increasing order.
Examples of results:

```
>>> is_sorted([1, 5, 5, 12, 25])
True
>>> is_sorted([4, 3, 2, 1])
False
>>> is_sorted([4])
True
>>> is_sorted([])
True
```

Exercise 4 (Merge sort (Tri fusion) – *10 points*)

1. Write the function `partition` that splits a list into two (new) lists of almost identical lengths: one half in each list.

   Application examples:

```
>>> partition([15, 2, 0, 4, 5, 8, 2, 3, 12, 25])
([15, 2, 0, 4, 5], [8, 2, 3, 12, 25])
>>> partition([5, 3, 2, 8, 7, 1, 5, 4, 0, 6, 1])
([5, 3, 2, 8, 7], [1, 5, 4, 0, 6, 1])
```

2. Write the function `merge` that merges two lists, sorted in increasing order, into one new sorted list.

   Application example:

```
>>> merge([1, 5, 8], [2, 3, 4, 8])
[1, 2, 3, 4, 5, 8, 8]
```

3. To sort a list `L`, we proceed (recursively) as follows:

   ▷ A list of length < 2 is sorted.

   ▷ A list of length ≥ 2:
   - the list is split into two lists `L1` and `L2` of almost identical lengths;
   - the two lists `L1` and `L2` are sorted recursively;
   - finally, the two lists `L1` and `L2` are merged into one sorted list.

   Use the two previous functions (written or not) to write the function `sort` that sorts a list in increasing order (not "in place": the function builds and returns a new list.)

   Application example:

```
>>> sort([5, 3, 2, 8, 7, 1, 5, 4, 0, 6, 1])
[0, 1, 1, 2, 3, 4, 5, 5, 6, 7, 8]
```

Exercise 5 (What is it? – *3 points*)

Let the function `what` be :

```
def what(L, x):
    i = 0
    n = len(L)
    while i < n and x > L[i]:
        i += 1
    j = i
    while j < n and x == L[j]:
        j += 1
    c = j - i
    for k in range(i, n-c):
        L[k] = L[k+c]
    for k in range(c):
        L.pop()
```

1. Give the value of the list L after calling `what(`l`,` x`)` with:
   (a) L `= [1, 2, 3, 4, 5, 6, 7]` and x `= 2`
   (b) L `= [1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 5, 5, 5]` and x `= 3`
   (c) L `= [1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 5, 5, 5]` and x `= 5`
   (d) L `= [1, 3, 5, 7, 9]` and x `= 2`
2. Let L be a list of integers sorted in increasing order and x an integer. What is `what(`L`,` x`)` doing ?

Appendix: Authorised functions and methods

You can use the methods `append`, `pop` (without argument only) and the function `len` on lists:

```
>>> help(list.append)
Help on method_descriptor:    append(...)
    L.append(object) -> None -- append object to end of L

>>> help(len)
Help on built-in function len in module builtins:    len(...)
    len(object)
    Return the number of items of a sequence or collection.

>>> help(list.pop)  # cannot be use with index here...
Help on method_descriptor:
pop(...)
    L.pop() -> item -- remove and return last item.
    Raises IndexError if list is empty.
```

No operators are allowed on lists (`+`, `*`, `==` ...).

You can also use the function `range` and `raise` to raise exceptions. Reminder:

```
>>> for i in range(10):
...     print(i, end=' ')
0 1 2 3 4 5 6 7 8 9

>>> for i in range(5, 10):
...     print(i, end=' ')
5 6 7 8 9

>>> raise Exception("blabla")
...
Exception: blabla
```

Your functions

You can write your own functions as long as they are documented (we have to know what they do).
In any case, the last written function should be the one which answers the question.