Lists

1 *Iterative List* → Python

Abstract data type: iterative list λ : List	**Python: type list** `type(L) = list`
$\lambda \leftarrow$ *emptylist*	`L = []`
length(λ)	`len(L)`
nth(λ, k)	`L[k]`

Exercise 1.1 (ALGO ↦ Python)

Translate the following algorithms in Python.

1.
```
function count(Element x, List λ) : integer
   variables
      integer    i, cpt
begin
   cpt ← 0
   for i ← 1 to length(λ) do
      if x = nth(λ, i) then
         cpt ← cpt + 1
   end for
   return cpt
end
```

2.
```
function is-present(Element x, List λ) : boolean
   variables
      integer    i
begin
   i ← 1
   while (i <= length(λ)) and (x <> nth(λ, i)) do
      i ← i + 1
   end while
   return (i <= length(λ))
end
```

3. Modify the last fonction: it returns the position of the first x found and the list is sorted (increasing order)

Exercise 1.2 (Build a list)

```
>>> help(list.append)
Help on method_descriptor:
append(...)
    L.append(object) -> None -- append object to end
>>> L = []
>>> L.append(1)
>>> L.append(2)
>>> L.append(3)
>>> L
[1, 2, 3]
```

Write a function that builds a new list with n values *val*.

Exercise 1.3 (Abstract Type $\mapsto$ Python)

Implement the following operations in Python.

1. *The delete operation*

OPERATIONS
$delete$: List × Integer → List

PRECONDITIONS
$delete(\lambda,\ k)$ **is-defined-iaoi** $1 \leq k \leq length(\lambda)$

AXIOMS
$\lambda \neq emptylist\ \&\ 1 \leq k \leq length(\lambda) \Rightarrow length(delete(\lambda, k)) = length(\lambda) - 1$
$\lambda \neq emptylist\ \&\ 1 \leq k \leq length(\lambda)\ \&\ 1 \leq i < k$
$\Rightarrow nth(delete(\lambda, k), i) = nth(\lambda, i)$
$\lambda \neq emptylist\ \&\ 1 \leq k \leq length(\lambda)\ \&\ k \leq i \leq length(\lambda) - 1$
$\Rightarrow nth(delete(\lambda, k), i) = nth(\lambda, i + 1)$

WITH
λ : List
k, i : Integer

2. *The insert operation*

OPERATIONS
$insert$: List × Integer × Element → List

PRECONDITIONS
$insert(\lambda,\ k,\ e)$ **is-defined-iaoi** $1 \leq k \leq length(\lambda) + 1$

AXIOMS
$1 \leq k \leq length(\lambda) + 1 \Rightarrow length(insert(\lambda, k, e)) = length(\lambda) + 1$
$1 \leq k \leq length(\lambda) + 1\ \&\ 1 \leq i < k \Rightarrow nth(insert(\lambda, k, e), i) = nth(\lambda, i)$
$1 \leq k \leq length(\lambda) + 1\ \&\ k = i \Rightarrow nth(insert(\lambda, k, e), i) = e$
$1 \leq k \leq length(\lambda) + 1\ \&\ k < i \leq length(\lambda) + 1$
$\Rightarrow nth(insert(\lambda, k, e), i) = nth(\lambda, i - 1)$

WITH
λ : List
k, i : Integer
e : Element

How these operations can be implemented "in-place"?

2 A Sense of Déjà Vu

Exercise 2.1 (Histogram)

1. Write a function that gives an histogram of characters in a given string: a list of length 256 that contains the number of occurrences of each letter in the string.
2. Write a function that calculates the number of different characters occurring in a string.
3. Write a function that returns the most frequent character in a string as well as its number of occurrences.

Exercise 2.2 (Eratosthenes)

Write a function that gives the list of all prime numbers up to a given value n. Use the "sieve of Eratosthenes" method (see wikipedia).

3 Order and sorts

Exercise 3.1 (Hill – *Final P1# 2017*)

A list is a hill if it consists of an increasing sequence (possibly empty) followed by a decreasing sequence (possibly empty).

Examples

The following lists are hills:

- 1, 4, 8, 12, 5, 2
- 12, 25, 40, 52
- 15, 8, 3

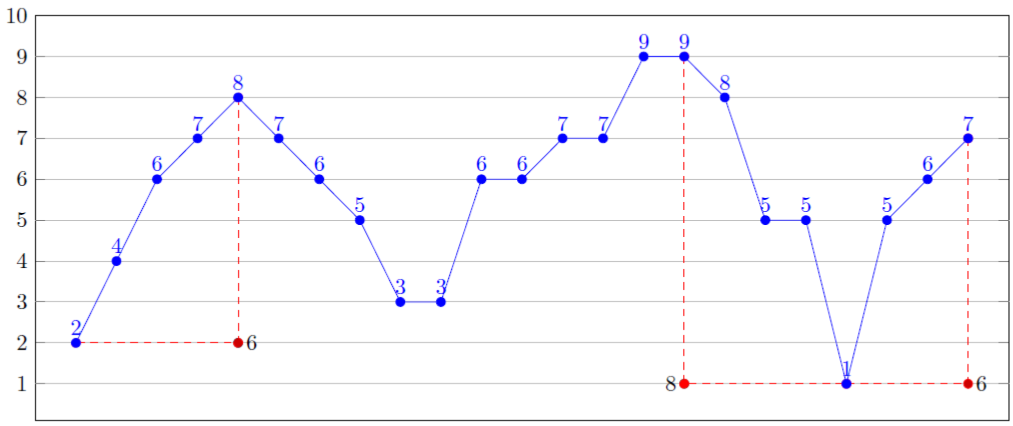
The following lists are not hills:

- 1, 5, 10, 8, 6, 12
- 15, 12, 11, 9, 10, 14, 16

1. Write a function that determines whether a non empty list is a hill.
2. Write the function `hills_size` that computes the maximum height of the hills in an integer list.

```
hills_size([3,6,9,10])                                        # returns 7
hills_size([8,5,2])                                           # returns 6
hills_size([2,10])                                            # returns 8
hills_size([6,6,6,6])                                         # returns 0
hills_size([2,4,6,7,8,7,6,5,3,3,6,6,7,7,9,9,8,5,5,1,5,6,7])   # returns 8
```

Visual example of the last tested list:



Exercise 3.2 (Shared - *Final P1# 2017*)

Let L_1 and L_2 be two lists ordered in strictly increasing order. Write the function `shared(`L_1`, `L_2`)` that computes the number of values that are the same in both lists L_1 and L_2.

Examples:

	0	1	2	3	4	5	6	7	8	9
L_1	15	12	7	5	4	-1	-2	-3	-6	-8

	0	1	2	3	4	5	6	7	8
L_2	8	7	4	3	0	-2	-3	-5	-6

the number of shared values in both lists L_1 and L_2 is 5.

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

>>> shared([15,12,7,5,4], [3,2,0,-2,-3,-5,-6])
0

```

Modify the function so that it builds the list shared values.

Exercise 3.3 (Select Sort (Tri par sélection))

1. Write the function `minimum` that returns the position of the minimum value in a list.
2. Use the previous function (you can modify it if necessary) to write a function that sorts a list in increasing order.

Exercise 3.4 (Insertion Sort (Tri par insertion))

1. Write a function that inserts an element x at its position in a sorted list.
2. Use the previous function to write a function that sorts a list.

Exercise 3.5 (Bubble Sort (Tri à bulles))

Implement in Python the *bubble sort*. (We might modify this sort to obtain a *sharker sort*…)

4 The Problem: Kaprekar

Kaprekar Routine:

Let a p-digit integer.

- Arrange the digits in descending and ascending order
- Subtract and add possibly '0' to obtain a p-digit number.
- Do this again with the result.

Comments:

- Here, we always compute p-digit numbers. That is, if a result is less than 10^{p-1} (for example 999 when $p = 4$), the two new numbers will be build from the number digits completed by 0 (9990).
- The digits from the initial number have to be not all equal.
- The Kaprekar routine can be done with any number of digits p. Depending on p, the sequence will stop on the same value, or will reach a cycle.

Write a Python script that performs the Kaprekar routine. The different values have to be displayed.