Lists

1 *Iterative List* → Python

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

In ALGO , list indexes start to 1, they start to 0 in Python.

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 if
   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
```

Exercise 1.2 (Abstract Type ↦ Python)

Implement the following operation in Python.

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

How can this operation be implemented "in-place"?

2 Construction

Exercise 2.1 (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 2.2 (List decomposition)

Write a function `get_even_odd(L)` that builds and returns a pair of lists `L1,L2` such that `L1` contains the elements in even positions in `L` and `L2` the elements in odd positions in `L`.

Application examples:

```
>>> get_even_odd([0])
([0], [])
>>> get_even_odd([4, 8, 6, 4, 2])
([4, 6, 2], [8, 4])
>>> get_even_odd([0, 1, 1])
([0, 1], [1])
>>> get_even_odd([4, 0, 0, 4])
([0, 4], [4, 0])
```

Exercise 2.3 (List sum)

Write a function `get_sum(L)` that computes the sum of the values of the list `L` and returns this integer as a list with the digits in increasing order of their positions (reverse order).

Application examples:

```
>>> get_sum([4, 0, 0, 4])
[8] # = 8
>>> get_sum([2, 6, 4])
[2, 1] # = 12
>>> get_sum([2, 5, 9])
[6, 1] # = 16
>>> get_sum([3, 1, 1])
[5] # = 5
>>> get_sum([9])
[9] # = 9
```

3 A Sense of Déjà Vu

Exercise 3.1 (Histogram)

1. Write a function that gives a 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.

4 Order

Exercise 4.1 (Search in a sorted list)

Modify the last fonction of exercise 1.1 (is-present): it returns the position of the first x found (-1 otherwise) and the list is sorted (increasing order).

Exercise 4.2 (Indexes search – *P1 - 01/2021*)

Write the function `search_indexes(L, val)` that returns the values of the first and last indexes of element *val* in the sorted in increasing order list L.

If element *val* is not found in the list, the function returns the pair `(-1, -1)`.

Application examples:

```
>>> search_indexes([1, 4, 12, 12, 42], 4)
(1, 1)

>>> search_indexes([1, 4, 12, 12, 42], 15)
(-1, -1)

>>> search_indexes([1, 4, 12, 12, 42], 12)
(2, 3)

>>> search_indexes([4, 4, 4], 4)
(0, 2)
```

Exercise 4.3 (Insertion in a sorted list)

Write the function `insert(L, x)` that removes the value x, if it exists, from the list L sorted in strictly increasing order. The function returns a boolean that indicates whether the deletion occurred.

Application examples:

```
>>> L = [1, 4, 5, 7, 8, 12, 15]
>>> insert(L, 7)
False
>>> L
[1, 4, 5, 7, 8, 12, 15]
>>> insert(L, 11)
True
>>> L
[1, 4, 5, 7, 8, 11, 12, 15]
```

5 Problems

Exercise 5.1 (Divisibility by 11 - Digit sum)

The integers will be given as lists. This list will be given with the digits in increasing order of their positions (reverse order), the last position corresponding to the first digit, the before last position corresponding to the second digit, etc...

Example: $12345 \rightarrow [5, 4, 3, 2, 1]$

An integer is divisible by 11 if and only if the difference between the sum of its digits at odd position and the sum of its digits at even position is divisible by 11.

For example, 915123 is divisible by 11 since (9 + 5 + 2) - (1 + 1 + 3) = 11.

Write a function `div11_sum(L)` that from an integer represented by a list `L` with the digits in increasing order of their positions (reverse order) checks if it is divisible by 11 using the procedure described at the beginning of the exercise. This procedure can be applied until a two digits number is obtained to know if the initial number is divisible by 11.

Application examples:

```
>>> div11_sum([0]) # = 0
True
>>> div11_sum([1, 1]) # = 11
True
>>> div11_sum([3, 2, 1, 5, 1, 9]) # = 915123
True
```

Some previous functions must be used and the function `get_diff(L1,L2)` specified below:

The function `get_diff(L1,L2)` takes as parameters two integers represented as lists with the digits in increasing order of their positions (reverse order) and returns the absolute value of the difference of those two integers as a list with the digits in increasing order of their positions (reverse order).

```
>>> get_diff([6, 1], [5]) # = |16 - 5|
[1, 1] # = 11
>>> get_diff([6, 2], [9]) # = |26 - 9|
[7, 1] # = 17
>>> get_diff([5, 3, 4], [9, 5]) # = |435 - 59|
[6, 7, 3] # = 376
>>> get_diff([9, 5], [5, 3, 4]) # = |59 - 435|
[6, 7, 3] # = 376
```

Bonus: Write the function `get_diff(L1,L2)`.

Exercise 5.2 (Bonus: 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).