Last name	
First name	
Group	

Grade	/ 4
-------	-----

# Algorithmics

	Algorithmics
	Midterm Exam 1 - Part 1
	Undergraduate $1^{st}$ year S1
	EPITA
	Oct. 2019
	This is the part 1 of the subject - You have to give back the two parts!
	You must answer on this subject.
	<ul> <li>Answer within the provided space. Answers outside will not be marked.</li> <li>Penciled answers will not be marked.</li> </ul>
	The presentation is marked.
Exe	$condom{r}{c}$ (A little coursework – 4 $points$ )
1.	What does an internal operation return?
2.	How do we call an operation used to specify the domain of definition of another one?
3.	What problems arise during the making of the set of axioms?
4.	Which areas make up the signature of an abstract type?
5	How do we write axioms?
ο.	TO TO THE WHOLE.

Last name	
First name	
Group	

Grade	/ 16
-------	------

## Algorithmics Midterm Exam 1 - Part 2

Undergraduate  $1^{st}$  year S1 EPITA Oct. 2019

- □ This is the part 2 of the subject You have to give back the two parts!
- $\square$  You must answer on **this subject.** 
  - Answer within the provided space. Answers outside will not be marked.
  - Penciled answers will not be marked.
- □ CAML:
  - All Caml code not indented will not be marked.
  - In the absence of any indication in the document, the only functions that you can use are failwith and invalid\_arg (no other predefined function of CAML).
  - Any CAML code must be followed by the result of its evaluation: the CAML answer.
- $\square$  The presentation is marked.

#### Exercise 2 (Dominoes - 4 points)

For this exercise, we are interested in the Domino game:

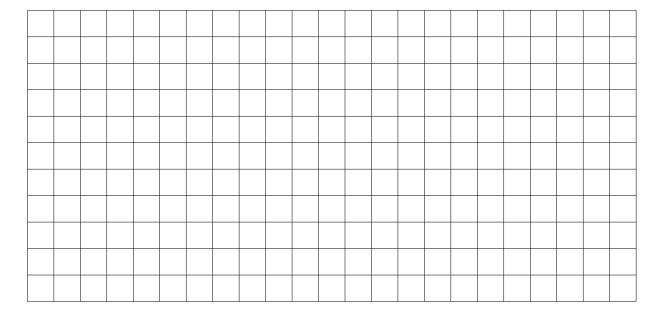
- A domino will be represented by a pair of integers (a, b)
- A sequence of dominoes (chain) will be a list of pairs of integers  $[(a_1, b_1); (a_2, b_2); \cdots; (a_n, b_n)]$ .

A domino chain is considered valid if the neighboring dominoes have the same number of points (see examples below).

Write the function is\_dominoes that checks if a chain of dominoes is valid.

Application examples:

```
# is_dominoes [] ;;
- : bool = true
# is_dominoes [(1,2); (2,3); (3,3); (3,6)] ;;
- : bool = true
# is_dominoes [(2,3); (2,4); (1,4)] ;;
- : bool = false
```

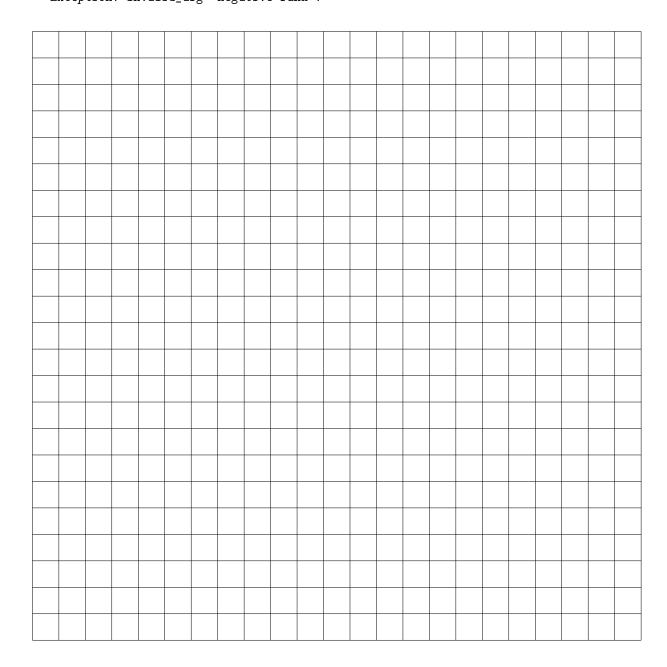


## Exercise 3 (Deletion at rank i-5 points)

Write the function  $remove_nth\ i\ list$  that deletes the value at the rank i in the list list. The function has to raise an exception  $Invalid_argument$  if i is negative or zero, or an exception Failure if the list is too short.

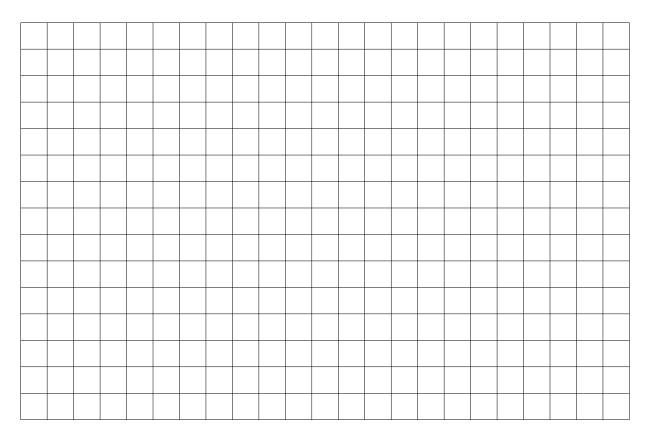
### $Application\ examples:$

```
# remove_nth 5 [1; 2; 3; 4; 5; 6; 7; 8; 9] ;;
- : int list = [1; 2; 3; 4; 6; 7; 8; 9]
# remove_nth 10 [1; 2; 3; 4; 5; 6; 7; 8; 9] ;;
Exception: Failure "out of bound".
# remove_nth (-2) [1; 2; 3; 4; 5; 6; 7; 8; 9] ;;
Exception: Invalid_arg "negative rank".
```



## Exercise 4 (for all 2-5 points)

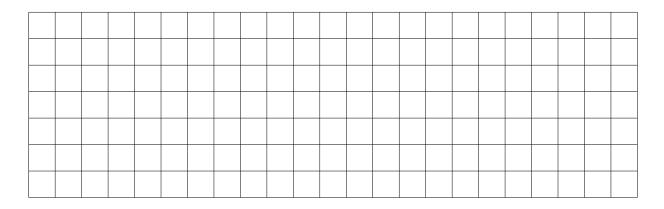
- 1. Write the Caml function for\_all2 with the following specifications:
  - It takes a two-argument boolean function (a predicate): p, and two lists:  $[a_1; a_2; \dots; a_n]$  and  $[b_1; b_2; \dots; b_n]$ , as parameters.
  - It checks whether all pairs of elements  $a_i$   $b_i$  satisfy the predicate p.
  - If a pair such that  $p \ a_i \ b_i$  is false is found, it returns false. Otherwise, it raises Invalid\_argument if the two lists have different lengths.



2. Use the function for\_all2 to define a function that verifies whether two integer lists are "quasi identical": difference between the values at the same place can not exceed 1. If no pair with a difference over one has been found and if the two lists have different lengths, an exception is raised.

Examples of utilisation:

```
# almost [1; 2; 3; 4; 5] [1; 3; 2; 4; 6];;
- : bool = true
# almost [1; 2; 3; 4; 5] [1; 4; 3; 4; 5];;
- : bool = false
```



## Exercise 5 (Mystery - 2 points)

Give the results of the successive evaluations of the following phrases.

```
# let mystery a b =
    let rec what = function
        ([], _) -> true
        | (_, []) -> false
        | (e::11, f::12) -> (e = f) && what (11, 12)
    in
    let rec is_that x y = match y with
        [] -> 0
        | e::q -> (if what (x, y) then 1 else 0) + (is_that x q)
    in
        is_that a b ;;
```

```
# mystery [1; 2] [1; 2] ;;
```

```
# mystery [1; 2] [1; 1; 2; 3; 3; 1; 2; 3] ;;
```

```
# mystery [1; 2] [2; 1] ;;
```



