Université Ibn Khaldoun Tiaret 18/12/2023
Faculté MI : Département d'informatique
Module : Programmation fonctionnelle (1ère Master GL)
Chargé du module : A. Chadli.

Examen de contrôle (durée : 1H30)

Exercice 1 : (5 pts)

Donnez la réponse de l'interpréteur ***Ocaml*** après l'exécution de chaque commande :

```
# let x = 3;;
  ..............
# x + x;;
  ..............
# let y = x;;
  ..............
# let x = 4;;
  ..............
# y;;
  ..............
```

```
# let a = 4 in a * a + 1;;
- : int = 17
# a;;
..............
```

```
# let a = 5;;
val a : int = 5
# let a = 4 in a * a;;
- : int = 16
# a;;
..............
```

Exercice 2 : (3 pts)

Ecrire une fonction "Contain" qui étant donné une sous chaine « a » et une chaine « b », retourne TRUE si la chaine « b » commence avec la sous chaine « a » et FALSE dans le cas contraire.

Exercice 3 : (4 pts)

Ecrire une fonction « makeliste » qui étant donné un nombre « n » crée la liste [n, n-1, n-2, …, 1,0]

Exercice 4 : (4 pts)

Ecrire la même fonction que celle de l'exercice 3 qui étant donné un entier « n » renvoie la liste [1 ;2 ;3 ; … ;n-2 ;n-1 ;n]

Exercice 5 : (4 pts)

Ecrire une fonction récursive "TRICROISSANT" qui renvoie TRUE si la liste d'entiers donnée en argument est triée dans l'ordre croissant, FALSE si elle n'est pas triée et affiche une erreur si la liste est vide.

Exemple : TRICROISSANT [1 ;1 ;2 ;3 ;4 ;5 ;5] -> TRUE , TRICROISSANT [1 ;1 ;2 ;3 ;4 ;6 ;5] -> FALSE

Solution du contrôle 2023/2024

Exercice 1 : (5 pts)

```
# let x = 3;;
val x : int = 3
# x + x;;
- : int = 6
# let y = x;;
val y : int = 3
# let x = 4;;
val x : int = 4
# y;;
- : int = 3
```

Exe

```
# let a = 4 in a * a + 1;;
- : int = 17
# a;;
  ^
  Error: Unbound value a
```

```
# let a = 5;;
val a : int = 5
# let a = 4 in a * a;;
- : int = 16
# a;;
- : int = 5
```

```
# let contain a b = if String.sub b 0 (String.length a) = a then true else false ;;
val contain : string -> string -> bool = <fun>
# contain "abc" "abcdef" ;;
- : bool = true
# contain "abc" "abdef" ;;
- : bool = false
```

Exercice 3 : (4 pts)

```
# let rec enliste n = match n with
  |0 -> []
  |_ -> n :: enliste (n-1) ;;
val enliste : int -> int list = <fun>
# enliste 4 ;;
- : int list = [4; 3; 2; 1]
```

Exercice 4 : (4 pts)

```
# let rec enliste n = match n with
  |0 -> []
  |_ -> enliste (n-1) @ [n] ;;
val enliste : int -> int list = <fun>
# enliste 4 ;;
- : int list = [1; 2; 3; 4]
```

Exercice 5 : (4 pts)

```
# let rec trie l = match l with
  | [] -> failwith "liste vide"
  | [a] -> true
  | h1::h2::t -> if h1<=h2 then trie (h2::t) else false ;;
val trie : 'a list -> bool = <fun>
# trie [1;1;2;3;4;5;5] ;;
- : bool = true
# trie [1;1;2;3;4;6;5] ;;
- : bool = false
```