Algorithmique Caml

Info-sup S1 Epita

Examen B1

octobre 2026

Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  Indiquez de manière lisible votre NOM (en MAJUSCULEs), prénom, UID et classe sur chaque feuille.
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées**.
  - Ne pas utiliser de crayon de papier, de stylo effaçable, de correcteur blanc (tippex, blanco...).
- ☐ Le barème est donné à titre indicatif. Des points de présentation peuvent être retirés.
- ☐ Caml :
  - Tout code Caml non indenté ne sera pas corrigé.
  - En l'absence d'indication dans l'énoncé, les seules fonctions que vous pouvez utiliser sont `failwith` et `invalid_arg` (aucune autre fonction prédéfinie de Caml).
  - Une seule version doit être présentée pour chaque fonction à écrire.
  - Vos fonctions doivent impérativement respecter les exemples d'applications donnés.
  - Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).
    Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.
  - **Tout code Caml doit être suivi du résultat de son évaluation (fait partie de la note) : la réponse de Caml.**
- ☐ La présentation est notée.
- ☐ Durée : 1h30

Exercice 1 (`nb_peaks` ≃ *4 points*)

Écrire la fonction `nb_peaks lst` qui renvoie le nombre de « pics » présents dans la liste `lst`. Un élément de la liste est considéré comme un pic s'il est strictement supérieur à l'élément qui le précède *et* à l'élément qui le suit.

Exemples d'applications :

```
# nb_peaks [];;
- : int = 0

# nb_peaks [1];;
- : int = 0

# nb_peaks [1; 2];;
- : int = 0

# nb_peaks [1; 2; 3; 4; 5];;
- : int = 0

# nb_peaks [1; 5; 2; 7; 3];;
- : int = 2

# nb_peaks [5; 1; 5; 1; 5];;
- : int = 1
```

Exercice 2 (`longest_equal` ≃ ***5 points***)

Écrire la fonction `longest_equal lst` qui renvoie la longueur de la plus longue suite d'éléments identiques dans la liste `lst`.

Exemples d'applications :

```
# longest_equal [];;
- : int = 0

# longest_equal [1; 2; 3; 4];;
- : int = 1

# longest_equal [1; 1; 2; 2; 2; 3; 3];;
- : int = 3

# longest_equal [5; 5; 5; 5];;
- : int = 4
```

Exercice 3 (`insert_end` ≃ ***7 points***)

Écrire la fonction `insert_end x i lst` qui insère l'élément `x` dans la liste `lst` à la position `i`, cette position étant comptée depuis la *fin* de la liste (la position 1 correspondant à la toute fin de la liste, après le dernier élément).
On suppose que `i` est correct pour sa borne supérieure, c'est-à-dire que `i` est inférieur ou égal à la longueur de `lst` plus 1 (la position longueur + 1 correspondant à la tête de liste).
La fonction déclenche une exception `Invalid_argument` si `i` est inférieur ou égal à 0.

Exemples d'applications :

```
# insert_end 9 0 [1; 2; 3; 4];;
Exception: Invalid_argument "".

# insert_end 9 1 [1; 2; 3; 4];;
- : int list = [1; 2; 3; 4; 9]

# insert_end 9 2 [1; 2; 3; 4];;
- : int list = [1; 2; 3; 9; 4]

# insert_end 9 3 [1; 2; 3; 4];;
- : int list = [1; 2; 9; 3; 4]

# insert_end 9 4 [1; 2; 3; 4];;
- : int list = [1; 9; 2; 3; 4]

# insert_end 9 5 [1; 2; 3; 4];;
- : int list = [9; 1; 2; 3; 4]
```

Exercice 4 (`Mystery` $\simeq$ ***4 points***)

Soit la fonction `mystery` définie ci-dessous

```
let rec mystery f n x =
  match n with
  | 0 -> []
  | n -> x :: mystery f (n-1) (f x)
;;
```

1. Donner le type de la fonction `mystery`.

2. On sait que l'évaluation d'un appel à `mystery` donne le résultat suivant.

   ```
   # mystery ...;;
   - : bool list = [true; false; true; false; true]
   ```

   Écrire un appel à `mystery` produisant cette évaluation.

3. On sait que l'évaluation d'un appel à `mystery` donne le résultat suivant.

   ```
   # mystery ...;;
   - : (int * int) list = [(10, 4); (14, 6); (20, 8); (28, 10)]
   ```

   Écrire un appel à `mystery` produisant cette évaluation.