Examen

NOM : **Prénom :**

Consignes : Téléphones, ordinateurs et calculatrices interdits.
Documents de cours/TD/TP autorisés.
Aucune question possible : en cas d'ambiguïté dans le sujet, signaler le choix fait sur la copie.

Conseil : Lire complètement le sujet avant de commencer à y répondre.

Barème indicatif :

exercice	1	2	3	4
points	6	10	2	2

Exercice 1 : Calculer une valeur sur les éléments d'une liste
Dans cet exercice, on utilisera le langage Python.
1. *Somme.* Écrire une fonction `somme` qui retourne la somme des éléments d'une liste. On n'utilisera pas les fonctions prédéfinies de Python comme par exemple **`sum`**.

Par exemple, appliquée sur la liste [1, -1, 2, 3], cette fonction retournera 5.

2. *Produit.* Écrire une fonction `produit` qui retourne le produit des éléments d'une liste. On n'utilisera pas les fonctions prédéfinies de Python comme par exemple `Math.prod`.

Par exemple, appliquée sur la liste [1, -1, 2, 3], cette fonction retournera -6.

3. *Généraliser.* Les deux fonctions précédentes sont sur le même principe : leur code a la même structure. Proposer une fonction que l'on peut appeler `reduce` qui généralise ces deux fonctions.
Indication : Il s'agit de transformer en paramètres de cette fonction les éléments différents dans les codes des deux fonctions `somme` et `produit`.
4. *Utiliser `reduce`.*
4.1. Réécrire `somme` en utilisant `reduce`.
4.2. En utilisant `reduce`, définir une fonction `frequence` qui calcule la fréquence (le nombre d'occurrences) d'un élément dans une liste. La fréquence de 3 dans [3, 1, 2, 3] est 2.

Exercice 2 : Les expressions arithmétiques revisitées
Dans cet exercice nous modélisons en OCaml, Python objet et Prolog des expressions arithmétiques simplifiées, limitées à la somme et à la différence. Nous proposons la modélisation suivante, en OCaml, des expressions arithmétiques.

```
type operateur =
    | Somme
    | Difference
;;

type expression =
    Constante of float
    | ExpressionBinaire of (operateur * expression * expression)
;;
```

1. Définir en OCaml l'expression $5 - (3 + 4)$ que l'on appellera e1.
2. Modélisons les expressions en objet en gardant la même logique.
2.1. Dessiner le diagramme de classe qui correspond à ces expressions artihmétiques.
2.2. Écrire en Python l'initialisation de la constante e1 ci-dessus.
3. Définir un prédicat Prolog d'arité 1 appelé e1 qui permet d'unifier sa variable avec l'expression e1 précédente.
4. *Nombres tirets.* On souhaite faire un premier traitement sur ces expressions qui consiste à obtenir le nombre de tirets utilisés dans les opérateurs d'une expressions.

Ce traitement peut être écrit en OCaml par la fonction `nombre_traits` suivante.

```
let rec nombre_traits e =
    let rec nombre_traits_operateur op gauche droite
        match op with
        | Somme -> 2
        | Difference -> 1
    in
        match e with
        | ExpressionBinaire (op, e1, e2) -> (nombre_traits_operateur op)
                + (nombre_traits e1) + (nombre_traits e2)
        | _ -> 0
;;
```

4.1. Expliquer le code ci-dessus.
4.2. Indiquer le résultat de l'application de ce traitement sur l'expression e1.
4.3. *En objet.* Expliquer comment écrire ce traitement dans la version objet et donner quelques exemples de code significatifs.
4.4. *En Prolog.* Écrire en Prolog ce traitement.
5. *Évaluation d'une expression.* On s'intéresse à un nouveau traitement qui consiste à obtenir la valeur d'une expression.
5.1. *En fonctionnel.* Écrire ce traitement en OCaml.
5.2. Écrire ce traitement en Python. On indiquera le code à ajouter et où l'ajouter.
5.3. *En déclaratif.* Écrire ce traitement en Prolog.

Exercice 3 Écrire en Prolog un prédicat max qui associe à une liste son plus grand élément.

Exercice 4 Indiquer les solutions qui seront trouvées par Prolog pour la requête `?- b(X).` sur les règles suivantes. Expliquer comment Prolog procède.

```
a(1).
a(3).
a(3).
a(2).
a(6).
a(5).

b(X) :- a(X), X > 2, X < 6.
```