# 1 Programmation de la fonction sinus

On propose de calculer la fonction sinus en utilisant son développement en série limitée, rappelé ci-après :

$$sin\ x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ...$$

On pose

$$S_n(x) = \sum_{n>0} (-1)^{(n-1)} . \frac{x^{(2n-1)}}{(2n-1)!}$$

1. Écrire une fonction récursive Ocaml `serie_sinus` de profil `float -> int -> float` qui calcule $S_n(x)$ pour des valeurs de x et n passées en paramètre. Il est suggéré de définir d'abord une fonction `terme` de profil `float -> int -> float` qui calcule un terme de la somme.
2. Écrire une fonction `diff` de profil `float -> int -> float` qui calcule la différence entre `sin x` (où `sin` est la fonction sinus prédéfinie dans la bibliothèque numérique Ocaml) et `serie_sinus` pour des valeurs de x et n passées en paramètre.

# 2 Polynômes creux

On décide de représenter un polynôme $P(X) = a_0 + a_1 X + \ldots a_n X^n$ par une liste de couples (coeff, degré) représentant les monômes non nuls de $P$.

Exemple : $P(X) = 12X^{17} - 4X^8 + 0.75X^3 - X + 1.3$ est représenté par la liste caml `[(12., 17); (-4., 8); (0.75, 3); (-1., 1); (1.3, 0)]`. De plus on impose que la liste soit triée par degrés décroissants, et on représente le polynôme nul par `[(0., 0)]`.

1. Écrire une fonction `decroiss` de type `(float * int) list -> bool` qui vérifie que le polynôme donné en entrée est bien trié par degrés décroissants.
2. Écrire une fonction `repr` de type `(float * int) list -> string` qui calcule une chaîne représentant le polynôme donné en entrée. Exemple :

```
# repr [(1., 42); (-17., 0)];;
- : string = "1.*X^42-17.*X^0"
```

3. Écrire une fonction `somme` qui calcule la somme de deux polynômes donnés en argument.
4. Écrire une fonction `evalue` qui renvoie la fonction polynôme de type `float -> float` associée au polynôme donné en argument. Par exemple `evalue [(1., 1)]` devra retourner la fonction $f(x) = x$.
5. Écrire une fonction `mult1` qui calcule la multiplication d'un polynôme par un monôme.
6. Écrire une fonction `multpol` qui calcule la multiplication de deux polynômes.
7. Écrire une fonction `multlist` qui calcule le produit de tous les polynômes éléments de la liste donnée en argument.

## 3 Table de multiplication

On veut écrire la fonction `multab : int -> int list` qui calcule la table de multiplication d'un entier passé en argument. A titre d'exemple

```
 multab 7;;
- : int list = [7; 14; 21; 28; 35; 42; 49; 56; 63; 70]
```

Pour cela,

1. on définit dans un premier temps la fonction `serie :int -> int -> int list`, qui à deux valeurs entières `m` et `n` passées en paramètre associe une liste des entiers de `m` à `n`. Par exemple,

   ```
   serie 1 4;;
   - : int list = [1;2;3;4]
   ```

2. à partir de la fonction `serie` et en utilisant la fonction `List.map`, proposer une définition pour la fonction `multab`.