Exercice 1 : Soit la fonction suivante :

```
let rec s n = match n with
  | 0 -> 0
  | n -> n + s(n-1)
;;
```

1. Quel est le type de la fonction ?
2. Que calcule-t-elle ?
3. En utilisant un accumulateur, la rendre récursive terminale.

Exercice 2 :

1. Écrire une fonction récursive `puissance` sur les entiers, utilisant l'égalité :

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ x \times x^{n-1} & \text{sinon} \end{cases}$$

2. Donner une version utilisant une fonction récursive terminale interne (utiliser un accumulateur).
3. Écrire une fonction d'exponentiation rapide récursive, basée sur l'égalité suivante :

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ (x^2)^{\frac{n}{2}} & \text{si } n \text{ est pair} \\ x \times (x^2)^{\frac{n-1}{2}} & \text{si } n \text{ est impair} \end{cases}$$

Exercice 3 : On considère la fonction récursive suivante :

```
let rec u n = match n with
  | 0 -> 1.
  | _ -> (u(n-1)) /. (3. +. u(n-1))
;;
```

1. Quel est le nombre d'appels récursif nécessaire pour calculer `u n` ?
2. Cette complexité vous semble-t-elle sous optimale ?
3. Comment peut on résoudre ce problème ?

Exercice 4 : Soit le code **OCaml** suivant :

```
let rec u n =
  if n = 0 then 2
  else if n = 1 then 1
    else u(n-1) - 2*u(n-2)
;;
```

1. Quelle suite est calculée ?
2. Proposer une version utilisant le filtrage par motifs.

Exercice 5 : Implémenter l'algorithme d'Euclide, dont la définition par récurrence est rappelée ci-dessous :

- $\text{pgcd}(u, 0) = u$;
- $\text{pgcd}(u, v) = \text{pgcd}(v, u \mod v)$ sinon.

Exercice 6 : Écrire une fonction récursive donnant le nombre de chiffres en base b d'un entier n strictement positif (on pourra convenir que zéro n'a aucun chiffre).

Exercice 7 : On considère la fonction **OCaml** suivante. Quel est son type ? Que calcule-t-elle ?

```
let rec mystere f n m = match n with
  | n when n > m -> 1.
  | _ -> f(float_of_int n) *. mystere f (n+1) m
;;
```

Exercice 8 : Il est possible de **tracer** les appels à une fonction `f` à l'aide de `trace`, qui s'utilise ainsi :

```
#trace f ;;
```

Coder une fonction récursive `fibo n` calculant le n-ième terme de la suite de Fibonacci définie par :

$$F_n = \begin{cases} 1 & \text{si } n = 0 \text{ ou } n = 1 \\ F_{n-1} + F_{n-2} & \text{sinon} \end{cases}$$

On fera deux appels récursifs. Regarder ce que donne le calcule de F_6 (après avoir tracé la fonction).

Exercice 9 (Détection de cycles par algorithme de Floyd) : À partir d'un ensemble fini E, d'une fonction $f : E \to E$ et d'un élément $x_0 \in E$, on appelle suite des itérés de x_0 la suite des valeurs $(x_n)n \in \mathbb{N}$ définie par la relation de récurrence : $x_{n+1} = f(x_n)$.

Puisque E est supposé fini, cette suite va atteindre deux fois la même valeur : il existe $i < j$ tel que $x_i = x_j$. Une fois que cette collision est obtenue, la suite des valeurs va répéter le cycle des valeurs de x_i à x_{j-1}. Nous allons nous intéresser au problème de la recherche de ce cycle, autrement dit déterminer les valeurs $\mu = i$ de la pré-période et $\lambda = j - i$ de la période du cycle minimal.

Par exemple, pour $f : x \mapsto (x^2 + 92) \bmod 32069$ et $x_0 = 33$ on trouve $\lambda = 8$ et $\mu = 313$. Ces valeurs pourront être utilisées pour tester les fonctions que vous écrirez.

1. Sachant que x_n est égal à x_0 si $n = 0$ et à $f(x_{n-1})$ sinon, rédiger une fonction `itere` de signature `('a -> 'a) -> 'a -> int -> 'a` qui prend en argument la fonction f , la valeur de x_0 et un entier $n \in \mathbb{N}$ et qui retourne la valeur de x_n.
2. On peut aussi remarquer que si $(\tilde{x}_n)_{n\in\mathbb{N}}$ est la suite des itérés de $f(x_0)$ alors $x_n = \tilde{x}_{n-1}$. Exploiter cette remarque pour rédiger une seconde version de la fonction `itere`.
3. On considère la suite $(y_n)_{n\in\mathbb{N}}$ définie par $y_0 = x_0$ et $y_{n+1} = f(f(y_n))$, ainsi que le plus petit entier $i > 0$ vérifiant $x_i = y_i$. Rédiger une fonction récursive `floyd1 : ('a -> 'a) -> 'a -> 'a` qui prend en arguments la fonction f et la valeur x_0 et qui retourne la valeur de x_i.
4. Modifier la fonction précédente pour obtenir une fonction `floyd2 : ('a -> 'a) -> 'a -> int` qui retourne cette fois la valeur de l'entier i.
5. Expliquer pourquoi ces algorithmes se terminent et pourquoi la valeur de i correspond au plus petit multiple de λ qui soit supérieur ou égal à μ.
6. Quel entier obtient-on si on applique la fonction `floyd2` à f et à x_i ? En déduire une fonction `periode : ('a -> 'a) -> 'a -> int` qui retourne la période λ de la suite des itérés de x_0.
7. Observer enfin que $x_{i+\mu} = x_\mu$ et en déduire une fonction `prePeriode : ('a -> 'a) -> 'a -> int` qui calcule la pré-periode de la suite des itérés de x_0.
8. L'algorithme de Floyd, implémenté dans les questions précédentes, permet de trouver une valeur de x_i dans le cycle et ensuite les valeurs de la période et de la pré-période en considérant la suite d'indices $(i, 2i)$ et en testant l'égalité $x_i = x_{2i}$. L'algorithme de Brent utilise la suite (i, j) et teste l'égalité $x_i = x_j$ en partant de $(i, j) = (0, 1)$ et en poursuivant avec :

$$\begin{cases} (i, j + 1) & \text{si } j \leq 2i \\ (j, j + 1) & \text{si } j = 2i + 1 \end{cases}$$

   Rédiger une fonction `brent : ('a -> 'a) -> 'a -> int * int` qui prend en arguments la fonction f et la valeur de x 0 et qui retourne le couple (i, j) trouvé par cet algorithme.
9. Quel(s) avantage(s) voyez-vous à utiliser l'algorithme de Brent plutôt que celui de Floyd ?