Exercice 1. Fonctions anonymes

Sur feuille, prévoir les types et les valeurs des expressions ci-dessous. Vérifier la réponse avec OCaml.

```
(fun a -> a + a*a) 2
fun x y z -> (x y) (z x)
fun x y z -> x (y z x)
fun f x -> f f x
```

Exercice 2. Flèches de Knuth

Déterminer ce que calcule cette fonction lorsque `r` $\in$ `{0, 1, 2, 3}` :

```
let rec knuth r n m = match (r,m) with
  | _ when n < 0 || m < 0 || r < 0 -> failwith "knuth: erreur d'argument"
  | 0,m -> 1 + m
  | 1,0 -> n
  | 2,0 -> 0
  | _,0 -> 1
  | r,m -> knuth (r-1) n (knuth r n (m-1));;
```

Remarque. Habituellement, pour $r \geqslant 3$, les entiers « `knuth 3 n m` », « `knuth 4 n m` », « `knuth 5 n m` », « `knuth 6 n m` » ... sont notés :

$$n \uparrow m \qquad n \uparrow\uparrow m \qquad n \uparrow\uparrow\uparrow m \qquad n \uparrow\uparrow\uparrow\uparrow m \qquad \dots$$

Exercice 3. Nombres de Catalan

1. Écrire une fonction « `somme: (int -> int) -> int -> int` » qui prend entrée une fonction f ainsi qu'un entier n et renvoie $\sum_{i=0}^{n} f(i)$. Vous devez écrire une fonction récursive et ne pas utiliser de fonction intermédiaire.
2. À l'aide de la fonction `somme`, écrire une fonction « `catalan: int -> int` » qui prend en entrée un entier n et renvoie le nombre de Catalan c_n où :

$$c_0 = 1, c_1 = 1 \qquad \text{et} \qquad \forall n \geqslant 2 : c_n = \sum_{p=0}^{n-1} c_p c_{n-1-p}$$

   On pourra vérifier les valeurs obtenues sur le site `https://oeis.org/`.
3. Vérifiez à l'aide d'OCaml que pour tout $n \in [\![0, 10]\!]$:

$$c_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} = \prod_{i=2}^{n} \frac{n+i}{i}.$$

   Dans cette question on s'interdit d'utiliser des flottants.

Exercice 4. Nombres parfaits

Un entier $\mathtt{n} \in \mathbb{N}^*$ est dit parfait s'il est égal à la somme de ses diviseurs stricts. Par exemple, 6 est parfait car $6 = 1 + 2 + 3$. Sans utiliser de liste ou de tableau, écrire une fonction qui prend en entrée un entier $\mathtt{n_{max}}$ et affiche tous les nombres parfaits de $[\![1\,;\,\mathtt{n_{max}}]\!]$ ainsi que leurs décompositions. Par exemple, pour $\mathtt{n_{max}} = 10000$:

```
6 = 1 + 2 + 3
28 = 1 + 2 + 4 + 7 + 14
496 = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248
8128 = 1 + 2 + 4 + 8 + 16 + 32 + 64 + 127 + 254 + 508 + 1016 + 2032 + 4064
```

Exercice 5. Partitions et compositions d'un entier

Une partition d'un entier $n \in \mathbb{N}$ consiste à écrire n comme une somme de termes strictement positifs. Par exemple $n = 5$ admet 7 partitions :

$$5 = 1+1+1+1+1, \quad 5 = 1+1+1+2, \quad 5 = 1+2+2, \quad 5 = 1+1+3,$$
$$5 = 2+3, \quad 5 = 1+4, \quad 5 = 5.$$

Dans une partition, l'ordre de la somme n'est pas important, par exemple $5 = 2+3$ et $5 = 3+2$ représentent la même partition. Dans la suite, on impose donc aux termes de la somme d'être croissants. Étant donnés deux entiers n et m, il est possible de générer toutes les partitions de n comportant m termes à l'aide d'une procédure récursive :

- ⋆ On génère d'abord toutes les partitions dont le premier terme est 1. Pour cela, il suffit de construire toutes les partitions de $n-1$ avec $m-1$ termes et d'y ajouter le terme 1. Avec $n = 6$ et $m = 3$:

$$\begin{cases} 5 = 2+3 \\ 5 = 1+4 \end{cases} \rightsquigarrow \begin{cases} 6 = 1+2+3 \\ 6 = 1+1+4 \end{cases}$$

- ⋆ On génère ensuite toutes les partitions dont le premier terme n'est pas 1. Pour cela, il suffit de construire toutes les partitions de $n-m$ avec m termes et d'ajouter 1 à chacun des termes. Avec $n = 8$ et $m = 3$:

$$\begin{cases} 5 = 1+1+3 \\ 5 = 1+2+2 \end{cases} \rightsquigarrow \begin{cases} 8 = 2+2+4 \\ 8 = 2+3+3 \end{cases}$$

1. Écrire une fonction récursive « `nb_part: int -> int -> int` » qui prend en entrée deux entiers n et m et renvoie le nombre de partitions de n avec m termes. On essaiera de minimiser le nombre de cas de base. Vérifiez les valeurs pour $n = 5$ ainsi que celles données dans le tableau à la fin de l'énoncé.
2. Écrire une fonction « `nb_total_part: int -> int` » qui prend en entrée un entier n et renvoie le nombre de partitions de n. Testez avec les valeurs données dans le tableau à la fin de l'énoncé.

Une composition d'un entier $n \in \mathbb{N}$ consiste à écrire n comme une somme, l'ordre des termes étant important. Par exemple, $n = 5$ admet 16 compositions :

$$5 = 1+1+1+1+1, \quad 5 = 1+1+1+2, \quad 5 = 1+1+2+1, \quad 5 = 1+2+1+1,$$
$$5 = 2+1+1+1, \quad 5 = 1+2+2, \quad 5 = 2+1+2, \quad 5 = 2+2+1,$$
$$5 = 1+1+3, \quad 5 = 1+3+1, \quad 5 = 3+1+1, \quad 5 = 2+3,$$
$$5 = 3+2, \quad 5 = 1+4, \quad 5 = 4+1, \quad 5 = 5.$$

Pour générer toutes les compositions d'un entier :

- ⋆ On génère toutes les compositions dont le premier terme est 1 en suivant la même logique que pour les partitions. Avec $n = 6$ et $m = 3$:

$$\begin{cases} 5 = 2+3 \\ 5 = 3+2 \\ 5 = 1+4 \\ 5 = 4+1 \end{cases} \rightsquigarrow \begin{cases} 6 = 1+2+3 \\ 6 = 1+3+2 \\ 6 = 1+1+4 \\ 6 = 1+4+1 \end{cases}$$

- ⋆ On génère ensuite toutes les compositions dont le premier terme n'est pas 1. Pour cela, il suffit de construire toutes les compositions de $n-1$ avec m termes et d'ajouter 1 au premier terme. Avec $n = 6$ et $m = 2$:

$$\begin{cases} 5 = 2+3 \\ 5 = 3+2 \\ 5 = 1+4 \\ 5 = 4+1 \end{cases} \rightsquigarrow \begin{cases} 6 = 3+3 \\ 6 = 4+2 \\ 6 = 2+4 \\ 6 = 5+1 \end{cases}$$

3. Écrire une fonction récursive « `nb_comp: int -> int -> int` » qui prend en entrée deux entiers n et m et renvoie le nombre de compositions de n avec m termes. On essaiera de minimiser le nombre de cas de base. Vérifiez les valeurs pour $n = 5$ ainsi que celles données dans le tableau à la fin de l'énoncé.
4. Écrire une fonction « `nb_total_comp: int -> int` » qui prend en entrée un entier n et renvoie le nombre de compositions de n. Vérifiez avec OCaml que pour tout $n \in [\![1, 10]\!]$, le nombre de compositions de n vaut 2^{n-1}.
5. Sans utiliser de liste ou de tableau, écrire une fonction qui prend en entrée un entier $n \in \mathbb{N}$ et affiche toutes les partitions de n. Par exemple avec $n = 5$ on obtient la figure 1.

   Indication. On pourra écrire une fonction auxiliaire (récursive) qui génère toutes les partitions. De plus, pour chaque partition, on construira une fonction « `f: unit -> unit` » qui affiche la partition dans la console.
6. Même question avec les compositions. Par exemple avec $n = 4$ on obtient la figure 2.

n	5	5	8	10	20	20
m	0	6	4	5	2	10
nb_part n m	0	0	5	7	10	42
nb_comp n m	0	0	35	126	19	92378

n	5	10	15	20
nb_total_part n	7	42	176	627

```
5 = 5
5 = 1 + 4
5 = 2 + 3
5 = 1 + 1 + 3
5 = 1 + 2 + 2
5 = 1 + 1 + 1 + 2
5 = 1 + 1 + 1 + 1 + 1
```

FIGURE 1

```
4 = 4
4 = 1 + 3
4 = 2 + 2
4 = 3 + 1
4 = 1 + 1 + 2
4 = 1 + 2 + 1
4 = 2 + 1 + 1
4 = 1 + 1 + 1 + 1
```

FIGURE 2

Exercices à rendre au plus tard le 22/02/2024 à 20h

Exercice 6. Algorithme de Babylone

Soit $x \in \mathbb{R}_+$ et $(u_n)_{n\in\mathbb{N}}$ la suite définie par :

$$u_0 = 1 \qquad \text{et} \qquad \forall n \in \mathbb{N} : u_{n+1} = \frac{1}{2}\left(u_n + \frac{x}{u_n}\right)$$

On admet que cette suite converge vers $\sqrt{x}$.

1. Écrire une fonction itérative « `racine_it: float -> float -> float` » qui prend en entrée x et ϵ, et renvoie le premier terme u_n tel que $\left|u_n - \frac{x}{u_n}\right| < \epsilon$.
2. De même en récursif : « `racine_rec: float -> float -> float` ».

Exercice 7. Suite de Fibonacci

On définit la suite de Fibonacci $(f_n)_{n\in\mathbb{N}}$ par :

$$f_0 = 0, f_1 = 1 \qquad \text{et} \qquad \forall n \in \mathbb{N} : f_{n+2} = f_{n+1} + f_n$$

1. Écrire une fonction « `fibo_aux: int -> int * int` » qui prend en argument un entier n et renvoie le couple (f_n, f_{n+1}). Vous devez écrire une fonction récursive et ne pas utiliser de fonction intermédiaire.
2. En déduire une fonction « `fibo: int -> int` » qui prend en entrée $n \in \mathbb{N}$ et renvoie f_n. Vérifier que votre fonction parvient à calculer f_{50000} instantanément (notez qu'il y a un overflow lors du calcul).

Exercice 8. Fractions continues

Le concept de fractions continues permet de donner une approximation de n'importe quel nombre réel à l'aide de nombres rationnels. Soient $n \in \mathbb{N}$, $a_0 \in \mathbb{Z}$ et $a_1, \ldots, a_n \in \mathbb{N}^*$. On définit un nombre rationnel noté $[a_0, a_1, \ldots, a_n]$ par :

$$[a_0, a_1, \ldots, a_n] = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \ldots}} \qquad \text{Exemple : } [a_0, a_1, a_2, a_3, a_4] = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{a_4}}}}$$

Pour tout $n \in \mathbb{N}$ et $a \in \mathbb{N}^*$, on pose $x_{a,n} = [a_0, a_1, \ldots, a_n]$ où $a_i = a$ pour tout i.

1. Écrire une fonction « `get_x: int -> int -> float` » qui prend en entrée a, n et renvoie $x_{a,n}$. Si $n < 0$ ou bien $a \leqslant 0$, votre fonction déclenchera une erreur. Vous devez écrire une fonction récursive et ne pas utiliser de fonction intermédiaire.

Afin de tester la fonction `get_x` (notez que vous devriez déjà l'avoir fait), on admet que pour tout $a \in \mathbb{N}^*$

$$\lim_{n \to +\infty} x_{a,n} = \ell_a \qquad \text{où} \qquad \ell_a = \frac{a + \sqrt{a^2 + 4}}{2}.$$

2. Écrire une fonction « `verif_fc: int -> bool` » qui prend en argument un entier k et indique si pour tout $a \in [\![1; k]\!]$:

$$|x_{a,24} - \ell_a| \leqslant 10^{-10}.$$

La valeur absolue sera calculée avec la fonction « `abs_float: float -> float` ». On utilisera une boucle `while` pour arrêter l'exécution dès que l'un des tests est faux.

Vérifiez que « `verif_fc 100000` » vaut `true` et que « `verif_fc 1000000` » vaut `false` (ceci est dû aux erreurs d'approximations sur les flottants).

Partie facultative. Soit $x \in \mathbb{R}$. On admet que x peut toujours être décomposé en fraction continue (si x n'est pas rationnel, cette décomposition est infinie) :

$$x = [a_0, a_1, a_2, \ldots] \qquad \text{avec} \qquad a_0 \in \mathbb{Z} \qquad \text{et} \qquad a_1, a_1, \ldots \in \mathbb{N}^*.$$

Pour tout $n \in \mathbb{N}$, on pose $t_{x,n} = [a_0, a_1, \ldots, a_n]$. On note également $n_0 \in \mathbb{N}$ le plus petit entier tel que $|x - t_{x,n_0}| \leqslant 10^{-10}$.

3. Sans utiliser de liste ou de tableau, écrire une fonction « `print_fc: float -> unit` » qui prend en entrée x et affiche $a_0, a_1, \ldots, a_{n_0}$. Par exemple :

```
print_fc 0.;;          (* Affiche 0 *)
print_fc 1.;;          (* Affiche 1 *)
print_fc (-1.);;       (* Affiche -1 *)
print_fc 1.4;;         (* Affiche 1 2 1 1 *)
print_fc 2.55;;        (* Affiche 2 1 1 4 1 1 *)
print_fc 0.123;;       (* Affiche 0 8 7 1 2 5 *)
print_fc (-4.123);;    (* Affiche -5 1 7 7 1 2 4 1 *)
print_fc (sqrt 2.);;   (* Affiche 1 2 2 2 2 2 2 2 2 2 2 2 2 2 *)
let pi = 4. *. atan 1.0 in
    print_fc pi;;      (* Affiche 3 7 15 1 292 1 1 1 *)
```