# DS 1 du 17 mars 2023 - 2 heures

*Calculatrice, téléphone et documents interdits*

*Les programmes non commentés ou dont le fonctionnement n'est pas expliqué ne seront pas relus. Une présentation raisonnablement aérée (avec une identation convenable des structures de choix ou de répétition), ainsi que des résultats soulignés ou encadrés, sont des éléments de l'évaluation... C'est vous qui voyez...*

**Exercice 1** La touche  du clavier est cassée... vous ne pouvez pas utiliser ce symbole dans cet exercice...

1. Écrire une fonction `produit` de type `int -> int -> int` qui reçoit 2 entiers naturels.
2. En déduire une fonction `puissanceN : int -> int -> int` qui reçoit deux entiers naturels non simultanément nuls
3. En déduire enfin une fonction `puissance : int -> int -> int` telle que `puiss a n` renvoie $\texttt{a}^{\texttt{n}}$ pour $(\texttt{a},\texttt{n}) \in \mathbb{Z} \times \mathbb{N}$ et $(\texttt{a},\texttt{n}) \neq (0,0)$.

**Exercice 2** On considère les suites $(u_n)$ et $(v_n)$ définies sur $\mathbb{N}^*$ par

$$\begin{cases} u_1 = 2 \\ v_1 = -1 \end{cases} \quad \text{et } \forall n \in \mathbb{N}, \begin{cases} u_{n+1} = 2u_n - v_n \\ v_{n+1} = u_n + v_n \end{cases}$$

1. Écrire deux fonctions `u` et `v` récursives (croisées) prenant chacune en entrée un paramètre entier non nul `n` et revoyant respectivement $u_{\texttt{n}}$ et $v_{\texttt{n}}$.
2. Réaliser la même fonction `uv` de type `int -> (int * int)` qui renvoie directement le couple $(u_n, v_n)$ et de complexité linéaire.
3. On note $a_n$ (respectivement $b_n$) le nombre d'appels à la fonction `u` (respectivement `v`) pour calculer $u_n$ (respectivement $v_n$).
   a. Déterminer $a_1$ et $b_1$.
   b. Monter que $\forall n \in \mathbb{N}^*,\ a_{n+1} = b_{n+1} = 1 + a_n + b_n$.
   c. Quelle est la nature de la suite $(c_n)_{n\in\mathbb{N}^*}$ définie par $c_n = a_n + b_n$.
   d. En déduire $a_n$ et $b_n$ en fonction de $n$.
   e. Que dire de la première méthode ?

**Exercice 3** Le **FizzBuzz** est un test de compétences utilisé lors des entretiens d'embauche en informatique qui permet d'éliminer les candidats les plus faibles. Il s'agit d'écrire une fonction qui affiche les entiers compris entre 1 et `n` donné en paramètre (`n` > 1), mais où les multiples de 3 sont remplacés par `"Fizz"`, les multiples de 5 par `"Buzz"` et les multiples à la fois de 3 et de 5 par `"FizzBuzz"`. Par exemple :

```
# fizzbuzz 20;;
1 ; 2 ; Fizz ; 4 ; Buzz ; Fizz ; 7 ; 8 ; Fizz ; Buzz ; 11 ; Fizz ; 13 ; 14 ;
FizzBuzz ; 16 ; 17 ; Fizz ; 19 ; Buzz ; - : unit = ()
```

**Exercice 4** On cherche dans cet exercice une solution efficace pour calculer $a^n$ pour deux entiers naturels $a$ et $n$ non simultanément nuls. On essaiera d'écrire les fonctions demandées sous forme récursive et on privilégiera les filtrages aux « `if` ».

1. Une première solution « naïve ».
   a. En vous basant sur le fait que $a^n = a \times a \times ... \times a$, écrire une fonction `puiss1` qui reçoit deux entiers naturels non simultanément nuls $a$ et $n$ et renvoie le résultat de $a^n$.
   b. On note $A_n$ le nombre de multiplications réalisées lors de l'appel `puiss1 a n`. Exprimer (en justifiant) $A_n$ en fonction de $n$.
2. On remarque que $\forall n \in \mathbb{N},\ a^n = \begin{cases} (a^2)^{\frac{n}{2}} \text{ si } n \text{ pair} \\ a \times (a^2)^{\frac{n-1}{2}} \text{ si } n \text{ impair} \end{cases}$.
   a. Écrire une fonction `puiss2` utilisant ce principe.
   On note $B_n$ le nombre de multiplications réalisées lors de l'appel `puiss2 a n`.
   b. Pour $p \in \mathbb{N}$, exprimer $B_{2p+1}$ et $B_{2p}$ en fonction de $B_p$.
   c. Démontrer par récurrence que $\forall n \in \mathbb{N},\ \log_2(n+1) \leqslant B_n \leqslant 2\log_2(n+1)$. Pour l'hérédité, on pourra raisonner par disjonction de cas selon la parité de $n+1$.
3. Démontrer que $B_n$ est négligeable par rapport à $A_n$, c'est à dire que $\dfrac{B_n}{A_n} \xrightarrow[n\to+\infty]{} 0$.

**Exercice 5** Sur les listes :

1. Quelques questions indépendantes :
   a. Écrire une fonction `pair : int -> bool` qui indique si un entier est pair ou non.
   b. Écrire une fonction `unite : int -> int` qui renvoie le chiffre des unités d'un entier naturel donné.
   c. En déduire une fonction `unichiffre : int -> bool` qui indique si un entier naturel n'est écrit qu'à l'aide d'un seul chiffre (par exemple 4 ; 11 ; 333 ; ....).
2. a. Écrire une fonction `tous : ('a -> bool) -> 'a list -> bool` qui, prenant comme arguments une fonction à valeurs booléennes $f$ et une liste $\ell$, renvoie `true` si $f(x)$ est vrai pour tout $x$ de $\ell$, et `false` sinon. Par exemple :

```
# tous pair [2;3;4;6];;
- : bool = false
# tous unichiffre [1;22;333;6666];;
- : bool = true
```

   b. En déduire une fonction `au_moins` avec la même signature, mais qui indique si au moins un élément $x$ vérifie $f(x)$. Par exemple :

```
# au_moins unichiffre [23; 44; 56];;
- : bool = true
# au_moins pair [3;7;11;31];;
- : bool = false
```

Quelques pistes pour la correction

***Correction 1***

1. On se base sur le fait que $a \times 0 = 0$ et $a \times b = a + a \times (b-1)$ si $b \in NN^*$.

```
let rec produit a = function
          0 -> 0
        | b -> a + (produit a (b-1))
;;
```

2. Même idée : $a^0 = 1$ et $a^n = a \times a^{n-1}$ si $n \in NN^*$.

```
let rec puissance a = function
          0 -> 1
        | n -> produit a (puissance a (n-1))
;;
```

3. Il faut faire attention aux boucles infinies et ne faire des puissance qu'avec des entiers naturels, une solution :

```
let puissance a n =
        if a > 0
        then puissanceN a n
        else if n mod 2 == 0
                  then puissanceN (-a) n
                  else -puissanceN (-a) n
;;
```

Une autre plus courte utilisant la valeur absolue :

```
let puissance a n =
        if a > 0 || n mod 2 == 0
        then puissanceN (abs a) n
        else -puissanceN (abs a) n
;;
```

Encore un peu plus court :

```
let puissance a n =
   let r = puissanceN (abs a) n in if a > 0 || n mod 2 = 0 then r else -r
;;
```

***Correction 2***

1. Solution utilisant la récursivité croisée :

```
let rec u = function
          1 -> 2
        | n -> 2*(u (n-1))-(v (n-1))
and v = function
          1 -> -1
        | n -> (u (n-1)) + (v (n-1))
;;
```

2. Solution de type `int -> (int * int)`

```
        let rec uv = function
          1 -> (2,-1)
        | n -> let u,v = uv (n-1) in (2*u-v, u+v)
;;
```

3. a. $a_1 = b_1 = 1$ (cas d'arrêt).

   b. Si `n` $\geqslant 1$, pour calculer `u (n+1) =2*(u n)-(v n)`, il faut
   - 1 appel pour entrer dans la fonction `u`.
   - $a_n$ appels pour recalculer `u n`
   - $b_n$ appels pour recalculer `v n`

   Donc $a_{n+1} = 1 + a_n + b_n$, c'est pareil pour $b_{n+1}$

   c. $\forall n \in \mathbb{N}^*,\ c_{n+1} = 2 + 2c_n$. $(c_n)$ est donc une suite arithético-géométrique que l'on résout (détail sur demande) : $\forall n \in \mathbb{N}, c_n = 2^{n+1} - 2$

   d. La première méthode n'est donc pas efficace : complexité exponentielle.

***Correction 3*** Pas de difficulté particulière pour cet exercice, un filtrage et une fonction récursive feront l'affaire (attention à l'ordre des filtrages et des affichages !) :

```
let rec fizzbuzz = function
          0 -> ();
        | n when n mod 15 = 0 -> fizzbuzz (n-1); print_string "FizzBuzz␣;␣";
        | n when n mod 3 = 0 -> fizzbuzz (n-1); print_string "Fizz␣;␣";
        | n when n mod 5 = 0 -> fizzbuzz (n-1); print_string "Buzz␣;␣";
        | n -> fizzbuzz (n-1); print_int n; print_string "␣;␣";
;;
```

***Correction 4*** 1. a. Une solution récursive :

```
let rec puiss1 a = function
    | 0 -> 1
    | n -> a * puiss1 a (n-1)
;;
```

   b. On a $A_0 = 0$ et la relation de récurrence $A_{n+1} = 1 + A_n$ pour $n \in \mathbb{N}$. Ainsi $A_n = n$.

2. a. En utilisant la formule, on obtient :

```
let rec puiss2 a = function
    | 0 -> 1
    | n when n mod 2 = 0 -> puiss2 (a*a) (n/2)
    | n -> a*puiss2 (a*a) (n/2)
;;
```

   b. Pour $p \in \mathbb{N}$,
   - $B_{2p} = 1 + B_p$ (le carré et l'appel suivant)
   - $B_{2p+1} = 2 + B_p$ (la multiplication, le carré et l'appel suivant)

c. Soit $\mathcal{P}(n)$ : « $\log_2(n+1) \leqslant B_n \leqslant 2\log_2(n+1)$ »
- Pour $n = 0$, il n'y a aucune multiplication ainsi $B_0 = 0$ et puisque $\log_2 1 = 0$, $\mathcal{P}(0)$ est vraie.
- Supposons les propositions $\mathcal{P}(0), \mathcal{P}(1), ..., \mathcal{P}(n)$ vraies pour $n$ fixé dans $\mathbb{N}$, alors :
  - ◇ Si $n+1$ est pair (sous la forme $n+1 = 2p$) : $Bn+1 = B_{2p} = 1 + B_p$. Or la proposition $P(p)$ est vraie donc :

$$\begin{gathered}
\log_2(p+1) \leqslant B_p \leqslant 2\log_2(p+1) \\
1 + \log_2(p+1) \leqslant 1 + B_p \leqslant 1 + 2\log_2(p+1) \\
\log_2 2 + \log_2(p+1) \leqslant 1 + B_p \leqslant 2\left(\log_2(p+1) + 1/2\right) \\
\log_2\left(2(p+1)\right) \leqslant B_{n+1} \leqslant 2\left(\log_2(p+1) + \log_2\sqrt{2}\right) \\
\log_2(2p+2) \leqslant B_{n+1} \leqslant 2\left(\log_2\left(\sqrt{2}(p+1)\right)\right) \\
\log_2\left(2p+1\right) \leqslant B_{n+1} \leqslant 2\left(\log_2\left(\sqrt{2}p + \sqrt{2}\right)\right) \\
\log_2\left(2p+1\right) \leqslant B_{n+1} \leqslant 2\log_2(2p+1)^{(*)} \\
\log_2\left(n+2\right) \leqslant B_{n+1} \leqslant 2\log_2(n+2)
\end{gathered}$$

$(*)$ : car $\sqrt{2}p + \sqrt{2} \leqslant 2p+1 \iff p \geqslant \sqrt{2}/2$ qui est vrai ici.

  - ◇ Si $n+1$ est impair (sous la forme $n+1 = 2p+1$) : $Bn+1 = B_{2p+1} = 2 + B_p$. Or la proposition $P(p)$ est vraie donc :

$$\begin{gathered}
\log_2(p+1) \leqslant B_p \leqslant 2\log_2(p+1) \\
2 + \log_2(p+1) \leqslant 2 + B_p \leqslant 2 + 2\log_2(p+1) \\
\log_2 4 + \log_2(p+1) \leqslant B_{n+1} \leqslant 2\left(1 + \log_2(p+1)\right) \\
\log_2(4p+4) \leqslant B_{n+1} \leqslant 2\left(\log_2 + \log_2(p+1)\right) \\
\log_2(2p+2) \leqslant B_{n+1} \leqslant 2\left(\log_2(2p+2)\right) \\
\log_2(n+2) \leqslant B_{n+1} \leqslant 2\left(\log_2(n+2)\right)
\end{gathered}$$

— Ainsi par récurrence forte $\mathcal{P}(n)$ est vraie pour tout entier naturel $n$.

3. Par le théorème d'encadrement et les croissances comparées : Si $n \in \mathbb{N}^*$, $\dfrac{\log_2 n}{n} \leqslant \dfrac{A_n}{B_n} \leqslant 2\dfrac{\log_2 n}{n}$.

Or $\dfrac{\log_2 n}{n} = \dfrac{1}{\ln 2} \times \dfrac{\ln n}{n} \to 0$. Ce qui prouve ce qui est demandé.

***Correction 5***

1. Pas de difficulté pour ces 3 premières questions :

```
let pair n = n mod 2 = 0;;

let unite n = n mod 10;;

let rec unichiffre = function
        n when n < 10 -> true;
        | n -> let m = n /10 in (unite n = unite m) && unichiffre m
;;
```

2. Les 2 suivantes (pour la seconde, on a utilisé la négation) :

```
let rec tous f = function
          [] -> true
        | h::q -> (f h) && (tous f q)
;;

let au_moins f l = not( tous (function x -> not(f x) ) l);;
```