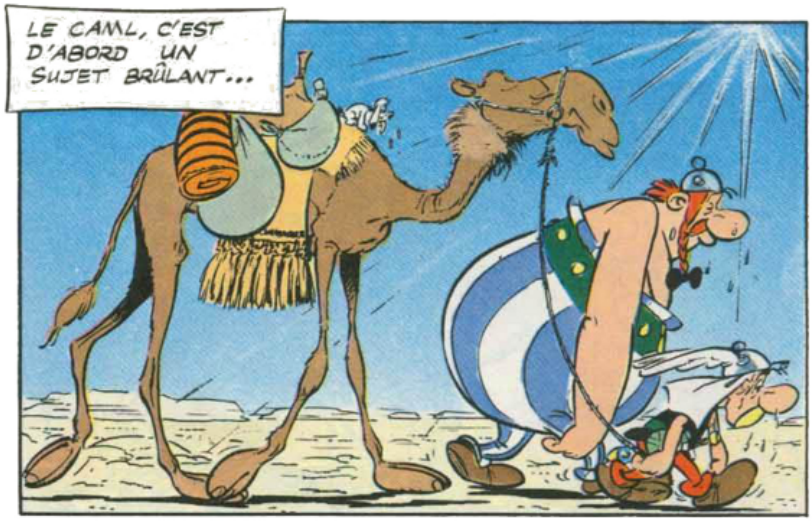
Algorithmique

Info-sup S1

EPITA

Séminaire CAML
Exercices

septembre 2025



Sauf indication contraire, l'environnement est supposé **vide** au début de chaque exercice.

1 Les bases

Expressions et définitions

Exercice 1.1 (Évaluations)

Quels seront les résultats des évaluations successives par CAML de chacune des phrases suivantes ?

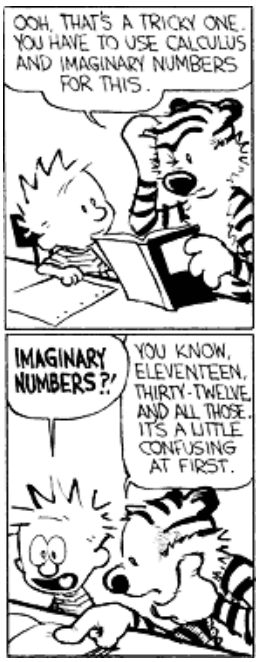
```
# let a = 11 + 7 * 3 ;;

# let b = 3 in 6 + b ;;

# let a = 3 in let c = 2 + a ;;

# let c = let a = 3 in 2 + a ;;

# let c = a * b ;;


# let a = 2 in b = 5 in a * b ;;

# let a = 5 in let b = 2 * a in a * b ;;

# let a = 5 and b = 2 * a in a * b ;;
```


```
# (let x = 2 in x * (x - 1)) + (let x = 3 in x * (x + 1)) ;;

# let x = 2 and y = 5 in
  let y = x and z = y in
  y + z ;;

# let x = 2 and y = 5 in
    let y = x in
      let z = y in
        y + z ;;

# let a=let a=let a=1 in 2*a and b=let b=2 in 2*b in let a=a+b and b=a-b in a*a-b*b;;
```

Les types simples

Exercice 1.2 (Types et valeurs)

Quels sont les types et les valeurs associés aux expressions suivantes ?




```
# 1. +. 2. ;;
# 1 + 1.5 ;;
# 1 +. 1.5 ;;

# 1 / 2 ;;
# 5 / 2 = 4 / 2 ;;
# 5 mod 2 ;;
# 4 mod 2 ;;

# 10-2 * 2 ;;
# 10 * 24 mod 10 ;;
# 24 mod 10 * 10 ;;

# "A string" ;;
# "" ;;
# 'a' ;;
# '' ;;
# "Hello Word" ^ '!' ;;

# 1. < 3 ;;
# 1 < 2 < 3 ;;
# "how do string comparisons work?" > "how do strings work?" ;;
# 'Z' < 'a' ;;
# let f x = x + 1 and g x = x + 1 in f = g ;;

# 1 < 2 && 3. > 5. ;;
# false || (1/0 = 0) ;;
# true || (1/0 = 0) ;;
# true && (1/0 = 0) ;;
# false && (1/0 = 0) ;;
# false && (1.5 > 1) ;;
```

Exercice 1.3 (Logique)

Soit a, b, c, trois valeurs booléennes.

1. Simplifier les expressions CAML suivantes :
   - a `&&` `true`
   - a `&&` `false`
   - a `&&` `not` a
   - a `||` `true`
   - a `||` `false`
   - a `||` `not` a
   - (a `&&` `not` b) `||` (`not` a `&&` b)
   - (a `&&` b) `||` (`not` a `&&` `not` b)

2. Remplacer les `??` par l'opérateur qui convient.
   - `not` (a `&&` b) équivalent à `not` a `??` `not` b
   - `not` (a `||` b) équivalent à `not` a `??` `not` b

3. Quelle(s) expression(s) est (sont) équivalente(s) à l'expression ci-dessous ?

```
(a && b) || (a && c)
```

   - `a && (b || a) && c`
   - `a && (b || c)`
   - `(a && b) || c`

2 Les fonctions

Un seul paramètre

Exercice 2.1 (Types et applications de fonctions)

Les fonctions suivantes sont dans l'environnement :
- `sqrt : float -> float` (racine carrée)
- `float_of_int : int -> float`
- `int_of_float : float -> int`

Évaluer les phrases suivantes, corriger les éventuelles erreurs :

```
# let f x = x * x ;;
# f -4 ;;
# f 2 + 3 ;;
# f (2 + 3) ;;

# let g x =
      let f x = x + 1 in
        f x*2 ;;
# g 5 ;;

# let f1 x = int_of_float x / 2 ;;
# let f2 x = sqrt float_of_int f1 x ;;

# let f3 x = x mod 2 = 0 in let f4 x = f3 x+1 in f4 5 ;;

# let x = 12 and f x = 2 * x ;;
# let a = 3 and f x = x + a ;;

# let g1 x = true && (x/0 = 0) ;;
# g1 0 ;;
# let g2 x = false && (1.5 > x+1) ;;
# g2 0 ;;
# let g3 x = let a = 12 in x * x ;;
# g3 30 ;;
```

Exercice 2.2 (Puissance)

Écrire la fonction `power30 : int -> int` qui à partir de x calcule x^{30} : la seule opération autorisée est la multiplication, et moins il y en a mieux c'est !

Exercice 2.3 (Take a Chance)

Écrire les fonctions CAML suivantes :

mirror : prend en paramètre un entier positif n à **deux chiffres non nuls** [1] et retourne son "miroir".

abba : Utiliser `mirror` pour écrire une fonction qui pour tout nombre à deux chiffres non nuls ab calcule $abba$.

stammer : Utiliser les deux fonctions précédentes pour écrire la fonction qui pour tout nombre de la forme ab, retourne $baababba$.

Exemples d'applications :

```
# mirror 15 ;;
 - : int = 51
# abba 42 ;;
 - : int = 4224
# stammer 18 ;;
 - : int = 81181881
```

1. On ne s'occupera pas pour l'instant des résultats lorsque le paramètre n'est pas un entier positif à 2 chiffres non nuls.

Plusieurs paramètres

Exercice 2.4 (Applications de fonctions)

Évaluer les phrases suivantes, corriger les éventuelles erreurs.

```
# let sum a b c d = a + b + c + d ;;
# sum 2 0 2 4 ;;

# let build a b c d = a * 1000 + b * 100 + c * 10 + d ;;
# build 2 0 2 9 ;;

# let f x y = x mod y ;;
# let g x y z = x + y = z ;;
# f (2, 5) ;;
# let a = 20 and b = 3 in f (a b) ;;
# g -4 -5 -9 ;;
# f (f 25 7) (f 4 3) ;;
# g (f 5 2 * 2) (5 - f 13 4) (f 20 7) ;;
# f (g 5 2 7) (4 * 12) ;;
```

Exercice 2.5 (Types de fonctions)

`float_of_int : int -> float` et `int_of_float : float -> int` sont dans l'environnement.

Typer les fonctions suivantes :

```
# let divmod x y = (x / y, x mod y) ;;

# let test a b c = (float_of_int a *. b) > 4. && c = 'A' ;;

# let chapi x y =
     let a = 3. *. x in
        let b = int_of_float a mod y in
              b = 0 ;;

# let chapo x y z =
     let g a =
         let b = float_of_int (a * x) in
             (b +. y) /. 2.
     in int_of_float (g z) ;;
```

Exercice 2.6 (Heures, minutes, secondes)

On suppose ici les paramètres "valides" (minutes et secondes dans $[0, 60[$, heures $\in \mathbb{N}$).

1. Écrire la fonction `sec_of_time` qui convertit une durée représentée par 3 entiers (heures, minutes, secondes) en secondes. Exemple : 23h 13m 59s = 83639s
2. Écrire la fonction `time_of_sec` qui convertit des secondes en heures, minutes, secondes (qui seront représentées par le triplet (h, mn, s)).
3. Utiliser les deux fonctions précédentes pour écrire la fonction `add_times` qui additionne deux durées (représentées chacune par 3 entiers).
   Exemple : 12h 22m 15s + 15h 13m 10s = 27h 35m 25s

Exemples d'applications :

```
# sec_of_time 23 13 59 ;;
 - : int = 83639

# time_of_sec 83639 ;;
 - : int * int * int = (23, 13, 59)

# add_times 12 22 15 15 13 10 ;;
 - : int * int * int = (27, 35, 25)
```

3 Analyses par cas

L'alternative

Exercice 3.1 (Évaluations)

Évaluer les phrases suivantes. Écrire, lorsque cela est possible, des fonctions équivalentes plus simples.

```
# let test b = if b then true else 0 ;;

# let add a b =
    (if a > b then a - b else b - a)
    + (if a <= b then b / a else a / b) ;;
# add 10 5 ;;

# let f a b c =
    (if a > b then
        if b > c then a + b else c + a
     else
        if a > c then a + b else b + c) *
      (if a > b then
         if b > c then a - b else a - c
       else
         if a > c then a - b else b - c) ;;
# f 1 2 3 ;;

# let all_even a b c d =
      if a mod 2 = 0 then
        if b mod 2 = 0 then
          if c mod 2 = 0 then
            if d mod 2 = 0 then
              true
            else
              false
          else
            false
        else
          false
      else
        false ;;

# all_even 0 2 4 6 ;;
```

Exercice 3.2 (Alternatives vs logique)

Pour chaque formule logique ci-dessous, écrire une fonction booléenne en n'utilisant que des alternatives (sans aucun opérateur booléen), en essayant de simplifier au maximum.

1. $a \wedge b$ *et* logique
2. $a \vee b$ *ou* logique
3. $\neg a \vee b$ *implication* logique : $a \to b$

Pour ces deux dernières formules, trouver un opérateur non logique équivalent.

4. $(a \wedge \neg b) \vee (\neg a \wedge b)$ *ou exclusif* logique
5. $(a \to b) \wedge (b \to a)$ *équivalence* logique : $a \equiv b$

Exercice 3.3 (Ordres)

Définir les fonctions suivantes :

1. `max2` et `min2` qui retournent respectivement le maximum et le minimum de 2 entiers ;
2. `max3` et `min3` qui retournent respectivement le maximum et le minimum de 3 entiers **sans** utiliser les fonctions définies / précédentes ;
3. `middle3` qui retourne l'élément du milieu parmi 3 entiers (vous pouvez utiliser les fonctions précédentes) ;
4. `max4` et `min4` qui retournent respectivement le maximum et le minimum de 4 entiers (utiliser les fonctions précédentes !).

Exercice 3.4 (Somme des carrés)

Écrire la fonction `highest_square_sum` qui à partir de trois entiers retourne la somme des carrés des deux plus grands.

Utiliser les fonctions définies à l'exercice 3.3.

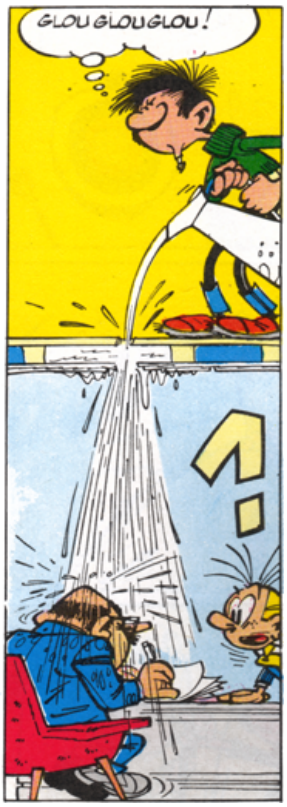
Exemples d'applications :

```
# highest_square_sum 3 2 1 ;;
- : int = 13

# highest_square_sum 1 (-5) 10 ;;
- : int = 101

# highest_square_sum (-1) (-2) (-3) ;;
- : int = 5
```

Le filtrage

Exercice 3.5 (Évaluations)

Quels seront les résultats des évaluations des phrases suivantes ? Corriger les éventuelles erreurs et tenter d'éliminer les "warning".




```
# let f1 x = match x with
    | 0 -> 0
    | y -> 1 / y ;;

# let switchonoff x = match x with
      "on" -> true
    | "off" -> false ;;

# let a = 42 and b = 666 in
    match a * b - 27972 with
      | 0            -> "zero"
      | x when x > 0 -> "positive"
      | _            -> "negative" ;;

# let square x = match x with
      x when x > 0  -> x * x
    | x when x <= 0 -> invalid_arg "x has to be positive" ;;
# square 5 ;;
# square (-5) ;;

# let parity n =
    if n >= 0 then
      match n with
          0 | 2 | 4 | 6 | 8 -> "even"
        | 1 | 3 | 5 | 7 | 9 -> "odd"
    else
      failwith "negative";;
  in
    parity 8 ;; (* test with (-10), 34 ... *)

# let grade a b =
    match (a + b) / 2 with
        x when x > 15 -> 'A'
      | x when x > 5 -> 'D'
      | 8 | 9 | 10    -> 'C'
      | x when x > 11 -> 'B'
      | _             -> 'E' ;;
# grade 12 14 ;;

# let h x y =
    match x with
        0 -> 0
      | y -> 1
      | _ -> -1 ;;
# h 5 5 ;;
# h 5 3 ;;
```

Exercice 3.6 (Simplifications)

Typer les fonctions suivantes. Que calculent-t'elles ?
Écrire des fonctions équivalentes plus simples !

1.
```
# let test a =
    let f n =
      if n < 0 then
        -1
      else
        1
    in
    match f a * a with
        0 | 1 | 2 | 3 | 4 -> true
      | n when n >= 10 -> false
      | _ -> true ;;
```

2.
```
# let f a b = match a with
      0             -> 0
    | x when x < 0  -> (match b with
                          0  -> 0
                        | _  -> -x + x/b*b)
    | x             -> (match b with
                        | 0          -> 0
                        | y when y<0 -> x - (x/(-y))*(-y)
                        | y          -> x - x/y*y) ;;
```

Exercice 3.7 (Affranchissement)

Le but de cet exercice est d'écrire une fonction calculant le tarif d'affranchissement d'un colis de moins de $3kg$ pour les envois en France. Les tarifs sont donnés par le tableau suivant :

Poids jusqu'à	Économique	Standard	Express
$500g$	3,4 €	4,6 €	9,1 €
$1000g$	4,6 €	5,9 €	11 €
$2000g$	5,1 €	6,5 €	13,5 €
$3000g$	6,9 €	7,2 €	14,2 €

Les 3 types de tarifs seront représentés par les chaînes de caractères `"eco"`, `"standard"` et `"express"`.

1. Écrire la fonction `rate_eco` qui calcule le tarif d'envoi d'un colis en économique, à partir du poids.
2. Écrire les fonctions `rate_standard` et `rate_express` analogues à `rate_eco`, pour les cas des envois aux tarifs normal et express.
3. Écrire la fonction `rate`, qui permet de calculer le tarif d'affranchissement d'un colis connaissant son poids et le code du tarif.
4. Peut-on résoudre ce problème avec moins de fonctions (essayer de réunir les fonctions de tarifs `rate_eco`, `rate_standard` et `rate_express` en une seule) ?

```
# rate 750 "eco" ;;
 - : float = 4.6
# rate (-150) "standard" ;;
 Exception: Invalid_argument "weight has to be positive".
# rate 150 "drone" ;;
 Exception: Failure "unknown code".
# rate 3500 "express" ;;
 Exception: Failure "too heavy".
```

Filtrage, fonctions anonymes et produit cartésien

Exercice 3.8 (Évaluations)

Que donneront les évaluations successives des phrases suivantes ?

```
# let a = let b = (0, true) in (b, "one") ;;
# let (x, y) = a in x ;;

# let (a, b) = (1, 2) ;;
# let (c, _) = (a, b) ;;

# let f c = let (x, y) = c in let z = (x+1, not y) in (x, z) ;;

# let g x y z = match (z, x) with
          (1, _) -> (1, x)
        | (y, true) -> (2 * y, x)
        | _ -> failwith "error" ;;
# g true "" 2 ;;

# let f x = match x with
          ((1, true), _)  -> 1
        | ((2, false), x) -> 3
        |  (x, "aa")      -> let (a, b) = x in a
        |  _              -> 2 ;;

# let f x y =
      let g x = (x + 1)/ 2
      in
      match (not x, g y) with
        (true, _) -> true
      | (_, 42) -> false
      | _ -> failwith "error" ;;

# let x = 5 in function a -> function b ->
  let y = 2 in float_of_int (a * x) > (float_of_int y +. b) ;;

# let test = function
      | (true, b) -> b
      | _         -> false ;;

# let f = function
       ((0, _), _) | (_, (0, _))       -> (0, false)
     | ((x, sx), (y, sy)) when sx = sy -> (x * y, false)
     | ((x, true), (y, sy))            -> (x * y, not sy)
     | ((x, _), (y, sy))               -> (x * y, sy) ;;

# let f = function
      "and" -> (function (v1, v2) -> v1 && v2)
    | "or" -> (function (v1, v2) -> v1 || v2)
    | "xor" -> (function (v1, v2) -> v1 <> v2)
    | "imply" -> (function (v1, v2) -> not v1 || v2)
    | _ -> failwith "unknown operator" ;;

# let h x y = match (x, y) with
        (x, x) -> true
      | _      -> false  ;;
```

Exercice 3.9 ("Filtrage anonyme")

Les fonctions à écrire doivent obligatoirement utiliser du filtrage mais sans `match`!

1. Écrire une fonction CAML qui compte le nombre de 0 d'un couple d'entiers.

```
# count (0, 0) ;;
 - : int = 2
# count (0, 1) ;;
 - : int = 1
# count (4, 5) ;;
 - : int = 0
```

2. Écrire en CAML la fonction suivante : $\forall (m,n) \in \mathbb{Z} \times \mathbb{Z}, f(m,n) = \begin{cases} 0 & \text{si } m = 0 \\ 2m & \text{si } m \neq 0 \,\&\, n = 0 \\ m.n & \text{si } m \neq 0 \,\&\, n \neq 0 \end{cases}$
3. Soit la fonction CAML `or3` définie ci-dessous. Trouver une fonction équivalente sans utiliser d'opérateurs logiques.

```
# let or3 (a, b, c) = a || b || c ;;
```

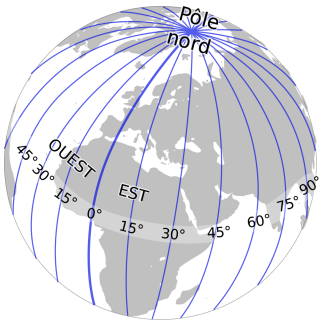
Exercice 3.10 (Décalage horaire)

On veut calculer le décalage horaire solaire exprimé en h, m, s entre 2 points du globe dont on connaît la longitude en °, ' et " Ouest ('W') ou Est ('E') par rapport au méridien de Greenwich. Par exemple, le point 3°12'15"E est en avance de 1h5mn31s sur le point 13°10'30"W.

Les longitudes sont exprimées de 180°W à 0°E et de 0°W à 180°E. La ligne de changement de date passe à la longitude 180°.
Pour calculer le décalage horaire entre 2 points, en évitant les problèmes de changement de date, on calculera pour chaque point son décalage avec Greenwich, puis on effectuera la différence des 2 décalages.

Indice : $360/24 = 15$
$\Rightarrow 15^o = 1h$

$\Rightarrow$ 3°12'15"E $= 11535"$
$= 769s$
$=$ 0h 12 mn 49 s



Écrire la fonction `time_difference` qui retourne le décalage horaire entre 2 points de longitudes.

Conseil :
Commencer par décomposer le problème en sous-problèmes, afin de déterminer les fonctions nécessaires.

Exemples :

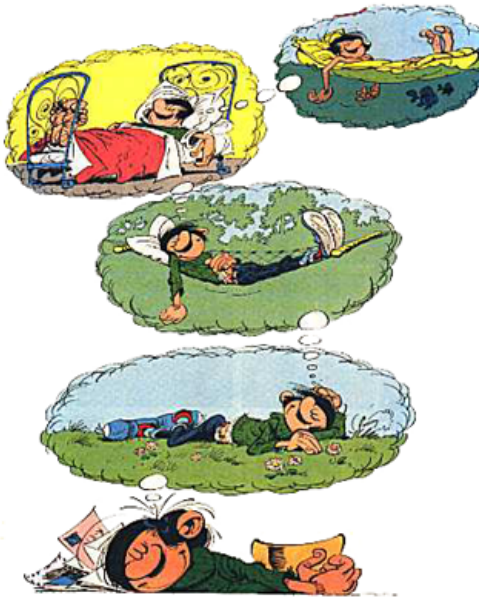
```
# time_difference (3, 12, 15, 'E') (0, 0, 0, 'W');;
 - : string = "earlier:0h 12mn 49s"

# time_difference (3, 12, 15, 'E') (13, 10, 30, 'W') ;;
 - : string = "earlier : 1h 5mn 31s"

# time_difference (0, 1, 4, 'W') (2, 0, 5, 'E') ;;
 - : string = "later: 0h 8mn 4s"
```

4 La récursivité

Exercice 4.1 (Qu'est-ce ?)

Que calculent les fonctions suivantes (étudier tous les cas) ?

```
# let rec f1 n = match n with
      0 -> 0
    | n -> n + f1 (n-1) ;;

# let rec f2 n d = match n with
      n when n < 0       -> invalid_arg "f2: n < 0"
    | 0                  -> 0
    | n when n mod d = 0 -> n + f2 (n-1) d
    | n                  -> f2 (n-1) d ;;

# let rec f3 d n =
    if n <= 1 then
      d
    else
      d * f3 (d - 1) (n - 1) ;;

# let rec f4 = function
      0 -> 0
    | n -> n / 2 + f4 (2*n) ;;

# let rec mystery x =
    if x < 10 then
      1
    else
      1 + mystery (x/10) ;;
```



Quels seront les résultats des évaluations successives des phrases suivantes et qu'est-ce qui sera affiché ?

```
# let rec print1 n =
    if n = 0 then
      ()
    else
      begin
        print_int n; print_char ' ';
        print1 (n-1)
      end ;;

# print1 5 ;;
```

```
# let rec print2 n =
    if n = 0 then
      ()
    else
      begin
        print2 (n-1);
        print_int n; print_char ' ';
      end ;;

# print2 5 ;;
```

```
# let rec g n d =
    if n = 0 then
      0
    else
      begin
        print_int n; print_char ' '; print_int d; print_newline();
        if n mod d = 0 then
          1 + g (n/10) d
        else
          g (n/10) (d-1);
      end ;;

# g 435567 7 ;;
```

Exercice 4.2 (Suite)

Écrire une fonction qui calcule pour un $n \in \mathbb{N}$ donné la valeur du $n^{ème}$ terme de la suite définie par $u_n = 4u_{n-1} + 2$ avec $u_0 = 1$.

Exercice 4.3 (Suite géométrique)

Écrire une fonction récursive qui calcule le $n^{ème}$ ($n \in \mathbb{N}$) terme d'une suite géométrique de premier terme u_0 et de raison q.

Exercice 4.4 (Euclide)

Écrire une fonction qui calcule le pgcd de 2 entiers naturels non nuls en utilisant l'algorithme d'Euclide dont le principe est "rappelé" ci-dessous.

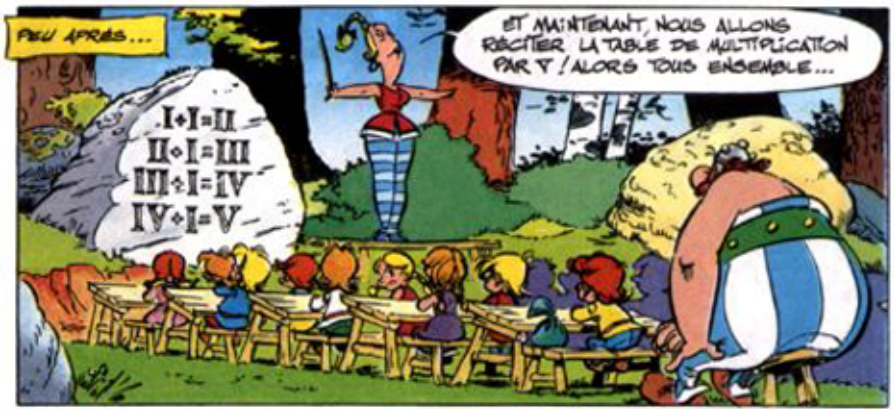
Algorithme d'Euclide :

Si a et b sont deux entiers strictement positifs avec $a \geq b$, si r est le reste non nul de la division euclidienne de a par b : $a = bq + r$ avec $0 < r < b$, alors le pgcd de a et b vaut le pgcd de b et r.

Si a est divisible par b, alors le pgcd de a et b est b.

Exercice 4.5 (Addition)

Écrire une fonction CAML qui calcule $x + y$ ($(x, y) \in \mathbb{N}^2$) en n'ajoutant que des 1.



Exercice 4.6 (Multiplication)

Écrire une fonction qui calcule $x \times y$ ($(x, y) \in \mathbb{N}^2$) en n'utilisant que les opérateurs + et -.

Comment gérer les cas où $(x, y) \in \mathbb{Z}^2$?

Exercice 4.7 (Division euclidienne)

Rappel : À deux entiers naturels a et b, avec b non nul, la division euclidienne associe un quotient q et un reste r, tous deux entiers naturels, vérifiant :

$$a = bq + r, r < b$$

1. Écrire une fonction calculant le quotient entier de a sur b, avec a et b deux entiers naturels, en n'utilisant que des opérateurs additifs (les seules opérateurs autorisés sont + et -).

2. Écrire une fonction calculant le reste de la division entière de a par b en n'utilisant que des opérateurs additifs.

3. **Bonus :** Écrire une fonction qui calcule le quotient et le reste (un couple) de la division euclidienne de deux naturels (utilisant une seule fonction récursive).

Exercice 4.8 (Fibonacci)

Écrire une fonction calculant le $n^{ème}$ terme ($n \in \mathbb{N}$) de la suite de Fibonacci définie par :

$$F(n) = \begin{cases} 1 & si\ n \leq 1 \\ F(n-1) + F(n-2) & sinon \end{cases}$$

Exercice 4.9 (Miroir)

Écrire une fonction qui "inverse" les chiffres d'un entier positif quelconque.

Deux versions :

1. Le résultat sera une chaîne de caractères contenant l'entier "miroir".
   Utiliser la fonction `string_of_int : int -> string`.
   ```
   # reverse 1234 ;;
   - : string = "4321"
   ```
2. Le résultat sera de type entier (sans utiliser la fonction précédente, ni chaînes de caractères).
   ```
   # reverse_int 1234 ;;
   - : int = 4321
   ```

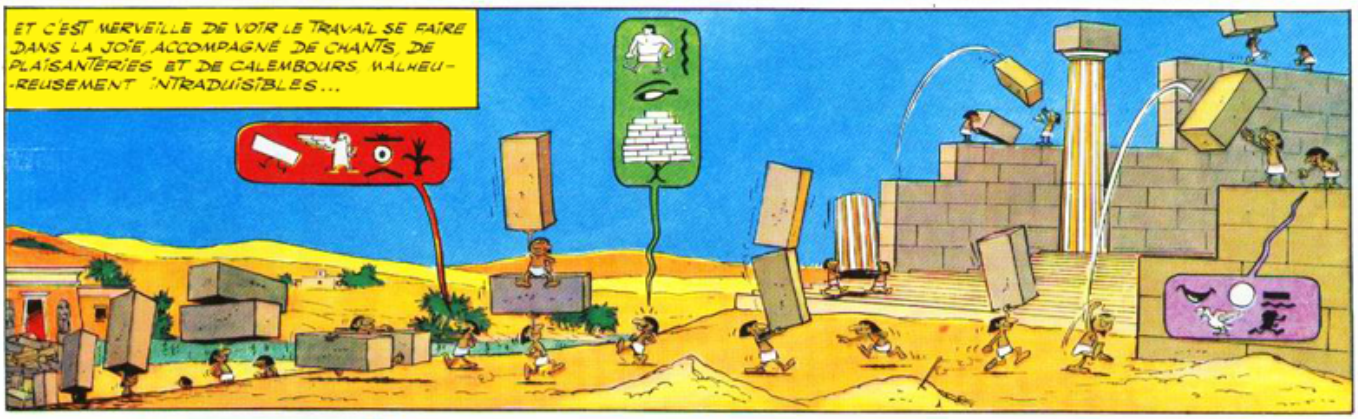
Exercice 4.10 (Multiplication égyptienne)

Écrire une fonction qui calcule $x \times y$ ($(x, y) \in \mathbb{N}^2$) en n'utilisant que des additions, des multiplications par 2 et des divisions par 2.

Indices :

— $10 \times 13 = 2 \times (5 \times 13)$ car 10 est pair.

— $11 \times 13 = 2 \times (5 \times 13) + 13$ car 11 est impair.



Exercice 4.11 (Puissance)

Soient $x \in \mathbb{R}$ et $n \in \mathbb{N}$.

1. Écrire une fonction récursive calculant x^n en utilisant la méthode "classique" (par multiplications successives).
   Quelle est la complexité de cette version ?
2. Écrire une nouvelle fonction calculant x^n en utilisant la méthode de l'*exponentiation rapide* : s'inspirer de la méthode de la multiplication égyptienne (exercice 4.10) pour réduire le nombre de multiplications.
   Quelle est la complexité de cette version ?

Exercice 4.12 (Nombre premier)

Écrire une fonction qui détermine si un entier strictement supérieur à 1 est premier.

Exercice 4.13 (Parfait)

Un nombre est dit parfait s'il est plus grand que 1 et égal à la somme de tous ses diviseurs qui lui sont strictement inférieurs (diviseurs propres).
Exemple : $28 = 1 + 2 + 4 + 7 + 14$.

Écrire une fonction qui détermine si un entier naturel n donné ($n \neq 1$) est parfait.

Bonus :
La fonction pourrait de plus afficher les diviseurs de n.

Exemple d'affichage :

```
Divisors:
1
2
4
7
14
28 is perfect
```

Comment améliorer la complexité de cette fonction ?