1 Fonctions et déclarations de variables

Exercice 1 : Prévoir la réponse de **OCaml** aux instructions suivantes :

```
let a = 42 ;;
let f x = 3*x+1 ;;
f 1 ;;
let f x = 3.*x+1 ;;
```

Exercice 2 : Déterminer le type des fonctions suivantes :

```
let f1 x y = 2*x-3*y ;;
let f2 = fun x y z -> x ;;
let f3 x y = x y ;;
let f4 x = sin x +. x ;;
let f5 x y = if x -. 1. > y then x else y ;;
let f6 f g h = f (g h) ;;
let f7 = fun x y z -> (x, y, z) ;;
let f8 (x, y) = x * (x * y) ;;
let f9 x y = x (x y) ;;
```

Exercice 3 : Écrire une fonction Ocaml `implication` de type `bool -> bool -> bool` implémentant l'implication logique.

Exercice 4 : **OCaml** dispose d'une fonction `min : 'a -> 'a -> 'a`. On considère la fonction suivante :

```
let min3 x y z = min (min x y) z ;;
```

1. Quel est le type de cette fonction ?
2. Que vaut `min3 2 3 5` ?
3. Déterminer (et comprendre) quel est le type de la fonction suivante :

```
let max4 x y z t = max (max (max x y) z) t ;;
```

Exercice 5 :

1. Calculer $\frac{1+\sqrt{2}+\sqrt{2}^3}{1+e^{\sqrt{2}}}$ à l'aide d'une affectation locale.
2. Écrire une fonction **OCaml** `th` implémentant la tangente hyperbolique en n'effectuant qu'un seul appel à la fonction `exp`.

Exercice 6 : Définir une fonction `carre` qui calcule le carré d'un entier.

Exercice 7 : Définir des fonctions **OCaml** prenant en entrée une fonction $f : \mathbb{R} \to \mathbb{R}$ (dont le type est `float -> float =<fun>` *a priori*), et renvoyant :

1. la valeur $\frac{f(0)+f(1)}{2}$;
2. la fonction f^2 (carré de f)
3. la fonction $f \circ f$;
4. la fonction $x \mapsto f(x+1)$.

Exercice 8 : Quel serait le type d'une fonction `c` telle que `c f g` soit la fonction $f \circ g$?
Écrire une telle fonction : objectif 20 caractères.

```
# c (function x -> x+1) (function x-> x*x) 4 ;;
- : int = 17
```

Exercice 9 :

1. Écrire une fonction `nombreChiffres : int -> int` donnant le nombre de chiffres d'un entier naturel écrit en base 10.
2. Modifier votre fonction pour qu'elle prenne en entrée également un entier b et renvoie le nombre de chiffres de l'entier naturel en base b.

Exercice 10 :

1. Quel est le type des fonctions suivantes ?

```
let mult x y = x * y ;;
let mult2(x,y) = x*y ;;
```

2. Proposer une fonction `decurrifier` qui prend en entrée une fonction curryfiée à deux arguments (comme `mult`) et qui renvoie la même fonction mais en version non-curryfiée.
3. Quel est le type de `decurrifier` ?

2 Boucles et références

Exercice 11 : En utilisant une boucle `for`, définir une fonction `puiss` qui calcule la puissance sur les entiers.

```
# puiss 3 7 ;;
- : int = 2187
```

Exercice 12 : En utilisant une boucle `while`, définir une fonction `sdc` qui effectue la somme des chiffres d'un entier. On utilisera des divisions euclidiennes par 10.

```
# sdc 4948353 ;;
- : int = 36
```

Exercice 13 : Écrire une fonction `miroir` qui renvoie le symétrique d'un entier.

```
# miroir 16163223 ;;
- : int = 32236161
```

Exercice 14 : Écrire une fonction `racine x epsilon` qui retourne $\sqrt{x}$ avec une précision de ε, en utilisant l'algorithme de Babylone qui consiste à calculer les termes de la suite suivante (on s'arrête lorsque $\left|u_n - \frac{x}{u_n}\right| < \varepsilon$) :

$$\begin{cases} u_0 = 1 \\ u_{n+1} = \frac{1}{2}\left(u_n + \frac{x}{u_n}\right) \end{cases}$$

La fonction `racine` aura pour type `float -> float -> float`. On pourra utiliser la fonction `abs_float`.

3 Fonctions sur les tableaux

Exercice 15 : Écrire une fonction calculant la somme des éléments d'un tableau d'entiers.

Exercice 16 : Écrire une fonction `inverse t` prenant en entrée un tableau `t` et produisant un nouveau tableau dont les éléments sont dans l'ordre inverse.

Exercice 17 : Même question avec une fonction qui modifie le tableau passé en entrée, c'est-à-dire une fonction de type `'a array -> unit`.

Exercice 18 : Écrire une fonction de tri (en place) d'un tableau, de type `'a array -> unit`. Évidemment on n'utilisera pas `Array.sort`.

4 Filtrage et création de types

Exercice 19 : En mathématiques, on définit la fonction *sinc* par $sinc(x) = \frac{\sin(x)}{x}$ si x est un réel différent de 0 et par 1 si $x = 0$. À l'aide du filtrage par motif, écrire une fonction permettant de calculer cette fonction.

Exercice 20 : Identifier et corriger les erreurs dans les programmes suivants :

```
let f x = 1 - sin x ;;
let g x = match x with
  | x when x<0 -> 0
  | x -> x**2. ;;
```

Exercice 21 : Un **entier de Gauss** est un nombre complexe dont la partie réelle et la partie imaginaire sont entières.

1. (question de maths) Montrer que l'ensemble des entiers de Gauss est stable par conjugaison, addition, multiplication.
2. Définir en **OCaml** un type enregistrement `entierGauss` ainsi qu'une constante I correspondant au nombre complexe i.
3. Définir les fonctions somme, produit et conjugaison sur les entiers de Gauss.

Exercice 22 :

1. Définir à l'aide d'un type somme un type `reelsEtendus` qui peut représenter soit :
   - un réel (à l'aide d'un flottant) ;
   - un infini ($+\infty$ ou $-\infty$) ;
   - un `NaN` (qui signifie **Not a Number**). Ce dernier est obtenu lorsqu'un calcule mène à une forme indéterminée.
2. Définir l'addition et la multiplication sur ce type.

5 Optimisation

Les exercices de cette section sont plus relevés et moins guidés, dans le but d'occuper les plus rapides.

Exercice 23 : Pour un tableau d'entiers (positifs et négatifs) `v`, on veut trouver le sous-tableau de somme maximale : il s'agit donc de trouver un couple d'indices (i, j) tel que $i \leq j$, et $\sum_{k=i}^{j}$ `v.(k)` est maximal.

1. Écrire une fonction qui trouve ce sous-tableau de somme maximale, et qui renvoie la somme.
2. Si ce n'est pas déjà le cas, l'optimiser pour qu'elle s'effectue en temps quadratique ($O(n^2)$).

⋆ 3. Écrire une fonction qui s'effectue ne temps linéaire ($O(n)$).
Indication : on note s_j la somme maximale d'une portion terminant à l'indice j.
Trouver une relation permettant de calculer les s_j en temps $O(n)$.

⋆ **Exercice 24 (Le problème des pièces de monnaie) :** On se donne une collection de valeurs de pièces de monnaies, sous forme d'un tableau `t` (comme `[|1; 2; 5; 10; 20; 50; 100|]`). On souhaite calculer le nombre de façons différentes que l'on a de décomposer une quantité d'argent `s` avec ces pièces.
Écrire une fonction `decomposition t s` résolvant ce problème.
Indication : c'est un problème difficile. On construira une matrice (tableau à deux dimensions) de taille $(s+1) \times (p+1)$, avec p le nombre de pièces. On calculera le nombre de façon que l'on a de décomposer k avec les i premières pièces du tableau de pièces, et ce pour tout $0 \leq k \leq s$ et $0 \leq i \leq p$.