Types Élémentaires

■ ■ ■

I. Expressions & Définitions

Exercice 1. Déterminer le type et la valeur affichés par Caml pour chacune des requêtes suivantes :

```
# 1 + 2;;
# 4 / 2 * 2;;
# -4 / (2 * 2);;
# 5 / 2;;
# 5 mod 2;;
# 1 / 0;;
# 1. + 2. ;;
```

```
# -. 1.0 /. 2.5;;
# 1 +. 3.6;;
# 1.2 + 3.4;;
# sqrt 4.0;;
# sqrt 4;;
# float_of_int 4 /. 2.0;;
# floor 2.6;;
```

Exercice 2. Supposons qu'une variable globale `x` soit définie. Définir une variable `x_power_8` qui utilise exactement 3 multiplications pour calculer x^8.

II. Fonctions

Exercice 3. (Norme d'un vecteur) Écrire une fonction `norme x y` qui retourne la norme du vecteur (x, y).
`norme : float -> float -> float`

Exercice 4. (Test de croissance) Écrire une fonction booléenne `croit` qui prend en argument une fonction f et renvoie `true` si et seulement si $f(0) \leqslant f(1)$.
`croit : (int -> 'a) -> bool`

Exercice 5. (Divisibilité) Écrire une fonction booléenne `divisible n k` qui renvoie `true` si et seulement si `k` divise `n`.
`divisible : int -> int -> bool`

III. Les conditionnelles

Exercice 6. Déterminer le résultat affiché dans chacun des cas suivants.

```
# 1 < 2 || 1.1 < 3.2;;
# 2 > 1 || 3 < 4 && 5 >= 6;;
# (2 > 1 || 3 < 4) && 5 >= 6;;
# not(2 < 1 || 3 < 4);;
```

```
# if 2 > 3 then 4;;
# if if 3 = 4 then 5 > 6 else 7 < 8
  then if 9 < 10 then 1 else 2
  else if 11 > 12 then 3 else 4;;
```

Exercice 7. Écrire une fonction `min3 a b c` qui, étant donné trois éléments `a`, `b` et `c`, renvoie le plus petit des trois.
`min3 : 'a -> 'a -> 'a -> 'a`

Exercice 8. (Années bissextiles) Écrire une fonction `bissextile` qui détermine si une année est bissextile. (Une année est bissextile si elle est divisible par 4, sauf si c'est une année séculaire, sauf tous les 400 ans).
`bissextile : int -> bool`

Exercice 9. (Logique) Écrire une fonction booléenne `imply a b` qui, étant donnés deux booléens, renvoie `a` $\Rightarrow$ `b`.
`imply : bool -> bool -> bool`

Exercice 10. (Gestion des erreurs) Lire et comprendre le code suivant.

```
let racines a b c =
  if a = 0. then  failwith "Ceci n'est pas un trinome"
  else
  let delta = b *. b -. 4. *. a *. c and aa = 2. *. a in
  if delta > 0. then let d = sqrt delta in
  ((-. b -. d) /. aa, (-.b +. d) /. aa)
  else if delta = 0. then (-. b /. aa, -. b /. aa)
  else failwith "pas de racine reelle" ;;
```

IV. Récursivité

Toutes les fonctions programmées dans cette partie devront être récursives.

Exercice 11. Écrire une fonction `s` qui prend en argument un entier n et renvoie la somme $\sum_{k=0}^{n} k$.

```
s : int -> int
```

Exercice 12. (Opérations élémentaires) Écrire une fonction...

1. `somme` qui calcule la somme de deux entiers naturels en n'utilisant que des additions et soustractions de 1.

```
somme : int -> int -> int
```

2. `produit` qui calcule le produit de deux entiers naturels en n'utilisant que des sommes.

```
produit : int -> int -> int
```

Exercice 13. (Puissance) Écrire une fonction `puissance` qui permet de calculer x^n. Vous déterminerez sa complexité en nombre de multiplications mais ne rechercherez pas nécessairement l'optimalité.

```
puissance : int -> int -> int
```

Exercice 14. (Algorithme d'Euclide) Écrire une fonction `pgcd` qui permet de calculer le plus grand commun diviseur de deux entiers.

```
pgcd : int -> int -> int
```

Exercice 15. (Nombres premiers)

1. Écrire une fonction booléenne `multiple_inf n r` qui prend en argument deux entiers `n` et `r` et renvoie `true` si et seulement si `n` possède un diviseur entre 2 et `r`.

```
multiple_inf : int -> int -> bool
```

2. En déduire une fonction booléenne `est_premier n` qui renvoie `true` si et seulement si l'entier `n` est premier.

```
est_premier : int -> bool
```

Exercice 16. (Recherche dichotomique) Écrire une fonction `dicho f a b eps` qui, étant donnée une fonction `f` qui s'annule entre `a` et `b`, renvoie une valeur approchée par défaut d'un zéro de `f` à `eps` près.

```
dicho : (float -> float) -> float -> float -> float
```

Exercice 17. (Tours de Hanoï) On dispose de trois tiges sur lesquelles s'enfilent des disques de tailles différentes. On déplace les disques un à un. Pour pouvoir déposer un disque sur une tige, son diamètre doit être plus petit que celui présent sur le haut de cette tige. Au départ, tous les disques se trouvent sur la tige n°1. Il faut empiler les disques sur la tige n°3. Écrire une fonction résolvant ce problème (vous pourrez l'écrire dans un premier temps en langage naturel). Vous déterminerez sa complexité.

```
hanoi : int -> string -> string -> string -> unit
```

La fonction `print_string` *permet d'afficher une chaîne de caractères, la commande* `^` *permet de les concaténer. Le* `;` *permet d'effectuer plusieurs effets de bords successivement.*

V. Programmation impérative

Exercice 18. (Somme) Écrire une fonction impérative `somme_inverses` qui calcule la somme des inverses des premiers entiers naturels non nuls.

```
somme_inverses : int -> float
```

Exercice 19. (Fonction mystère) Déterminer les résultats de la fonction suivante.

```
let f n =
  let x = ref n and y = ref n in
    while !y <> 0 do
      x := !x + 2;
      y := !y - 1;
    done;
  !x;;
```

Exercice 20. (Fonction mystère)

```
let g n =
  let m = ref 2 in
  for i = 1 to n do
    m := !m * !m
  done;
  !m;;
```

1. Que calcule cette fonction ?

2. Déterminer sa complexité en nombre de multiplications.

Exercice 21. (Nombres parfaits) Écrire une fonction booléenne `est_parfait` qui détermine si un nombre entier est parfait, i.e. est égal à la moitié de la somme de ses diviseurs.
`est_parfait : int -> bool`

Exercice 22. (Division euclidienne) Écrire une fonction `modulo` qui renvoie le reste de la division euclidienne en utilisant l'alogrithme des soustractions successives.
`modulo : int -> int -> int`

Exercice 23. Écrire une fonction `racine_entiere n` qui, étant donné un entier `n`, renvoie le plus grand entier `r` tel que `r * r <= n`.
`racine_entiere : int -> int`

Exercice 24. (Primalité) Écrire une fonction booléenne `est_premier` qui détermine si un nombre entier est premier.
`est_premier : int -> bool`

Exercice 25. (Algorithme de Gentleman) Déterminer le contenu de la variable `b` à l'issue de la suite d'instructions suivantes :

```
let a, b = ref 1., ref 1.;;

while ((!a +. 1.) -. !a) -. 1. = 0. do
   a := 2. *. !a
done;;

while ((!b +. !a) -. !a) -. !b <> 0. do
  b := !b +. 1.
done;;
```