TD n° 2 : Fonctions

Exercice 1 *Définition de fonctions*

Déterminer le type et le comportement des fonctions suivantes :

OCaml
```
let fonction1 = fun f -> f 0;;
let fonction2 f = f 0;;
let fonction3 f g = fun x -> (f x) + (g x);;
let fonction4 f g x = (f x) + (g x);;
let fonction5 = fonction3 (fun x -> x);;
let fonction6 = fun f -> fun x -> f x;;
let fonction7 f = fun x -> f x;;
let fonction8 f x = f x;;
```

Exercice 2

Proposer une fonction Caml pour chacun des types suivants :

1. `int -> (int -> int)`
2. `(int -> int) -> int`
3. `int -> int -> int`

Exercice 3 *Composition*

On considère la fonction suivante :

OCaml
```
let compose f g = fun x -> f (g x)
```

1. Quel est son type ?
2. Pourquoi les parenthèses sont-elles nécessaires ?
3. Redéfinir cette fonction de quatre autres manières différentes.
4. Définir cette fonction en Python.

Exercice 4 *Typage*

Quel est le type des fonctions suivantes ?

OCaml
```
let f1 n m = if n = m then n else m;;
let f2 x y = let z = x + 1 in y || z > 10;;
let f3 x y = x y;;
let f4 x y z = (x y) z;;
let f5 x y z = x (y z);;
let f6 x y z = x y z;;
let f7 (x, y, z) = fun x -> (x, y, z);;
```

Exercice 5

Que pensez-vous de `let max3 x y z = max max x y z;;` ?

Exercice 6 *Curryfication*

Quel est le type des deux fonctions suivantes ?

OCaml
```
let add1 (x, y) = x + y;;
let add2 x y = x + y;;
```

Écrire une fonction `uncurry` qui prend en argument une fonction curryfiée à deux arguments (par exemple `add2`) et retourne la même fonction non curryfiée (par exemple `add1`). Écrire sa réciproque `curry`. Quels sont leurs types ?