LANGAGE OCaml – TP 2
Fonctions et Filtrage - Complément

Exercice 1

1) Ecrire deux fonctions de signature `int -> int = <fun>` renvoyant -1 si l'entier passé en argument est strictement négatif, 0 s'il est nul et 1 s'il est strictement positif. La première fonction utilisera une conditionnelle, la seconde un filtrage.
2) Ecrire deux fonctions de signature `int -> int = <fun>` qui renvoie la valeur absolue d'un entier passé en argument. La première fonction utilisera une conditionnelle, la seconde un filtrage.
3) Ecrire deux fonctions de signature `int -> bool = <fun>` qui testent si la valeur absolue de leur argument est pair. La première appellera une des fonctions définies dans la question précédente et la seconde définira une fonction locale.

Exercice 2

1) Ecrire deux fonctions de signature `'a -> 'a -> 'a = <fun>` qui renvoient le maximum deux entiers passés en argument. La première fonction utilisera le filtrage, la seconde une conditionnelle.
2) Ecrire une fonction de signature `'a -> 'a -> 'a -> 'a = <fun>` qui renvoie le maximum de trois entiers passés en argument en utilisant le filtrage. Trois motifs seront définis.
3) Ecrire une fonction de signature `'a -> 'a -> 'a -> 'a = <fun>` qui renvoie le maximum de trois entiers passés en argument, utilisant le filtrage et appelant la fonction écrite en question 1.
4) Même question que précédemment mais en définissant une fonction locale permettant de déterminer le plus grand de deux entiers.

Exercice 3

1) Ecrire une fonction de signature `int -> int -> int -> int = <fun>` qui détermine le nombre de solutions réelles d'une équation du second degré à coefficients réels. Un `failwith` sera utilisé dans le cas où cette équation serait du premier degré.
2) Ecrire une fonction de signature `int -> int -> int -> unit = <fun>` qui fait appel à la fonction écrite dans la première question et **affiche** les valeurs des racines réelles lorsqu'elles existent.
3) Ecrire une fonction de signature `int -> int -> int -> float * float = <fun>` qui fait appel à la fonction écrite dans la première question et **renvoie** les valeurs des racines réelles lorsqu'elles existent. Dans le cas d'une racine double, la racine double sera renvoyée deux fois ; l'absence d'une racine réelle sera gérée par un `failwith`.

   Ainsi, les appels suivants donneront les sorties indiquées :

`resolution2 0 1 1;;`	`# Exception: Failure "Equation de premier degre".`
`resolution2 1 2 1;;`	`# - : float * float = (-1., -1.)`
`resolution2 1 2 4;;`	`# Exception: Failure "Aucune solution dans R".`
`resolution2 1 (-2) 1;;`	`# - : float * float = (1., 1.)`
`resolution2 1 3 1;;`	`# - : float * float = (-0.3819660112501051, -2.6180339887498949)`