UNIVERSITÉ D'ORLÉANS — DEUG MIAS

Informatique — Année **2001-2002**

Feuille d'exercices OCaml n°2

Exercice 1. Écrire une fonction qui teste si un nombre entier est divisible par 3.

Exercice 2. Écrire des fonctions récursives calculant la somme, la différence, puis le produit de deux nombres entiers, en supposant que les seuls opérateurs dont on dispose sont les comparaisons et les deux fonctions definies ainsi :

```
let succ = function x -> x + 1 and pred = function x -> x - 1 ;;
```

Exercice 3. Écrire une fonction qui prend en argument un réel x et un entier n et qui calcule x^n.

Écrire une autre fonction qui calcule la même chose en utilisant le fait que :

$$x^{2n} = (x^n)^2 \text{ et } x^{2n+1} = x \times (x^n)^2$$

Exercice 4. Écrire une fonction récursive qui prend en paramètres deux entiers `x` et `y` et retourne la valeur de `x div y` (division entière), sans utiliser l'opérateur `div` de Caml.

Par exemple, `14 = 3 * 4 + 2` et donc `14 div 4 = 3`.

Exercice 5. Écrire une fonction *palindrome* qui teste si la chaîne de caractères donnée en argument est un palindrome.

Exercice 6. Écrire une fonction qui étant donnés deux entiers m et n renvoie le $n^{\text{ème}}$ chiffre de l'écriture décimale de m.

Exercice 7. Écrire une fonction qui pour une fonction $f : int \to float$ et deux entiers a et b calcule la somme définie par

$$\sum_{n=a}^{n=b} f(n)$$

Exercice 8. Écrire une fonction qui pour une fonction $f : float \to float$ continue sur $[a, b]$ et trois réels a, b et h tels que $a < b$ et $h < (b - a)$ calcule une approximation de l'intégrale de f entre a et b par $\sum_{n=0}^{n=E(\frac{b-a}{h})} h * f(a + n * h)$ où E calcule la partie entière.

Exercice 9. Méthode de Newton.

Le calcul d'un zéro d'une fonction f dans un intervalle $[a, b]$ peut se faire par la méthode de Newton, en calculant une suite $(z_n)_{n\geq 0}$ par

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

Remarques :

- on dit que a est un zéro pour la fonction f si $f(a) = 0$,
- z_0 doit être *convenablement choisi* dans $[a, b]$, c'est-à-dire tel que la suite (z_n) converge,
- la fonction f
  - est définie et continûment dérivable sur $[a, b]$
  - possède un zéro dans $[a, b]$
- la fonction f' ne s'annule en aucun des points de (z_n)

Pour implanter cette méthode, écrire les fonctions suivantes :

- `derive_en_x` qui étant donnés une fonction $f : float \to float$, un réel h suffisamment petit et un réel x calcule l'approximation de la dérivée de f en x donnée par :

$$f'(x) = \frac{f(x + h) - f(x)}{h},$$

- `assez_précis` qui prend en arguments y et ϵ et qui teste si $-\epsilon < y < +\epsilon$,
- `améliore` qui prend en paramètres f, z_n et ϵ et qui calcule z_{n+1}
- `newton` qui prend en paramètres une fonction f, une précision ϵ et z_0 et qui retourne une approximation d'un zéro de f avec la précision ϵ.

Pour tester la fonction `newton`, écrire une fonction `racine_carrée` qui calcule la racine carrée d'un réel avec une précision donnée.

Exercice 10. Écrire une fonction qui trie trois objets quelconques à l'aide d'une fonction de comparaison donnée en paramètre. Typer la fonction et donner des exemples d'application.

Exercice 11. Soit la déclaration :

```
let double_applic = function f -> function x -> f ( f x) ;;
```

1. Que calcule `(double_applic succ)` ? Quel est son type ?
2. Quel est le résultat de l'application de `double_applic` à
   `(function x -> char_of_int (1 + int_of_char x))`
3. Typer et évaluer cette application.