Corrigé TP 3

let rec Fact n = if n=0 then 1 else n*Fact (n-1) ;;

let rec Exp (x,y) = if y=0 then 1 else x*Exp (x,y-1) ;;

let rec Sigma x = if x=0 then 0 else x+Sigma (x-1) ;;

let rec Sigma2_1 a b = if a=b then a else a+Sigma2_1 (a+1) b ;;

let rec Sigma2_2 a b = if a=b then a else b+Sigma2_2 a (b-1) ;;

let rec Fib n = if n=0 then 0 else if n=1 then 1 else Fib (n-1)+Fib (n-2) ;;

PGCD (algorithme d'Euclide) :

PGCD (a,b)=$\begin{cases} a & si\ b = 0, \\ PGCD\ (b, a\ mod\ b) & sinon \end{cases}$

let rec PGCD (a,b) = if b=0 then a else PGCD (b,a mod b) ;;

PGCD (méthode des soustractions successives) :

let rec PGCD2 (a,b) = if a=b then a else if a>b then PGCD2 (a-b,b) else PGCD2 (a,b-a) ;;

let rec SumSerie n = if n=0 then 0. else 1./.float_of_int n+.SumSerie (n-1) ;;

ndiv_se x i= nombre de diviseurs de x supérieurs ou égaux à i

$$= \begin{cases} 0 & si\ i>x, \\ \text{ndiv\_se x 1} & \text{si i<1}, \\ 1 & si\ i = x, \\ 1+ndiv\_se\ x\ (i+1) & si\ x\ mod\ i=0, \\ ndiv\_se\ x\ (i+1) & sinon \end{cases}$$

Let rec ndiv_se x I = if i>x then 0

else if i<1 then ndiv_se x 1

else if i=x then 1

else if x mod I =0 then 1+ndiv_se x (i+1)

else ndiv_se x (i+1) ;;

Prem x=$\begin{cases} faux & si\ x = 1, \\ vrai\ si\ ndiv\_se\ x\ 1 = 2, \\ faux & sinon \end{cases}$

Let Prem x = if x=1 then false else if ndiv_se x 1=2 then true else false ;;

Corrigé TP 4

Exercice 1 :

```
let rec mult_egypt (x,y)=        if x=0 then 0
                                 else    if x mod 2=0 then mult_egypt (x/2,2*y)
                                         else mult_egypt (x-1,y)+y ;;
```

Exercice 2 :

On utilise la formule suivante : a*b=pgcd a b*ppcm a b

```
let ppcm a b=a*b/pgcd a b ;
```

Exercice 3 :

1)
```
let sc x y z=sub_string x y z ;;
let lc x=string_length x ;;
let rec inv x= if lc x<=1 then x else inv (sc x 1 (lc x -1))^sc x 0 1 ;;
```
2)
```
let palin x=if x=inv x then true else false ;;
```

Exercice 4 :

```
let bissextile x= if x mod 4=0 then        if x mod 100=0 && x mod 400<>0 then false
                                           else true
                  else false ;;
```

Exercice 5 :

1)
```
let rec prod x y=if x<0 then -prod (-x) y else if x=0 then 0 else y+prod (x-1) y ;;
```
2)
```
let rec puiss x y=if y=0 then 1 else prod x (puiss x (y-1)) ;;
```

Exercice 6 :

1)
```
let rec reste x y=if x<y then x else reste (x-y) y ;;
```
2)
```
let rec quotient x y=if x<y then 0 else 1+quotient (x-y) y ;;
```

Corrigé TP 5

01) let rec nbElm l=if l=[] then 0 else 1+nbElm (tl l) ;;

02) let rec somElm l=if l=[] then 0 else hd l+somElm (tl l) ;;

03) let moyElm l=float_of_int (somElm l)/.float_of_int (nbElm l) ;;

04) let g x y=x ::y ;;

La fonction g insère l'élément x au début de la liste y.

05) let h x y=y@(x::[]);;

La fonction h insère l'élément x à la fin de la liste y.

06) let sdElm l=tl l ;;

La fonction sdElm retourne la queue de la liste l, c'est-à-dire l sans son premier élément.

07) let rec sfElm l=if nbElm l=1 then [] else hd l::sfElm (tl l) ;;

La fonction sfElm supprime de la liste l le tout dernier élément (la liste est supposée non vide). En d'autres termes, elle retourne la liste l sans son dernier élément.

08) let rec inv l=if l=[] then [] else inv (tl l)@(hd l ::[]) ;;

La fonction inv inverse la liste l.

09) let rec proj l i=if i=1 then hd l else proj (tl l) (i-1) ;;

La fonction proj retourne la projection numéro i de la liste l, c'est-à-dire le $i^{ème}$ élément de l.

10) let rec appartient x l=if l=[] then false else if x=hd l then true else appartient x (tl l) ;;

11) let rec n_occ x l=if l=[] then 0 else if x=hd l then 1+n_occ x (tl l) else n_occ x (tl l) ;;

12) let rec Map f l=if l=[] then [] else f (hd l)::Map f (tl l);;

let carre x=x*x;;

let Map_c l=Map carre l;;

let rec inv_s x=if x="" then "" else inv_s (sub_string x 1 (string_length x -1))^sub_string x 0 1;;

let Map_i l=Map inv_s l;;

13) Tri d'une liste d'entiers par ordre croissant : avec la méthode du ''tri par sélection''

min l=

- hd l si l a un seul élément,
- min ''l sans le 1er élément'' si ''1er élément de l''>=''2ème élément de l'',
- min ''l sans le 2ème élément'' si ''1er élément de l''<''2ème élément de l''

min l=

- hd l si tl l=[],
- min (tl l) si hd l>=hd (tl l),
- min (hd l :: tl (tl l)) sinon

```
let rec min l=if tl l=[] then hd l+0 else if hd l>=hd (tl l) then min (tl l) else min (hd l :: tl (tl l)) ;;

let rec supp_occ1 x l=if x=hd l then tl l else hd l ::supp_occ1 x (tl l) ;;

let rec tri l=if  l=[] then [] else min l ::tri (supp_occ1 (min l) l) ;;
```

Deuxième solution pour la fonction min :

```
let min_paire (x,y)=if x<=y then x else y ;;

let rec min l=if tl l=[] then hd l+0 else min_paire(hd l,min (tl l)) ;;
```

La fonction min_paire calcule le minimum de deux entiers donnés sous forme de paire. La fonction min calcule le plus petit élément d'une liste d'entiers (liste supposée non vide). La fonction supp_occ1 supprime la première occurrence de l'élément x de la liste l (ici, on suppose que la liste est non vide et contient l'élément x). La fonction tri trie la liste l par ordre croissant. L'expression hd l+0 permet d'imposer le type entier au 1er élément de l, et donc à tous les éléments de l.

Exercices supplémentaires :

14) Ecrire une fonction **sup_rep** supprimant les répétitions dans une liste.

```
let rec sup_occ x l = if l=[] then [] else if x=hd l then sup_occ x (tl l) else hd l :: sup_occ x (tl l) ;;

let rec sup_rep l =if l=[] then [] else if tl l=[] then l else hd l :: sup_rep (sup_occ (hd l) (tl l)) ;;
```

15) Ecrire une fonction **voyelle** testant si un caractère est une voyelle. Utiliser la chaîne de caractères ''aeiouyAEIOUY''.

```
let rec occurre c cc=if cc="" then false else if c=cc.[0] then true else occurre c (sub_string cc 1 (string_length cc-1));;

let voyelle c=occurre c "aeiouyAEIOUY";;
```