Solutions de TPs CAML de l'USTHB

❖ TP2 :

1) let Succ x = x + 1 ;;
2) let Pred x = x – 1 ;;
3) let Sum x y = x + y ;;
4) let Max x y = if x > y then x else y ;;
5) let Max3 x y z =
   if x > y then if x > z then x else z else if y > z then y else z ;;
   - let Max3 x y z = if max x y > z then max x y else z ;;
6) let MinMax x y = if x > y then y, x else x, y ;;
7) let Carre x = x * x ;;
8) let SCarre x y = Carre x + Carre y ;;
9) let ValAbs x = if x < 0 then –x else x ;;
10) let Abs x y = ValAbs (x – y) ;;
11) let pi = 3.14 ;;
    let Surf r = r *. r *. pi ;;
12) let pair x = if x mod 2 = 0 then true else false;;
13) Prem (avec TP3)

    let rec f x i = if i = x then true else if x mod i = 0 then false else f x (i + 1) ;;

    let Prem x = if x = 1 then false else if f x 2 = true then true else false ;;

❖ TP3 :

1) let rec Fact n = if n = 0 then 1 else n * Fact (n – 1) ;;
2) let rec Exp x y = if y = 0 then 1 else x * Exp x (y – 1) ;;
3) let rec Sigma x = if x = 0 then 0 else x + Sigma (x – 1) ;;
4) let Sigma2 a b =

   if a > b then Sigma a – Sigma b – a

   else Sigma b – Sigma a – b ;;

   - let rec Sigma2 a b =

     if a - 1 > b then a – 1 + Sigma2 (a – 1) b

     else if b - 1 > a then b – 1 + Sigma2 a (b – 1) else 0;;

5) let rec Fib n =

   if n = 0 then 0 else if n = 1 then 1

   else Fib (n – 1) + Fib (n – 2) ;;

6) let rec pgcd x y = if x mod y = 0 then y else pgcd y (x mod y) ;;
7) let rec pgcd2 a b =

   if a > b then pgcd (a – b) b

   else if b > a then pgcd a (b – a) else a ;;

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

{Le n est un réel}

❖ TP4 :

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 ;;

2) let rec pgcd x y = if x mod y = 0 then y else pgcd y (x mod y) ;;

   let ppcm x y = x * y / pgcd x y ;;

3) let long x = string_length x – 1 ;;

   let rec inverse x =

    if long x = 0 then x

    else sub_string x (long x) 1 ^ inverse (sub_string x 0 (long x)) ;;

- let palind x = if x = inverse x then true else false ;;

4) let bessex x =

    if (x mod 4 = 0) or ((x mod 100 > 0) && (x mod 400 = 0)) then true else false ;;

5) let rec reste x y = if x >= y then reste (x – y) y else x ;;

- let rec quotient x y = if x >= y then 1 + quotient (x – y) y else 0 ;;

❖ TP5 :

1) let rec nbElm liste = if liste = [] then 0 else 1 + nbElm (tl liste) ;;
2) let rec somElm liste = if liste = [] then 0 else (hd liste) + somElm (tl liste) ;;
3) let moyElm liste = float_of_int(somElm liste) /. float_of_int(nbElm liste) ;;
4) let inserD liste val = [val] @ liste ;; {ou : val :: liste }
5) let inserF liste val = liste @ [val] ;;
6) let suppD liste = tl liste ;;
7) let rec suppF liste = if nbElm liste = 1 then [] else [hd liste] @ suppF (tl liste) ;;
8) let rec inv liste = if liste = [] then [] else inv (tl liste) @ [hd liste] ;;
9) let rec projett liste i =

```
if i > nbelm liste then liste
else if i = 1 then tl liste
else [hd liste] @ projett (tl liste) (i – 1) ;;
```

10) let rec appart liste val =

```
if liste = [] then false
else if hd liste = val then true
else appart (tl liste) val ;;
```

11) let rec occur liste val =

```
if liste = [] then 0
else if hd liste = val then 1 + occur (tl liste) val
else occur (tl liste) val ;;
```

12)

- let f x = x * x ;;

let rec Map liste = if liste = [] then [] else [f (hd liste)] @ Map (tl liste) ;;

- let long x = string_length x – 1 ;;

let rec inverse x =

if long x = 0 then x

else sub_string x (long x) 1 ^ inverse (sub_string x 0 (long x)) ;;

let rec Map2 liste =

if liste = [] then [] else [inverse (hd liste)] @ Map2 (tl liste) ;;

13) trie par sélection (minimum)

- let rec suppVal liste val =

  if liste = [] then []

  else if hd liste = val then tl liste

  else [hd liste] @ suppVal (tl liste) val ;;

- let rec Min liste val =

  if liste = [] then val

  else if hd liste < val then Min (tl liste) (hd liste)

  else Min (tl liste) val ;;

- let rec trie liste =

  if liste = [] then []

  else [Min liste (hd liste)] @ trie (suppVal liste (Min liste (hd liste))) ;;

```
let rec fusion liste1 liste2 =
    if liste1 = [] then liste2
    else if liste2 = [] then liste1
    else if hd liste1 < hd liste2 then [hd liste1] @ fusion (tl liste1) liste2
    else if hd liste2 < hd liste1 then [hd liste2] @ fusion liste1 (tl liste2)
    else [hd liste1] @ fusion (tl liste1) (tl liste2) ;;
```

ex :

fusion [3 ;7 ;11 ;14 ;19] [4 ;5 ;11 ;18 ;19 ;27] = [3 ;4 ;5 ;7 ;11 ;14 ;18 ;19 ;27]