Table of Contents

TP 1 - Programmation pour la préparation à l'agrégation maths option info

- En OCaml.

In [12]:
```
let print = Printf.printf;;

Sys.command "ocaml -version";;
```

Out[12]:
```
val print : ('a, out_channel, unit) format → 'a = <fun>
The OCaml toplevel, version 4.04.2
- : int = 0
```

Fonctions

Exercice 4

In [2]:
```
let successeur (n : int) : int =
  n + 1
;;
```

Out[2]:
```
val successeur : int → int = <fun>
```

In [3]:
```
successeur(3);;
successeur(2 * 2);;
successeur(2.5);;
```

Out[3]:
```
- : int = 4
```

Out[3]:
```
- : int = 5
```

```
¤M¤M¤M¤M¤M# successeur(3);;
  successeur(2 * 2);;
  successeur(2.5);;


File "[3]", line 3, characters 10-15:
Error: This expression has type float but an expression was expected of type
         int
```

Exercice 5

In [4]:
```
let produit3 (x : int) (y : int) (z : int) : int =
  x * y * z
;;

let produit3_2 =
  fun (x : int) (y : int) (z : int) : int ->
    x * y * z
;;

let produit3_3 =
  fun (x : int) ->
    fun (y : int) ->
      fun (z : int) : int ->
        x * y * z
;;

let produit3_4 (tuple : (int * int * int)) : int =
  let x, y, z = tuple in
  x * y * z
;;
```

Out[4]:
```
val produit3 : int → int → int → int = <fun>
```

Out[4]:
```
val produit3_2 : int → int → int → int = <fun>
```

Out[4]:
```
val produit3_3 : int → int → int → int = <fun>
```

Out[4]:
```
val produit3_4 : int * int * int → int = <fun>
```

In [5]:
```
produit3 1 2 3;;
produit3_2 1 2 3;;
(produit3_3 1)(2);;  (* fun (z : int) -> int : 1 * 2 * z *)
((produit3_3 1)(2))(3);;
produit3_4 (1, 2, 3);;
```

Out[5]:
```
- : int = 6
```

Out[5]:
```
- : int = 6
```

Out[5]:
```
- : int → int = <fun>
```

Out[5]:
```
- : int = 6
```

Out[5]:
```
- : int = 6
```

Exercice 6

In [8]:
```
let print = Printf.printf;;

let exercice6 (n : int) : unit =
  for i = 1 to n do
    print "%i\n" i;
  done;
  for i = n downto 1 do
    print "%i\n" i;
  done;
  flush_all ();
;;
```

Out[8]:
```
val print : ('a, out_channel, unit) format → 'a = <fun>
```

Out[8]:
```
val exercice6 : int → unit = <fun>
```

In [10]:
```
exercice6(4)
```

```
1
2
3
4
4
3
2
1
```

Out[10]:
```
- : unit = ()
```

Récursivité

Exercice 7

In [13]:
```
let rec factoriel = function
  | 0 -> 1
  | n -> n * factoriel ( n - 1 );;

let rec factoriel = fun n ->
  match n with
  | 0 -> 1
  | n -> n * factoriel ( n - 1 );;

let rec factoriel (n:int) : int =
  match n with
  | 0 -> 1
  | n -> n * factoriel ( n - 1 );;

let rec factoriel (n:int) : int =
  if n = 0
  then 1
  else n * factoriel ( n - 1 );;
```

Out[13]:
```
val factoriel : int → int = <fun>
```

Out[13]:
```
val factoriel : int → int = <fun>
```

Out[13]:
```
val factoriel : int → int = <fun>
```

Out[13]:
```
val factoriel : int → int = <fun>
```

In [16]:
```
factoriel 4;;
factoriel 0;;

for i = 1 to 8 do
   print "%i! = %i\n" i (factoriel i)
done;;
flush_all ();;
```

Out[16]:
```
- : int = 24
```

Out[16]:
```
- : int = 1
```

Out[16]:
```
- : unit = ()

1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
```

Out[16]:
```
- : unit = ()
```

Exercice 8

In [17]:
```
(* Remarque: si a>b alors pgcd a b = pgcd (a-b) b *)
let rec pgcd (a : int) (b : int) : int =
   if a = b
   then a
   else
      if (a > b)
      then pgcd (a-b) b
      else pgcd a (b-a)
;;
```

Out[17]:
```
val pgcd : int → int → int = <fun>
```

In [18]:
```
pgcd 16 1024;;
pgcd 25 130;;
```

Out[18]:
```
- : int = 16
```

Out[18]:
```
- : int = 5
```

Utilisons le générateur de nombres aléatoires pour faire quelques tests :

In [19]:
```
Random.self_init();;
```

Out[19]:
```
- : unit = ()
```

In [20]:
```
for _ = 1 to 10 do
  let x = 1 + Random.int 100
  and y = 1 + Random.int 100 in
  let d = pgcd x y in
  print " %i ^ %i = %i\n" x y d;
done;;
flush_all ();;
```

Out[20]:
```
- : unit = ()
```

```
 93 ^ 87 = 3
 91 ^ 54 = 1
 84 ^ 4 = 4
 32 ^ 99 = 1
 77 ^ 24 = 1
 83 ^ 22 = 1
 18 ^ 64 = 2
 77 ^ 20 = 1
 14 ^ 21 = 7
 80 ^ 72 = 8
```

Out[20]:
```
- : unit = ()
```

Exercice 9

In [21]:
```
(* fonction naive *)
let rec fibonacci (n : int) : int =
  match n with
  | 0 -> 1
  | 1 -> 1
  | n -> fibonacci (n-1) + fibonacci (n-2)
;;
```

Out[21]:
```
val fibonacci : int → int = <fun>
```

Avec ce morceau de code on peut facilement mesurer le temps d'exécution :

In [22]:
```
let time (f : unit -> 'a) : 'a =
  let t = Sys.time() in
  let res = f () in
  Printf.printf "    Temps en secondes: %fs\n" (Sys.time() -. t);
  flush_all ();
  res
;;
```

Out[22]:
```
val time : (unit → 'a) → 'a = <fun>
```

In [23]:
```
fibonacci 5;;
fibonacci 17;;
```

Out[23]:
```
- : int = 8
```

Out[23]:
```
- : int = 2584
```

In [24]:
```
time (fun () -> fibonacci 40);;
```

```
    Temps en secondes: 4.928000s
```

Out[24]:
```
- : int = 165580141
```

In [26]:
```
let fibonacci_lin (n : int) : int =
  (*  invariant:
     m >= 1
     u = fibo(n-m+1)
     v = fibo(n-m)
     aux m u v = fibo(n) *)
   let rec aux (m : int) (u : int) (v : int) : int =
     assert (m>0);
     if m = 1 then u
     else aux (m-1) (u+v) u
   in aux n 1 1
;;
```

Out[26]:
```
val fibonacci_lin : int → int = <fun>
```

In [27]:
```
for i = 1 to 10 do
   assert ((fibonacci i) = (fibonacci_lin i))
done;;
```

Out[27]:
```
- : unit = ()
```

In [28]:
```
time (fun () -> fibonacci_lin 35);;
```

Out[28]:
```
- : int = 14930352

    Temps en secondes: 0.000000s
```

In [29]:
```
time (fun () -> fibonacci_lin 40);;
```

Out[29]:
```
- : int = 165580141

    Temps en secondes: 0.000000s
```

Voilà la différence.

Exercice 10

Aucune hypothèse n'est faite sur les arguments de la fonction, on supposera seulement qu'elle est itérable sur sa sortie.

In [30]:
```
let rec itere (f : 'a -> 'a) (n : int) : 'a -> 'a =
   match n with
   | 0 -> (fun x -> x);
   | n -> (fun x -> f (itere (f) (n - 1) x))
;;
```

Out[30]:
```
val itere : ('a → 'a) → int → 'a → 'a = <fun>
```

In [31]:
```
(itere (fun x -> x + 1) 10)(1);;
```

Out[31]:
```
- : int = 11
```

Exercice 11

In [33]:
```
let print = Printf.printf;;

let rec hanoi (n : int) (a : string) (b : string) (c : string) : unit =
  if n > 1 then
    hanoi (n-1) a c b;
  print "%s -> %s\n" a c;
  if n > 1 then
    hanoi (n-1) b a c;
  flush_all ();
;;
```

Out[33]:
```
val print : ('a, out_channel, unit) format → 'a = <fun>
```

Out[33]:
```
val hanoi : int → string → string → string → unit = <fun>
```

In [34]:
```
hanoi 1 "T1" "T2" "T3";;
```

```
T1 → T3
```

Out[34]:
```
- : unit = ()
```

In [35]:
```
hanoi 2 "T1" "T2" "T3";;
```

```
T1 → T2
T1 → T3
T2 → T3
```

Out[35]:
```
- : unit = ()
```

In [36]:
```
hanoi 5 "T1" "T2" "T3";; (* 2^5 - 1 = 31 coups *)
```

```
T1 → T3
T1 → T2
T3 → T2
T1 → T3
T2 → T1
T2 → T3
T1 → T3
T1 → T2
T3 → T2
T3 → T1
T2 → T1
T3 → T2
T1 → T3
T1 → T2
T3 → T2
T1 → T3
T2 → T1
T2 → T3
T1 → T3
T2 → T1
T3 → T2
T3 → T1
T2 → T1
T2 → T3
T1 → T3
T1 → T2
T3 → T2
T1 → T3
T2 → T1
T2 → T3
T1 → T3
```

Out[36]:
```
- : unit = ()
```

Avec un compteur de coups joués (ce sera toujours $2^n - 1$) :

In [38]:
```
let rec hanoi2 (n : int) (a : string) (b : string) (c : string) : int =
  if n > 1 then begin
    let coup = ref 0 in
    coup := !coup + (hanoi2 (n-1) a c b);
    print "%s -> %s\n" a c;
    coup := !coup + (hanoi2 (n-1) b a c);
    flush_all ();
    !coup + 1
  end else begin
    print "%s -> %s\n" a c;
    flush_all ();
    1;
  end;
;;
```

Out[38]:
```
val hanoi2 : int → string → string → string → int = <fun>
```

In [39]:
```
hanoi2 2 "T1" "T2" "T3";;
```

```
T1 → T2
T1 → T3
T2 → T3
```

Out[39]:
```
- : int = 3
```

Listes

Exercice 12

Les listes en Python sont des `list` . Elles ne fonctionnent **pas** comme des listes simplement chaînées comme en Caml.

In [40]:
```
let rec concatenation (l1 : 'a list) (l2 : 'a list) : 'a list =
  match l1 with
  | [] -> l2
  | t :: q -> t :: (concatenation q l2)
;;
```

Out[40]:
```
val concatenation : 'a list → 'a list → 'a list = <fun>
```

In [41]:
```
concatenation [1; 2; 3] [4; 5];;
```

Out[41]:
```
- : int list = [1; 2; 3; 4; 5]
```

In [42]:
```
List.append [1; 2; 3] [4; 5];;
```

Out[42]:
```
- : int list = [1; 2; 3; 4; 5]
```

Exercice 13

In [43]:
```
let rec applique (f : 'a -> 'b) (liste : 'a list) : 'b list =
  match liste with
  | [] -> []
  | t :: q -> (f t) :: (applique f q)
;;
```

Out[43]:
```
val applique : ('a → 'b) → 'a list → 'b list = <fun>
```

In [44]:
```
applique (fun x -> x + 1) [1; 2; 3];;
```

Out[44]:
```
- : int list = [2; 3; 4]
```

In [45]:
```
let plus_un_liste = applique ((+) 1);;
plus_un_liste [1; 2; 3];; (* syntaxe sympa *)
```

Out[45]:
```
val plus_un_liste : int list → int list = <fun>
```

Out[45]:
```
- : int list = [2; 3; 4]
```

Exercice 14

Avantage à la notation concise `applique f l` autorise à avoir `(applique f)(l)` au lieu de faire `fun l → applique l f` si les deux arguments étaient dans l'autre sens.

In [46]:
```
let liste_carree = applique (fun x -> x*x);;
```

Out[46]:
```
val liste_carree : int list → int list = <fun>
```

In [47]:
```
liste_carree [1; 2; 3];;
```

Out[47]:
```
- : int list = [1; 4; 9]
```

Exercice 15

In [1]:
```
let rec miroir_quad : 'a list -> 'a list = function
   | [] -> []
   | a :: q -> (miroir_quad q) @ [a]
;;

let miroir_lin (liste : 'a list) : 'a list =
  (* sous fonction utilisant un deuxieme argument d'accumulation du resultat *)
  let rec miroir (l : 'a list) (accu : 'a list) : 'a list =
    match l with
      | [] -> accu
      | a :: q -> miroir q (a::accu)
  in
  miroir liste []
;;
```

Out[1]:
```
val miroir_quad : 'a list → 'a list = <fun>
```

Out[1]:
```
val miroir_lin : 'a list → 'a list = <fun>
```

In [49]:
```
miroir_quad [1; 2; 3];;
miroir_lin [1; 2; 3];;
```

Out[49]:
```
- : int list = [3; 2; 1]
```

Out[49]:
```
- : int list = [3; 2; 1]
```

In [50]:
```
time (fun () -> miroir_quad [1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20]);;
time (fun () -> miroir_lin [1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20]);;
(* Pas de différence observable ici *)
```

```
    Temps en secondes: 0.000000s
```

Out[50]:
```
- : int list =
[20; 19; 18; 17; 16; 15; 14; 13; 12; 11; 10; 9; 8; 7; 6; 5; 4; 3; 2; 1]

    Temps en secondes: 0.000000s
```

Out[50]:
```
- : int list =
[20; 19; 18; 17; 16; 15; 14; 13; 12; 11; 10; 9; 8; 7; 6; 5; 4; 3; 2; 1]
```

In [51]:
```
let longue_liste (n : int) : int list =
  Array.to_list (Array.init n (fun i -> i))
;;
```

Out[51]:
```
val longue_liste : int → int list = <fun>
```

Avec de grandes listes, on voit la différence.

In [52]:
```
let _ = time (fun () -> miroir_quad (longue_liste 10000));;
let _ = time (fun () -> miroir_lin (longue_liste 10000));;
```

```
   Temps en secondes: 1.220000s
```

Out[52]:
```
- : int list =
[9999; 9998; 9997; 9996; 9995; 9994; 9993; 9992; 9991; 9990; 9989; 9988;
 9987; 9986; 9985; 9984; 9983; 9982; 9981; 9980; 9979; 9978; 9977; 9976;
 9975; 9974; 9973; 9972; 9971; 9970; 9969; 9968; 9967; 9966; 9965; 9964;
 9963; 9962; 9961; 9960; 9959; 9958; 9957; 9956; 9955; 9954; 9953; 9952;
 9951; 9950; 9949; 9948; 9947; 9946; 9945; 9944; 9943; 9942; 9941; 9940;
 9939; 9938; 9937; 9936; 9935; 9934; 9933; 9932; 9931; 9930; 9929; 9928;
 9927; 9926; 9925; 9924; 9923; 9922; 9921; 9920; 9919; 9918; 9917; 9916;
 9915; 9914; 9913; 9912; 9911; 9910; 9909; 9908; 9907; 9906; 9905; 9904;
 9903; 9902; 9901; 9900; 9899; 9898; 9897; 9896; 9895; 9894; 9893; 9892;
 9891; 9890; 9889; 9888; 9887; 9886; 9885; 9884; 9883; 9882; 9881; 9880;
 9879; 9878; 9877; 9876; 9875; 9874; 9873; 9872; 9871; 9870; 9869; 9868;
 9867; 9866; 9865; 9864; 9863; 9862; 9861; 9860; 9859; 9858; 9857; 9856;
 9855; 9854; 9853; 9852; 9851; 9850; 9849; 9848; 9847; 9846; 9845; 9844;
 9843; 9842; 9841; 9840; 9839; 9838; 9837; 9836; 9835; 9834; 9833; 9832;
 9831; 9830; 9829; 9828; 9827; 9826; 9825; 9824; 9823; 9822; 9821; 9820;
 9819; 9818; 9817; 9816; 9815; 9814; 9813; 9812; 9811; 9810; 9809; 9808;
 9807; 9806; 9805; 9804; 9803; 9802; 9801; 9800; 9799; 9798; 9797; 9796;
 9795; 9794; 9793; 9792; 9791; 9790; 9789; 9788; 9787; 9786; 9785; 9784;
 9783; 9782; 9781; 9780; 9779; 9778; 9777; 9776; 9775; 9774; 9773; 9772;
 9771; 9770; 9769; 9768; 9767; 9766; 9765; 9764; 9763; 9762; 9761; 9760;
 9759; 9758; 9757; 9756; 9755; 9754; 9753; 9752; 9751; 9750; 9749; 9748;
 9747; 9746; 9745; 9744; 9743; 9742; 9741; 9740; 9739; 9738; 9737; 9736;
 9735; 9734; 9733; 9732; 9731; 9730; 9729; 9728; 9727; 9726; 9725; 9724;
 9723; 9722; 9721; 9720; 9719; 9718; 9717; 9716; 9715; 9714; 9713; 9712;
 9711; 9710; 9709; 9708; 9707; 9706; 9705; 9704; 9703; 9702; 9701;  ... ]
```

```
   Temps en secondes: 0.000000s
```

Out[52]:
```
- : int list =
[9999; 9998; 9997; 9996; 9995; 9994; 9993; 9992; 9991; 9990; 9989; 9988;
 9987; 9986; 9985; 9984; 9983; 9982; 9981; 9980; 9979; 9978; 9977; 9976;
 9975; 9974; 9973; 9972; 9971; 9970; 9969; 9968; 9967; 9966; 9965; 9964;
 9963; 9962; 9961; 9960; 9959; 9958; 9957; 9956; 9955; 9954; 9953; 9952;
 9951; 9950; 9949; 9948; 9947; 9946; 9945; 9944; 9943; 9942; 9941; 9940;
 9939; 9938; 9937; 9936; 9935; 9934; 9933; 9932; 9931; 9930; 9929; 9928;
 9927; 9926; 9925; 9924; 9923; 9922; 9921; 9920; 9919; 9918; 9917; 9916;
 9915; 9914; 9913; 9912; 9911; 9910; 9909; 9908; 9907; 9906; 9905; 9904;
 9903; 9902; 9901; 9900; 9899; 9898; 9897; 9896; 9895; 9894; 9893; 9892;
 9891; 9890; 9889; 9888; 9887; 9886; 9885; 9884; 9883; 9882; 9881; 9880;
 9879; 9878; 9877; 9876; 9875; 9874; 9873; 9872; 9871; 9870; 9869; 9868;
 9867; 9866; 9865; 9864; 9863; 9862; 9861; 9860; 9859; 9858; 9857; 9856;
 9855; 9854; 9853; 9852; 9851; 9850; 9849; 9848; 9847; 9846; 9845; 9844;
 9843; 9842; 9841; 9840; 9839; 9838; 9837; 9836; 9835; 9834; 9833; 9832;
 9831; 9830; 9829; 9828; 9827; 9826; 9825; 9824; 9823; 9822; 9821; 9820;
 9819; 9818; 9817; 9816; 9815; 9814; 9813; 9812; 9811; 9810; 9809; 9808;
 9807; 9806; 9805; 9804; 9803; 9802; 9801; 9800; 9799; 9798; 9797; 9796;
 9795; 9794; 9793; 9792; 9791; 9790; 9789; 9788; 9787; 9786; 9785; 9784;
 9783; 9782; 9781; 9780; 9779; 9778; 9777; 9776; 9775; 9774; 9773; 9772;
 9771; 9770; 9769; 9768; 9767; 9766; 9765; 9764; 9763; 9762; 9761; 9760;
 9759; 9758; 9757; 9756; 9755; 9754; 9753; 9752; 9751; 9750; 9749; 9748;
 9747; 9746; 9745; 9744; 9743; 9742; 9741; 9740; 9739; 9738; 9737; 9736;
 9735; 9734; 9733; 9732; 9731; 9730; 9729; 9728; 9727; 9726; 9725; 9724;
 9723; 9722; 9721; 9720; 9719; 9718; 9717; 9716; 9715; 9714; 9713; 9712;
 9711; 9710; 9709; 9708; 9707; 9706; 9705; 9704; 9703; 9702; 9701;  ... ]
```

Exercice 16

In [53]:
```
let rec insertionDansListeTriee (liste : 'a list) (x : 'a) : 'a list =
  match liste with
  | [] -> [x]
  | t :: q when t < x -> t :: (insertionDansListeTriee q x)
  | _ -> x :: liste  (* x est plus petit que le plus petit de la liste *)
;;
```

Out[53]:
```
val insertionDansListeTriee : 'a list → 'a → 'a list = <fun>
```

In [54]:
```
insertionDansListeTriee [1; 2; 5; 6] 4;;
```

Out[54]:
```
- : int list = [1; 2; 4; 5; 6]
```

In [55]:
```
let triInsertion (liste : 'a list) : 'a list  =
  let rec tri (l : 'a list) (accu : 'a list) : 'a list =
    match l with
    | [] -> accu
    | t :: q -> tri q (insertionDansListeTriee accu t)
  in
  tri liste []
;;
```

Out[55]:
```
val triInsertion : 'a list → 'a list = <fun>
```

In [56]:
```
triInsertion [5; 2; 6; 1; 4];;
```

Out[56]:
```
- : int list = [1; 2; 4; 5; 6]
```

Exercice 17

Pour un ordre, de type `ordre : 'a → 'a → 'a` :

- `x < y` $\Longrightarrow$ `ordre x y = -1`,
- `x = y` $\Longrightarrow$ `ordre x y = 0`,
- `x > y` $\Longrightarrow$ `ordre x y =1`.

In [57]:
```
type 'a ordre = 'a -> 'a -> 'a;;
```

Out[57]:
```
type 'a ordre = 'a → 'a → 'a
```

In [58]:
```
let ordre_croissant : int ordre =
  fun (x : int) (y : int) ->
  match y with
  | yv when yv = x -> 0
  | yv when yv < x -> +1
  | yv when yv > x -> -1
  | _ -> failwith "Erreur comparaison impossible (ordre_decroissant)"
;;

let ordre_decroissant : int ordre =
  fun (x : int) (y : int) ->
  match y with
  | yv when yv = x -> 0
  | yv when yv < x -> -1
  | yv when yv > x -> +1
  | _ -> failwith "Erreur comparaison impossible (ordre_decroissant)"
;;
```

Out[58]:
```
val ordre_croissant : int ordre = <fun>
```

Out[58]:
```
val ordre_decroissant : int ordre = <fun>
```

In [59]:
```
let rec insertionDansListeTrieeOrdre (ordre : 'a ordre) (liste : 'a list) (x : 'a) : 'a list =
  match liste with
  | [] -> [x]
  | t :: q when (ordre t x < 0) -> t :: (insertionDansListeTrieeOrdre ordre q x)
  | _ -> x :: liste  (* x est plus petit que le plus petit de la liste *)
;;
```

Out[59]:
```
val insertionDansListeTrieeOrdre : int ordre → int list → int → int list =
  <fun>
```

In [60]:
```
insertionDansListeTrieeOrdre ordre_decroissant [6; 5; 2; 1; 2] 4;;
```

Out[60]:
```
- : int list = [6; 5; 4; 2; 1; 2]
```

In [61]:
```
let triInsertionOrdre (ordre : 'a ordre) (liste : 'a list) : 'a list  =
  let rec tri (l : 'a list) (accu : 'a list) : 'a list =
    match l with
    | [] -> accu
    | t :: q -> tri q (insertionDansListeTrieeOrdre ordre accu t)
  in
  tri liste []
;;
```

Out[61]:
```
val triInsertionOrdre : int ordre → int list → int list = <fun>
```

In [62]:
```
triInsertionOrdre ordre_decroissant [5; 2; 6; 1; 4];;
```

Out[62]:
```
- : int list = [6; 5; 4; 2; 1]
```

In [63]:
```
triInsertionOrdre ordre_croissant [5; 2; 6; 1; 4];;
```

Out[63]:
```
- : int list = [1; 2; 4; 5; 6]
```

Exponentiation rapide

Exercice 18

In [64]:
```
let rec puiss (x : int) : int -> int = function
  | 0 -> 1
  | n -> x * (puiss x (n-1))
;;
```

Out[64]:
```
val puiss : int → int → int = <fun>
```

Complexité : linéaire ($\mathcal{O}(x)$)...

In [67]:
```
let x = 3 in
for n = 0 to 10 do
   print "  %i ** %i = %i\n" x n (puiss x n);
done;;
flush_all ();;
```

Out[67]:
```
- : unit = ()

  3 ** 0 = 1
  3 ** 1 = 3
  3 ** 2 = 9
  3 ** 3 = 27
  3 ** 4 = 81
  3 ** 5 = 243
  3 ** 6 = 729
  3 ** 7 = 2187
  3 ** 8 = 6561
  3 ** 9 = 19683
  3 ** 10 = 59049
  3 ** 0 = 1
  3 ** 1 = 3
  3 ** 2 = 9
  3 ** 3 = 27
  3 ** 4 = 81
  3 ** 5 = 243
  3 ** 6 = 729
  3 ** 7 = 2187
  3 ** 8 = 6561
  3 ** 9 = 19683
  3 ** 10 = 59049
```

Out[67]:
```
- : unit = ()
```

Exercice 19

In [68]:
```
let rec puissRapide (x : int) : int -> int = function
   | 0 -> 1
   | n when (n mod 2) = 0 -> puissRapide (x * x) (n / 2)
   | n -> (puissRapide (x * x) ((n-1)/2)) * x
   (* Important de mettre * x à droite pour être récursive terminale. *)
;;
```

Out[68]:
```
val puissRapide : int → int → int = <fun>
```

Complexité : logarithmique ($\mathcal{O}(\log_2 x)$).

In [69]:
```
let x = 3 in
for n = 0 to 10 do
   print "  %i ** %i = %i\n" x n (puissRapide x n);
done;;
flush_all ();;
```

Out[69]:
```
- : unit = ()

  3 ** 0 = 1
  3 ** 1 = 3
  3 ** 2 = 9
  3 ** 3 = 27
  3 ** 4 = 81
  3 ** 5 = 243
  3 ** 6 = 729
  3 ** 7 = 2187
  3 ** 8 = 6561
  3 ** 9 = 19683
  3 ** 10 = 59049
```

Out[69]:
```
- : unit = ()
```

Exercice 20

In [1]:
```
(* le type monoide *)
type 'a monoide = { mult : 'a -> 'a -> 'a; neutre : 'a };;
```

Out[1]:
```
type 'a monoide = { mult : 'a → 'a → 'a; neutre : 'a; }
```

Avec des champs d'enregistrement, c'est concis :

In [2]:
```
let floatMonoide : 'float monoide = {
    mult = ( *. ); (* fun x y -> x *. y *)
    neutre = 1.
};;
```

Out[2]:
```
val floatMonoide : float monoide = {mult = <fun>; neutre = 1.}
```

Par contre, impossible d'avoir un neutre de taille quelconque donc on doit écrire un monoied pour les matrices qui soit dépendent d'une taille n.

In [3]:
```
let mult_matrice (x : int array array) (y : int array array) : int array array =
    let n = Array.length x in
    let z = Array.make_matrix n n 0 in
    for i = 0 to n-1 do
        for j = 0 to n-1 do
            for k = 0 to n-1 do
                z.(i).(j) <- z.(i).(j) + x.(i).(k) * y.(k).(j)
            done
        done
    done;
    z
;;
```

Out[3]:
```
val mult_matrice : int array array → int array array → int array array =
  <fun>
```

In [4]:
```
mult_matrice [|[|1; 1|]; [|1; 1|]|] [|[|1; 2|]; [|3; 4|]|];;
```

Out[4]:
```
- : int array array = [|[|4; 6|]; [|4; 6|]|]
```

Manuellement ce n'est pas trop dur :

In [5]:
```
let matrixMonoide n = {
    mult = mult_matrice;
    neutre = Array.init n (fun i -> Array.init n (fun j -> if i = j then 1 else 0));
};;
```

Out[5]:
```
val matrixMonoide : int → int array array monoide = <fun>
```

Exercice 21

Première approche naïve :

In [6]:
```
let rec exp_rapide (m : 'a monoide) (x : 'a) (n : int) : 'a =
    match n with
    | 0 -> m.neutre
    | n -> m.mult (exp_rapide m x (n-1)) x
;;
```

Out[6]:
```
val exp_rapide : 'a monoide → 'a → int → 'a = <fun>
```

Avec l'approche récursive :

In [10]:
```
let rec exp_rapide (m : 'a monoide) (x : 'a) (n : int) : 'a =
  match n with
  | 0 -> m.neutre
  | n when (n mod 2) = 0 -> exp_rapide m (m.mult x x) (n / 2)
  | n -> m.mult (exp_rapide m (m.mult x x) ((n-1)/2)) x
;;
```

Out[10]:
```
val exp_rapide : 'a monoide → 'a → int → 'a = <fun>
```

Exercice 22

In [11]:
```
let exp_rapide_float = exp_rapide floatMonoide;;
```

Out[11]:
```
val exp_rapide_float : float → int → float = <fun>
```

In [12]:
```
exp_rapide_float 2.0 8;;
```

Out[12]:
```
- : float = 256.
```

In [13]:
```
exp_rapide_float 0.2 8;;
```

Out[13]:
```
- : float = 2.5600000000000217e-06
```

Et pour les matrices, un petit piège à cause des tailles :

In [79]:
```
let exp_rapide_mat x n = exp_rapide (matrixMonoide (Array.length x)) x n;;
```

Out[79]:
```
val exp_rapide_mat : int array array → int → int array array = <fun>
```

In [80]:
```
exp_rapide_mat [|[|1; 1|]; [|1; 1|]|] 0;;
exp_rapide_mat [|[|1; 1|]; [|1; 1|]|] 1;;
exp_rapide_mat [|[|1; 1|]; [|1; 1|]|] 2;;
exp_rapide_mat [|[|1; 1|]; [|1; 1|]|] 3;;
exp_rapide_mat [|[|1; 1|]; [|1; 1|]|] 4;;
```

Out[80]:
```
- : int array array = [|[|1; 0|]; [|0; 1|]|]
```

Out[80]:
```
- : int array array = [|[|1; 1|]; [|1; 1|]|]
```

Out[80]:
```
- : int array array = [|[|2; 2|]; [|2; 2|]|]
```

Out[80]:
```
- : int array array = [|[|4; 4|]; [|4; 4|]|]
```

Out[80]:
```
- : int array array = [|[|8; 8|]; [|8; 8|]|]
```

In [81]:
```
(* nilpotente ! *)
exp_rapide_mat [|[|0; 1; 2|]; [|0; 0; 1|]; [|0; 0; 0|]|] 0;;
exp_rapide_mat [|[|0; 1; 2|]; [|0; 0; 1|]; [|0; 0; 0|]|] 1;;
exp_rapide_mat [|[|0; 1; 2|]; [|0; 0; 1|]; [|0; 0; 0|]|] 2;;
exp_rapide_mat [|[|0; 1; 2|]; [|0; 0; 1|]; [|0; 0; 0|]|] 3;;
exp_rapide_mat [|[|0; 1; 2|]; [|0; 0; 1|]; [|0; 0; 0|]|] 4;;
```

Out[81]:
```
- : int array array = [|[|1; 0; 0|]; [|0; 1; 0|]; [|0; 0; 1|]|]
```

Out[81]:
```
- : int array array = [|[|0; 1; 2|]; [|0; 0; 1|]; [|0; 0; 0|]|]
```

Out[81]:
```
- : int array array = [|[|0; 0; 1|]; [|0; 0; 0|]; [|0; 0; 0|]|]
```

Out[81]:
```
- : int array array = [|[|0; 0; 0|]; [|0; 0; 0|]; [|0; 0; 0|]|]
```

Out[81]:
```
- : int array array = [|[|0; 0; 0|]; [|0; 0; 0|]; [|0; 0; 0|]|]
```

Exercice 23

In [82]:
```
let monoideFonction = {
   mult = (fun f g x -> f (g x) );
   neutre = (fun x -> x)
};;

let itereMonoide f n = exp_rapide monoideFonction f n;;
```

Out[82]:
```
val monoideFonction : ('a → 'a) monoide = {mult = <fun>; neutre = <fun>}
```

Out[82]:
```
val itereMonoide : ('a → 'a) → int → 'a → 'a = <fun>
```

In [83]:
```
itereMonoide (fun x -> x + 1) 10 0;;
itereMonoide ((+) 1) 10 0;;
```

Out[83]:
```
- : int = 10
```

Out[83]:
```
- : int = 10
```

Formule du calcul propositionnel

Exercice 24

In [2]:
```
type variable = string;;

type formule =
   | V of variable
   | Non of formule
   | Conj of formule * formule
   | Disj of formule * formule
;;
```

Out[2]:
```
type variable = string
```

Out[2]:
```
type formule =
    V of variable
  | Non of formule
  | Conj of formule * formule
  | Disj of formule * formule
```

In [3]:
```
let f = (
   Conj(
      Non(V("p")),
      Disj(
         Conj(V("q"), Non(V("p"))),
         Disj(V("r"), V("q"))
      )
   )
) ;;
```

Out[3]:
```
val f : formule =
  Conj (Non (V "p"), Disj (Conj (V "q", Non (V "p")), Disj (V "r", V "q")))
```

Exercice 25

In [4]:
```
let rec taille : formule -> int = function
   | V(_) -> 1
   | Non(f) -> 1 + (taille f)
   | Conj(f,g) -> 1 + (taille f) + (taille g)
   | Disj(f,g) -> 1 + (taille f) + (taille g)
;;
```

Out[4]:
```
val taille : formule → int = <fun>
```

In [87]:
```
taille f;;
```

Out[87]:
```
- : int = 11
```

Exercice 26

In [5]:
```
let rec formule_to_string : formule -> string = function
    | V(p) -> p
    | Non(f) -> "non "^(formule_to_string f)
    | Conj(f,g) -> "("^(formule_to_string f)^" ^ "^(formule_to_string g)^")"
    | Disj(f,g) -> "("^(formule_to_string f)^" v "^(formule_to_string g)^")"
;;
```

Out[5]:
```
val formule_to_string : formule → string = <fun>
```

In [6]:
```
let print = Printf.printf;;

let affiche (f : formule) : unit =
    print "%s\n" (formule_to_string f);
    flush_all ();
;;
```

Out[6]:
```
val print : ('a, out_channel, unit) format → 'a = <fun>
```

Out[6]:
```
val affiche : formule → unit = <fun>
```

In [7]:
```
affiche f;;
```

```
(non p ^ ((q ^ non p) v (r v q)))
```

Out[7]:
```
- : unit = ()
```

Et voilà. Pas si difficile non ?

Exercice 27

Les valeurs des variables seront données comme une fonction associant nom de variable à valeurs booléennes. On a l'avantage de pouvoir mettre les valeurs par défaut à `true` ou `false` via la filtration.

In [8]:
```
type valuation = variable -> bool;;

let rec eval (v : valuation) : formule -> bool = function
    | V(x) -> v(x)
    | Non(f) -> not (eval v f)
    | Conj(f,g) -> (eval v f) && (eval v g)
    | Disj(f,g) -> (eval v f) || (eval v g)
;;

let valuFalse : valuation = function
    | "p" -> true
    | "q" -> false
    | "r" -> false
    | _ -> false
;;

let valuTrue : valuation = function
    | "p" -> false
    | "q" -> false
    | "r" -> true
    | _ -> false
;;
```

Out[8]:
```
type valuation = variable → bool
```

Out[8]:
```
val eval : valuation → formule → bool = <fun>
```

Out[8]:
```
val valuFalse : valuation = <fun>
```

Out[8]:
```
val valuTrue : valuation = <fun>
```

In [93]:
```
eval valuTrue f;;
```

Out[93]:
```
- : bool = true
```

In [94]:
```
eval valuFalse f;;
```

Out[94]:
```
- : bool = false
```

Exercice 28

In [9]:
```
let rec inserUneFois (x : 'a) : ('a list -> 'a list) = function
    | [] -> [x]
    | t :: q when (x = t) -> t :: q
    | t :: q -> t :: (inserUneFois x q)
;;
```

Out[9]:
```
val inserUneFois : 'a → 'a list → 'a list = <fun>
```

In [10]:
```
let recupererVariable (f : formule) : variable list =
    let rec recup (l : variable list) : formule -> variable list = function
        | V(x) -> inserUneFois x l
        | Non(f) -> recup l f
        | Conj(f,g) -> recup (recup l f) g
        | Disj(f,g) -> recup (recup l f) g
    in
    recup [] f
;;
```

Out[10]:
```
val recupererVariable : formule → variable list = <fun>
```

In [11]:
```
recupererVariable f;;
```

Out[11]:
```
- : variable list = ["p"; "q"; "r"]
```

In [12]:
```
let rec nouvelleValu (v : valuation) : 'a list -> valuation = function
  | [] -> v
  | t :: q ->
  if (v t) then
    let nv x = if (x = t) then false else v x in
    nouvelleValu nv q
  else function x ->
    if (x = t) then true else v x
;;
```

Out[12]:
```
val nouvelleValu : valuation → variable list → valuation = <fun>
```

In [13]:
```
let rec isTrue (v : valuation) : variable list -> bool = function
  | [] -> true
  | t :: q -> if v t then isTrue v q else false
;;
```

Out[13]:
```
val isTrue : valuation → variable list → bool = <fun>
```

In [14]:
```
let rec valuToString (v : valuation) : variable list -> string = function
  | [] -> ""
  | t :: q -> (if v t then "1" else "0") ^ " " ^ (valuToString v q)
;;
```

Out[14]:
```
val valuToString : valuation → variable list → string = <fun>
```

In [17]:
```
let print = Printf.printf;;

let rec printVariableList : variable list -> unit = function
  | [ ] -> print "| "
  | t :: q ->
    print "%s " t;
    flush_all ();
    printVariableList q
;;
```

Out[17]:
```
val print : ('a, out_channel, unit) format → 'a = <fun>
```

Out[17]:
```
val printVariableList : variable list → unit = <fun>
```

In [18]:
```
let tableVerite (f : formule) : unit =
  let variables = recupererVariable f in
  let valu = ref (function _ -> false) in
  (* on construit dynamiquement la nouvelle valuation... moche mais ça marche *)
  printVariableList variables;
  affiche f;
  while not (isTrue (!valu) variables) do
    print_string ( (valuToString (!valu) variables)^"| "^(if eval (!valu) f then "1" else "0")^"\n");
    valu := nouvelleValu (!valu) variables
  done
;;
```

Out[18]:
```
val tableVerite : formule → unit = <fun>
```

In [19]:
```
tableVerite f;;
```

```
p q r | (non p ^ ((q ^ non p) v (r v q)))
```

Out[19]:
```
- : unit = ()
```

On peut vérifier, par exemple sur Wolfram|Alpha (https%3A%2F%2Fwww.wolframalpha.com%2Finput%2F%3Fi%3D%28%7Ep%2B%2526%2B%28%28q%2B%2526%2B%
que l'on obtient bien le bon résultat...

Conclusion

Voilà pour aujourd'hui !

Cette solution est aussi disponible en Python : TP1__Python.ipynb (https://nbviewer.jupyter.org/github/Naereen/notebooks/tree/master/agreg/TP_Programmation_2017-18/TP1__Python.ipynb/).

Là où Caml excelle pour les types définis, le filtrage et la récursion, Python gagne en simplicité sur l'affichage, sa librairie standard et les dictionnaires et ensembles