



Programmation fonctionnelle CM1

Evaluation conditionnel

```
f :: Integer -> Integer
f n = if n == 0 then 1
      else n * f (n-1)
main :: IO ()
main = print (f 5)
```

Output 120

Filtrage par motif (Pattern matching)

```
f 0 = 1
f n = n * f (n-1)
```

```
f n = case n of
  0 -> 1
  _ -> n * f (n-1)
main = print(f 5)
```

Les noms de fonctions et variables commencent par une minuscule

```
square x = x*x
inc x = x + 1

f x = (inc.square) x -- ou juste f x = inc (square x)
main = print( f 3 )
```

Output 10

Evaluation gradée

```
signe n
  | n < 0    = -1
  | n > 0    = 1
  | otherwise = 0
main = print(signe 10)
```

Output 1

Fonction récursive terminale

```
f' 0 r = r
f' n r = f' (n-1) (n*r)
f n = f' n 1
main = print (f 3)
```

Output 6

Ex slide 15

```
puiss x 0 = 1
puiss x n = x * (puiss x (n-1))
main = print (puiss 4 4)
```

Output 256

```
p x n = case n of
  0 -> 1
  _ -> if (even n)
    then p (x*x) (div n 2)
    else x* p (x*x) (div n 2)
main = print (p 4 3)
```

Output 64

```
fib0 0 = 1
fib0 1 = 1
fib0 n = fib0(n-1) + fib0(n-2)
main = print (fib0 4)
```

Output 5

Ex slide 16

La hauteur palindromique(123->321)

```

r :: Integer -> Integer    -- fonction donnée
raux n x = if n == 0
    then x
    else raux (div n 10) (10*x + (mod n 10))
r n = raux n 0
main = print (r 123)

```

Output 123

```

r :: Integer -> Integer    -- fonction donnée
raux n x = if n == 0
    then x
    else raux (div n 10) (10*x + (mod n 10))
r n = raux n 0

hpal n = if (r n) == n -- calcule de fois
    then 0
    else 1 + hpal(n + r n)
main = print (hpal 454)

```

Output 0

Déclaration locale

```

sommeCube x y = let z = x + y in z*z*z
main = print(sommeCube 2 1)

```

```

sommeCube x y = let add u v = u + v
                  z = x + y
                  in z*z*z
main = print(sommeCube 2 1)

```

Output 27

Ex slide 22

Ecrire une fonction et qui réalise le et logique entre deux Bool

```

et :: Bool -> Bool -> Bool
et True True = True
et _ _ = False
main = print(et True True)

```

Output `True`

Ex : Type algébrique slide 23

Nom de type commerce par MAJUSCULE

```
data Point = Coord Double Double
distance :: Point -> Point -> Double
distance (Coord x1 y1 ) (Coord x2 y2 ) = sqrt( (x1-x2)^2 + (y1-y2)^2 )
main = let p1 = Coord 1.1 1.1 in print (distance p1 (Coord 3.1 3.1))
```

Output `2.8284271247461903`

```
data Point = Coord Double Double
distance :: Point -> Point -> Double
distance (Coord x1 y1 ) (Coord x2 y2 ) = sqrt( (x1-x2)^2 + (y1-y2)^2 )
--constructeur de donnée
data Figure = FigP Point | FigC Double | FigCa Point Point
-- type de sortie de donnée est Double
perimetre(FigP _ ) = 0.0
perimetre(FigC r) = 2*3.14*r
perimetre(FigCa p1 p2) = (distance p1 p2 ) * 2 * sqrt 2
main = print (perimetre(FigP (Coord 2.1 2.1)))
```

Output `0.0`

```
main = print (perimetre(FigC 2.1))
```

Output `13.188`

```
main = print (perimetre (FigCa (Coord 2.1 2.1) (Coord 3.1 3.1)))
```

Output `4.0000000000000001`

erreur 26 test.hs ???

Les tuples

fst et snd pour les couples

pourquoi ??? page 26

```
maxi ::(Integer, Integer)-> Integer
maxi x = let a = fst x
          b = snd x in
          if a > b then a else b
main =print (maxi (20,3))
```

pattern matching

```
maxi ::(Integer, Integer)-> Integer
maxi (a,b) = if a > b then a else b
main =print (maxi (20,3))
```

Output 20

???

Pas que pour les couples

```
ror :: (Char, Char,Char) ->(Char, Char,Char)
ror(a,b,c) = (c,b,a)
main = print (r)
```

Type: les enregistrements

```
data Canard = Coin {nom::String, enverg:: Double}

info::Canard-> String
info c= "Oh le beau canard " ++ (nom c) ++
        " d'envergure " ++ show (enverg c) ++ " "

main = let c1 = Coin "Coin" 0.5
          c2 = Coin {nom = "Gaga" , enverg = 0.2}
          c3 = c2 {enverg= 0.3}
        in
          print(info c1 ++ info c3)
```

Output

```
Oh le beau canard Coin d'envergure 0.5 Oh le beau canard Gaga d'envergure 0.3
```

Types algébriques récursifs

??? comment tester

```
data Nat = Zero | Succ Nat
intVal :: Nat -> Integer
intVal Zero = 0
intVal (Succ x) = (intVal x) + 1

addition :: Nat -> Nat -> Nat
addition (Succ x) y = addition x (Succ y) -- ou Succ(addition x y)
addition (Succ x) Zero = Succ x
addition (Succ x) (Succ y) = addition x (Succ (Succ y))
main = print (intVal(addition (Succ Zero) (Succ Zero)))
```

Ex liste et arbres slide 29

```
data Liste = Vide | Cons Integer Liste
somme :: Liste -> Integer
somme Vide = 0
somme (Head x l) = x + somme l --permettre de parcourir tous les éléments dans la liste

data ArbreBinaire = VideA | Head Integer ArbreBinaire ArbreBinaire
hauteur VideA = 0
hauteur (Noeud x g d) = 1 + max (hauteur g) (hauteur d) --gauche et droit
```