Les objets mutables et les conditions

Plan

- Les objets mutables et immutables
- Expressions conditionnelles et définitions
- Boucles

Syntaxe basée sur l'indentation

F# utilise l'indentation du code pour le comprendre et le compiler correctement

```
let a =
  let x = 10
  let y = 15
  x * y
 let b = a % 5
 let c =
  if b < 3 then
    "oui"
  else
    "non"
 let d = c.[0..1]
 printfn "a = %d, b = %d, c = %s, d = %s" a b c d
```

Objets mutables

- Vous pouvez utiliser F # pour la programmation fonctionnelle pure
- Certains problèmes sont reliés aux I/O sont presque impossibles à résoudre sans un changement d'état.
- Solution: utiliser des identificateurs mutables dont les valeurs peuvent changer au fil temps.

Le type unit

- Toute fonction qui n'accepte pas ou ne retourne pas des valeurs est une fonction de type unit (comme void dans C#)
- Programmeur fonctionnel: une fonction qui n'accepte pas ou ne renvoie pas une valeur semble ne pas être intéressante (ne fait rien)
- Chaque fois que vous voulez une fonction qui ne prend pas ou ne retourne pas de valeur, vous mettez () dans le code:

```
let aFunction() =
        ()
```

- Vous pouvez regrouper plusieurs affichages dans une fonction de type Unit. Mais faites attention à l'indentation:

```
let poem() =
    printfn “message 1"
    printfn “message 2"
    printfn “message 3"
    printfn “message 4"
poem()
```

Variable Mutable

- Dans certaines circonstances, vous modifier, les identifiants.
- Dans un langage impure (cas de F#) on utilise la close **mutable**

```
let mutable phrase = “Cours 420-GEN-HY"
// imprime la phrase
printfn "%s" phrase
// mise à jour de la phrase
phrase <- “Les étudiants du cours 420-GEN-HY  sont super"
// reimprime la phrase
printfn "%s" phrase
```

Variable Mutable

```
let redefineX() =
    let x = "One"
    printfn "Redefining:\r\nx = %s" x
    if true then
        let x = "Two"
        printfn "x = %s" x
    printfn "x = %s" x
// demonstration of mutating X
```

Variable Mutable

```
let mutableX() =
        let mutable x = "One"
        printfn "Mutating:\r\nx = %s" x
        if true then
            x <- "Two"
            printfn "x = %s" x
        printfn "x = %s" x
// run the demos
redefineX()
mutableX()
```

Les conditions

- Plusieurs facons d'appliquer les conditions:
- if /then :

  if <expr1> then <expr2>
- if/then/ else:

  if <expr1> **then** <expr2> **else** <expr3>
- if/then/elif/else

  if <expr1> **then** <expr2> elif <expr3>**else**
- If imbriqués:

```
if expr1 then
        expr2
        if expr3 then
                expr4
        else
                expr5
else
        expr6
```

Exemples sur les conditions

```
let a : int16 = 10s
if (a < 20s) then
   printfn "a est petit que 20\n"
   printfn "La valeur de a est: %d" a
```

Exemples sur les conditions

```
let a : int16 = 21s
if (a < 20s) then
   printfn "a est petit que 20\n"
   printfn "La valeur de a est: %d" a
else
    printfn "a est plus grand ou égale 20\n"
```

Exemples sur les conditions

```
let a : int32 = 100
if (a = 10) then
   printfn "valeur de a est 10\n"
elif (a = 20) then
   printfn " valeur de a est 20\n"
elif (a = 30) then
   printfn " valeur de a est 30\n"
else
   printfn "a non trouvé\n"
   printfn "valeur de a est: %d" a
```

Exemples sur les conditions

```
let a : int32 = 100
let b : int32 = 200
if (a = 100) then
   if (b = 200) then
      printfn “valeur de a est 100 et b est 200\n"
printfn “a: %d" a
printfn “b: %d" b
```

Quelques expressions mal typées

- Attention à ce qu'elle retourne une condition:
- **if** 5 **then** 1 **else** 2 *// 5 n'est pas un booléen*
- **if true then** 5.5 **else** 3 *// 5.5 et 3 n'ont pas le même type*

Déclarations locale

- Pour associer un nom à une valeur, on utilise la construction **let..in**

let <ident> = <expr1> **in** <expr2>

let x = 6 **in** x * x;;

Affichez x!!

let x = "a" **in** x + x + x;;

Affichez x!!

À refaire

Let x=10;

let x = 6 **in** x * x;;

Affichez x!!!!

- un bloc "let in" étant lui-même une expression et il est possible de les imbriquer

let x = 5 **in let** y = x + 1 **in** x + y;;

- Partout, absolument partout, où l'on attend une expression, il est possible de définir localement une valeur

[| 1 .. **let** x = 3 **in** x * x |];;

Boucles

- for… to
- for… downto
- for … in
- While…do

for… to/for… downto

for var = start-expr to end-expr do

... // boucle

Exemples:

```
let main() =
   for i = 1 to 20 do
      printfn "i: %i" i
main()
```

```
let main() =
   for i = 20 downto 1 do
      printfn "i: %i" i
main()
```

for … in

```
let list1 = [ 10; 25; 34; 45; 78 ]
for i in list1 do
   printfn "%d" i
```

While.. do

- while test-expression do body-expression

```
let mutable a = 1
while (a < 10) do
   printfn “valeur de a: %d" a
   a <- a + 1
```

Pattern Matching (filtrage de motif ☺)

- Le filtrage vous permet de " comparer les données avec une structure ou des structures logiques , décomposer les données en parties constitutives , ou extraire des informations à partir des données de diverses manières"
- Il fournit un moyen plus flexible et puissant de tester les données contre une série de conditions et en effectuant des calculs basés sur la condition remplies.

Pattern Matching (filtrage de motif ☺)

```
match expr with
| pat1 - >result1
| pat2 -> result2
| pat3 when expr2 -> result3
| _ -> defaultResult
```

Pattern Matching (filtrage de motif ☺)

```
[<Literal>]
let Three = 3
let filter123 x =
    match x with
    // The following line contains literal patterns combined with an OR pattern.
    | 1 | 2 | Three -> printfn "Found 1, 2, or 3!"
    // The following line contains a variable pattern.
    | var1 -> printfn "%d" var1
for x in 1..10 do filter123 x
```

Exercices

- Écrivez un programme capable de calculer itérativement le factoriel d'un nombre
- Écrivez un programme capable d'afficher les nombres paires entre deux bornes x et y
- Écrivez un programme calculatrice qui prends en paramètre deux en paramètre deux nombres et un opérande. L'opérande peut être: +, -, *, /, puis, rac, fact, trun, min, max.
  - Puis: c'est la puissance de deux nombre x puis y = x^y
  - Rac: c'est la racine carré du premier terme.
  - Fact: est le factoriel des deux nombres
  - Trun: c'est la partie entière des deux nombres