Programmation fonctionnelle

Trois principaux paradigmes de programmation

- Le paradigme fonctionnel
  - Paradigme de programmation qui considère le calcul en tant qu'évaluation de fonctions mathématiques.
  - But: concevoir des programmes comme des fonctions mathématiques que l'on compose entre elles.
- Le paradigme procédural (ou impératif)
  - paradigme qui se fonde sur le concept d'appel procédural.
  - Une procédure, aussi appelée routine, sous-routine
- Le paradigme à objets:
  - manipulation des objets (ensemble groupé de variables et méthodes associées à des entités)

Programmation fonctionnelle

- La programmation fonctionnelle est un paradigme de programmation qui considère le calcul en tant qu'évaluation de fonctions mathématiques.
- La programmation fonctionnelle est un style de programmation dont les concepts fondamentaux sont les notions de valeur , de fonction et d‘ application d'une fonction à un valeur

Programmation fonctionnelle

- Dans la programmation fonctionnelle:
  - Les fonctions sont des objets de première classe
  - La récursivité est toujours favorisée comme structure de contrôle
  - Une grande utilisation des listes
  - l'affectation est déconseillé

  - Un programme n'est pas une séquence d'instructions mais une expression à évaluer et des définitions de fonctions

Pourquoi utiliser la programmation fonctionnelle

- Écrire moins de code
- Écrire du code plus robuste
- Écrire du code plus expressif

- “The ratio of time spent reading (code) versus writing is well over 10 to 1 … (therefore) making it easy to read makes it easier to write.” par Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship

Langages fonctionnels

- Un langage fonctionnel est donc un langage de programmation dont la syntaxe et les caractéristiques encouragent la programmation fonctionnelle
- L'origine de la programmation fonctionnelle peut être trouvée dans le **lambda-calcul**
- L'idée de base du lambda-calcul est que *tout est fonction*. Une fonction est en particulier exprimée par une expression qui peut contenir des fonctions qui ne sont pas encore définies et qui sont alors remplacées par des variables.

Langages fonctionnels

- Impurs:
  - langages fonctionnels autorisant la programmation impérative
  - Peuvent modifier un état autre que la valeur de retour (acceptent l'effets de bord)
  - Exemples: Scala, Javascript, Lisp. Ocaml, F#, Python, etc.
- Purs:
  - langages fonctionnels n’autorisant pas la programmation impérative
  - ils sont dénués d'effets de bord
  - Exemples: Haskell, Hope, etc.

F#

- F# est un langage de programmation **fonctionnel**, **impératif** et **orienté objet** pour la plate-forme .NET.
- F# est un langage fortement typé utilisant l'inférence de types (les types sont associés à des expressions, sans qu'ils soient indiqués explicitement dans le code source).
- Il s'agit d'une version dérivée du langage OCaml adaptée pour la plate-forme .NET.

F#

- F # est également sensible à la casse

$$x = 10 \neq X = 10$$

- Le nom d'une valeur peut être une combinaison de lettres, de chiffres, un trait de soulignement _, ou un Apostrophe '.
- Toutefois, le nom doit commencer par une lettre ou un trait de soulignement.

Deux choses importantes

- Les liaisons
- Les fonctions

Les liaisons

- Les liaisons: associe un identificateur à une valeur ou une fonction.
- Vous utilisez le mot clé **let** pour lier un nom à une valeur ou une fonction
- La forme la plus simple de l'expression let lie un nom à une valeur simple

(*ce sont deux identifiants*)

let i = 1 ou let unIdentifiantTresLong =3 * 4 + 5 * 6

Les fonctions

- Les fonctions sont l'unité fondamentale de l'exécution d'un programme dans n'importe quel langage de programmation.
- Comme dans d'autres langages, une fonction F# possède un nom, peut avoir des paramètres et accepter des arguments, et contient un corps.
- Pour définir des fonctions, utilisez le mot clé **let** ou, si la fonction est récursive, la combinaison de mots clés **let rec**.

Installation de F#

- http://fsharp.org/use/windows/

- Assurer vous que Visual F# Power Tools est installé

- Groupes d'utilisateurs ☺ http://community.fsharp.org/

Hello World
fsc + editeur

- Dans un éditeur écrivez :

  printfn "Hello World "

  Sauvegarder l'ensemble dans un fichier dont l'extension est **fs**

  Générer un fichier exécutable à l'aide de FSC.exe:

  fsc votreFichier.fs

  Cette instruction génère un fichier exécutable

  votreFichier.exe

  Exécuter le fichier **votreFichier.exe**

Les liaisons de valeurs

- Liaison par simple nom: *let i = 10*
- Au lieu d'un simple nom, un modèle qui contient des noms peut être spécifié.
  - Exemple de liaison d'un tuple: *let i, j, k = (1, 2, 3)*
- Les noms liés peuvent être utilisés après le point de définition, mais pas avant l'apparition de la liaison let:

```
// Error:
    printfn "%d" x
    let x = 100
// OK:
printfn "%d" x
```

Les liaisons de fonction

- Les liaisons de fonction suivent les règles des liaisons de valeur, à la différence que les liaisons de fonction incluent le nom et les paramètres de fonction

```
let maFonction1 a =
    a + 1
```

- les paramètres peuvent être aussi des modèles

```
let maFonction2 (a, b) = a + b
```

- Une expression de liaison let prend la valeur de la dernière expression

```
let result =
    let maFonction3 (a, b) = a + b
    100 * maFonction3 (1, 2)
```

Hello World
fsc + Visual Studio

- Créer un projet Visual F# puis application console.
- Écrivez le bout de code:

```
open System
[<EntryPoint>]
let main argv =
    printfn "Hello World"
    Console.ReadLine() |> ignore
    0
```

- Exécuter le (comme il était habituel avec C#)

Hello World

- printfn "Hello World"
- Exécuter en mode interactif (alt + entré)

Les types de données

Les types de données en F # peuvent être classés comme suit:

- Les types Intégral
- Les types à virgule flottante
- Les types de texte
- Autres types

Les types Intégral

F# Type	Size	Range	Exemple	Remarques
sbyte	1 byte	-128 to 127	42y -11y	8-bit signed integer
byte	1 byte	0 to 255	42uy 200uy	8-bit unsigned integer
int16	2 bytes	-32768 to 32767	42s -11s	16-bit signed integer

Les types Intégral

F# Type	Size	Range	Exemple	Remarques
uint16	2 bytes	0 to 65,535	42us 200us	16-bit unsigned integer
int/int32	4 bytes	-2,147,483,648 to 2,147,483,647	42 -11	32-bit signed integer
uint32	4 bytes	0 to 4,294,967,295	42u 200u	32-bit unsigned integer

Les types Intégral

F# Type	Size	Range	Exemple	Remarques
int64	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	42L -11L	64-bit signed integer
uint64	8 bytes	0 to 18,446,744,073,709,551,615	42UL 200UL	64-bit unsigned integer
bigint	At least 4 bytes	any integer	42I 1499999 9999999 9999999 9999999 9999I	arbitrary precision integer

Les types à virgule flottante

F# Type	Size	Range	Exemple	Remarques
float32	4 bytes	±1.5e-45 to ±3.4e38	42.0F -11.0F	32-bit signed floating point number (7 significant digits)
float	8 bytes	±5.0e-324 to ±1.7e308	42.0 -11.0	64-bit signed floating point number (15-16 significant digits)
decimal	16 bytes	±1.0e-28 to ±7.9e28	42.0M -11.0M	128-bit signed floating point number (28-29 significant digits)
BigRational	At least 4 bytes	Any rational number.	42N -11N	Arbitrary precision rational number. Using this type requires a reference to FSharp.PowerPack.dll.

Les types de texte

F# Type	Size	Range	Example	Remarks
char	2 bytes	U+0000 to U+ffff	'x' '\t'	Single unicode characters
string	20 + (2 * string's length) bytes	0 to about 2 billion characters	"Hello" "World"	Unicode text

Autres types

F# Type	Size	Range	Exemple	Remarques
bool	1 byte	Only two possible values, true or false	true false	Stores boolean values

Formatage et affichage

- F # prend en charge deux différents styles de formatage de texte:
  - La technique standard .NET Console.WriteLine
  - La technique du langage C en utilisant printf et les fonctions qui lui sont associées telles que ***printfn***.

```
printfn “Ceci est un entier: %d” 5
System.Console.WriteLine(“Ceci est un entier: {0}” , 5)
```

Les spécifications de format de printfn

Type	Description
%b	Met en forme un bool, mis en forme comme true ou false.
%c	Met en forme un caractère.
%s	Met en forme une string, mise en forme comme son contenu, sans interpréter de caractères d'échappement.
%d, %i	Met en forme tout type entier de base mis en forme comme un entier décimal, signé si le type entier de base est signé.
%u	Met en forme tout type entier de base mis en forme comme un entier décimal non signé.
%x	Met en forme tout type entier de base mis en forme comme un entier hexadécimal non signé, utilisant les lettres minuscules a à f.

Les spécifications de format de printfn

%X	Met en forme tout type entier de base mis en forme comme un entier hexadécimal non signé, utilisant les lettres majuscules A à F.
%o	Met en forme tout type entier de base mis en forme comme un entier octal non signé.
%e, %E, %f, %F, %g, %G	Met en forme tout type à virgule flottante de base (float, float32) mis en forme à l'aide de spécifications de format à virgule flottante de style C.
%e, %E	Met en forme une valeur signée ayant le format [-]d.ddde[sign]ddd où d correspond à un chiffre décimal unique, dddd correspond à un ou plusieurs chiffres décimaux, ddd correspond à exactement trois chiffres décimaux, et le signe est + ou -.
%f	Met en forme une valeur signée ayant le format [-]dddd.dddd, où dddd correspond à un ou plusieurs chiffres décimaux. Le nombre de chiffres avant que la virgule décimale dépend de la grandeur du nombre, et le nombre de chiffres après la virgule décimale dépend de la précision demandée.
%g, %G	Met en forme une valeur signée imprimée au format f ou e, selon celui qui est le plus compact pour la valeur et la précision données.
%M	Met en forme une valeur Decimal.

Les spécifications de format de printfn

%O	Met en forme toute valeur, imprimée en effectuant un boxing de l'objet et en utilisant sa méthode ToString.
%A	Met en forme toute valeur, imprimée avec les paramètres de disposition par défaut.
%a	Spécificateur de format général, requiert deux arguments. Le premier argument est une fonction qui accepte deux arguments : d'abord, un paramètre de contexte du type approprié pour la fonction de mise en forme donnée (par exemple, un TextWriter) et ensuite, une valeur à imprimer qui génère ou retourne le texte approprié. Le deuxième argument est la valeur particulière à imprimer.
%t	Spécificateur de format général, requiert un argument : une fonction qui accepte un paramètre de contexte du type approprié pour la fonction de mise en forme donnée (un TextWriter) et qui génère ou retourne le texte approprié. Les types entiers de base sont byte, sbyte, int16, uint16, int32, uint32, int64, uint64, nativeint et unativeint. Les types de virgule flottante de base sont float et float32.

Exemples sur les types et les formatages

```
(* single byte integer *)
let x = 268.97f
let y = 312.58f
let z = x + y

printfn "x: %f" x
printfn "y: %f" y
printfn "z: %f" z
```

```
(* unsigned 8-bit natural number *)


let p = 2uy
let q = 4uy
let r = p + q

printfn "p: %i" p
printfn "q: %i" q
printfn "r: %i" r
```

Exemples sur les types et les formatages

```
(* signed 16-bit integer *)
let a = 12s
let b = 24s
let c = a + b

printfn "a: %i" a
printfn "b: %i" b
printfn "c: %i" c
```

```
(* signed 32-bit integer *)


let d = 212l
let e = 504l
let f = d + e

printfn "d: %i" d
printfn "e: %i" e
printfn "f: %i" f
```

Exemples sur les types et les formatages

```
(* 32-bit signed floating point number *)
(* 7 significant digits *)

let d = 212.098f
let e = 504.768f
let f = d + e

printfn "d: %f" d
printfn "e: %f" e
printfn "f: %f" f

(* 64-bit signed floating point number *)
(* 15-16 significant digits *)
let x = 21290.098
let y = 50446.768
let z = x + y

printfn "x: %g" x
printfn "y: %g" y
printfn "z: %g" z
```

Exemples sur les types et les formatages

```
let choice = 'y'
let name = “Boushaba"
let org = “cegepsth"

printfn "Choice: %c" choice
printfn "Name: %s" name
printfn "Organisation: %s" org
```

Autre exemple

```
open System
let sign num =
   if num > 0 then "positive"
   elif num < 0 then "negative"
   else "zero"

let main() =
   let s = Console.ReadLine()|>int
   Console.WriteLine("sign {0}: {1}",s, (sign s))
   Console.ReadLine()

main()
0
```

Exercices

- Soit la fonction f (x, y) définit comme suit:
  - $F(x, y)=x^2 + 2y$
  - En F# programmer cette fonction de deux façons différentes:
    - 1: en utilisant 3 fonctions séparées
    - 2: en utilisant 3 fonctions imbriquées
- Écrivez une seule fonction qui calcule la somme des sommes de deux paires de nombres. Pensez comme ce qu'il a été demander.
- Trouver la différence entre la somme des carrés de deux nombres et le carré de leurs sommes