le cnam

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL ASSOCIE DE STRASBOURG

Module : Paradigmes de programmation

Séance 4 : Programmation Fonctionnelle en F# : les basiques

Intervenant : Edouard MANGEL

A l'issue de ce cours théorique, il est temps de passer à la pratique. Vous réaliserez ainsi le programme suivant en F# dans Visual Studio.

Vous écrirez **un** programme composé de plusieurs fonctions, éventuellement réparties en plusieurs fichiers qui seront en rapport avec les différents paragraphes suivants. Vous serez évalués sur la qualité du code, l'application des bonnes pratiques évoquées lors des cours théoriques et l'absence d'erreurs dans vos programmes.

Les sources sont à mettre sur votre repo git **public** avant la veille de la prochaine séance à 23h59. Elles devront se trouver dans une branche appelée functional_programming.

L'adresse donnée doit permettre d'exécuter la commande « git clone » sans action supplémentaire (autre qu'éventuellement l'authentification).

I. Travailler avec des nombres et les afficher dans la console.

A. Nombres entiers

En F#, l'attribution d'une valeur à un nom est appelée une liaison. Les liaisons sont immuables, ce qui les rend similaires aux constantes dans d'autres langages. Comme F# est un langage à typage statique, chaque liaison a un type au moment de la compilation.

Les liaisons sont définies à l'aide du mot-clé let. La spécification du type d'une liaison est facultative pour la plupart des liaisons, car l'inférence de type de F# peut généralement déduire le type sur la base de leur valeur.

- Programmez une liaison qui associe une valeur entière de 5 à une constante appelée « five »
- Effectuez une liaison avec une autre constante appelée « seven » associée au chiffre 7.

Les fonctions sont également des liaisons régulières, mais avec un ou plusieurs paramètres. Une fonction renvoie automatiquement sa dernière expression. L'inférence de type fonctionne également pour la plupart des fonctions, en analysant avec quelles valeurs la fonction est appelée et quelle valeur la fonction renvoie.

- Créez une liaison appelée « cube » qui prend en paramètre un entier et qui renvoie le cube de la valeur passée en paramètre.
- Passez lui successivement les constantes five et seven.

B. Nombres à virgule

Dans cet exercice, vous allez travailler avec des comptes d'épargne. Chaque année, le solde de votre compte d'épargne est mis à jour en fonction de son taux d'intérêt. Le taux d'intérêt que votre banque vous accorde dépend de la quantité d'argent sur votre compte (son solde) :

3,213 % pour un solde négatif (entraîne des intérêts négatifs).

0,5% pour un solde positif inférieur à 1000 euros.

1,621% pour un solde positif supérieur ou égal à 1000 euros et inférieur à 5000 euros.

2,475% pour un solde positif supérieur ou égal à 5000 euros.

Chaque année, le gouvernement vous permet de donner un pourcentage de votre argent à une œuvre de charité, sans payer d'impôts. Comme vous êtes une bonne personne, si votre solde est positif à la fin de l'année, vous donnez le double de ce montant à des œuvres de charité, arrondi au dollar inférieur le plus proche.

Vous avez trois tâches à accomplir, chacune d'entre elles traitant votre solde et son taux d'intérêt.

1. Calculer le taux d'intérêt

Implémentez la fonction interestRate pour calculer le taux d'intérêt sur la base du solde spécifié :

```
let interestRate (balance: decimal): single = //implémenter la fonction

interestRate 200.75m

// => 0.5f
```

Notez que la valeur renvoyée est un simple.

2. Calculez les intérêts

Implémentez la méthode interest pour calculer les intérêts sur la base du solde spécifié :

```
let interest (balance: decimal): decimal = // implementer ici
```

interest 200.75m

// => 1.00375m

Notez que la valeur renvoyée est un decimal.

3. Calculer la mise à jour annuelle du solde

Mettez en œuvre la fonction annualBalanceUpdate pour calculer la mise à jour annuelle du solde, en tenant compte du taux d'intérêt :

let annualBalanceUpdate(balance: decimal): decimal = // Implémenter

annualBalanceUpdate 200.75m

// => 201.75375m

Notez que la valeur renvoyée est une valeur décimale.

4. Calculez le montant à donner aux œuvres de charité

Mettez en œuvre la fonction amountToDonate pour calculer le montant à donner aux associations caritatives en fonction du solde et du pourcentage d'exonération fiscale autorisé par le gouvernement :

let amountToDonate(balance: decimal) (taxFreePercentage: float): int =

let solde = 550.5m

let taxFreePercentage = 2,5

amountToDonate balance taxFreePercentage

// => 27

Notez que la valeur renvoyée est un nombre entier.

II. Les chaînes de caractère

A. Découper des parties de chaînes de caractères

Dans cet exercice, vous allez traiter les lignes de journal.

Chaque ligne de journal est une chaîne de caractères formatée comme suit : "[<LEVEL>] : <MESSAGE>".

Il existe trois niveaux de journal différents :

- INFO
- WARNING
- ERREUR

Vous avez trois tâches, chacune d'entre elles prenant une ligne de journal et vous demandant d'en faire quelque chose.

1. **Implémentez la fonction message pour retourner le message d'une ligne de journal :**

message "[ERREUR] : Opération invalide"

// => "Opération non valide".

Tout espace blanc de début ou de fin de ligne doit être supprimé :

message "[WARNING] : Disque presque plein"

// => "Disque presque plein"

2. **Implémentez la fonction logLevel pour renvoyer le niveau de journalisation d'une ligne de journal, qui doit être renvoyé en minuscules :**

logLevel "[ERREUR] : Opération invalide "

// => "erreur

3. **Implémentez la fonction reformat qui reformate la ligne de journal, en mettant le message en premier et le niveau de journal après entre parenthèses :**

reformat "[INFO] : Opération terminée"

// => "Opération terminée (info)"

B. Manipuler les caractères

Implémentez la fonction qui renvoie la réponse de Bob quand on lui parle. Les règles de gestion sont les suivantes :

Bob est un adolescent nonchalant. Dans la conversation, ses réponses sont très limitées.

Bob répond "Bien sûr" si vous lui posez une question, comme "Comment ça va ?".

Il répond "Whoa, calme-toi !" si vous lui criez dessus (en majuscules).

Il répond "Calme-toi, je sais ce que je fais" si tu lui cries une question.

Il répond "Très bien. Sois comme ça" si tu t'adresses à lui sans rien dire.

Il répond "Peu importe" à tout le reste.

Le partenaire de conversation de Bob est un puriste en matière de communication écrite et suit toujours les règles normales de ponctuation des phrases en français.

III. Les structures de données

A. Les records

Instructions

Dans cet exercice, vous êtes un grand amateur de sport et vous venez de vous découvrir une passion pour le basket-ball de la NBA. Comme vous êtes novice en la matière, vous vous plongez dans l'histoire de la NBA, en suivant les équipes, les entraîneurs, leurs statistiques de victoires et de défaites et en les comparant les unes aux autres.

Comme vous n'avez pas encore d'équipe favorite, vous allez également développer un algorithme pour déterminer si vous devez soutenir une équipe particulière.

Vous avez sept tâches à accomplir pour vous aider à développer votre propre algorithme de soutien à une équipe.

1. Tâche 1 Définir le modèle

Définissez le record Coach avec les deux champs suivants :

Name : le nom de l'entraîneur, de type chaîne de caractères.

FormerPlayer : indique si le coach était un ancien joueur, de type bool.

Définissez le record Stats avec les deux champs suivants :

Wins : le nombre de victoires, de type int

Pertes : le nombre de pertes, de type int.

Définissez le record Team avec les trois champs suivants :

Name : le nom de l'équipe, de type string.

Coach : le coach de l'équipe, de type Coach.

Stats : les statistiques de l'équipe, de type Stats.

2. Tâche 2 Créer le coach d'une équipe

Implémentez la fonction createCoach qui prend le nom du coach et son statut d'ancien joueur comme paramètres, et renvoie son enregistrement Coach :

```
createCoach "Larry Bird" true
// => { Name = "Larry Bird" ; FormerPlayer = true }
```

3. Tâche 3 Créer les statistiques d'une équipe

Implémentez la fonction createStats qui prend le nombre de victoires et le nombre de défaites comme paramètres, et renvoie son enregistrement Stats :

```
createStats 58 24
// => { Victoires = 58 ; Pertes = 24 }
```

4. Tâche 4 Créer une équipe

Implémentez la fonction createTeam qui prend le nom de l'équipe, le coach et le record comme paramètres, et renvoie son enregistrement Team :

```
let coach = createCoach "Larry Bird" true
let record = createStats 58 24
createTeam "Indiana Pacers" coach record
// => { Nom = "Indiana Pacers" }
// Coach = { Nom = "Larry Bird" ; FormerPlayer = true }
// Stats = { Victoires = 58 ; Défaites = 24 } }
```

5. Tâche 5 Remplacer l'entraîneur

Les propriétaires de la NBA étant impatients, vous avez constaté que les mauvais résultats de l'équipe conduisaient souvent au remplacement de l'entraîneur. Implémentez la fonction replaceCoach qui prend l'équipe et son nouveau coach comme paramètres, et retourne l'équipe mais avec le nouveau coach :

```
let coach = createCoach "Larry Bird" true
let record = createStats 58 24
let team = createTeam "Indiana Pacers" coach record
```

```
let newCoach = createCoach "Isiah Thomas" true

replaceCoach équipe newCoach

// => { Nom = "Indiana Pacers"

// Coach = { Nom = "Isiah Thomas" ; FormerPlayer = true }

// Stats = { Victoires = 58 ; Défaites = 24 } }
```

6. Tâche 6 Vérifier si l'équipe est la même

En fouillant dans les statistiques, vous tenez des listes d'équipes et de leurs records. Parfois, vous vous trompez et il y a des doublons dans votre liste. Implémentez la fonction isSameTeam qui prend deux équipes et renvoie true s'il s'agit de la même équipe ; sinon, elle renvoie false :

```
let pacersCoach = createCoach "Larry Bird" true

let pacersStats = createStats 58 24

let pacersTeam = createTeam "Indiana Pacers" pacersCoach pacersStats

let lakersCoach = createCoach "Del Harris" false

let lakersStats = createStats 61 21

let lakersTeam = createTeam "LA Lakers" lakersCoach lakersStats


isSameTeam pacersTeam lakersTeam

// => false
```

7. Tâche 7 Vérifiez si vous devez soutenir une équipe

Après avoir examiné de nombreuses équipes et de nombreux matchs, vous avez élaboré un algorithme. Si l'une des affirmations suivantes est vraie, vous soutenez cette équipe :

Le nom de l'entraîneur est "Gregg Popovich".

L'entraîneur est un ancien joueur

L'équipe s'appelle les "Chicago Bulls".

L'équipe a gagné 60 matchs ou plus

L'équipe a plus de défaites que de victoires

Implémentez la fonction rootForTeam qui prend une équipe et renvoie true si vous devez soutenir cette équipe ; sinon, elle renvoie false :

```
let spursCoach = createCoach "Gregg Popovich" false
let spursStats = createStats 56 26
let spursTeam = createTeam "San Antonio Spurs" spursCoach spursStats
rootForTeam spursTeam
// => vrai
```

B. Les enums (Discriminated Unions)

Instructions

Dans cet exercice, c'est la Saint-Valentin et vous et votre partenaire avez prévu de faire quelque chose de sympa ensemble. Votre partenaire a beaucoup d'idées, et vous demande maintenant de les évaluer, afin de trouver l'activité à réaliser.

Les idées suivantes sont proposées par votre partenaire :

- Jouer à un jeu de société
- Se détendre
- Regarder un film
- Aller au restaurant
- Faire une promenade

Vous avez six tâches à accomplir pour choisir votre activité de la Saint-Valentin.

1. Définir l'approbation

Pour chaque idée proposée par votre partenaire, vous répondez par l'une des trois options suivantes : oui, non ou peut-être.

Définissez l'union discriminante Approbation pour représenter ces options comme les trois cas suivants : Oui, Non et Bof

2. Définissez les cuisines

Votre partenaire a sélectionné deux restaurants possibles : un basé sur la cuisine coréenne et l'autre sur la cuisine turque.

Définissez l'union discriminée Cuisine pour représenter ces cuisines comme les deux cas suivants : Coréen et Turc.

3. Définir les genres de films

Il y a des tonnes de films parmi lesquels choisir, alors pour réduire le nombre de films, votre partenaire doit également indiquer leur genre.

Définissez l'union discriminée Genre pour représenter les genres suivants comme des cas : Crime, Horreur, Romance et Thriller.

4. Définissez l'activité

Comme nous l'avons dit, votre partenaire a proposé cinq activités différentes : jouer à un jeu de société, se détendre, regarder un film, aller au restaurant et se promener.

Définissez l'union discriminée Activité pour représenter ces types d'activité :

BoardGame : aucune donnée associée.

Chill : aucune donnée associée.

Film : a son Genre comme donnée associée.

Restaurant : a sa Cuisine comme donnée associée.

Walk : a pour donnée associée un int représentant le nombre de kilomètres à parcourir.

5. Évaluez l'activité

Enfin, vous êtes prêt à évaluer les idées de votre partenaire. Il s'agit de savoir ce que vous pensez de l'idée de votre partenaire :

Jouer à un jeu de société : non.

Se détendre : non.

Regarder un film : oui si c'est un film romantique ; sinon, non.

Aller au restaurant : oui si la cuisine est coréenne, peut-être si elle est turque.

Faire une promenade : oui si la promenade est inférieure à trois kilomètres ; peut-être si elle est inférieure à cinq kilomètres ; sinon, non.

Implémentez une fonction nommée rateActivity qui prend une valeur Activity et renvoie l'approbation en fonction des sentiments ci-dessus :

```
rateActivity (Restaurant Turc)
// => Peut-être
```

IV. Les collections

A. Les tableaux

Vous êtes chargé de compter le nombre de clients qui ont visité un magasin au cours des sept derniers jours.

Vous avez six tâches, qui portent toutes sur le nombre de clients ayant visité votre magasin.

1. Tâche 1 : Vérifiez les chiffres de la semaine dernière.

A des fins de comparaison, vous gardez toujours à proximité une copie des comptages de la semaine dernière, qui étaient les suivants : 0, 2, 5, 3, 7, 8 et 4. Définissez la liaison lastWeek qui contient les comptages de la semaine dernière .

2. Tâche 2 : Vérifiez combien de clients ont visité votre magasin hier

Mettez en œuvre la fonction visitesHier pour obtenir le nombre de clients qui ont visité votre magasin hier. Le nombre de clients est ordonné par jour, le premier élément étant le nombre du jour le plus ancien et le dernier élément étant le nombre d'aujourd'hui.

visitesHier [| 3 ; 5 ; 0 ; 7 ; 4 ; 1 |]

// = 4

3. Tâche 3 Calculer le nombre total de visiteurs

Mettez en œuvre la fonction total pour obtenir le nombre total de clients qui ont visité votre magasin :

total [| 3 ; 5 ; 0 ; 7 ; 4 ; 1 |]

// => 20

4. Tâche 4 : Vérifier s'il y a eu un jour sans clientx visiteurs

Implémentez la fonction joursSansVisite qui renvoie vrai si un jour aucun client n'a visité le magasin ; sinon, renvoyez faux :

joursSansVisite [| 3 ; 5 ; 0 ; 7 ; 4 ; 1 |]

// => vrai

5. Tâche 5 Incrémenter le compte du jour

Implémentez la fonction incrementTodaysCount pour incrémenter le compte du jour et renvoyer les comptes mis à jour :

let birdCount = [| 3 ; 5 ; 0 ; 7 ; 4 ; 1 |]

incrementTodaysCount compte-client

// => [| 3 ; 5 ; 0 ; 7 ; 4 ; 2 |]

6. Tâche 6 Vérification de la semaine impaire

Au cours de l'année dernière, vous avez constaté que certaines semaines présentent les mêmes schémas bizarres :

Chaque jour pair de la semaine, il n'y avait aucun client.

Chaque jour pair de la semaine, exactement 10 clientx ont été repérés.

Chaque jour impair de la semaine, exactement 5 clientx ont été repérés.

Implémentez la fonction oddWeek qui renvoie vrai si le modèle de comptage des clientx de cette semaine correspond à l'un des modèles impairs :

```
oddWeek [| 1 ; 0 ; 5 ; 0 ; 12 ; 0 ; 2 |]
// => vrai

oddWeek [| 5 ; 0 ; 5 ; 12 ; 5 ; 3 ; 5|]
// => vrai
```

B. Les listes

Instructions

Dans cet exercice, vous allez écrire du code pour garder la trace d'une liste de langages de programmation que vous souhaitez apprendre sur Exercism.

Vous avez six tâches, qui impliquent toutes de traiter des listes.

1. Tâche 1 Créer une nouvelle liste

Pour garder la trace des langages que vous voulez apprendre, vous devez créer une nouvelle liste. Définissez le binding newList qui contient une nouvelle liste vide.

newList

// => []

2. Tâche 2 Définir une liste existante

Actuellement, vous avez une feuille de papier sur laquelle figurent les langages que vous souhaitez apprendre : F#, Clojure et Haskell. Définissez la liaison existingList pour représenter cette liste.

existingList

// => ["F#" ; "Clojure" ; "Haskell"]

3. Tâche 3 Ajouter un nouveau langage à une liste

Au fur et à mesure que vous explorez Exercism et que vous trouvez des langues intéressantes, vous souhaitez les ajouter à votre liste. Implémentez la fonction addLanguage pour ajouter un nouveau langage au début de votre liste.

addLanguage "TypeScript" ["JavaScript" ; "CoffeeScript"]

// => ["TypeScript" ; "JavaScript" ; "CoffeeScript"]

4. Tâche 4 Comptez les langues dans la liste

Compter les langues une par une n'est pas pratique. Implémentez la fonction countLanguages pour compter le nombre de langues de votre liste.

countLanguages ["C#" ; "Racket" ; "Rust" ; "Ruby"]

// => 4

5. Tâche 5 Inversez la liste

À un moment donné, vous vous rendez compte que votre liste est en fait ordonnée à l'envers ! Implémentez la fonction reverseList pour inverser votre liste.

reverseList ["Prolog" ; "C" ; "Idris" ; "Assembly"]

// => ["Assembly" ; "Idris" ; "C" ; "Prolog"]

6. Tâche 6 Vérifier si la liste est passionnante

Bien que vous aimiez tous les langages, F# a une place spéciale dans votre cœur. En tant que tel, vous êtes vraiment excité par une liste de langages si :

Le premier de la liste est F#.

Le deuxième élément de la liste est F# et la liste contient deux ou trois langues.

Implémentez la fonction excitingList pour vérifier si une liste de langues est excitante

excitingList ["Nim" ; "F#"]

// => vrai