Programmation applicative – L2
TP n° 0 : Prise de contact avec l'interpréteur Scheme

C. Dony {Christophe.Dony@umontpellier.fr}
S. Daudé {Sylvain.Daude@umontpellier.fr}
D. Catta {Davide.Catta@umontpellier.fr}

1 Préambule

Cet enoncé concerne la première séance de TP, consacrée à la prise en main de l'interpréteur Scheme que vous utiliserez tout au long du semestre. Le but n'étant pas de vous noter mais de vous accompagner dans la découverte du langage Scheme et de l'interpréteur associé, ce TP ne sera pas évalué. Vous n'avez donc rien à rendre à la fin de la séance.

Pour les séances suivantes, vos chargés de TP pourront à tout moment relever votre travail afin de l'évaluer. **En conséquence, il faudra sauvegarder précieusement votre travail au fur et à mesure des séances.**

2 L'environnement de développement

L'interpréteur Scheme que nous allons utiliser dans les TPs est DrRacket, accessible via le terminal avec la commande `drracket`. Pour ceux qui souhaiteraient l'installer sur leur machine personnelle, il est en libre téléchargement sur le site *https ://racket-lang.org/download/*.

2.1 Introduction

DrRacket est un environnement interactif de programmation en langage Scheme. La fenêtre qui apparaît est divisée en deux zones principales (elle comporte aussi un certain nombre de menus et de boutons en haut sous la ligne de menu) :

- la zone blanche supérieure est la **zone de définitions** : c'est là que vous allez entrer les nouvelles fonctions que vous voulez définir (c'est-à-dire des expressions Scheme (parenthésées et préfixées) commençant par le mot clef *define*).
- la zone blanche inférieure est la **zone d'interactions** : c'est là que vous allez demander à Scheme d'évaluer des expressions. La présence de l'invite (un signe '>') vous indique que Scheme est déjà prêt à recevoir vos ordres.

Le curseur clignotant ('|') indique où vont être retransmis les prochains caractères que vous taperez (dans une zone ou dans l'autre). Pour écrire des commentaires dans votre code, il suffit de faire précéder la ligne de commentaire d'un point-virgule " ;".

2.2 Premiers pas

2.2.1 Interactions

Dans la zone d'interactions, positionnez-vous après le signe '>' en cliquant à cet endroit avec le bouton gauche de la souris. Entrez au clavier :

```
(+ 3 4)
```

en séparant les arguments par des espaces puis tapez sur la touche "entrée". Après la phase **lecture**, Scheme passe dans la phase d'**évaluation** (calcule rapidement le résultat) puis dans la phase **résultat** (où le résultat de l'évaluation, 7, s'affiche à l'écran). Puis, suivant le principe de la *boucle interactive*, il se repositionne dans la phase lecture (affichage de l'invite, attente de votre prochain ordre à lire au clavier).

Petite astuce (qui sauve la vie) : si vous voulez éditer une commande que vous avez tapée précédemment, il est possible d'accéder à l'historique des commandes avec (CTRL + flêche du haut) ...

Exercice 1 *Tapez maintenant les expressions suivantes en essayant de prévoir leur résultat :*

```
(< 10 100)
(and (< 55 100) (> 55 0))
(* 1 2 3)
(if (10 < 20) #t #f)
```

Exercice 2 *Corrigez la dernière expression pour que son évaluation ne produise pas d'erreur.*

Simplifiez maintenant cette expression (trouvez une expression équivalente mais plus courte).

Exercice 3 *Mettez les expressions suivantes sous forme parenthésée et préfixée, puis vérifiez que vous avez bon en les tapant dans la zone d'interactions, puis notez leur résultat :*

expressions initiales	*expressions Scheme*	*résultats de l'évaluation*
`2 - 6 + 10`		
`2 + 5 * ( -3 + 12)`		
`(9/ 2) / (2 / 4)`		

Exercice 4 *Évaluez les expressions logiques ci-dessous, pouvez-vous prévoir leur résultat ?*

expressions logiques	*résultats*
`(not (and (< 55 100) (> 55 0)))`	
`(or (not (< 55 100)) (not (> 55 0)))`	

Exercice 5 *Pouvez-vous expliquer pourquoi elles donnent toutes les deux le même résultat (aussi dans le cas où tous les 55 sont remplacés par une autre valeur numérique dans les deux expressions) ?*

Indice : considérez les deux sous-expressions `(< 55 100)` et `(> 55 0)` comme étant deux expressions logiques A et B et établissez la table de vérité des expressions du tableau ci-dessus en fonction des valeurs de A et de B (soit #t, soit #f).

2.2.2 Définition de nouvelles fonctions

2.2.3 Entrée des fonctions

Exercice 6 *Positionnez maintenant le curseur dans la zone de définitions (zone supérieure). Tapez la suite de lignes suivante en respectant les espaces, les passages à la ligne et l'indentation (espaces insérés en début de certaines lignes) :*

```
(define (puis2 x)
   (* x x))
```

2.2.4 Vérification de la syntaxe des fonctions

Pour vérifier la syntaxe dans la zone de définitions, cliquez ensuite sur le bouton "*Vérifier*" ("*Check Syntax*" en anglais).

Amenez maintenant le pointeur de la souris (sans cliquer) au dessus du caractère '`x`' de la première ligne : l'environnement vous indique par des flèches dans quelle(s) sous-expression(s) de la définition courante est utilisé cet argument. Si vous positionnez la souris au dessus d'un des '`x`' d'une sous-expression de la fonction (ici de "`(* x x)`"), vous remarquez qu'une flèche vous indique où cet argument que vous utilisez a été défini.

2.2.5 Évaluation et appel des fonctions

Cliquez maintenant sur le bouton "*Exécuter*" : la zone d'interactions est vidée de son ancien contenu, et les expressions de la fenêtre supérieure sont évaluées.

Si aucune erreur ne survient lors de l'évaluation (c'est-a-dire si votre fonction est syntaxiquement correcte), le curseur est positionné après l'invite et vous engage à taper un ordre.

Exercice 7 *Vérifiez que la fonction* `puis2` *est maintenant une fonction que connaît Scheme en tapant les expressions suivantes dans la zone d'interactions :*

```
(puis2 3)
(puis2 -4)
(puis2 1)
(puis2 (puis2 2))
(puis2 (puis2 1))
```

Dans la suite, nous vous recommandons de suivre toujours cette démarche en trois temps : entrée des fonctions, vérification de leur syntaxe (jusqu'à obtenir une définition sans erreurs), essais d'appels de ces fonctions pour vérifier qu'elles fonctionnent correctement.

Exercice 8 *Repositionnez-vous maintenant dans la zone de définitions (après la définition de puis2) et entrez texto (sans modification) les lignes suivantes :*

```
(define (puis4 x)
   (puis2 (puis2))
)
```

Vérifiez la syntaxe, et positionnez la souris sur un "`puis2`" de la ligne centrale de cette définition. Automatiquement l'environnement vous indique où est définie cette fonction que vous appelez.

Exercice 9 *Après avoir cliqué sur le bouton* Exécuter*, passez maintenant dans la zone d'interactions et essayez d'appeler la nouvelle fonction, par exemple en tapant :*

```
(puis4 10)
```

Quel est le résultat ? Corrigez la partie définition suivant ce que vous indique Scheme, puis vérifiez que vous avez bon (dans l'ordre, syntaxe, exécute, essai d'appel de la fonction). Recommencez jusqu'à obtenir des résultats corrects.

2.2.6 Sauvegarde du travail réalisé

Cliquez sur l'icône "**Sauvegarder**" (ou "**Save**"), dans la fenêtre apparaissant, dans la zone où le curseur est positionné, indiquez un nom de fichier (par exemple *tp0.scm*) dans lequel l'environnement va sauver les fonctions que vous avez actuellement définies (contenu de la zone *définition de fonctions*). Ceci permettra de ne pas avoir à retaper ces fonctions, si vous en avez encore besoin une prochaine fois.

3 Obtenir de l'aide

Cliquez sur le menu d'aide (*Help*). Recherchez la description complète du langage Scheme telle qu'elle est décrite dans le document officiel "Revised 5 Report".

Exercice 10 *Recherchez dans l'aide la réponse aux questions suivantes :*

- *Comment calculer une expression comme* 12345^{12345}*, que l'on calculera (chercher comment calculer les exponentielles).*
- *Quelle est la fonction Scheme qui permet de calculer la partie entière d'un nombre ?*
- *Quelles sont les deux fonctions Scheme qui permettent de calculer la division entière d'un entier par un autre, et le reste de cette division ? En déduire une fonction* `divisible?` *qui teste si son premier argument est divisible par le second.*
- *Comment écrire une conditionnelle (*`cond`*) ? Écrivez une expression dans la* zone de définitions *qui dépend de* x *et qui rend 3 si* $x = 5$*,* x^2 *si* $x \geq 8$*, "toto" si* $x = 6$*,* `#t` *si* $x = 7$ *et* `#f` *dans tous les autres cas.*

Programmation applicative – L2
Première série de TP : Premiers programmes en Scheme

C. Dony {Christophe.Dony@umontpellier.fr}
S. Daudé {Sylvain.Daude@umontpellier.fr}
D. Catta {Davide.Catta@umontpellier.fr}

1 Préambule

Cet enoncé concerne 2 séances de TP. **N'oubliez pas sauvegarder précieusement votre travail au fur et à mesure des séances.** Choisissez le langage "Assez gros scheme".

2 Variables et contextes d'évaluation

Exercice 1 *Dans la zone* interactions*, évaluez (dans l'ordre) les expressions suivantes en essayant de prévoir leur résultat :*

```
(+ taille 7)
(define taille 4)
taille
(puis2 taille)
taille
(define taillebis taille)
taillebis
(define taille 8)
taille
taillebis
(+ taille taillebis)
```

Voyons maintenant ce qui se passe dans le cas où plusieurs contextes d'évaluation sont inclus les uns dans les autres.

Exercice 2 *Pouvez-vous prévoir la valeur renvoyée suite à l'évaluation, dans la zone de définitions, de la dernière expression de cette séquence :*

```
(define x 5)
(define (f x) (* x x))
(f 10)
```

Vérifiez.

Exercice 3 *Même question pour la suite d'expressions :*

```
(define d 1)
(define (plusd x) (+ x d))
(plusd 10)
```

Exercice 4 *Même question si à la suite on ajoute les expressions :*

```
(define d 5)
(plusd 10)
```

Exercice 5 *Passons maintenant à la composition de fonctions. Quel est le résultat de l'évaluation si la séquence suivante est tapée :*

```
(define (h x) (* x x))
(define (g x) (+ 1 (h x)))
(g 10)
```

Se passe-t-il quelque chose si on inverse l'ordre de déclaration des fonctions `h` *et* `g` *?*

3 Les nombres

Scheme distingue les nombres entiers et les nombres flottants.
— `1` est un nombre entier.
— `1.0` est un nombre flottant.
En Scheme, les opérations sur des entiers renvoient des valeurs exactes alors que les opérations sur les flottants renvoient des valeurs approchées.

Exercice 6 *Evaluer les expressions suivantes :*

```
(/ 1 3)
(/ 1.0 3)
(/ 2 6)
(exact->inexact (/ 1 3))
(inexact->exact 0.5)
(/ 3 2)
```

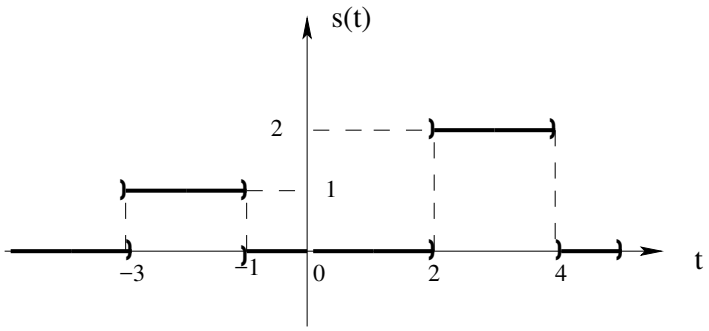
Remarque — On peut utiliser les nombres rationnels en scheme : $\frac{1}{2}$ se note `1/2`. Notez la différence entre `(/ 1 2)` qui est l'application de l'opérateur de division et `1/2` qui est un nombre rationnel.

4 Les conditions

Exercice 7 *Définir une fonction* `monabs` *qui renvoie la valeur absolue d'un nombre.*

Exercice 8 *Définir la fonction* `care-div` *qui gére le problème de la division par* 0 *(utiliser* `display` *pour renvoyer un message d'erreur).*

Exercice 9 *La figure suivante illustre un signal s fonction d'une variable t. Définir à l'aide de* `cond` *une fonction qui calcule la valeur du signal s en un point. Faire varier t de $-\infty$ à $+\infty$.*



5 Calcul de taux d'intérêt

Exercice 10 *Définir une fonction* `placement` *qui prend en argument un capital, un taux, et un nombre d'années. Le résultat de cette fonction doit être le montant que recevra quelqu'un qui a placé ce capital à ce taux pendant cette durée.*

Rappel : Si un placement est au taux d'intérêt t, alors un euro placé pendant n années vaut à la fin $(1+t)^n$ euros.

6 La tortue

La tortue est un petit animal qui permet de faire des graphiques en la déplaçant au moyen de commandes *simples* (ce n'est qu'une tortue).

Avant de pouvoir utiliser la tortue, sélectionnez « Assez gros Scheme » dans le menu `langage` pour disposer du langage graphique. Entrez ensuite la ligne suivante dans la fenêtre des définitions (en haut) :

```
(require (lib "turtles.ss" "graphics"))
```

et cliquez ensuite sur le bouton `Execute` pour que votre choix soit pris en compte.

Entrez dans la fenêtre d'évaluation la commande :

`(turtles #t)`

Une fenêtre graphique apparaît avec une petite flèche indiquant la position actuelle de la tortue. Celle-ci accepte les ordres suivants :

- `(turn x)` où x est un angle en degrés : tourne de x degrés dans le sens inverse des aiguilles d'une montre.
- `(turn/radians x)` idem mais cette fois l'angle x est indiqué en radians.
- `(move x)` avance de x unités.
- `(draw x)` avance de x unités en traçant un trait
- `(clear)` efface l'aire de jeu de la tortue

Pour fermer la fenêtre de la tortue, il suffit d'évaluer `(turtles #f)`.

Exercice 11 *Essayez de manipuler la tortue en lui demandant de faire deux tours de spirale, en changeant de direction par morceaux de 30 degrés et en avançant d'une unité de plus à chaque changement de direction.*

Exercice 12 *Dans la fenêtre de définitions, écrivez une fonction* `carré` *qui prend un argument* `lgr` *et fait dessiner à la tortue un carré de côté* `lgr`.

Exercice 13 *Écrivez maintenant une fonction* `hexagone`*, qui lui fait dessiner un hexagone dont la longueur des côtés est indiqué en paramètre.*

Exercice 14 *Généralisez à n côtés : n devient l'argument d'une fonction* `figure`*, que vous devez écrire.*

6.1 Calcul du jour de l'année

Pour savoir si le jour de sa naissance était par exemple un lundi ou un dimanche, on va écrire un programme SCHEME général qui répondra à ce type de questions. Pour cela, il est considéré une date de référence à partir de laquelle les fonctions suivantes feront références : le lundi 01 01 1900. On pourra aussi exécuter la commande Unix tt cal, par exemple : `cal 1900`.

Exercice 15 *Définir une fonction* `bissextile?` *qui prend en argument une année et qui répond #t si elle est bissextile. Pour écrire cette fonction, se rappeler qu'une année est bissextile si elle est divisible par 4 mais pas par 100, sauf si elle est multiple de 400.*

Exercice 16 *Définir une fonction* `nb-annees-bissextiles` *qui prend en argument une année et qui calcule le nombre d'années bissextiles entre 1900 et cette année (incluse). On vérifiera qu'il y en avait 24 en 1999 et 25 en 2001.*

Exercice 17 *Définir une fonction* `nb-jours-au-1-jan` *qui prend en argument un mois (un nombre de 1 à 12), et qui calcule le nombre de jours écoulés entre le premier janvier et le premier jour de ce mois. On supposera que l'année considérée n'est pas bissextile. Par exemple, le nombre de jours pour le mois de janvier (01) sera 0, pour le mois de février (02) sera 31, pour le mois de mars (03) sera 31+28, etc.*

Exercice 18 *Définir une fonction* `nb-jours` *qui prend en argument une date sous la forme de 3 entiers (par exemple, 21 05 2006 pour le 21 mai de l'année 2006). Cette fonction doit calculer le nombre de jours écoulés depuis le 01 01 1900 jusqu'à cette date. Par exemple, le nombre de jours pour le 5 01 1900 sera 4 jours. Vous devrez vous poser la question : au sein de l'année donnée en argument, a-t-on dépassé ou non la fin février ?*

Exercice 19 *Ecrire une fonction* `jour-de-semaine` *qui rend en valeur la bonne chaîne de caractères parmi "Lundi", "Mardi", ... "Dimanche" correspondant à la date passée en argument. Par exemple, l'évaluation de* `(jour-de-semaine 30 09 2008)` *donnera "Mardi".*

7 Deux petits exercices supplémentaires pour les gourmands...

Exercice 20 *Calcul de volumes*
On veut calculer le volume d'une sphère de rayon r. Prenons d'abord une valeur de rayon, par exemple 10, en saisissant l'expression : `(define r 10)`*.*

Puis, donnez l'expression qui a pour valeur le volume de la sphère de rayon r. Modifiez la valeur de r, et réévaluez l'expression. De même, definissez une hauteur h et un rayon r et donner une expression qui a pour valeur le volume du cylindre de rayon r et de hauteur h.

Exercice 21 *Relativité*
La fonction `(einstein u v)` *: loi d'addition des vitesses en relativité restreinte, avec la vitesse de la lumière* $c \approx 300000\ km.s^{-1}$ *:*

$$(u, v) \to \frac{u+v}{1+\frac{uv}{c^2}}$$

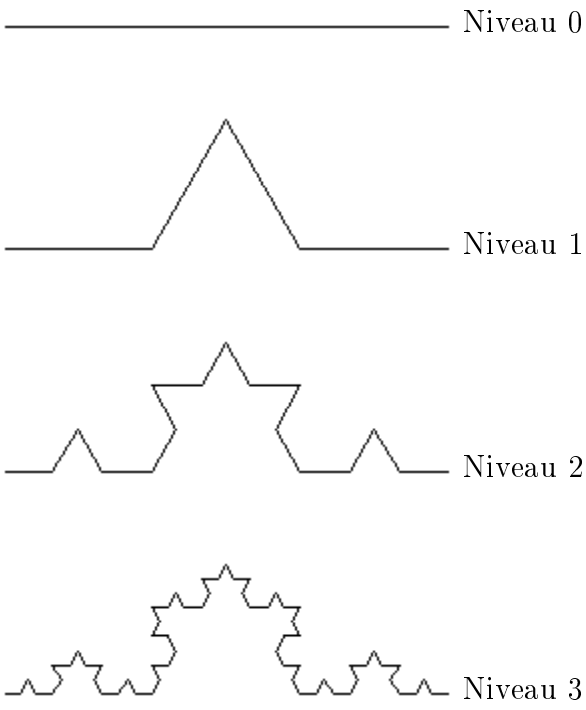
Application numérique Rien ne vaut un problème concret : un voyageur court (assez vite ;-)) à $250000\ km.s^{-1}$ *dans le couloir d'un train, dans le sens de la marche. Et le train roule à* $270000\ km.s^{-1}$*. Quelle est la vitesse du voyageur par-rapport à la voie ?*

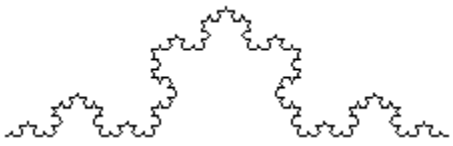
Programmation applicative – L2
Première série d'exercices supplémentaires pour ceux qui ont tout fini...

C. Dony {Christophe.Dony@umontpellier.fr}
S. Daudé {Sylvain.Daude@umontpellier.fr}
D. Catta {Davide.Catta@umontpellier.fr}

1 Le flocon de Von Koch

Soit la courbe de Von Koch définie de la façon suivante : On considère un triangle équilatéral. C'est le niveau 0. Ensuite, on prend chaque côté et on le divise en trois. Le tiers du milieu est remplacé par deux segments de même longueur. C'est le niveau 1. On réitère le processus sur chaque segment...Voici quelques niveaux illustrant la transformation associée à chaque segment :




Niveau 4

Exercice 1 *Soit a_0 la longueur du trait au niveau 0; que vaut a_1 la longueur du trait global au niveau 1 en fonction de a_0 ?*

De façon générale, que vaut a_i la longueur du trait au niveau i en fonction de a_{i-1} la longueur du trait au niveau $(i-1)$? puis en fonction de a_0 ? Que se passe-t-il pour a_i quand i tend vers l'infini ?

Exercice 2 *Ecrire la fonction* `(von_koch prof a)` *où l'on passera en argument la longueur a du trait de niveau 0 et le niveau de profondeur ou l'on veut s'arrêter. Comme dans toute fonction récursive, il faut réfléchir :*

- *à la condition d'arrêt. et ce que l'on fait alors*
- *à la définition du problème au niveau i en fonction du même problème au niveau (i-1)*

Exercice 3 *Écrire une fonction* `(prepa-von-koch)` *qui nettoie l'écran et positionne la tortue de telle façon à exécuter la courbe de Von Koch sur toute la surface de la fenêtre de dessin.*

Exercice 4 *En utilisant la primitive DrScheme* `(sleep N)` *qui met le système en attente pendant N secondes (par exemple* `(sleep 3)` *met le système en attente pendant 3 s), écrire une fonction* `(anim-von-koch n)` *qui fasse apparaître successivement la courbe de von koch au niveau 0, 1, 2, jusqu'à n, en laissant 3 s entre les apparitions de chaque niveau.*

Exercice 5 *En faire de même, avec la courbe de Von Koch modifiée :*

Niveau 1

Exercice 6 *Modifier les fonctions précédentes pour pouvoir commencer le flocon à partir d'un polygône régulier à un nombre quelconque n de côtés, passé en paramètre.*

2 Généralisation

Les mathématiciens ont commencé à démontrer des théorèmes de géométrie en dimension 2, puis en dimension 3, puis... en dimension N. La **généralisation** étant une activité importante de la démarche scientifique, pourquoi ne pas s'en servir en programmation ? Souvent, il vous arrivera d'écrire un algorithme adapté à un problème particulier et de vous apercevoir qu'à peu de frais, il vous est possible d'en déduire un algorithme plus général qui pourra être ré-utilisé par la suite dans d'autre situations.

Autre intérêt de la généralisation : résoudre des problèmes particuliers. C'est peut être surprenant, mais il est parfois plus facile [en programmation comme en maths] de résoudre des problèmes généraux que des problèmes particuliers !

Essayons de généraliser la fonction factorielle. Pourquoi se borner à ne faire que des multiplications ? Remplaçons celle-ci par une opération binaire associative et commutative `combiner` d'élément neutre `null-value` tel que `(combiner x null-value)` = `(combiner null-value x)` = valeur de `x`. La fonction d'accumulation est :

```
(define (accumulate combiner null-value a b)
  (if (> a b)
      null-value
      (combiner a (accumulate combiner null-value (+ a 1) b))))
```

```
> (accumulate * 1 10 20)      ; calcule le produit: 10 × 11 × ··· × 20
⇒ 6704425728000
```

Exercice 7

1. *Écrivez une définition de la fonction factorielle* `fact-accu` *qui utilise cette fonction d'accumulation.*
2. *Écrivez une définition de la fonction* `somme-accu` *qui à entier n associe la somme des entiers de 1 à n, en utilisant cette fonction d'accumulation.*
3. *Écrivez une définition de la fonction* `liste-accu` *qui à entier n associe la liste des entiers de 1 à n, en utilisant cette fonction d'accumulation.*

Exercice 8 *Cette généralisation est insuffisante pour calculer une somme de carrés par exemple. On veut pouvoir indiquer le terme de la fonction à "accumuler".*

1. *Introduisez une fonction* `term` *en paramètre qui s'appliquera à chaque élément de l'intervalle $[a, b]$, et écrivez la fonction plus générale* `(accumulate-term combiner null-value term a b)` *telle que :*

   ```
   > (accumulate-term + 0 square 10 20) ; calcule la somme: 10² + 11² + ··· + 20²
   ⇒ 2585
   ```

2. *Redéfinissez les fonctions* `fact-accu, somme-accu, liste-accu` *avec cette nouvelle version de la fonction d'accumulation.*
3. *Calculez une valeur approchée de la constante $e = 2.718\cdots$ en exprimant le développement limité $1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{15!} \cdot + \frac{1}{n!} \cdots$. Comparez avec la valeur de* `(exp 1)` *où* `exp` *est prédéfini en Scheme. Pour quelle valeur de n approche-t-on e avec une précision inférieure à 10^{-11} ?*
4. *Calculez le produit de tous les nombres impairs de $[1, 50]$.*
5. *Calculez le produit de tous les nombres pairs de $[1, 50]$.*
6. *Calculez le produit de tous les nombres égaux à 1 modulo 3 de $[1, 50]$.*

Exercice 9 *La question précédente soulève la question : pourquoi se limiter à aller de 1 en 1 dans les appels de la fonction* `accumulate` *? Nous pourrions parcourir l'intervalle* $[a, b]$ *2 en 2, ou de 5 en 5, etc. Et d'ailleurs, pourquoi avec un pas constant ? Utilisons plutôt une fonction* `next` *pour calculer le suivant d'un nombre dans l'intervalle.*

1. *Écrivez* `(accumulate-term-next combiner null-value term a next b)` *plus générale encore où* `next` *est une fonction qui à partir de a calcule le suivant de a dans l'intervalle* $[a, b]$.
2. *Redéfinissez les fonctions* `fact-accu, somme-accu, liste-accu` *avec cette nouvelle version de la fonction d'accumulation.*
3. *Calculez une valeur approchée de la somme* $S = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \cdots$. *Pour une grande valeur devinez-vous vers quoi tend la limite ?*

Exercice 10 *Enfin, toujours à l'aide de la fonction d'accumulation, écrivez une fonction* `pi-product` *qui calcule le produit :* $\frac{2\times2\times4\times4\times6\times6\times8\cdots\times2n\times2n}{1\times3\times3\times5\times5\times7\times7\cdots\times(2n-1)\times(2n+1)}$ *qui converge vers* $\frac{\pi}{2}$. *Précisez pour quelle valeur de n la convergence s'effectue avec une précision inférieure à* 10^{-3}. *Comparer avec l'approximation de e faite précédemment.*

Programmation applicative – L2
Deuxième série de TP : Le compte est bon

C. Dony {Christophe.Dony@umontpellier.fr}
S. Daudé {Sylvain.Daude@umontpellier.fr}
D. Catta {Davide.Catta@umontpellier.fr}

1 Préambule

Cet enoncé concerne 3 séances de TP. L'évaluation des TPs se fera en deux phases :

— Pendant les TPs, votre présence et votre progression seront notées. À la fin de chaque séance de TP, vous indiquerez sur la feuille de présence le numéro de la dernière question à laquelle vous avez entièrement répondu.

— Pendant la dernière séance, une évaluation de votre travail tout au long des séances sera effectuée, sous la forme d'un petit problème auquel il faudra répondre en vous servant des fonctions que vous aurez écrites durant les TPs.

En conséquence, il faudra sauvegarder précieusement votre travail au fur et à mesure des séances, pour pouvoir l'utiliser lors de l'examen final.

2 Description du jeu

Le problème est le suivant : nous disposons d'un ensemble d'entiers naturels (les valeurs de départ) et d'un entier cible n, et nous voulons combiner les valeurs de départ au moyen des quatre opérations $+, -, /, \times$ de manière à obtenir l'entier p le plus proche possible de n, sachant que nous pouvons utiliser au plus une fois chacune des valeurs de départ. Nous sommes intéressés non seulement par la valeur de p mais aussi par la formule qui nous a permis de l'obtenir.

Si vous n'avez jamais vu la version télévisée du jeu, voici le lien vers une vidéo montrant une partie : `http://www.youtube.com/watch?v=h9vH7YGotQc`

3 Structures de données

3.1 Données de départ

Nous travaillerons à partir des données suivantes : une liste d'opérateurs, et une liste de valeurs de départ. Par exemple, on peut définir :

```
; Définitions des objets de base
(define LVal '(1 2 3 4 5 6 7 8 9 10 25 50 75 100))
(define Op '(+ * - /))
```

(define Op (list + * - /))

3.2 Définition d'un tirage

Un tirage se compose d'une part d'un nombre pris aléatoirement entre 100 et 999, et d'une série de 6 plaques prises au hasard parmi les valeurs de départ. Ainsi, plusieurs plaques parmi les 6 peuvent prendre les mêmes valeurs.

Exemple de tirage :

Exercice 1 *Écrire deux fonctions* `make-cible` *et* `make-tirage` *qui réalisent respectivement le tirage du nombre à atteindre (entre 100 et 999), et le tirage aléatoire de 6 valeurs parmi les plaques disponibles. On utilisera la fonction de base du langage* `random`*. À noter que* `(random n)` *renvoie une valeur entière aléatoire entre 0 et* $n-1$*.*

Exemple :

```
> (make-cible)
833
> (make-tirage)
(4 10 25 10 5 25)
```

Exercice 2 *Écrire une fonction* `estDans?` *qui prend en paramètres un entier et une liste d'entiers, et renvoie vrai si l'entier est dans la liste, faux sinon. Cette fonction nous servira à tester si le nombre cible est dans un tirage ou une liste de nombres issus d'un tirage.*

Exemple :

```
> (estDans? 4 '(1 2 3 4 5 6))
#t
> (estDans? 2 '(10 20 30))
#f
```

3.3 Opérations autorisées

Afin de déterminer si le compte est bon, l'idée de base, sans optimisation, est d'essayer toutes les possibilités de combinaisons deux à deux des plaques du tirage. Mais certaines opérations ne sont pas permises, car il faut que les valeurs restent des valeurs entières et strictement positives à chaque étape.

Exercice 3 *Écrire une fonction* `estValide?` *qui prend en paramètres un opérateur et deux valeurs entières, et renvoie vrai si l'opération renvoie un résultat entier strictement positif. On aura besoin d'évaluer l'opérateur afin de réaliser le calcul, cela se fait avec la fonction* `eval`.

Exemple :

```
> (eval +)
#<procedure:+>
> ((eval +) 4 5)
9
> (estValide? - 2 14)
#f
> (estValide? / 3 0)
#f
> (estValide? / 1 3)
#f
> (estValide? + 2 6)
#t
```

4 Moteur de jeu, première version

Dans cette première version, on va répondre au problème simplifié suivant : "est-il possible (oui/non) de trouver une combinaison des plaques de façon à obtenir exactement le nombre cible ?".

Exercice 4 *Écrire une fonction* `opere` *qui prend en paramètre une liste d'opérateurs et deux entiers, et renvoie une liste de valeurs obtenues en appliquant les opérations valides sur ces opérateurs. Le résultat doit être symétrique, c'est à dire que pour les opérateurs non symétriques tels que* `-` *et* `/`*, si l'opération dans un sens n'est pas valide, on doit également tester si l'opération dans l'autre sens est valide. Ce n'est pas la peine pour les opérateurs* `+` *et* `*` *qui sont commutatifs.*

Exemple :

```
> Op
(+ * - /)
> (opere Op 2 10)
(12 20 8 5)
```

Exercice 5 *Écrire une fonction* `genere_plaques` *qui prend en paramètre une liste de valeurs entières, et produit une liste de listes repésentant les plaques que l'on peut générer en choisissant deux valeurs parmi la liste initiale et en appliquant les opérations valides. Lorsque l'on choisit deux éléments, les autres plaques ne sont pas modifiées.*

Exemple :

```
> Op
(+ * - /)
> (genere_plaque Op '(10 3 7))
((13 7) (30 7) (7 7) (3 17) (3 70) (3 3) (10 10) (10 21) (10 4))
```

Remarque : Cette dernière fonction peut nécessiter la définition de plusieurs fonctions auxilliaires intermédiaires, c'est le nœud du problème, on prendra donc le temps de réflêchir sur papier !

Exercice 6 *Écrire une fonction* `ceb` *qui prend en argument une liste d'opérateurs, une liste de plaques et un entier cible, et qui affiche "le compte est bon" si on peut obtenir la cible exactement à partir des plaques et des opérateurs, et qui affiche "le compte n'est pas bon" sinon.*

Exemple :

```
> Op
(+ * - /)
> (define jeu (make-tirage))
> jeu
(4 50 9 2 6 3)
> (ceb Op jeu 2641)
le compte n'est pas bon
> (ceb Op jeu 418)
le compte est bon
```

5 Version approchée

On veut maintenant, si jamais le compte exact n'est pas bon, proposer le nombre qui s'en approche le plus. Pour cela, au lieu de tester si un element entier est dans une liste d'entiers, on va récupérer le nombre qui s'approche le plus du nombre cible.

Exercice 7 *Écrire une fonction* `approche` *qui prend en paramètre un entier et une liste, et renvoie l'élément de la liste le plus proche de cet entier.*

Exemple :

```
> (approche '(1 4 7 9) 3)
4
> (approche '(1 4 7 9) 7)
7
> (approche '(1 4 7 9) 45)
9
> (approche '(1 4 7 9) 8)
7
```

Exercice 8 *Modifier la fonction* `ceb` *pour afficher, si le compte n'est pas bon, la meilleure approximation possible.*

6 Liste des opérations effectuées

Dans la version originale du jeu, il est nécessaire de conserver une trace des calculs effectués jusqu'à l'obtention de la valeur conservée. Pour ce faire, on peut attacher à chaque valeur l'expression qui a permis de la calculer.

Exercice 9 *Modifier les structures de données, pour stocker non seulement les valeurs calculées, mais aussi les calculs qui ont permis de l'obtenir.*

Programmation applicative – L2
Troisième série de TP : Structure de données et graphisme, le jeu des dominos

C. Dony {Christophe.Dony@umontpellier.fr}
S. Daudé {Sylvain.Daude@umontpellier.fr}
D. Catta {Davide.Catta@umontpellier.fr}

1 Introduction

Cet enoncé concerne 3 séances de TP. Pensez à sauvegarder précieusement votre travail au fûr et à mesure des séances, pour pouvoir l'utiliser lors de l'examen final.

On décide de représenter un domino par une liste de deux entiers compris entre 0 et 6, 0 représentant le blanc. Le jeu dont un joueur dispose est représenté par une liste de dominos. Sur la table il y a une chaîne de dominos que nous représenterons aussi par une liste de dominos. Le joueur peut jouer s'il a un domino dont un des entiers est présent à une extrémité de la chaîne. Le cas échéant, la chaîne s'agrandit et le domino placé est retiré du jeu du joueur.

Nous allons mettre l'accent sur les structures de données mises en œuvre. Il est notamment **indispensable** d'identifier dans vos commentaires les accesseurs, constructeurs et fonctions de test pour chaque structure de données que nous utiliserons.

2 Structures de données `domino` et `jeu`

Exercice 1 *Écrire l'interface constructeur et accesseur de la structure de données domino.*

Exercice 2 *Écrire l'interface constructeur et accesseur de la structure de données jeu.*

Exercice 3 *Écrire une fonction* `est_double?` *qui teste si un domino est un double. Exemple :*

```
(est_double? '(3 3)) ==> \#t
```

Exercice 4 *Écrire ensuite une fonction* `doubles` *qui détermine tous les doubles d'un jeu. Exemple :*

```
(doubles '( (1 2) (3 3) (4 4) )) ==> ( (3 3) (4 4)).
```

Exercice 5 *Écrire une fonction* `retourner` *qui retourne un domino. Exemple :*

```
(retourner '(3 4)) ==> (4 3)
```

Exercice 6 *Écrire une fonction* `peut-jouer?` *qui prend en paramètre un entier n compris entre 0 et 6, un jeu j, et qui teste si le jeu j contient au moins un domino qui contient l'entier n. Dans ce cas la fonction rendra #t, sinon #f*

Exercice 7 *Écrire la fonction* `extraire` *qui prend en paramètre un entier n compris entre 0 et 6, un jeu j, et qui retourne le premier domino de j qui contient n. Pour cette fonction on fera l'hypothèse que j contient toujours un domino qui convient. Exemple :*

```
(extraire 3 '( (1 2) (3 3) (3 5))) ==> (3 3).
```

3 Manipulations des jeux et de la chaîne de dominos

Jouer au dominos consiste à bâtir une chaîne de dominos. Cette chaîne ne peut être agrandie qu'en ajoutant un domino compatible au début ou à la fin : si la chaîne commence (ou finit) par un n, on peut ajouter au début (ou à la fin) un domino contenant n. Exemple : si la chaîne est `( (3 0) (0 6) (6 6) (6 5) (5 2) )` alors on peut ajouter le domino `(3 4)` au début de la chaîne pour obtenir la chaîne `( (4 3) (3 0) (0 6) (6 6) (6 5) (5 2) )` ou le domino `(2 1)` à la fin pour obtenir la chaîne `((3 0) (0 6) (6 6) (6 5) (5 2) (2 1))`. Une chaîne est donc valide si le second élément de toute liste représentant un domino est égal au premier élément de la liste suivante (représentant le domino suivant).

Exercice 8 *Écrire une fonction* `chaine_valide?` *qui teste si une chaîne est valide.*

Exercice 9 *Écrire les fonctions* `ext_g` *et* `ext_d` *qui calculent respectivement la valeur de l'extrémité gauche et droite d'une chaîne. Exemple :*

```
(ext_g '( (3 0) (0 6) (6 6) (6 5) (5 2) ) ) ==> 3.
```

Exercice 10 *Écrire une fonction* `supprimer` *qui prend comme paramètre un domino d et un jeu j, et qui rend le jeu j duquel on a retiré d. Le domino d est supposé exister dans le jeu j.*

Exercice 11 *Écrire une fonction* `ajouter` *qui prend comme paramètre un domino d et une chaîne, et qui ajoute de façon cohérente le domino à la chaîne ch. On supposera que le domino peut toujours être ajouté à la chaîne. Exemples :*

```
(ajouter '(3 4) ' ((3 0) (0 6) (6 6) (6 5) (5 2) ))
  ==> ( (4 3) (3 0) (0 6) (6 6) (6 5) (5 2) )
(ajouter '(6 2) ' ((3 0) (0 6) (6 6) (6 5) (5 2) ))
  ==> ( (3 0) (0 6) (6 6) (6 5) (5 2) (2 6))
```

Exercice 12 *Écrire une fonction* `pose` *qui prend comme paramètre une liste (j ch) composée d'un jeu j et d'une chaîne ch et qui calcule la liste (jp chp) obtenue en ajoutant (si cela est possible) un domino d du jeu j à la chaîne ch. Le cas échéant, jp est j duquel on a retiré le domino d, et chp est ch à laquelle on a ajouté de manière cohérente le domino d. Dans le cas où aucun domino de j ne peut être ajouté à ch, le résultat est la liste (j ch).*

4 Partie graphique

Pour cette partie du TP, le niveau de langage utilisé doit être « niveau avancé ». D'autre part, choisissez l'item d'ajout d'un TeachPack dans le menu Langage, cliquez deux fois sur htdp, choisissez draw.ss et cliquez sur ok. Pressez ensuite le bouton executer.

4.1 Ouvrir une fenêtre

```
(start nombre1 nombre2) ==> void
```

Cette fonction ouvre une fenêtre dans laquelle pourront s'exécuter les fonctions de dessin[1]. Les deux arguments sont des nombres représentant respectivement la longueur et la hauteur de la fenêtre en pixels[2]. Cette fonction est évaluée à void, c'est-à-dire qu'elle ne retourne aucune valeur. Elle a cependant un effet sur l'écran puisqu'elle affiche une fenêtre. Exemple :

```
(start 320 200)
```

Toutes les fonctions suivantes auront besoin qu'une zone pour dessiner (canvas en anglais), ouverte avec la fonction start.

4.2 Les positions

Les « positions » sont des objets de scheme qui possèdent au moins deux valeurs. Les deux valeurs représentent des coordonnées dans le plan (exprimées en nombre de pixels) ; autrement dit, une position. Pour créer un objet « position » vous devez utiliser la fonction suivante :

```
(make-posn nombre1 nombre2) ==> position
```

Le repère utilisé est raccordé au coin supérieur gauche de la fenêtre ouverte. L'axe des abscisses est orienté de la gauche vers la droite et l'axe des ordonnées est orienté du haut vers le bas. Exemple :

```
(make-posn 20 34)
```

Étant donnée une position vous disposez également de deux fonctions, `posn-x` et `posn-y` qui permettent d'accéder aux valeurs des coordonnées en x et y de cette position.

```
(posn-x (make-posn 4 5)) ==> 4
(posn-y (make-posn 4 5)) ==> 5
```

4.3 Tracer une ligne

```
(draw-solid-line position1 position2) ==> true
```

Cette fonction trace une ligne entre les deux positions passées en arguments.

```
(draw-solid-line (make-posn 20 25) (make-posn 24 34))
```

1. aussi appelée « canvas »
2. L'image affichée à l'écran est constituée d'un assemblage de points minuscules appelés « pixels ». Le pixel est la plus petite surface de l'écran sur laquelle il peut agir.

4.4 Dessiner un disque

```
(draw-solid-disk position nombre) ==> true
```

dessine un disque centré autour de la position passée en premier argument et dont le rayon est spécifié par le nombre passé en second argument. Exemple :

```
(draw-solid-disk (make-posn 20 25) 5)
```

Commencez par ouvrir une fenêtre de 342 pixels sur 256 pixels

Exercice 13 *Écrire une fonction* `(dessiner-gros-point position) ==> true` *qui, étant donné une position, dessine un disque centré autour de cette position avec un rayon de 2 pixels.*

Exercice 14 *Écrire une fonction* `(dessiner-rectangle position1 position2) ==> true` *qui étant donné deux positions représentant respectivement le coin supérieur gauche et inférieur droit d'un rectangle, trace ce rectangle.*

Exercice 15 *Écrire une fonction* `(dessiner-demi-dominos position nombre) ==> true` *qui dessine un carré de 24 pixels de côté, centré autour de la position, avec autant de point(s) à l'intérieur que le nombre passé en second argument, compris entre 1 et 6, le spécifie.*

Exercice 16 *Écrire une fonction* `(dessiner-dominos position domino) ==> true` *qui, étant donné un domino, dessine ce domino autour de la position précisée. Le domino est dessiné horizontalement : un demi-domino à gauche et un demi-domino à droite.*

Exercice 17 *Écrire une fonction* `(dessiner-jeu-dominos jeu nombre) ==> true` *qui, étant donné un jeu de dominos (autrement dit une liste de dominos) et un numéro de joueur, affiche ces dominos empilés horizontalement en bas à gauche de la fenêtre pour le joueur 1 ou en bas à droite de la fenêtre pour le joueur 2. Chaque colonne ne peut contenir plus de 5 dominos. Si la colonne est pleine, on continuera d'empiler juste à côté. Un espace vide et large d'un pixel sera laissé entre les colonnes.*

Exercice 18 *Écrire une fonction* `(dessiner-chaîne-dominos chaîne) ==> true` *qui, étant donné une chaîne de dominos (autrement dit une liste de dominos), affiche ces dominos en ligne, tout en haut de la fenêtre. Si la longueur de la fenêtre est trop courte la chaîne doit continuer juste en dessous. Un espace vide et large d'un pixel sera laissé entre les colonnes.*

Exercice 19 *l'expression* `(random 7)` *renvoie un nombre au hasard entre 0 et 6. Utiliser cette expression pour écrire la fonction* `(générer-jeu x)` *qui génère au hasard un jeu de x dominos (donc une liste de x dominos). Écrire aussi une fonction* `InitChaîne` *qui génère une liste contenant un domino choisi au hasard pour servir de point de départ à la chaîne de jeu.*

Exercice 20 *Écrire la fonction* `début-jeu` *qui renvoie une liste composée de trois éléments : deux jeux et une chaîne de jeu générés au hasard. La chaîne chaîne de jeu initiale ne contient qu'un domino.*

Exercice 21 *Écrire une fonction* `(jouer j1 j2 ch)` *qui prend en entrée deux jeux de dominos et une chaîne, puis fait jouer alternativement chaque joueur jusqu'à ce que l'un d'entre eux n'ait plus de dominos dans son jeu, ou que l'un d'entre eux ne puisse plus poser de dominos. Graphiquement, les jeux et la chaîne seront redessinés dès qu'un nouveau domino est posé (on utilisera la fonction* `clear-all` *qui efface tout ce qui a été dessiné) après un temps d'attente de quelques secondes (on utilisera la fonction* `(sleep- for-a-while x)` *qui permet d'attendre x secondes avant l'évaluation de l'expression suivante).*

Exercice 22 *il ne reste plus qu'à écrire la fonction* `jeu` *qui fait appel à la fonction début- jeu et appelle, avec les jeux et la chaîne obtenus, l'expression* `(jouer j1 j2 ch)`*. Admirer le (votre) travail!!!*