Programmation Fonctionnelle I, Printemps 2017 – TD5

`http://deptinfo.unice.fr/~roy`

Où l'on s'aperçoit de l'inutilité théorique des mots while, for pour effectuer des calculs répétitifs... Récurrence, récurrence !

Exercice 5.1 Travail sur les **chiffres d'un entier**.

a) Si a et b sont deux entiers, avec b ≠ 0, rappelez quelles sont les expressions Scheme permettant d'obtenir le quotient de a par b, et le reste de la division de a par b.

b) Quelle est l'expression permettant d'obtenir le chiffre des unités d'un entier n [en base 10] ? Et le nombre obtenu en supprimant le chiffre des unités ? Autrement dit, à partir de 9876, comment obtenir 6 et 987 ?

c) Définissez une fonction `(nb-chiffres n)` retournant le *nombre* de chiffres de l'entier naturel n [en base 10]. Exemple :

```
(nb-chiffres 9876) → 4
```

Exercice 5.2 *Typique examen.* Soient d et n deux entiers ≥ 1. On dit que d est un « **diviseur strict** » de n si d divise n et si d < n. Par exemple les diviseurs stricts de 6 sont 1, 2, et 3. Un nombre entier n ≥ 2 est dit « **parfait** » s'il est égal à la somme de ses diviseurs stricts. Par exemple, 6 est parfait (6 = 1+2+3) mais 8 ne l'est pas (8 ≠ 1+2+4).

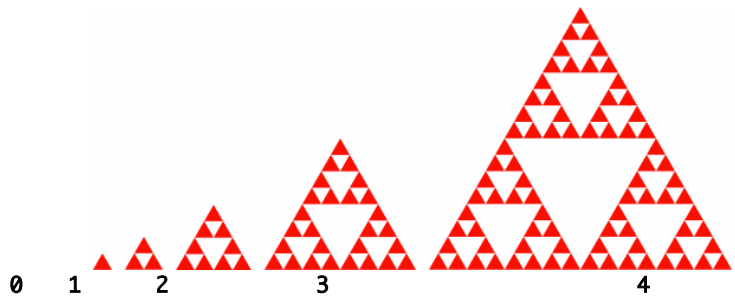
a) Définissez un prédicat `parfait?` prenant un entier n ≥ 1, et retournant `#t` si et seulement si `n` est parfait. Il vous faudra peut-être introduire un paramère supplémentaire ! Quel est l'ordre de grandeur du nombre d'operations lors du calcul de la valeur de `(parfait? n)` ?

b) Définissez une fonction `(nb-parfaits n)` prenant un entier n ≥ 1 et retournant le nombre d'entiers parfaits de l'intervalle `[1,n]`. Quel est l'ordre de grandeur du nombre d'opérations lors du calcul de `(nb-parfaits n)` ?

N.B. Il n'y a que trois nombres parfaits ≤ 1000. Personne n'a encore trouvé de nombre parfait impair ! Les mathématiciens ne savent même pas s'il y en a une infinité… http://fr.wikipedia.org/wiki/Nombre_parfait

Exercice 5.3 ***Courbe fractale.*** Définissez la fonction `(sierpinsky n)` retournant une image contenant le **triangle de Sierpinsky** de niveau `n`. Les figures ci-dessous montrent les 5 premiers niveaux (`n = 0...4`). Ces courbes se construisent par récurrence. Remarquez dans la courbe de niveau `n` la présence de 3 courbes de niveau `n-1`, et programmez...

N.B. La figure de niveau 0 est un triangle plein (cherchez la primitive `triangle` dans Racket).



Exercice 5.4 Programmez la fonction `(partition n m)` du cours 5 page 20.

Annexe : Python vs Scheme - Séance 5

Peu de choses à dire ici. Scheme gère le plus proprement possible (du point de vue de la mémoire) la **récurrence**. Python de son côté en décourage l'utilisation ☹ car :

a) il fait **exploser la pile de récursion** - *control stack* - assez tôt (cette pile mémorise entre autres les calculs restant à effectuer). Typiquement il fait planter 1000! par récurrence. Il est possible de régler manuellement la hauteur de cette pile avec le module *sys* mais encore faut-il prévoir à l'avance sa hauteur limite ! En Scheme, on calcule 50000! sans problème par récurrence, malgré quand même plus de 213000 chiffres...

b) il n'autorise **pas** les **calculs itératifs par récurrence**, en obligeant des constructions spéciales comme *while* ou *for*. En Scheme, nous verrons plus tard une forme de récurrence qui possède la même efficacité qu'une boucle *while* ou *for*, car Scheme n'utilise pas de pile de récursion lorsqu'elle ne sert à rien. La théorie date de la fin des années 1970 au MIT par Guy Steele, l'un des créateurs de Scheme (et actuel directeur de la recherche langages chez Oracle). Oui, la fin des années 1970, gasp Python !

Programmation Fonctionnelle I, Printemps 2017 – TP5

http://deptinfo.unice.fr/~roy

Exercice 5.1 a) Vérifiez en programmant une fonction bien choisie, que $1^2 + 3^2 + 5^2 + \cdots + 99^2 = 166650$

b) Calculez la somme des entiers de [50,100]. *Rép: 3825*

Exercice 5.2 a) Programmez la fonction récursive `(interfac a b)` prenant deux entiers a et b, et retournant le produit des entiers de `[a,b]`. Par exemple, `(interfac 5 10)` → `151200`.

• Que vaut `(interfac 10 5)` ? *Rappel : [a, b] = {x : a ≤ x ≤ b}, que vaut [10, 5] ?...*

b) En déduire la fonction factorielle `(fac n)`, avec `n` entier naturel.

c) Combien y-a-t-il de mains de 13 cartes dans un jeu de 52 cartes ?

Exercice 5.3 a) Programmez la fonction récursive `(integrale f a b h)` retournant l'intégrale approchée de la fonction `f` sur l'intervalle `[a,b]`, obtenue en découpant `[a,b]` en sous-intervalles de largeur `h`. Regardez le cours page 15. Exemple :

`(integrale sin 0 pi 0.01)` → `#i1.9999900283082575` ; *résultat inexact*

b) *A.N.* Calculez une valeur approchée de la constante π comme aire du cercle de centre 0 et de rayon 1, aire que vous ramènerez à un calcul d'intégrale.

c) *A.N.* Calculez une valeur approchée de l'intégrale $\int_{-1}^{1} e^{-x^2} dx$. *Rép ≈ 1.493*

Exercice 5.4 a) Définissez la fonction `(somme-chiffres n)` retournant la *somme* des chiffres de l'entier naturel `n` [en base 10]. Exemple : `(somme-chiffres 9876)` → `30`

b) Définissez par récurrence une fonction `(nb1 n)` retournant le nombre de 1 dans l'écriture binaire de l'entier naturel n. *Pour vérifier au toplevel, utilisez `(number->string n 2)` qui retourne l'écriture binaire de n sous la forme d'une chaîne de caractères.*

Exercice 5.5 Résolvez le problème suivant : combien y a-t-il de 0 à la fin du nombre `(fac 1000)` ? *Rép : 249.* A vous de trouver à quelle fonction récursive ce problème se réduit…

Exercice 5.6 On dispose n points distincts (n ≥ 0) sur la circonférence d'un cercle, et l'on note D_n le nombre de segments obtenus en joignant tous ces points deux à deux. Par exemple $D_4 = 6$ et $D_{10} = 45$.

a) Programmer par récurrence la fonction `(nb-segments n)` qui retourne D_n.

b) Montrez que D_n est en réalité un polynôme en n, ce qui donne une seconde programmation possible (plus rapide évidemment).

Exercice 5.7 Programmez par récurrence la fonction `(oursin n)` retournant une image contenant un oursin muni de `n` épines. Vous êtes libres des détails, obtenez une image réaliste, avec un soupçon d'aléatoire.