Planche de TD numéro II

1 L'instruction while

La boucle `for` est la plus naturelle des instructions de répétition, mais elle suppose de connaitre à l'avance le nombre d'itérations à réaliser. Il est fréquent qu'on veuille répéter un même calcul un nombre de fois a priori inconnu, mais tant qu'une certaine condition est satisfaite. On utilise pour cela l'instruction `while` dont la syntaxe est

```
while UneExpressionBooleenne :
      Bloc
```

L'effet de cette instruction est :

1. L'expression booléenne est évaluée.
2. Si sa valeur est Faux l'instruction `while` se termine. immédiatement.
3. Sinon on retourne au point 1 et ainsi de suite ...jusqu'à obtenir faux (où indéfiniment).

La conjecture de Collatz, ou d'Ulam ou encore de Syracuse Il s'agit d'un problème arithmétique d'énoncé très simple.*On se donne un entier naturel x et on considère la suite définie par $u_0 = x$, puis $u_{n+1} = u_n/2$ si u_n est pair, $(3u_n + 1)/2$ si u_n est impair. Il s'agit de démontrer que, quelle que soit la valeur initiale x, il existe une entier n tel que* $u_n = 1$. Autrement dit : si f est la fonction de Collatz définie dans la fiche précédente, démontrez que quel que soit l'entier $x \geq 1$ il existe une valeur de k telle que $f^k(x) = 1$. Ce problème n'est toujours pas résolu [1].

Exercice 1

1. Afficher, pour $x = 23$ les 15 premiers termes de cette suite. Quelle est la plus petite valeur de k telle que $f^k(23) = 1$?
2. Ecrire une fonction `Ulam(x)` qui calcule les u_n lorsque $u_0 = x$, et ne s'arretera que lorsque la valeur 1 sera atteinte. Il suffit pour cela d'initialiser une variable `u` dont la valeur initiale est x et dont les valeurs successives seront les u_n. Tant que `u` n'est pas égal à 1, vous remplacez la valeur de `u` par `f(u)`. Il suffit d'écrire

   ```
   def Ulam(x) :
      u=x
      while u != 1 :
            print 'u = ', u
            u = f(u)
   ```

   A quelle condition cet algorithme termine-t-il ? Cet exemple tout simple vous montre que la correction des programmes est difficile. On ne sait pas toujours répondre à la question : *Le programme suivant s'arrete t-il, quelles que soient les valeurs des paramètres ?*

[1] Cf. le lien http ://fr.wikipedia.org/wiki/Conjecture_de_Syracuse

3. Le lecteur attentif aura remarqué que le fonction `Ulam` définie ci-dessus ne renvoie pas de valeur. Elle se contente d'afficher des lignes à l'écran. Modifiez cette définition en ajoutant une variable locale `k` initialisée à à 0 et incrémentée de 1 à chaque appel de `f`. La fonction `Ulam(x)` rendra cette fois la valeur terminale de la variable `k` c'est à dire le plus petit entier tel que $f^k(x) = 1$.

Résolution d'une équation numérique par dichotomie Soit f une fonction réelle continue sur $I = [a, b]$ telle que $f(a)f(b) < 0$. On sait que f admet au moins un zéro dans l'intervalle $]a, b[$ (pourquoi ?). On se donne un nombre $\varepsilon > 0$ et on se propose de déterminer un intervalle J, de longueur plus petite que ε, qui contient un zéro de f.

1. Si la longueur $b - a$ est plus petite que ε il n'y a rien à faire, la réponse est $[a, b]$.
2. Si $b - a > \varepsilon$. Soit c le milieu de $[a, b]$. Si par extraordinaire $f(c) = 0$ on renvoie l'intervalle fermé réduit à c et tout est terminé. Si $f(a)f(c) > 0$ c'est que $f(b)f(c) < 0$, on remplace l'intervalle $[a, b]$ par l'intervalle $[b, c]$, sinon on remplace $[a, b]$ par $[a, c]$. Et on recomence tant que $b - a > \varepsilon$.

```
def dichotomie(f,a,b,eps) :
    while b-a > eps :
          c = (a+b)/2
          if f(c) == 0
              return [c,c]
          if f(a)f(c) > 0 :
              a=c
          else :
              b=c
    return [a,b]
```

Exercice 2 Démontrez que l'équation $x = \cos x$ admet une unique racine réelle, et que cette racine appartient à $[0, 1]$. Résolvez par dichotomie à 10^{-8} près l'équation $x = \cos x$. Vérifiez votre résultat à l'aide de la fonction prédéfinie `find_root`.

Exercice 3 Démontrez que la suite (x_n) définie par $x_0 = 0$ et $x_{n+1} = \cos(x_n)$ est convergente et que sa limite est solution de l'équation $x = \cos x$. Démontrez de plus que x_n est une valeur approchée par défaut de x lorsque n est pair, et une valeur approchée par excès lorsque n est impair. Calculez au moyen d'une boucle `for` les 30 premiers termes de la suite (x_n) et retrouvez ainsi une valeur approchée de la solution de $x = \cos(x)$.

2 Une définition de fonction peut être récursive

Nous avons vu que la syntaxe de définiton d'une fonction est la suivante

```
def f(x) :
    corps_de_fonction
```

On pourrait penser qu'aucun appel de la fonction `f` ne peut figurer dans une des instructions constituant le corps de la fonction, parce que cela n'a pas de sens d'utiliser un mot pour

définir ce mot. Et pourtant ceci vous est deja familier, dans le cas des suites par exemple. Vous avez deja rencontré la définition suivante : *Soit la suite* (u_n) *définie par* $u_0 = 1$ *et, pour* $n \geq 1$, $u_n = \cos(u_{n-1})$. Cette définition par récurrence de la fonction u s'écrit tout naturellement en SAGE

```
def u(n) :
    if n == 0 :
         return 1.0
    else :
         return cos(u(n-1))
```

Exercice 4

1. Vérifiez que cela fonctionne en affichant les 10 premiers termes de la suite $(u(n))$,

   ```
   [u(n) for n in [0..9]]
   ```

2. Que se passe t'il si dans la défintion de `u` vous remplacez `return 1.0` par `return 1` ?

En mathématique on parle de *définition par récurrence*, en informatique on dit plus volontiers *définition récursive*. N'hésitez pas à utiliser les définitions récursives. Vous découvrirez vite que c'est une technique qui vous permettra d'écrire rapidement des programmes justes.

Exercice 5 Écrivez une définition récursive de la fonction de Ulam de l'exercice 1

```
def UlamRec(N) :
    if N==1 :
       return 0
    else :
       return ...
```

3 La structure de dictionnaire

Le type **Dictionnaire** est une notions essentielle en algorithmique. Cette structure modélise la notion d'application définie sur un ensemble fini. Un **dictionnaire** `d` est une application $d : K \longrightarrow E$ sur un ensemble fini K à valeurs dans un ensemble quelconque E.[2] Les éléments de K sont appelées les **clefs** ou les entrées du dictionnaire. La valeur `d[k]` que prend la fonction `d` sur la clef `k` est souvent appelée l'information associée à `k`.

Exemple : Soit $K = \{1, 20, 3.14\}$, $E = \mathbb{R}$ et $f : K \longrightarrow N$ l'application définie par

$$f(1) = 0, \quad f(20) = 4.6, \quad f(3.14) = 100$$

On implémente ce dictionnaire en SAGE en écrivant,

```
f = {1 : 0, 20 : 4.6, 3.14 : 100}
```

[2] Cette strucuture est encore appelée *table associative*

Opération sur les dictionnaires Si `d` : $K \longrightarrow E$ est un dictionnaire défini sur `K` :

1. (a) La méthode `d.has_key(x)` renvoie `True` si `x` $\in K$ et `False` dans le cas contraire.
   (b) Si `x` est une clef, `d[x]` renvoie l'information associée à `x`.
   (c) La méthode `d.keys()` renvoie la liste des clefs, c'est à dire des éléments de K.
   (d) La méthode `d.values()` renvoie la liste `[f(k) for k in K]`.
   (e) La méthode `d.items()` renvoie la liste des `(k, d[k])` pour `k` dans K.
2. L'instruction `d[x] = y` ajoute à `K` une nouvelle clef `x` à laquelle est associée l'information `y`.

Exercice 6 Testez toutes ces méthodes sur le dictionnaire `f` crée dans la cellule ci dessus.

Exercice 7 La suite de Fibonacci est la suite définie par $u_0 = 0$, $u_1 = 1$ et, pour $n \geq 2$, $u_n = u_{n-1} + u_{n-2}$.

1. Définissez une fonction SAGE `fib0` qui calcule u_n

```
def fib0(n) :
    if n < 2 :
        return n
    return fib0(n-1) + fib0(n-2)
```

2. Calculez u_{10}, u_{20}, u_{30}.
3. Sauriez vous expliquer pourquoi le calcul `fib0(30)` est il si lent ?
4. Pour ne pas relancer à de nombreuses reprises le calcul de valeurs `fib0(n)` déja calculées, un moyen tout naturel est, chaque fois qu'on vient de calculer une valeur $f(m)$ de ranger le résultat dans un dictionnaire des valeurs de la fonction f, qu'on appellera `dicoFib`. On conserve la définition récursive, mais on n'utilise la relation de récurrence que pour calculer une valeur qui n'a encore jamais été calculée. Complétez pour cela la cellule

```
dicoFib={0 :0, 1 :1}
def fib1(n) :
    if dicoFib.has_key(n) :
           return dicoFib[n]
    else :
           dicoFib[n] = fib1(n-2)+fib1(n-1)
           return dicoFib[n]
```

5. Calculez u_{1000}.

Exercice 8 L'exercice précédent avait pour but d'illustrer la notion de dictionnaire. C'était tout de même prendre un marteau pilon pour écraser une mouche.

1. Réécrivez une définition non récursive de la fonction `fib2`, en utilisant deux variables locales `u0, u1` initialisées à 0 et 1, et une boucle `for`.
2. Calculez u_{10000}.

4 Exercices complémentaires

Exercice 9 (Visualisations de développements limités) Soit

$$f(x) = \frac{\cos(2x)}{2\sqrt{1+\ln(x^2+x+1)}}$$

1. Déterminer les parties régulières f_1, f_2, f_3, f_4 des développements limités d'ordres $1, 2, 3, 4$ de f au voisinage de 0.
2. Construire les graphes $p, p_1, \ldots, p_4$ de $f, f_1, \ldots, f_4$, pour x variant de -2 à 2. Afficher la superposition de ces 5 graphes.

Exercice 10 (À propos de la représentation des réels en virgule flottante)

1. Que pensez vous de la suite (x_n) définie par $x_0 = 1/3$, et $x_{n+1} = 4x_n - 1$?
2. Lancez la commande

```
x=1/3
for n in range(30):
        x=4*x-1
print " x = " , x
```

3. Refaites le même calcul en remplaçant simplement la première ligne par

```
x=1.0/3
```

4. Que s'est il passé ?

Exercice 11 (Compléments à propos du précédent)

1. Quelle est l'écriture en base 2 de la fraction $1/3$?
2. Soit $y = \sum_{n=1}^{27} \frac{1}{4^n}$. Quelle est l'écriture de y en base 2 ?
3. Exécutez la séquence

```
y = add([1/4^k for k in [1..27]])
for n in range(30):
     y = 4*y-1
print " y = " , y
```

On est tenté d'"en déduire que la valeur du nombre flottant mémorisé par la commande `Sage` $x = 1./3$ n'est pas $1/3$ mais le nombre y dont l'écriture binaire est $\hat{x} = 0,0101\ldots01$ où la séquence 01 est répétée exactement 27 fois. C'est vrai.

Explication : Dans `Sage`, par défaut, les nombres réels sont représentés en suivant la norme IEEE754 sur 64 bits. Si $x \neq 0$ il s'écrit de manière unique

$$x = \pm |x| = \pm 2^E m$$

où m est un nombre de l'intervalle $[1, 2[$ qu'on appelle *la mantisse de* x. Le premier bit est un 0 ou un 1 selon que $x \geq 0$ où $x < 0$. Les 11 bits suivant sont utilisés pour représenter

l'entier E. Enfin les 52 bits restant servent à représenter la mantisse. Puisque la mantisse est dans l'intervalle $[1, 2[$ l'écriture en base 2 de m est de la forme

$$m = 1, c_1 c_2 \dots c_k \dots$$

où les c_i sont des 0 où des 1. Les 52 bits représentant m sont les 52 premiers bits $c_1 c_2 \dots c_{52}$ de la partie fractionnaire de l'écriture binaire de m. Le nombre mémorisé dans la machine est donc

$$\hat{x} = \pm 2^E \left(1 + \sum_{k=1}^{52} c_k 2^{-k}\right)$$

Dans le cas de $x = 1/3$, on a $x = 2^{-2}\frac{4}{3}$. La mantisse est donc $4/3 = 1,0101\dots$, et l'exposant est -2 d'où

$$\hat{x} = 2^{-2}\left(1 + \sum_{k=1}^{26} \frac{1}{4^k}\right) = \sum_{k=1}^{27} \frac{1}{4^k}.$$

Exercice 12 Ecrire une définition récursive de la fonction `dichotomie(f,a,b,eps)`.

Exercice 13 (Le jeu des tours de Hanoï.) Le matériel est composé de, trois plots verticaux, nommés `A,B,C`, et de N disques percés en leur centre d'on orifice de diamètre égal au diamètre des plots `A,B,C`. Ces N disques sont de tailles différentes, numérotés de 1 à N, du plus petit au plus gros. Initialement ils sont empilés sur le plot A, par ordre de taille décroissante, c'est à dire, de bas jusqu'en haut, $N, N-1, \dots, 2, 1$. Le but du jeu est de réaliser la même configuration, mais avec tous les disques empilés dans le même ordre sur le plot `C`. La règle du jeu est : On prend le disque le plus haut placé sur l'un des 3 plots, et on peut le poser au sommet de l'un des autres plots, pourvu qu'on ne pose jamais un disque sur un disque plus petit que lui. Nous appelerons `Hanoi(A,B,C,N)` la suite des tansferts à effectuer pour résoudre ce problème.
Soit $N \geq 2$. Supposons que l'on sache résoudre le problème à l'ordre $N-1$. Vu la règle du jeu, on ne pourra déplacer le plus gros disque de `A` vers `C` qu'après avoir déplacé les $N-1$ autres disques depuis `A` vers `B`.

1. On commence done par transférer les $N-1$ plus petits disques de `A` vers `B`.
2. Le plus gros disque, celui de numéro N est alors tout seul sur le plot `A`, tandis que le plot `C` est vide. On transfère donc le disque N de `A` vers `C`.
3. Le plot `A` est maintenant vide. Il ne reste plus qu'à transférer les $N-1$ plus petits disques de `B` vers `C`.

Autrement dit résoudre `Hanoi(A,B,C,N)` avec $N \geq 2$ c'est résoudre `Hanoi(A,C,B,N-1)`, puis transférez le disque N de `A` vers `C`, et enfin résoudre `Hanoi(B,A,C,N-1)`.

Une objection à laquelle on pourrait songer est la suivante : *est il possible de résoudre les problèmes* `Hanoi(A,C,B,N-1)` *et* `Hanoi(B,A,C,N-1)` *sans tenir compte de la présence du gros disque, le disque de numéro* `N` *?* Oui bien sûr, parce que le disque numéro N étant plus gros que tous les autres, tout se passe comme s'il n'était pas la. On peut poser sur lui n'importe lequel des autres disques.

1. Complétez la cellule suivante pour définir la fonction `Hanoi(A,B,C,N)` qui ne revoie pas de résultats mais qui affiche la suite des déplacements pour transférer N disques de `A` vers `C`, en utilisant le plot intermédiaire `C`.

```
def Hanoi(A,B,C,N) :
  if N==1 :
       print 'De ', A, ' vers ', C
  else :
       ...
```

2. Combien de transferts faut il effectuer pour résoudre `Hanoi(A,B,C,N)` ?

Exercice 14 (Le paradoxe des anniversaires) La paradoxe des anniversaires est un résultat combinatoire simple et amusant et, ceci est plus inattendu, il joue en grand rôle en algorithmique, en particulier dans le domaine de la cryptographie.
Soit E un ensemble de cardinal n. On se donne un entier k et on tire successivement, avec remise, et avec chaque fois la probabilité uniforme, un élément de E. Cela définit une suite $(x_1, x_2, \ldots, x_k)$ d'éléments de E. Quelle est la probabilité p_k de l'évenement *tous les x_i sont distincts* ?

1. Quel est le cardinal de E^k ?
2. Combien y a-t-il de k uplets $(x_1, x_2, \ldots, x_k) \in E^k$ dont les composantes $x_1, x_2, \ldots, x_k$ sont deux à deux distinctes ?
3. En déduire que la probabilité p_k est donnée par $p_k = \prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right)$.
4. Prouver que pour tout réel x on a $1 + x \leq e^x$.
5. En déduire que $p_k \leq e^{-\frac{k(k-1)}{2n}}$.
6. En déduire que si $k \geq \frac{1}{2}\left(1 + \sqrt{1 + 8n \ln 2}\right) \approx 1.17\sqrt{n}$, on a $p_k < 1/2$. Autrement dit : Si on tire au hasard avec remise et probabilité uniforme, un peu plus de $\sqrt{n}$ éléments dans un ensemble de cardinal n, la probabilité d'obtenir au moins deux fois le même élément est plus grande que $1/2$.
7. Pour $n = 365$ démontrez que si 23 personnes sont réunies dans une salle la probabilité que deux au moins d'entre elles aient leur anniversaire le même jour, est plus grande que $1/2$.
8. **Verification expérimentale** Réalisons l'expérience suivante : on choisit un entier N, par exemple $N = 365$, puis on tire au hasard des éléments de $\{1, 2, \ldots, N\}$ (les dates d'anniversaires des élèves qui entrent 1 à 1 dans la classe). On arretera le tirage la première fois que le tirage redonnera un entier déja obtenu.

   La solution la plus naturelle est d'utiliser un dictionnaire, dont les clefs sont les valeurs `x` déja tirées, et l'information associée à la clef `x` est le rang du tirage qui a donné cette valeur `x`. C'est pourquoi nous appellerons `rang` ce dictionnaire.

   Lors de chaque tirage on regarde si la valeur tiréex a déja été tirée : si oui l'expérience s'arrete et on affiche le rang `k` de ce tirage ainsi que le numérp du tirage qui avait le premier donnné `x`. Cela conduit immédiatement à l'écriture suivante :

```
Paradox(N) :
rang = {}
for k in range(N) :
   x = randint(1,N)
   if rang.has_key(x) :
      print 'k = ', k, ' donne', x, ' deja obtenu pour k= ',rang[x]
      return
   else rang[x] = k+1
```

9. En relançant plusieurs fois le calcul Paradox(365) vérifiez que le nombre de tirages jusq'à la première apparition d'une valeur déjà tirée oscille autour de 23.

Exercice 15 (Sur la calcul récursif des nombres de Fibonacci)
On considère la définition récursive de la fonction de Fibonacci, donnée à la question 1 de l'exercice 7. On appelle $t(n)$ le temps de calcul de l'appel `fib0(n)` qui renvoie la valeur du nombre de Fibonacci u_n.

1. Les opérations arithmétiques sur les entiers entiers qui interviennent dans la calcul de `fib0(n)` ont un coût croissant avec la longueur de leurs écritures. Cependant les entiers ne dépassant pas $2^{31}-1$ n'occupent qu'un mot mémoire de la machine, et, pour ces entiers il est raisonnable de considérer que les opérations arithmétiques sont de coût constant. Pour $n \leq 46$ on a `fibo(n)` $\leq 1836311903 < 2^{31}$. On considèrera donc que toutes les opérations arithmétiques intervenant dans le calcul de `fib0(n)` pour $n \leq 46$ sont de coût constant. Sous cette hypotèse démontrer qu'il existe deux réels a, b, strictement positifs tels que $t(n)$ satisfait

$$t(n) = \begin{cases} a & \text{si } n \leq 1 \\ b + t(n-1) + t(n-2) & \text{sinon} \end{cases}$$

   Notons $v_n = t(n) + b$. Quelle relation de récurrence plus simple est elle satisfaite par v_n ? En déduire que $t(n) = (a+b)u_{n+1}$.

2. Soit $\varphi = \dfrac{1+\sqrt{5}}{2}$ le nombre d'or. On admet que u_n est l'entier le plus proche de $\dfrac{\varphi^n}{\sqrt{5}}$. Démontrez que le temps de calcul de l'appel `fib0(n)` est environ multiplié par 11 chaque fois que l'on ajoute 5 à `n`. Vérifiez le en évaluant par exemple

```
time fib0(25)
```

   puis la même chose en remplaçant 25 par 30.

Exercice 16 Reprenez maintenant la fonction `fib2(n)` de l'exercice 8 qui calcule aussi le nombre de Fibonacci u_n, pour de grandes valeurs de n. Pour ces valeurs de n les nombres manipulés ont un grand nombre de chiffres, proportionnel à $\log(u_n)$.

1. En tenant compte de ceci montrer que le temps du calcul `fib2(n)` est proportionnel à n^2.
2. Vérifiez en évaluant successivement

   `time x=fib2(50000)`, `time x=fib2(100000)`, `time x=fib2(200000)`.