TD n°05 : Petits programmes simples

3 heures

Rédigé par Pascal Delahaye

14 novembre 2014

Quelques rappels :

1. Sur les listes :
   (a) Les composantes d'une liste L sont indicés de 0 à $n-1$ et sont appelées par l'instruction L[i].
   (b) On obtient la longueur d'une liste à l'aide de la fonction : `len()`

2. Sur les chaînes de caractères :
   (a) La concaténation de deux chaînes de caractères C1 et C2 se fait par l'instruction : C1 + C2
   (b) On peut transformer une donnée en chaîne de caractères à l'aide de la fonction : `str()`

1 Exercices sur les structures de contrôles

1. Construire une fonction d'arguments les flottants x et t qui renvoie la valeur 1 si $t = x$ et 0 sinon.

2. Construire une fonction d'arguments deux listes L1 et L2 de longueur 2 qui nous dit si L1 = L2.
   On s'interdira l'utilisation de la condition `L1 == L2`.

3. Construire une fonction d'argument deux entiers n et p qui nous dit si $n = p$, si $n < p$ ou si $n > p$.

4. Construire une fonction d'arguments deux flottants `r` et `alpha` qui renvoie un argument du complexe $re^{i\texttt{alpha}}$.

5. Construire une fonction d'argument un flottant q qui nous dit si la suite q^n est convergente ou pas.

2 Exercices sur les boucles "for"

1. Construire une fonction d'arguments deux listes L1 et L2 qui nous dit si les deux listes sont identiques ou pas.
   On pourra utiliser la fonction `len()` *ainsi qu'une variable auxilliaire initialisée à* 0 *et prenant la valeur* 1 *si deux composantes sont distinctes.*

2. Construire une fonction qui affiche les composantes d'une liste L en les séparant par des tirés.
   On pourra construire une chaîne de caractères et utiliser la fonction `str()`.

3. Construire une fonction qui teste si la somme des composantes d'indices pairs d'une liste est égale à la somme des composantes d'indices impairs. *On fera des cas selon la parité de la longueur de la liste.*

4. Construire une fonction qui compte le nombre de fois où l'on rencontre la valeur x dans une liste L donnée entre les composantes d'indices i et j.

5. Soit (u_n) la suite définie par la donnée de $u_0 = x > 0$ et la relation $u_{n+1} = \frac{1}{2}(u_n + \frac{x}{u_n})$.

   (a) Construire une fonction d'arguments le flottant x et l'entier $n > 0$ renvoyant la valeur de u_n.

   (b) Comparer les résultats renvoyés par cette fonction avec les valeurs de $\sqrt{x}$ pour $n = 100$ et $x \in [\![1, 10]\!]$.
   Bien entendu, on utilisera pour cela une boucle "for".

6. Soit (a_n) et (b_n) deux suites récurrentes définies par la donnée de $\begin{cases} a_0 > 0 \\ b_0 > 0 \end{cases}$ et les relations $\begin{cases} a_{n+1} = \sqrt{a_n.b_n} \\ b_{n+1} = \dfrac{a_n + b_n}{2} \end{cases}$.

   Construire une fonction d'arguments les flottants a_0 et b_0 et l'entier N qui affiche de façon explicite les valeurs de a_n et b_n pour $n \in [\![0, N]\!]$.

7. Les deux suites précédentes sont adjacentes et convergent vers la même limite appelée *moyenne arithmético-géométrique* de a_0 et b_0.

   Construire une fonction d'arguments l'entier naturel p et les flottants a et b qui donne la valeur de la moyenne arithmético-géométrique de a et b à 10^{-p} près.

8. Construire une fonction MAX donnant le maximum des composantes d'une liste.

9. Les nombres de catalan sont les nombres définis par $c_0 = 1$ et la relation $c_n = \frac{2(2n-1)}{n+1} c_{n-1}$.

   Construire une fonction renvoyant la valeur du nombre c_n.

   Remarque : vous comparerez les résultats renvoyés par les deux programmes utilisant les formules respectives suivantes : `c = (2*(2*k-1)*C)/(k+1)` *et* `c = (2*(2*k-1)/(k+1))*C` *(justes en théorie)*

10. Les nombres de catalan sont aussi donnés par la formule $c_n = \frac{1}{n+1}\binom{2n}{n}$.

    Construire une fonction renvoyant c_n et qui utilise la formule précédente.

11. — Construire une fonction d'arguments un entier naturel n et un flottant x qui renvoie la valeur de $\sum_{k=0}^{n} \frac{x^k}{k!}$.

    — Comparer les valeurs renvoyées par cette fonction avec les valeurs de e^x lorsque $n = 10$ et $x \in \{0,\ 0.1,\ 0.2,\ \ldots,\ 1.9,\ 2\}$.
    Bien entendu, on utilisera pour cela une boucle "for".

12. On dit qu'un entier naturel est parfait s'il est égal à la somme de ses diviseurs positifs stricts.
    Construire une fonction d'argument un entier $n \geq 2$ qui nous dit si un nombre n est parfait.

3 Exercices sur les boucles "while"

Construire les fonctions :

1. Construire une fonction d'argument un entier m qui renvoie le plus petit entier n tel que $2^n \geq m$.

2. Construire une fonction d'argument un flottant A qui renvoie le plus petit entier n tel que $\sum_{k=1}^{n} \frac{1}{k} \geq A$.

3. Construire une fonction qui renvoie les n premiers nombres premiers.

4. Construire une fonction qui détermine la longueur d'une liste.
   On utilisera la fonction `L.pop(0)` mais pas la fonction `len()`.

5. L'algorithme de Syracus consiste à transformer la valeur d'un entier naturel non nul tant que celui-ci n'est pas égal à 1 de la façon suivante :
   — On divise par 2 cet entier s'il est pair
   — On le multiplie par 3 et on lui ajoute 1 s'il est impair

   Construire une fonction qui détermine le nombre d'étapes pour atteindre la valeur 1 en partant d'un entier n.

6. Soit (u_n) définie par $u_n = \sum_{k=0}^{n} \frac{1}{k!}$ et (v_n) défnie par $v_n = u_n + \frac{1}{n!}$.
   Nous avons vu en cours de math que les suites (u_n) et (v_n) suivantes sont adjacentes et tendent vers e.
   Déterminer une fonction d'argument p qui renvoie une approximation de e à 10^{-p} près.

7. Construire une fonction qui détermine par dichotomie si une liste L de composantes entières triées admet une de ses composantes égale à e.

8. Construire une fonction qui détermine la solution d'une équation de la forme $f(x) = 0$ (f continue) à ε près par dichotomie.

4 Exercices sur les itérables

Exercice : 1

(∗) Construire un programme :

1. permettant de compter le nombre de voyelles contenues dans une phrase.
2. permuttant toutes les voyelles d'une phrase.

Exercice : 2

(∗) Construire un programme permettant d'obtenir les n premiers termes d'une suite récurrente.

Exercice : 3

(∗) Construire un programme permettant de savoir si un élément e se trouve dans une liste L.

Exercice : 4

(∗) Construire un algorithme permettant de déterminer dans le tableau T contenant des entiers, la ligne dont la somme des éléments est la plus petite et celle dont la somme est la plus grande.
Ce programme renverra les numéros de lignes et les valeurs des sommes correspondantes.

Exercice : 5

(∗) En utilisant le principe de décomposition vu en début d'année, contruire un programme renvoyant sous la forme d'une chaine de caractères la décomposition d'un entier n en base b.
On pourra utiliser la fonction `str()` qui transforme un entier en chaine de caractères

Exercice : 6

(∗∗) En utilisant le principe du crible d'Eratosthène, construire un programme donnant la liste des nombres premiers inférieurs ou égaux à un entier N donné.

Exercice : 7

(∗∗) Construire un tableau de taille $n \times n$ contenant le triangle de pascal.
Le calcul devra se faire en utilisant la formule d'addition portant sur les coefficients binomiaux.

Exercice : 8

(∗∗) Construire un programme permettant de trier *par sélection* une liste de nombres entiers distincts.