Série 3 – Apparition des fonctions

Exo 3.1 – Puissance d'un nombre, version itérative

Écrivez une fonction `puissance(x, n)` qui calcule et rend comme résultat la valeur de x^n pour un nombre flottant x quelconque et un nombre entier n vérifiant $n \geq 0$.

A. Dans un premier temps, programmez l'algorithme simple qui effectue $n-1$ multiplications.

B. Dans une deuxième version, on va s'intéresser à un algorithme plus efficace (c.-à-d. exécutant moins de multiplications) que le précédent. Pour cela, vous programmerez une boucle mettant à profit la remarque suivante :

$$x^n = \begin{cases} \left(x^2\right)^{\frac{n}{2}} & \text{et si } n \text{ est pair, } \frac{n}{2} \text{ est entier} \\ x \times x^{n-1} & \text{et si } n \text{ est impair, } n-1 \text{ est pair} \end{cases}$$

Indication. Initialisez à 1 une variable r qui, à la fin, contiendra le résultat de la fonction, puis écrivez une boucle dans laquelle vous testez la parité de n : si n est pair (c'est-à-dire si $n\%2 = 0$) remplacez x par x^2 et n par $\frac{n}{2}$. Si n est impair ($n\%2 \neq 0$) remplacez r par $r \times x$ et n par $n-1$. Notez qu'à chaque tour la valeur de n diminue : arrêtez la boucle lorsque $n = 0$.

Pour illustrer le fonctionnement de la boucle que vous devez écrire, voici les valeurs successives (chaque ligne correspond à une itération) des variables x, n et r lors du calcul de la valeur de 3^{22} :

x	n	r
3,0	22	1,0
9,0	11	1,0
9,0	10	9,0
81,0	5	9,0
81,0	4	729,0
6 561,0	2	729,0
43 046 721,0	1	729,0
43 046 721,0	0	31 381 059 609,0

Observez cette importante propriété : à toutes les lignes on a $x^n \times r = 3^{22}$. Or, à la dernière ligne, $n = 0$ donc $x^n = 1$ et $r = 3^{22}$.

Exo 3.2 – Évaluation d'un polynôme : schéma de Horner

Écrivez une fonction `eval` qui calcule et rend comme résultat la valeur d'un polynôme $P(x) = a_0X^n + a_1X^{n-1} + \ldots + a_{n-1}X + a_n$ pour une valeur donnée $X = x_0$ de l'inconnue X. La fonction prendra pour arguments la *liste* des coefficients $(a_0, a_1, \ldots a_{n-1}, a_n)$ du polynôme et la valeur x_0 en question.

Par exemple, pour obtenir la valeur du polynôme $P(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$ pour $x_0 = 3$ il suffira d'écrire

```
eval([1, -10, 35, -50, 24], 3)
```

Pour minimiser le nombre de multiplications faites par votre fonction, écrivez une boucle mettant à profit la décomposition suivante, appelée *schéma de Horner* :

$$P(x_0) = ((((a_0x_0 + a_1)\,x_0 + a_2)\,x_0 + \ldots)\,x_0 + a_{n-1})\,x_0 + a_n$$

T.S.V.P.

Indication. Cherchez ce qui est commun aux transformations suivantes :

$$a_0 \to a_0 x_0 + a_1$$

puis

$$a_0 x_0 + a_1 \to (a_0 x_0 + a_1) x_0 + a_2$$

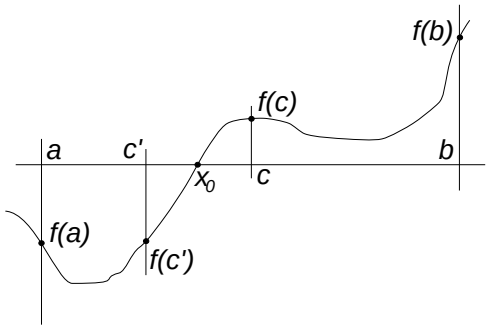
puis

$$(a_0 x_0 + a_1) x_0 + a_2 \to ((a_0 x_0 + a_1) x_0 + a_2) x_0 + a_3$$

et ainsi de suite. Regardez-bien, vous devriez voir apparaître la boucle à écrire...

Exo 3.3 – Résolution de $f(x) = 0$ par dichotomie

Si f est une fonction définie et continue sur un intervalle $[a, b]$ vérifiant $f(a) < 0 < f(b)$, il existe certainement au moins une valeur $x_0 \in [a, b]$ telle que $f(x_0) = 0$; on dit que x_0 est un *zéro* de f sur $[a, b]$



Une valeur approchée à ϵ-près d'un zéro de f sur $[a, b]$ est une valeur $x_1 \in [a, b]$ telle qu'il existe $x_0 \in [a, b]$ vérifiant $|x_1 - x_0| < \epsilon$ et $f(x_0) = 0$. Autrement dit, x_1 n'est pas un zéro de la fonction, mais il est plus près que ϵ d'un tel zéro.

Écrivez une fonction $zero(f, a, b, epsilon)$ qui recherche et renvoie une valeur approchée à ϵ-près d'un zéro de la fonction f sur l'intervalle $[a, b]$.

Cette fonction devra se comporter correctement aussi bien lorsque $f(a) < 0 < f(b)$ que lorsque $f(a) > 0 > f(b)$.

Pour essayer votre fonction calculez :
- une valeur approchée de $\sqrt{2}$ à 10^{-10} près,
- une solution approchée à 10^{-6} près de l'équation $cos(x) = x$.

Indication. La méthode dite *par dichotomie* recherche une telle valeur en exploitant la remarque suivante : posant $c = \frac{a+b}{2}$, si $y_c = f(c) > 0$ alors il existe un zéro de f sur $[a, c]$ (c'est le cas de la figure), sinon il en existe un sur $[c, b]$.

Il suffit alors de remplacer l'intervalle initial $[a, b]$ par un des intervalles $[a, c]$ ou $[c, b]$, puis de recommencer ces opérations. On itère de la sorte, jusqu'à ce que la longueur de l'intervalle retenu soit inférieure à ϵ ; n'importe lequel de ses points répond alors à la question.

Exo 3.4 – Puissance d'un nombre, version récursive

Donnez une version récursive de la fonction de l'exercice 3.1. C'est-à-dire, écrivez une fonction qui au lieu de faire tourner une boucle, règle le problème par un appel d'elle-même, après avoir modifié les valeurs des arguments.

Indication. Inspirez-vous fortement de ceci :

$$\begin{cases} \text{si } n = 0,\ x^n = 1 \\ \text{sinon, si } n \text{ est pair, } x^n = (x^2)^{\frac{n}{2}} \\ \text{sinon, } x^n = x \times x^{n-1} \end{cases}$$

Pour vous assurer que vous avez compris comment travaille votre fonction, représentez la suite d'appels (expressions de la forme *puissance(x,n)*) et de retours (instructions *return expr*) qui ont lieu à l'occasion d'un appel tel que `y = puissance(3, 22)`.

Exo 3.5 – Calcul de C_n^p

Les coefficients binomiaux C_n^p, avec $n \geq 0$ et $0 \leq p \leq n$, définis par $C_n^p = \frac{n!}{p!(n-p)!}$, vérifient

$$C_n^p = \begin{cases} 1, \text{ si } p = 0 \text{ ou } p = n \\ C_{n-1}^{p-1} + C_{n-1}^p, \text{ si } 0 < p < n \end{cases}$$

Cette propriété est à la base du *triangle de Pascal* bien connu (chaque ligne correspond à une valeur de n, chaque colonne à une valeur de p) :

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5   10   10    5    1
1    6   15   20   15    6    1
1    7   21   35   35   21    7    1
1    8   28   56   70   56   28    8    1
```

Écrivez une fonction `C(n, p)` qui calcule et renvoie C_n^p en mettant à profit la propriété précédente.

Pour constater le grand nombre d'appels d'elle-même que fait cette fonction, ajoutez-y un compteur qui augmente de 1 chaque fois que la fonction est appelée et qu'on affiche à la fin du programme.

Remarque. Pour qu'une variable *var* puisse être modifiée dans une fonction, par exemple par une affectation telle que « *var* `=` *var* `+ 1` », sans que cela crée une nouvelle variable locale, il faut que *dans la fonction* la variable soit déclarée globale :

```
global var
```

• • •