TD4 Python : Fonctions

EXERCICE 1: Factoriel

Écrire un programme avec une fonction factoriel (donnée en cours), de paramètre un entier n, qui renvoie n !. Dans le corps principal du programme, on demandera à l'utilisateur de rentrer n (on redemande la saisie jusqu'à ce que n≥0), ensuite on fera appel à la fonction pour afficher n!.

EXERCICE 2: Disque et cylindre

En Python, `pi` (obtenue par : `from math import pi`) correspond à une valeur approchée de π.

a) Écrire un programme qui calcule le périmètre d'un cercle: on écrira d'abord une fonction dépendant du rayon (réel) qui retourne le périmètre d'un cercle, puis dans le corps principal on demandera à l'utilisateur de rentrer le rayon r, ensuite on fera appel à la fonction et on affichera le résultat. Exemple: si r=3, périmètre~18.85

b) Améliorer le programme en vérifiant après la saisie de r, que ce rayon est positif et en le redemandant jusqu'à ce qu'il soit positif.

c) Rajouter une fonction qui calcule la surface du disque de rayon r. Dans le corps principal après la saisie et le contrôle de r, demander à l'utilisateur de choisir entre p et s, la saisie de p engendrant le calcul du périmètre, la saisie de s engendrant le calcul de la surface. Exemple: si r=3, surface~28.274

d) Créer deux nouvelles fonctions exploitant les fonctions de a) et c), avec de façon analogue la surface et le volume d'un cylindre de rayon r, de hauteur h. Exemple: avec r=2 et h=3, on obtient une surface de 62,83 et un volume de 37,70.

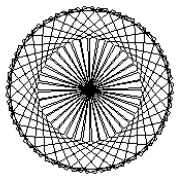
EXERCICE 3: Affichage des termes successifs d'une suite

Écrire un programme avec une fonction de paramètre n qui affiche la valeur des termes successifs u_0 jusqu'à u_n, d'une suite (u_n) définie par $u_0=1$, $\forall\ n \geq 0$, $u_{n+1}=2*u_n-3$. Dans le programme on demandera à l'utilisateur de saisir n, puis on appellera la fonction (u_1=-1 , u_2=-5 , u_3=-13 , u_4=-29 , u_5=-61 , ...) .

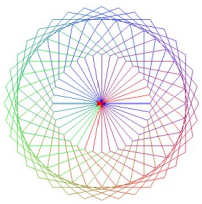
EXERCICE 4: Rosace

a) En utilisant le module **turtle,** écrire une fonction **polygone**(taille, nbCotes) qui dessine un polygone régulier dont la taille et le nombre de côtés sont donnés en paramètres. On rappelle que si un polygone régulier a n côtés l'angle entre deux côtés consécutifs vaut 360°/n.

b) Écrire une fonction **rosace**(taille, nbCotes) qui dessine une rosace en dessinant plusieurs polygones dont la taille et le côté sont donnés en paramètres. Le premier côté du k-ième polygone dessiné sera incliné d'un angle 10°, et on fera varier cet angle de 0 à 350 degrés, par pas de 10 degrés. On dessine donc au total 36 polygones, inclinés de 10 degrés en 10 degrés. Exemple : ici une rosace obtenue avec des pentagones



c) Ajouter une fonction **rosace_coloree**(taille, nbCotes) qui dessine une rosace comme dans la question précédentes mais en colorant chaque polygone de sorte à obtenir un dégradé du rouge au vert, du vert au bleu, du bleu au rouge. Pour cela on utilisera la fonction **color**(r,g,b) où r, g, b sont des réels compris entre 0 et 1 qui indiquent les doses de rouge, vert, bleu.
Exemple : ici une rosace colorée obtenue avec des hexagones



EXERCICE 5: Coefficients du binôme

a) Écrire un programme avec une fonction de paramètres n et p, qui calcule C_n^p

(rappel: $C_n^p = \frac{n!}{p!(n-p)!}$, on recopiera la fonction factoriel de l'exercice 1)

La fonction doit seulement renvoyer la valeur (pas d'affichage dans la fonction).
Dans le corps principal du programme, on demande à l'utilisateur un entier naturel n et un entier p compris entre 0 et n, on vérifie que la condition sur p est bien réalisée et on affiche C_n^p (exemple : C_8^5 =56).

b) Écrire une fonction de paramètre n, qui affiche le développement de $(a+b)^n$ correspondant sous la forme (pour n=4 par exemple): (on reprendra les fonctions précédentes)

```
(a+b)^4 = a^4 + 4 a^3 b^1 + 6 a^2 b^2 + 4 a^1 b^3 + b^4
```

Rappel: $(a+b)^n = \sum_{i=0}^{n} C_n^i \; a^{n-i} \; b^i$. (N.B.: on appellera la fonction du a) pour le calcul des C_n^p). Dans le corps principal, on demande à l'utilisateur d'entrer un entier naturel n, puis on appelle la fonction.

c) Écrire une fonction de paramètre n, qui affiche le triangle de Pascal correspondant sous la forme ci-dessous (pour n=4 par exemple):

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

EXERCICE 6: Somme des 1/n²

On peut démontrer que $\sum_{i=1}^{n} \frac{1}{i^2}$ tend vers de $\frac{\pi^2}{6}$ quand n tend vers +∞. On va le vérifier. Écrire un programme avec une fonction :

- `saisie` qui demande à l'utilisateur de saisir un entier n>0 (on redemande la saisie jusqu'à ce qu'elle soit correcte) et qui renvoie cet entier,
- `afficherTerme`, de paramètre un entier n, qui affiche le résultat de $\sum_{i=1}^{n} \frac{1}{i^2}$ ainsi que la valeur de $\frac{\pi^2}{6}$.

Dans le corps principal, on appelle les fonctions de façon adéquate (par exemple si l'utilisateur saisit n=1000, on obtient une somme=1.643934566... et $\frac{\pi^2}{6}$ =1.644934066...).

EXERCICE 7: Somme deux dés

1) Écrire une fonction `saisieSomme`, sans paramètre, qui permet de saisir un entier entre 2 et 12 inclus (on redemande jusqu'à ce que la saisie soit correcte), **on retourne la somme saisie.**

2) Ajouter une fonction `sommeDeuxDés`**,** sans paramètre, qui simule le tirage de deux dés: elle affiche le résultat de chaque dé (entier aléatoire entre 1 et 6 inclus), affiche et retourne la somme obtenue (donc entier entre 2 et 12 inclus, on aura besoin du retour dans la question suivante).

Tester en appelant la fonction dans le corps principal. **On devra obtenir la présentation suivante :**

```
dé 1 : 5   dé 2 : 3
somme : 8
```

3) Ajouter une fonction `jouer`**,** sans paramètre, qui appelle les fonctions précédentes et permet à l'utilisateur de jouer. Il gagne s'il a deviné la somme. Exemple d'exécution :

```
Nombre entre 2 et 12 inclus : 10
dé 1 : 6   dé 2 : 5
somme : 11
perdu!
```

EXERCICE 8: Extra-terrestre

a) En utilisant le module **turtle** écrire un programme avec une fonction **antenne**(x,y) qui dessine une antenne partant du point de coordonnées (x,y) de hauteur aléatoire (forme au choix mais se terminant par un cercle rempli), exemples (trois exécutions différentes) :

b) Ajouter une fonction **et**() qui dessine un extra-terrestre (forme au choix) avec au moins quatre antennes (utiliser la fonction antenne précédente), deux bras, un œil et une bouche, exemple :

EXERCICE 9: Somme de cubes

a) On remarque que $153=1^3+5^3+3^3$. Écrire un programme avec une fonction `afficher`, sans paramètre, qui affiche tous les nombres compris entre 100 et 999, vérifiant ce type d'égalité.

b) Ajouter une deuxième fonction `afficher2`, de paramètre un entier n, qui affiche tous les nombres compris entre 100 et n, vérifiant ce type d'égalité (attention si le nombre a 4 chiffres, il faut élever chaque chiffre à la puissance 4).

EXERCICE 10: Approximation de $\sqrt{x}$

Soit x un réel positif, on considère les suites u_n et v_n définies par $u_0=x$, $v_0=1$ et $u_{n+1}=\dfrac{u_n+v_n}{2}$ $v_{n+1}=\dfrac{2u_nv_n}{u_n+v_n}$

a) Écrire une fonction qui calcule et affiche les valeurs successives u_0, v_0 jusqu'à u_n et v_n. Dans le corps principal du programme, on demandera à l'utilisateur un entier naturel n et un réel x>0, on testera si les nombres saisis sont corrects, puis on appellera la fonction et enfin on affichera la valeur de $\sqrt{x}$ calculée par `sqrt(x)`. On obtiendra par exemple:

```
Entrer un entier naturel n : 3
Entrer un réel x > 0 : 2
u_0 = 2.0 v_0 = 1
u_1 = 1.5 v_1 = 1.333333333333333
u_2 = 1.4166666666666665 v_2 = 1.411764705882353
u_3 = 1.4142156862745097 v_3 = 1.41421143847487
racine carrée de 2.0 = 1.4142135623730951
```

b) (facultatif = obligatoire pour ceux qui sont en avance!) Dans un autre programme on demandera à l'utilisateur de saisir un réel x≥0 et un réel d>0, et dans une fonction on calculera u_n et v_n de sorte que $u_n - \sqrt{x} \leq d$ et $\sqrt{x} - v_n \leq d$.
L'affichage sera fait dans la fonction, on obtiendra si l'utilisateur rentre x=2 et d=0.001:

```
valeurs appr. de sqrt(2.0) à 0.001 près:
par défaut: 1.41421143847487
par excès: 1.4142156862745097
obtenues au rang n=3
```

EXERCICE 11: Jeu 1 ou 2

a) Écrire un programme avec une fonction `jeu1`, sans paramètre, qui demande à l'utilisateur un entier, et qui indique de façon aléatoire gagné ou perdu selon que le nombre entré est pair ou impair. On doit obtenir à l'exécution selon les cas:

```
Entrer un entier : 245
Perdu on attendait un nombre pair
Entrer un entier : 245
Gagné on attendait un nombre impair
```

b) Ajouter une fonction `jeu2`, sans paramètre, qui demande à l'utilisateur d'entrer soit +, soit =, soit -, qui génère de façon aléatoire un smiley, et qui indique gagné ou perdu selon que le symbole saisi par l'utilisateur correspond au smiley généré par l'ordinateur ou pas. On doit obtenir à l'exécution selon les cas :

```
Entrer +, - ou = : +
:-) perdu on attendait :-I
```

On écrira une fonction qui affiche le smiley correspondant au caractère : on aura

`:-)` pour `+`
`:-I` pour `=`
`:-(` pour `-`

c) Ajouter une fonction `jeu`, sans paramètre, qui demande à l'utilisateur d'entrer 1 ou 2 selon le jeu souhaité l'utilisateur rejoue autant de fois qu'il le souhaite, si l'utilisateur saisit autre chose le jeu s'arrête.

EXERCICE 12: pH d'une solution aqueuse

En Python, la fonction logarithme népérien se note `log` et la fonction logarithme décimal se note `log10` (accessibles par : **`from`** `math` **`import`** `log, log10`). On rappelle que si x>0, log10(x) = $\frac{\ln(x)}{\ln(10)}$

On rappelle que le pH est donné par la formule pH=-log10([H_3O^+]) où [H_3O^+] est la concentration molaire en ions hydronium, exprimée en mol.l^{-1} et que cette relation n'est correcte que si 10^{-14}mol.l^{-1}<[H_3O^+]<10^{-1}mol.l^{-1} (NB: 10^{-14} s'écrit 1E-14) . Écrire un programme avec une fonction `test_pH`, qui demande la concentration molaire en ions hydronium, qui teste si cette concentration permet le calcul du pH avec la formule ci-dessus, si oui qui calcule le pH (à 10^{-2} près) et qui précise si la solution est acide (pH≤ 6.5), à peu près neutre (6.5 < pH <7.5), ou basique (pH≥7.5).
N.B.: pour une solution d'acide chlorhydrique avec [H_3O^+]=0.011 mol.l^{-1} on trouve un pH=1.96.

EXERCICE 13: Nombre de chiffres

On reprend la fonction `log10`, logarithme décimal, de l'exercice précédent. Remarquer que log10(100)=2, log10(1000)=3, log10(5247)≈3.72, en déduire une formule pour obtenir le nombre de chiffres d'un entier.
a) Écrire une fonction qui renvoie le nombre de chiffres d'un entier >0. Dans le corps principal du programme, on demandera un entier >0 à l'utilisateur et on affichera le nombre de chiffres de cet entier.

b) Rajouter une fonction qui calcule la somme des chiffres d'un entier. Tester.

EXERCICE 14: Suite de Syracuse

La suite de Syracuse est une suite d'entiers définie de la manière suivante :
On part de u_0 entier naturel non nul, puis pour n≥0, si un est pair, on le divise par 2, sinon on le multiplie par 3 et on ajoute 1. On remarque que si on obtient 4, la suite deviendra cyclique (4,2,1,4,2,1,...). Écrire une fonction de paramètre un entier correspondant à u_n qui renvoie le terme suivant u_{n+1}. Le programme demandera la valeur de u_0 à l'utilisateur puis affichera les valeurs des termes successifs jusqu'au 1er 1 trouvé.

EXERCICE 15: Suites

a) Écrire une fonction qui calcule le terme un d'une suite arithmétique de premier terme u_0, de raison r. On aura en premier paramètre u_0, en deuxième paramètre r, en troisième paramètre n. Tester dans un programme en demandant chacun des paramètres à l'utilisateur (par exemple: u_0=-1, r=2, n=3 donne 5).

b) Ajouter une fonction qui calcule le terme un d'une suite géométrique de premier terme u_0, de raison r.

Tester (par exemple: u_0=-1, q=2, n=3 donne -8). Dans le programme, après les saisies de u_0, de la raison et de n, on demandera le type de suite souhaité (a pour arithmétique, g pour géométrique), puis on affichera le résultat ou le message "erreur" si le type ne correspond ni à a, ni à g.