Reprise !

Vendredi 9 (MP) et 16 (MP) septembre 2022*

Buts du TP

Remettre en place quelques automatismes sur :
- l'environnement Python (IDE), les boucles et tests ;
- la façon d'écrire des petites fonctions ;
- les listes : leurs créations et parcours ;
- les graphes (pour les plus rapides).

Exercice 1. *Créer (à un endroit raisonnable, donc pas à la racine de votre répertoire personnel...) un dossier associé à ce TP (et uniquement à lui, donc probablement dans un dossier qui contiendra tous vos TP de seconde année). Y placer une copie du fichier* `cadeau_reprise.py` *récupéré dans le dossier partagé de travail de la classe, ainsi que le très mystérieux fichier* `G2000_1000.gr`.
Lancer Spyder/Pyzo/Idle, sauvegarder immédiatement au bon endroit un nouveau fichier (`tp_reprise.py` *par exemple) ; écrire une commande absurde, de type* `print(5*3)` *dans l'éditeur, sauvegarder et exécuter.*

À cet instant, vous devez avoir deux fichiers d'ouverts dans votre éditeur : votre fichier de travail, et le fichier-cadeau (d'où vous copierez des fragments de code vers votre fichier de travail).

1 Boucles, tests

Pour calculer une somme, on peut faire une boucle, en faisant évoluer la valeur d'une variable (initialement mise à zéro) représentant les sommes partielles.

Exercice 2. *Calculer* $\sum_{k=1}^{100} k^2$ *à l'aide d'une boucle, puis directement avec* `sum`

On veillera à bien calculer la somme demandée, et non $\sum_{k=1}^{99} k^2$...

Exercice 3. *Calculer la somme des entiers* $k \in [\![100, 1000]\!]$ *tels que* $k^2 + 2k + 1$ *est divisible par* 42.
On fera le calcul avec une boucle.

2 Fonctions

Exercice 4. *Définir en Python la fonction* $f : x \mapsto x^2 + 2$. *Valider en évaluant* $f(100)$ *dans la console.*

Exercice 5. *Définir en Python la fonction* $f : x \mapsto \begin{cases} 20\sin(x/5) & \text{si } x < 0 \\ x & \text{si } 0 \leqslant x \leqslant 10 \\ 5x - 40 & \text{si } 10 < x \leqslant 20 \\ -3x + 120 & \text{sinon} \end{cases}$
Représenter son graphe pour vérifier (le code qui suit est dans le fichier donné en cadeau) :

```
import matplotlib.pyplot as pypl
from numpy import arange

les_x = arange(-50, 50, 0.1)
les_y = [f(x) for x in les_x]
pypl.plot(les_x, les_y)

pypl.grid()
pypl.axhline(color='black')
pypl.axvline(color='black')
pypl.savefig('graphe_de_f.pdf')
```

Essayez de comprendre chacune des lignes qui précèdent. Avec le temps il faudrait que vous soyez autonomes pour ce type de graphiques.

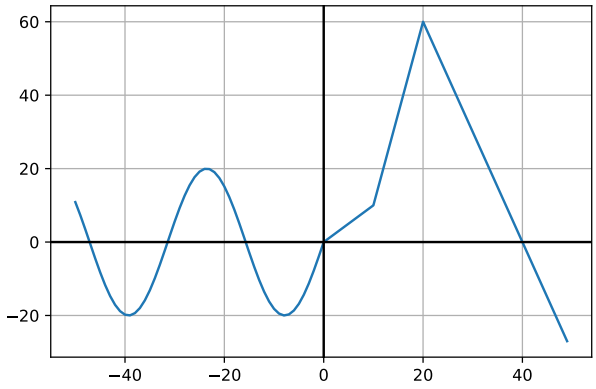

FIGURE 1 – le graphe de f

Pour l'exercice suivant, vous avez le choix entre une version itérative ou récursive.

Exercice 6. *Écrire une fonction calculant la factorielle d'un entier donné en paramètre (et qui n'utilise pas la fonction* `factorial` *de la librairie* `math`*, merci...).*

```
>>> facto(10)
3628800
```

3 Listes (/tableaux)

Tout d'abord, créons rapidement des listes simples. Outre `list(range(...))`, on dispose des compréhensions de listes, d'utilisation typique :

```
[foo(x) for x in whatever if bar(x)]
```

On construit ici la liste constituée des $foo(x)$, pour x décrivant tous les éléments de l'ensemble *whatever* vérifiant la condition $bar(x)$.

Exercice 7. *Créer (en une seule ligne) :*
- *la liste des entiers positifs impairs majorés par* 30 *;*
- *la liste constituée des carrés des entiers précédents ;*
- *celle constituée des éléments de la liste précédente (la deuxième !) égaux à* 1 *modulo* 12
- *(optionnel) celle constituée de* 5 *listes elles-mêmes constituées de* 4 *entiers (il s'agit donc d'un* tableau 5×4*) telle que* $t_4[i][j] = 2^i(2j+1)$ *:*
  `t4 = [[1, 3, 5, 7], [2, 6, 10, 14], ..., [16, 48, 80, 112]]`

Il faut savoir parcourir une liste pour faire des choses simples...

Exercice 8. *Écrire des fonctions prenant en entrée une liste d'entiers (ou flottants), et retournant respectivement :*
- *la somme des éléments de la liste ;*
- *leur produit ;*
- *leur maximum ;*
- *le* premier *indice correspondant à ce maximum.*

Pour les tests, en notant t_1 *la première liste de l'exercice précédent :*

```
>>> somme(t1), produit(t1), maximum([10, 42, 5, 42, 30]), indice_maximum([10, 42, 5, 42, 30])
(225, 6190283353629375L, 42, 1)
```

Maintenant, un parcours double.

Exercice 9. *Écrire une fonction calculant le nombre d'inversions d'une liste (disons t), c'est-à-dire le nombre de couples (i, j) tels que $0 \leqslant i < j < |t|$ et $t[j] < t[i]$.*

```
>>> inversions([1, 2, 3, 0])
3
>>> inversions([4, 3, 2, 1])
6
```

Les deux exercices suivants sont à nouveau optionnels (à reporter à la fin ou après le TP) :

Exercice 10. *Avec des* `append` *successifs, construire la liste t de longueur 11 telle que $t_0 = 1$, et pour tout $j \in [\![1, 10]\!]$, $t_j = \sum_{i=0}^{j-1} (i+1) t_{j-1-i}$.*

```
>>> t
[1, 1, 3, 8, 21, 55, 144, 377, 987, 2584, 6765]
```

Exercice 11. *Écrire une fonction calculant le triangle de Pascal à un ordre donné, en ne faisant que des additions :*

```
>>> pascal(4)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
```

4 Des graphes

On s'intéresse ici à des graphes non valués symétriques (bref : des sommets reliés avec des traits!!!). On les représente par listes d'adjacence : les arêtes sont indexées de 0 à $n-1$, et les sommets reliés au sommets s par une arête sont dans une liste placée dans `G[s]`, où `G` est donc une liste de n listes. Pour comprendre le formalisme, on pourra par exemple représenter le graphe de liste d'adjacence :

```
G0 = [[5], [2, 4], [1], [], [1], [0]]
```

Exercice 12. *Écrire une fonction calculant la composante connexe d'un sommet donné dans un graphe.*

Je ne donne pas de directive stricte... mais je suggère de ne pas utiliser de structures autres que des listes, et vous autorise à utiliser les tests d'appartenance à des listes : `x in L`

```
>>> composante(0, G0)
[0, 5]

>>> [composante(s, G0) for s in range(6)]
[[0, 5], [1, 2, 4], [2, 1, 4], [3], [4, 1, 2], [5, 0]]
```

Pour quelques efforts supplémentaires on doit pouvoir obtenir l'ensemble des composantes connexes...

Exercice 13. *Écrire un programme calculant la liste des composantes connexes d'un graphe.*

```
>>> composantes(G0)
[[0, 5], [1, 2, 4], [3]]
```

Vous disposez d'un programme permettant de lire un graphe écrit dans un fichier : c'est cadeau !

Exercice 14. *Combien de composantes connexes possède le graphe écrit dans le fichier* `G1000_2000.gr` *? Et quelle est la taille de celle contenant le sommet 0 ?*

Vous pouvez jouer à créer (avec le fichier fourni en cadeau) des graphes à 1000 sommets et « un certain nombre de milliers d'arêtes ». Un phénomène bien connu dit que quand on ajoute des arêtes successivement dans un graphe initialement sans arête, les composantes connexes s'agglutinent pour devenir une composante géante à coté de plein de petites composantes... jusqu'à ce que la grosse ai mangé les petites, ce qui se passe grosso modo vers $\dfrac{n \ln(n)}{2}$ arêtes.