Lycée Pierre-Gilles de Gennes 2024-2025

TP Python – 1

Révisions : suites et fonctions

L'objet de ce TP est d'une part de revenir sur les fondamentaux du langage Python (boucles, fonctions) et quelques algorithmes élémentaires (calculs de sommes, dichotomie) et, d'autre part d'illustrer les chapitres de révision sur les suites et fonctions.

1 Pour bien débuter

Pour démarrer, créez un *répertoire/dossier de travail* `TP 1` où vous voulez et repérez l'emplacement de ce répertoire.
Ouvrez *Spyder* et dans l'éditeur, tapez (on peut aussi faire un copier-coller du .pdf vers l'éditeur, à condition de remettre les tabulations/indentations correctes, de gérer correctement les espacements et quelques autres détails) :

```
#-*- coding: utf-8 -*-
"""
rev_python.py :
        algorithmes de somme,
        definition de fonction,

"""
#Zone (optionnelle) d'importation des modules externes

#Zone de définition des constantes et fonctions communes

# %% Cellule 1 Script principal

# %% Cellule 2 Script principal
```

et sauver ce fichier sous le nom `rev_pyhton.py` dans le répertoire de travail `TD 1`, ce qui nomme le script présent dans l'éditeur.
On y met la `docstring` (entre `"""`) décrivant ce que fait le script et des commentaires (débutant par `#`) signalant les différentes zones du script.

Il est important de systématiquement documenter (`"""`) et commenter les scripts (`#`) ! !
Autant prendre de bonnes habitudes !

2 Fonctions

Quand on écrit un programme sous Python, il est fréquent de devoir / vouloir utiliser des fonctions. Généralement, elles peuvent avoir 0, 1 ou plusieurs arguments (certains optionnels) nécessaires à son fonctionnement.
Par exemple, dans la définition de la fonction `nom_fonction` ci-dessous, les arguments `arg_1`,...,`arg_n` sont nécessaires et l'argument `arg_option1` est optionnel et vaudra 10 par défaut :

```
def ma_fonction(arg_1,...,arg_n, arg_option1 = 10):
    ...
    return resultat_1,...,resultat_p
```

3 Algorithmes à accumulateur

Pour calculer la somme des éléments d'une liste `L`, on se base sur une boucle et un « accumulateur » `S`. L'algorithme se décrit en mots de la façon suivante :

1. Initialiser l'accumulateur à 0
2. Boucler sur les éléments de la liste `L` et, dans la boucle, ajouter l'élément courant de la liste à l'accumulateur `S`.

Des codes Python typiques pour cela sont donc

```
#Liste L de nombres, préexiste

S=0 #Accumulateur pour la somme
for i in range(len(L)): #indices pour L
    S+L[i] #ou S+=L[i]
#A ce point S= la somme des éléments de L
```

```
S=0
for elt in L: #Possibilité propre à Python
    S+=elt
```

```
S=0
i=0 #indices pour L
while i< len(L):
    S+=L[i]
    i+=1 #incrementation de l'indice
```

```
S=0
i=len(L) #indices pour L
while i>0: #En décrémentant
    i=i-1 #décrementation de l'indice
    S+=L[i]
```

Travail demandé

1. (a) Écrire, dans le script `rev_python.py`, dans la zone des fonctions communes, une fonction `Somme(L)` prenant en argument une liste Python `L` de nombres et retournant la somme.

   (b) Écrire dans la zone de script principal les lignes suivantes

```
#%% Cellule 1 du script principal
liste_entiers=[2,1,4,3,6,7,5]
print("Test Somme :",Somme(liste_entiers))
```

(c) Exécuter le script, le résultat est-il correct ?

2. (a) Écrire, dans le script `rev_python.py`, dans la zone des fonctions communes, une fonction

`Denombrement(L,mini=1,maxi=5)`

prenant en argument une liste Python `L` de nombres, deux bornes `mini` et `maxi`, passées en *arguments nommés avec valeurs par défaut* et retournant le nombre d'éléments de la liste `L` qui sont compris (au sens strict) entre `mini` et `maxi`.

(b) Écrire dans une nouvelle cellule du script principal à la suite des lignes de code déja écrites

```
print("Test Denombrement : ",Denombrement(liste_entiers))
```

(c) Exécuter le script, le résultat est-il correct ?

3. (a) Écrire, dans le script `rev_python.py`, dans la zone des fonctions communes, une fonction

`SommeFiltree(L,mini=1,maxi=5)`

prenant en argument une liste Python `L` de nombres, deux bornes `mini` et `maxi`, passées en arguments nommés avec valeurs par défaut et retournant la somme des éléments de la liste `L` qui sont compris (au sens strict) entre `mini` et `maxi`.

(b) Écrire dans une nouvelle cellule

```
print("Test SommeFiltree : ",SommeFiltree(liste_entiers))
```

(c) Exécuter le script, le résultat est-il correct ?

4. (a) Écrire, dans le script `rev_python.py`, dans la zone des fonctions communes, une fonction `Produit(L)` prenant en argument une liste Python `L` de nombres et retournant le produit des éléments de la liste `L`.

(b) Écrire dans une nouvelle cellule

```
print("Test produit : ",Produit(liste_entiers))
```

(c) Exécuter le script, le résultat est-il correct ?

3.1 Minima, maxima, minimant, maximant

Pour calculer le minimum des éléments d'une liste de nombre `L`, on se base là encore sur une boucle et un « accumulateur » `m`. L'algorithme se décrit en mots de la façon suivante :

1. initialiser `m` l'accumulateur à l'un des éléments de la liste ;
2. boucler sur les éléments de la liste `L` et, dans la boucle, tester si l'élément courant de la liste est inférieur à l'accumulateur `m`, si c'est le cas, affecter cette valeur à l'accumulateur.

Des codes Python typiques pour cela sont donc

```
#La liste L est composée de nombres, préexiste
m=L[0]  #Accumulateur pour le minimum
for i in range(len(L)): #indices pour L
    if L[i] < m:
        m=L[i]
#A ce point m = minimum des éléments de L
```

```
m=L[-1] #Accumulateur pour le minimum
i=len(L)#indices pour L
while i>0:
    i=i-1#On décrémente l'indice
    if L[i] <= m:
        m=L[i]
```

Travail demandé

1. (a) Écrire, dans le script `rev_python.py`, dans la zone des fonctions communes, une fonction `Minimum(L)` retournant le minimum des éléments de la liste numériqueL.
   (b) Écrire de même une fonction `Maximum(L)` retournant le maximum de `L`.
   (c) Écrire dans une nouvelle cellule

```
print("Test Minimum : ",Minimum(liste_entiers))
print("Test Maximum : ",Maximum(liste_entiers))
```

   (d) Exécuter le script, le résultat est-il correct ?
2. (a) Écrire, dans le script `rev_python.py`, dans la zone des fonctions communes, une fonction `IndiceMinimum(L)` retournant le *premier* indice d'un élément minimum de la liste numériqueL.
   (b) Écrire de même une fonction `IndiceMaximum(L)` retournant le *dernier* indice d'un élément maximum de la liste numériqueL.
   (c) Écrire dans une nouvelle cellule

```
print("Test IndiceMinimum : ",IndiceMinimum(liste_entiers))
print("Test IndiceMaximum : ",IndiceMaximum(liste_entiers))
```

   (d) Exécuter le script, le résultat est-il correct ?

4 Étude d'une suite récurrente

On considère la fonction f définie sur $\mathbb{R}$ par : $\forall x \in \mathbb{R}, \quad f(x) = (x-1)^2$.
On définit une suite (u_n) en posant : $u_0 \in \mathbb{R}$ et $\forall n \in \mathbb{N}, \quad u_{n+1} = f(u_n)$.

1. (a) Créer un nouveau fichier sous le nom `suite_recurrente.py` dans le répertoire de travail `TD 1` et y mettre une description en `docstring`.
   Dans la zone d'importation des modules externes recopier :

```
import numpy as np
import matplotlib.pyplot as plt
```

   (b) Dans la zone de définition des fonctions communes définir des fonctions `f`, `fof` qui prennent en argument un réel x et renvoie respectivement $f(x)$ et $f \circ f(x)$.
2. (a) Dans une nouvelle cellule, recopier et compléter le programme suivant pour qu'il affiche le graphe de f sur $[0, 3]$:

```
1 X = np.linspace(0,3,100) #vecteur de 100 points régulièrement espacés
  entre 0 et 3
2 Y = ...   # vecteur contenant les images des éléments de X
3 plt.plot(X,Y)
4 plt.show()
```

(b) En déduire graphiquement les variations de f.

(c) Ajouter une ligne au script ci-dessus pour qu'il affiche également la droite d'équation $y = x$ sur le même graphique.

Qu'en déduit-on sur les points fixes de f ?

3. On admet que f admet deux points fixes $0 < a < 1 < b < 3$.

(a) Dans une nouvelle cellule, écrire une fonction `dicho(eps)` qui prend en argument un réel `esp`> 0 et détermine, par un algorithme de dichotomie, une valeur approchée de a à `eps` près.

(b) Déterminer la valeur exacte de a et comparer avec `dicho(10**(-3))`.

4. Dans la zone de définition des fonctions communes écrire une fonction `suite(n,u0)` qui prend en argument un entier n et un réel `u0` et qui renvoie la liste $[u_0, \dots, u_n]$.

5. Dans cette question on suppose que $u_0 = 3$.

(a) Dans une nouvelle cellule, recopier et compléter le programme suivant pour qu'il affiche les dix premiers termes de la suite $(\ln(u_n))_n$:

```
1 N = range(10)
2 U = suite(...,3.0)
3 U = [ .... for u in U]
4 plt.plot(N,U,'o')
5 plt.show()
```

Que peut-on conjecturer sur la nature de (u_n) dans ce cas ?

(b) Démontrer cette conjecture (on pourra commencer par montrer que $[3, +\infty[$ est stable par f puis étudier la monotonie de la suite).

6. Dans cette question on suppose que $u_0 \in [0, 1]$ (pour les simulations numériques, on prendra $u_0 = 0.5$).

(a) Dans une nouvelle cellule, écrire le programme permettant d'afficher les 100 premiers termes de la suite $(u_n)_n$.

Que peut-on conjecturer sur la nature de (u_{2n}) ? de (u_{2n+1}) ? de (u_n) ?

(b) Tracer sur le même graphique les fonctions f, $f \circ f$ et la droite d'équation $y = x$ entre $[0, 3]$. En déduire les points fixes de $f \circ f$ et ses variations.

(c) Retrouver ces résultats par le calcul.

(d) Montrer que $[0, 1]$ est stable par $f \circ f$ et en déduire que (u_{2n}) et (u_{2n+1}) sont monotones.

(e) Déterminer la limite de (u_{2n}) et (u_{2n+1}) et comparer avec les résultats de la question 6.(a).

7. Étudier le cas $u_0 \in]1, b[$. On pourra étudier numériquement quelques cas ($b = \dfrac{3+\sqrt{5}}{2}$).