Séance 3

Variables et sommes en Python

Séries numériques et TP1 d'intégration

Dans cette séance, nous allons voir comment Python associe des noms à des objets, puis utiliser une boucle pour construire une somme terme après terme. Les séries nous permettront de distinguer un résultat numérique d'une preuve mathématique. Nous réutiliserons ensuite ces outils pour calculer une intégrale à la main, au tableur, puis en Python, en vérifiant les résultats à chaque étape.

I. Affectation et mémoire

Les variables nous permettent de donner des noms aux valeurs utilisées par le programme. En Python, un nom fait référence à un objet en mémoire. Cet objet peut être un entier, un flottant, une chaîne de caractères, une liste, un dictionnaire, etc.

I.1. Affectation de variables

L'affectation d'une variable signifie attribuer une valeur à une variable. En Python, cela se fait à l'aide de l'opérateur =.

```
# Affectation de valeurs
x = 10          # Entier
y = 3.14        # Flottant
nom = "Arthur"   # Chaîne de caractères
is active = True  # Booléen
# Affectation de collections
liste = [1, 2, 3]
dictionnaire = {"clé": "valeur"}
a, b, c = 1, 2, 3  # Affectation multiple
x = y = z = 0  # Toutes les variables reçoivent la même valeur
```

En Python, les variables n'ont pas de type fixe. Une variable peut changer de type au cours de l'exécution du programme.

```
x = 10       # Entier
x = 3.14     # Flottant
x = "Arthur"  # Chaîne de caractères
```

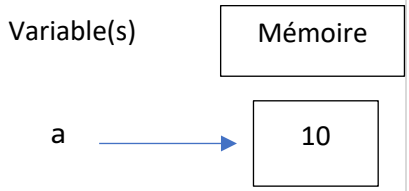
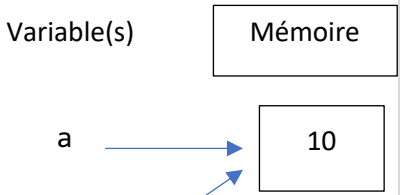
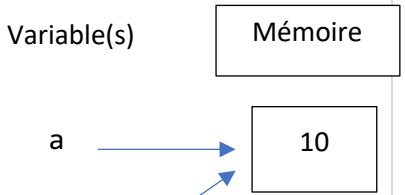
I.2. Gestion de la mémoire

En Python, la gestion de la mémoire est automatique grâce au **ramasse-miettes** (garbage collector). Cependant, il est utile de comprendre comment les variables et les objets sont stockés en mémoire.

Lorsque nous affectons une valeur à une variable, nous associons ce nom à un objet. L'objet peut être nouvellement créé ou déjà exister ; une affectation ne signifie donc pas forcément une copie.

```
a = 10
b = a
b = 20
# Ici, 'a' est toujours 10, 'b' est 20
```

Au départ, a et b font référence au même objet, l'entier 10. Lorsque nous écrivons b = 20, nous associons b à l'objet qui représente 20. L'association de a ne change pas : a vaut toujours 10. Ce mécanisme sera à distinguer de la modification d'un objet mutable, comme une liste.

Ligne 1	Ligne 2	Ligne 3
		

- **Types immuables** : Les objets de ces types ne peuvent pas être modifiés après leur création. Les types immuables incluent les entiers, flottants, chaînes de caractères, tuples, etc.
- **Types mutables** : Les objets de ces types peuvent être modifiés après leur création. Les types mutables incluent les listes, dictionnaires, ensembles, etc.

C'est un point important : une affectation entre deux noms ne crée pas une copie indépendante d'une liste. Voyons cela sur les exemples suivants.

Voyons cela :

Types immuables	Types mutables

`a = 1`
`b = a`
`a = 2`
	`l1 = [1, 2, 3]`
	`l2 = l1`
	`l1[0] = 7`

On aura la variable a qui vaudra 2 et la variable b vaudra 1	La modification que nous faisons de la liste l1 en ligne 3, va aussi affecter la liste l2 à cause de la ligne 2 et du type mutable des listes. Donc, vigilance !!!

Ramasse-miettes (Garbage Collector)

Python gère automatiquement la mémoire. Dans Python, l'implémentation la plus courante, repose principalement sur le comptage des références, complété par un ramasse-miettes qui repère certains cycles de références. Nous n'avons pas à libérer nous-mêmes les objets utilisés dans ces exercices.

II. Sommer une série / boucles

Une instruction du type S = S + x permet d'ajouter une valeur à une somme déjà calculée. En répétant cette instruction, nous construisons la somme progressivement.

Une suite d'instructions telles que :

```
x = 1
S = 0
S = S + x
S = S + x
S = S + x
```

va avoir l'effet suivant:

Nous associons d'abord x à la valeur 1 et S à la valeur 0. À chaque instruction S = S + x, Python calcule la somme des valeurs actuelles de S et de x, puis associe S au résultat. Ici, x reste égal à 1 : nous ajoutons donc 1 à chaque étape.

On a ainsi S = 1 à l'issue de cette première étape.

On recommence l'opération précédente et on obtient S = 2, et on recommence une dernière fois pour arriver à S = 3.

Afin d'éviter d'écrire l'instruction S = S + x 3 fois, on utilise une boucle for.

Initialisation :

```
x = 1, S = 0
Pour i allant de 1 à 3 faire
    S = S +x
Fin Pour
Afficher S
```

Une instruction de type boucle for introduit un indice, que l'on a noté ici i.

Cet indice va prendre successivement les valeurs de 1 à 3 à chaque nouvelle boucle.

Remarque :

Dans Python, pour l'instruction for i in range(1,3):, i prendra les valeurs 1 et 2 et non 3…

Essayons de calculer (avec la précision numérique permise par la machine), la somme de la série géométrique :

$$S = \sum_{k=1}^{\infty} (\frac{1}{4})^k$$

Exercice 1:

Calculer (à la main),

$$S = \sum_{k=1}^{\infty} (\frac{1}{4})^k$$

Nous allons maintenant voir comment retrouver cela avec la machine en retenant les 2 principes de base du calcul scientifique :

1. Un résultat calculé sur ordinateur peut être approché : il faut connaître et contrôler les erreurs possibles.

2. On teste un algorithme pour un problème dont on connait la solution.

Le premier point a été abordé durant la séance 1.

Pour le second, c'est un point essentiel qui nous permet de relever les erreurs de programmation qui conduiraient à un résultat inexact. Et c'est une habitude qu'il faudra garder.

Pour évaluer approximativement S, nous allons introduire la suite géométrique $(y_k)_{k\in\mathbb{N}}$, telle que :

$$y_0 = 1 \; et \; \forall k \in \mathbb{N}, y_{k+1} = \frac{1}{4} y_k$$

On a ainsi,

$$S_n = \sum_{k=1}^{n} y_k = S_{n-1} + y_n$$

On initialise l'algorithme avec : $\boldsymbol{y = 1, S = 0}$.

Puis, on effectue « un certain nombre de fois » l'opération qui consiste à calculer le nouveau terme y de la suite géométrique : $\boldsymbol{y = \frac{y}{4}}$ et à ajouter le résultat obtenu à la somme partielle : $\boldsymbol{S = S + y}$.

On obtient ce script en Python :

```
y=1
S=0
for i in range(1, 26):
    y=y/4
    S=S+y
print(S)
```

On obtient ainsi une bonne approximation de $\frac{1}{3}$.

Remarque :

L'ordre des instructions est très important, il ne faut pas ici intervertir les lignes y=y/4 et S=S+y, au risque de trouver un mauvais résultat.

III. Séries convergentes/divergentes

Une série est une somme de termes dont on étudie les sommes partielles. Nous pouvons calculer ces sommes avec un ordinateur et observer leur évolution. Cette observation aide à formuler une conjecture ; elle ne suffit pas à démontrer la convergence ou la divergence.

Définitions

- **Série convergente** : Une série $\sum_{n=1}^{\infty} a_n$ est convergente si la suite de ses sommes partielles $S_N = \sum_{n=1}^{N} a_n$ tend vers une limite finie L lorsque $N \to \infty$.
- **Série divergente** : Une série $\sum_{n=1}^{\infty} a_n$ est divergente si la suite de ses sommes partielles $S_N = \sum_{n=1}^{N} a_n$ ne tend pas vers une limite finie lorsque $N \to \infty$.

Le calcul numérique permet de formuler une conjecture sur la convergence. Après un certain nombre d'étapes, les sommes calculées peuvent sembler se stabiliser, voire devenir identiques en machine. Cela peut être un effet de l'arrondi ou de l'affichage : nous ne disposons pas encore d'une preuve de convergence.

Pour conclure rigoureusement à la convergence ou à la divergence, nous devons utiliser une justification mathématique. La série harmonique en donne un bon exemple.

Considérons la série harmonique S_N suivante :

$$S_N = \sum_{k=1}^{N} \frac{1}{k}\ , N \geq 1$$

Le terme général de la série, $u_k = \frac{1}{k}$ tend vers 0 lorsque k tend vers l'infini.

L'expérience numérique indique :

$S_{10} \approx 2{,}929 \quad S_{100} \approx 5{,}187 \quad S_{1000} \approx 7{,}485 \quad S_{10000} \approx 9{,}788$

Soit une croissance lente.

Comment prouver que $S_N \longrightarrow \infty$ quand $N \longrightarrow \infty$?

On a plusieurs façons de le montrer :

Méthode 1 : Comparaison série – intégrale

Nous posons $f(t) = \frac{1}{t}$ pour $t > 0$, de sorte que $u_n = f(n)$ pour $n \geq 1$.

La fonction f est décroissante sur $]0; +\infty[$. Pour tout entier $n \geq 1$, nous pouvons donc écrire :

$$\forall\, t \in [n; n+1], f(n+1) \leq f(t) \leq f(n)$$

$$\int_n^{n+1} f(n+1)\, dt \leq \int_n^{n+1} f(t)\, dt \leq \int_n^{n+1} f(n)\, dt$$

car $f(n)$ et $f(n+1)$ sont des constantes

$$\Leftrightarrow f(n+1) \int_n^{n+1} dt \leq \int_n^{n+1} f(t)\, dt \leq f(n) \int_n^{n+1} dt$$

$$\Leftrightarrow f(n+1).[\mathrm{t}]_n^{n+1} \leq \int_n^{n+1} f(t)\, dt \leq f(n).[\mathrm{t}]_n^{n+1}$$

$$\Leftrightarrow f(n+1).(n+1-n) \leq \int_n^{n+1} f(t)\, dt \leq f(n).(n+1-n)$$

$$\Leftrightarrow f(n+1) \leq \int_n^{n+1} f(t)\, dt \leq f(n)$$

Pour majorer $f(n)$ par une intégrale à gauche, nous devons prendre $n \geq 2$, afin de rester sur un intervalle où la fonction inverse est intégrable :

$$f(n) \leq \int_{n-1}^{n} f(t)\, dt\,, pour\ n \geq 2$$

En sommant la minoration précédente pour n de 1 à N, nous obtenons :

$$\int_1^{N+1} f(t)\, dt \leq S_N$$

Pour la majoration, nous isolons le premier terme, qui vaut 1, puis sommons de n = 2 à N :

$$S_N = 1 + \sum_{n=2}^{N} f(n) \leq 1 + \int_1^N f(t)\, dt$$

Nous n'avons ainsi pas besoin d'intégrer la fonction inverse à partir de zéro. Pour $N = 1$, les mêmes bornes se vérifient directement. Pour tout $N \geq 1$:

$$ln(N+1) \leq S_N \leq 1 + ln(N)$$

En effet, une primitive de $\frac{1}{t}$ sur $]0; +\infty[$ est $ln(t)$.

Quand N tend vers l'infini, $\ln(N+1)$ et $\ln(N)$ tendent vers l'infini. Notre série est donc divergente.

Méthode 2 : $S_{2N} - S_N$

On a :$S_{2N} - S_N = \sum_{k=N+1}^{2N} \frac{1}{k} \geq \frac{1}{2N} + \frac{1}{2N} + \cdots + \frac{1}{2N} = N \times \frac{1}{2N} = \frac{1}{2}$

$$S_2 - S_1 \geq \frac{1}{2}$$

$$S_4 - S_2 \geq \frac{1}{2}$$

$$\vdots$$

$$S_{2^k} - S_{2^k-1} \geq \frac{1}{2}$$

En sommant, on obtient :

$$S_{2^k} - S_1 \geq \frac{k}{2} \Leftrightarrow S_{2^k} \geq \frac{k}{2} + S_1 \Leftrightarrow S_{2^k} \geq \frac{k}{2} + 1$$

Ainsi, si $k \longrightarrow \infty$, $S_{2^k} \longrightarrow \infty$.

IV. TP1 Calcul intégral à la main

Nous allons calculer une même intégrale à la main, au tableur, puis en Python. Nous commencerons par une fonction dont nous connaissons une primitive : cela permettra de vérifier chaque étape avant de passer à une autre intégrale. Simpson sera proposé en bonus.

A. Calculer et contrôler avec quatre intervalles

Nous prenons $f(x) = x^2$ sur $[0\,;\,1]$ et $n = 4$ sous-intervalles de même largeur. Dans tout le TP, « rectangles » désigne les rectangles à gauche. Nous noterons R_n leur somme et T_n celle des trapèzes.

1. Calculer la valeur exacte $I = \int_0^1 x^2\,dx$, en utilisant une primitive.

.....

.....

2. Calculer le pas h = $\frac{b-a}{n}$, puis compléter les cinq points du tableau.

i	$x_i = a + ih$	$f(x_i)$
0		
1		
2		
3		
4		

3. Faire un croquis des quatre rectangles à gauche. Pour chaque intervalle $[x_i\,;\,x_{i+1}]$, donner la contribution $h\,f(x_i)$, puis calculer R_4. Attention, le point $x_4 = 1$ n'est pas une hauteur de rectangle à gauche !

.....

.....

.....

4. Pour ces quatre mêmes intervalles, calculer les contributions $\frac{h(f(x_i)+f(x_{i+1}))}{2}$, puis leur somme T_4. Comparer R_4 et T_4 à I et calculer les erreurs absolues $|R_4 - I|$ et $|T_4 - I|$. Pourquoi les rectangles à gauche sous-estiment-ils ici l'intégrale ?

.....

.....

.....

TP1 Construire le calcul au tableur

B. Une ligne par point de subdivision

Créer une feuille « Carré ». Placer a en H1, b en I1, n en J1 et h en K1 ; saisir 0, 1 et 4 dans H2:J2. Calculer le pas en K2. Les paramètres seront ainsi regroupés au même endroit.

A1	**B1**	**C1**	**D1**	**E1**
i	x_i	$f(x_i)$	Rectangle à gauche	Trapèze

5. En A2, saisir 0. Construire les colonnes A, B et C jusqu'au point i = 4, situé à la ligne 6. Écrire les formules de K2, B2 et C2, puis celles de A3, B3 et C3. Utiliser des références absolues pour les paramètres ; ne pas saisir les abscisses à la main.

Cellule	**Formule proposée**
K2	
B2	
C2	
A3	
B3	
C3	

6. En D2, calculer la contribution du premier rectangle. En E2, calculer celle du premier trapèze, à l'aide des valeurs de C2 et C3. Recopier ces deux formules jusqu'à la ligne 5 seulement : quatre intervalles utilisent cinq points. Laisser D6 et E6 vides.

.....

.....

7. Dans une zone libre, additionner D2:D5 puis E2:E5 avec SOMME. Comparer aux calculs à la main et calculer les deux erreurs absolues avec ABS. Conserver les valeurs internes ; un arrondi d'affichage ne modifie pas les calculs.

8. Refaire les calculs pour $n = 8$, puis n = 20. Conserver les trois essais dans des feuilles distinctes. En changeant n, adapter le nombre de lignes et les plages des sommes : modifier le paramètre seul ne crée pas les lignes manquantes !

n	R_n	**Erreur rectangles**	T_n	**Erreur trapèzes**
4				
8				
20				

9. Préparons le programme : quelles valeurs faut-il initialiser ? Quelles instructions seront répétées sur chaque intervalle ? Que faut-il ajouter à la somme ? Expliquer à quoi correspondent, dans une boucle, les lignes des colonnes D et E.

.....

TP1 Programmer les deux méthodes

C. Passer des cellules aux variables

Nous allons construire deux fonctions générales : elles recevront la fonction à intégrer, les bornes a et b et le nombre n d'intervalles. Nous supposons a < b et n entier ≥ 1. Les calculs doivent utiliser des boucles ; aucun appel à une fonction d'intégration prête à l'emploi dans vos deux méthodes.

10. Écrire integrale_rectangles(f, a, b, n). Calculer le pas, initialiser une somme, puis ajouter la contribution de chacun des n rectangles à gauche. Renvoyer le résultat avec return.

11. Écrire integrale_trapezes(f, a, b, n) sur le même principe. Pour chaque intervalle, il faut utiliser les deux valeurs aux extrémités. Vous pouvez aussi utiliser la formule globale des trapèzes vue dans le cours.

Rappel : range(n) parcourt 0, 1, …, $n-1$. Il y a n intervalles, mais $n+1$ points. Les fonctions doivent renvoyer une valeur ; les print des tests sont placés à l'extérieur des fonctions.

Définir une fonction puis la transmettre

```
def carre(x):
    return x**2

# Appels à placer après la définition de vos méthodes :
R = integrale rectangles(carre, 0, 1, 4)
T = integrale trapezes(carre, 0, 1, 4)
print(R, T)
```

Dans l'appel, carre est transmis sans parenthèses : notre méthode pourra l'évaluer aux abscisses nécessaires. Écrire carre(0) transmettrait seulement le nombre obtenu en 0, ce qui n'est pas le but.

12. Tester les deux fonctions avec $n = 4$, 8 et 20. Comparer les résultats avec les feuilles du tableur. Vérifier notamment le cas $n = 4$ avant de poursuivre. Si les résultats diffèrent, chercher d'abord une erreur de borne de boucle ou un terme ajouté en trop.

.....

.....

.....

13. Modifier la fonction test pour intégrer une fonction constante $f(x) = 2$ sur $[0; 3]$. Quelle valeur attendez-vous ? Tester avec $n = 1$ puis $n = 10$. Que vérifie ce test ?

.....

.....

.....

Commenter les étapes utiles : rôle des paramètres, pas, somme et boucle. Si un programme ne fonctionne pas, conserver votre essai et expliquer le blocage. Les calculs précédents sont là pour vous aider à le repérer.

TP1 Réutiliser les méthodes et comparer

D Une intégrale liée à la loi normale

$$J = \int_0^1 exp\left(-\frac{x^2}{2}\right) dx$$

Cette intégrale intervient dans des calculs liés à la loi normale. La fonction à intégrer est g(x) = $exp\left(-\frac{x^2}{2}\right)$. Elle n'est pas, à elle seule, la densité normalisée de la loi normale.

```
from math import exp

def g(x):
    return exp(-x**2 / 2)

J_ref = 0.8556243918921488
```

J_ref est une référence numérique très précise, et non une valeur exacte. Pour ce TP, nous calculerons les écarts ε = |approximation − J_ref|. Ces écarts permettent la comparaison au seuil 10^{-4} ; ils ne constituent pas une preuve générale de précision pour d'autres fonctions.

14. Dans une nouvelle feuille du tableur, remplacer x² par $exp\left(-\frac{x^2}{2}\right)$. En C2, utiliser =EXP(-(B2^2)/2). Calculer R_n et T_n pour $n = 4$, 8 et 20. Réutiliser ensuite vos fonctions Python avec g et vérifier l'accord avec le tableur.

15. Avec Python, tester $n = 4$, 10, 20, 100 et 1000. Présenter les approximations et les écarts à J_ref dans un tableau. Quelle méthode semble atteindre le seuil avec moins d'intervalles ?

16. Pour chaque méthode, chercher le plus petit entier $n \geq 1$ tel que l'écart à J_ref soit ≤ 10^{-4} : tester les entiers successifs et arrêter au premier succès. Relever n et les écarts pour n et $n - 1$ (si $n > 1$). Si vous testez seulement des puissances de 2, vous trouvez le premier succès parmi vos essais, pas nécessairement le plus petit entier.

Bonus La méthode de Simpson

17. Écrire integrale_simpson(f, a, b, n) avec la formule composite du cours. Ici, n doit être pair et au moins égal à 2. Vérifier sur x² avec $n = 4$: on attend $\frac{1}{3}$, à l'arrondi machine près. Tester ensuite g, puis chercher le premier n pair donnant un écart à J_ref ≤ 10^{-4}. Comparer avec les deux autres méthodes et conclure pour cet exemple.

Référence facultative avec SciPy

Si SciPy est disponible, vous pouvez contrôler J_ref avec le code fourni. Il ne remplace pas vos méthodes. quad renvoie une approximation et une estimation de son erreur absolue ; cette estimation n'est pas l'erreur de votre méthode des rectangles ou des trapèzes.

```
from scipy.integrate import quad
J ref, erreur estimee = quad(g, 0, 1)
print(J_ref, erreur_estimee)
```

Documentation : https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html

À faire

Un classeur contenant les essais conservés, un fichier Python avec les fonctions et les appels de test, puis une courte conclusion sur les résultats et les écarts. Nommer les fichiers avec votre nom et votre prénom. Distinguer clairement le travail demandé et le bonus Simpson.

Éléments de correction de l'exercice sur la série géométrique

Cette correction concerne l'exercice 1 de la partie II, et non le TP1 d'intégration. Chercher d'abord par vous-même, puis comparer les étapes de votre calcul.

Pour $q \neq 1$, la somme de q^k de k = 0 à n vaut $\frac{1-q^{n+1}}{1-q}$. Dans notre exercice, la somme commence à k = 1 : il faut donc retirer le terme de rang 0, qui vaut 1.

$S_n = \left(\frac{1}{3}\right)(1 - 4^{-n})$, pour $n \geq 1$.

Lorsque n tend vers l'infini, 4^{-n} tend vers 0. La somme de la série est donc $\frac{1}{3}$. Dans l'algorithme, l'initialisation y = 1 permet de calculer le premier terme y/4 ; elle ne signifie pas que le terme 1 est inclus dans la somme.