Informatique Appliquée au Calcul Scientifique 1

Séance 3

Affection et mémoire

Calcul de série

Table des matières

I. Affectation et mémoire

Les variables sont des conteneurs utilisés pour stocker des données en mémoire. En Python, les variables peuvent stocker différents types de données, tels que des entiers, des flottants, des chaînes de caractères, des listes, des dictionnaires, 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
```

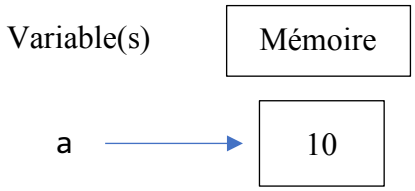
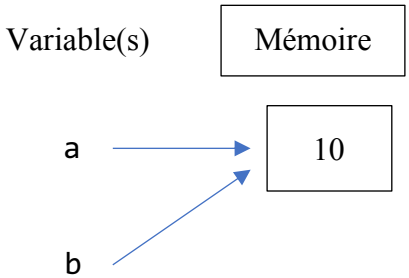
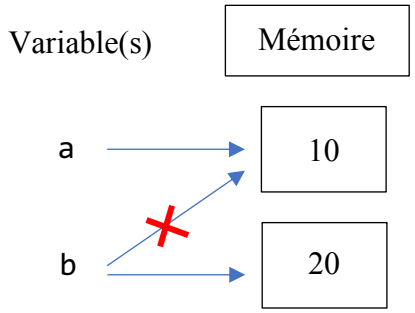
I.2. Gestion de la mémoire

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 vous assignez une valeur à une variable, vous créez un objet en mémoire et la variable fait référence à cet objet.

```
1. a = 10
2. b = a
3. b = 20
# Ici, 'a' est toujours 10, 'b' est 20
```

Dans cet exemple, `a` et `b` sont initialement des références au même objet (10). Cependant, lorsqu'une nouvelle valeur (20) est affectée à `b`, un nouvel objet est créé en mémoire et `b` fait référence à ce nouvel objet. `a` reste inchangé.

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 et il faudra être vigilant, notamment dans le cas d'une copie d'une liste avec une affectation.
Voyons cela :

Types immuables	Types mutables

`a = 1`
`b = a`
`a = 2`
On aura la variable `a` qui vaudra 2 et la variable `b` vaudra 1
	`l1 = [1, 2, 3]`
	`l2 = l1`
	`l1[0] = 7`
	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 Collection)
Python utilise un **ramasse-miettes** automatique pour gérer la mémoire et libérer de l'espace occupé par des objets qui ne sont plus référencés. Cela se fait principalement par comptage de références et détection de cycles.

II. Sommer une série / boucles

Une instruction du type `x = x + 1`, permet, quand on l'itère, de calculer des sommes.
Une suite d'instructions telles que :

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

`S = S + x`
va avoir l'effet suivant:
Introduire le nombre 1 dans la machine, dans une case mémoire qui le contiendra sous l'appellation `x`, introduire de même, dans une case mémoire qui contient la variable `S` le nombre 0, aller chercher les contenus des deux cases mémoires précédentes, les ajouter et replacer le résultat dans la case qui contenait la variable `S`.
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. L'ordinateur calcul toujours faux !
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

En mathématiques, déterminer si une série converge (s'approche d'une limite finie) ou diverge (ne s'approche pas d'une limite finie) est une question fondamentale. En calcul scientifique et en informatique, utiliser un ordinateur pour évaluer la convergence ou la divergence des séries est courant, notamment pour les séries complexes ou difficiles à analyser analytiquement.

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 découvrir expérimentalement qu'une série converge. Après un nombre fini (qui peut être grand) d'étapes, on aura la somme partielle qui deviendra une suite stationnaire et l'expérience numérique indique une convergence.
La certitude de divergence est plus compliquée à atteindre. Dans ce cas l'analyse mathématiques du problème peut aider.

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 :

$S_{10} = 2{,}929$ $S_{100} = 5{,}187$ $S_{1000} = 7{,}485$ $S_{10000} = 9{,}78$

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

Soit la fonction f tel que $u_n = \frac{1}{n} = f(n)$.

f est décroissante sur $]0;\ +\infty[$, on peut dont encadrer

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

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

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

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

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

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

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

Avec $f(n+1) \leq \int_n^{n+1} f(t)\,dt$, en remplaçant $n+1$ par n, on trouve : $f(n) \leq \int_{n-1}^{n} f(t)\,dt$

Ce qui nous donne ainsi :

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

En sommant, on obtient :

$$\sum_{n=1}^{N}\int_n^{n+1} f(t)\,dt \leq \sum_{n=1}^{N} f(n) \leq \sum_{n=1}^{N}\int_{n-1}^{n} f(t)\,dt$$

$$\Leftrightarrow \int_1^{N+1} f(t)\,dt \leq \sum_{n=1}^{N} f(n) \leq \int_0^{N} f(t)\,dt$$

Dans notre cas, l'intégration est simple :

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

$$\Leftrightarrow [\ln(t)]_1^{N+1} \leq \sum_{n=1}^{N} f(n) \leq 1 + [\ln(t)]_1^N$$

$$\Leftrightarrow \ln(N+1) \leq \sum_{n=1}^{N} f(n) \leq 1 + \ln(N)$$

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

L'objectif de ce TP sera de calculer

$$\int_0^1 e^{-\frac{x^2}{2}} dx$$

C'est une intégrale que l'on retrouve notamment dans la loi Normal.

Vous allez calculer une valeur approchée de cette intégrale à l'aide des 3 méthodes vues dans le cours :

1. Méthode des rectangles (Riemann),
2. Méthodes des trapèzes,
3. Méthode de Simpson.

Pour chaque méthode, vous calculerez epsilon (ε) qui sera l'écart entre la valeur approchée et la valeur exacte afin de pouvoir comparer cet écart en fonction du nombre n d'intervalles choisis et vous calculerez combien il faut d'intervalles pour obtenir $\varepsilon \leq 10^{-4}$.

Il faudra impérativement créer une fonction `integrale_rectangles`, `integrale_trapezes` et `integrale_simpson` pour le calcul approché des intégrales.

Vous conclurez sur la méthode qui semble avoir le moins besoin d'intervalles pour obtenir une bonne approximation.

Méthode	Nombre d'intervalles pour atteindre $\varepsilon \leq 10^{-4}$
Méthode des rectangles (Riemann)	
Méthodes des trapèzes	
Méthode de Simpson	

Aide Python :

Pour la valeur absolue : `abs(valeur)`

exponentielle :

```
import numpy as np
np.exp(valeur)
```

```
Ou
from numpy import *
expl(valeur)
```

Calcul d'une intégrale :

```
import scipy.integrate as spi
integrale = spi.quad(fonction, borne inf, borne sup)
```

Solution :

Exercice 1 :

On sait que :

$$S_n = a\sum_{k=0}^{n} q^k = a\frac{1-q^{n+1}}{1-q} \; avec\; a\; premier\; terme\; et\; q\; la\; raison$$

Pour notre somme, on a donc avec notre somme :

$$S_n = \frac{1}{4}\sum_{k=0}^{n}(\frac{1}{4})^k = \frac{1}{4}\frac{1-(\frac{1}{4})^{n+1}}{1-\frac{1}{4}} = \frac{1}{4}\frac{1-(\frac{1}{4})^{n+1}}{\frac{3}{4}} = \frac{1}{3}(1-\left(\frac{1}{4}\right)^{n+1})$$

Ainsi :

$$S = \lim_{n\to+\infty} S_n = \lim_{n\to+\infty} S_n = \frac{1}{3}\left(1-\left(\frac{1}{4}\right)^{n+1}\right) = \frac{1}{3}$$