Fiche-résumé de la séance 2 (Listes)

Méthodes numériques

2023–2024

Rappel concernant les fichiers notebook .ipynb/.md

- ☞ On exécute les cellules en combinant les deux touches MAJ + Entrer
- ☞ **Relancer périodiquement le noyau de votre notebook**, en particulier le jour de l'examen ou du partiel, pour bien vérifier que le code fonctionne vraiment.

Déclaration de variables, de listes

☞ La déclaration d'une variable consiste à associer un objet à une étiquette (le nom de la variable).

☞ **Attention !** Python est sensible à la casse : les majuscules et les minuscules sont bien distinctes. Ainsi « `N` » et « `n` » sont deux noms de variables différents.

☞ **Conseil/Bonne habitude :**

- Éviter d'utiliser des noms de variables à un seul caractère, sauf de manière locale ;
- Pour les noms de variables on utilisera les caractères alpha-numériques non accentués (`a`–`z`, `A`–`Z`, les chiffres `0`–`9` et le tiret-bas « `_` ») ;
- Dans le cas des flottants, ne pas oublier d'utiliser la notation scientifique en « `X.YYYeN` ».

Exemple :

```
Ma_Liste = [0, 1.0, "2", 6.67e-11]
```

Une liste se définit ainsi entre crochets « `[` » et « `]` » comme délimiteurs et avec des virgules pour séparer les éléments de la liste. `[]` désigne une liste vide.

Utilisation de formules mathématiques

☞ Pour ceux qui connaissent son existence, on oubliera l'existence de la libraire `math`, on la remplacera par `numpy` et `scipy`

```
import numpy as np
x = 1.5
print(np.exp(x))
```

Erreurs fréquentes : Elles sont souvent dues à la retranscription directe de formules tapées où les conventions font que des parenthèses ou des multiplications sont omises. Ainsi :

- Erreur de parenthésage : une parenthèse ouvrante ou fermante en trop, un parenthèse fermante mal placée. Pour cela, regardez les couleurs des parenthèses associées ;
- Oubli de « * » pour les multiplications ;
- Oubli des parenthèses au numérateur ou au dénominateur d'une fraction.

Slicing de liste/chaîne de caractères/tableau

Nous partons de la chaîne de caractères `s = "ABCDEFGHIJKL"`

indice ii	0	1	2	3	4	5	6	7	8	9	10	11
caractère s[ii]	A	B	C	D	E	F	G	H	I	J	K	L

Action	Code	Exécution
Extraction	`s[2:7]`	`CDEFG`
Les 4 premiers	`s[:4]`	`ABCD`
Les 4 derniers	`s[-4:]`	`IJKL`
Tous sauf les 4 premiers	`s[4:]`	`EFGHIJKL`
Tous sauf les 4 derniers	`s[:-4]`	`ABCDEFGH`
Partitionner	`s[:3], s[3:7], s[7:]`	`"ABC" "DEFG" "HIJKL"`
De 3 en 3	`s[::3]`	`ADGJ`
De 3 en 3 à partir de la fin	`s[::-3]`	`LIFC`
Les indices pairs	`s[::2]`	`ACEGIK`
Les indices impairs	`s[1::2]`	`BDFHJL`
Copie des valeurs seules	`s[::]`	`ABCDEFGHIJKL`
Copie à l'envers	`s[::-1]`	`LKJIHGFEDCBA`

Boucle `for`

☞ Elle permet d'exécuter de manière itérée un bloc d'instruction, en fonction du parcours d'un itérable (par exemple un `range`, une `list`)

Syntaxe :

```
for variable in iterable:
    instruction1
    instruction2
    instruction3
```

- La fin du `for` doit se terminer par un deux-points « `:` »
- Le bloc d'instruction doit être indenté (à 4 caractères)

Liste en compréhension

☞ Cette notion permet de construire des listes de manière itérative et conditionnelle en parcourant un iterable (et optionnellement, en faisant un test sur le contenu de l'iterable).

Syntaxe :

```
liste_sans_test = [valeur for variable in iterable]
liste_avec_test = [valeur for variable in iterable if TEST]
```

Exemple :

```
maliste = [ ii**2 for ii in range(1,12,2) if ii%3==0 ]
print(maliste)
```

rend

```
[9, 81]
```

car les entiers `ii`, multiples de 3, impairs, entre 1 et 11 inclus (`range(1,12,2)`) sont : 3 et 9. Dans la liste obtenue, on en a pris les carrés.