Initiation aux tableaux

Activité 02

Partie 1 : le tour des boucles

Exercice 1

1. Écrire sur papier une fonction `somme_inverse` de paramètre un entier `n` et renvoie la somme des inverses des entiers de 1 à `n` (par exemple, $\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4} \simeq 2.0833333333$).
2. **Donner votre code à votre voisin·e de droite, et tester sur machine le code qui vous est donné.** Corriger le code donné si besoin est.
3. Tester le code ci-contre, et en déduire la signification de `for j in range(d,f)`. Utiliser cette dernière syntaxe pour modifier votre code précédent.

```
for j in range(3,7):
    print(j)
```

Exercice 2

Les programmes suivants ne compilent pas, ou n'ont pas l'effet escompté. Corriger ces programmes, puis tester sur machine.

```
def factorielle(n):
    p = 1
    for i in range(n):
        p = p * i
    return p
```

```
def somme_alternee(n):
    s = 0
    for k in rage(n+1):
        s += ((-1)**k)*k
        return s
```

```
Def binaire(n):
    k = n
    for i in range(n)
        print(k//2)
        k = n%2
```

Exercice 3

Il existe un autre type de boucle fondamental : les boucles « tant que ». Plutôt que répéter un bloc `n` fois, l'idée est qu'on va répéter le bloc tant qu'une condition est vérifiée. Si la condition n'est plus vérifiée, on n'exécute plus la boucle, et on passe à la suite du programme.

Tester le programme ci-contre.

```
a = 9
while a > 0 :
    print(a)
    a = a - 2
print(a)
```

Exercice 4

Compléter les codes suivants :

Objectif : trouver la première puissance de 2 plus grande que 123 456 789.

```
compteur = 1
while ... :
    compteur = compteur * 2
print(compteur)
```

Objectif : trouver le premier entier naturel j tel que $|\cos(j)| < 0.001$.

```
from math import cos
j = 0
while ... :
    ...
print(j)
```

Exercice 5

Transformer les programmes suivants pour avoir une boucle « tant que » au lieu d'une boucle « pour ».

```
x = 3
for i in range(16) :
    x += 3
print(x)
```

```
t = ""
for x in range(10):
    t += str(x)
print(t)
```

Exercice 6

Une suite de Collatz (ou de Syracuse, ou 3n+1) est une suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 \in \mathbb{N}^*$, et :

$$u_{n+1} = \begin{cases} \dfrac{u_n}{2} \text{ si } u_n \text{ pair} \\ 3u_n + 1 \text{ sinon.} \end{cases}$$

On conjecture, à l'heure actuelle, que pour tout $u_0 \in \mathbb{N}^*$, la suite de Collatz associée atteindra à un moment donné le nombre 1.

1. Soit $u_0 = 9$. Déterminer, à la main, le premier $n \in \mathbb{N}$ tel que $u_n = 1$.
2. Écrire en Python une fonction `collatz(u0)` qui renvoie le premier $n \in \mathbb{N}$ tel que $u_n = 1$, avec $u_0 =$ `u0`.

Partie 2 : tableaux et parcours

En informatique, le calcul numérique est essentiel, mais ne suffit pas : il faut aussi structurer les données ... dans des structures de données. La structure de données fondamentales, en Python, est le **tableau dynamique** (ou liste).

En Python, un tableau ressemble à quelque chose comme `[4,1,8,2,16,4]` : des crochets aux extrêmités, des virgules pour séparer les éléments.

Exercice 7

Tester **successivement** les commandes suivantes dans l'invite de commandes (**la fenêtre de droite**). Expliquer brièvement ce que fait chaque ligne, et en déduire notamment :

— comment désigner la première case d'un tableau ;

— comment obtenir la longueur d'un tableau ;

— comment modifier une case d'un tableau.

```
>>> tab = [4,1,8,2,16,4]
>>> tab
>>> tab[0]
>>> tab[1]
>>> len(tab)
>>> tab[2] = 55555
>>> tab
```

Exercice 8

À faire sur papier avant de tester sur machine.

1. À votre avis, que devrait contenir la variable `tab3` ?
2. Maintenant, tester les commandes suivantes dans l'invite de commandes.
3. En déduire ce que représente l'opérateur `+` pour un tableau : on appelle cette opération la **concaténation**.

```
>>> tab1 = [1,2,3,4]
>>> tab2 = [5,6,7,8]
>>> tab3 = tab1 + tab2
>>> tab3
```

Exercice 9

1. Tester successivement les commandes suivantes dans l'invite de commandes.
2. En déduire une syntaxe pour rajouter une case à un tableau.
3. Tester la commande « `tab+= 2` » : que se passe-t-il ? Quelle est l'erreur de syntaxe ?

```
>>> tab = []
>>> tab += [0]
>>> tab
>>> tab += [1]
```

4. Écrire en Python une fonction `premiers_nombres(n)` qui renvoie un tableau contenant dans l'ordre les `n` premiers nombres à partir de 0 inclus (attention à aller jusqu'à la bonne longueur !).
5. Écrire en Python une fonction `puissances_de_2(n)` qui renvoie un tableau contenant dans l'ordre des `n` premières puissances de 2 à partir de 1 inclus.
6. Écrire en Python une fonction `premiers_fibo(n)` qui renvoie un tableau contenant les `n` premiers termes de la suite de Fibonacci, définie par $f_0 = 0, f_1 = 1$ et $f_{n+2} = f_{n+1} + f_n$: par exemple, `premiers_fibo(8)` renverra `[0,1,1,2,3,5,8,13]`.

Exercice 10

1. Tester le programme suivant.
2. Modifier la 3ème ligne de ce programme pour qu'il affiche 12, 10, 8, 6, 4, 2 et 0.

```
tab = [6,5,4,3,2,1,0]
for i in range(len(tab)) :
    print(tab[i])
```

3. Écrire en Python une fonction `affiche(tab)` qui affiche les éléments d'un tableau donné en argument.
4. Écrire en Python une fonction `est_dedans(elem,tab)` qui renvoie `True` si `elem` est un élément de `tab`, et `False` sinon.
5. Essayer de modifier la fonction précédente pour ne pas avoir besoin de créer une variable.
6. Écrire en Python une fonction `nb_occurrences(x,t)` qui renvoie le nombre d'occurrences de l'élément `x` dans le tableau `t` (si `x` n'apparaît pas, ce nombre vaudra 0).

Exercice 11

1. Écrire en Python une fonction `maximum(tab)` qui renvoie l'élément maximum d'un tableau `tab` (qu'on suppose non vide). Par exemple, le maximum de `[5,1,3,9,103,5,-5]` est 103.
2. Écrire en Python une fonction `indice_max(tab)` qui renvoie **l'indice** du maximum d'un tableau. Par exemple, `indice_max([5,1,3,9,103,5,-5])` renverra 4, qui est l'indice correspondant à 103.
3. Écrire en Python une fonction `sousmaximum(tab)` qui renvoie la deuxième plus grande valeur d'un tableau `tab` (qu'on supposera avoir au moins deux valeurs distinctes). Par exemple, sur le tableau `[8,3,21,11,-1,21]`, on renverra 11.
4. Écrire en Python une fonction `somme(tab)` qui renvoie la somme des éléments d'un tableau. Par exemple, `somme([4,1,8,1,16,1])` renverra $4 + 1 + 8 + 1 + 16 + 1 = 31$.
5. Écrire en Python une fonction `moyenne(tab)` qui renvoie la moyenne des éléments d'un tableau.

Exercice 12

1. Écrire en Python une fonction `begayant(tab)` qui renvoie le bégayant de `tab`, où toutes les valeurs sont dédoublées : `begayant([1,2,3,4])` renverra `[1,1,2,2,3,3,4,4]`.
2. Écrire une fonction `positifs(tab)` qui renvoie un tableau ne contenant que les valeurs positives de `tab`.

Exercice 13

1. En utilisant la fonction `est_dedans`, écrire une fonction `element_en_commun(tab1,tab2)` qui renvoie `True` si `tab1` et `tab2` partagent un élément en commun.
2. Modifier la fonction précédente pour qu'elle n'appelle pas la fonction `est_dedans` : vous la recoderez peu ou prou à l'intérieur de la boucle « pour ».

Exercice 14

Une structure très similaire aux tableaux est celle des **uplets** (ou tuples). En Python, un tuple ressemble à `(4,1,8,2,16,4)` : cette fois, on trouve des parenthèses aux extrêmités.

1. Tester les commandes ci-contre successivement.
2. Quelle est la seule commande interdite ?
3. On dit qu'une structure de données est **mutable** lorsqu'on peut en modifier le contenu des cases. Les uplets sont-ils mutables ? Les tableaux sont-ils mutables ?

```
>>> tup = (4,1,8,2,16,4)
>>> tup
>>> tup[0]
>>> len(tup)
>>> tup[2] = 55555
>>> tup + (1,2,3,4)
```

Exercice 15

Une dernière structure importante, que nous reverrons incessamment, est celle de **chaîne de caractères** (ou string). Les strings servent spécifiquement à stocker du texte, et ressemblent à `"Bonjour, je suis du texte."` (guillemets doubles) ou `'youpi youpi youpi'` (guillemets simples).

1. Tester les commandes suivantes.
2. Les strings sont-ils mutables ?
3. En reprenant tel quel l'exercice 10, tester `est_dedans("j","bonjour")` : qu'en conclure ?
4. En reprenant tel quel l'exercice 12, tester `begayant("bonjour")` : qu'obtenez-vous ?

```
>>> str = "J'adore l'informatique"
>>> str
>>> str[3]
>>> len(str)
>>> tup[2] = "o"
>>> tup + " et les mathématiques"
```

5. Réécrire une fonction `begayant_string(s)` qui renvoie le bégayant du string `s` sous forme d'un string.

Exercice 16

Écrire en Python une fonction `nb_voyelles(str)` qui renvoie le nombre de voyelles dans le string `str` (dont on supposera toutes les lettres en minuscule).

Exercice 17

1. Écrire une fonction `lettres(texte)` qui renvoie le tableau des caractères apparaissant dans `texte`. Par exemple, `lettres("abracadabra")` renverra `["a","b","r","c","d"]`.
2. Écrire une fonction `lettre_la_plus_commune(texte)` qui renvoie ... c'est dans le titre.
3. Écrire une fonction `mots(texte)` qui renvoie le tableau des mots présents dans le string `texte`. On suppose que les mots sont séparés par des espaces (et aucun autre signe de ponctuation). Par exemple, `mots("Bonjour j'aime beaucoup l'informatique")` renverra `["Bonjour","j'aime","beaucoup","l'informatique"]`.
4. On dit qu'un mot `m1` est une anagramme du mot `m2` s'il est possible de réarranger les lettres de `m1` pour

former le mot `m2`. Écrire une fonction Python `est_anagramme(m1,m2)` qui renvoie `True` si `m1` est une anagramme de `m2`.

Indication : pourquoi est-ce que `maman` *et* `mana` *ne sont pas des anagrammes ?*

Exercice 18 : chiffre de César

La cryptographie est la branche de l'informatique qui s'intéresse au codage et au décodage de texte. Cet exercice s'attache à étudier l'algorithme de cryptographie le plus simple et le plus classique.

1. La commande `ord` permet d'obtenir le code ASCII d'une lettre : pour faire court, il s'agit d'un entier compris entre 97 (pour `'a'`) et 122 (pour `'z'`). La commande `chr` permet l'opération retour : elle transforme un entier compris entre 97 et 122 en le caractère correspondant. Tester `ord('g')` et `chr(103)`.
2. Écrire une fonction `decalage(n,d)` qui, étant donnés un entier `n` compris entre 97 et 122 et un entier `d`, renvoie l'entier `n+d` modulo ce-qu'il-faut pour rester entre 97 et 122. Par exemple, `decalage(115,20)` renverra 109 : en partant de 115 et en décalant à droite de 20 rangs, tout en bouclant entre 97 et 122, on atterrit sur 109.

   Indication : il faut trouver la bonne combinaison de modulo, d'additions et de soustractions.
3. Écrire une fonction `cesar(texte,d)` qui renvoie le décalage de `texte` avec le décalage `d` (on suppose que `texte` n'est constitué que de lettres minuscules). Par exemple, `cesar("desopilant",13)` renverra `"qrfbcvynag"` : c'est le mot où toutes les lettres ont été décalées de 13.
4. Écrire une fonction `cesar_avec_espace(texte,d)` procédant à la même opération, mais où les espaces sont préservées.
5. On vous donne `texte`, obtenu après **chiffrement**, et le décalage utilisé pour le chiffrer. Comment faire pour obtenir le texte d'origine ? Tester.
6. On vous donne le texte `"cfexkvdgj av dv jlzj tfltyv uv sfeev yvliv"`, mais on ne vous donne pas le décalage. Déchiffrer le texte en le moins de tentatives possibles.

   Indication : le texte d'origine était en français.
7. À votre avis, le chiffre de César est-il sécurisé ?

Pour la prochaine fois :

— Écrire une fonction `miroir(tab)` qui renvoie le miroir d'un tableau : `miroir([1,2,3,4])` renverra `[4,3,2,1]`.

— Écrire une fonction `palindrome(texte)` qui renvoie `True` si le string `texte` est un palindrome, `False` sinon. Pour rappel, un palindrome est un string qui se lit identiquement de gauche à droite et de droite à gauche (par exemple `"kayak"` ou `"hannah"`).

— Écrire une fonction `est_inferieur(tab1,tab2)` qui renvoie `True` si, case par case, les valeurs de `tab1` sont inférieures à celles de `tab2` (et `False` sinon). Par exemple, `[1,2,3,4,5]` est inférieur à `[2,3,4,5,6]`. On supposera, sans le vérifier, que `tab1` et `tab2` sont de mêmes longueurs.

— Écrire une fonction `sans_doublon(tab)` qui renvoie un tableau contenant les mêmes éléments que `tab`, mais sans doublon : par exemple, `sans_doublon([1,2,3,2,4,1,2,2,5,6])` renverra `[1,2,3,4,5,6]`.