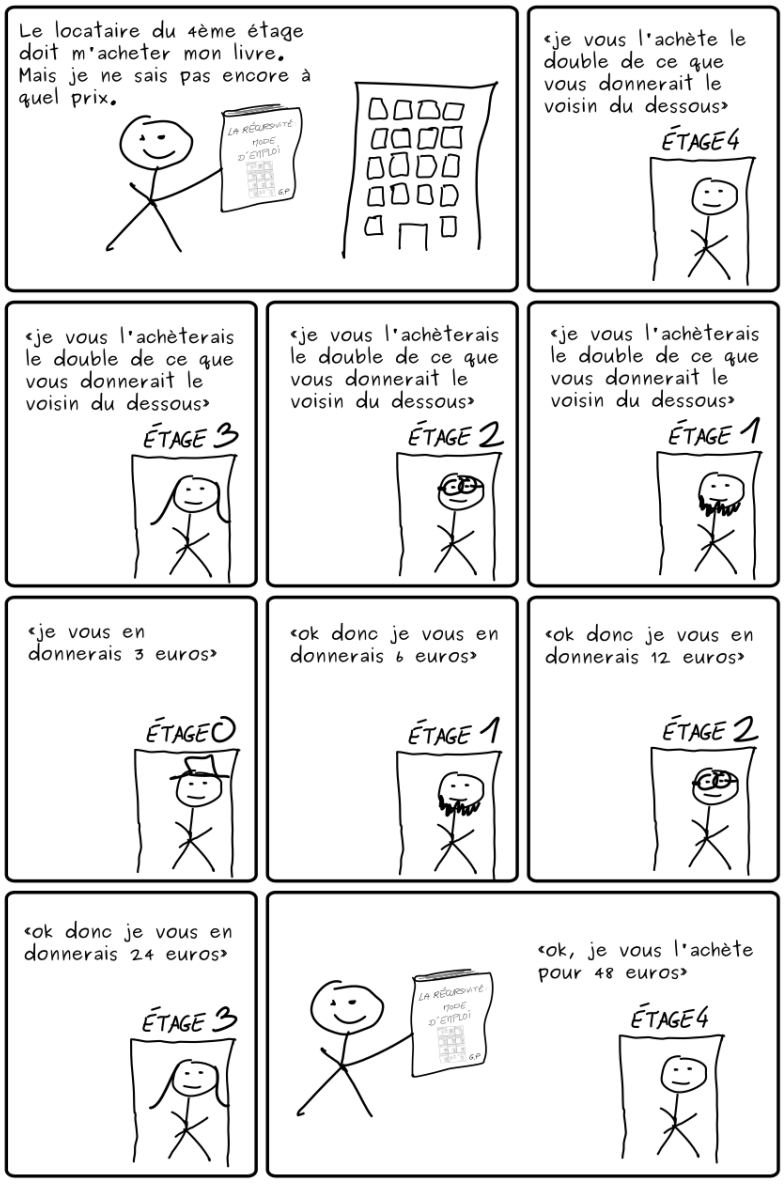
CAHIER D'ENTRAINEMENT SUR LA RÉCURSIVITÉ



EXERCICE 1

Cet exercice porte sur la programmation en général et la récursivité en particulier.
On s'intéresse dans cet exercice à un algorithme de mélange des éléments d'une liste.

1. Pour la suite, il sera utile de disposer d'une fonction `echange` qui permet d'échanger dans une liste `lst` les éléments d'indice `i1` et `i2`. Expliquer pourquoi le code Python ci-dessous ne réalise pas cet échange et en proposer une modification.

```
def echange(lst, i1, i2):
    lst[i2] = lst[i1]
    lst[i1] = lst[i2]
```

2. La documentation du module `random` de Python fournit les informations ci-dessous concernant la fonction `randint(a, b) : Renvoie un entier aléatoire N tel que` $a \leq N \leq b$`. Alias pour randrange(a, b+1).`

   Parmi les valeurs ci-dessous, quelles sont celles qui peuvent être renvoyées par l'appel `randint(0, 10)` ?

0	1	3.5	9	10	11

3. Le mélange de Fischer Yates est un algorithme permettant de permuter aléatoirement les éléments d'une liste. On donne ci-dessous une mise en œuvre récursive de cet algorithme en Python.

```
from random import randint
def melange(lst, ind):
    print(lst)
    if ind > 0:
        j = randint(0, ind)
        echange(lst, ind, j)
        melange(lst, ind-1)
```

   a. Expliquer pourquoi la fonction `melange` se termine toujours.
   b. Lors de l'appel de la fonction `melange`, la valeur du paramètre `ind` doit être égal au plus grand indice possible de la liste `lst`. Pour une liste de longueur n, quel est le nombre d'appels récursifs de la fonction `melange` effectués, sans compter l'appel initial ?
   c. On considère le script ci-dessous :

```
lst = [v for v in range(5)]
melange(lst, 4)
```

   On suppose que les valeurs successivement renvoyées par la fonction `randint` sont 2, 1, 2 et 0. Les deux premiers affichages produits par l'instruction `print(lst)` de la fonction `melange` sont :

```
[0, 1, 2, 3, 4]
[0, 1, 4, 3, 2]
```

   Donner les affichages suivants produits par la fonction `melange`.

   d. Proposer une version itérative du mélange de Fischer Yates.

EXERCICE 2

Cet exercice porte sur la programmation en général et la récursivité en particulier.

On considère un tableau de nombres de $n$lignes et $p$colonnes. Les lignes sont numérotées de 0 à $n-1$et les colonnes sont numérotées de 0 à $p-1$. La case en haut à gauche est repérée par $(0,0)$et la case en bas à droite par $(n-1,p-1)$.

On appelle **chemin** une succession de cases allant de la case $(0,0)$à la case $(n-1,p-1)$, en n'autorisant que des déplacements case par case : soit vers la droite, soit vers le bas.

On appelle **somme d'un chemin** la somme des entiers situés sur ce chemin.

Par exemple, pour le tableau Tsuivant :

4	1	1	3
2	0	2	1
3	1	5	1

- Un chemin est $(0,0),(0,1),(0,2),(1,2),(2,2),(2,3)$(en gras sur le tableau) ;
- La somme du chemin précédent est 14 ;
- $(0,0),(0,2),(2,2),(2,3)$n'est pas un chemin.

L'objectif de cet exercice est de déterminer la somme maximale pour tous les chemins possibles allant de la case $(0,0)$à la case $(n-1,p-1)$.

Question 1

On considère tous les chemins allant de la case $(0,0)$à la case $(2,3)$du tableau Tdonné en exemple.

1. Un tel chemin comprend nécessairement 3 déplacements vers la droite. Combien de déplacements vers le bas comprend-il ?
2. La longueur d'un chemin est égale au nombre de cases de ce chemin. Justifier que tous les chemins allant de $(0,0)$à $(2,3)$ont une longueur égale à 6.

Question 2

En listant tous les chemins possibles allant de $(0,0)$à $(2,3)$du tableau T, déterminer un chemin qui permet d'obtenir la somme maximale et la valeur de cette somme.

Question 3

On veut créer le tableau T'où chaque élément $T'[i][j]$est la somme maximale pour tous les chemins possibles allant de $(0,0)$à (i,j).

1. Compléter et recopier sur votre copie le tableau T'donné ci-dessous, associé au tableau

$$T = \begin{bmatrix} 4 & 1 & 1 & 3 \\ 2 & 0 & 2 & 1 \\ 3 & 1 & 5 & 1 \end{bmatrix}$$

$$T' = \begin{bmatrix} 4 & 5 & 6 & ? \\ 6 & ? & 8 & 10 \\ 9 & 10 & ? & 16 \end{bmatrix}$$

Justifier que si $j$est différent de 0, alors : $T'[0][j] = T[0][j] + T'[0][j-1]$.

Question 4

Justifier que si $i$et $j$sont différents de 0, alors :

$$T'[i][j] = T[i][j] + \max\left(T'[i-1][j], T'[i][j-1]\right).$$

Question 5

On veut créer la fonction récursive `somme_max` ayant pour paramètres un tableau T, un entier $i$et un entier j. Cette fonction renvoie la somme maximale pour tous les chemins possibles allant de la case $(0,0)$à la case (i,j).

1. Quel est le cas de base, à savoir le cas qui est traité directement sans faire appel à la fonction `somme_max` ? Que renvoie-t-on dans ce cas ?
2. À l'aide de la question précédente, écrire en Python la fonction récursive `somme_max`.
3. Quel appel de fonction doit-on faire pour résoudre le problème initial ?

EXERCICE 3

Cet exercice porte sur l'algorithmique et la programmation.

Un palindrome est un mot qui se lit de la même manière de la gauche vers la droite que de la droite vers la gauche (exemple : *« kayak »* est un palindrome).

On propose ci-dessous une fonction pour tester si un mot est un palindrome.

Précisions sur les instructions Python utilisées :

- `len(chaine)` : renvoie la longueur de la chaîne de caractères.
- `chaine[-1]` : renvoie le dernier caractère de la chaîne.
- `chaine[1:-1]` : renvoie la chaîne privée de son premier et de son dernier caractère.

Code de la fonction `tester_palindrome` :

```
def tester_palindrome(chaine):
    if len(chaine) < 2:
        return True
    elif chaine[0] != chaine[-1]:
        return False
    else:
        chaine = chaine[1:-1]
        return tester_palindrome(chaine)
```

1. On saisit, dans la console, l'instruction suivante :

   ```
   tester_palindrome('kayak')
   ```

   Combien de fois est appelée la fonction `tester_palindrome` lors de l'exécution de cette instruction ? *(On veillera à compter l'appel initial.)*

2.
   a. Justifier que la fonction `tester_palindrome` est récursive.
   b. Expliquer pourquoi l'appel à la fonction `tester_palindrome` se terminera quelle que soit la chaîne de caractères sur laquelle elle s'applique.

3. La saisie, dans la console, de l'instruction `tester_palindrome(53235)` génère une erreur.

   a. Parmi les quatre propositions suivantes, indiquer le type d'erreur affiché :

   - `ZeroDivisionError`
   - `ValueError`
   - `TypeError`
   - `IndexError`

   b. Proposer une ou plusieurs instructions qu'on pourrait écrire entre la ligne 1 et la ligne 2 du code de la fonction `tester_palindrome` pour afficher clairement cette erreur à l'utilisateur.

4. Écrire le code d'une fonction itérative (non récursive) `est_palindrome` qui prend en paramètre une chaîne de caractères et renvoie un booléen égal à `True` si la chaîne de caractères est un palindrome, `False` sinon.

EXERCICE 4

Cet exercice est consacré à l'analyse et à l'écriture de programmes récursifs.

1. Fonctions récursives
   a. Expliquer en quelques mots ce qu'est une fonction récursive.
   b. On considère la fonction Python suivante :

```
def compte_rebours(n):
    """ n est un entier positif ou nul """
    if n >= 0:
        print(n)
        compte_rebours(n - 1)
```

   L'appel `compte_rebours(3)` affiche successivement les nombres 3, 2, 1 et 0. Expliquer pourquoi le programme s'arrête après l'affichage du nombre 0.

2. Calcul de la factorielle

   En mathématiques, la factorielle d'un entier naturel $n$est le produit des nombres entiers strictement positifs inférieurs ou égaux à n. Par convention, la factorielle de 0 est 1.

   Exemples :

   - La factorielle de 1 est 1.
   - La factorielle de 2 est $2 \times 1 = 2$.
   - La factorielle de 3 est $3 \times 2 \times 1 = 6$.
   - La factorielle de 4 est $4 \times 3 \times 2 \times 1 = 24$.

   Recopier et compléter le programme ci-dessous afin que la fonction récursive `fact` renvoie la factorielle de l'entier passé en paramètre.

```
def fact(n):
    """ Renvoie le produit des nombres entiers strictement positifs
inférieurs à n """
    if n == 0:
        return à compléter
    else:
        return à compléter
```

Exemple : `fact(4)` doit renvoyer 24.

3. Somme des entiers récursive

La fonction `somme_entiers_rec` ci-dessous permet de calculer la somme des entiers de 0 à l'entier naturel $n$passé en paramètre.

Exemples :

- Pour $n = 0$, la fonction renvoie 0.
- Pour $n = 1$, la fonction renvoie $0 + 1 = 1$.
- Pour $n = 4$, la fonction renvoie $0 + 1 + 2 + 3 + 4 = 10$.

```
def somme_entiers_rec(n):
    """ Permet de calculer la somme des entiers, de 0 à l'entier
naturel n """
    if n == 0:
        return 0
    else:
        print(n)  # pour vérification
        return n + somme_entiers_rec(n - 1)
```

a. Écrire ce qui sera affiché dans la console après l'exécution de la ligne suivante :

```
res = somme_entiers_rec(3)
```

b. Quelle valeur sera alors affectée à la variable `res` ?

4. Somme des entiers itérative

Écrire en Python une fonction `somme_entiers` non récursive : cette fonction devra prendre en argument un entier naturel $n$et renvoyer la somme des entiers de 0 à $n$compris. Elle devra donc renvoyer le même résultat que la fonction `somme_entiers_rec` définie à la question 3.

Exemple : `somme_entiers(4)` doit renvoyer 10.

EXERCICE 5

Cet exercice traite du thème « programmation », et principalement de la récursivité.

On rappelle qu'une chaîne de caractères peut être représentée en Python par un texte entre guillemets "".
Voici quelques rappels sur les chaînes de caractères en Python :

- `len(chaine)` : renvoie la longueur de la chaîne de caractères.
- `chaine[0]` : renvoie le premier caractère de la chaîne.
- `chaine[1]` : renvoie le deuxième caractère de la chaîne.
- L'opérateur + permet de concaténer deux chaînes de caractères.

Exemples :

```
>>> texte = "bricot"
>>> len(texte)
6
>>> texte[0]
"b"
>>> texte[1]
"r"
>>> "a" + texte
"abricot"
```

On s'intéresse à la construction de chaînes de caractères suivant certaines règles :

Règle A : Une chaîne est construite suivant la règle A dans les deux cas suivants :

- Soit elle est égale à `"a"`.
- Soit elle est de la forme `"a" + chaine + "a"`, où chaine est une chaîne de caractères construite suivant la règle A.

Règle B : Une chaîne est construite suivant la règle B dans les deux cas suivants :

- Soit elle est de la forme `"b" + chaine + "b"`, où chaine est une chaîne de caractères construite suivant la règle A.
- Soit elle est de la forme `"b" + chaine + "b"`, où chaine est une chaîne de caractères construite suivant la règle B.

La fonction `A()` ci-dessous renvoie une chaîne de caractères construite suivant la règle A, en choisissant aléatoirement entre les deux cas de figure de cette règle.

```
from random import choice
def A():
    if choice([True, False]):
        return "a"
    else:
        return "a" + A() + "a"
```

1.

a. Cette fonction est-elle récursive ? Justifier.

b. La fonction `choice([True, False])` peut renvoyer `False` un très grand nombre de fois consécutives. Expliquer pourquoi ce cas de figure amènerait à une erreur d'exécution.

2. Dans la suite, on considère une deuxième version de la fonction `A`. À présent, la fonction prend en paramètre un entier n tel que, si la valeur de n est négative ou nulle, la fonction renvoie "a". Si la valeur de n est strictement positive, elle renvoie une chaîne de caractères construite suivant la règle A avec un n décrémenté de 1, en choisissant aléatoirement entre les deux cas de figure de cette règle.

```
def A(n):
    if ... or choice([True, False]):
        return "a"
    else:
        return "a" + ... + "a"
```

a. Recopier et compléter le code de cette nouvelle fonction `A` aux emplacements des points de suspension ....

b. Justifier le fait qu'un appel de la forme `A(n)` avec n un nombre entier positif inférieur à 50 termine toujours.

3. On donne ci-après le code de la fonction récursive `B` qui prend en paramètre un entier n et qui renvoie une chaîne de caractères construite suivant la règle `B`.

```
def B(n):
    if n <= 0 or choice([True, False]):
        return "b" + A(n - 1) + "b"
    else:
        return "b" + B(n - 1) + "b"
```

On admet que :

- Les appels `A(-1)` et `A(0)` renvoient la chaîne `"a"`.
- L'appel `A(1)` renvoie la chaîne `"a"` ou la chaîne "aaa".
- L'appel `A(2)` renvoie la chaîne `"a"`, la chaîne `"aaa"` ou la chaîne `"aaaaa"`.

Donner toutes les chaînes possibles renvoyées par les appels `B(0)`, `B(1)` et `B(2)`.

4. On suppose maintenant qu'on dispose d'une fonction raccourcir qui prend comme paramètre une chaîne de caractères de longueur supérieure ou égale à 2, et renvoie la chaîne de caractères obtenue en lui ôtant le premier et le dernier caractère.

Exemples :

```
>>> raccourcir("abricot")
"brico"
>>> raccourcir("ab")
""
```

a. Recopier et compléter les points de suspension ... du code de la fonction `regleA` ci-dessous pour qu'elle renvoie `True` si la chaîne passée en paramètre est construite suivant la règle A, et `False` sinon.

```
def regleA(chaine):
    n = len(chaine)
    if n >= 2:
        return chaine[0] == "a" and chaine[n - 1] == "a" and
regleA(...)
    else:
        return chaine == ...
```

b. Écrire le code d'une fonction `regleB`, prenant en paramètre une chaîne de caractères et renvoyant `True` si la chaîne est construite suivant la règle B, et `False` sinon.

EXERCICE 6

Cet exercice aborde les notions de classes, itération et récursivité.

Une petite société immobilière utilise une structure simplifiée pour stocker ses annonces de villas. Les données sont temporairement stockées dans une liste `v` dont voici la structure :

```
class Piece:
    def __init__(self, a, b):
        self.nom = a  # nom de la pièce
        self.sup = b  # superficie de la pièce

    def superficie(self):
        return self.sup

class Villa:
    def __init__(self, a, b, c, d, e):
        self.nom = a          # nom de la villa
        self.sejour = b       # caractéristiques du séjour
        self.ch1 = c          # caractéristiques de la 1ère chambre
        self.ch2 = d          # caractéristiques de la 2ème chambre
        self.eqCuis = e       # équipement de la cuisine ("eq" ou
"noneq")

    def nom(self):
        return self.nom

    def surface(self):
        return ...  # À compléter

    def equip(self):
        return self.eqCuis
```

Programme principal :

```
v = []
v.append(Villa("Les quatre vents", Piece("séjour", 40), Piece("ch1",
10), Piece("ch2", 20), "eq"))
v.append(Villa("Les goélands", Piece("séjour", 50), Piece("ch1", 15),
Piece("ch2", 15), "eq"))
```

```
v.append(Villa("Rêve d'été", Piece("séjour", 30), Piece("ch1", 15),
Piece("ch2", 20), "non eq"))
v.append(Villa("Les oliviers", Piece("séjour", 30), Piece("ch1", 10),
Piece("ch2", 20), "eq"))
v.append(Villa("Bellevue", Piece("séjour", 30), Piece("ch1", 10),
Piece("ch2", 20), "non eq"))
```

Partie A : Analyse du code et complétion

1.

   a. Combien d'éléments contient la liste `v` ?
   b. Que retourne l'instruction `v[1].nom()` ?
   c. Compléter la méthode `surface()` pour qu'elle renvoie la surface habitable totale de la villa (séjour + chambres) :

```
def surface(self):
    return ...  # À compléter
```

2. L'agent immobilier veut consulter les villas équipées d'une cuisine équipée. Écrire une portion de programme qui affiche la liste des noms des villas équipées. *(Parcourir la liste `v` et afficher le nom des villas où `eqCuis` est égal à `"eq"`.)*

Partie B : Récursivité

3. Parmi les propositions suivantes, recopier celle qui caractérise un appel récursif :
   - Appel d'une fonction par elle-même.
   - Appel dont l'exécution est un processus itératif.
   - Appel d'une fonction comportant une boucle.

4. L'agent immobilier veut trouver la villa la plus grande. Écrire une fonction récursive `max_surface(v)` qui extrait de la liste `v` la villa avec la plus grande surface.

Résolution des Tours de Hanoï en Python

1. Présentation du problème

Règles des Tours de Hanoï :

- On dispose de 3 tiges (A, B, C) et de N disques de tailles différentes.
- Les disques sont empilés sur la tige A, du plus grand (en bas) au plus petit (en haut).
- But : Déplacer tous les disques de la tige A vers la tige C, en respectant les règles suivantes :
  - On ne peut déplacer qu'un disque à la fois.
  - Un disque ne peut être placé que sur un disque plus grand ou sur une tige vide.

```
Tige A : [3, 2, 1]
Tige B : []
Tige C : []
```

Question 1 : Donner la solution optimale en 7 coups.

2. Approche récursive

Pour déplacer N disques de la tige A vers la tige C :

1. Déplacer les N-1 disques de A vers B (en utilisant C comme tige intermédiaire).
2. Déplacer le disque restant (le plus grand) de A vers C.
3. Déplacer les N-1 disques de B vers C (en utilisant A comme tige intermédiaire).

Question 2 : Donner le cas d'arrêt

3. Implémentation en Python

```
def hanoi(n, depart, arrivee, intermédiaire):
    """
    Résout le problème des Tours de Hanoï pour n disques.
    :param n: Nombre de disques.
    :param depart: Tige de départ (ex: "A").
    :param arrivee: Tige d'arrivée (ex: "C").
    :param intermédiaire: Tige intermédiaire (ex: "B").
    """
    if n == 1:
        print(f"Déplacer le disque 1 de {depart} vers {arrivee}.")
    else:
        hanoi(n - 1, depart, intermédiaire, arrivee)
        print(f"Déplacer le disque {n} de {depart} vers
{arrivee}.")
        hanoi(n - 1, intermédiaire, arrivee, depart)

# Exemple d'utilisation :
hanoi(3, "A", "C", "B")
```

Question 3 : Comprendre la récursivité

1. Que se passe-t-il si on appelle `hanoi(1, "A", "C", "B")` ?
2. Combien d'appels récursifs sont effectués pour `hanoi(4, "A", "C", "B")` ?

Question 4 : Calculer le nombre minimal de coups

Combien de coups sont nécessaires pour 5 disques ?

Question 5 : Modifier le code de la fonction `hanoi` pour compter les coups.