Programmation récursive :

Exercice 1 :

On considère la fonction`mystere(n)` définit par le code :

```
def mystere(n):
    if n == 0:
        return 1
    else:
        return n * mystere(n-1)
```

1. Quel est le résultat renvoyé par `mystere(4)` ? 24

   > `mystere(4)` = 4 × `mystere(3)` = 4 × 3 × `mystere(2)` = 4 × 3 × 2 ×`mystere(1)` = 4 × 3 × 2 × 1 × `mystere(0)` = 4 × 3 × 2 × 1 × 1 = 24

2. Décrire, en français, ce que fait cette fonction :

   > Cette fonction calcule le produit de tous les entiers de 1 à n, c'est à dire la factorielle de n.

3. Proposer une version itérative documentée pour cette fonction :

```
def mystere(n : int) ->  int :
    '''
    Calcule le produit de tous les entiers de 1 à n,
    c'est à dire la factorielle de n.
    Précondition :
        n (int) : entier naturel
    Postcondition :
        n! (int)
    Exemple :
    >>> mystere(4)
    24
    '''
    factorielle = 1
    for i in range(n) :
        factorielle = factorielle * (i + 1)
        print(factorielle)
    return factorielle
```

Exercice 2 :

On considère la fonction`sigma(n)` définit par le code :

```
def sigma(n):
    resultat = 0
    for i in range(n+1) :
        resultat = resultat + i
    return resultat
```

1. Quel est le résultat renvoyé par `sigma(0)` ?

> 0

1. Quel est le résultat renvoyé par `sigma(4)` ?

> 10

1. Décrire, en français, ce que fait cette fonction :

   > Cette fonction calcule la somme de tous les entiers de 1 à n

2. Proposer une version récursive documentée pour cette fonction :

```
def sigma(n : int) -> int:
    '''
    Calcule la somme de tous les entiers de 1 à n.
    Précondition :
        n (int) : entier naturel
    Postcondition :
        somme de 1 à n (int)
    Exemple :
    >>> sigma(4)
    10
    '''
    if n==0 :
        return 0
    else :
        return n + sigma(n-1)
```

Exercice 3 :

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.

On précise que, pour une chaîne de caractères `chaine` :

- l'instruction `len(chaine)` renvoie sa longueur ;
- l'instruction `chaine[-1]` renvoie son dernier caractère ;
- l'instruction `chaine[1:-1]` renvoie la chaîne privée de son premier caractère et de son dernier caractère.

Numéro de lignes	Fonction `tester_palindrome`
1	`def tester_palindrome(chaine):`
2	`    if len(chaine) < 2:`
3	`        return True`
4	`    elif chaine[0] != chaine[-1]:`
5	`        return False`
6	`    else:`
7	`        chaine = chaine[1:-1]`
8	`        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.

1. Pour le 1er appel on a chaine = 'kayak', pour le 2e appel chaine = 'aya', pour le 3e appel chaine = 'y'. On a donc 3 appels.

2. a. La fonction tester_palindrome s'appelle elle-même, elle est donc récursive.

   b. À chaque appel récursif, la chaine de caractère perd 2 caractères (le premier et le dernier). Après un certain nombre d'appels, il restera donc moins de 2 caractères. Si la chaine de caractères contient moins de 2 caractères, on tombe alors dans le cas de base (`len(chaine) < 2`), ce qui provoque l'arrêt des appels récursifs.

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

3. a. La fonction prend en paramètre une chaine de caractères, or, ici, nous passons en paramètre un entier, nous allons donc avoir un `TypeError` (car cela n'a aucun sens d'appliquer la méthode len sur un entier) b. `assert type(chaine) is str, "paramètre de type str attendu"`

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.

4.

```
def est_palindrome(chaine) :
    assert type(chaine) == str, "paramètre de type str attendu"
    while len(chaine) > 1 :
        if chaine[0] != chaine[-1] :
            return False
        chaine = chaine[1:-1]
    return True
```

```
def est_palindrome(chaine) :
    assert type(chaine) == str, "paramètre de type str attendu"

    while len(chaine) > 1 and chaine[0] == chaine[-1] :
        chaine = chaine[1:-1]
        print(chaine)
    if len(chaine) < 2 :
        return True
    return False
```

```python
def est_palindrome(chaine) :
    assert type(chaine) is str, "paramètre de type str attendu"
    i = 0
    j = len(chaine) - 1
    while i < j :
        if chaine[i] != chaine[j]:
            return False
        i = i + 1
        j = j - 1
    return True
```