

1. Fonction myst()

On considère la fonction `myst(1)` qui prend une liste en entrée :

```
def myst (l):
    if l == []:
        return 0
    else :
        return 1 + myst (l [1:])
```

✂ Pourquoi la fonction `myst()` est-elle récursive ?

✂ Quelle est la condition d'arrêt de la récursivité ?

✂ Qu'est-ce qui décroît à chaque appel récursif permettant d'affirmer que la récursivité s'arrête ? Justifier

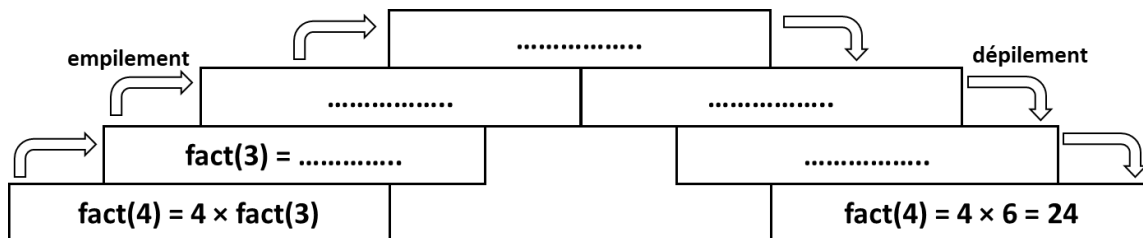
✂ Que fait cette fonction ?

2. Fonction fact()

Soit la fonction suivante :

```
def fact(n):
    if n <= 1:
        return 1
    else:
        return n * fact(n-1)
```

✂ Compléter les étapes d'empilement et de dépilement de la fonction `fact(4)`.



✂ Que réalise la fonction ?

3. Fonction et fonction_bis

```
def fonction(n):
    if n > 0:
        print(n)
        fonction(n-1)
```

```
def fonction_bis(n):
    if n > 0:
        fonction_bis(n-1)
        print(n)
```

✂ Quels sont les affichages produits par ces deux fonctions avec `n = 5` ?

✂ Vérifier les résultats dans une console.

4. Fonction minListe()

Soit l'algorithme de la fonction ci-dessous qui détermine le minimum des entiers d'une liste.

```
fonction minListe(liste):
    si taille(liste) = 1
        alors renvoyer liste[0]
    sinon
        mini = minListe(liste[1 :])
        si liste[0] < mini
            alors mini = liste[0]
        fin_si
    fin_si
    renvoyer mini
```

✍ Implémenter la fonction

5. Recherche du plus grand élément dans une liste non vide

On considère l'algorithme suivant qui prend une liste en entrée, une liste **l** non vide :

- Si la liste ne contient qu'un élément, on renvoie cet élément.
- Sinon, soit **p** le premier élément de la liste :
 - on calcule récursivement **m** le maximum des éléments de la liste **l[1 :]**
 - on renvoie le maximum de **p** et **m**

✍ Écrire une fonction **maxliste()** qui implémente cet algorithme en Python.

6. Recherche d'un préfixe dans un mot

On cherche à déterminer si un préfixe est présent dans un mot. Pour cela, on utilisera des chaînes de caractères en Python. On considérera que la chaîne vide "" est préfixe de toutes les autres chaînes.

✍ Compléter la fonction **prefixe(pre, mot)** qui répond au docstring et au jeu de tests fournis.

```
def prefixe(pre, mot):
    """Renvoie True si le début de la chaîne mot est pre. Sinon renvoie False
    pre, mot : String
    jeu de tests
    >>> prefixe("inf", "informatique")
    True
    >>> prefixe("nf", "informatique")
    False
    """
    if pre == "":
        . . .
    else :
        if pre[0] == mot[0]:
            . . .
        else:
            . . .

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

7. Rendu de monnaie

On prend comme système monétaire le système suivant représenté par un tableau.

`syst = [500, 200, 100, 50, 20, 10, 1]`

On suppose que le montant à rendre est un nombre entier.

Le programme renvoi le résultat sous forme d'une liste de listes :

```
>>> rendu(97, [500, 200, 100, 50, 20, 10, 1])
[[1, 50], [2, 20], [7, 1]]
```

Rappel de l'implémentation en itératif :

```
def rendu(n, s):
    reste_a_payer = n
    monnaie = []
    for piece in s :
        quotient = reste_a_payer // piece
        reste = reste_a_payer % piece
        if quotient != 0:
            monnaie.append([quotient , piece])
            reste_a_payer = reste
    return monnaie
```

✍ Compléter la version récursive `rendurec(n, s)` de la fonction rendu de monnaie.

```
def rendurec (n,s):
    if n == 0:
        . . .
    else :
        piece = s[0]
        quotient = n // piece
        reste = n % piece
        if quotient != 0:
            . . .
        else:
            return rendurec (n, s[1:])
```

La fonction donne :

```
>>> rendurec(97, [500, 200, 100, 50, 20, 10, 1])
[[1, 50], [2, 20], [7, 1]]
```

8. Pyramide de chaînes

✍ Implémenter un programme qui demande à l'utilisateur de saisir une chaîne de caractères et un entier (> 0), puis affiche sur la première ligne une fois la chaîne de caractères, sur la deuxième deux fois et ainsi de suite jusqu'à la dernière ligne.

Le programme doit faire appel à la fonction récursive `repete_chaine(texte, nbre)`. La fonction retourne une chaîne de caractère constitué de `texte` écrit `nbre` fois. `nbre` est un entier naturel.

```
Saisir la chaîne de caractères : bla
Saisir le nombre de répétitions : 5
bla
blabla
blablabla
blablablabla
blablablablabla
```

✍ Compléter la docstring de la fonction et la compléter avec des tests unitaires pertinents.