NSI (TERMINALE) :

Programmation par **récursivité**

I) Implémentation d'une fonction puissance

Exercice 1 : calcul de 2^n avec n entier naturel

1ère définition : **Implémentation de la fonction puissance par la méthode itérative**

si $n = 0$ alors $2^n = 1$, sinon $2^n = \underbrace{2 \times 2 \times 2 \times \ldots\ldots\ldots\ldots \times 2}_{n \text{ fois}}$

1°) Implémenter une fonction `calc(n)` prenant en paramètre un entier naturel n et retournant en sortie la valeur de 2^n avec une méthode itérative (boucle !!!)

2ième définition : **Implémentation de la fonction puissance par récursivité.**

si $n = 0$ alors $2^n = 1$, sinon $2^n = 2 \times 2^{n-1}$

Notons puissance(n) la fonction donnant la valeur de 2^n.
Avec cette notation, la définition devient : **si $n = 0$ alors puissance (0) = 1 sinon puissance (n) = $2 \times$ puissance($n - 1$)**

ALGORITHME :

Donnée(s) : n un entier naturel
fonction puissance(n)
Si n=0 **alors**
  renvoyer 1
Sinon
  renvoyer 2*puissance(n-1)

On en déduit que si $n > 0$ alors la fonction puissance s'appellera elle-même. On parle de définition par **récursivité**.

2°) Implémenter cette fonction `puissance(n)`

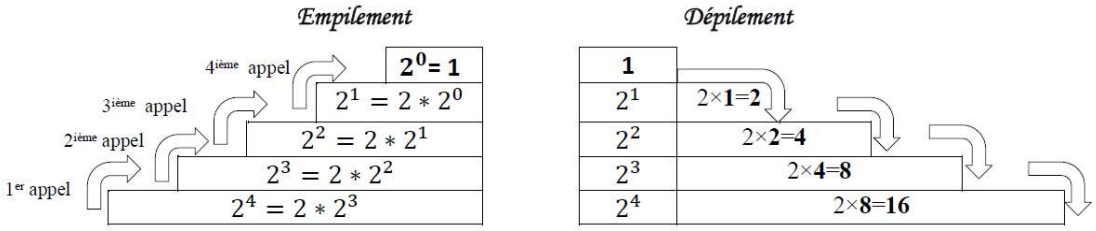
Exercice 2 : comparaison des vitesses d'éxécution

Comparer les vitesses d'exécutions des programmes version itérative et version récursive de l'exemple précédent. Pour tester la vitesse d'exécution d'une fonction on utilise le module **`timeit`**, comme le montre le code ci-dessous pour un entier naturel aléatoirement choisi entre 400 et
une liste de 1000 entiers choisis aléatoirement entre 500 et 1000 avec le module **`random`**.

```
from timeit import default_timer as timer
from random import randint

# Les deux fonctions ici

N=randint(500,1000)

debut=timer()
print(calc(N))
fin=timer()
print(f"temps version itérative {fin-debut}")
debut=timer()
print(puissance(N))
fin=timer()
print(f"temps version récursive {fin-debut}")
```

Que pouvez-vous en conclure ici ?

Le principe de programmation par récursivité est basé sur le fonctionnement de *« l'empilement – dépilement »* à l'aide d'une **pile d'exécution** stockant l'adresse mémoire de la prochaine instruction machine à exécuter et conservant une "trace" des valeurs des variables



Pour réaliser tout cela, le processeur gère une ou plusieurs piles dans lesquelles il stocke les adresses de retour des procédures et les valeurs des différentes variables. La récursivité est donc généralement plus lente en raison des frais généraux liés à la maintenance de la pile.

Les 3 règles fondamentales à respecter en paradigme récursif :

- Un fonction récursive doit avoir un " état trivial " , cela permet d'avoir une condition d'arrêt.
- Un algorithme récursif doit avoir une terminaison, c'est-à-dire conduire vers cet " état d'arrêt " (pas une infinité d'appels récursifs)
- Une fonction récursive doit s'appeler elle-même...

Il faut savoir que ces deux paradigmes de programmation sont équivalents, c'est-à-dire que tout algorithme itératif possède une version récursive, et réciproquement. Le type de langage de programmation utilisé peut conduire à programmer d'une manière ou d'une autre : par exemple Caml est un langage conçu pour exploiter la récursivité. À l'inverse, Python, même s'il l'autorise, ne favorise pas l'écriture récursive (limitation basse (1000 par défaut) du nombre d'appels récursifs).
Enfin, le choix d'écrire une fonction récursive ou itérative peut dépendre du problème à résoudre : certains problèmes se résolvent particulièrement simplement sous forme récursive)
La version récursive d'un algorithme est souvent plus succincte et plus lisible

	Récursivité	Itération
Signification	La fonction appelle elle-même.	Permet d'exécuter plusieurs fois l'ensemble des instructions.
Format	Dans une fonction récursive, seule la condition de terminaison est spécifiée.	L'itération comprend l'initialisation, la condition, l'exécution de l'instruction dans une boucle et la mise à jour (incrémenter et décrémenter) la variable de contrôle.
Terminaison	Une instruction conditionnelle est incluse dans le corps de la fonction pour forcer la fonction à retourner sans que l'appel de récurrence soit exécuté.	L'instruction d'itération est exécutée plusieurs fois jusqu'à ce qu'une certaine condition soit atteinte.
Condition	Si la fonction ne converge pas vers une condition appelée (cas de base), elle conduit à une récursion infinie.	Si la condition de contrôle dans l'instruction d'itération ne devient jamais fausse, elle conduit à une itération infinie.
Répétition infinie	Une récursion infinie peut bloquer le système.	Une boucle infinie utilise les cycles du processeur de manière répétée.
Application	La récursivité est toujours appliquée aux fonctions.	L'itération est appliquée aux instructions d'itération « ex : des boucles ».
Pile	La pile est utilisée pour stocker l'ensemble des nouvelles variables et paramètres locaux chaque fois que la fonction est appelée.	N'utilise pas la pile.
La vitesse	Lent	Rapide
Taille du code	La récursivité réduit la taille du code.	L'itération rend le code plus long.