TD3 : AP.

5 Divisibilité par trois

Il est bien connu qu'un nombre entier est divisible par trois si et seulement si la somme des chiffres de son écriture décimale l'est aussi. Voilà donc un bel algorithme récursif.

1. Étudiez le (ou les) cas de base nécessaire(s). Qu'est ce qui garantit l'arrêt de cet algorithme ?
2. Écrivez une fonction récursive de calcul de la somme des chiffres de l'écriture décimale d'un entier.
3. Écrivez un prédicat récursif renvoyant `True` si, et seulement si, l'entier passé en paramètre est divisible par trois *sans utiliser l'opérateur* `%` ni la fonction `str`.
4. Donnez un variant de l'algorithme `est_divisible_par_trois` garantissant son arrêt

Le nombre passé en paramètre doit avoir une longueur de 1.

2.
```
def calcul_somme_chiffres(n:int)->int:
    if nbre_chiffre(n)==1:
        Return n
    else
        Return n%10 + calcul_somme_chiffres(n//10)
```

3.
```
def est_divisible_3(n:int)->int:
    if nbre_chiffre(n)==1:
      Return n==0 OR n==3 OR n=6 OR n==9
    else
      return est_divisible(calcul_somme_chiffres(n)).
```

4. La taille de n est un variant de l'algo car tend vers 1 et est s+ ↘

6 Algorithme d'Euclide

L'algorithme d'Euclide permet de calculer le pgcd de deux nombres entiers, c'est–à–dire le plus grand entier positif divisant ces deux nombres, par des divisions successives.

Voici le déroulement de cet algorithme pour le calcul du pgcd de $a = 119$ et $b = 544$

$$\begin{array}{rcrcrcr} 119 & = & 544 & \times & 0 & + & 119 \\ 544 & = & 119 & \times & 4 & + & 68 \\ 119 & = & 68 & \times & 1 & + & 51 \\ 68 & = & 51 & \times & 1 & + & \boxed{17} \\ 51 & = & \boxed{17} & \times & 3 & + & 0 \end{array}$$

Le pgcd de 119 et 544 est le dernier reste non nul, c'est–à–dire 17.

Le pgcd n'est pas défini lorsque les deux nombres sont nuls.

1. Exprimez de manière récursive cet algorithme. Vous pourrez supposer que les deux entiers a et b sont positifs ou nuls, et que l'un au moins de ces deux entiers n'est pas nul.
2. Codez cet algorithme en python en utilisant l'opérateur modulo.

1.
```
def euclide(a:int, b:int)->int:
    if a%b == 0:
        Return b
    else:
        Return euclide(b, a%b)
```

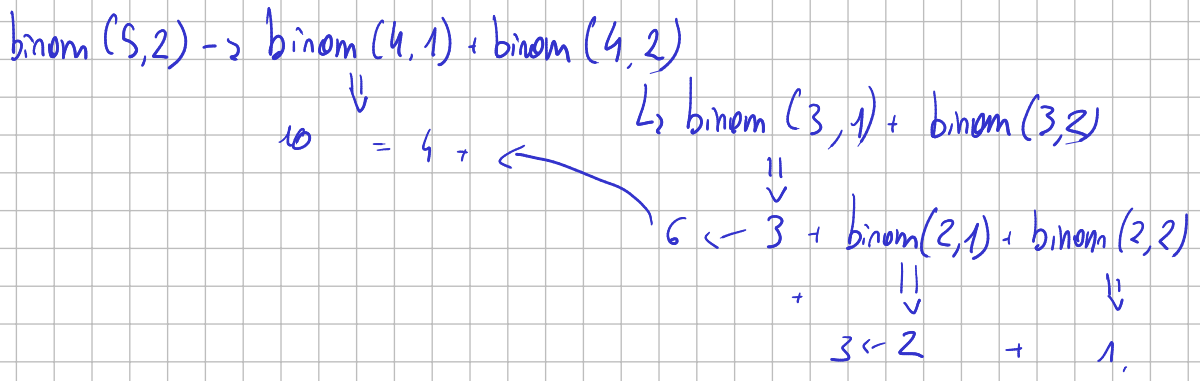
7 Coefficient binomial

Les coefficients binomiaux sont définis pour les entiers naturels $n \geq p \geq 0$ par $\binom{n}{p} = \frac{n!}{p!(n-p)!}$. Une relation de récurrence valable pour $n > p > 0$ est

$$\binom{n+1}{p+1} = \binom{n}{p} + \binom{n}{p+1}$$

1. Réalisez une fonction effectuant un calcul récursif des coefficients binomiaux qui n'utilise que l'addition des entiers.
2. Tracez le calcul du coefficient binomial pour $n = 5$ et $p = 2$.

1.
```
def binom(n, k):
    if k == 0 OR n == 0 OR n == k
        return 1
    if k==1 : Return n
    else:
        return binom(n-1,k-1) + binom(n-1, k)
```



8 Palindromes

Un *palindrome* est un mot dont les lettres lues de gauche à droite sont les mêmes que celles lues de droite à gauche. Les mots `radar`, `elle`, `été`, `ici` sont des palindromes.

1. Réalisez un prédicat récursif qui teste si un mot est un palindrome.

```
def palindrome(mot:str)->bool
    if len(mot)==0 or len(mot)==1:
        Return True
    elif mot[0]==mot[-1]:
        Return palindrome(mot[1:-1])
    else:
        return False.
```

9 Occurrences dans une chaîne

1. Réalisez une fonction récursive `nb_occurrences` d'entête

```
def nb_occurrences(chaine: str, car: str) -> int:
```

qui renvoie le nombre d'occurrences du caractère (second paramètre) dans la chaîne (premier paramètre).

Par exemple, `nb_occurrences('abcdebdb','z')` vaut 0 et `nb_occurrences('abcdebdb','b')` vaut 3.

2. Réalisez maintenant une fonction récursive `indice` paramétrée de la même façon que la fonction précédente

   - qui renvoie l'indice de la première occurrence du caractère dans la chaîne si le caractère est effectivement présent,
   - et qui déclenche une exception `ValueError: char not found`

   Cette fonction `indice` est en fait une méthode existante des chaînes de caractères nommée `index`. Une solution possible pour cette question est donc

```
def indice(s,c):
    return s.index (c)
```

mais ce n'est évidemment pas ce qui est attendu.

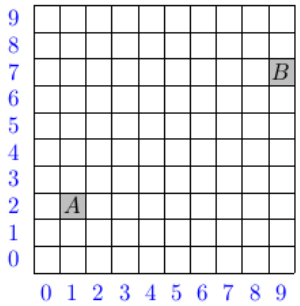
```
def nb_occurrences(chaine:str, car:str)->int:
    if len(chaine)==1:
        return int(chaine==car)
    else:
        m=len(chaine)//2
        return nb_occurrences(chaine[:m])+nb_occurrences(chaine[m:])
```

2.
```
def indice(chaine:str, car:str)->int:
    if len(chaine)==0:
        raise ValueError("char not found")
    elif chaine[0]==car:
        return 0
    else:
        return 1+indice(chaine[1:], car)
```

10 Tracer un segment

En informatique graphique, tous les points ont des coordonnées entières. Soit donc deux points A de coordonnées (x_A, y_A) et B de coordonnées (x_B, y_B).

Proposez un algorithme récursif pour tracer à l'écran le segment $[A, B]$. La commande primitive que vous utiliserez est la commande `plot(x, y)` qui dessine le pixel de coordonnées (x, y).




```
def dist(a:tuple[int,int], b:tuple[int,int])->int:
    return sqrt((b[0]-a[0])**2+(b[1]-a[1])**2).
```

```
def ligne(a: tuple[int, int], b: tuple[int, int]) -> None:
    if dist(a, b) <= 1:
        plot(a) ; plot(b)
    else
        M = ((a[0]+b[0])//2, (a[1]+b[1])//2)
        ligne(A, M) ; ligne(B, M)
```

11 Coloriage

Supposons données les coordonnées (entières) (x, y) d'un pixel situé à l'intérieur d'une région du plan délimitée par une courbe fermée. Les points sur la courbe délimitant la région sont de couleur noire, et ceux à l'intérieur sont blancs. On souhaite donner la couleur rouge à tous les points à l'intérieur de la région.

Proposez un algorithme récursif pour effectuer ce coloriage. Vous pourrez utiliser deux fonctions `get_color(x, y)` qui renvoie la couleur du pixel de coordonnées (x, y) et `set_color(x, y, color)` qui fixe la couleur du pixel de coordonnées (x, y).

on supposera définie trois constantes BLANC, ROUGE et NOIR représentant les trois couleurs