Séance 1 : Récursions

L1 – Université Côte d'Azur

Dans ce sujet, on s'interdit l'usage des boucles pour travailler exclusivement la récursion. Ceux et celles ayant déjà oublié de quoi il s'agit, peuvent se référer au second cours de bases de l'info 1 «Itérations et récursions » à la page : https://upinfo.univ-cotedazur.fr/~obaldellon/python Chacune de vos fonctions devront être testée plusieurs fois. Les tests seront écrit avec `assert` (lorsque c'est possible) sous les fonctions correspondantes.

Exercice 1 — Suites récurrentes

On considère la suite $(u)_n$ définie par récurrence par la formule suivante : $\begin{cases} u_0 &= 234 \\ u_{n+1} &= \frac{2}{3}u_n + \frac{1}{2} \end{cases}$

1. Écrivez la fonction `u(n)` correspondant à la suite. Vers quelle valeur semble converger la suite ?
2. Écrivez la fonction factorielle `fact(n)` $= 1 \times 2 \times ... \times n$.
3. Écrivez la fonction `somme_cube(n)` calculant la somme des n premiers cubes de manières récursives. Combien vaut-elle pour $n = 9$? On supposera $n \geq 1$.

$$\texttt{somme\_cube}(n) = \sum_{i=1}^{n} i^3$$

Exercice 2 — Logarithme entier

Soit n un entier positif. On appelle *logarithme entier* de n l'entier `le(n)` correspondant au nombre de fois où il faut diviser n par deux avant d'atteindre 1 ou 0. Par exemple, `le(5)` $= 1 +$ `le(2)` $= 1 + (1 +$ `le(1)`$) = 2$.

1. Calculez à la main `le(3)`, `le(5)`, et `le(11)` puis écrivez les tests correspondants avec `assert`.
2. Écrivez la fonction récursive `le(n)` qui renvoie le logarithme entier de `n`.

Exercice 3 — Exponentiation rapide

On souhaite calculer de manière efficace la puissance entière d'un nombre. L'idée de l'algorithme est le suivant pour calculer a^b il y a trois possibilités :

- $b = 0$ dans ce cas, $a^b = 1$
- $b = 2n$ (b pair) dans ce cas $a^{2n} = (a^n)^2$
- $b = 2n + 1$ (b impair) dans ce cas $a^{2n+1} = a \times (a^n)^2$

Dans les deux cas récursifs, on se ramène à un calcul avec un exposant qui est deux fois plus petit.

Écrivez la fonction `expo_rapide(a,b)`.

Exercice 4 — PGCD

1. On cherche à calculer le PGCD de deux nombres entier. Par exemple, si $n = 21$ et $m = 15$, leur PGCD vaut 3. En effet 3 est un diviseur en commun à $21 = 3 \times 7$ et $15 = 3 \times 5$ et c'est même le plus grand possible.
   Pour calculer ce PGCD, on fait un raisonnement par cas
   - on se ramène au cas dans lequel $n \geq m$ quitte à échanger n et m
   - si n et m sont égaux, c'est fini ; ils sont égaux à leur pgcd.
   - sinon, le PGCD de n et m est égal au PGCD de $n - m$ et m

   Écrivez une fonction *récursive* `affiche_et_calcule_pgcd_naïf(n,m)` qui affiche les étapes du calcul du PGCD et renvoie le résultat. Par exemple,

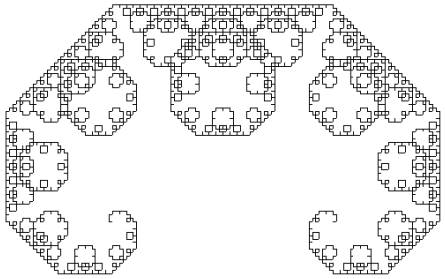
```
>>> d=affiche_et_calcule_pgcd_naïf(21,15)
calcule le pgcd de n=21 et m=15
calcule le pgcd de n=6 et m=15
calcule le pgcd de n=15 et m=6
calcule le pgcd de n=9 et m=6
calcule le pgcd de n=3 et m=6
calcule le pgcd de n=6 et m=3
calcule le pgcd de n=3 et m=3
>>> print('Le PGCD de 21 et 15 vaut',d)
Le PGCD de 21 et 15 vaut 3
```

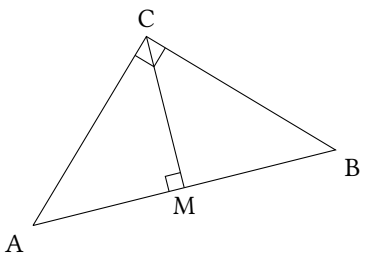
2. L'agorithme précedent n'est pas efficace. Par exemple si $n = 100$ et $m = 12$, il va falloir faire de nombreuses soustractions du nombre 12 : $100 \to 88 \to 76 \to 64 \to \cdots \to 4$. On peut calculer par avance le nombre de soustraction à faire (ici 8) en faisant la division euclidienne de 100 par 12 ; ce qui nous donne 8 reste 4.
   On peut donc accélérer l'algorithme en passant non pas de n à $n - m$ mais de n à $n - qm$, où q est le quotient de la division euclidienne de n par m. C'est le fameux algorithme d'Euclide.
   Écrivez une fonction `affiche_et_calcule_pgcd(n,m)` pour effectuer le calcul dans sa version accélérée. On fera attention au cas d'arrêt.

```
>>> d=affiche_et_calcule_pgcd(21,15)
calcule le pgcd de n=21 et m=15
calcule le pgcd de n=15 et m=6
calcule le pgcd de n=6 et m=3
calcule le pgcd de n=3 et m=0
>>> print('Le PGCD de 21 et 15 vaut',d)
Le PGCD de 21 et 15 vaut 3
```

Exercice 5 — Dragon

On souhaite tracer la courbe fractale du Dragon (ci-dessous à gauche). Pour cela, on introduit la notion de coude d'un segment. Soit A et B deux points. On appelle coude du segment $[AB]$ le point obtenu en traçant un triangle isocèle rectangle dont la base est $[AB]$. Ci-dessous, à droite, le coude de $[AB]$ est le point C.





1. Écrire une fonction `coude(A,B)` qui à partir de deux couples de coordonnées A et B, calcule les coordonnées du point C. *Indice : on pourra d'abord calculer les coordonnées de M, milieu de $[AB]$ puis remarquer que C s'obtient à partir de A par une translation de vecteur $\overrightarrow{AM} + \overrightarrow{MC}$. On rappelle que si $\overrightarrow{AM} = \begin{pmatrix} x \\ y \end{pmatrix}$ alors $\overrightarrow{MC} = \begin{pmatrix} -y \\ x \end{pmatrix}$*

2. Pour tracer la courbe du dragon de niveau n pour le segment $[AB]$:
   - si $n = 0$, on trace le segment $[AB]$
   - sinon, on trace les courbes du dragon de niveau $n - 1$ pour les segments $[AC]$ et $[CB]$ où C est le coude de $[AB]$.

```
import tkinter as tk
root = tk.Tk()
root.title("La fureur du dragon")
Hauteur = 600
Largeur = 600
Dessin=tk.Canvas(root,height=Hauteur,width=Largeur,bg="white")
Dessin.pack()

A = (175, 400)
B = (425, 400)
```

Exercice 6 — Recherche dichotomique

On souhaite écrire une fonction pour rechercher une valeur dans un tableau trié. Le principe est le suivant, pour savoir si `x` est dans la tableau, on regarde la valeur `m` située au milieu du tableau.

- si $m = x$ alors, c'est bon, on a trouvé x
- si $x < m$, alors x (s'il existe) se trouve dans la première partie du tableau
- si $m < x$, alors x (s'il existe) se trouve dans la seconde partie du tableau

Dans les deux cas, on s'est ramené à un tableau deux fois plus petit sur lequel on peut recommencer la recherche récursivement.

Écrivez une fonction `recherche_aux(x, tableau, a, b)` qui recherche `x` entre les indices `a` (inclu) et `b` (exclu). On renverra l'indice de `x` si `x` est dans le tableau et `-1` sinon. Votre fonction devra être récursive ; il faudra faire attention à bien gérer la cas d'arrêt.

```
def recherche_aux(x, tableau, a, b):
    # À vous de l'écrire

def recherche(x, tableau):
    recherche_aux(x, tableau, 0, len(tableau))
```

Exercice 7 — Écriture binaire

1. Écrivez une fonction `affiche_calcul_binaire(n)` qui affiche le calcul de la représentation binaire de `n`, de sorte que l'on peut lire *verticalement* la représentation en binaire de n. Par exemple, pour $13 = (1101)_2$, on obtiendra dans la console :

```
>>> affiche_calcul_binaire(13)
1 car 13 = 2 × 6 + 1
0 car 6 = 2 × 3 + 0
1 car 3 = 2 × 1 + 1
1
```

2. Écrivez une fonction récursive `affiche_binaire(n)` qui affiche l'écriture binaire de n dans le sens de lecture **usuel**. Par exemple, on doit obtenir :

```
>>> affiche_binaire(13)
1101
```