A.1 La fonction factorielle.

i) La fonction "Factorielle(n)" est définie comme suit :
Factorielle(n) = n · (n-1) · (n-2) · ... · 3 · 2 · 1 ; on écrit aussi n!
Exemple, 4! = 4 · 3 · 2 · 1 = 24
On convient aussi que Factorielle(0) = 1.
Écrivez une fonction "Factorielle(n)" qui retourne la factorielle d'un nombre entier positif ou nul.

ii) Une autre manière de définir la fonction factorielle est la suivante.
Fact(n) = n * Fact(n-1) et Fact(0) = 1. C'est une définition récursive.
Écrivez une fonction "Fact(n)" récursive, qui retourne la factorielle de n.

A.2 La suite de Fibonacci.

La suite de Fibonacci est définie ainsi.
Fib(0) = 1 ; Fib(1) = 1 ; Fib(n) = Fib(n-1) + Fib(n-2), pour $n \geq 2$.

i) Écrivez une fonction "Fib(n)" qui retourne la nième valeur de la suite de Fibonacci.

ii) Écrivez une autre fonction, de manière récursive, qui retourne la nième valeur de la suite de Fibonacci. C'est très peu efficace, juste pour s'exercer.

A.3 Les coefficients binomiaux.

On définit les coefficients binomiaux comme suit : $C(n,k) = \frac{n!}{k! \cdot (n-k)!}$.

Elle satisfait la propriété : $C(n,k) = C(n-1,k-1) + C(n-1,k)$, avec
$C(n,0) = 1$; $C(n,n) = 1$.

i) Écrivez de manière récursive une fonction "Coef(n, k)" qui retourne le coefficient binomial C(n, k). $n \geq k$.
Encore une fois, c'est très peu efficace, juste pour s'exercer.

A.4 Une curiosité, sans importance pour vous, la fonction d'Ackermann.

Cette fonction a une importance historique en algorithmique théorique, car on montre qu'il est impossible de la programmer en utilisant uniquement les opérations standards et les boucles "for". Elle croit extrêmement rapidement.
Elle est définie ainsi : $Ack(m,n) = Ack(m-1, Ack(m, n-1))$, avec
$Ack(0,n) = n+1$; $Ack(m,0) = Ack(m-1,1)$.

i) Si vous êtes curieux (curieuse), écrivez de manière récursive une fonction "Ack(m, n)" qui retourne la valeur de Ack(m, n).
Pour m = 4, cela dépasse les limite de calculs de Python.
Pour m = 3 et n > 7, cela dépasse les limite de calculs de Python.

Suite au dos, pour de joli dessins avec Turtle ...

A.5 Un dessin type "flocon de Koch"

i) En utilisant la librairie "turtle", écrivez une fonction "`Trait(niveau, longueur)`" qui est défini comme suit :

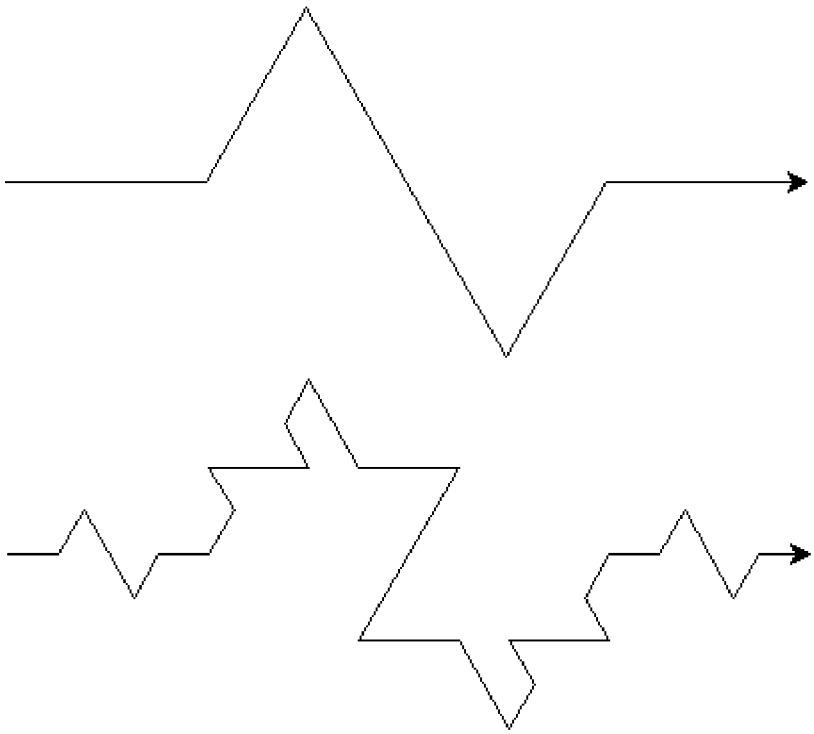
° Si `niveau` est nul, alors un trait de longueur "`longueur`" est dessiné.

° Sinon, un `Trait` de niveau "`niveau - 1`" et de longueur "`longueur/4`" est dessiné,
suivit d'une rotation à gauche de 60°,
suivit d'un `Trait` de niveau "`niveau - 1`" et de longueur "`longueur/4`" est dessiné,
suivit d'une rotation à droite de 120°,
suivit d'un `Trait` de niveau "`niveau - 1`" et de longueur "`longueur/2`" est dessiné,
suivit d'une rotation à gauche de 120°,
suivit d'un `Trait` de niveau "`niveau - 1`" et de longueur "`longueur/4`" est dessiné,
suivit d'une rotation à droite de 60°,
suivit d'un `Trait` de niveau "`niveau - 1`" et de longueur "`longueur/4`" est dessiné.

Exemples :

"`Trait(0, 400)`" :

"`Trait(1, 400)`" :



ii)

° Ouvrir la fenêtre avec : `setup(width=900, height=950, startx=10, starty=15)`

° Définir la vitesse maximale avec : `speed(0)`

° Sans tracer de trait, allez en position (-400, 0)

° `seth(0) # défini l'orientation de la tortue`

° Testez votre fonction avec "`Trait(0, 800)`", puis "`Trait(1, 800)`"

A.6 Les spirales

i) En utilisant la librairie "turtle", écrivez une fonction "`Spirale(nbSeg, Power)`" qui dessinera une spirale en suivant les indications suivantes :

° Ouvrir la fenêtre avec : `setup(width=900, height=950, startx=10, starty=15)`

° Définir la vitesse maximale avec : `speed(0)`

° Sans tracer de trait, allez en position (-400, -420)

° Pour `nn` variant de 1 à `nbSeg` répétez la séquence :
avancer de 30, puis tourner à gauche de nn^{Power}, où

ii) Appelez la fonction "`Spirale(1000, 0.6)`"

iii) Si on augmente la valeur de "`Power`", la spirale sera-t-elle plus grande ou plus petite ?