TP 12
Révisions et consolidation 2

À nouveau ce TP ne contient aucun concept nouveau. Il s'agit uniquement d'un TP de révisions.
Les fonctions doivent être écrites de façon récursive uniquement quand la question demande explicitement d'écrire une fonction récursive. On parle en général de méthode *itérative* quand on traite le problème avec une boucle et une fonction qui n'est pas récursive.

Exercice 1. Écrire une fonction récursive `decompte(n)` qui affiche la chose suivante (il s'agit ici d'un exemple avec $n = 5$) :

```
>>> decompte(5)
5
4
3
2
1
Bonne année !!!
```

I Avec la combinatoire

Exercice 2. On rappelle qu'on peut définir les coefficients binomiaux $\binom{n}{k}$ de la façon suivante (on suppose $n \geqslant 0$) :

$$\binom{n}{k} = \begin{cases} 0 & \text{si } k < 0 \text{ ou } k > n \\ 1 & \text{si } n = 0 \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{sinon} \end{cases} \tag{1}$$

Écrire une fonction récursive `binome(k, n)` qui traduit le plus directement possible cette définition. Testez la fonction pour calculer par exemple $\binom{25}{10}$ ou $\binom{30}{10}$. Que se passe-t-il ?

Exercice 3.

1. Écrire une fonction itérative `factoriel(n)` qui calcule le nombre $n!$.
2. Écrire une fonction itérative `arrangements(k, n)` qui calcule le nombre $\frac{n!}{(n-k)!}$. On rappelle que ceci est le produit
$$n \times \cdots \times (n - k + 1). \tag{2}$$
3. Reprendre la même question avec une fonction récursive `arrangements2(k, n)`, en trouvant d'abord une bonne formulation récursive du problème en fonction de l'entier k.
4. En déduire une fonction `binome2(k, n)` qui calcule le coefficient binomial $\binom{n}{k}$ simplement par la formule $\frac{n!}{k!\,(n-k)!}$.

Remarque 1. Pour ceux qui n'y arrivent pas mais qui ont besoin des résultats, on trouve les fonction suivantes dans le module `math`, à importer avec `import math` :
- `math.factorial(n)` : retourne la factoriel de n
- `math.perm(n, k)` : retourne le nombre d'arrangements de k objets parmi n (= k-liste sans répétition parmi n objets), c'est à dire le produit qui est écrit ci-dessus (2).
- `math.comb(n, k)` : retourne le coefficient binomial $\binom{n}{k}$. Attention à l'ordre des arguments qui n'est pas le même que présenté ici (le premier argument de `math.comb` est n). Cela provient du fait que la notation anglaise-américaine pour nos coefficients binomiaux $\binom{n}{k}$ est C_n^k ...

Éventuellement le comportement de ces fonctions peut être un peu différents de celles que nous avons écrites si on leur donne des arguments $n < 0$ (affiche une erreur plutôt que retourner 0), mais cela est aussi décrit précisément dans leur aide accessible avec `help()`.

Exercice 4. *Rendu de monnaie*
On souhaite écrire une fonction `rendu(n)` qui prend en argument un entier n représentant une somme en euros et qui nous affiche comment il faut arriver à cette somme avec des billets et des pièces. Pour simplifier disons qu'on utilise uniquement pour l'instant des billets de 5 euros et des pièces de 1 et 2 euros. On voudrait par exemple obtenir les résultats suivants :

```
>>> rendu(13)
Billet de 5
Billet de 5
Pièce de 2
Pièce de 1
```

1. Proposer une formulation récursive du problème : une formule pour `rendu(n)` en fonction du rendu d'une somme plus petite.
2. Écrire la fonction récursive `rendu(n)`. La fonction produit des affichages mais ne retourne rien.

Exercice 5. *Palindromes*
On rappelle qu'un mot est un *palindrome* s'il se lit aussi bien de gauche à droite que de droite à gauche, par exemple le mot KAYAK ou le prénom ANNA. On souhaite écrire une fonction qui teste si un mot donné (chaine de caractère, de type `str`) est un palindrome

1. Proposer une formulation récursive du problème : une condition pour qu'un mot soit un palindrome, en fonction d'une partie de ce mot.
   Attention car il ne se passe pas exactement la même chose selon que le mot soit de longueur paire ou impaire (comme d'ailleurs dans les deux exemples donnés). Cela se traduit en fait dans la condition d'initialisation.
2. Écrire une fonction récursive `est_palindrome(s)` qui teste (ce qui signifie toujours : renvoie un booléen `True` ou `False`) si le mot donné dans la chaine `s` est bien un palindrome. On pourra utiliser toutes les fonctions de sélections d'indices et de tranches (les `s[a:b]`) sur les chaines.

Exercice 6. *Nombres de Catalan*
On s'intéresse au nombre C_n de façons possible de placer des parenthèses autour de multiplications dans une expressions de $n+1$ facteurs, en gardant l'ordre des facteurs. Par exemple pour $n=2$ le produit $a\times b\times c$ peut être parenthésé comme $a\times(b\times c)$ ou bien $(a\times b)\times c$, on a donc $C_2=2$. Pour $n=3$ il s'agit du produit $a\times b\times c\times d$ qui peut être parenthésé comme suit :

$$((a\times b)\times c)\times d \tag{3}$$
$$(a\times b)\times(c\times d) \tag{4}$$
$$(a\times(b\times c))\times d \tag{5}$$
$$a\times((b\times c)\times d) \tag{6}$$
$$a\times(b\times(c\times d)) \tag{7}$$

et donc $C_3=5$. On pose aussi $C_1=1$ (car $a\times b=(a\times b)$ et rien d'autre) et $C_0=1$ (considérant que $a=(a)$).

1. Énumérer à la main les parenthésages de $a\times b\times c\times d\times e$. Quelle est la valeur de C_4 ?
2. Montrer que la suite $(C_n)_{n\in\mathbb{N}}$ vérifie la relation de récurrence
$$C_n=\sum_{k=0}^{n-1}C_k\times C_{n-1-k} \tag{8}$$
   et calculer la valeur de C_5.
3. Est-ce une bonne idée d'écrire une fonction récursive qui calcule C_n à partir de cette formule ?
4. Écrire une fonction itérative `catalan(n)` qui retourne la liste de $n+1$ premiers nombres de Catalan (les nombres $C_0,\dots,C_n$). Dès le début la fonction initialise une liste de taille $n+1$ remplie de zéros, puis dans une boucle calcule un terme en fonction des précédents.

Exercice 7. *Mots bien parenthésés*
On s'intéresse aux expressions écrites uniquement avec des parenthèses ouvrantes et fermantes. Parmi celles-ci, certaines sont dites *bien parenthésées* quand les parenthèses sont correctement emboitées, par exemple `"((()()))"` ou bien `"()(())"`, mais d'autres sont dites *mal parenthésées* comme `")()("` ou bien `"(()()"`. L'algorithme pour tester si un mot est bien parenthésé est le suivant :

1. On initialise une variable p à 0,
2. On lit les caractères uns par uns,
3. À chaque parenthèse ouvrante on augmente p de 1, à chaque parenthèse fermante on diminue p de 1.

Alors :

1. Quelle(s) condition(s) peut-on dégager pour qu'une expression soit bien parenthésée ?
2. Écrire une fonction itérative `bien_parenthesee(s)` qui teste (renvoie un booléen `True` ou `False`) si l'expression `s` (une variable de type `str`) est une expression bien parenthésée. Si la chaine contient un caractère qui n'est pas une parenthèse (ni ouvrante ni fermante) on pourra tout simplement l'ignorer.

II Avec la tortue

On rappelle les commandes suivantes du module `turtle` :
- `import turtle` : charge le module `turtle` (une seule fois, hors de la fonction)
- `turtle.forward(n)` avance de `n` pas
- `turtle.backward(n)` : recule de `n` pas
- `turtle.left(t)` : tourne à gauche de l'angle `t` exprimé en degrés
- `turtle.right(t)` : idem, à droite
- `turtle.circle(r, t)` : trace un cercle, de rayon donné `r`, sur une portion d'angle `t` (`turtle.circle(r)` trace un cercle complet) ; le cercle est tracé dans le sens trigonométrique à partir de la position actuelle de la tortue.
- documentation officielle
- aide-mémoire

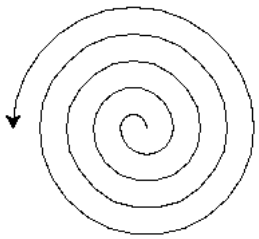
Ainsi un programme avec `turtle` se présentera idéalement sous la forme

```
###
import turtle

def f(...):
    ...

turtle.reset()
f(...)
# turtle.done()
###
```

Pour éviter les erreurs du type `turtle.Terminator` il est préférable de **ne pas fermer la fenêtre** et **enlever la ligne** `turtle.done()` ; la ligne `turtle.reset()` est alors bien nécessaire pour effacer le contenu de la fenêtre.
Cette section ne contient pas de fonctions récursives.

Exercice 8. Écrire une fonction `polygone(n)` qui utilise une boucle `for` et trace un polygone régulier à `n` côtés.

Exercice 9. Écrire une fonction `spirale(n)` qui trace une spirale régulière à `n` étapes :

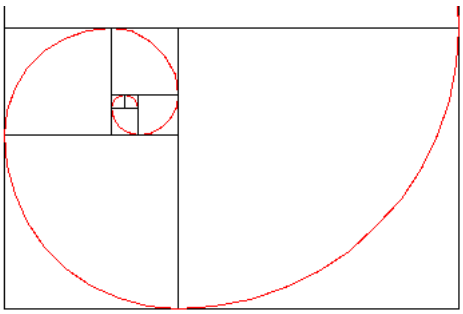


Il s'agit simplement d'une suite de demi-cercles dont le rayon augmente en progression arithmétique.
Amélioration : pouvez-vous reproduire ce dégradé de couleurs ?

Pour cela on change la couleur avec `turtle.color(r, g, b)` qui prend trois arguments qui sont des nombres entre 0 et 1, où r représente l'intensité du rouge, g du vert et b du bleu. Ainsi `(0, 0, 0)` correspond au noir et `(1, 0, 0)` au rouge, et les couleurs intermédiaires correspondent... aux valeurs intermédiaires. Pour un meilleur effet il faut aussi augmenter l'épaisseur du trait.

Remarque 2. On n'obtiendra pas exactement le même résultat selon l'orientation de la tortue au départ et selon le rayon de départ (on peut sans problème démarrer à 0 mais, trop petit, la première boucle se voit à peine et tout se passe comme si on avait démarré avec un rayon plus grand). Cela n'est cependant pas très important dans ce TP.

Exercice 10. *La spirale de Fibonacci*
Le but est de tracer la spirale suivante :



Si on y regarde bien, chaque carré a pour côté la somme des côtés des deux carrés précédents. Ainsi les longueurs successives des côtés forment une suite de Fibonacci, et la spirale est constituée de quarts de cercles dont les rayons sont les termes successifs d'une suite de Fibonacci (ou de ses multiples : un rayon 1 ou 2 se voit à peine sur l'écran). Écrire une fonction `spirale_fibonacci(n)` qui trace `n` étapes de cette spirale.
Amélioration : pouvez-vous aussi tracer les carrés ? Il suffit de tracer, avant l'arc de cercle de rayon r, un carré de côté r qui revient à son point de départ... On peut aussi tracer la spirale en une couleur et une épaisseur différente des carrés : il suffit de changer la couleur avant de tracer le carré, et ne pas oublier de la remettre pour tracer l'arc de cercle !