Lycée Les 3 Sources — **Bourg-Lès-Valence**
NSI T^le — Année **2023-24**

Test n°1

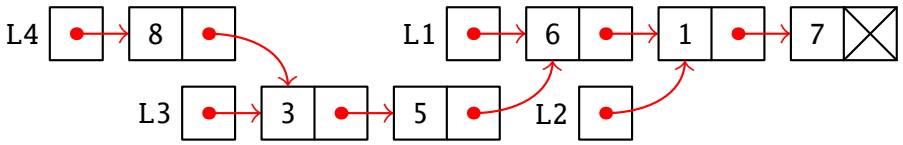
Nom et prénom :

Listes chaînées

Pour toute cette partie, on considère des listes chaînées, avec la liste vide notée `nil` et les fonctions suivantes :

Fonction	Description
`tete(liste)`	Renvoie la valeur du premier maillon de `liste`, qui ne doit pas être vide.
`queue(liste)`	Renvoie la liste sur laquelle pointe le premier maillon de `liste`, qui ne doit pas être vide.
`cons(valeur, liste)`	Renvoie une nouvelle liste correspondant à l'ajout de `valeur` en début de `liste`.
`est_vide(liste)`	Renvoie un booléen indiquant si `liste` est vide ou non.

Exercice 1 : (4pt) Le schéma suivant correspond à la représentation en mémoire des listes chaînées L1, L2, L3 et L4. Pour chacune des commandes suivantes, déterminer la réponse obtenue.



1) Pour chacune des commandes suivantes, déterminer la réponse obtenue.

```
>>> tete(L2)
1
>>> est_vide(queue(L1))
False
```

```
>>> tete(queue(queue(queue(L4))))
6
>>> est_vide(queue(queue(L2)))
True
```

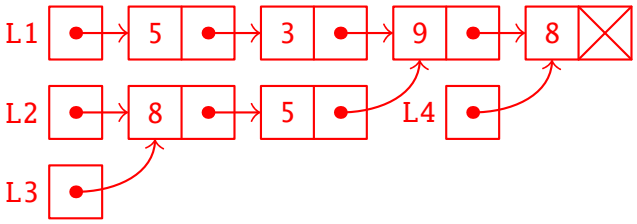
2) Donner les définitions des listes à l'aide des fonctions `cons` et `queue`. Il faut utiliser les listes précédentes pour définir les suivantes.

```
L1 = cons(6, cons(1, cons(7, nil)))
L2 = queue(L1)
L3 = cons(3, cons(5, L1))
L4 = cons(8, L3)
```

Exercice 2 : (2pt) Représenter l'état de la mémoire après les instructions suivantes. Vous pouvez vous inspirer du schéma de l'exercice précédent.

```
L1 = cons(5, cons(3, cons(9, cons(8, nil))))
L2 = cons(8, cons(5, queue(queue(L1))))
L3 = L2
L4 = queue(queue(queue(L2)))
```

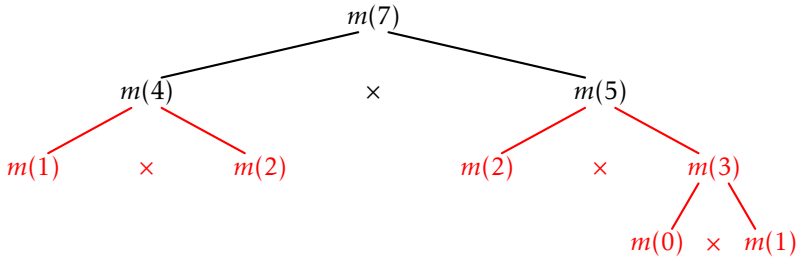
Solution :



Exercice 3 : (2pt) On considère la fonction suivante :

$$m(n) = \begin{cases} 2 & \text{si } n = 0 \\ 3 & \text{si } n = 1 \\ 5 & \text{si } n = 2 \\ m(n-3) \times m(n-2) & \text{si } n > 2 \end{cases}$$

1) Voici le début de l'arbre d'appel de $m(7)$ qui représente les appels récursifs et les opérations à faire pour obtenir le résultat. Compléter cet arbre.



2) Calculer la valeur de $m(7)$ en indiquant les valeurs intermédiaires.
Solution : On fait les calculs intermédiaires :

$m(0) = 2$
$m(1) = 3$
$m(2) = 5$
$m(3) = m(0) \times m(1) = 6$
$m(4) = m(1) \times m(2) = 15$
$m(5) = m(2) \times m(3) = 30$
$m(6) = m(3) \times m(4) = 90$
$m(7) = m(4) \times m(5) = 450$

Exercice 4 : (3pt) On considère la fonction ci-contre.

```
def mystere(n):
    if n == 0:
        return 1
    else:
        return n * mystere(n-1)
```

1) Quel est le résultat de `mystere(4)` ? 24
2) Décrire, en français, ce que fait cette fonction.
Solution : Cette fonction calcule le produit de tous les entiers de 1 à n, c'est-à-dire la factorielle de n.

$$\begin{aligned} \texttt{mystere(4)} &= 4 \times \texttt{mystere(3)} \\ &= 4 \times 3 \times \texttt{mystere(2)} \\ &= 4 \times 3 \times 2 \times \texttt{mystere(1)} \\ &= 4 \times 3 \times 2 \times 1 \times \texttt{mystere(0)} \\ &= 4 \times 3 \times 2 \times 1 \times 1 \\ &= 24 \end{aligned}$$

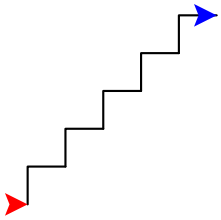
3) Écrire une version **itérative** de cette fonction. Vous pouvez utiliser **for** ou **while**. On rappelle que **range**(a, b) renvoie toutes les valeurs entières allant de a à b-1 inclus.

```
def mystere(n):
    res = 1
    for i in range(1, n+1):
        res = res * i
    return res
```

Exercice 5 : (2pt) On souhaite écrire une fonction récursive `compter(val, liste)` qui compte le nombre d'occurrences de `val` dans la **liste chaînée** `liste`.
Compléter le code de la fonction `compter(val, liste)` en Python en utilisant les fonctions sur les listes chaînées.

```
def compter(val, liste):
    if est_vide(liste):
        return 0
    elif val == tete(liste):
        return 1 + compter(val, queue(liste))
    else:
        return compter(val, queue(liste))
```

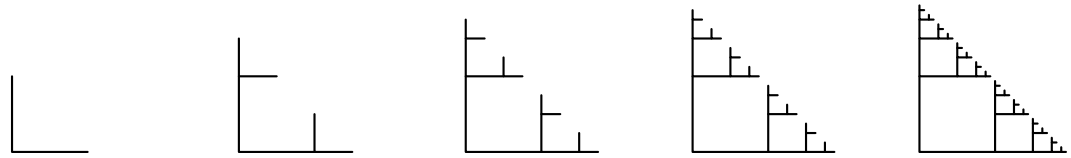
Exercice 6 : (2pt) On contrôle une tortue qui trace le chemin qu'elle parcourt à l'aide des commandes `haut(longueur)` et `droite(longueur)` qui la déplacent en haut et vers la droite de la longueur indiquée en paramètre. Compléter la fonction **récursive** `escalier(longueur, n)` qui trace un escalier de `n` marches dont la hauteur et la longueur est `longueur` chacune. La figure ci-contre correspond à `escalier(50, 5)`. Les deux triangles correspondent aux positions initiale et finale de la tortue. Vous ne pouvez pas utiliser de boucles **`for`** et **`while`**.



Conseil de maçon : *"S'il reste au moins une marche à faire, on la construit et on fait le reste de l'escalier."*

```
def escalier(l, n):
    if n > 0:
        haut(l)
        droite(l)
        escalier(l, n-1)
```

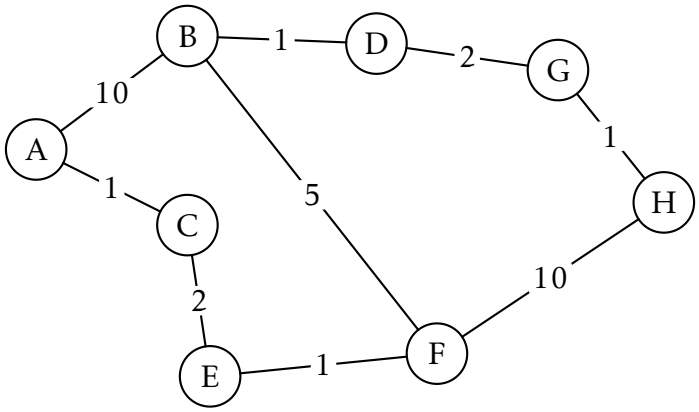
Exercice 7 : (2pt) Afin de tracer la fractale ci-dessous, on dispose d'une commande `segment(x1, y1, x2, y2)` qui trace un segment entre les points de coordonnées `(x1, y1)` et `(x2, y2)`. Écrire une fonction **récursive** `fractale(x, y, l, etape)` qui trace la fractale à l'étape `etape`, avec le point en bas à gauche qui a pour coordonnées `(x, y)` et avec les plus gros segments de longueurs `l`. Les extrémités des deux autres segments sont donc `(x+l, y)` et `(x, y+l)`. La longueur des segments est divisée par deux à chaque étape.




```
def fractale(x, y, l, e):
    segment(x, y, x+l, y)
    segment(x, y, x, y+l)
    if e > 0:
        fractale(x+l, y, l/2, e-1)
        fractale(x, y+l, l/2, e-1)
```

Routage

Exercice 8 : (4pt) On considère le réseau ci-dessous, où A, B, C, D, E, F, G et H sont des routeurs. Les nombres sur les liaisons sont les coûts de ces liaisons selon leur débit.



1) Compléter la table de routage de A selon le protocole RIP.

Routeur A (Selon RIP)		
Destination	Passerelle	Distance
B	B	1
C	C	1
D	B	2
E	C	2
F	B	2
G	B	3
H	B	3

2) Compléter la table de routage de A selon le protocole OSPF.

Routeur A (Selon OSPF)		
Destination	Passerelle	Coût
B	C	9
C	C	1
D	C	10
E	C	3
F	C	4
G	C	12
H	C	13