Nom, prénom, classe :

Examen de TP, fin du premier semestre

Ouvrez dans pyzo le fichier `TP_note_sem1.py` situé dans le répertoire `TP/TP_note_sem1_2021/` de votre répertoire habituel sur Gitlab. Après l'avoir ouvert, et **avant toute autre chose**, changez la valeur de la variable `numero_du_TP` et mettez-la à la valeur précisée ci-après :

```
numero_du_TP = 1
```

N'oubliez pas que l'exécution doit se faire via `Ctrl-SHIFT-E` pour que tout marche bien.

Il n'y a pas de fonction « interdite » dans ce TP, il faut juste que vous trouviez un moyen de réaliser les opérations demandées. Vous avez droit à votre exemplaire de AP1 ou à google si vous ne vous souvenez plus de la manière d'utiliser une fonction prédéfinie par Python.

De manière générale, vous avez le droit de chercher des idées dans vos fichiers de tous les TP ou les PP que vous avez déjà traités. Rien ne sera jamais directement copiable-collable, mais cela peut vous inspirer au besoin (à condition de savoir ce que vous voulez chercher).

IMPORTANT : n'oubliez pas de faire des commits réguliers pour sauvegarder vos avancées (en gros dès que vous avez du code qui marche à peu près).

Partie I — Construction de liste

Écrire une fonction `construction_de_liste(n)` qui prend en entrée un entier n et renvoie une liste constituée des n premières puissances de 2, à commencer par $2^1 = 2$ et jusqu'à 2^n.
Contraintes supplémentaires :

- Pour $n = 0$, votre fonction doit renvoyer la liste vide `[]`.
- Si n n'est pas un entier naturel, votre fonction doit renvoyer `None`.

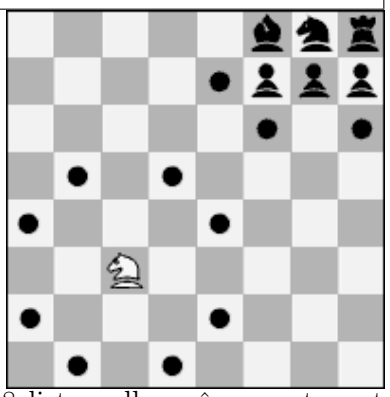
Partie II — Recherche dans une liste

Écrire une fonction `recherche_dans(liste)` qui recherche l'élément de la liste `liste` le plus proche de 11 par valeur inférieure. S'il n'y en a pas, la fonction doit renvoyer `None`. En particulier, on devra avoir

```
>>> recherche_dans([13.5, 11.5, 15.5, 9.5, 7.5])
9.5
>>> recherche_dans([9, 13, 7, 11, 5])
11
>>> recherche_dans([13, 10, 12, 9, 11])
11
>>> recherche_dans([14, 20, 18, 16, 12])
None
```

Partie III

Tempête sur l'échiquier



III.1 Introduction

Vous devez écrire un programme qui détermine, dans une partie d'échecs, si un cavalier peut prendre une pièce de l'adversaire. Rappelons que les échecs se jouent sur un plateau carré de 8 cases par 8. Un cavalier se déplace en « L », c'est-à-dire en avançant de deux cases horizontalement ou verticalement, puis en avançant d'une case à la perpendiculaire de la direction précédente. Le cavalier peut se déplacer même si les cases intermédiaires de son déplacement contiennent une pièce (amie ou ennemie). La figure ci-contre montre toutes les possibilités de déplacements de deux cavaliers.

L'échiquier vous sera à chaque fois fourni sous la forme d'une liste contenant 8 listes, elles-même contenant chacune 8 éléments représentatifs des pièces posées à cet endroit. Les lettres majuscules représentent les pièces blanches, les minuscules les pièces noires, et les `'.'` représentent les cases vides. Les cavaliers sont représentés par la lettre `'c'` (ou `'C'`), et les autres pièces par d'autres lettres de l'alphabet.

Pour faciliter vos explorations, vous disposez d'un echiquier d'exemple nommé `exemple_echiquier` que vous pouvez afficher à l'aide de la fonction `affiche_echiquier`, soit sous forme de liste de listes, soit sous forme plus compacte en ne gardant que les caractères adéquats.

```
>>> affiche_echiquier(exemple_echiquier)
[['t', 'c', '.', 'd', 'r', 'f', '.', 't'],
 ['p', 'p', 'p', '.', 'p', 'p', 'p', 'p'],
 ['.', '.', '.', 'p', '.', '.', '.', 'c'],
 ['.', '.', '.', '.', '.', 'f', '.', '.'],
 ['.', '.', 'C', '.', '.', 'P', '.', '.'],
 ['.', '.', 'P', '.', 'D', '.', 'P', '.'],
 ['P', 'P', '.', '.', '.', '.', '.', 'P'],
 ['T', '.', 'F', '.', 'R', 'F', 'C', 'T']]
>>> affiche_echiquier(exemple_echiquier,raw=True)
tc.drf.t
ppp.pppp
...p...c
.....f..
..C..P..
..P.D.P.
PP.....P
T.F.RFCT
```

III.2 Détection des cavaliers

Écrire une première fonction `detection_cavaliers` qui détecte la position de tous les cavaliers blancs présents sur le plateau. En particulier,

— s'il n'y en a pas, elle doit renvoyer `None` ;

— s'il y en a un ou plus [1], elle doit renvoyer une liste des doublets des positions [2] (sous forme `(ligne,colonne)`) de tous les cavaliers blancs présents sur l'échiquier.

1. NB : il peut y en avoir plus de deux du fait du phénomène de « promotion de pion » : quand un pion arrive au bout de l'échiquier, il peut se transformer en n'importe quelle pièce de sa couleur, notamment un troisième ou quatrième cavalier (il peut au maximum y avoir 10 cavaliers sur l'échiquier).

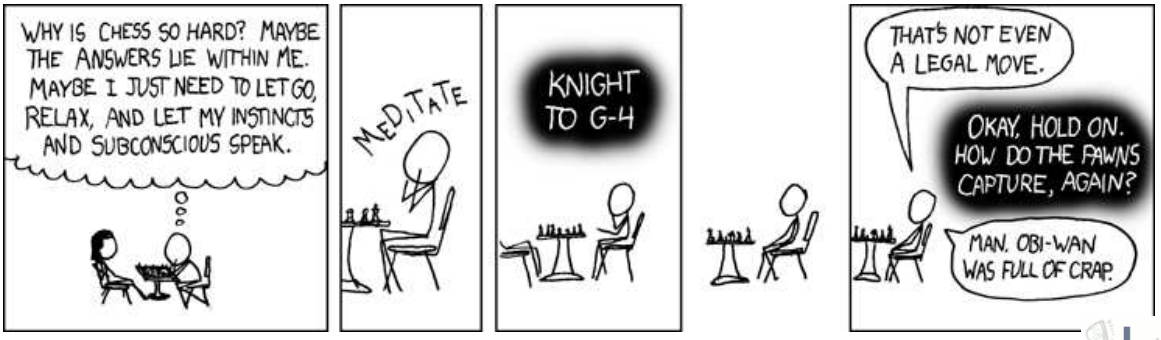
2. L'ordre dans lequel vous renvoyez les positions n'aura pas d'incidence dans les tests de ce TP.

Exemple :

```
>>> detection_cavaliers(exemple_echiquier)
[(4, 2), (7, 6)]
```

On rappelle à toutes fins utiles que les *positions* dans une liste de taille n commencent à 0 et finissent à $n-1$.

III.3 Positions possibles



`You know that 'sweep the pieces off the board and see it in your mind' thing? Doesn't work.`

Une fois la détection des cavaliers effectuée, écrire une fonction `deplacements_possibles` qui détermine, à partir d'une position donnée sous forme d'un doublet `(ligne,colonne)` la liste des positions accessibles[3] si le cavalier était à la position de départ indiquée. Par exemple

```
>>> position_initiale = (5, 2)
>>> deplacements_possibles(position_initiale)
[(3, 1), (3, 3), (4, 0), (4, 4), (6, 0), (6, 4), (7, 1), (7, 3)]
>>> position_initiale2 = (0, 6)
>>> deplacements_possibles(position_initiale2)
[(1, 4), (2, 5), (2, 7)]
```

Il n'y a pas besoin de donner l'échiquier en argument car on se contente de vérifier que le cavalier peut se déplacer sur le plateau à partir de la position donnée, indépendamment de la présence d'autres pièces aux points d'arrivée ou même de la présence effective d'un cavalier au point de départ[4].

III.4 Plat de résistance

Vous avez à présent toutes les armes pour répondre à la question posée. En particulier, votre dernière fonction `prise_possible` doit renvoyer :

- `True` s'il existe un cavalier blanc en position de prendre une pièce de l'équipe adverse[5] ;
- `False` si aucun cavalier blanc ne peut prendre de pièce adverse ;
- `None` s'il n'y a plus de cavalier blanc sur l'échiquier.

Exemple :

```
>>> prise_possible(exemple_echiquier)
True
```

3. L'ordre dans lequel vous donnez ces positions n'aura pas d'importance pour les tests de ce TP.
4. En d'autres termes : *Si* il y avait un cavalier à cet endroit, quels seraient toutes les cases *potentiellement* accessibles en l'absence de toute autre pièce.
5. Y compris le roi, même si aux échecs une telle possibilité n'existe pas.

Partie IV

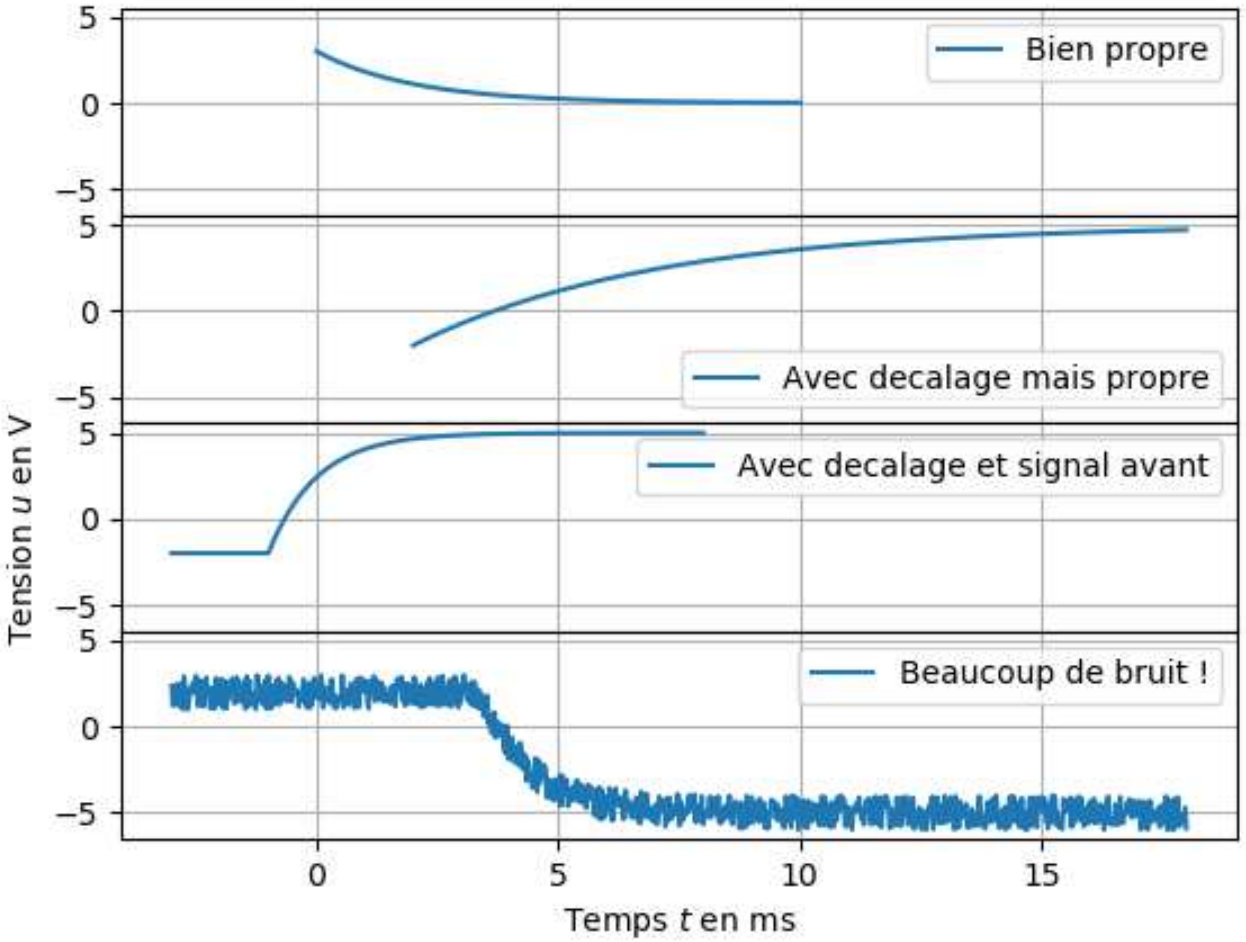
PythonPhysique

IV.1 Attendus

Vous disposez de diverses mesures de charges et de décharges de condensateurs, certaines ayant été prises avec soin (démarrage à $t = 0$, pas de bruit), d'autre beaucoup moins (démarrage à t quelconque et/ou signal bruité). Vous devez écrire une fonction `determine_tau(t,u)` qui, en ayant la liste des temps et tensions correspondantes, renvoie [6] une estimation du temps caractéristique τ du circuit [7].

IV.2 Exemples de courbes

Voici quelques exemples de courbes auxquelles sera confrontée votre fonction. Elle doit renvoyer le bon résultat (à 5% près) dans tous les cas. Commencez peut-être par traiter le cas simple puis essayez, à l'aide de fonctions annexes, de vous ramener au cas simple que vous savez traiter.



6. Attention, il s'agit bien d'utiliser un `return` et non un `print` comme dans les PP.

7. À toutes fins utiles, on rappelle que τ est notamment le temps au bout duquel l'écart entre le point de départ et l'asymptote a été divisé par exp(1). D'autres méthodes existent aussi (histoire de 3τ ou 5τ, tangente à l'origine, ajustement exponentiel), vous êtes libres d'utiliser celle qui vous sied le mieux.