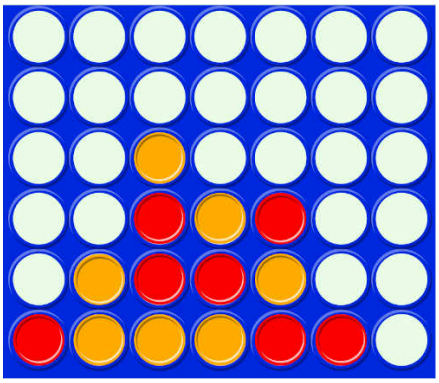
# Projet "Puissance 4"

## Présentation générale

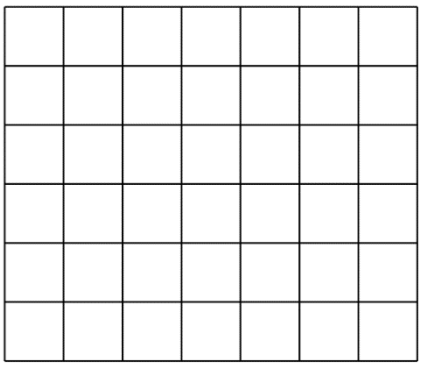
Voilà un projet qui met en jeu, comme beaucoup de choses en informatique, des *capacités d'abstraction* pour représenter en mémoire d'un ordinateur le plateau d'un jeu de société et permettre de jouer à ce jeu.
Un grand classique qu'on ne présente plus...Vous coderez un jeu à deux joueurs jouant successivement.
Dans un premier temps, la console sera utilisée pour afficher le plateau de jeu et pour interagir avec les utilisateurs.

```
Entrez la colonne joueur Jaune : 2

1  2  3  4  5  6  7
x  x  x  x  x  x  x
x  x  x  x  x  x  x
x  x  J  x  x  x  x
x  x  R  J  R  x  x
x  J  R  R  J  x  x
R  J  J  J  R  R  x
```

Dans un deuxième temps, une fois que le jeu fonctionnera, vous utiliserez une interface graphique pour pouvoir notamment contrôler le jeu à la souris : un clic sur une colonne de la grille doit faire tomber le pion du joueur, alternativement rouge ou jaune.



Le "plateau de jeu" peut être vu comme une *grille* de 6 lignes et 7 colonnes ; dans chaque "case" de cette grille peut être affichée un rond rouge ou jaune.



Ce plateau de jeu sera modélisé dans le script par un *tableau à 2 dimensions* de 6 lignes sur 7 colonnes, soit autant de "cases" qu'il y a dans la grille du jeu ; chaque élément de ce tableau peut contenir une valeur qui code le "contenu" de la "case" correspondante de la grille, par exemple :

- 0 = case vide
- 1 = pion rouge
- 2 = pion jaune

Ce tableau permettra de *garder en mémoire* l'état de la grille du jeu et de pouvoir gérer celle-ci, l'afficher, déterminer si un joueur gagne, etc...

```
grille = [
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 2, 0, 0, 0, 0],
    [0, 0, 1, 2, 1, 0, 0],
    [0, 2, 1, 1, 2, 0, 0],
    [1, 2, 2, 2, 1, 1, 0]]
```

Ce script demande à chaque joueur, à tour de rôle, d'indiquer la colonne (de 1 à 7) dans laquelle il souhaite placer un pion ; le script déterminera alors comment modifier le tableau pour représenter le coup joué.
A chaque pion joué, le script devra également mettre à jour l'affichage du plateau de jeu, et bien entendu déterminer si le coup est gagnant.

## Run1 : création de la grille de jeu et fonction d'affichage

Deux fonctions à coder, ce qui peut se faire en parallèle par deux personnes différentes.

- Ecrire une fonction genere_grille() qui crée le tableau représentant la grille du jeu, l'initialise et renvoie ce tableau.

```
def genere_grille()->list:
    '''Fonction qui initialise la grille du jeu
            Entrée : aucune
             Sortie : le tableau de tableaux représentant la grille du jeu
            initialisée
            '''
            ...

            return grille
```

- Ecrire une fonction affiche_grille() qui permet d'afficher le jeu dans la console en faisant apparaître le "contenu" de chaque "case" conformément aux informations stockées dans le tableau passé en paramètre.

```
def affiche_grille(grille: list):
    '''Fonction qui affiche la grille du jeu.
    '.' = case vide, 'R' = pion rouge, 'J' = pion jaune ( par exemple ).
    Les numéros de colonne de 1 à 7 doivent apparaître au-dessus de la grille.
    Entrée : grille = le tableau de tableau représentant la grille du jeu
    Sortie :aucune
            '''
    ...
```

Bien entendu, il faudra tester chacune de ces fonctions. A vous de réfléchir aux tests à réaliser.

## Run2 : un joueur joue

Après avoir réfléchi au problème, écrire ***sur papier*** l'algorithme à suivre pour savoir si une colonne est pleine, et si non, à déterminer le numéro de la ligne où un pion pourrait y être placé en "tombant" de puis le haut de la colonne. *Travailler sur un exemple de grille* pour bien comprendre le problème.

- Ecrire une fonction est_jouable() qui :
  - prend comme paramètre le tableau de tableau représentant la grille du jeu, et un entier représentant le numéro d'une colonne.
  - détermine si un pion peut être placé dans cette colonne (c'est à dire si cette colonne n'est pas pleine), et dans l'affirmative détermine le numéro de la ligne à laquelle le pion se placera en "tombant" depuis le haut de la grille.
  - renvoie ce numéro de ligne si le pion peut être placé, ou la valeur -1 si la colonne est pleine.

```
def est_jouable(grille: list, colonne: int)->int:
    """Fonction qui détermine si un pion peut être placé dans une colonne donnée.
        Entrées : grille = le tableau de tableau représentant la grille du jeu
                colonne = le numéro de la colonne où placer un pion
        Sorties :  le numéro de la ligne où le pion se placera, -1 si la colonne est
        pleine
    """
    ...
    return ligne
```

- Ecrire une fonction place_pion() qui prend comme paramètres :
  - le tableau de tableaux représentant la grille du jeu
  - le joueur qui souhaite placer un pion (information à coder par exemple avec un entier : 1 = joueur 1, 2 = joueur 2 ou autre... )
  - le numéro de colonne dans laquelle un joueur souhaite placer son pion.

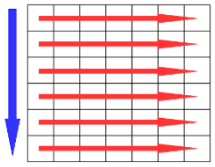
  Elle utilisera la fonction est_jouable() pour savoir si le coup est possible, et renverra le True ou False si le pion est placé ou non.

```
def place_pion(grille: list, joueur: int, colonne: int)->list:
    '''Fonction qui place le pion d'un joueur dans une case de la grille
        Entrée : grille = le tableau de tableaux représentant la grille du jeu
                joueur = le numéro du joueur
                colonne = le numéro de colonne dans laquelle le joueur
                souhaite placer un pion
        Sortie : True si le pion a pu être placé dans une colonne, False dans le cas
        contraire
    '''
```

## Run 3 : un joueur gagne ?

C'est le cœur du problème : comment le script va-t-il pouvoir déterminer si, après qu'un pion a été placé, un des joueurs est gagnant ?
Pour bien comprendre le problème, il faut le décomposer en sous-problèmes plus simples.
Il y a plusieurs méthodes pour traiter ces problèmes ; nous vous en proposons une ici, mais rien ne vous empêche de développer la vôtre, à condition bien sûr qu'il ne s'agisse pas d'un copier-coller depuis le web. Pensez à expliquer votre démarche !

**Vérification d'un alignement sur une ligne**
Nous utiliserons la méthode dite "par force brute", aussi dite "naïve", qui consiste à parcourir la grille ligne par ligne, et à compter, en partant du début de la ligne, le nombre de pions d'une couleur donnée placés les uns à la suite des autres.



- Ecrire une fonction alignement_ligne() qui permet de savoir si une suite de 4 pions d'une même couleur se trouve sur une ligne de la grille.

Cette fonction prend comme paramètres le tableau de tableaux représentant la grille du jeu, et renverra une information indiquant si un des joueurs est gagnant ou s'il n'y a pas de gagnant.

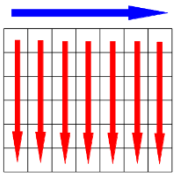
```
fonction alignement_ligne(grille):
        pour chaque ligne de la grille :
                nb_rouge ← 0  (nombre de pions rouges successifs sur la ligne)
                nb_jaune ← 0 (nombre de pions jaunes successifs sur la ligne)
                pour chaque colonne de la ligne :
                        si la case contient un pion ROUGE :
                                nb_rouge ← nb_rouge + 1
                                nb_jaune ← 0
                                si nb_rouge = 4:
                                        renvoyer gagnant = ROUGE
                        sinon si la case contient un pion JAUNE :
                                nb_jaune ← nb_jaune + 1
                                nb_rouge ← 0
                                si nb_jaune = 4:
                                        renvoyer gagnant = JAUNE
                        sinon (si la case est vide) :
                                nb_rouge ← 0
                                nb_jaune ← 0

        renvoyer Pas de gagnant !
```

```
def alignement_ligne(grille: list)->int :
    '''Fonction qui détermine si une suite de 4 pions d'une même couleur se
    trouve sur une des lignes de la grille.
        Entrée : grille = le tableau de tableau représentant la grille du jeu
        Sortie : une information (un entier par exemple) indiquant si un joueur
        est gagnant (1 ou 2 par exemple) ou s'il n'y a pas de gagnant (-1 par
        exemple)
    '''
    ...
```

**Vérification d'un alignement sur une colonne**
Le principe est le même mais le parcours devra se faire colonne par colonne, et la recherche sur une même colonne...

- Ecrire une fonction alignement_colonne() qui permet de savoir si une suite de 4 pions d'une même couleur se trouve sur une colonne de la grille.

Cette fonction prend comme paramètres le tableau de tableaux représentant la grille du jeu, et renverra une information indiquant si un des joueurs est gagnant ou s'il n'y a pas de gagnant.

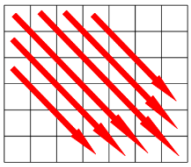


```
def alignement_colonne(grille: list)->int :
    '''Fonction qui détermine si une suite de 4 pions d'une même couleur se
    trouve sur une des colonnes de la grille.
        Entrée : grille = le tableau de tableau représentant la grille du jeu
        Sortie : une information (un entier par exemple) indiquant si un joueur
        est gagnant (1 ou 2 par exemple) ou s'il n'y a pas de gagnant (-1 par
        exemple)
    '''
    ...
```

**Vérification d'un alignement sur une diagonale**
Ah, là c'est un peu plus compliqué...Et il faut en plus distinguer les diagonales descendantes (de haut en bas vers la droite) des ascendantes (de bas en haut vers la droite) ...

**Alignement sur une diagonale descendantes**

L'idée : toujours avec l'approche "force brute", nous allons, en partant de leur case la plus en haut à gauche, parcourir les diagonales descendantes jusqu'en bas de la grille, et compter sur ces diagonales les nombres de pions successifs d'une même couleur en utilisant le même principe que précédemment.



```
fonction alignement_descendant(grille):
        pour chaque case de départ d'une diagonale de la matrice :
                nb_rouge ← 0
                nb_jaune ← 0
                x ← numéro de la colonne
                y ← numéro de la ligne
                tant qu'on n'est pas arrivé ni tout à droite ni tout en bas de la grille (tests à faire sur les valeurs de x et y ) :
                        compter les pions ROUGES et JAUNES successifs de la même manière que précédemment
                        modifier x et y pour passer à la case en bas à droite de la précédente

        renvoyer Pas de gagnant !
```

Remarque : certaines diagonales contiennent moins de 4 cases, ce n'est donc pas la peine de les parcourir !

- A partir d'un ***schéma sur papier***, déterminer les diagonales descendantes de la grille qui contiennent au moins 4 cases.
- Pour parcourir la grille, on choisit un parcours *par indice*, déterminer alors entre quelles valeurs doivent varier le numéro de ligne et le numéro de colonne
  Pour ne parcourir que les cases d'où partent des diagonales "à explorer" (c'est à dire qui contiennent au moins 4 cases); en déduire que 2 parcours différents sont alors nécessaires.

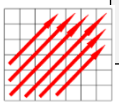
- Ecrire l'algorithme précédent en une fonction Python alignement_descendant() qui :
  - prend comme paramètre le tableau de tableau représentant la grille
  - renvoie une information indiquant si un joueur est gagnant ou s'il n'y a pas de gagnant.

```
def alignement_descendant(grille: list)->int :
    '''Fonction qui détermine si une suite de 4 pions d'une même couleur se trouve sur une des diagonales descendantes de la grille.
        Entrée : grille = le tableau de tableau représentant la grille du jeu
        Sortie : une information (un entier par exemple) indiquant si un joueur est gagnant (1 ou 2 par exemple) ou s'il n'y a pas de gagnant (-1 par exemple)
    '''
```

**Alignement sur une diagonale descendantes**

Enfin, il faut vérifier si un alignement de 4 pions d'une même couleur se trouve sur une diagonale ascendante de la grille.

- Ecrire une fonction Python alignement_ascendant() qui :
  - prend comme paramètre le tableau de tableau représentant la grille
  - renvoie une information indiquant si un joueur est gagnant ou s'il n'y a pas de gagnant.



```
def alignement_ascendant(grille: list)->int :
    '''Fonction qui détermine si une suite de 4 pions d'une même couleur se trouve sur une des diagonales ascendantes de la grille.
        Entrée : grille = le tableau de tableau représentant la grille du jeu
        Sortie : une information (un entier par exemple) indiquant si un joueur est gagnant (1 ou 2 par exemple) ou s'il n'y a pas de gagnant (-1 par exemple)
        '''
```

## Run4 : le jeu !

Vous avez maintenant les principales fonctions nécessaires pour gérer une partie entre deux joueurs humains…

- Ecrire l'algorithme à suivre pour, en utilisant les fonctions précédentes, simuler le déroulement d'une partie de puissance 4. Il peut être utile de créer des fonctions qui vérifie si la grille est pleine et qui détermine quel est le joueur gagnant
- En utilisant les fonctions écrites précédemment, compléter un script qui permet à deux joueurs de disputer une partie de Puissance 4.

## Run5 : interface graphique

Le jeu fonctionne ? Très bien ! Il serait sans doute plus convivial avec un affichage plus joli...
Dans le fichier « Puissance4_run5_tkinter.py » vous trouverez des fonctions qui utilisent le module Tkinter pour gérer l'affichage d'une grille et des pions d'un Puissance 4.
A vous de compléter ce script avec votre propre code en "l'intégrant" dans l'interface graphique.

- au lancement du script, la grille du jeu s'affiche, mais c'est tout !
- il est indispensable que votre programme soit placé dans la partie indiquée du script.
- vous disposez de deux fonctions déjà écrites :
    - une fonction dessin_jeu(grille) pour afficher la grille
    - une fonction clic(event) qui détermine le numéro de la colonne de la grille sur laquelle un joueur a cliqué et s'occupe de la gestion du jeu.

*Attention, cette dernière fonction ne peut pas être appelée directement par votre script* : elle est en réalité appelée *automatiquement* dès que Python détecte un clic souris.
Vous pouvez par contre très bien la compléter avec votre propre code !
La logique de cette interface graphique est donc un peu différente de la programmation Python telle que vous l'avez faite jusqu'à maintenant : on parle ici de *programmation évènementielle*, car certaines portions du code ne s'exécuteront que si certains évènements (ici, un clic-droit) se produisent.
On n'a donc plus vraiment cette exécution "linéaire" du script où les instructions s'exécutent les unes à la suite des autres dans l'ordre où elles ont été écrites.
Il faudra tenir compte de ce fait pour intégrer votre code à cette interface graphique.