But

Le but est d'écrire un code `python` permettant de coder un jeu simple utilisant une grille : morpion (3x3) (ou puissance4 (7x6) si vous êtes à l'aise).

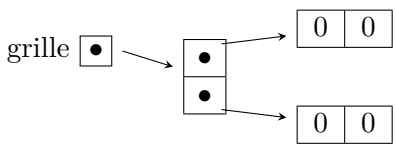
Le jeu

Le morpion (ou tic-tac-toe) se joue sur un damier 3×3 sur lequel chacun des deux joueurs place l'un après l'autre son symbole (un rond ou une croix). Le premier a avoir aligné son symbole gagne la partie.

Aide

◇ Le joueur A pourra être représenté par 'x' ou par 1 et le joueur B par 'o' ou par -1.
L'absence de coup par '-' ou 0.

◇ Une grille 2x2 peut être vue comme un tableau de 2 cases contenant chacune un tableau de 2 cases. On peut donc initialisé une grille carrée par le code suivant.




```
grille = 2*[0]
for i in range(len(grille)):  # len(grille) vaut 2
    grille[i] = 2*[0]
```

◇ Pour afficher la grille 2x2, on peut construire la fonction ci-contre

qui fera l'affichage ci-dessous pour une grille vide :

```
  A B
0 0 0
1 0 0
```

```
def affichage(map : list) -> None:
    lign = ' '
    for e in "AB" :
        lign = lign+' '+e
    print(lign)
    i = 0
    while i<len(map) :
        lign = f"{i:1}" # au moins 1 caractère
        for e in map[i] :
            lign = lign+' '+str(e)
        print(lign)
        i = i+1
```

◇ La fonction `ord(a)` permet d'afficher le code ASCII du caractère `a`.
La fonction `m.upper()` permet de renvoyer le caractère (ou la chaîne de caractères) `m` en majuscules.

```
ord('B')           # renvoie 66
ord('C')-ord('B')  # renvoie 1
'c'.upper()        # renvoie 'C'
'nsi'.upper()      # renvoie 'NSI'
```

Étapes (pour le morpion)

a) Choisir un binôme et réfléchir au découpage du jeu et à l'articulation de chaque partie.
Appeler votre professeur

b) Écrire une fonction `creer_grille() -> list`
qui créé une grille vide et la renvoie

c) Écrire une fonction `affiche_grille(g : list) -> None`
qui affiche la grille g.
Proposer un jeu de tests et appeler votre professeur.

d) Écrire une fonction `lettre2chiffre(lettre : str) -> int`
qui renvoie le nombre de lettres dans l'alphabet entre le caractère lettre et 'A'.
e.g. `lettre2chiffre('A')` renvoie l'entier 0 ; `lettre2chiffre('C')` renvoie l'entier 2.

e) Écrire une fonction `lecture_coordonnees(player : int) -> tuple`
qui demande, tant que les coordonnées ne sont pas dans la grille, au joueur `player` d'entrer les coordonnées où il souhaite jouer.
e.g. si le joueur A saisit "8D" un message lui demande de saisir de nouveau, s'il saisit "0A" la fonction renvoie le tuple (0, 0).
Rq : on pourra utiliser une boucle `while` avec une variable booléenne `bInvalid` qui est mise à `True` uniquement si les coordonnées sont valides.
Proposer un jeu de tests et appeler votre professeur.

f) Écrire une fonction `coup_possible(g : list, x : int, y : int) -> bool`
qui renvoie `True` si la case (x,y) est libre dans la grille g, `False` sinon.

g) Écrire une procédure `joueur_joue(g : list, player : int) -> None`
qui affiche la grille, demande au joueur `player` des coordonnées où jouer tant que celles-ci sont invalides ou déjà utilisées. Une fois les coordonnées correctes, la grille est alors mises à jour avec le coup du joueur.

h) Écrire une fonction `verif_ligneH(g : list, player : int, y : int) -> bool`
qui renvoie `True` si à partir de la case (x,y) une ligne horizontale est gagnante pour le joueur `palyer`, `False` sinon.

i) Écrire une fonction `verif_ligneV(g : list, player : int, x : int) -> bool`
idem pour une ligne verticale.

j) Écrire une fonction `verif_ligneD(g : list, player : int, x : int, y : int) -> bool`
idem pour les diagonales.

k) Écrire une fonction `joueur_gagne(g : list, player : int) -> bool`
qui renvoie `True` si le joueur `player` a gagné, `False` sinon

l) Écrire une fonction `main() -> None`
qui définit les variables `grille`, `player`, `nbCoups` et `aGagne`, puis qui coordonne chacune de ces fonctions afin de répondre au problème.
Proposer un jeu de tests et appeler votre professeur.

m) Approfondissement :

- ◇ prévoir une procédure `menu()` qui demande au joueur s'il veut jouer contre un autre joueur ou contre l'ordinateur. On pourrait aussi transformer `menu()` en une fonction qui demande le nom du joueur et le renvoie afin de personnaliser les affichages.
- ◇ prévoir une fonction `ordi_joue(g : list) -> tuple` fait jouer un ordinateur en tant que joueur2. À partir d'une grille g, la fonction renvoie un couple d'entiers (x, y) où l'ordinateur jouerait.
- ◇ améliorer l'intelligence artificielle de votre joueur ordinateur

Rendu

À la fin du temps imparti, chaque groupe rendra 2 travaux :

a) un compte-rendu (au maximum 2 pages) contenant l'organigramme du code général et l'organigramme d'une fonction, un résumé de votre projet, les jeux de tests effectués sur les fonctions (cette dernière partie pourra être contenue dans le fichier du code `python` sous forme de fonction ou de commentaires)

b) le fichier contenant le code `python` sera envoyé sous format numérique via l'ENT (ne pas oublier les spécifications des fonctions)