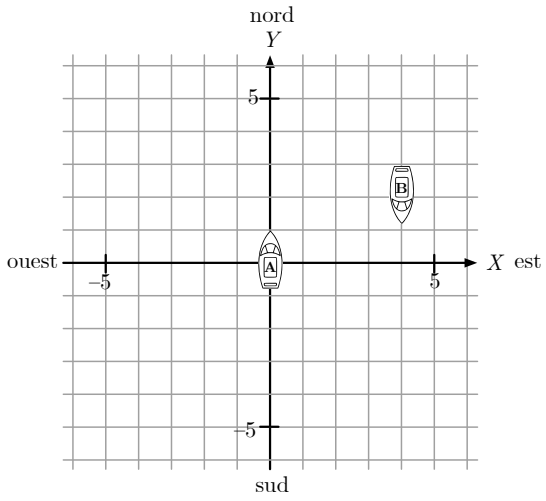
Problème 4. (12 points)

On décide de modéliser les mouvements d'un petit bateau robot dans un le plan avec une classe Python. Un tel bateau se déplace sur une grille comme illustré ci-dessous et a toujours des coordonnées x et y entières. Il peut être orienté vers le nord, l'est, le sud ou l'ouest. Son état initial est toujours en position $(0, 0)$, orienté vers le nord, comme montré ici avec le bateau **A** :



Le bateau **B**, dans le premier quadrant, est ici en position $(4, 2)$ et est orienté vers le sud.

a) Écrivez une **`dataclass`** pour modéliser la position et l'orientation du bateau. Pour chacun des champs que vous déclarez, indiquez (en commentaires) quelles valeurs ce champ pourra prendre selon la modélisation que vous choisissez.

```
@dataclass
class Boat:
```

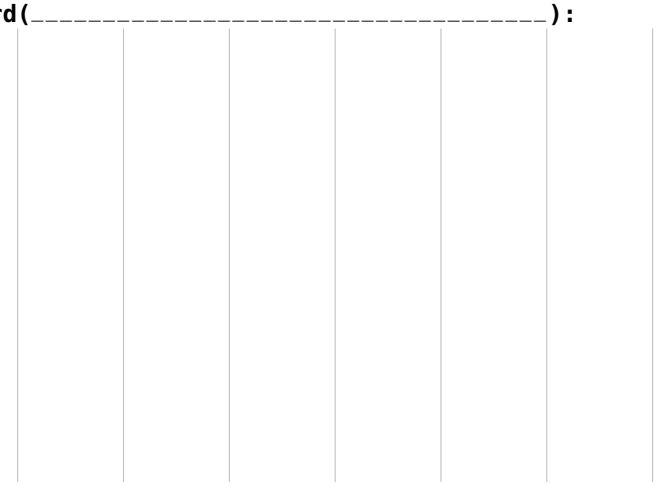
Le bateau sait faire deux types de mouvement : (1) avancer d'une unité dans la direction dans laquelle il est orienté ; (2) pivoter sur place de 90 degrés dans le sens des aiguilles d'une montre ou dans le sens contraire.

b) Sur la page suivante, complétez la classe **`Boat`** en ajoutant deux méthodes : (1) la première, **`forward`**, doit faire avancer le bateau d'un nombre entier d'unités passé en paramètre ; (2) la seconde, **`turn`**, doit le faire pivoter de 90 degrés dans le sens choisi par un paramètre booléen appelé **`clockwise`**. Indiquez également les types des paramètres.

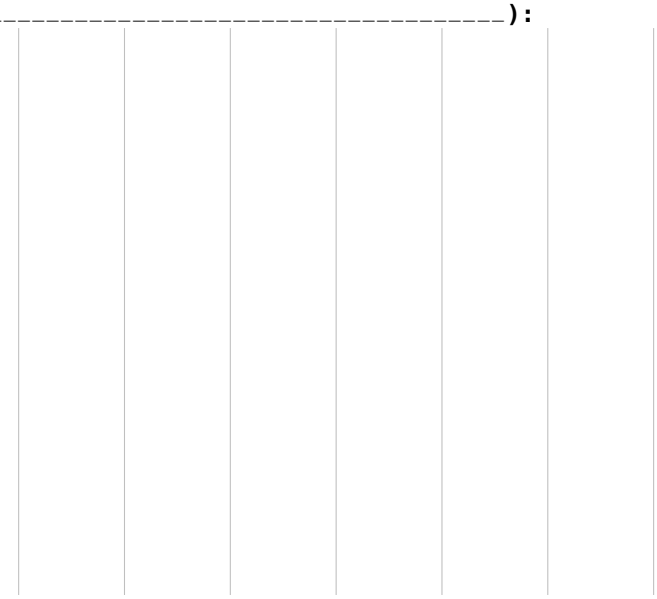
Complétez l'affection de la variable **`boat`** pour créer un bateau dans son état initial. Veillez à ce que vos méthodes soient définies de manière telle que les appels des lignes suivantes soient possibles et emmènent le bateau dans la position **B** du premier quadrant montrée dans la figure ci-dessus.

```
@dataclass
class Boat:
        # (les champs comme ci-dessus, puis...)

        def forward(_____________________________________):
```



```
        def turn(_____________________________________):
```



```
boat = _________________________________
boat.forward(3)
boat.turn(clockwise=True)
boat.forward(4)
boat.turn(clockwise=True)
boat.forward(1)
```

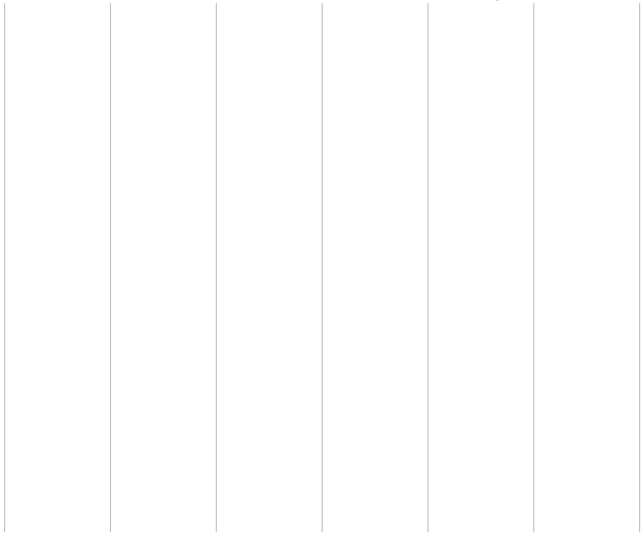
On cherche maintenant à piloter le bateau en lui donnant une série d'instructions, représentées par une chaîne de caractères dans laquelle chaque caractère est :
— soit `"F"` pour représenter une avancée d'une unité ;
— soit `"R"` pour représenter un pivotement dans le sens des aiguilles d'une montre ;
— soit `"L"` pour représenter un pivotement dans le sens contraire.

Par exemple, cette chaîne de caractères doit effectuer les mêmes mouvements que le code de l'extrait ci-dessus : **`"FFFRFFFFRF"`**.

c) Ajoutez à la classe **`Boat`** une méthode **`exec`** qui prend en paramètre une telle chaîne de caractères et exécute les mouvements spécifiés par les caractères individuels, en ignorant les éventuels caractères non valides.

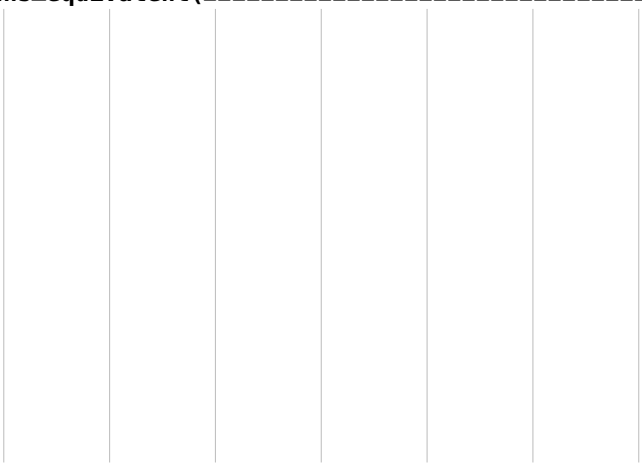
```
@dataclass
class Boat:
        # (les champs et méthode comme ci-dessus, puis...)

        def exec(_____________________________________):
```



d) Écrivez une fonction **`are_paths_equivalent`** (donc pas une méthode de la classe **`Boat`**) qui accepte deux chaînes d'instructions et qui renvoie **`True`** si les deux chaînes mènent le bateau au même endroit et que l'orientation finale est la même, et qui renvoie **`False`** sinon. Indiquez aussi son type de retour. Pour l'implémenter, vous pouvez faire appel à ce qui a été défini plus haut comme champs et méthodes ainsi qu'implémenter des fonctions auxiliaires — si cela vous aide, bien sûr.

```
def are_paths_equivalent(_____________________________________________) -> ____________:
```



e) Parfois, une chaîne d'instructions fait plus de détours que nécessaire pour arriver à sa destination. Écrivez une fonction **`optimize`** (aussi en dehors de la classe **`Boat`**) qui accepte une chaîne d'instructions et qui renvoie la chaîne d'instruction la plus courte possible qui permette au bateau d'arriver à la même position (sans se préoccuper de son orientation finale). Par exemple :
— l'appel **`optimize("FFFRFFFFRF")`** doit renvoyer **`"FFRFFFF"`** ;
— l'appel **`optimize("FLFLFLFLLRL")`** doit renvoyer **`""`** (aucun déplacement n'est nécessaire) ;
— l'appel **`optimize("RRFFF")`** peut renvoyer soit **`"RRFFF"`**, soit **`"LLFFF"`**.

Comme pour d), vous pouvez faire appel à du code défini plus haut et implémenter des fonctions auxiliaires. Comme indice, rappelez-vous que l'expression **`"x" * 3`** donne comme valeur **`"xxx"`**.

```
def optimize(__________________________) -> ___________:
```

Résultat :

a) (1 pt)	**b)** (3 pts)	**c)** (2 pts)	**d)** (2 pts)	**e)** (4 pts)	**Total** (12 pts)

(Laisser en blanc)