Q1: Confidence Interval

The bounds of a confidence interval CI are given by the formula:

$CI = \bar{x} \pm z\frac{s}{\sqrt{n}}$

where

- $\bar{x}$ is the sample mean
- z is the confidence level value (e.g., for a 95% confidence interval, z ≈ 1.96)
- s is the sample standard deviation
- n is the sample size

Define a function `ci` which takes each of the above as parameters in the given order, and prints the bounds of the confidence interval, or prints an error if the sample size is zero or negative.

Here are example runs:

```
>>> ci(100,1.96,10,5)
interval: [91.23461352820082, 108.76538647179918]

>>> ci(100,1.645,10,0)
Error: empty sample
```

Q2: Scrambling

Your task is to implement text scrambling and unscrambling. The scrambling algorithm works as follows: for every word in a text, keep the first and the last letter as they are, but alter the order of the letters in between.

Altering here means that you swap the first letter with the second, the third with the fourth, and so on. If there are an odd number of letters, the last letter remains in its original place.

Your scrambler must consist of three functions:

- `alter(s)`, which performs a swap for **all** pairs of adjacent letters of string `s`, as described in the second paragraph. (5pts)
- `scramble_word(s)`, which keeps the first and last letters of string `s` unchanged, and applies `alter` to the letters between them (5pts)
- `scramble_text(s)`, which splits the string `s` into words and applies `scramble_word` to each of them (you can assume that punctuation marks are separated by spaces and that words are separated by exactly one space) (5pts)

Example run:

```
>>> scramble_text("Your task is to implement a word scrambling algorithm
'Yuor tsak is to ipmelemnt a wrod srcmalbnig aglrotihm'
```

Then write a function `unscramble_text` which is the left inverse of the function `unscramble_text`. That is, for any valid string `s`, you have `unscramble_text(scramble_text(s)) == s`. You are free to implement this last function as you prefer. (5pts)

Q3: Offside Rules in Rugby

The objective of rugby is to move the ball forward (towards the goal zone).

Keeping track of who is onside or offside is critical to the game. Here is a simplified set of rules to determine if a player is offside:

- A player can either be onside or offside (never both at the same time)
- A player A is said to in front of player B if A is closer to the goal zone than B.
- If a player kicks the ball, tackles another player or is tackled, then all teammates in front of the player become offside.
- When an onside player ends up in front of an offside player, then this other player becomes onside again. This condition can be achieved by movement of either player.

A player which is offside can make no action other than moving backwards. **Otherwise it is foul play**.

Write a class `Player` which keeps track of the status of one player and update the offside status of its team as appropriate. Each `Player` object it should keep track of:

- whether the player is onside or offside
- where the player is on the pitch. For this position you will store a floating point number.

- additionally there should be an attribute which holds a list of all players in the team (including the player `self`).

A player can make either of the following actions:

- move (run or walk) on the pitch
- kick the ball
- catch the ball
- tackle a player (of the opposite team)

The `Player` class should have the following methods:

- `__init__(self)` which creates a player. The player is assumed to be onside and at position zero. The team attribute is initially an empty list, which is populated by the `initialise_team` function which you define later. (3 points)
- `move(self,dx)` move a the player by a distance `dx` on the pitch, in the direction of the goal zone. That is, a positive value of `dx` means moving forward, and a negative value means moving backwards. (In reality players can move side to side as well, but this does not matter when determining if they are offside or not.) (10 points)
- `tackling_or_kick(self)` Simulates tackling or kicking according to the ruleset. (5 points)
- `catch_ball(self)` Simulates catching the ball according to the ruleset. (2 points)

For each method, update the attributes of the current player *and other players in the team* according to the above ruleset. Print the message "Foul" in case of foul play. (For this exercise, you don't have to verify if a player has the ball or not; this is why tackling and kicking have the same implementation.)

Additionally, write a *function* `initialise_team(n)` which creates a list of `n` players in the same team and return this list. All players can be assume to start at position zero. Note: you will have to modify the `team` attribute of each player. (5 points)

Here is an example run:

```
>>> players = initialise_team(2) # a team of just 2 players for testing
>>> players[0].move(5) # player 0 is now in front
>>> players[1].tackling_or_kick() # the other player is tackled
>>> players[0].move(5) # player 0 keeps moving forward: this is a foul.
Foul
```