MY RECURSIVE APPROACH TO WOLF-CABBAGE-SHEEP

```
# Created by Kurtymittens (Kurt Russel Villamor)
```

"Due to the comment presented to my teammate "Mr. JEMUEL DE GUZMAN". I proceed to make a recursive approach"

ALGORITHM

1. Take Sheep
2. Return No Passenger
3. Take Wolf
4. Switch Wolf to Sheep
5. Return Sheep
6. Switch Cabbage and Sheep
7. Put Sheep

CODE

```
def crossRiver(orig, goto, boat, steps): # Accepts parameters
  if len(goto) == 3:
    return orig, goto
  else: # Brute Forcing Solutions
    if steps == 0 or steps == 3:
      if 'sheep' in orig:
        prioty = orig.index("sheep")
        goto.append(orig[prioty])
        orig.remove(orig[prioty])
    else:
      if len(orig) == 2:
        boat.append(orig.pop())
        tempBoat = boat.pop()
        temGoto = goto.pop()
        goto.append(tempBoat)
        boat.append(temGoto)
      else:
        tempBoat = boat.pop()
        temOrig = orig.pop()
        orig.append(tempBoat)
        goto.append(temOrig)
    return crossRiver(orig, goto, boat, steps+1)
```

This Brute Forcing allows me to eradicate tha use of loop. Note: It Works but still anxious while presenting my code. I think I can do more optimized code than this

INITIALIZATION

```
# I just focused in functional programming
orig = ["wolf", "sheep", "cabbage"] # Starting Position
destination = [] # End Position
boat = [] # Self explainatory BOAT
```

ITERATION PROGRESS

- **A creation of a function that put the name of the original place, the boat, and the Go to plcae of the passengers.**

```
# def crossRiver(a, b, c, d) <--- accepts 4 parameters for time face value
```

*FIRST ITERATION

- **First Iteration automatically get sheep in the current table or original table and put in the goto array to represent a movement**

```
#  crossRiver(orig, destination, boat, 0) this is the command

#if steps == 0 or steps == 3:
#    if 'sheep' in orig
#    goto.append(orig[prioty])
#    orig.remove(orig[prioty])
```

First Iteration Status:
orig = ["wolf", "sheep", "cabbage"] -> ["wolf", "cabbage"]
boat = [ ] -> [ ]
goto = [ ] -> ["sheep"]

*SECOND ITERATION

- **Second Iteration is the movement of the remaining passengers, either of wolf and cabbage has no constraints to each other.**

```
#      if len(orig) == 2:
#        boat.append(orig.pop())
#        tempBoat = boat.pop()
#        temGoto = goto.pop()
#        goto.append(tempBoat) # Switching
#        boat.append(temGoto) # Switching
```

2nd Iteration Status:

orig = ["wolf", "cabbage"] -> ["wolf"]

boat = [ ] -> ["Cabbage"]

goto = [ ] -> ["sheep"]

SWITCH

boat = ["Cabbage"] -> ["sheep"]

goto = [" "] -> ["Cabbage"]

*THIRD ITERATION

- **Third Iteration focuses in the swapping of individual places/slots, example is the singularity of the boat and a place can be either the orig place or the goto place**

```
#       tempBoat = boat.pop()
#       temOrig = orig.pop()
#       orig.append(tempBoat)
#       goto.append(temOrig)
```

3rd Iteration Status:

SWITCH

orig = ["wolf"] -> ["sheep"]

boat = ["sheep"] -> ["wolf"]

goto = ["cabbage"] -> ["cabbage"]

DROPPING

boat = ["wolf"] -> [ ]

goto = ["cabbage"] -> ["cabbage", "wolf"]

*4TH ITERATION

- **This iteration just repeats the first iteration where the sheep will be taken and dropped**

```
#if steps == 0 or steps == 3:
#    if 'sheep' in orig
#    goto.append(orig[prioty])
#    orig.remove(orig[prioty])
```

4th Iteration Status:

orig = ["sheep"] -> []

boat = [ ] -> [ ]

goto = ["cabbage", "wolf"] -> ["cabbage", "wolf" , "sheep"]

*SAMPLE OUTPUT

```
1 orig # Original State
```

```
['wolf', 'sheep', 'cabbage']
```

```
1 destination # Original State
```

```
[]
```

```
1 crossRiver(orig, destination, boat, 0) # Finish State
```

```
([], ['cabbage', 'wolf', 'sheep'])
```

```
1 orig1 = ["wolf", "sheep", "cabbage"] # Starting Position Different Order
2 destination1 = [] # End Position
3 boat1 = [] # Self explainatory BOAT
```

```
1 crossRiver(orig1, destination1, boat1, 0) # Printing the iterations
```

```
([], ['cabbage', 'wolf', 'sheep'])
```

```
1 # Created by Kurtymittens
```