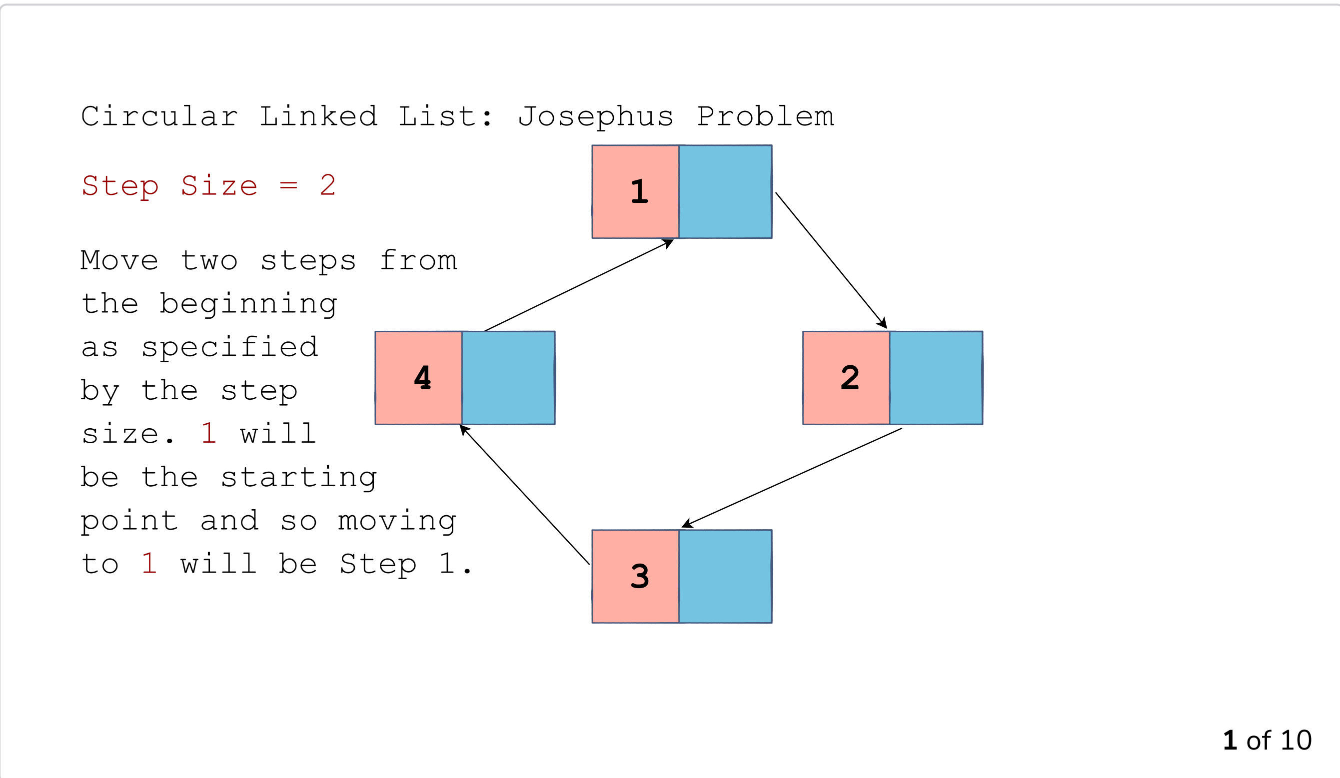
Josephus Problem

In this lesson, we will learn how to solve the Josephus Problem using a circular linked list in Python.

WE'LL COVER THE FOLLOWING

- Implementation
- Explanation

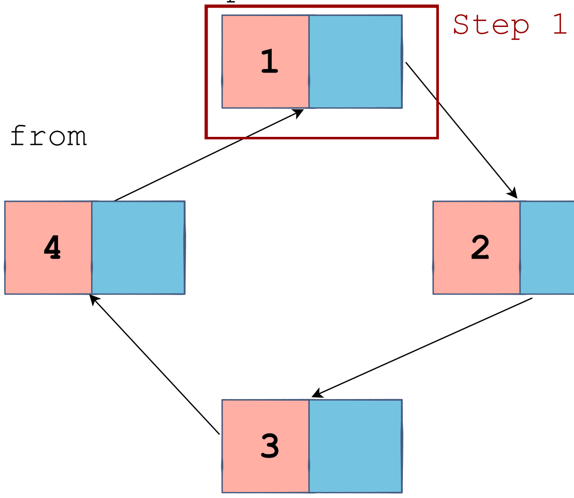
In this lesson, we investigate how to solve the "Josephus Problem" using the circular linked list data structure. Let's find out what the "Josephus Problem" is through an example in the illustration below:



Circular Linked List: Josephus Problem

Step Size = 2

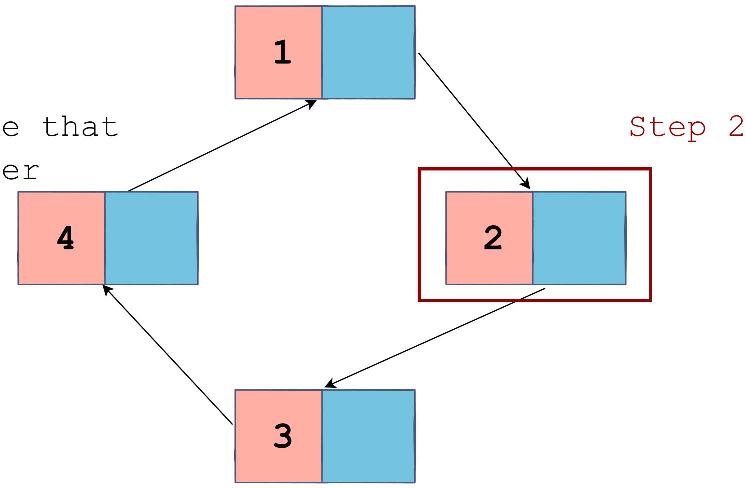
Move two steps from the beginning as specified by the step size



Circular Linked List: Josephus Problem

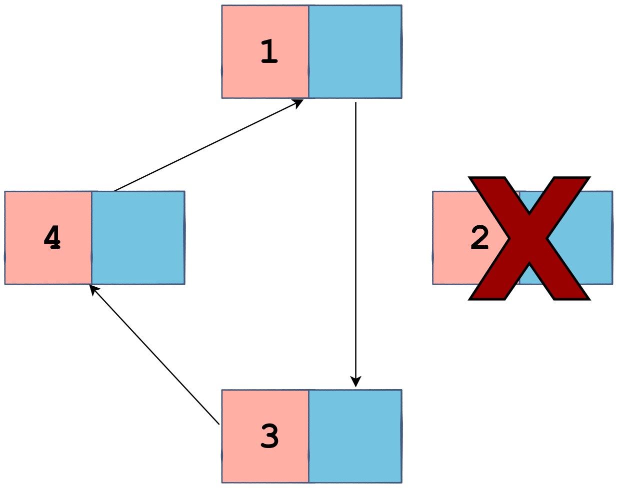
Step Size = 2

Remove the node that you are on after you have completed the steps



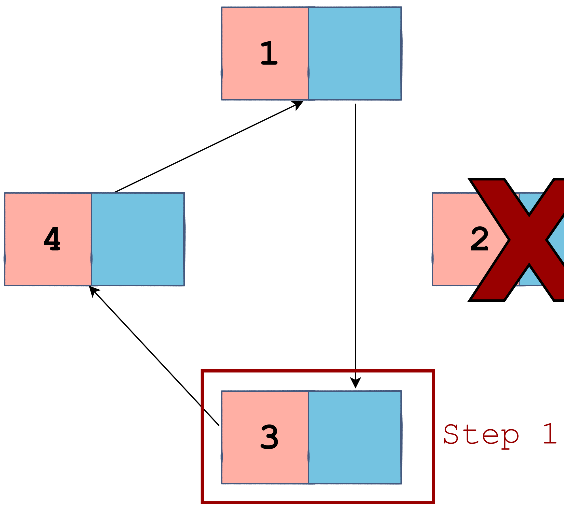
Circular Linked List: Josephus Problem

Step Size = 2



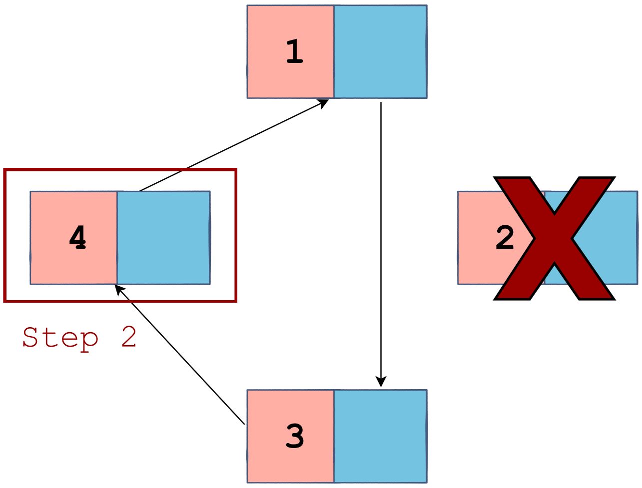
Circular Linked List: Josephus Problem

Step Size = 2



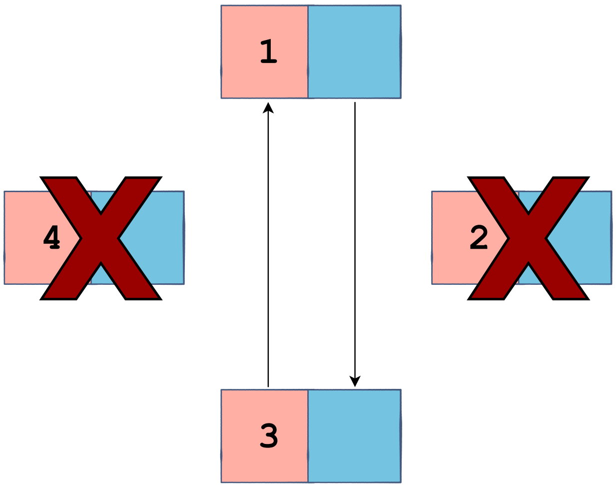
Circular Linked List: Josephus Problem

Step Size = 2



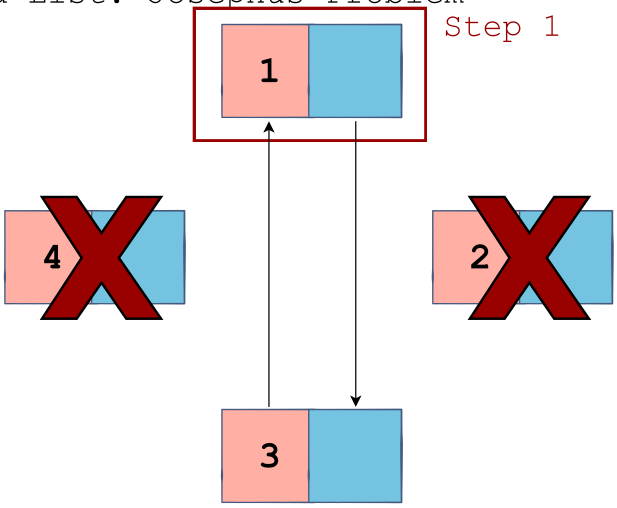
Circular Linked List: Josephus Problem

Step Size = 2



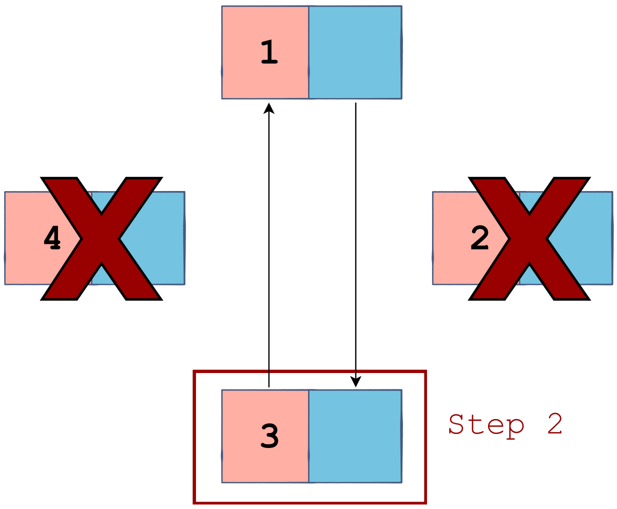
Circular Linked List: Josephus Problem

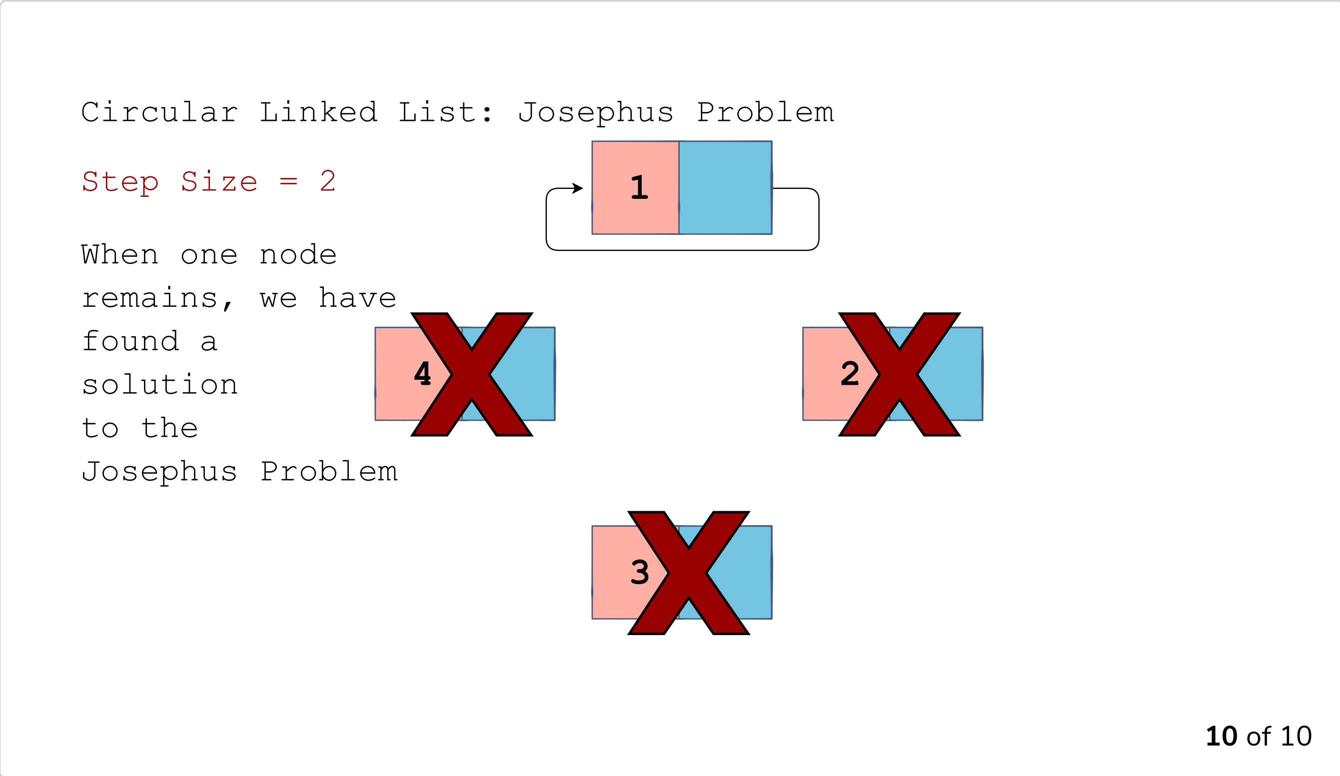
Step Size = 2



Circular Linked List: Josephus Problem

Step Size = 2





After having a look at the illustration, you'll hopefully understand the Josephus Problem. For this lesson, we have to count out the nodes from the linked list one by one according to the step size until one node remains. To solve this problem, we will tweak the `remove` method from one of the previous lessons so that we can remove nodes by passing the node itself instead of a key. To avoid confusion, we'll use the code from `remove` and paste it in a new method called `remove_node` with some minor modifications.

The modifications are as follows:

```
if self.head.data == key:
```

changes to

```
if self.head == node:
```

and

```
if cur.data == key:
```

changes to

```
if cur == node:
```

As you can see, instead of comparing the data of the node for a match, we compare the entire node itself.

You can check out the method below where the changed code is highlighted:

```
def remove_node(self, node):
  if self.head == node:
    cur = self.head
    while cur.next != self.head:
      cur = cur.next
    if self.head == self.head.next:
      self.head = None
    else:
      cur.next = self.head.next
      self.head = self.head.next
  else:
    cur = self.head
    prev = None
    while cur.next != self.head:
      prev = cur
      cur = cur.next
      if cur == node:
        prev.next = cur.next
        cur = cur.next
```

remove_node(self, node)

Implementation

Now as we're done with the `remove_node` method, let's go ahead and look at the solution for "Josephus Problem":

```
def josephus_circle(self, step):
  cur = self.head

  while len(self) > 1:
    count = 1
    while count != step:
      cur = cur.next
      count += 1
    print("KILL:" + str(cur.data))
    self.remove_node(cur)
    cur = cur.next
```

josephus_circle(self, step)

Explanation

`step` is passed as one of the arguments to the method `josephus_cirle`. On **line 2**, we initialize `cur` to `self.head` and set up a `while` loop on **line 4** that will keep running until the length of the linked list becomes `1`. We set `count` to `1` at the beginning of the iteration on **line 5**. Next, we have another nested `while` loop on **line 6** which will run until `count` is not equal to `step`. In this nested `while` loop, we move from node to node by updating `cur` to `cur.next` on **line 7**, and in each iteration, we increment the `count` by `1`. As soon as `count` becomes equal to `step`, the `while` loop breaks, and the execution jumps to **line 9**. On **line 9**, we print the node that we land on, so you can visualize the nodes that we will remove. In the next line, we remove the node (`cur`) as the `while` loop ended with that being the current node. On **line 11**, we update `cur` to `cur.next` to repeat the entire process until we are left with one node which will break the `while` loop on **line 4**.

Yes, the solution is as simple as that. You can play around with the entire code in the widget below, which contains all the code that we have implemented for this chapter so far.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class CircularLinkedList:
    def __init__(self):
        self.head = None

    def prepend(self, data):
        new_node = Node(data)
        cur = self.head
        new_node.next = self.head

        if not self.head:
            new_node.next = new_node
        else:
            while cur.next != self.head:
                cur = cur.next
            cur.next = new_node
        self.head = new_node

    def append(self, data):
        if not self.head:
            self.head = Node(data)
            self.head.next = self.head
        else:
            new_node = Node(data)
```

```
        cur = self.head
        while cur.next != self.head:
            cur = cur.next

        cur.next = new_node
        new_node.next = self.head

def print_list(self):
    cur = self.head

    while cur:
        print(cur.data)
        cur = cur.next
        if cur == self.head:
            break

def __len__(self):
    cur = self.head
    count = 0
    while cur:
        count += 1
        cur = cur.next
        if cur == self.head:
            break
    return count

def split_list(self):
    size = len(self)

    if size == 0:
        return None
    if size == 1:
        return self.head

    mid = size//2
    count = 0

    prev = None
    cur = self.head

    while cur and count < mid:
        count += 1
        prev = cur
        cur = cur.next
    prev.next = self.head

    split_cllist = CircularLinkedList()
    while cur.next != self.head:
        split_cllist.append(cur.data)
        cur = cur.next
    split_cllist.append(cur.data)

    self.print_list()
    print("\n")
    split_cllist.print_list()

def remove(self, key):
    if self.head:
        if self.head.data == key:
            cur = self.head
            while cur.next != self.head:
                cur = cur.next
            if self.head == self.head.next:
```

```
                self.head = None
            else:
                cur.next = self.head.next
                self.head = self.head.next
        else:
            cur = self.head
            prev = None
            while cur.next != self.head:
                prev = cur
                cur = cur.next
                if cur.data == key:
                    prev.next = cur.next
                    cur = cur.next

    def remove_node(self, node):
        if self.head:
            if self.head == node:
                cur = self.head
                while cur.next != self.head:
                    cur = cur.next
                if self.head == self.head.next:
                    self.head = None
                else:
                    cur.next = self.head.next
                    self.head = self.head.next
            else:
                cur = self.head
                prev = None
                while cur.next != self.head:
                    prev = cur
                    cur = cur.next
                    if cur == node:
                        prev.next = cur.next
                        cur = cur.next

    def josephus_circle(self, step):
        cur = self.head

        while len(self) > 1:
            count = 1
            while count != step:
                cur = cur.next
                count += 1
            print("KILL:" + str(cur.data))
            self.remove_node(cur)
            cur = cur.next


cllist = CircularLinkedList()
cllist.append(1)
cllist.append(2)
cllist.append(3)
cllist.append(4)


cllist.josephus_circle(2)
cllist.print_list()
```

By now, you will hopefully be familiar with the circular linked lists and challenges related to it. We have an exercise prepared for you in the next lesson!