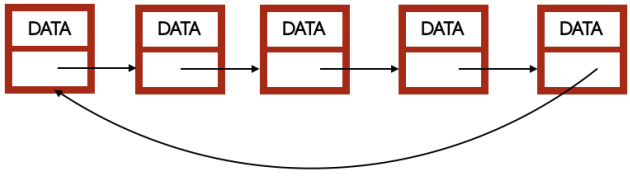
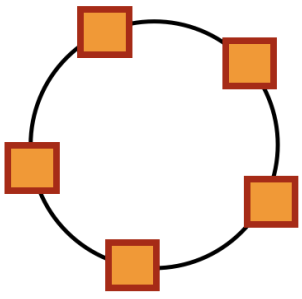
[redacted] (/)

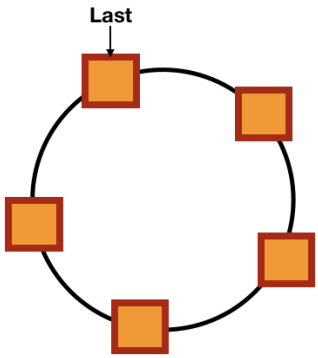
Circular Linked Lists

A **Circular Linked List** is just a simple linked list with the *next* pointer of the last node of a linked list connected to the first node.





As you can see in the above picture that there is no start and end of a circular linked list, but we will maintain a pointer to represent the last element of the linked list as shown in the picture given below.



We have chosen to keep a record of the last element and not the first because inserting a new node after the last node will be efficient in this case instead of traversing over the entire list in the case of keeping the record of the first element.

There are also many applications of a circular linked list like:

- In a multiplayer game, a circular linked list can store the players and give each of them change one by one.
- Computers also need to share their resources with each running application turn by turn and this is done with circular linked list.
- Circular Linked Lists are also used to develop other data structures like Queue.

Now, we know about a circular linked list, so let's start making one.

Coding Up a Circular Linked List

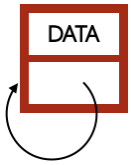
We will start by making a function to initialize and return a circular linked list - `INIT_CIRCULAR_LINKED_LIST(key)`.

key is the data which we are going to store in the first node of the linked list. So, we will start by making a new node and setting its data equal to the key.

```
z = new node
z.data = key
```

Now we have a node and to make it a circular linked list, we will point its *next* pointer to itself.

```
z.next = z
```



This is our last node as well as the first node, so we will initialize a new circular linked list and mark it as the last node and return it.

```
c = new circular_linked_list
c.last = z
return c
```

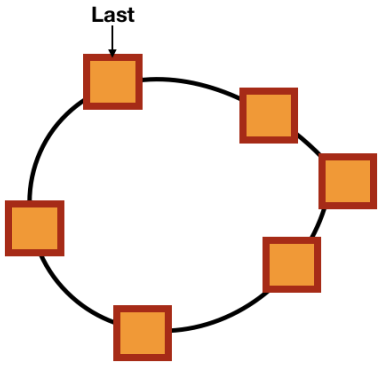
```
INIT_CIRCULAR_LINKED_LIST(key)
    z = new node
    z.data = key
    z.next = z

    c = new circular_linked_list
    c.last = z
    return c
```

As we have initialized our circular linked list, let's move ahead and discuss about inserting and deleting nodes from it.

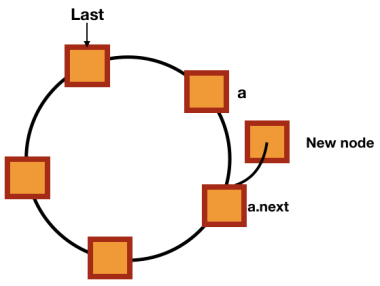
Inserting a new node in Circular Linked List

To insert a new node at any position, we will have to break the existing link and add the new node at that position.

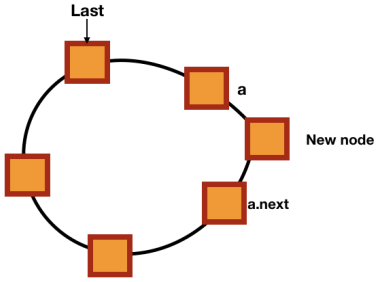


So, let's make a function which will take the new node (n) and the node after which we are going to insert this node (a) - `INSERT_AFTER(n, a)`.

The *next* of the new node will point to *next* of the node a i.e., `n.next = a.next`.



After this, we will point *next* of the node *a* to the new node *n* - `a.next = n`.

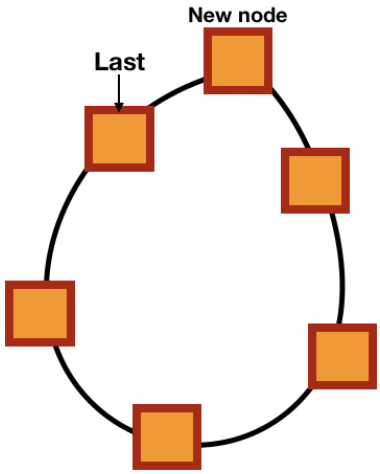



```
INSERT_AFTER(n, a)
    n.next = a.next
    a.next = n
```

Suppose we want to insert a node after the last node. We can simply use the `INSERT_AFTER` function to do so but what if someone wants to make this newly added node the last node of the linked list? We will make a different function for this case.

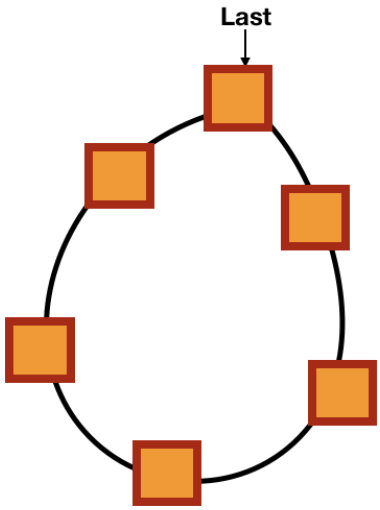
We will pass the linked list (L) and the new node (n) to the function i.e., `INSERT_AT_LAST(L, n)`.

We will simply insert the node by connecting the new node to the first node and the last node to the new node.




```
n.next = L.last.next
L.last.next = n
```

At last, we will update the last pointer of the linked list.




```
L.last = n
```

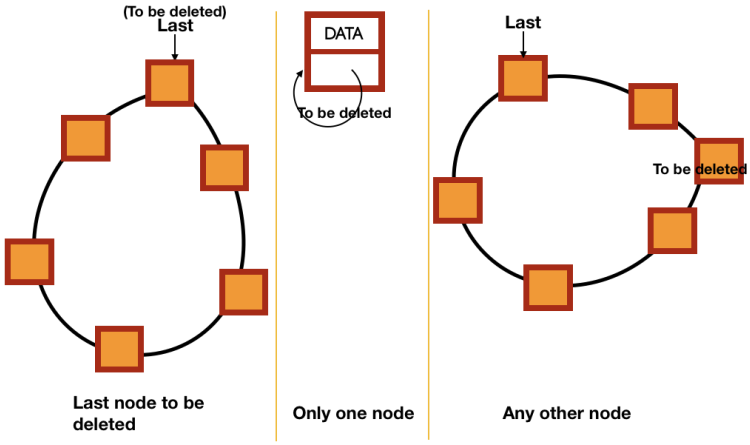
```
INSERT_AT_LAST(L, n)
     n.next = L.last.next
     L.last.next = n
     L.last = n
```

Deleting a Node from Circular Linked List.

Let's start writing our function by passing the node to be deleted (n) and the linked list (L) to it i.e., `DELETE(L, n)`.

We will first make a temporary pointer to the last node of the linked list i.e., `temp = L.last`. After this, we will check if the node to be deleted is the last node or not. If it is the last node, then we will have to update the last pointer. In the case of the last node, it is also possible that the node is the only node of

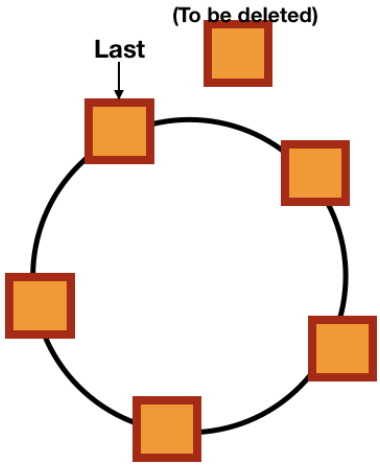
the linked list, we will handle this case differently.



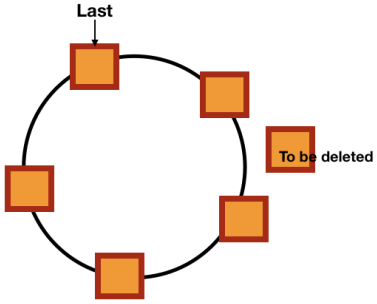

```
if n == l.last //last node
    if n.next == n: //only one node
        l.last = NULL
    else: //more than one node and last node
        temp.next = n.next
        l.last = temp //updating last pointer
```

When the linked list has only one node (`if n.next == n`), we are making it null - `l.last = NULL` .

If the linked list has more than one node and the node to be deleted is the last node, we are pointing the *next* pointer of the node previous to the node to be deleted to the next of it - `temp.next = n.next` . At last, we are updating the *last* pointer - `l.last = temp` .



When the node to be deleted is not the last node, then we don't have to update the last pointer. We will just connect the previous node to the next node of the node which we are going to delete.




```
temp.next = n.next
```

```
DELETE(L, n)
     temp = L.last
     while temp.next != n
          temp = temp.next

     if n == L.last //last node
          if n.next == n //only one node
               L.last = NULL
          else //more than one node and last node
               temp.next = n.next
               L.last = temp //updating last pointer
     else //not last node
          temp.next = n.next
```

C **Python** **Java**

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
  int data;
  struct node *next;
}node;

typedef struct circular_list {
  struct node *last;
}circular_linked_list;

//to make new node
node* new_node(int data) {
  node *z;
  z = malloc(sizeof(struct node));
  z->data = data;
  z->next = NULL;

  return z;
}

circular_linked_list* init_circular_linked_list(int key) {
  node *z;
  z = new_node(key);
  z->next = z;

  circular_linked_list *c = malloc(sizeof(circular_linked_list));
  c->last = z;
  return c;
}

void insert_after(node *n, node *a) {
  n->next = a->next;
  a->next = n;
}

void insert_at_last(circular_linked_list *l, node *n) {
  n->next = l->last->next;
  l->last->next = n;
  l->last = n;
}

void delete(circular_linked_list *l, node *n) {
  node *temp = l->last;
  while(temp->next != n) {
    temp = temp->next;
  }
  if(n == l->last) { //last node
    if(n->next == n) { //only one node
      l->last = NULL;
    }
    else {//more than one node and last node
      temp->next = n->next;
      l->last = temp; //updating last pointer
    }
  }
  else {//not last node
    temp->next = n->next;
  }
  free(n);
}

void traversal(circular_linked_list *l) {
  node *temp = l->last;
  printf("%d\t",temp->data);
  temp = temp->next;

  while(temp != l->last) {
    printf("%d\t",temp->data);
    temp = temp->next;
  }
  printf("\n");
}

int main() {
  circular_linked_list *l = init_circular_linked_list(10);
```