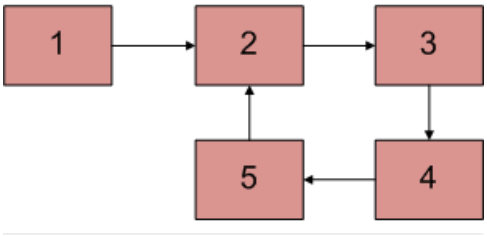
## Write a program function to detect loop in a linked list

Given a linked list, check if the the linked list has loop or not. Below diagram shows a linked list with a loop.




Following are different ways of doing this

**Use Hashing:**

Traverse the list one by one and keep putting the node addresses in a Hash Table. At any point, if NULL is reached then return false and if next of current node points to any of the previously stored nodes in Hash then return true.

**Mark Visited Nodes:**

This solution requires modifications to basic linked list data structure. Have a visited flag with each node. Traverse the linked list and keep marking visited nodes. If you see a visited node again then there is a loop. This solution works in O(n) but requires additional information with each node.

A variation of this solution that doesn't require modification to basic data structure can be implemented using hash. Just store the addresses of visited nodes in a hash and if you see an address that already exists in hash then there is a loop.

**Floyd's Cycle-Finding Algorithm:**

This is the fastest method. Traverse linked list using two pointers. Move one pointer by one and other pointer by two. If these pointers meet at some node then there is a loop. If pointers do not meet then linked list doesn't have loop.

**Implementation of Floyd's Cycle-Finding Algorithm:**

### C/C++

```
// C program to detect loop in a linked list
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
    int data;
    struct node* next;
};

void push(struct node** head_ref, int new_data)
{
    /* allocate node */
    struct node* new_node =
          (struct node*) malloc(sizeof(struct node));

    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

int detectloop(struct node *list)
{
    struct node  *slow_p = list, *fast_p = list;

    while (slow_p && fast_p && fast_p->next )
    {
        slow_p = slow_p->next;
        fast_p  = fast_p->next->next;
        if (slow_p == fast_p)
        {
           printf("Found Loop");
           return 1;
        }
    }
    return 0;
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct node* head = NULL;

    push(&head, 20);
    push(&head, 4);
    push(&head, 15);
    push(&head, 10);

    /* Create a loop for testing */
    head->next->next->next->next = head;
    detectloop(head);

    return 0;
}
```

### Java

```
// Java program to detect loop in a linked list
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }

    /* Inserts a new Node at front of the list. */
    public void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }

    int detectLoop()
    {
        Node slow_p = head, fast_p = head;
        while (slow_p != null && fast_p != null && fast_p.next != null) {
            slow_p = slow_p.next;
            fast_p = fast_p.next.next;
            if (slow_p == fast_p) {
                System.out.println("Found loop");
                return 1;
            }
        }
        return 0;
    }

    /* Drier program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist = new LinkedList();

        llist.push(20);
        llist.push(4);
        llist.push(15);
        llist.push(10);

        /*Create loop for testing */
        llist.head.next.next.next.next = llist.head;

        llist.detectLoop();
    }
}
/* This code is contributed by Rajat Mishra. */
```

### Python

```
# Python program to detect loop in the linked list

# Node class
class Node:

    # Constructor to initialize the node object
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:

    # Function to initialize head
    def __init__(self):
        self.head = None

    # Function to insert a new node at the beginning
    def push(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node

    # Utility function to prit the linked LinkedList
    def printList(self):
        temp = self.head
        while(temp):
            print temp.data,
            temp = temp.next


    def detectLoop(self):
        slow_p = self.head
        fast_p = self.head
        while(slow_p and fast_p and fast_p.next):
            slow_p = slow_p.next
            fast_p = fast_p.next.next
            if slow_p == fast_p:
                print "Found Loop"
                return

# Driver program for testing
llist = LinkedList()
llist.push(20)
llist.push(4)
llist.push(15)
llist.push(10)

# Create a loop for testing
llist.head.next.next.next.next = llist.head
llist.detectLoop()

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
Found loop
```

**Time Complexity:** O(n)
**Auxiliary Space:** O(1)

**References:**
http://en.wikipedia.org/wiki/Cycle_detection
http://ostermiller.org/find_loop_singly_linked_list.html