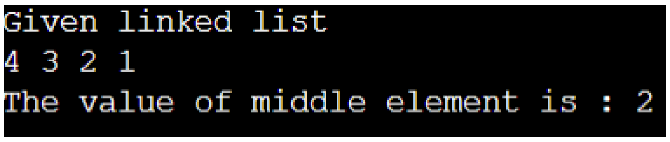
Input: head = [1,2,3,4,5,6]
Output: 4
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

Solution : Code : LP_Code7.java

Output :



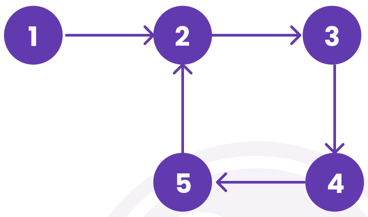
```
Given linked list
4 3 2 1
The value of middle element is : 2
```

Approach :

- Traverse linked list using two-pointers. Move one pointer by one and the other pointers by two.
- When the fast pointer reaches the end, the slow pointer will reach the middle of the linked list.

Cycle detection in a linked list :

Question : Given a linked list, check if the linked list has a loop or not. The below diagram shows a linked list with a loop.

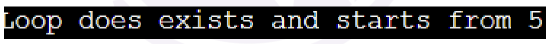


$$\begin{tikzcd} 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[d] \\ & 5 \arrow[u] & 4 \arrow[l] \end{tikzcd}$$

Solution :

Code : LP_Code8.java

Output :



```
Loop does exists and starts from 5
```

Approach :

- We will be using Floyd's cycle finding algorithm.
- Floyd's cycle finding algorithm or Hare-Tortoise algorithm is a pointer algorithm that uses only two pointers, moving through the sequence at different speeds. This algorithm is used to find a loop in a linked list. It uses two pointers, one moving twice as fast as the other one. The faster one is called the faster pointer and the other one is called the slow pointer.
- While traversing the linked list one of these things will occur-
  - The Fast pointer may reach the end (NULL) this shows that there is no loop in the linked list.
  - The Fast pointer again catches the slow pointer at some time therefore a loop exists in the linked list.

Proof of floyd's cycle finding algorithm:

Let,
X = Distance between the head(starting) to the loop starting point.
Y = Distance between the loop starting point and the first meeting point of both the pointers.
C = The distance of the loop

So before both the pointer meets-

The slow pointer has traveled $X + Y + s * C$ distance, where s is any positive constant number.

The fast pointer has traveled $X + Y + f * C$ distance, where f is any positive constant number.Since the fast pointer is moving twice as fast as the slow pointer, we can say that the fast pointer covered twice the distance the slow pointer covered. Therefore-

$X + Y + f * C = 2 * (X + Y + s * C)$

$X + Y = f * C - 2 * s * C$

We can say that,

$f * C - 2 * s * C = (\text{some integer}) * C = K * C$

Thus,

$X + Y = K * C \quad - (1)$

$X = K * C - Y \quad - (2)$

Where K is some positive constant.

Now if reset the slow pointer to the head(starting position) and move both fast and slow pointer by one unit at a time, one can observe from 1st and 2nd equation that both of them will meet after traveling X distance at the starting of the loop because after resetting the slow pointer and moving it X distance, at the same time from loop meeting point the fast pointer will also travel $K * C - Y$ distance(because it already has traveled Y distance).

From equation (2) one can say that $X = K * C - Y$ therefore, both the pointers will travel the distance X i.e. same distance after the pink node at some point to meet at the starting point of the cycle.

Here, by some point, it means that the fast pointer can complete the $K * C$ distance out of which it has already covered the Y distance.

Conclusively, we find that slow pointer moves 1 step at a time while fast pointer moves 2 steps at a time so the fast pointer catches or surpasses slow pointer if there is a loop, just like start < end in Binary Search. Else if there exists no loop then fast pointer reaches the NULL before than slow.