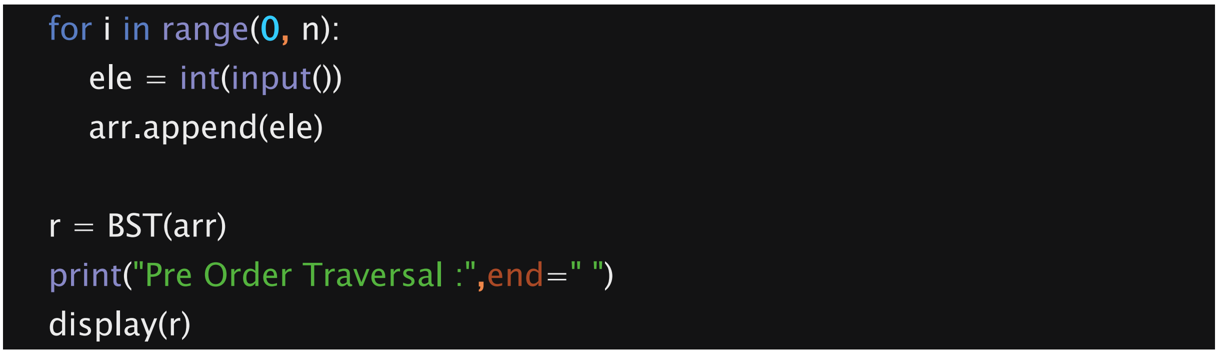
S Abhishek AM.EN.U4CSE19147

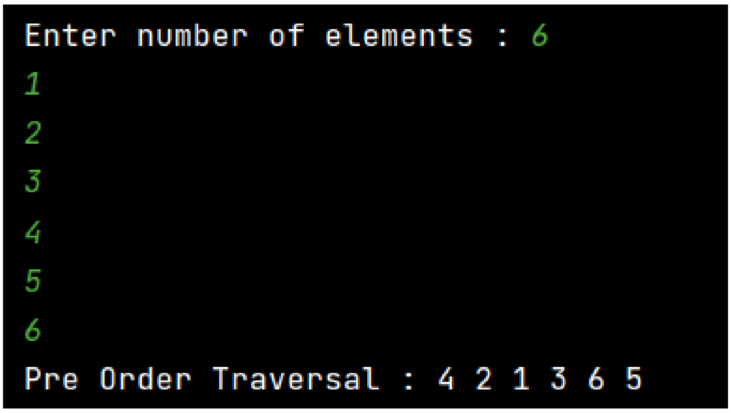
Data Structures

1. Given a sorted (increasing order) array with unique integer elements, implement an algorithm to create a binary search tree with minimum height.

```
class Node(object):
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

def BST(arr):
    if not arr:
        return None
    root = len(arr) // 2
    node = Node(arr[root])
    node.left = BST(arr[:root])
    node.right = BST(arr[root + 1:])
    return node

def display(node):
    if not node:
        return
    print(node.val,end= " ")
    display(node.left)
    display(node.right)

if __name__=='__main__':

    arr=[]
    n = int(input("Enter number of elements : "))
```

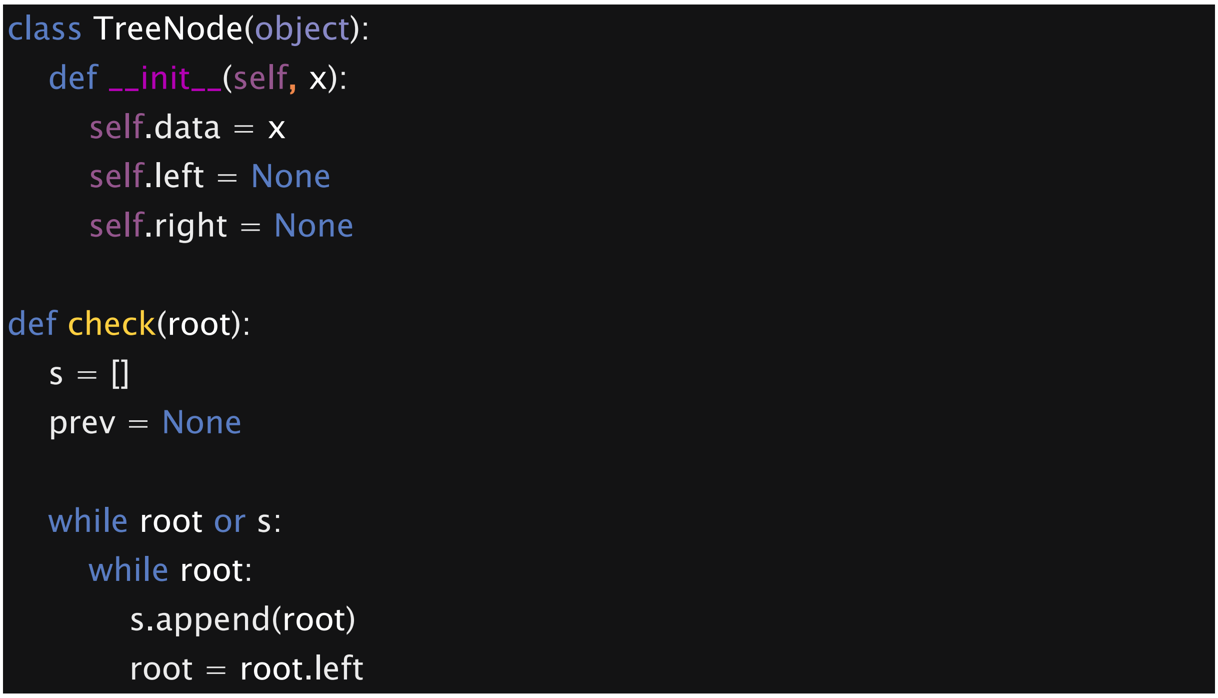


```
    for i in range(0, n):
        ele = int(input())
        arr.append(ele)

    r = BST(arr)
    print("Pre Order Traversal :",end=" ")
    display(r)
```

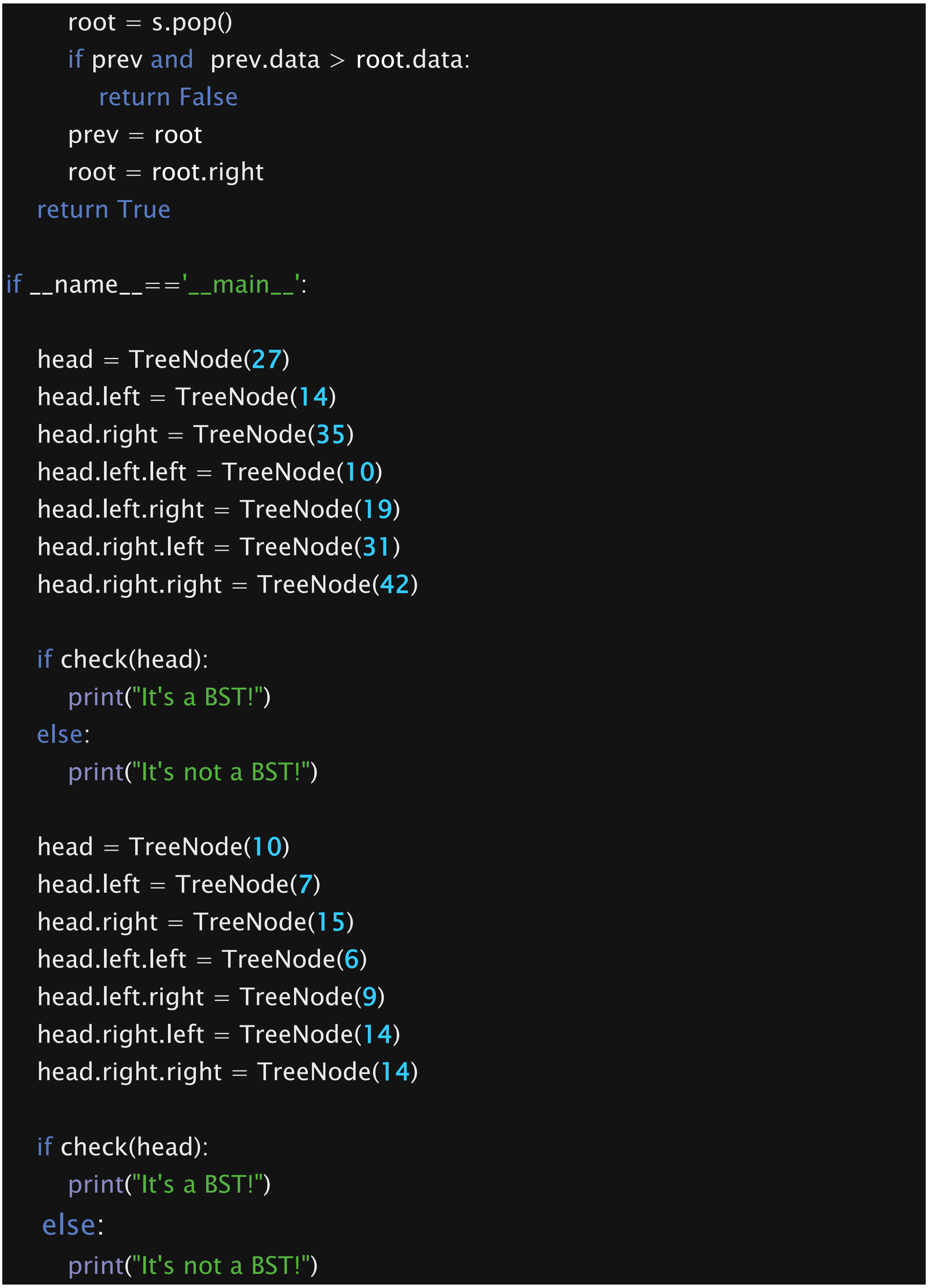


```
Enter number of elements : 6
1
2
3
4
5
6
Pre Order Traversal : 4 2 1 3 6 5
```

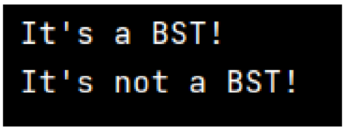
2. Implement a function to check if a binary tree is a BST or not.



```
class TreeNode(object):
    def __init__(self, x):
        self.data = x
        self.left = None
        self.right = None

def check(root):
    s = []
    prev = None

    while root or s:
        while root:
            s.append(root)
            root = root.left
```



```python
        root = s.pop()
        if prev and  prev.data > root.data:
            return False
        prev = root
        root = root.right
    return True

if __name__=='__main__':

    head = TreeNode(27)
    head.left = TreeNode(14)
    head.right = TreeNode(35)
    head.left.left = TreeNode(10)
    head.left.right = TreeNode(19)
    head.right.left = TreeNode(31)
    head.right.right = TreeNode(42)

    if check(head):
        print("It's a BST!")
    else:
        print("It's not a BST!")

    head = TreeNode(10)
    head.left = TreeNode(7)
    head.right = TreeNode(15)
    head.left.left = TreeNode(6)
    head.left.right = TreeNode(9)
    head.right.left = TreeNode(14)
    head.right.right = TreeNode(14)

    if check(head):
        print("It's a BST!")
    else:
        print("It's not a BST!")
```

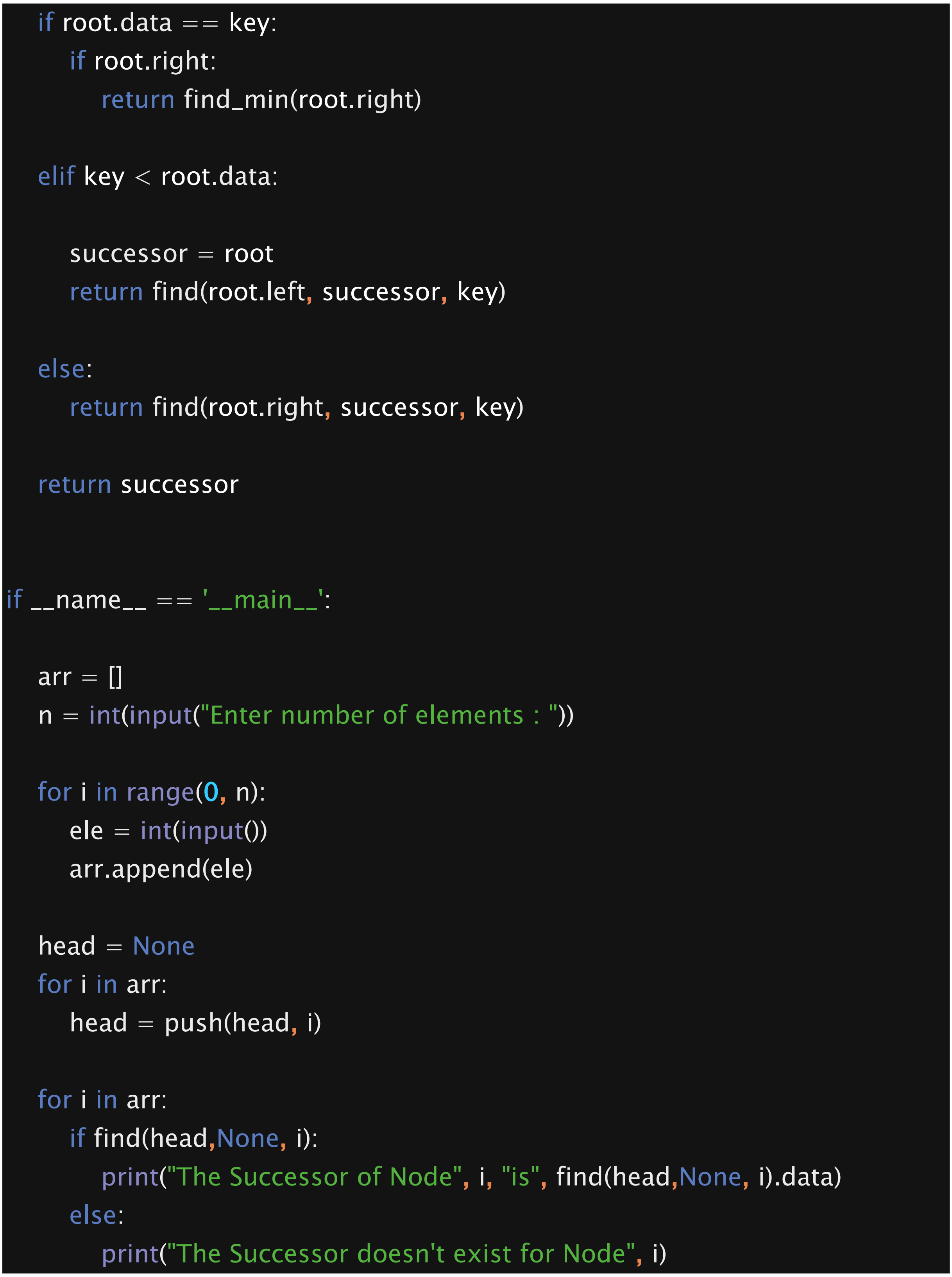


```
It's a BST!
It's not a BST!
```

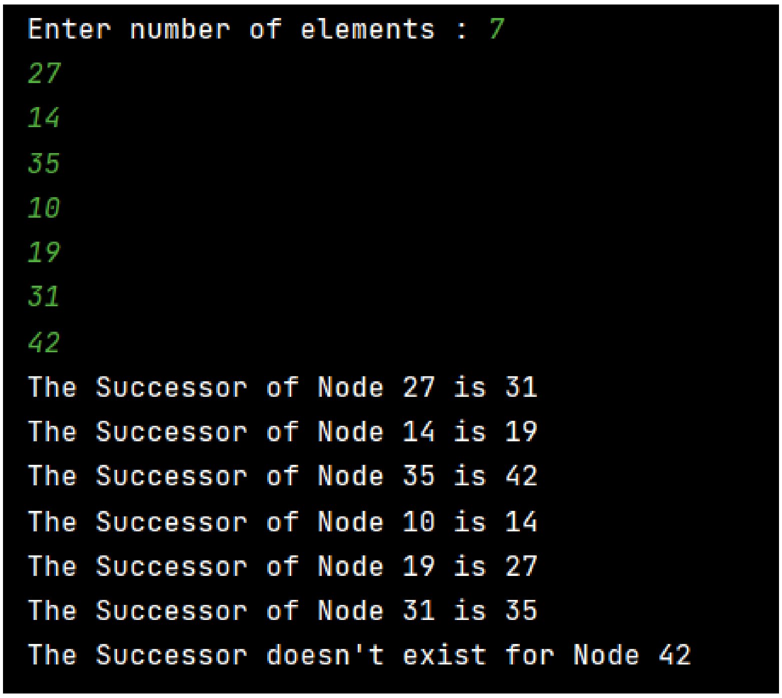
3. Implement a function to find the in-order successor of a given node in a BST.

You may assume that each node has a link to its parent.

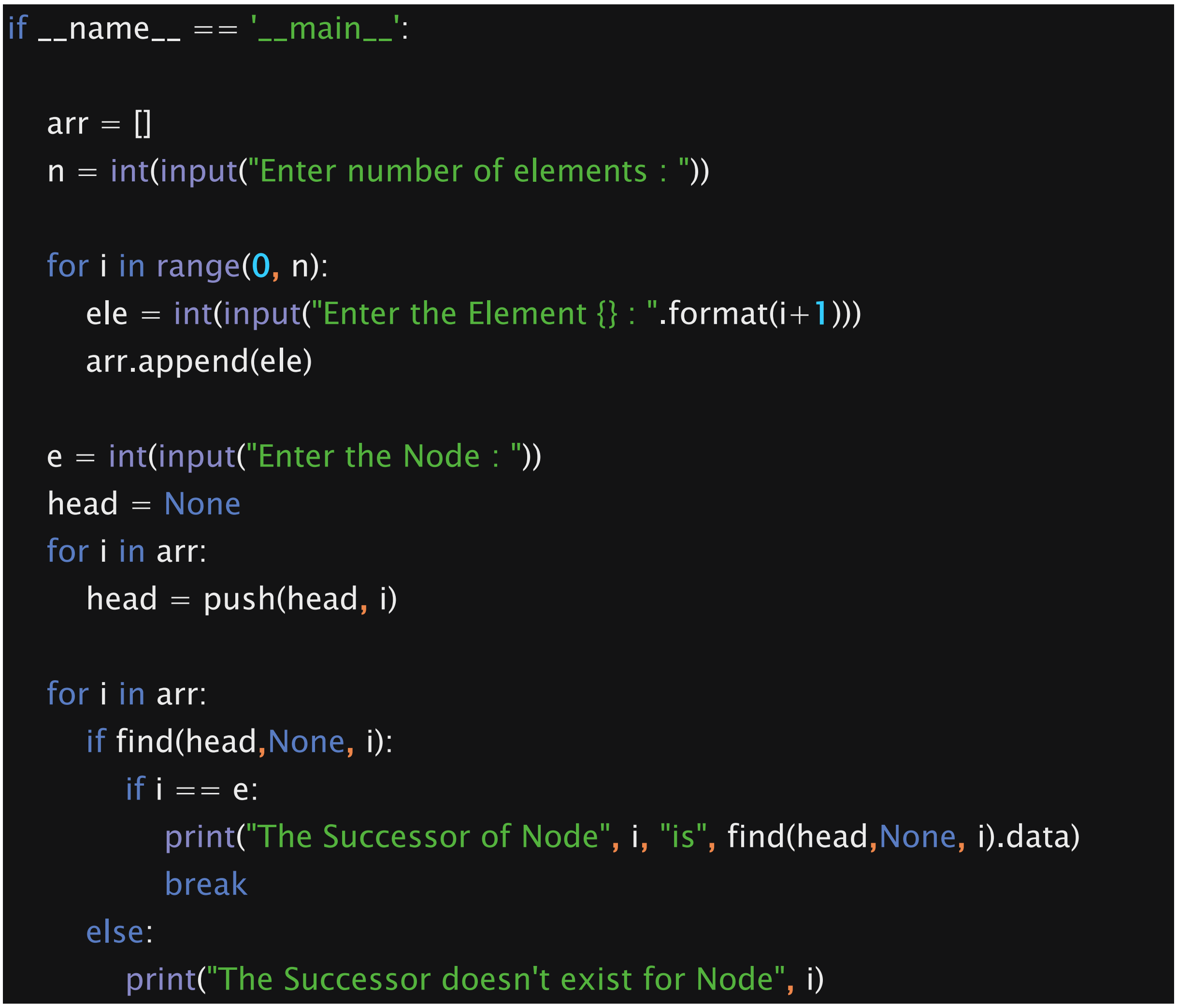
```
class Node:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right


def push(root, key):

    if root is None:
        return Node(key)

    if key < root.data:
        root.left = push(root.left, key)

    else:
        root.right = push(root.right, key)

    return root


def find_min(node):
    while node.left:
        node = node.left
    return node


def find(root, successor, key):
    if root is None:
        return None
```



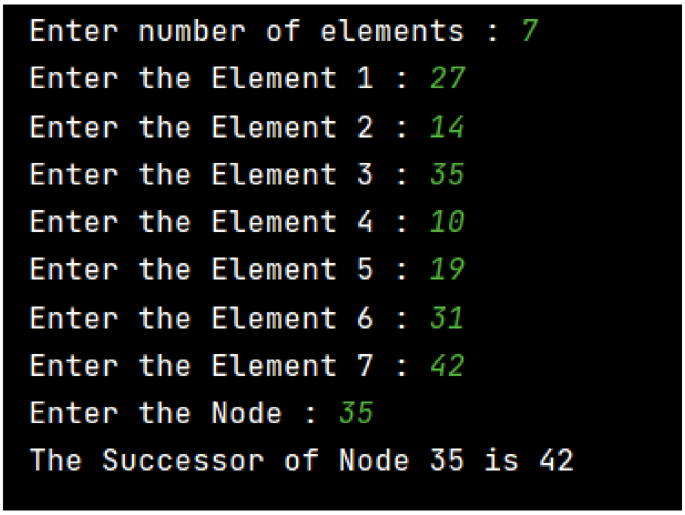
```python
    if root.data == key:
        if root.right:
            return find_min(root.right)

    elif key < root.data:

        successor = root
        return find(root.left, successor, key)

    else:
        return find(root.right, successor, key)

    return successor


if __name__ == '__main__':

    arr = []
    n = int(input("Enter number of elements : "))

    for i in range(0, n):
        ele = int(input())
        arr.append(ele)

    head = None
    for i in arr:
        head = push(head, i)

    for i in arr:
        if find(head,None, i):
            print("The Successor of Node", i, "is", find(head,None, i).data)
        else:
            print("The Successor doesn't exist for Node", i)
```



```
Enter number of elements : 7
27
14
35
10
19
31
42
The Successor of Node 27 is 31
The Successor of Node 14 is 19
The Successor of Node 35 is 42
The Successor of Node 10 is 14
The Successor of Node 19 is 27
The Successor of Node 31 is 35
The Successor doesn't exist for Node 42
```

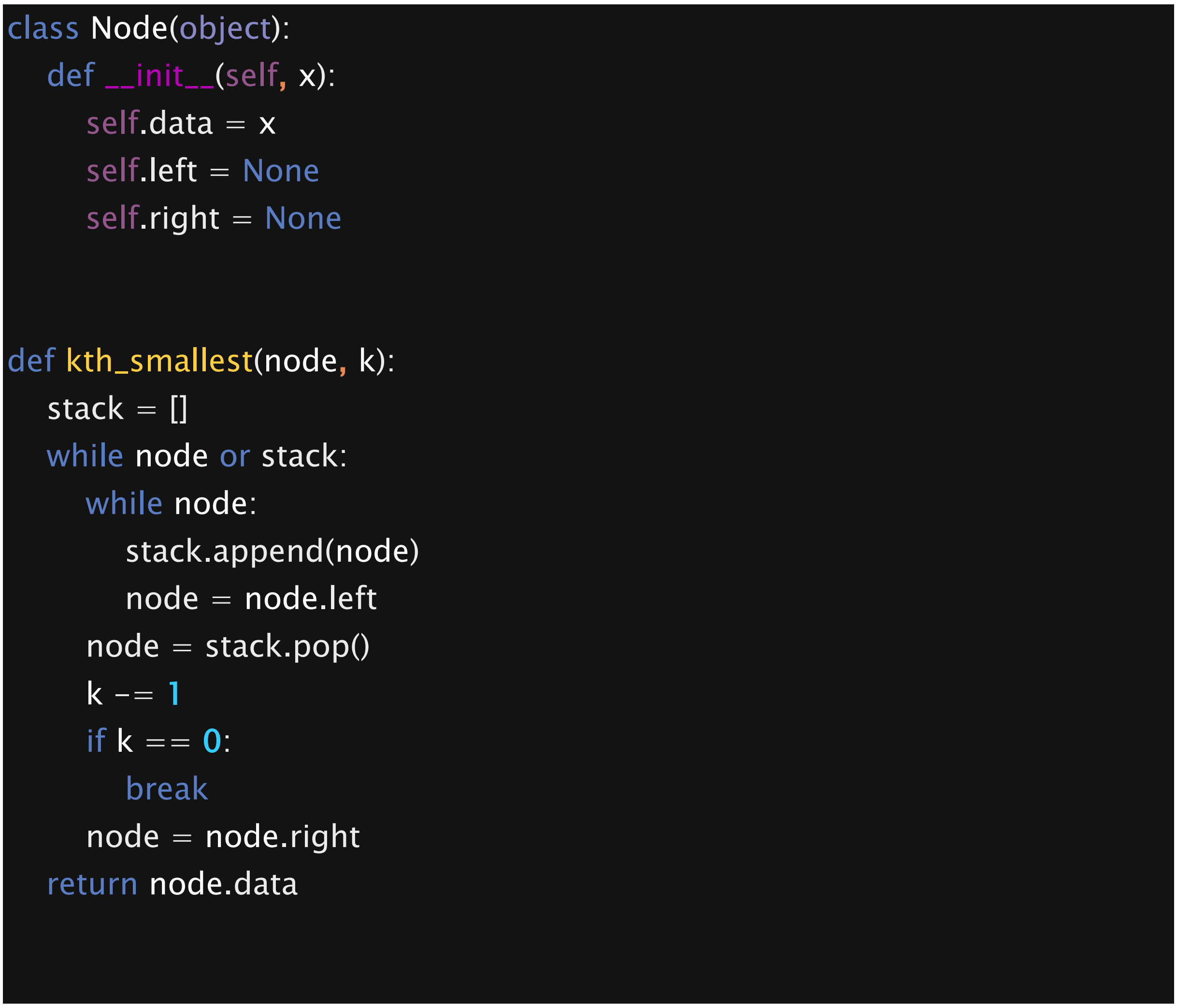


```python
if __name__ == '__main__':

    arr = []
    n = int(input("Enter number of elements : "))

    for i in range(0, n):
        ele = int(input("Enter the Element {} : ".format(i+1)))
        arr.append(ele)

    e = int(input("Enter the Node : "))
    head = None
    for i in arr:
        head = push(head, i)

    for i in arr:
        if find(head,None, i):
            if i == e:
                print("The Successor of Node", i, "is", find(head,None, i).data)
                break
        else:
            print("The Successor doesn't exist for Node", i)
```

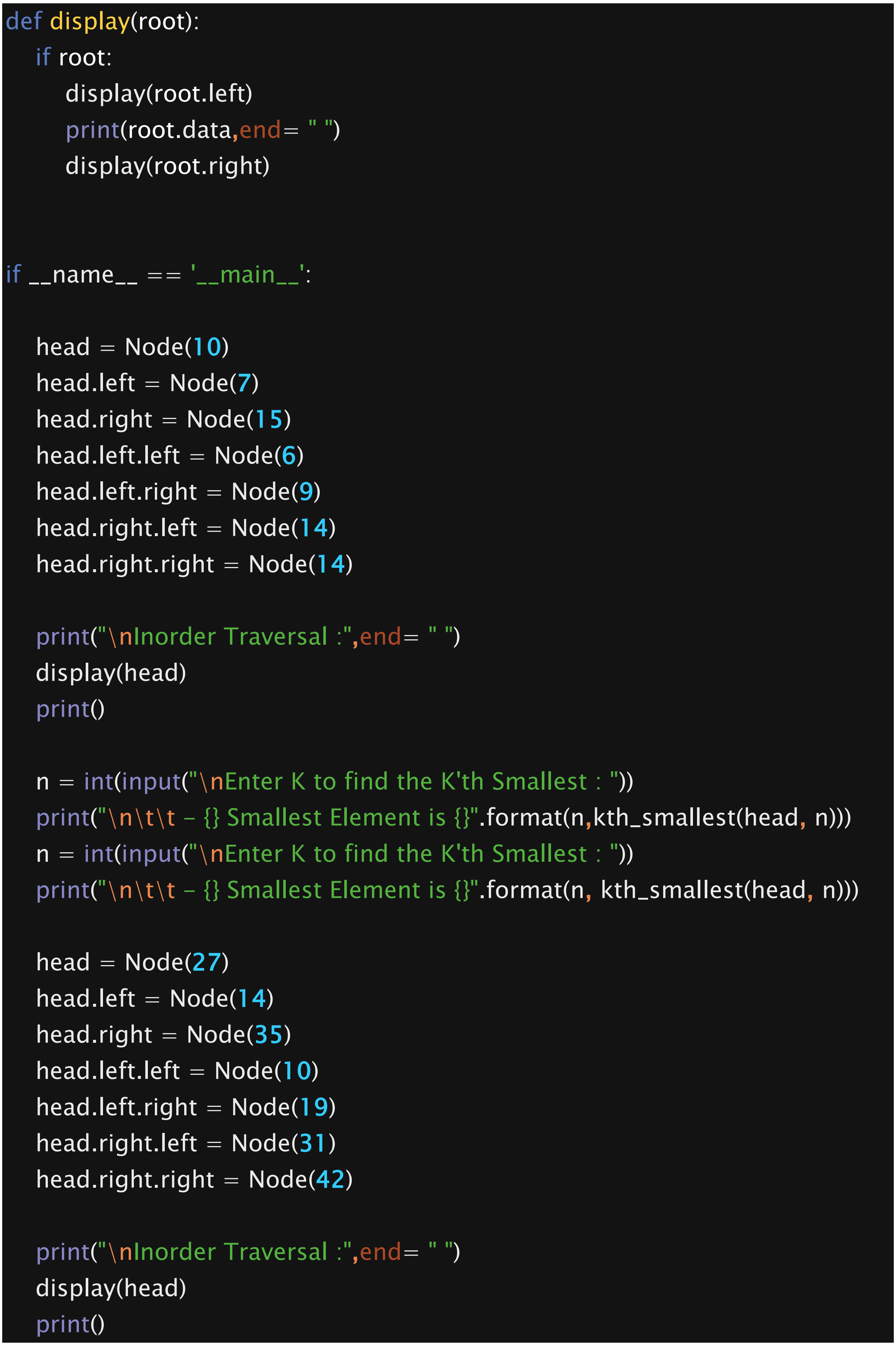


```
Enter number of elements : 7
Enter the Element 1 : 27
Enter the Element 2 : 14
Enter the Element 3 : 35
Enter the Element 4 : 10
Enter the Element 5 : 19
Enter the Element 6 : 31
Enter the Element 7 : 42
Enter the Node : 35
The Successor of Node 35 is 42
```

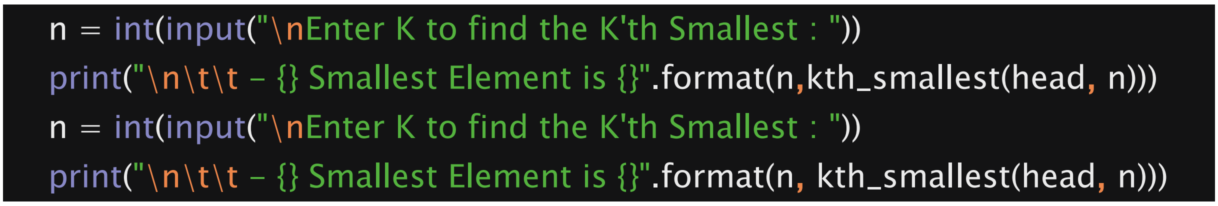
4. Find the kth smallest element in a BST.



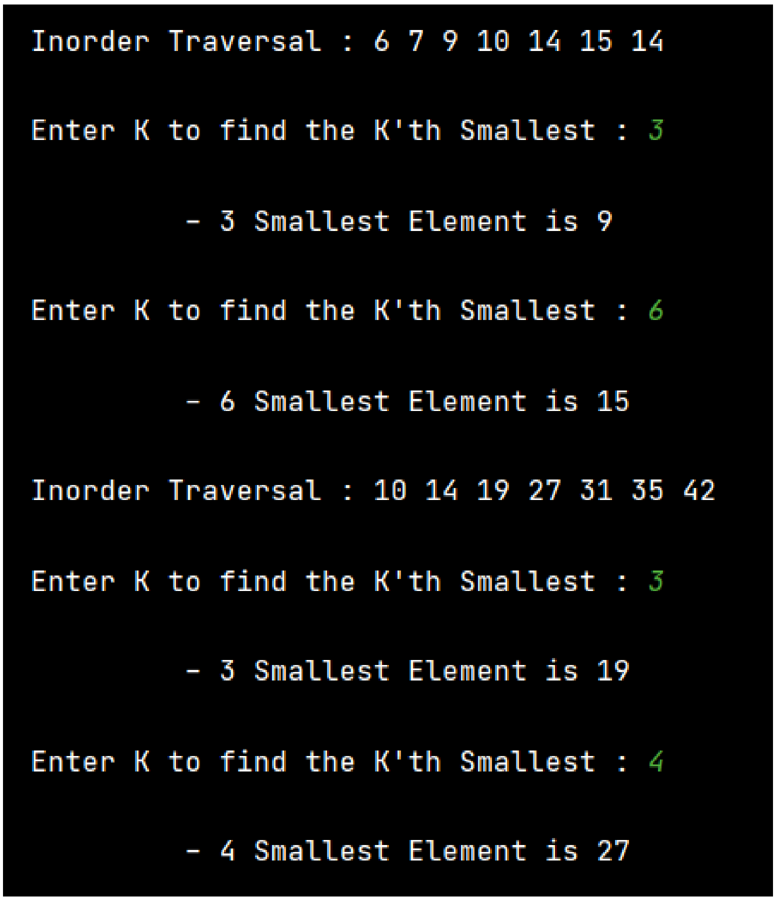
```
class Node(object):
    def __init__(self, x):
        self.data = x
        self.left = None
        self.right = None


def kth_smallest(node, k):
    stack = []
    while node or stack:
        while node:
            stack.append(node)
            node = node.left
        node = stack.pop()
        k -= 1
        if k == 0:
            break
        node = node.right
    return node.data
```



```python
def display(root):
    if root:
        display(root.left)
        print(root.data,end= " ")
        display(root.right)


if __name__ == '__main__':

    head = Node(10)
    head.left = Node(7)
    head.right = Node(15)
    head.left.left = Node(6)
    head.left.right = Node(9)
    head.right.left = Node(14)
    head.right.right = Node(14)

    print("\nInorder Traversal :",end= " ")
    display(head)
    print()

    n = int(input("\nEnter K to find the K'th Smallest : "))
    print("\n\t\t - {} Smallest Element is {}".format(n,kth_smallest(head, n)))
    n = int(input("\nEnter K to find the K'th Smallest : "))
    print("\n\t\t - {} Smallest Element is {}".format(n, kth_smallest(head, n)))

    head = Node(27)
    head.left = Node(14)
    head.right = Node(35)
    head.left.left = Node(10)
    head.left.right = Node(19)
    head.right.left = Node(31)
    head.right.right = Node(42)

    print("\nInorder Traversal :",end= " ")
    display(head)
    print()
```



```
n = int(input("\nEnter K to find the K'th Smallest : "))
print("\n\t\t - {} Smallest Element is {}".format(n,kth_smallest(head, n)))
n = int(input("\nEnter K to find the K'th Smallest : "))
print("\n\t\t - {} Smallest Element is {}".format(n, kth_smallest(head, n)))
```



```
Inorder Traversal : 6 7 9 10 14 15 14

Enter K to find the K'th Smallest : 3

          - 3 Smallest Element is 9

Enter K to find the K'th Smallest : 6

          - 6 Smallest Element is 15

Inorder Traversal : 10 14 19 27 31 35 42

Enter K to find the K'th Smallest : 3

          - 3 Smallest Element is 19

Enter K to find the K'th Smallest : 4

          - 4 Smallest Element is 27
```

One Drive : Click Me!!

Thankyou!!