Sir Padampat Singhania University, Udaipur.

Academic Year 2023 -24

Department: School of Liberal Studies

Name of Assignment: Linked List Implication.

Full Name: Naman Jain

Roll No.: 23MCA00031

Subject: Advanced Data Structures

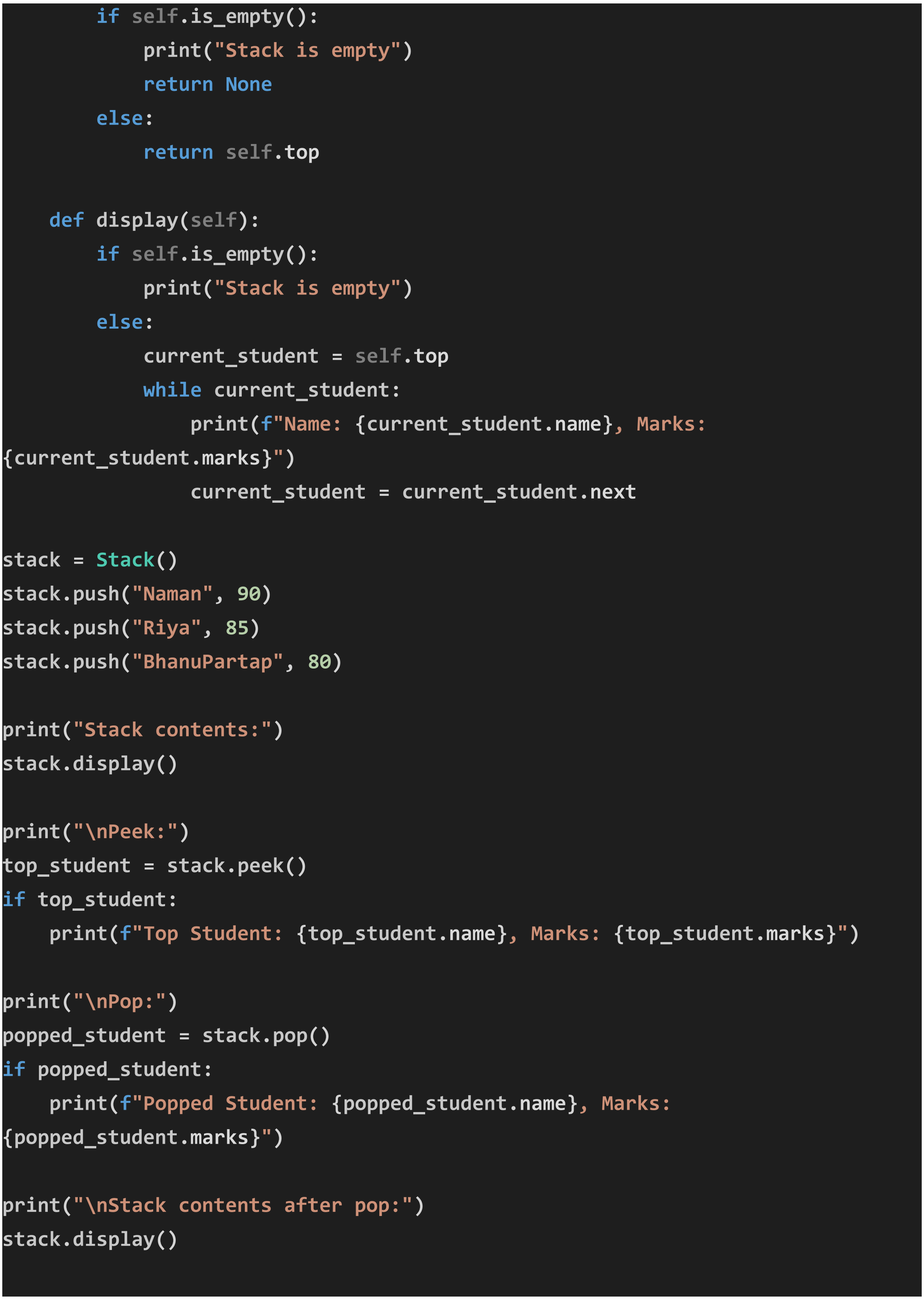
Date of Submission: 18/09/2023

1. Write a Python program to implement Stack data structure using the Linked list data structure.

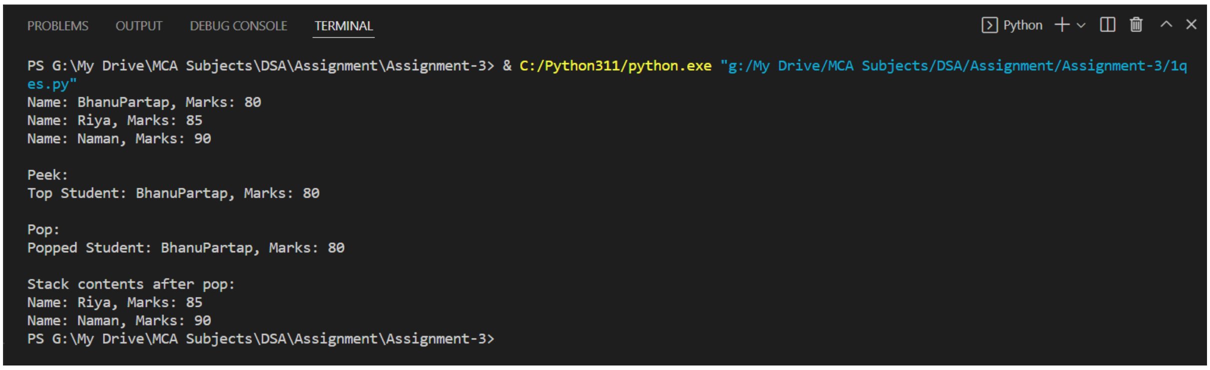
The required program needs to fulfill the following requirements

- **Create a Node (Student) class to save the information (name and marks) of a student.**
- **Create one or multiple stacks to save student's data into the stack(s).**
- **Use linked list data structure for the implementation of stack.**

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks
        self.next = None

class Stack:
    def __init__(self):
        self.top = None

    def is_empty(self):
        return self.top is None

    def push(self, name, marks):
        new_student = Student(name, marks)
        if self.is_empty():
            self.top = new_student
        else:
            new_student.next = self.top
            self.top = new_student

    def pop(self):
        if self.is_empty():
            print("Stack is empty")
            return None
        else:
            popped_student = self.top
            self.top = self.top.next
            popped_student.next = None
            return popped_student

    def peek(self):
```



```
        if self.is_empty():
            print("Stack is empty")
            return None
        else:
            return self.top

    def display(self):
        if self.is_empty():
            print("Stack is empty")
        else:
            current_student = self.top
            while current_student:
                print(f"Name: {current_student.name}, Marks: 
{current_student.marks}")
                current_student = current_student.next

stack = Stack()
stack.push("Naman", 90)
stack.push("Riya", 85)
stack.push("BhanuPartap", 80)

print("Stack contents:")
stack.display()

print("\nPeek:")
top_student = stack.peek()
if top_student:
    print(f"Top Student: {top_student.name}, Marks: {top_student.marks}")

print("\nPop:")
popped_student = stack.pop()
if popped_student:
    print(f"Popped Student: {popped_student.name}, Marks: 
{popped_student.marks}")

print("\nStack contents after pop:")
stack.display()
```



```
PS G:\My Drive\MCA Subjects\DSA\Assignment\Assignment-3> & C:/Python311/python.exe "g:/My Drive/MCA Subjects/DSA/Assignment/Assignment-3/1q
es.py"
Name: BhanuPartap, Marks: 80
Name: Riya, Marks: 85
Name: Naman, Marks: 90

Peek:
Top Student: BhanuPartap, Marks: 80

Pop:
Popped Student: BhanuPartap, Marks: 80

Stack contents after pop:
Name: Riya, Marks: 85
Name: Naman, Marks: 90
PS G:\My Drive\MCA Subjects\DSA\Assignment\Assignment-3>
```

2. Take the choice of operation from the user and perform the following operations which depends on the choice.

1. To add a student in Stack.
2. To remove a Student from Stack.
3. Display all students of Stack.
4. Display top 3 positions of students. (Student with the highest marks is in first place, other positions are according to the same highest marks criteria).
5. Press 5 or any other key to close the program.

Ans.

For sorting of students according to their marks

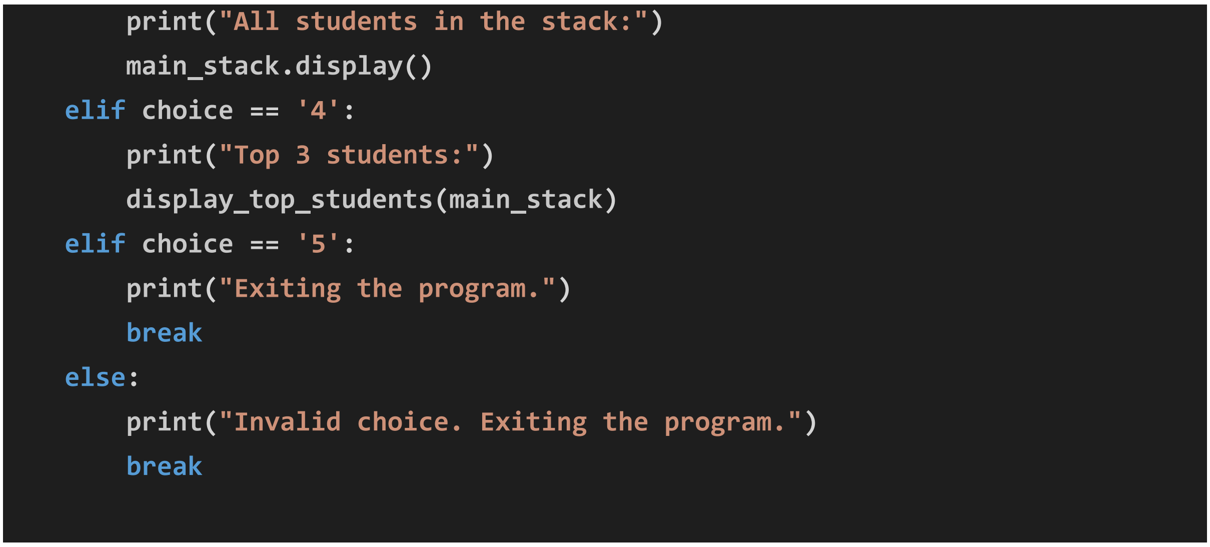
```
def sort_descending(self):
        if self.is_empty():
            return

        sorted_stack = Stack()
        while not self.is_empty():
            current_student = self.pop()
            while not sorted_stack.is_empty() and sorted_stack.peek().marks < current_student.marks:
                temp = sorted_stack.pop()
                self.push(temp.name, temp.marks)
            sorted_stack.push(current_student.name, current_student.marks)

        while not sorted_stack.is_empty():
            temp = sorted_stack.pop()
            self.push(temp.name, temp.marks)
```

For displaying top 3 students

```python
def display_top_students(stack):
    if stack.is_empty():
        print("Stack is empty")
        return

    sorted_stack = Stack()
    sorted_stack = stack.sort_descending()
    count = 0
    while count < 3 and not sorted_stack.is_empty():
        student = sorted_stack.pop()
        print(f"Name: {student.name}, Marks: {student.marks}")
        count += 1
```

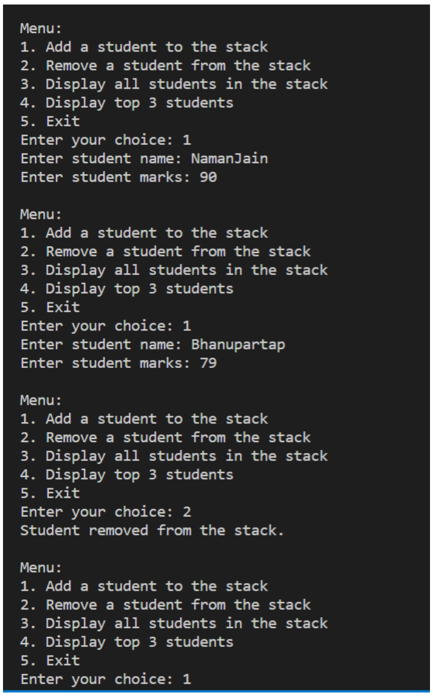
Taking input from user

```python

main_stack = Stack()
secondary_stack = Stack()

while True:
    print("\nMenu:")
    print("1. Add a student to the stack")
    print("2. Remove a student from the stack")
    print("3. Display all students in the stack")
    print("4. Display top 3 students")
    print("5. Exit")
    choice = input("Enter your choice: ")

    if choice == '1':
        name = input("Enter student name: ")
        marks = int(input("Enter student marks: "))
        main_stack.push(name, marks)
    elif choice == '2':
        main_stack.pop()
        print("Student removed from the stack.")
    elif choice == '3':
```

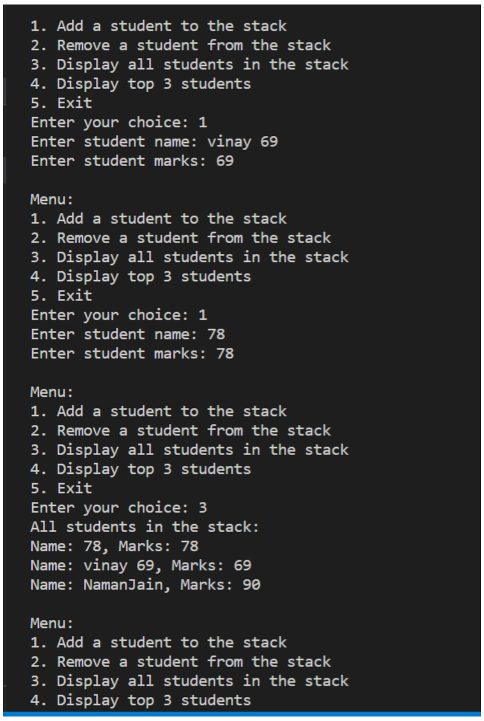


```
        print("All students in the stack:")
        main_stack.display()
    elif choice == '4':
        print("Top 3 students:")
        display_top_students(main_stack)
    elif choice == '5':
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Exiting the program.")
        break
```

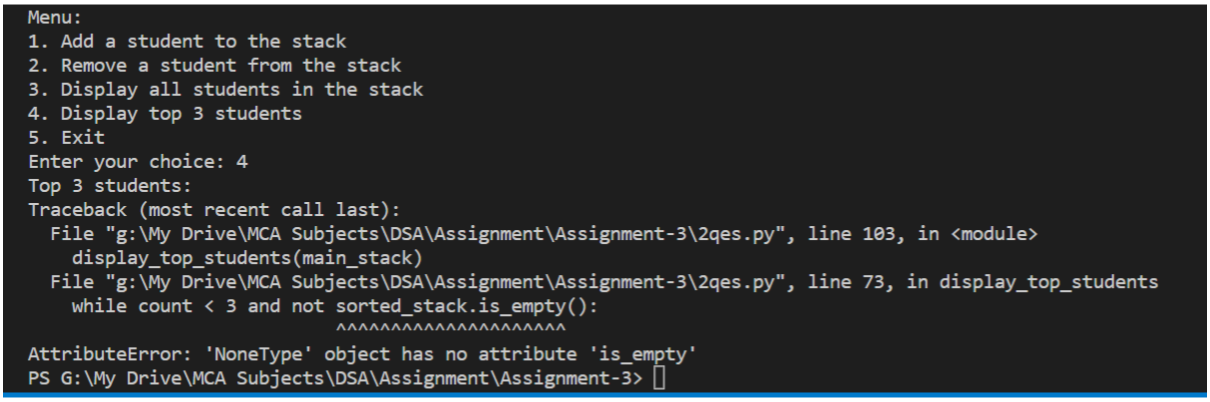
output:



```
Menu:
1. Add a student to the stack
2. Remove a student from the stack
3. Display all students in the stack
4. Display top 3 students
5. Exit
Enter your choice: 1
Enter student name: NamanJain
Enter student marks: 90

Menu:
1. Add a student to the stack
2. Remove a student from the stack
3. Display all students in the stack
4. Display top 3 students
5. Exit
Enter your choice: 1
Enter student name: Bhanupartap
Enter student marks: 79

Menu:
1. Add a student to the stack
2. Remove a student from the stack
3. Display all students in the stack
4. Display top 3 students
5. Exit
Enter your choice: 2
Student removed from the stack.

Menu:
1. Add a student to the stack
2. Remove a student from the stack
3. Display all students in the stack
4. Display top 3 students
5. Exit
Enter your choice: 1
```



```
1. Add a student to the stack
2. Remove a student from the stack
3. Display all students in the stack
4. Display top 3 students
5. Exit
Enter your choice: 1
Enter student name: vinay 69
Enter student marks: 69

Menu:
1. Add a student to the stack
2. Remove a student from the stack
3. Display all students in the stack
4. Display top 3 students
5. Exit
Enter your choice: 1
Enter student name: 78
Enter student marks: 78

Menu:
1. Add a student to the stack
2. Remove a student from the stack
3. Display all students in the stack
4. Display top 3 students
5. Exit
Enter your choice: 3
All students in the stack:
Name: 78, Marks: 78
Name: vinay 69, Marks: 69
Name: NamanJain, Marks: 90

Menu:
1. Add a student to the stack
2. Remove a student from the stack
3. Display all students in the stack
4. Display top 3 students
```



```
Menu:
1. Add a student to the stack
2. Remove a student from the stack
3. Display all students in the stack
4. Display top 3 students
5. Exit
Enter your choice: 4
Top 3 students:
Traceback (most recent call last):
  File "g:\My Drive\MCA Subjects\DSA\Assignment\Assignment-3\2qes.py", line 103, in <module>
    display_top_students(main_stack)
  File "g:\My Drive\MCA Subjects\DSA\Assignment\Assignment-3\2qes.py", line 73, in display_top_students
    while count < 3 and not sorted_stack.is_empty():
                            ^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'is_empty'
PS G:\My Drive\MCA Subjects\DSA\Assignment\Assignment-3> 
```

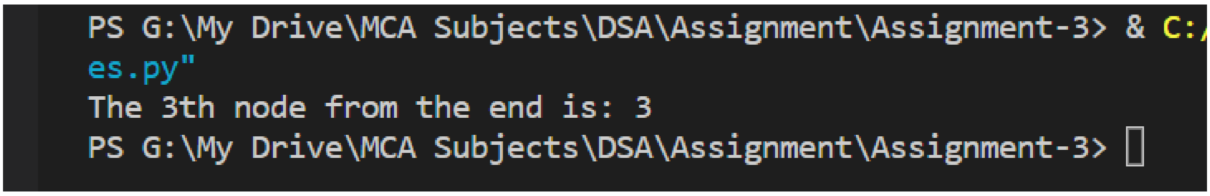
3. Write a program to find out the nth node form the end of a linked list.

Ans.

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

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

    def add_node(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = new_node

    def find_nth_node_from_end(self, n):
        if not self.head or n <= 0:
            return None

        first = self.head
        second = self.head

        for i in range(n):
            if first is None:
                return None
            first = first.next

        while first:
            first = first.next
            second = second.next

        return second.data if second else None

# example
```

```
llist = LinkedList()
llist.add_node(1)
llist.add_node(2)
llist.add_node(3)
llist.add_node(4)
llist.add_node(5)

n = 3
result = llist.find_nth_node_from_end(n)
if result:
    print(f"The {n}th node from the end is: {result}")
else:
    print("Invalid input or node not found.")
```

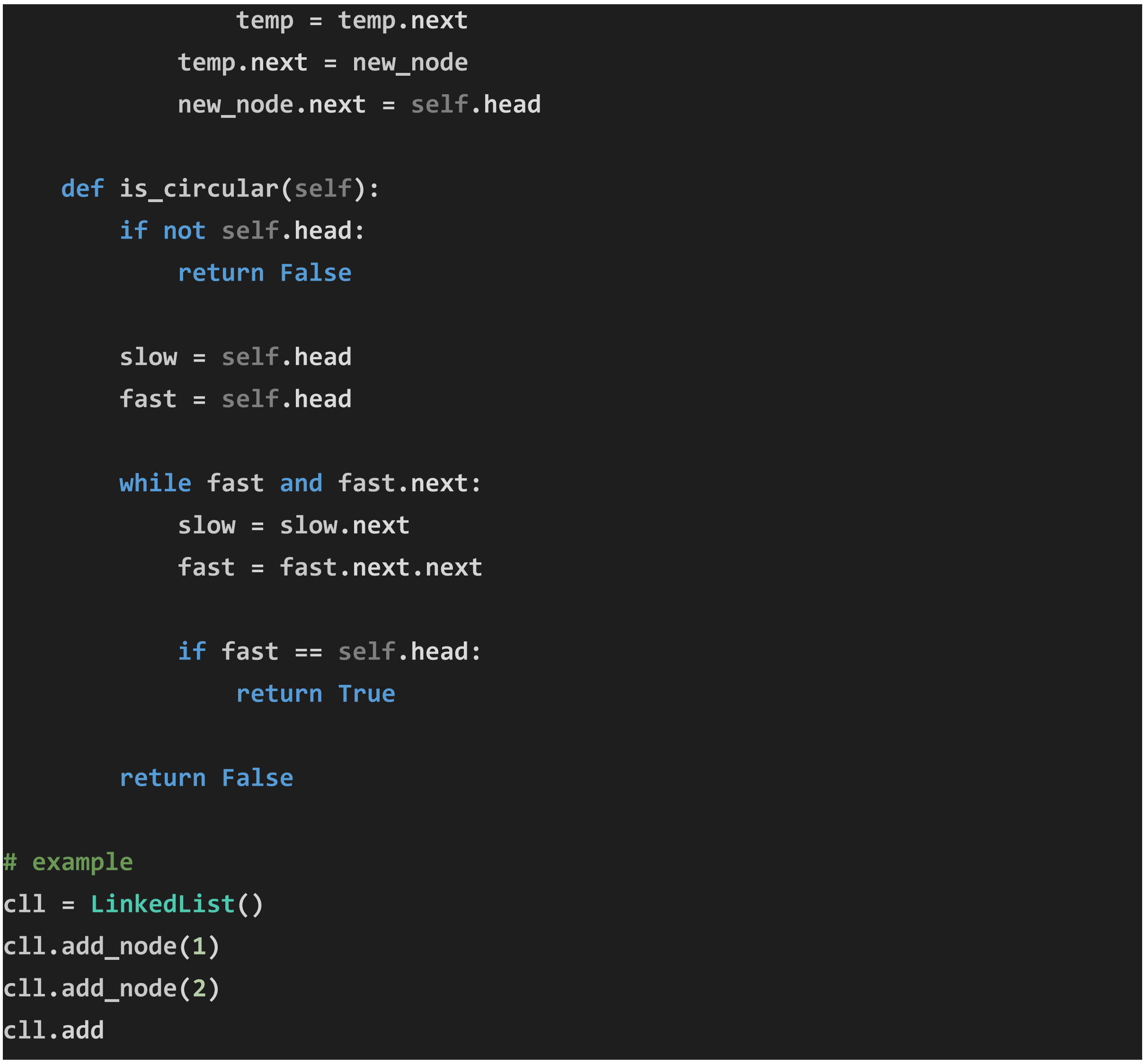
output:



```
PS G:\My Drive\MCA Subjects\DSA\Assignment\Assignment-3> & C:/
es.py"
The 3th node from the end is: 3
PS G:\My Drive\MCA Subjects\DSA\Assignment\Assignment-3> 
```

4. Write a program to check if a linked list is Circular Linked List.

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

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

    def add_node(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            self.head.next = self.head
        else:
            temp = self.head
            while temp.next != self.head:
```



```
                temp = temp.next
            temp.next = new_node
            new_node.next = self.head

    def is_circular(self):
        if not self.head:
            return False

        slow = self.head
        fast = self.head

        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

            if fast == self.head:
                return True

        return False

# example
cll = LinkedList()
cll.add_node(1)
cll.add_node(2)
cll.add
```