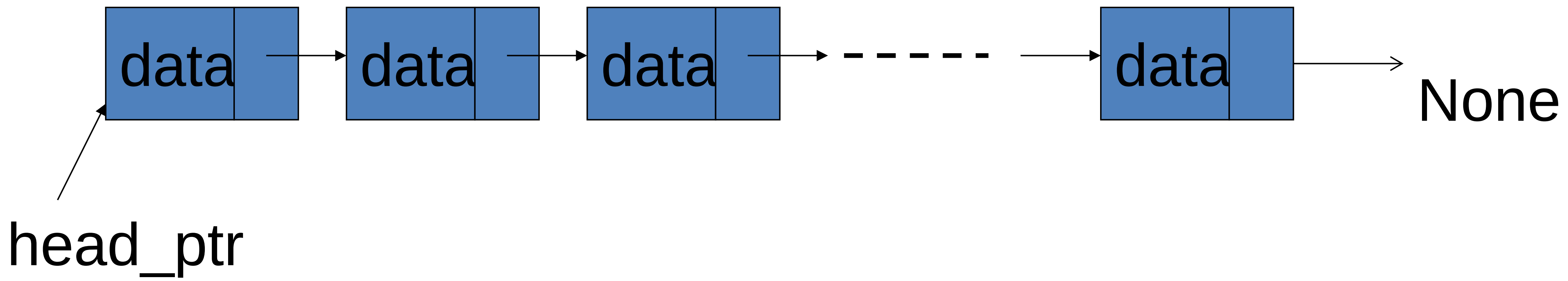
Linked Lists

Definition of Linked Lists

- A linked list is a sequence of items (objects) where every item is linked to the next.
- Graphically:

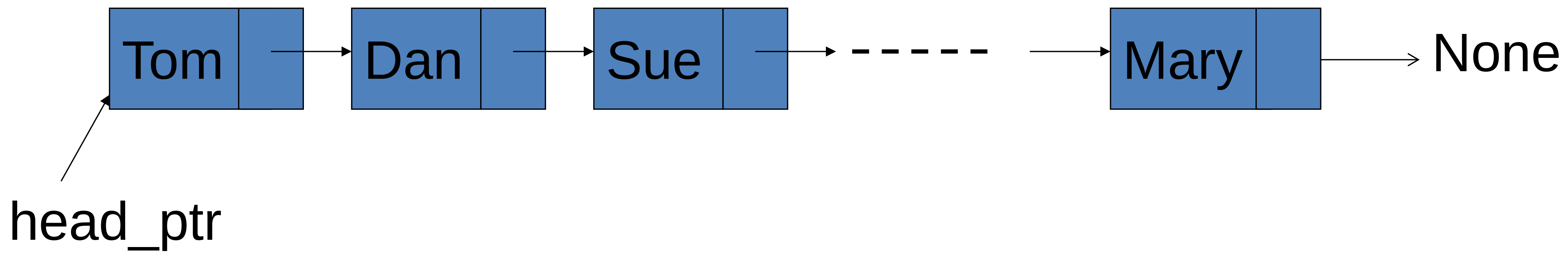


Definition Details

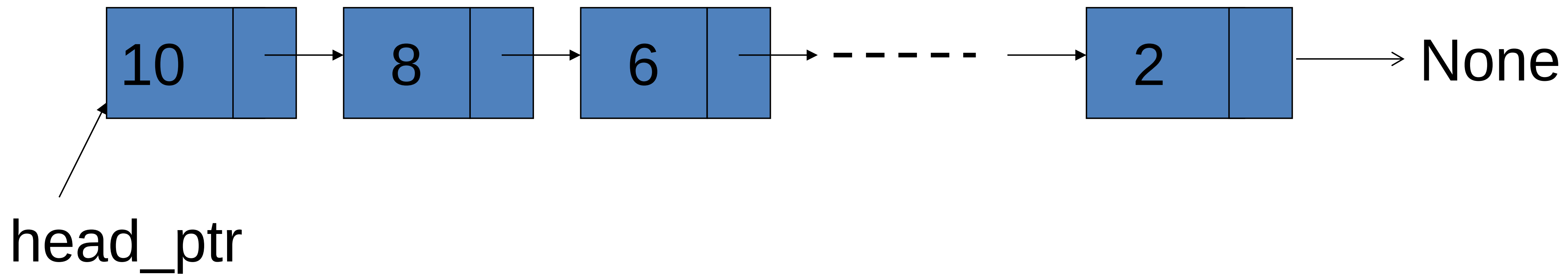
- Each item has a data part , and a link that points to the next item.
- One natural way to implement the link is as a pointer; that is, the link is the address of the next item in the list.
- It makes good sense to view each item as an object, that is, as an instance of a class.
- We call that class: Node
- The last item does not point to anything. We set its link member to **None**.

Examples of Linked Lists

- A linked list of strings can represent a waiting line of customers.



- A linked list of integers can represent a stack of numbers.



Node class

- Every Node has a value and a pointer to the next node.
- When a node is first created, its data is set to None and does not point to any node.

```python
"""Node class"""


class Node:
    """By default the data and next are none"""
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

    def __str__(self):
        return str(self.data)
```

LinkedList class

- LinkedList holds a pointer to the first (head) node in the list and an integer that contains the length of the list.
- A linked list is empty when created; thus the "head" node is None and the length is 0.

```
"""LinkedList class"""


class LinkedList:
    """Handler for manipulating list of Node objects"""
    def __init__(self):
        self.length = 0
        self.head = None
```

Operations on Linked Lists

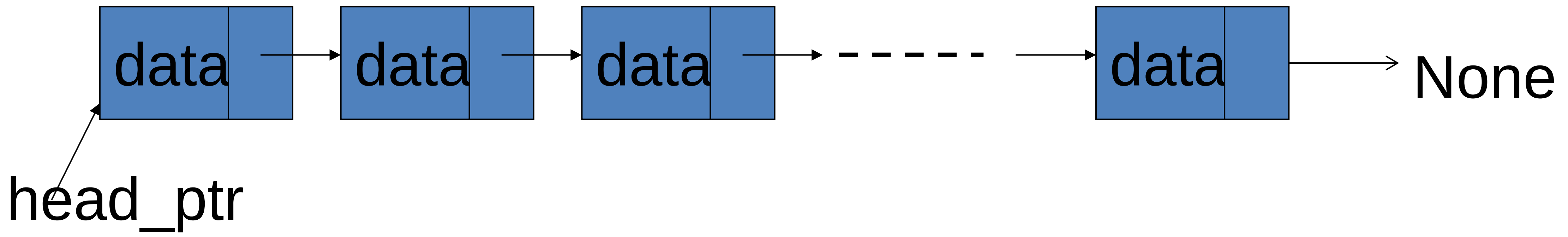
- **Insert** a new item
  ✓ At the head of the list, or
  ✓ At the tail of the list, or
  ✓ Inside the list, in some designated position

- **Delete** an item from the list
  ✓ Search for and locate the item, then remove the item, and finally adjust the surrounding pointers

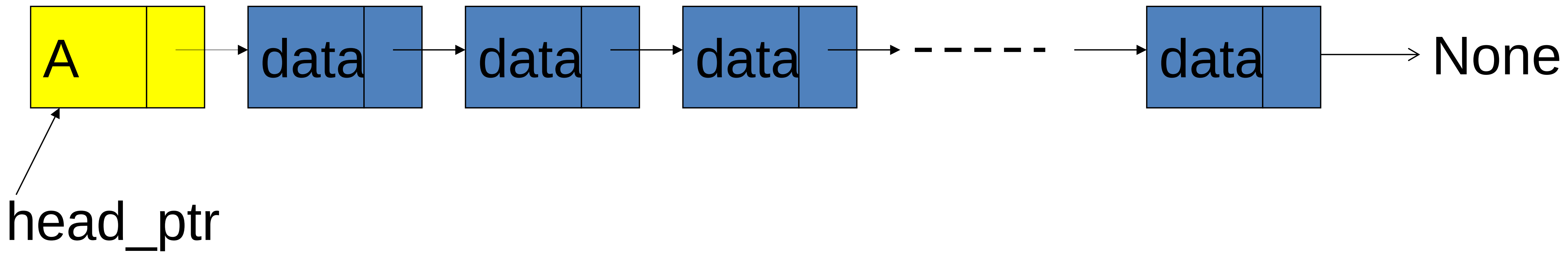
Insert– At the Head

- Insert a new data A.

List before insertion:



- After insertion to head:

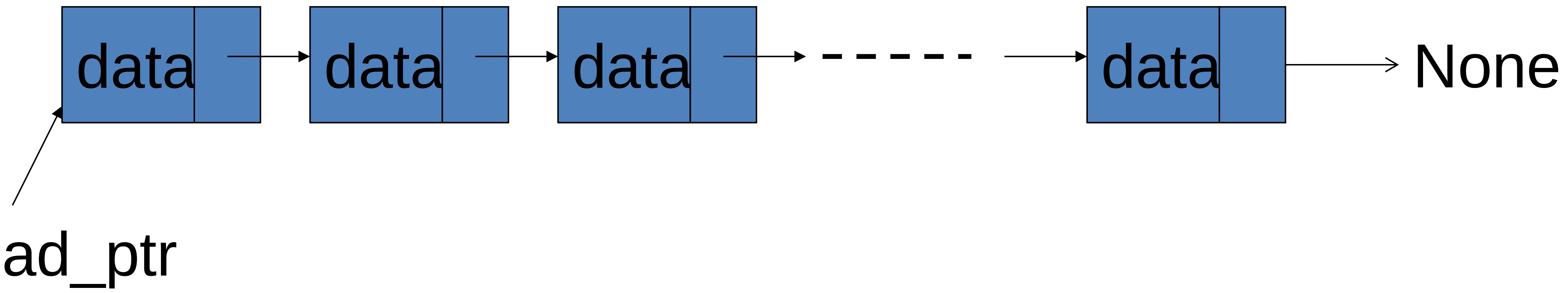


•The link value in the new item = old head_ptr
•The new value of head_ptr = newPtr

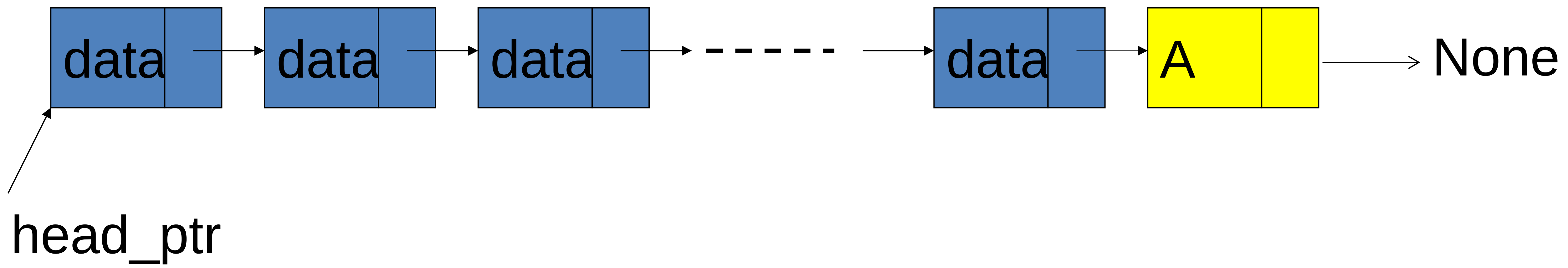
Insert – at the Tail

- Insert a new data A.

  List before insertion



- After insertion to tail:

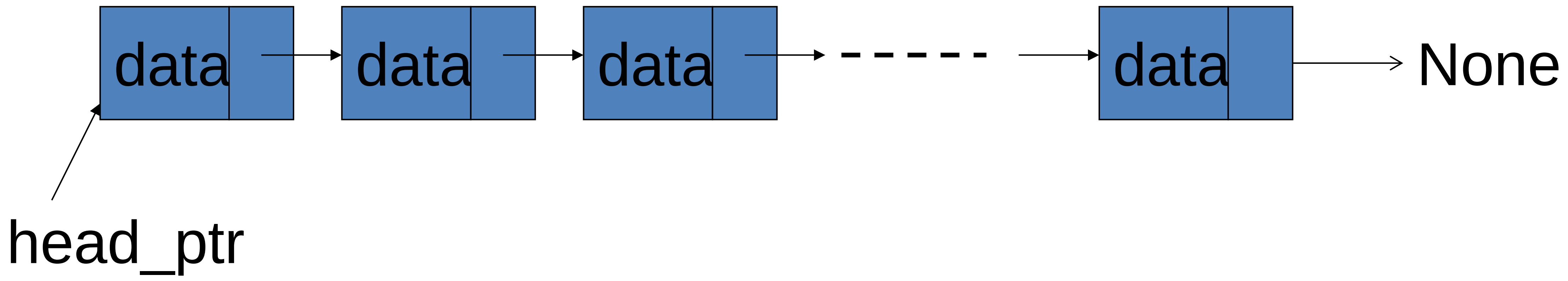


•The link value in the new item = None
•The link value of the old last item = newPtr

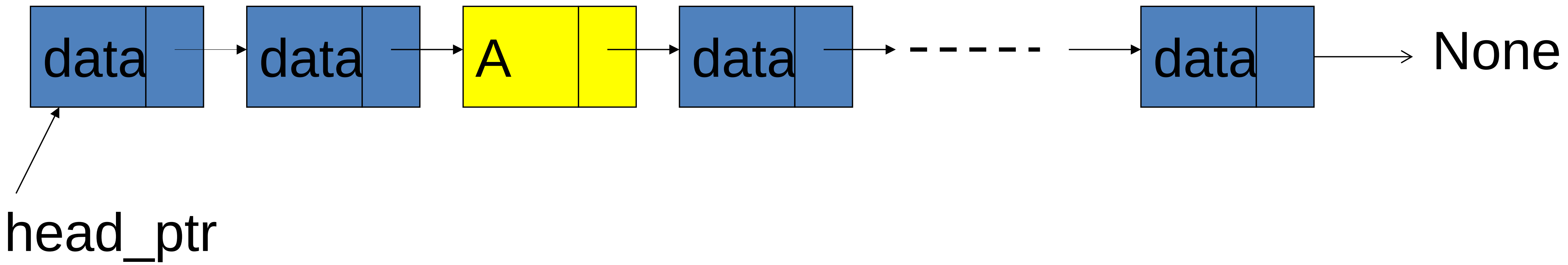
Insert – inside the List

- Insert a new data A.

  List before insertion:



- After insertion in 3rd position:



•The link-value in the new item = link-value of 2nd item

•The new link-value of 2nd item = newPtr

Insert – at the Tail Example

Steps:

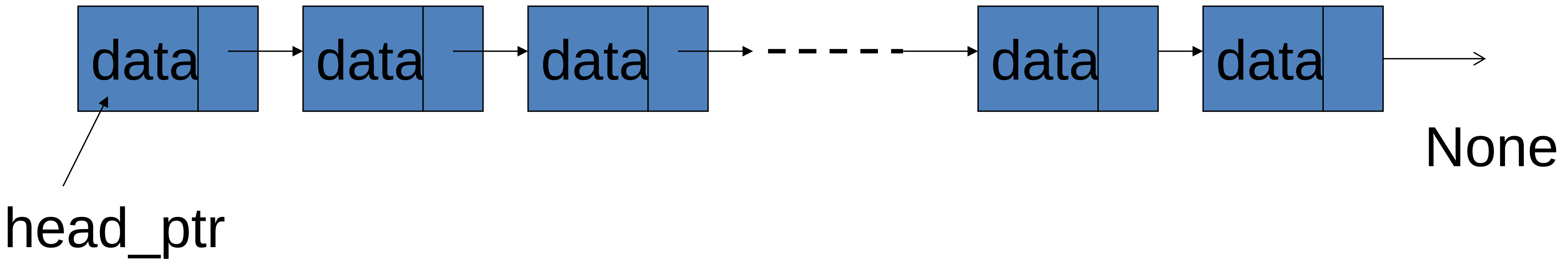
1. Create a node.
2. If the list is empty, make the new node the head of the list.
3. Else traverse the list till the end, make the last node point to the new node and the new node point to None.
4. Increase the length of the list.

Insert – at the Tail Example

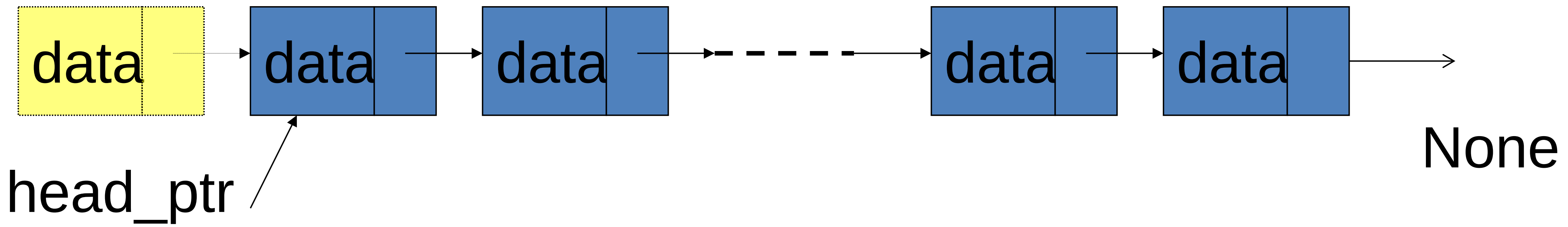
```
def addnode(self, data):
    """Adds a node to the tail of the List"""
    new_node = Node(data)   # Create a node
    if self.length <= 0:  # if the list is empty
        self.head = new_node
        self.length += 1  # increase the length
    else:
        current = self.head
        while current.next is not None:
            current = current.next
        current.next = new_node  # Assign the new node
        self.length += 1  # increase the length
```

Delete – the Head Item

- List before deletion:



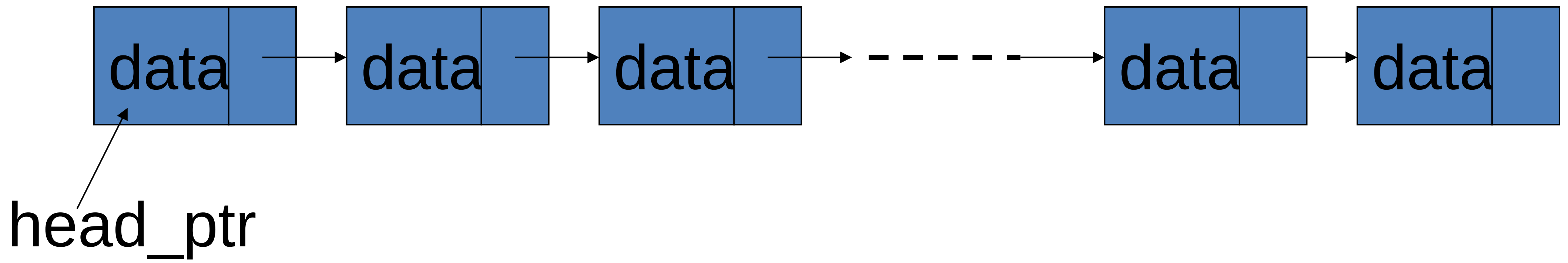
- List after deletion of the head item:



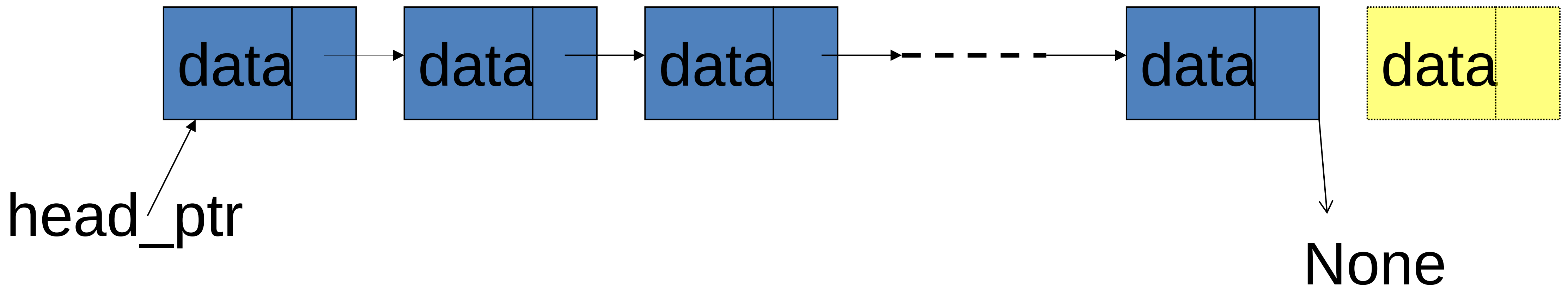
•The new value of head_ptr = link-value of the old head item
•The old head item is deleted and its memory returned

Delete – the Tail Item

- List before deletion:

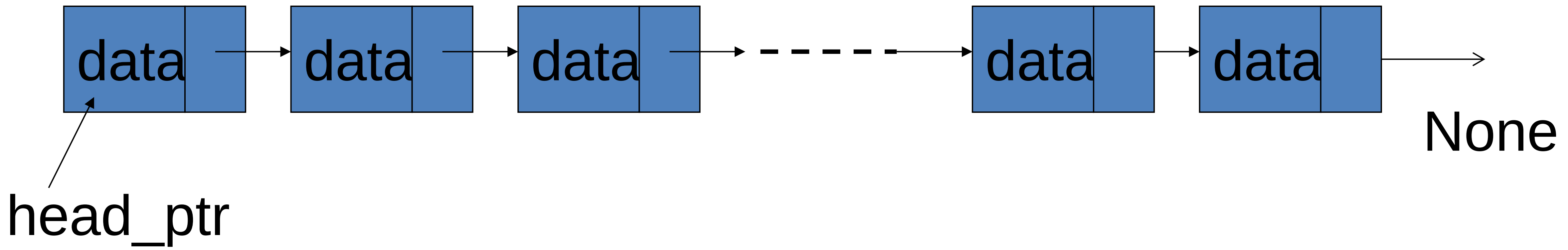


- List after deletion of the tail item:

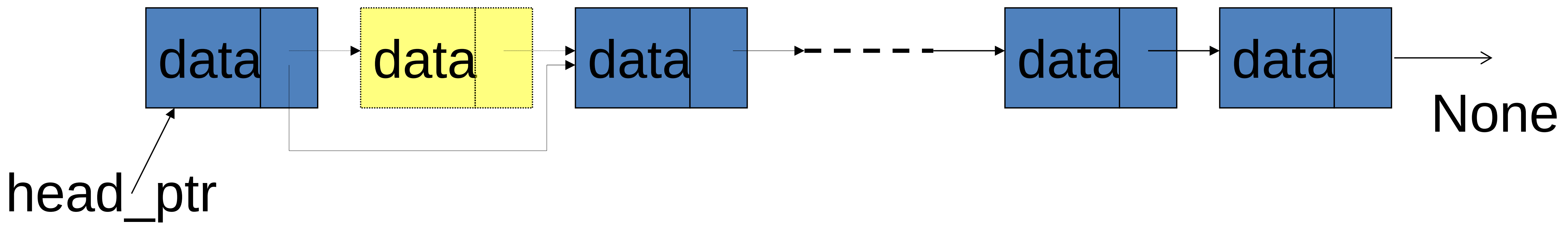


Delete – an inside Item

- List before deletion:



- List after deletion of the 2nd item:



•New link-value of the item located before the deleted one = the link-value of the deleted item

Delete node at a given Index

```
def removenode(self, index):
    """Removes node at a given index"""
    if self.length <= 0:  # check if the list is empty
        print "The list is empty"
    else:
        prev = None
        current = self.head
        i = 0
        while (current is not None) and (i < index):
            prev = current
            current = current.next
            i += 1
        if prev is None:  # the head element is to be removed
            self.head = current.next
            self.length -= 1  # decrease the length of the list
        else:
            prev.next = current.next
            self.length -= 1  # decrease the length of the list
```

Function to print items in a Linked List

```
def printlist(self):
    """Function to print items one by one"""
    current = self.head
    while current is not None:
        print current,
        current = current.next
    print
```

Linked List: Example

- Main function using the Node and LinkedList class to create a Linked List.

```
"""Main program that creates the Linked List"""

if __name__ == "__main__":
    mylist = LinkedList()
    mylist.addnode(1)
    mylist.addnode(2)
    mylist.addnode(3)
    mylist.printlist()
    mylist.removenode(1)
    print "After removing the node at index 1"
    mylist.printlist()
```

Output:

```
1 2 3
After removing the node at index 1
1 3
```

Bitwise Operators in Python

Python Bitwise Operators:

Operator	Description
&	Binary AND Operator copies a bit to the result if it exists in both operands.
|	Binary OR Operator copies a bit if it exists in either operand.
^	Binary XOR Operator copies the bit if it is set in one operand but not both.
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.

Bitwise Operators Example

```
#!/usr/bin/python
a = 60  # 60 = 0011 1100
b = 13  # 13 = 0000 1101
c = 0
c = a & b  # 12 = 0000 1100
print "Line 1 - Value of c is ", c
c = a | b  # 61 = 0011 1101
print "Line 2 - Value of c is ", c
c = a ^ b  # 49 = 0011 0001
print "Line 3 - Value of c is ", c
c = ~a  # -61 = 1100 0011
print "Line 4 - Value of c is ", c
c = a << 2  # 240 = 1111 0000
print "Line 5 - Value of c is ", c
```

to be continued...