Singly-Linked Lists
Linked Structure

• Constructed using a collection of objects called **nodes**.

• Each node contains data and at least one reference or **link** to another node.

• **Linked list** – a linked structure in which the nodes are linked together in linear order.
Linked List

- Terms:
  - **head** – first node in the list.
  - **tail** – last node in the list; link field has a null reference.
Linked List

- Most nodes have no name; they are referenced via the link of the preceding node.
- **head reference** – the first node must be named or referenced by an external variable.
  - Provides an entry point into the linked list.
  - An empty list is indicated by a null head reference.
Singly Linked List

• A linked list in which
  • each node contains a single link field and
  • allows for a complete linear order traversal from front to back.
Node Definition

- The nodes are constructed from a simple storage class:

```python
class ListNode:
    def __init__(self, data, next_node):
        self.data = data
        self.next = next_node
```
Prepending Nodes

- **Example:** add value 96 to the sample list.
Prepending Nodes

• Create a new node for the new item.

• Connect the new node to the list.
Prepending Nodes

- The resulting list.

- Python code

```
# Given the head reference and the new item.
new_node = ListNode( new_item, self._head )
self._head = new_node
```
Removing The First Node

• The head reference must be reposition to reference the next node in the list.
Linked Stack...
Using a Tail Reference

• Some applications require items be appended to the end of the linked list.
  
  • tail reference – a second external reference indicating the tail or last node in the list.
Appending Nodes

Example: append 21 to the list.
Appending Nodes

• Given the head and tail reference, we can add an item to a linked list.

```python
new_node = ListNode(item, None)
if self._head is None:  # list is empty
    self._head = new_node
else:
    self._tail.next = new_node
self._tail = new_node
```
Linked Queue...
Traversing the Nodes

• We can traverse the nodes using a temporary external reference variable.

  ```python
  self._head
  ```

  ![Linked List Diagram](image)

• Initialize a temporary reference to the head node.

  ```python
  self._head
  ```

• Visit the node.
Traversing the Nodes

- Advance the temporary reference to the next node using the link field and visit that node.
Traversing the Nodes

- Repeat the process until the reference falls off the end of the list.
Traversing the Nodes

- Repeat the process until the reference falls off the end of the list.

```
self._head

2 -> 52 -> 18 -> 36 -> 13
```

```python
cur_node
```
Traversal Code

- Given the head reference, we can traverse the nodes.

```python
def print_list(self):
    cur_node = self._head
    while cur_node is not None:
        print(cur_node.data)
        cur_node = cur_node.next
```