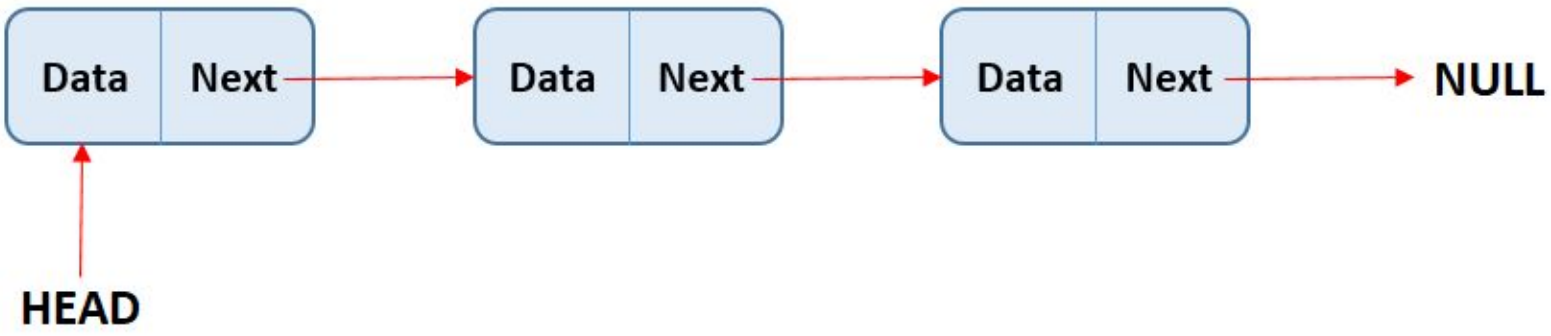
Data Structures

Manage how data is stored and accessed

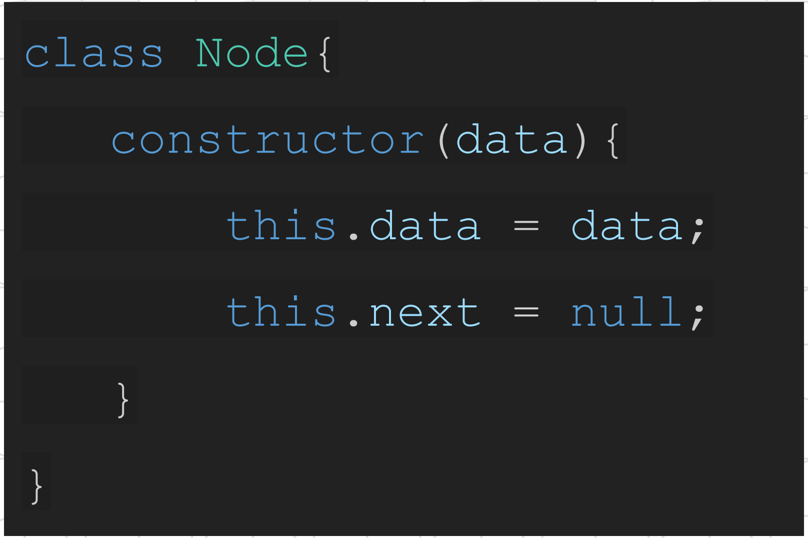
Linked List

A list where nodes are **linked**, each containing data and a pointer to the **next node**, stored in free memory space.



Node

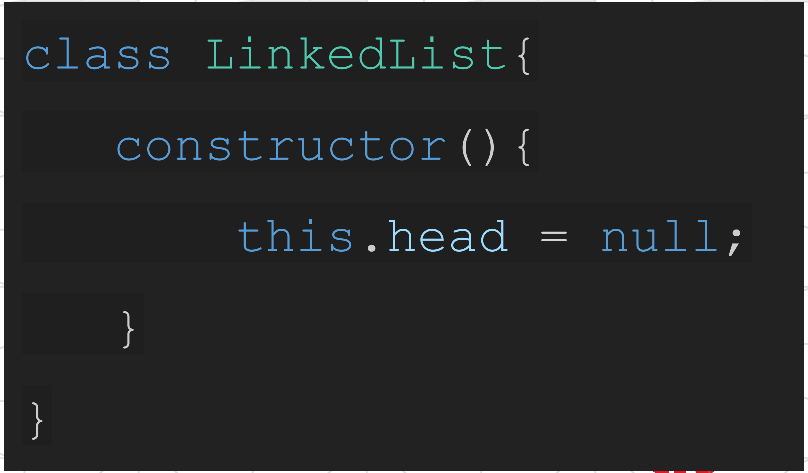
Nodes hold a **value** and the link to the **next node**



```
class Node{
    constructor(data){
        this.data = data;
        this.next = null;
    }
}
```

Linked List

Starts with a **head** property that references the first node. When we start, the list is empty, so the head pointer as **null**



```
class LinkedList{
    constructor(){
        this.head = null;
    }
}
```

Demonstration

appendNode

Demonstration

prependNode

Demonstration

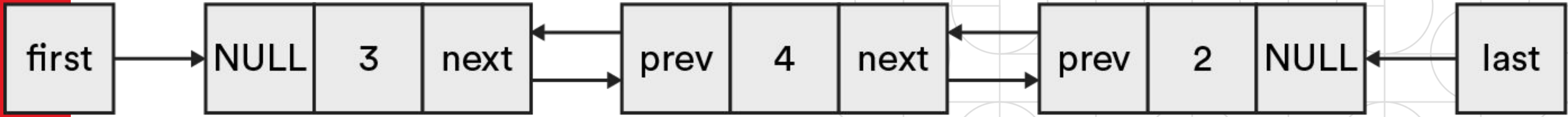
pop

Demonstration

removeFromFront

Doubly Linked Lists

A list where nodes are **linked**, each containing data and a pointer to the **next node** and **previous node**.



Examples

- Playlist
- Photo album
- Web browser

GENERAL ASSEMBLY

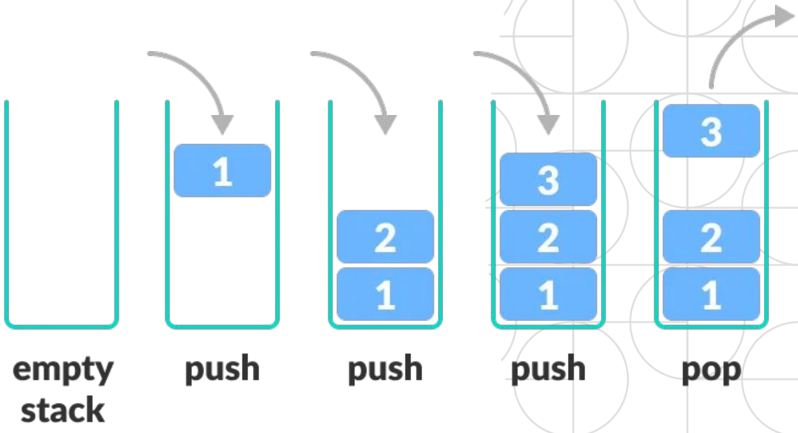
Linked Lists Exercise

Implement the required methods and classes according to the specifications in the comments

https://git.generalassemb.ly/ENT-SDA-SEB-216-IP/SDA-SIRAJ/tree/main/resources/Computer-Science/11.%20Linked-Lists/exercises

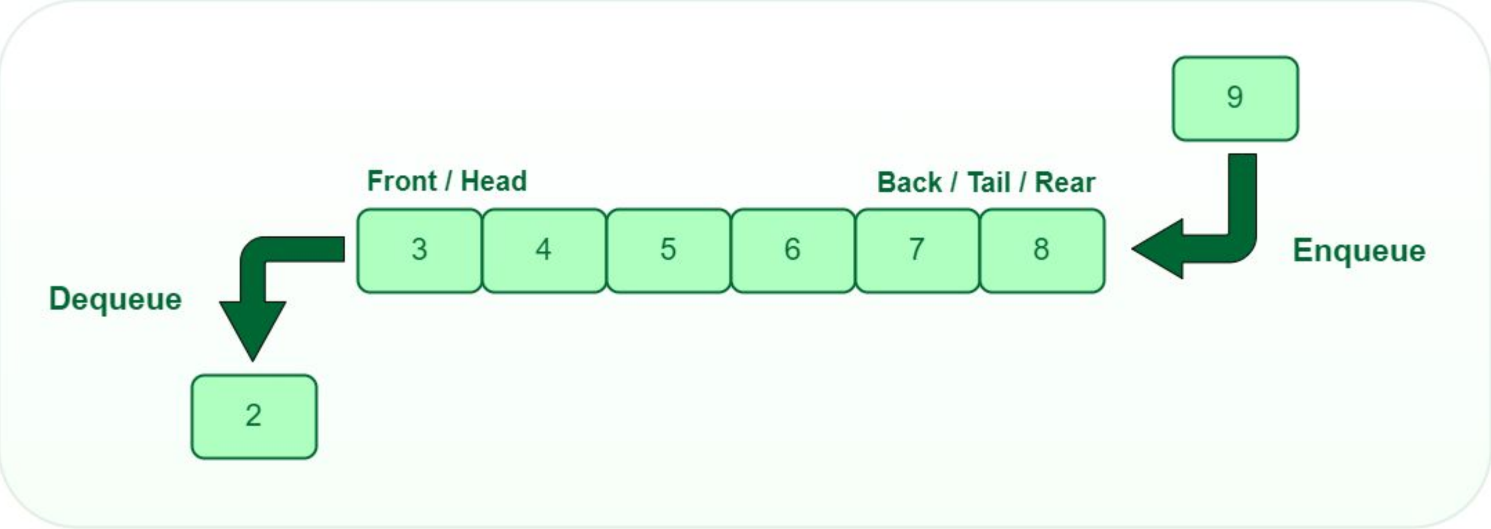
Stacks

Last in, First out behavior. The last, most recently added item, is first to be removed.



Queues

First in, First out behavior. Elements are processed in the order they are added.



Demonstration

Push / Enqueue

Demonstration

Pop / Dequeue

Demonstration

Bracket Matching

Demonstration

EXTRA: Priority Queue

PARTNER EXERCISE: Bracket Matching Exercise

Bracket Matching Exercise

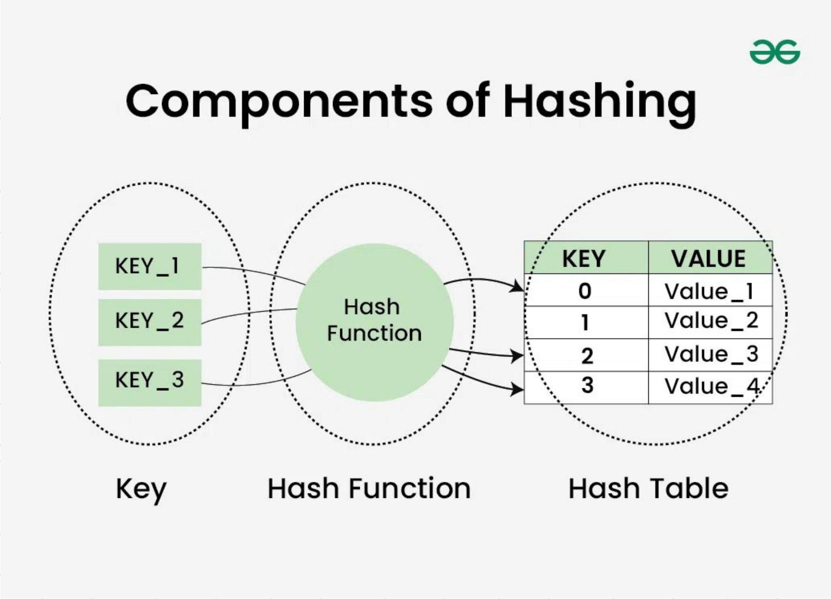
Implement the required methods and classes according to the specifications in the comments

https://git.generalassemb.ly/ENT-SDA-SEB-216-IP/SDA-SIRAJ/tree/main/resources/Computer-Science/12.%20Stacks-And-Queues/exercises

Hash Tables

A data structure that stores **key-value pairs**. It allows fast data retrieval based on a **unique key**.

Keys are passed through a **hash function** to determine where values are stored.



Components of Hashing

KEY	VALUE
0	Value_1
1	Value_2
2	Value_3
3	Value_4

How does it work?

- When creating a hash table, it is important to specify its **size**.
- A **key** is given.
- The **hash function** computes an index in the array.
- The **value** is stored at that index.
- When retrieving, the same hash function **maps the key** to the index.
- **IMPORTANT**: Using a prime number as the hash table size helps distribute keys more evenly, reducing collisions and improving lookup performance.

Where are they used?

- **Dictionaries:** Mapping words to definitions
- **Caching:** Storing previously computed results
- **Database Indexing:** Fast lookup of records
- **Counting occurrences:** e.g., word frequency in text