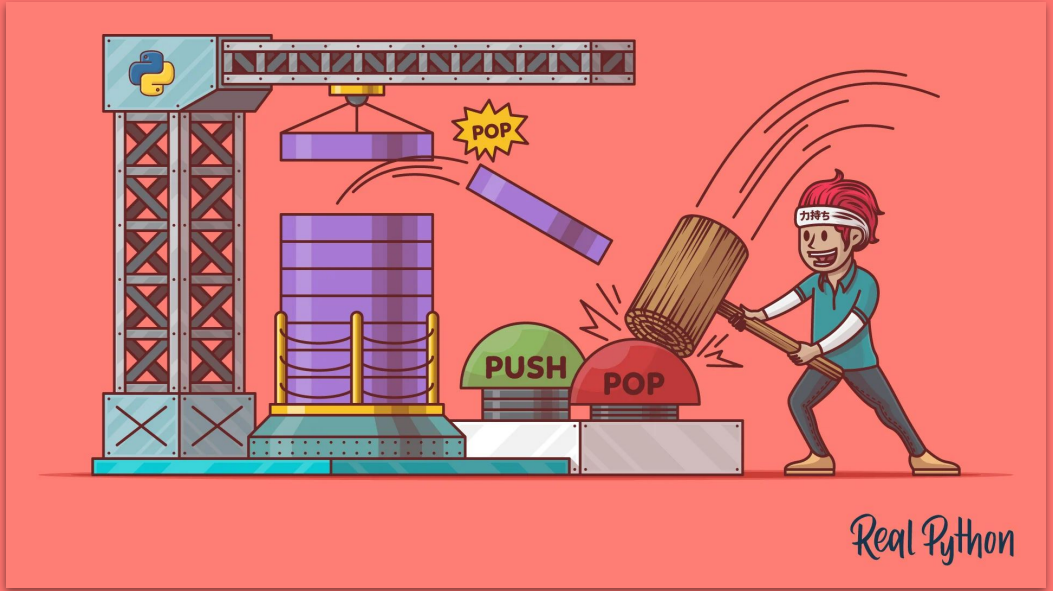
Week 03

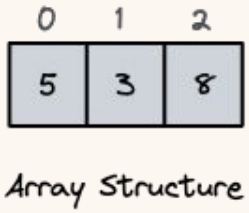
Linked Lists
Queues
Stacks

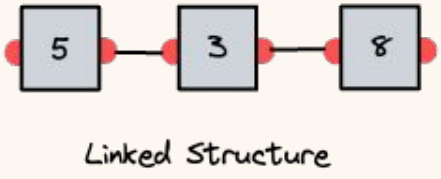
Ivanovitch Silva
ivanovitch.silva@ufrn.br

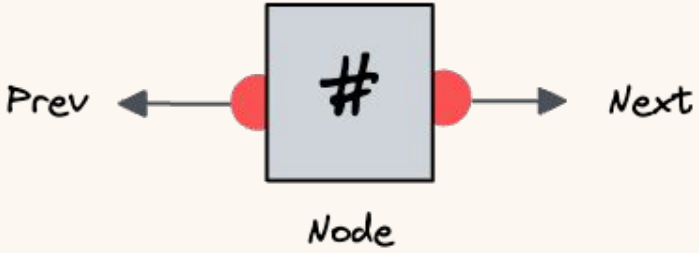


Data Structure Review

Linked Lists



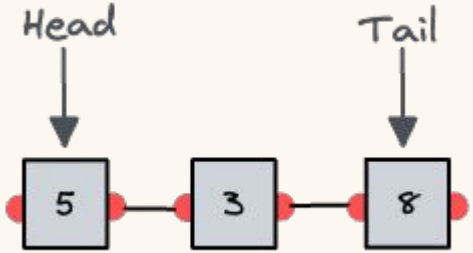





```python
class Node:
    """Represents a node in a doubly linked list.

    Attributes:
        data: The value held by the node.
        prev: A pointer to the previous node in the list.
        next: A pointer to the next node in the list.
    """

    def __init__(self, data):
        """Initializes a new instance of the Node class.

        Args:
            data: The value to be stored in the node.
        """
        self.data = data
        self.prev = None
        self.next = None

node = Node(42)
```

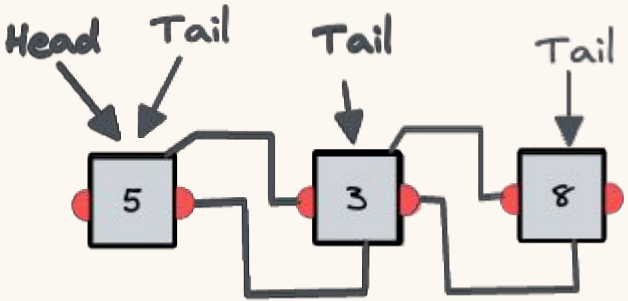
Linked List vs List

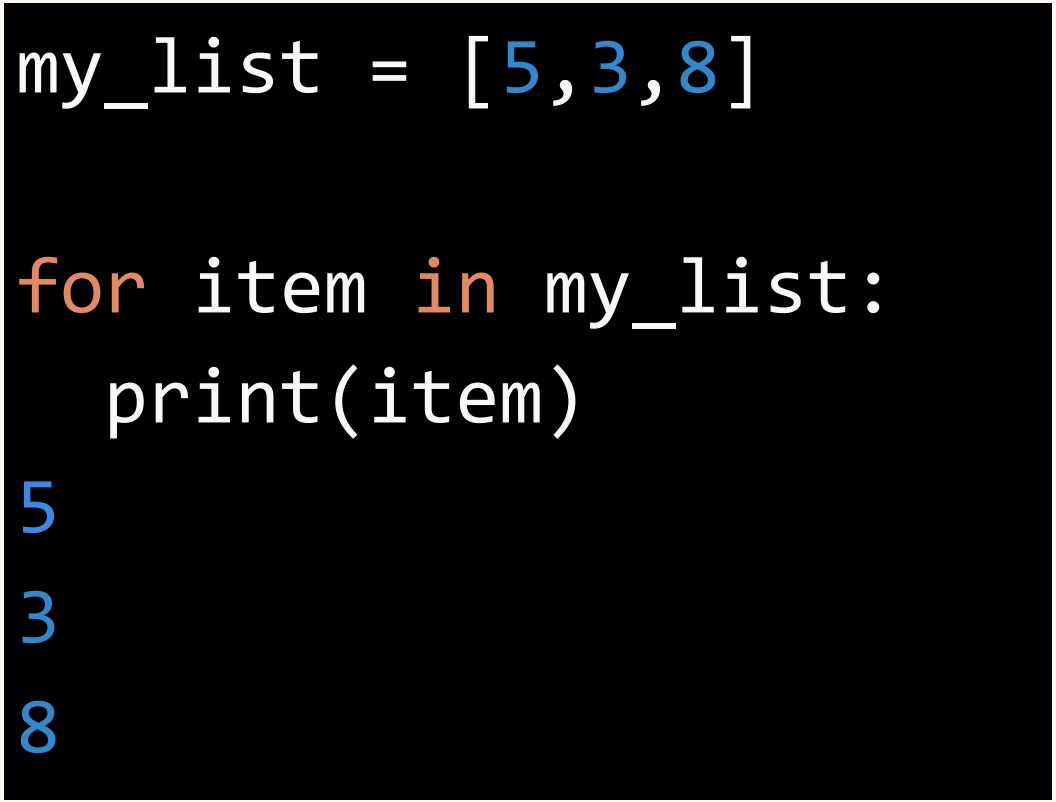
Feature	Linked List	List (Python)
Data Structure	Node with pointers to neighbors	Array-based
Memory Usage	Less efficient (extra pointers)	More efficient
Access Time	O(n) - linear time	O(1) - constant time
Insertion/Deletion Time	O(1) at begging/middle/end	O(n) at begging/middle, O(1) at end
Implementation in Python	Custom or external libraries	Built-in
Resizing	Allocated/deallocated as needed	Automatic resizing (overhead)
Indexing/Searching	Not supported directly, O(n) time	Supported, O(1) time



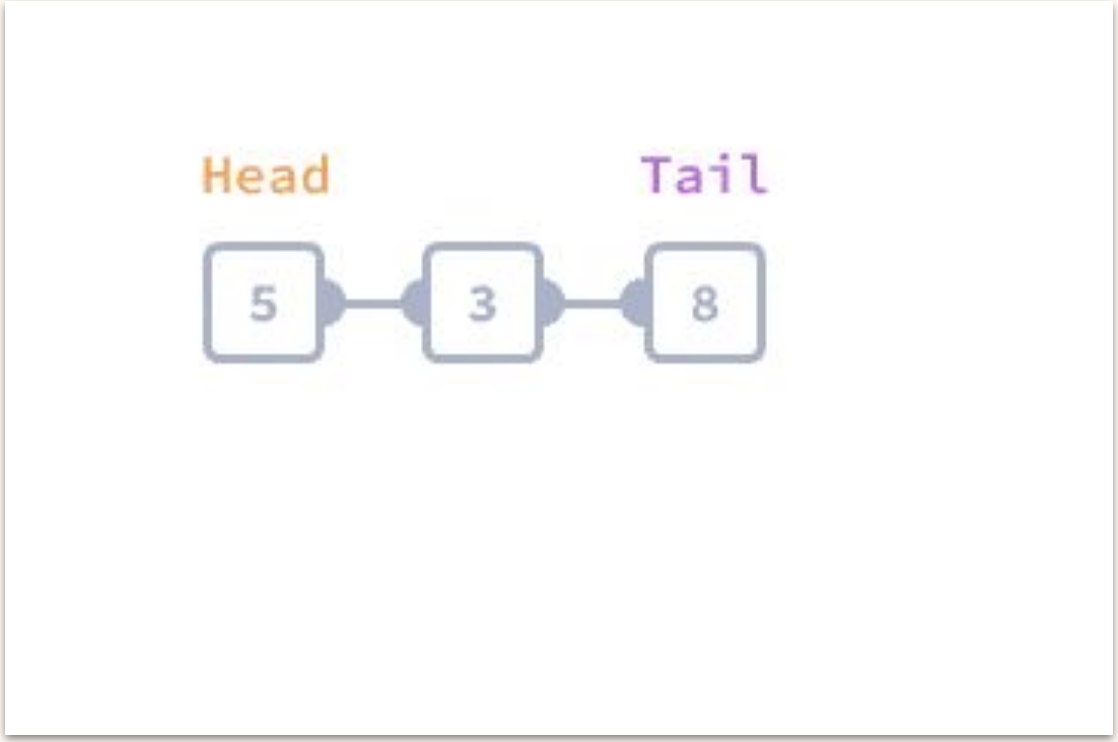

```python
class LinkedList:
    """Represents a doubly linked list.

    Attributes:
        head: The first node in the list.
        tail: The last node in the list.
        length: The number of nodes in the list.
    """

    def __init__(self):
        """Initializes a new instance"""
        self.head = None
        self.tail = None
        self.length = 0
```

```python
def append(self, data):
        """Adds a new node to the end of the list.

        Args:
            data: The value to be stored in
            the new node.
        """
        new_node = Node(data)
        if self.length == 0:
            self.head = self.tail = new_node
        else:
            self.tail.next = new_node
            new_node.prev = self.tail
            self.tail = new_node
        self.length += 1
```





Iterating Over List Elements



Iterating Over LinkedList Elements

In practice, to enable `for` loops, we define two methods in our class:

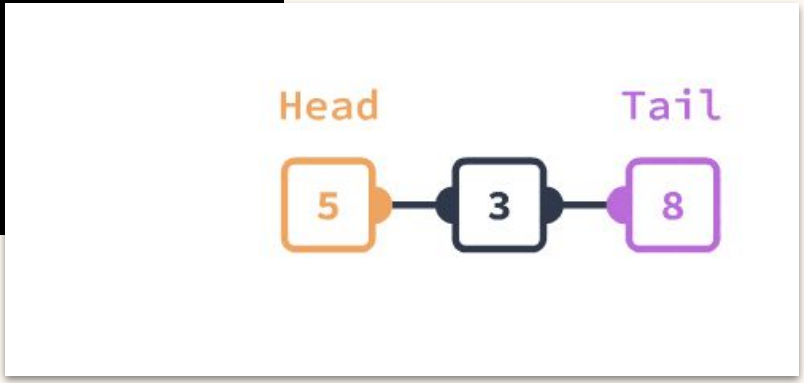
1. The `__iter__()` method: This method should set up all the necessary data to start a new iteration. When making a class iterable in this way, this method should always return `self`.

2. The `__next__()` method: This method should return the current iteration element and move on to the next one. It should also notify when the iteration is over.

```
  def __iter__(self):
        """Returns an iterator for the list."""
        self._iter_node = self.head
        return self

    def __next__(self):
        """Returns the next value in the list.

        Raises:
            StopIteration: If there are no more values in the list.
        """
        if self._iter_node is None:
            raise StopIteration
        ret = self._iter_node.data
        self._iter_node = self._iter_node.next
        return ret
```

```python
def prepend(self, data):
        """Adds a new node containing the given
         data to the beginning of the list.

        Args:
            data: The value to be stored in the new node.
        """
        new_node = Node(data)
        if self.length == 0:
            self.head = self.tail = new_node
        else:
            self.head.prev = new_node
            new_node.next = self.head
            self.head = new_node
        self.length += 1
```

Prepending Elements



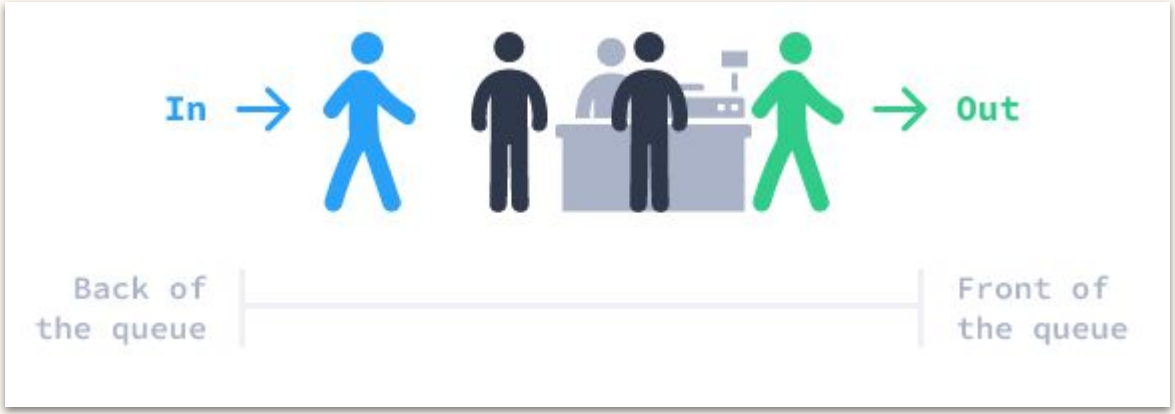
```
def __len__(self):
        """Returns the number of nodes in the list."""
        return self.length

def __str__(self):
        """Returns a string representation of the list."""
        return str([value for value in self])
```

```
lst = LinkedList()
print(lst) #[]

lst.append(1)
print(lst) #[1]
```

```
lst.append(2)
print(lst) #[1,2]

print(len(lst)) #2
```

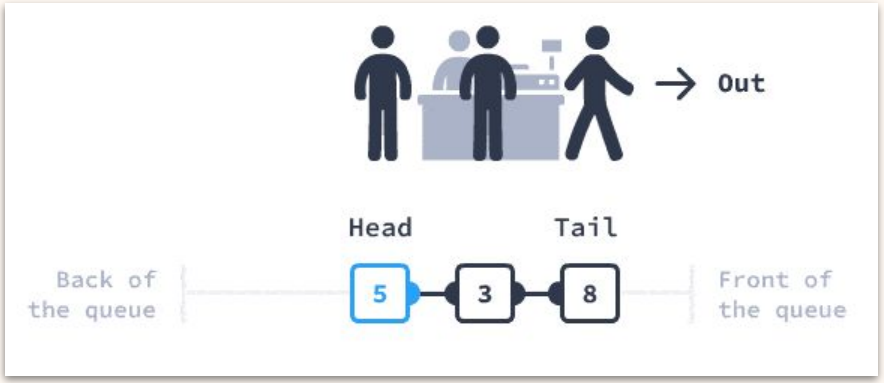
Data Structure Review

Queue



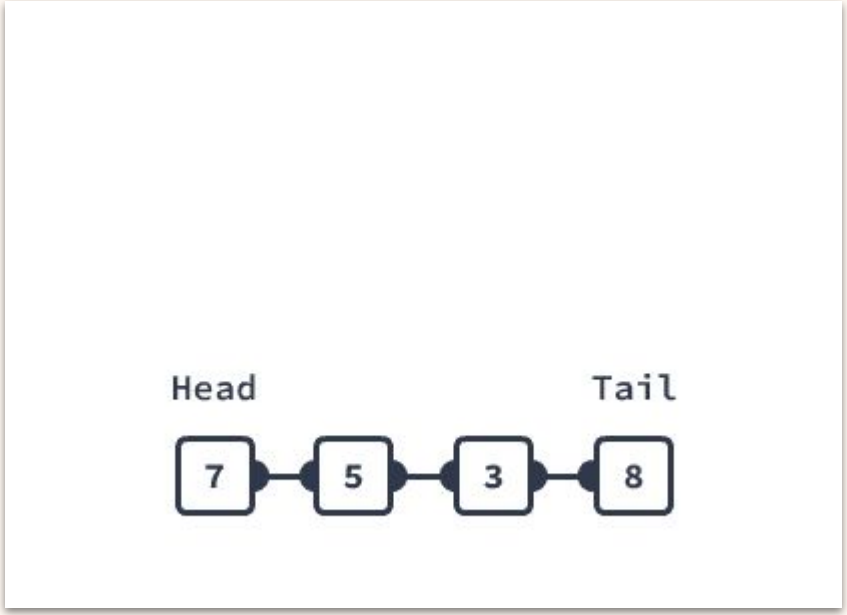
First-in, First-out (FIFO)

- enqueue()
- get_front()
- dequeue()
- FCFS Process Scheduling



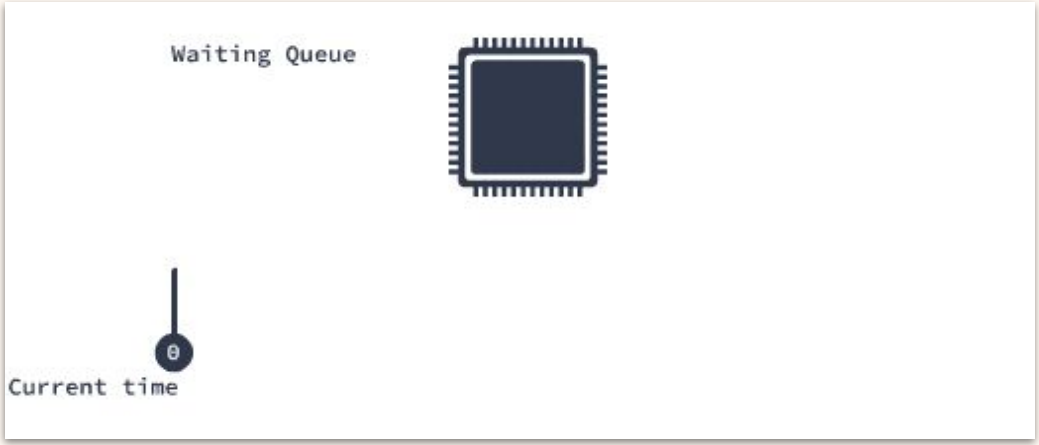

```
queue = Queue()
for i in [8, 3, 5, 1]:
   queue.enqueue(i)

print(queue)   #[1,5,3,8]
print(queue.get_front()) #8
```

```
class Queue(LinkedList):

    def enqueue(self, data):
        self.prepend(data)

    def get_front(self):
        return self.tail.data

    def dequeue(self):
        ret = self.tail.data
        if self.length == 1:
            self.tail = self.head = None
        else:
            self.tail = self.tail.prev
            self.tail.next = None
        self.length -= 1
        return ret
```




```
queue = Queue()
for i in [8, 3, 5, 7]:
   queue.enqueue(i)
front = queue.dequeue()
print(queue.get_front()) #3
```

```
class Queue(LinkedList):

    def enqueue(self, data):
        self.prepend(data)

    def get_front(self):
        return self.tail.data

    def dequeue(self):
        ret = self.tail.data
        if self.length == 1:
            self.tail = self.head = None
        else:
            self.tail = self.tail.prev
            self.tail.next = None
        self.length -= 1
        return ret
```



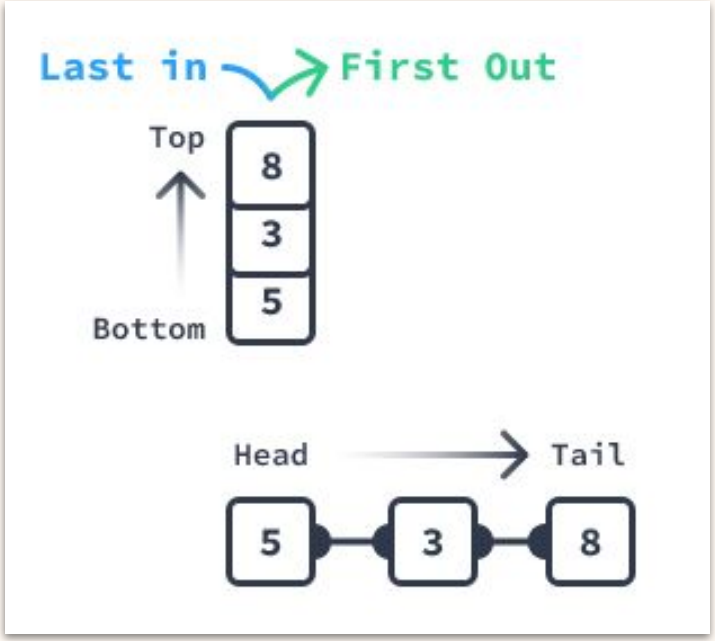
First Come - First Service (FCFS) Process Scheduling

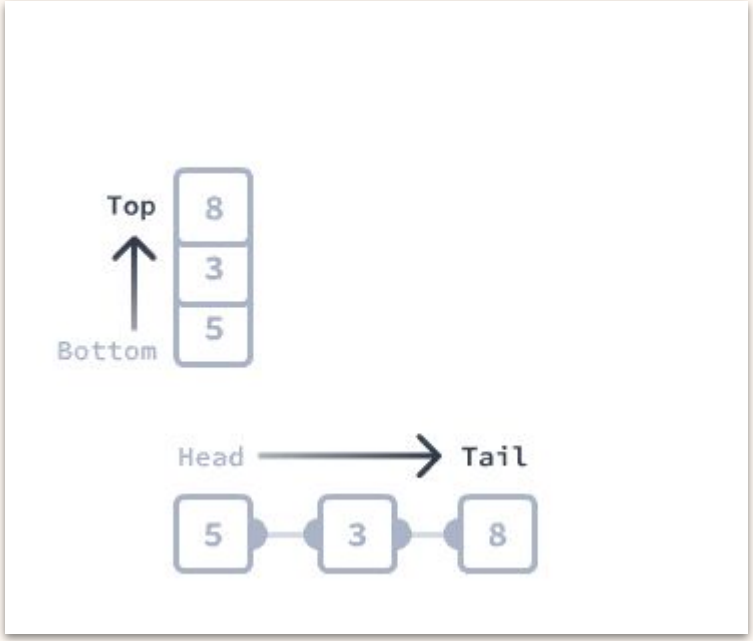
Non-Preemptive

PID	Arrival	Duration
P1	2	2
P2	0	1
P3	3	3
P4	3	5
P5	4	4

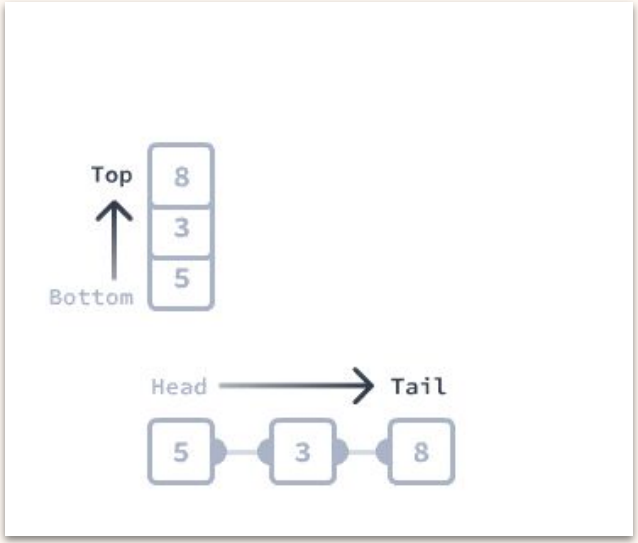
Data Structure Review

Stack



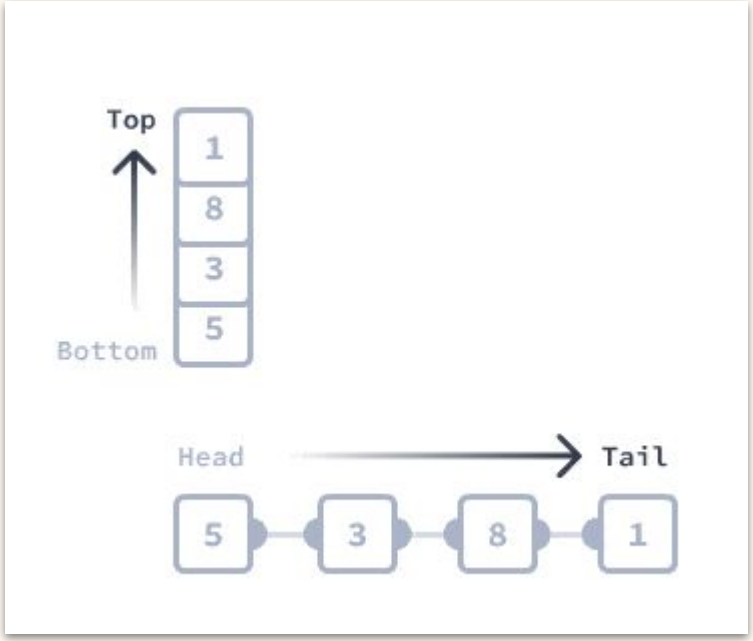


- push(), peek(), pop()
- LCFS Process Scheduling




```
stack = Stack()
for i in [5, 3, 8, 1]:
   stack.push(i)

print(stack.peek()) #1
print(stack) #[5,3,8,1]
```

```
class Stack(LinkedList):

    def push(self, data):
        self.append(data)

    def peek(self):
        return self.tail.data

    # Add pop() method here
    def pop(self):
        ret = self.tail.data
        if self.length == 1:
            self.tail = self.head = None
        else:
            self.tail = self.tail.prev
            self.tail.next = None
        self.length -= 1
        return ret
```



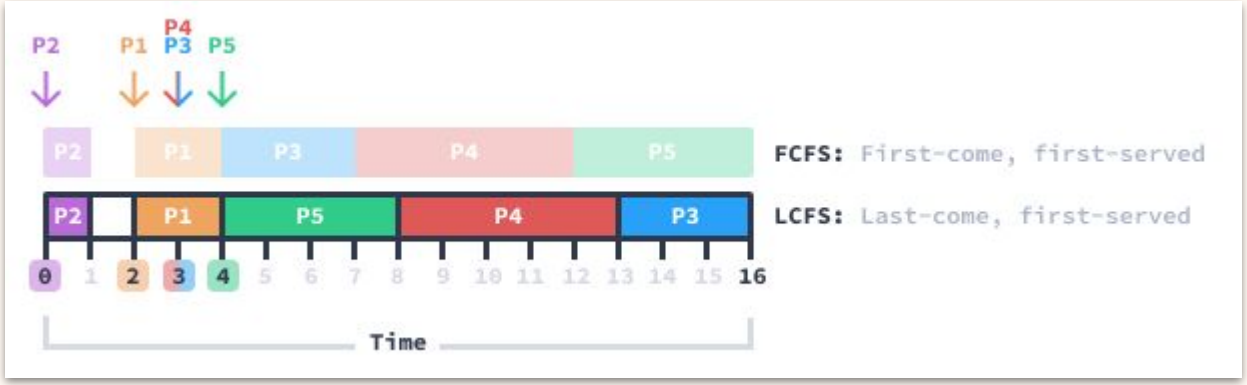

```
stack = Stack()
for i in [5, 3, 8, 1]:
   stack.push(i)
top = stack.pop()
print(stack.peek())#8
```

```
class Stack(LinkedList):

    def push(self, data):
        self.append(data)

    def peek(self):
        return self.tail.data
```

```
    # Add pop() method here
    def pop(self):
        ret = self.tail.data
        if self.length == 1:
            self.tail = self.head = None
        else:
            self.tail = self.tail.prev
            self.tail.next = None
        self.length -= 1
        return ret
```

Last Come - First Service (LCFS) Process Scheduling

PID	Arrival	Duration
P1	2	2
P2	0	1
P3	3	3
P4	3	5
P5	4	4



Course

DATAQUEST

Introduction to Data Structures

In this course, you'll learn how to optimize your data analysis using data structures — and how to improve performance on common tasks like searching and sorting.

Enroll For Free

Linked Lists · 2h
Lesson Objectives

Queues · 1h
Lesson Objectives

Stacks · 1h
Lesson Objectives

Dictionaries · 2h
Lesson Objectives

Guided Project: Analyzing Stock Prices · 1h
Lesson Objectives

Guided Project: Evaluating Numerical Expressions · 1h
Lesson Objectives

- Key requirements to excel in programming interviews
- Strong grasp of common data structures
- Problem-solving techniques

Hard: "Mind Bender"

Medium: "Brain Teaser"

Easy: "Piece of Cake"