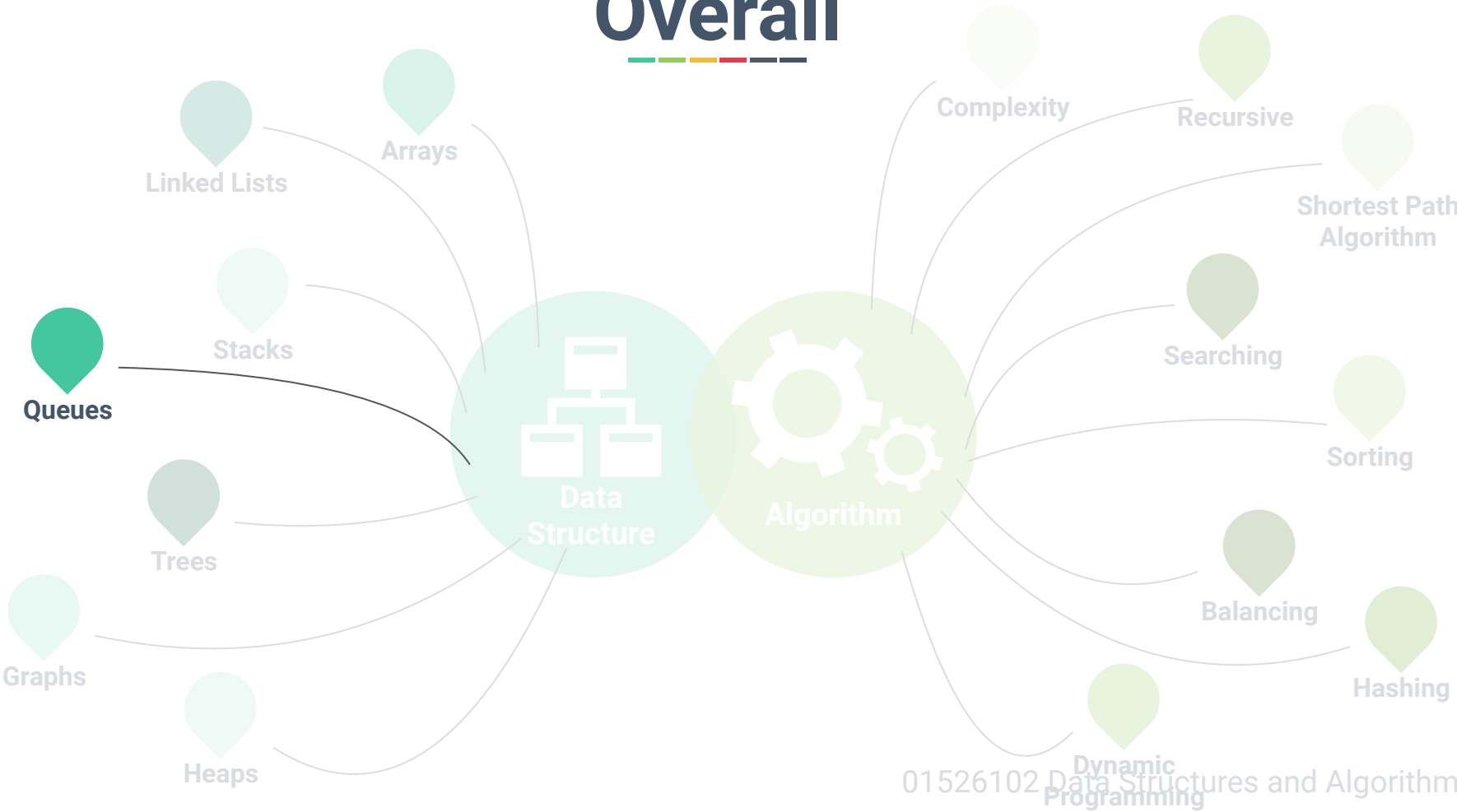
Chapter 4: Queues

Dr. Sirasit Lochanachit

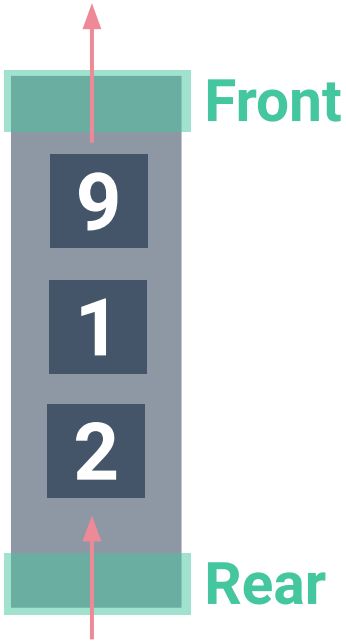
Overall


Linked Lists
Arrays
Stacks
Queues
Trees
Graphs
Heaps
Data Structure
Algorithm
Complexity
Recursive
Shortest Path Algorithm
Searching
Sorting
Balancing
Hashing
Dynamic Programming

Today's Outline

1. What is a Queue?
2. Queue Abstract Data Type
3. Array-Based Queue Implementations
    - Simple Queue
    - Revised Queue
    - Circular Queue

Queues



A queue is a collection, which keeps objects in a sequence, that are inserted and removed according to the **first-in first-out** (FIFO) principle [1].

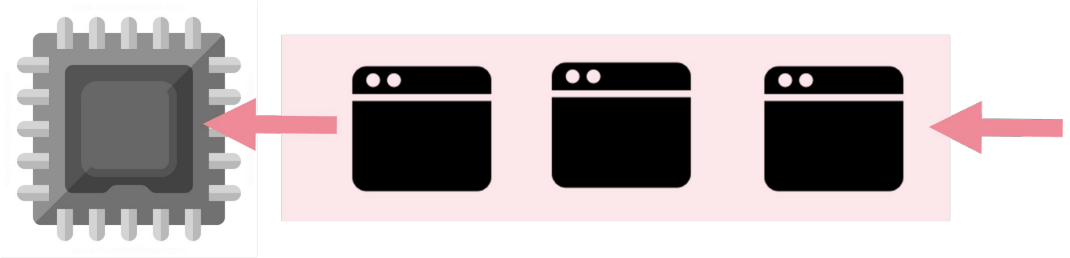
[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

Queues

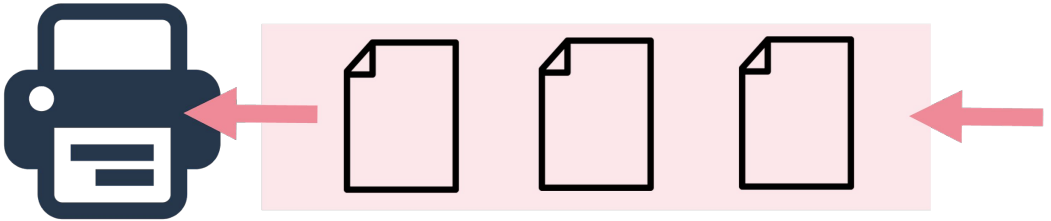
Real-life examples of queue:

Queues

Examples of queue:

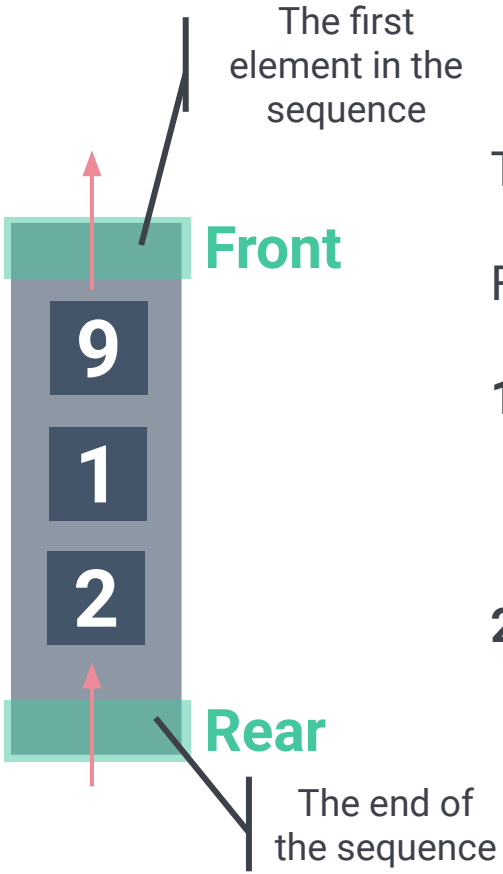


(i) a line of processes waiting to enter the CPU.



(ii) a line of paper waiting to print out.

The Queue Abstract Data Type



To create an instance of a queue, use Queue().

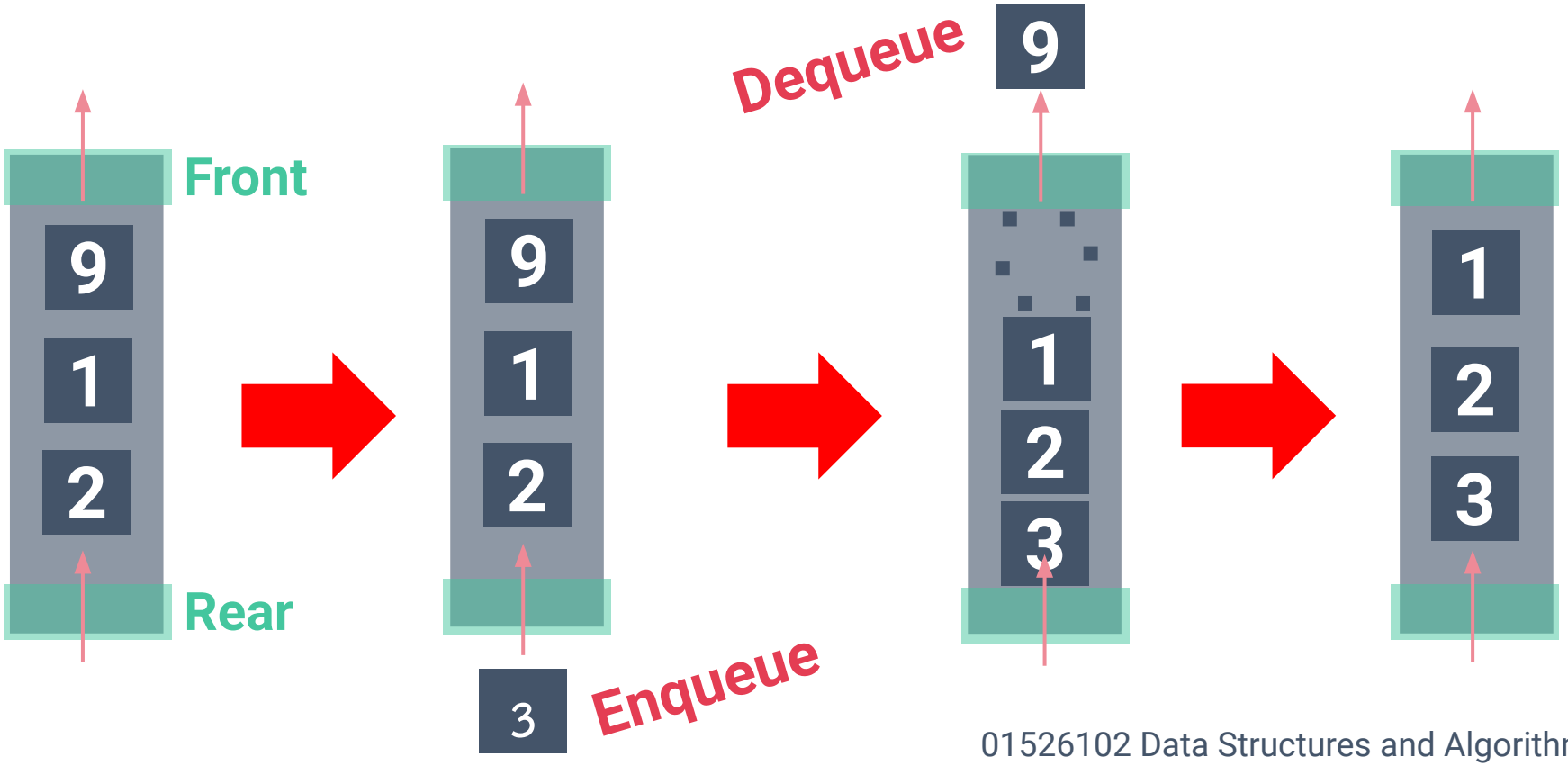
Formally, there are 2 main operations for queues:

1) enqueue(*item*)

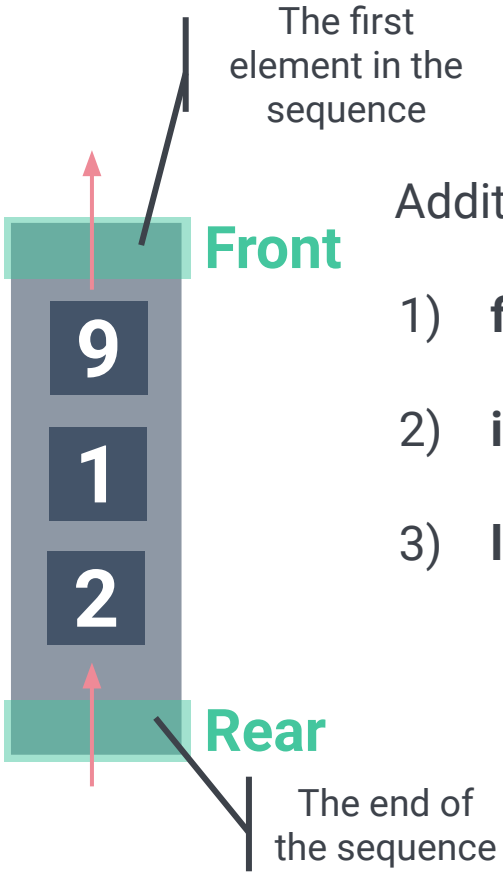
2) dequeue()

[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

Queues



The Queue Abstract Data Type

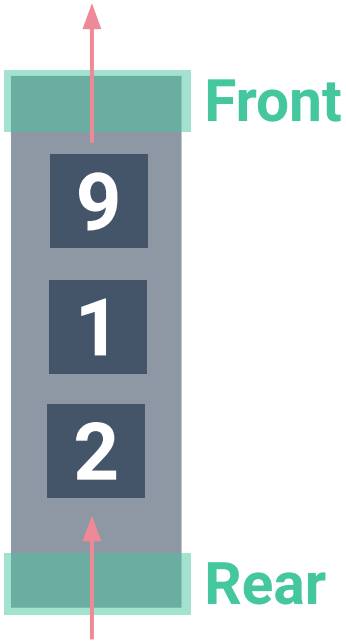


Additional operations for queues:

1) **first()**
2) **is_empty()**
3) **len(Queue)**

[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

Queues



How to Implement a Queue?

Operation Example

Operation	Return Value
Q.enqueue(9)	-
Q.enqueue(5)	-
Q.first()	
Q.enqueue(2)	-
Q.dequeue()	
Q.is_empty()	
len(Q)	

Queue
[first, ..., last]

Asymptotic Performance

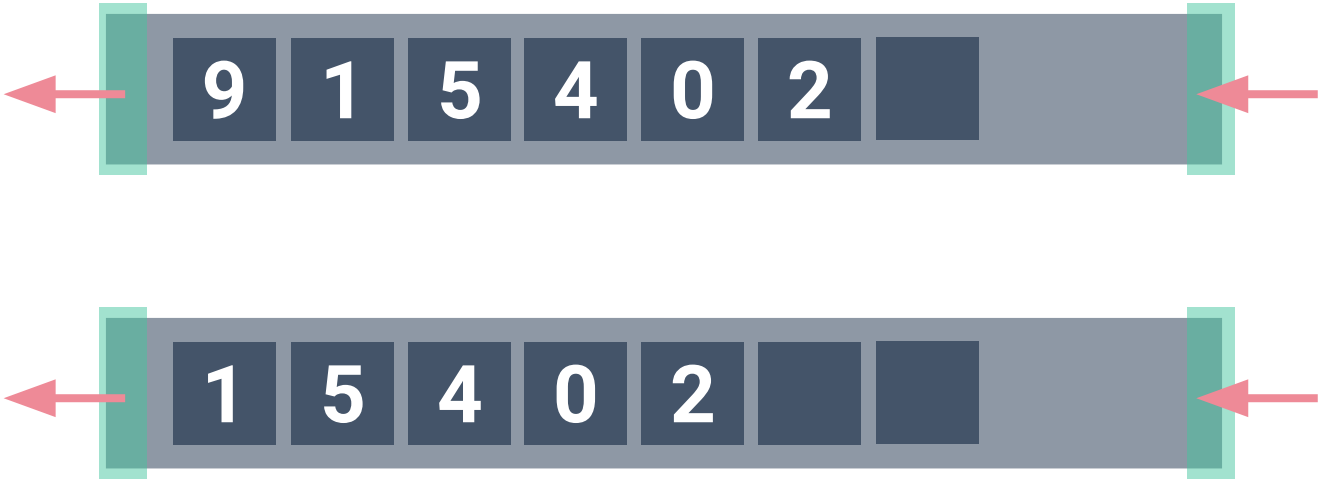
Operation	Running Time
Q.enqueue(item)	
Q.dequeue()	
Q.first()	
Q.is_empty()	
len(Q)	

Simple Queue

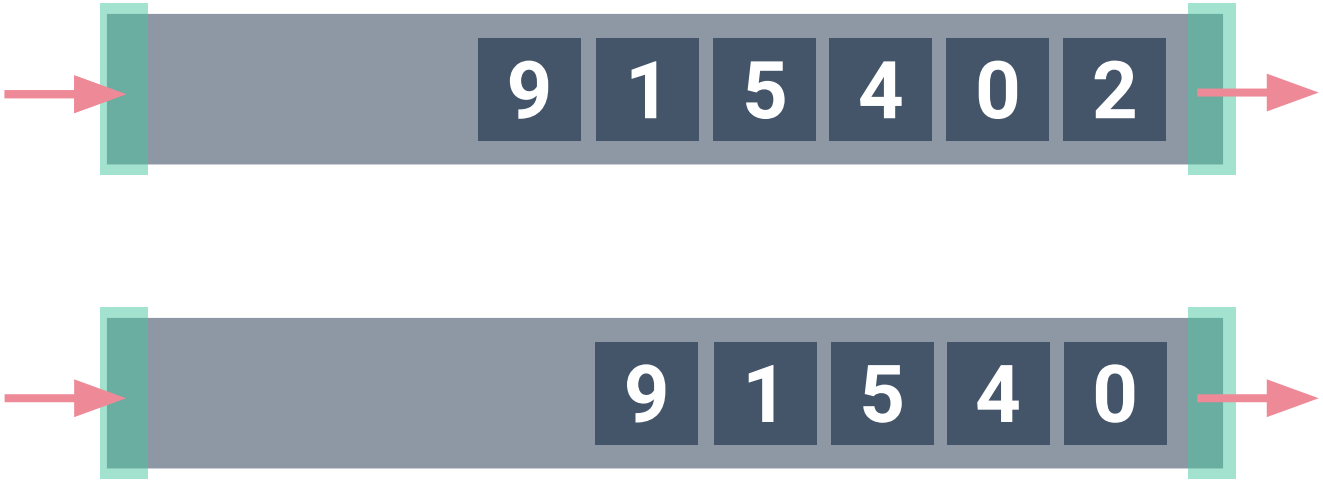
Suppose a Python list is created.

- Enqueue
- Dequeue

Simple Queue



Alternative Simple Queue



Suppose a Python list is created.

- Enqueue
- Dequeue

Revised Queue

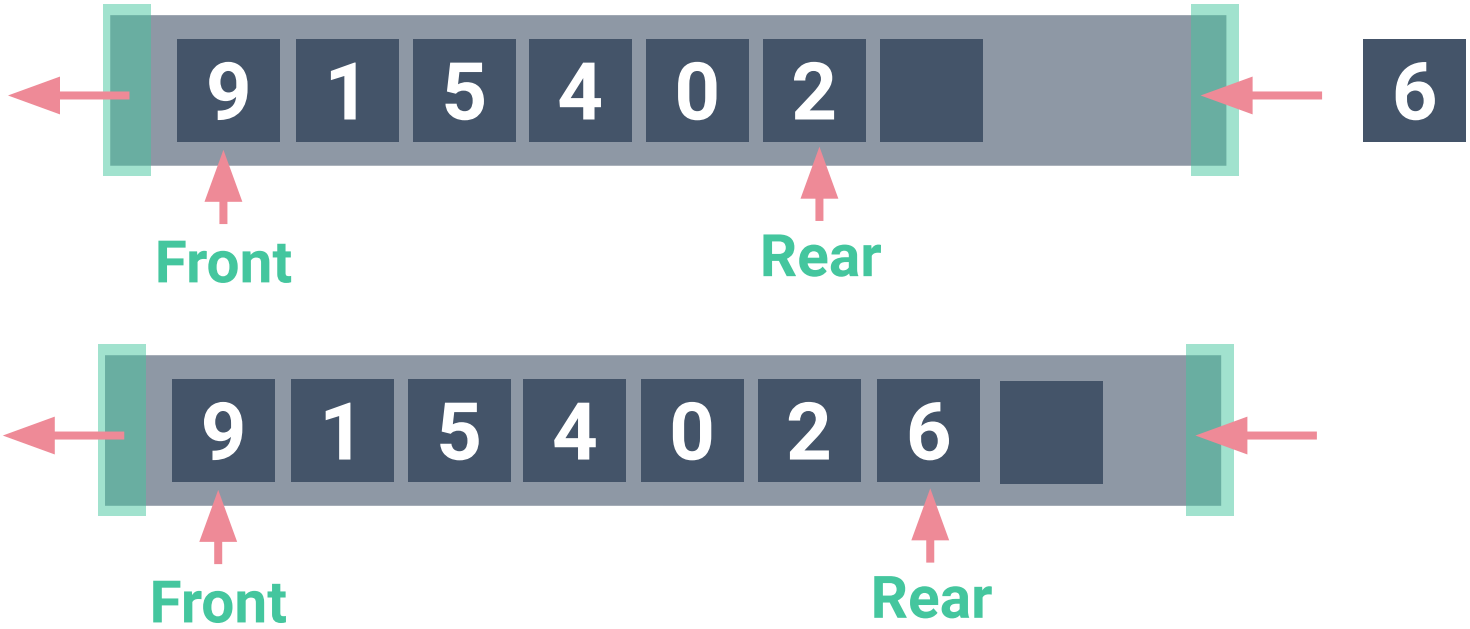
A better approach is to assign two main pointers to store indices that refer to:

the first element of the queue, which is called the ***Front***,

and the last element of the queue, which is known as the ***Rear***.

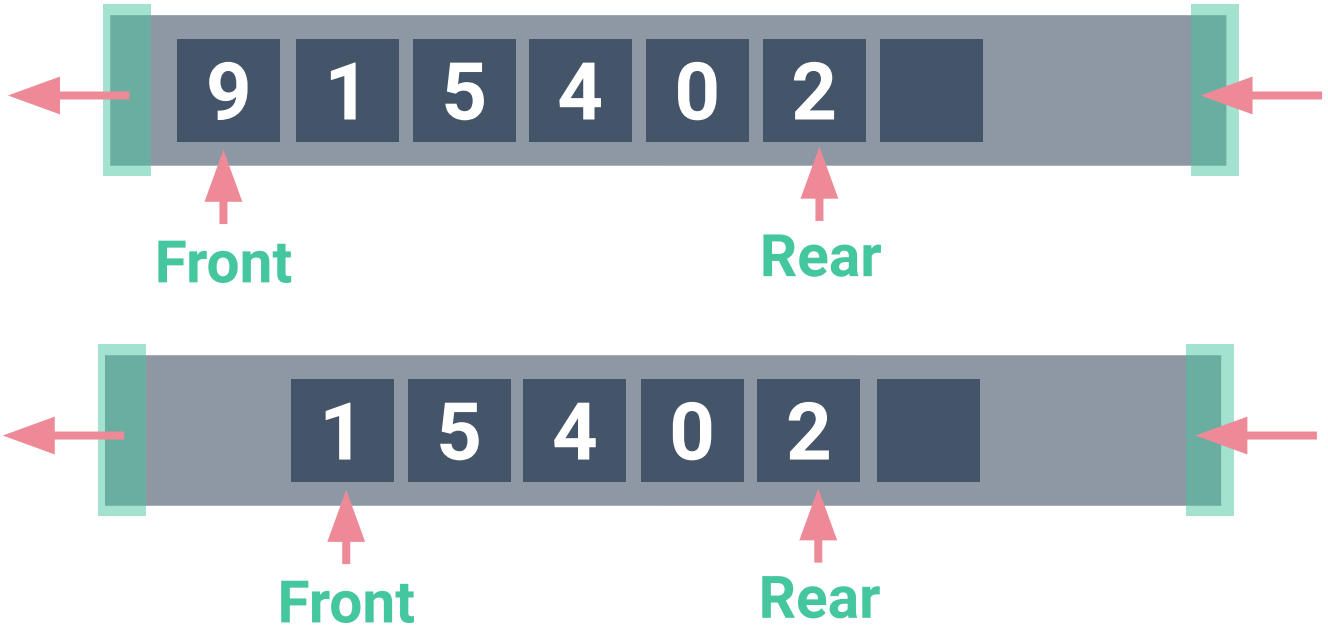
Revised Queue

- **Enqueue**

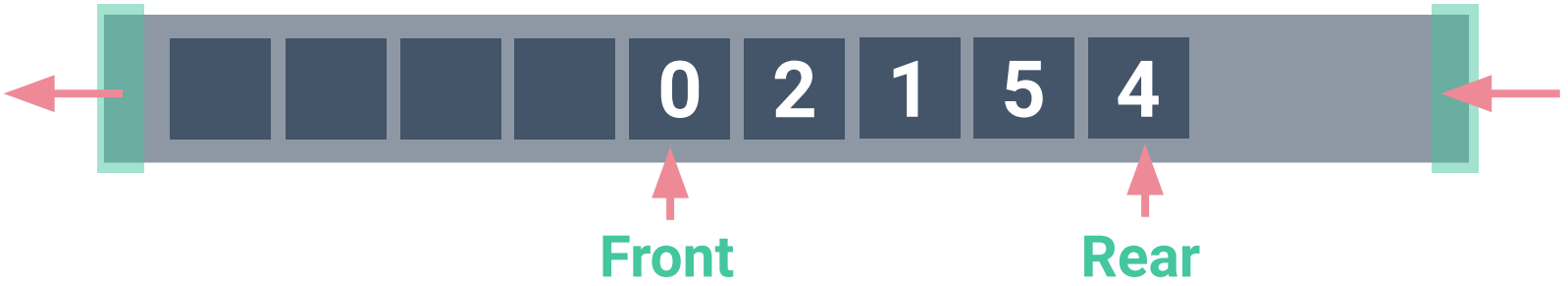


Revised Queue

- **Dequeue**



Revised Queue

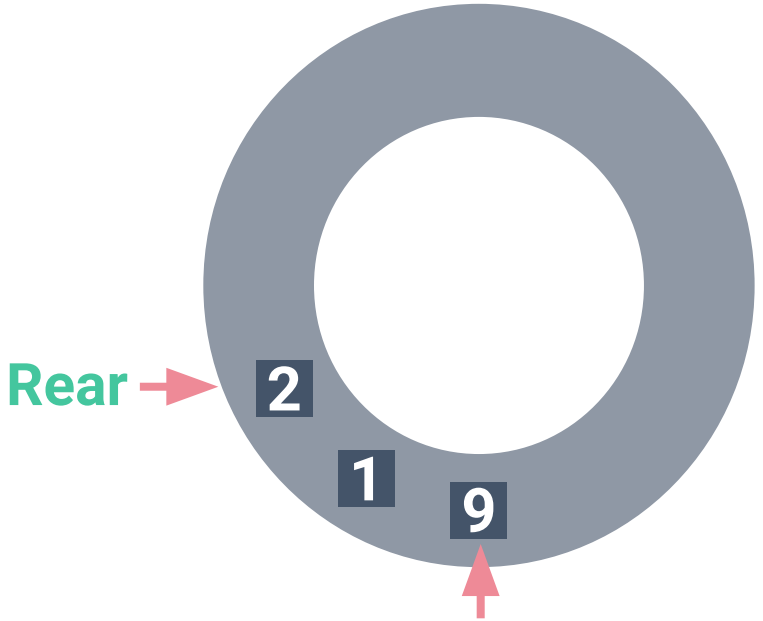


Drawback:

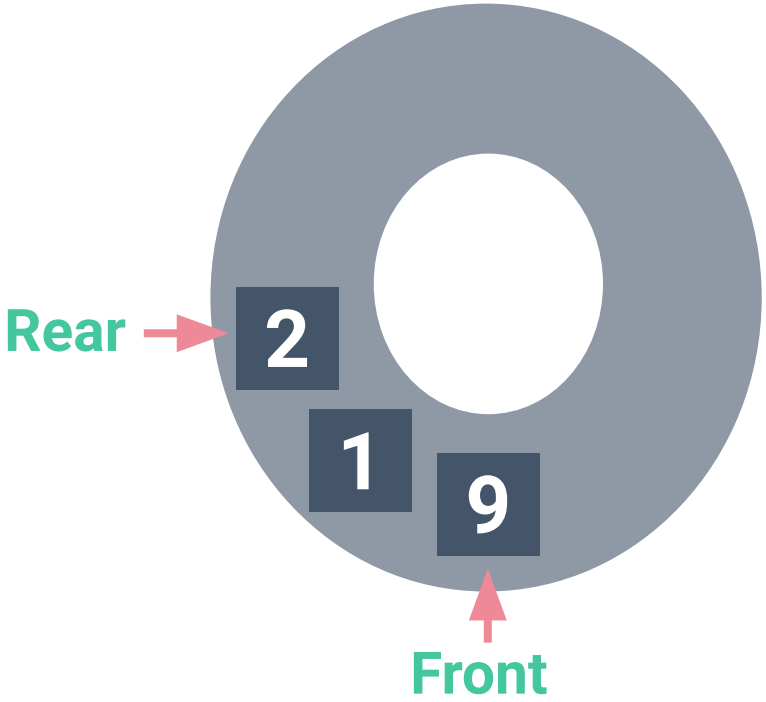
Types of Queue

- Queue

- Circular Queue



Circular Queue

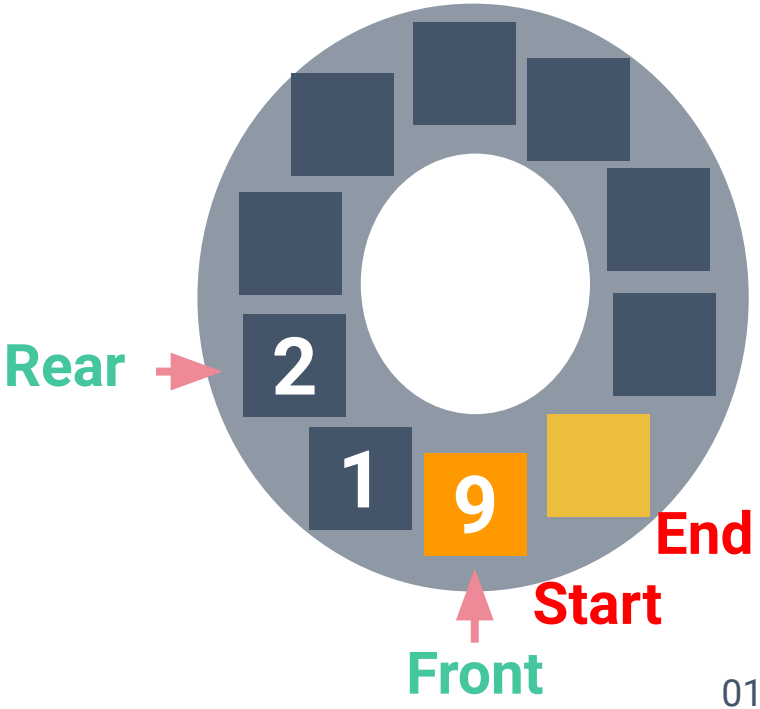


In addition to allowing the pointers to shift rightward, the elements of the queue are wrap around at the end of an array.

Circular Queue

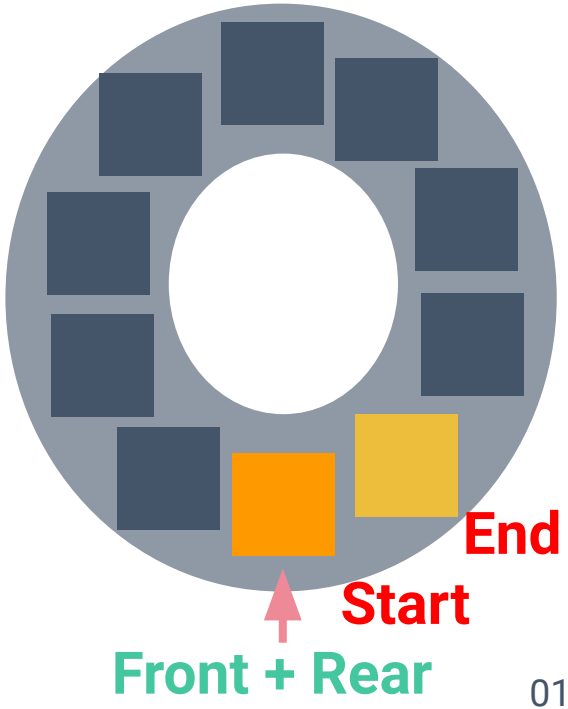
Suppose a queue of length 10,

Circular Queue

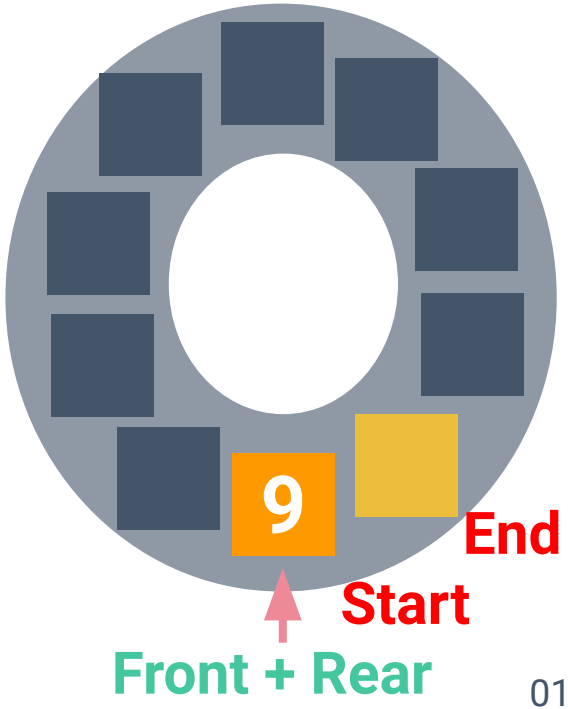


Circular Queue

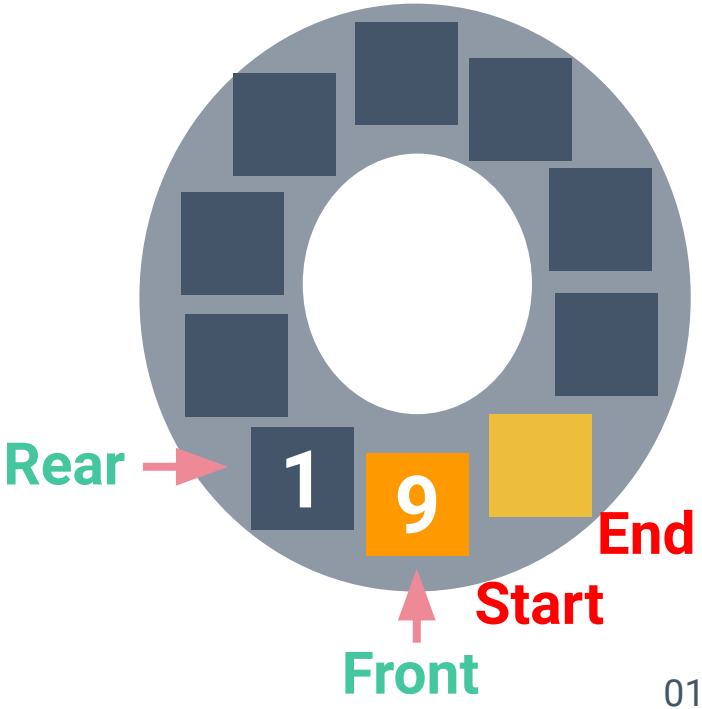
Empty Queue



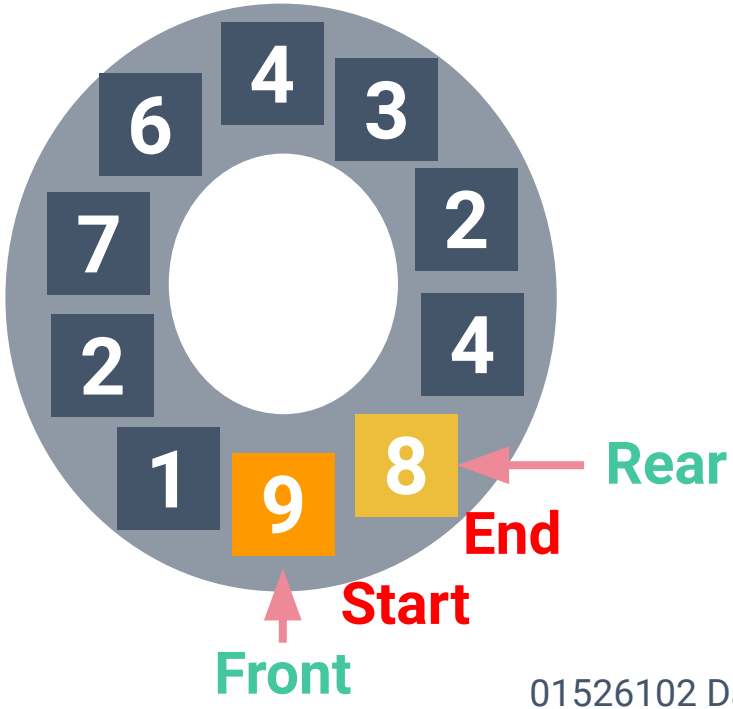
Circular Queue



Circular Queue

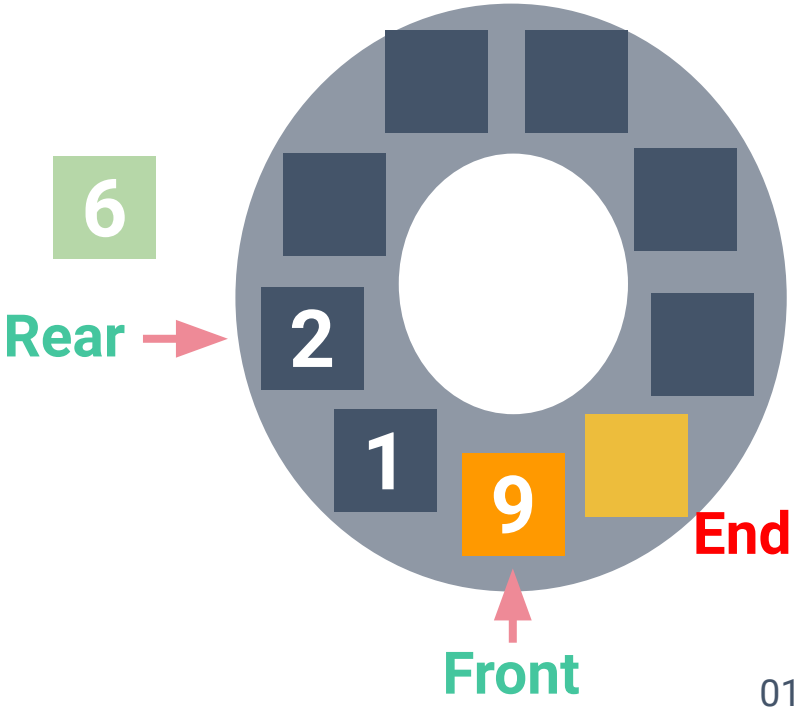


Circular Queue



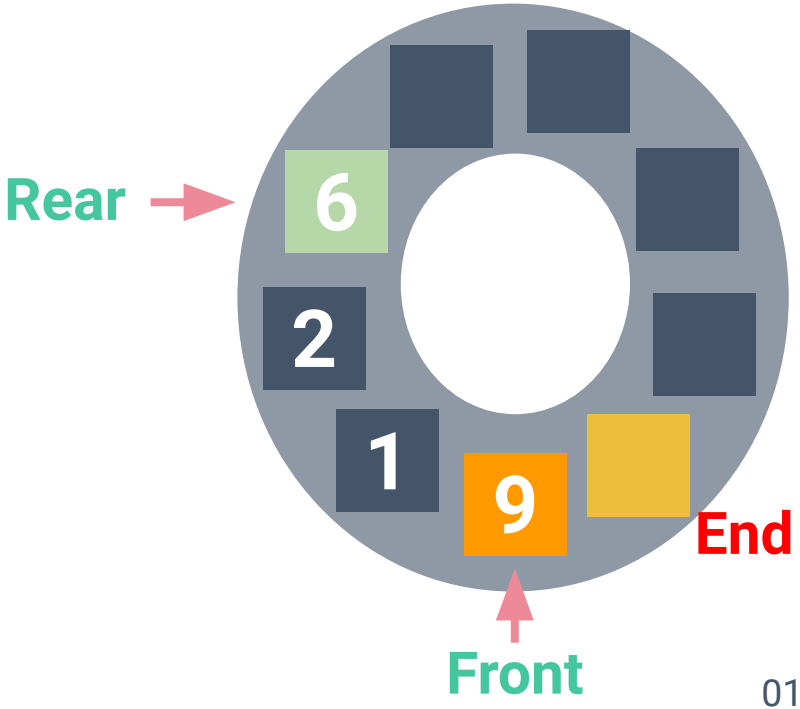
Circular Queue

Enqueue

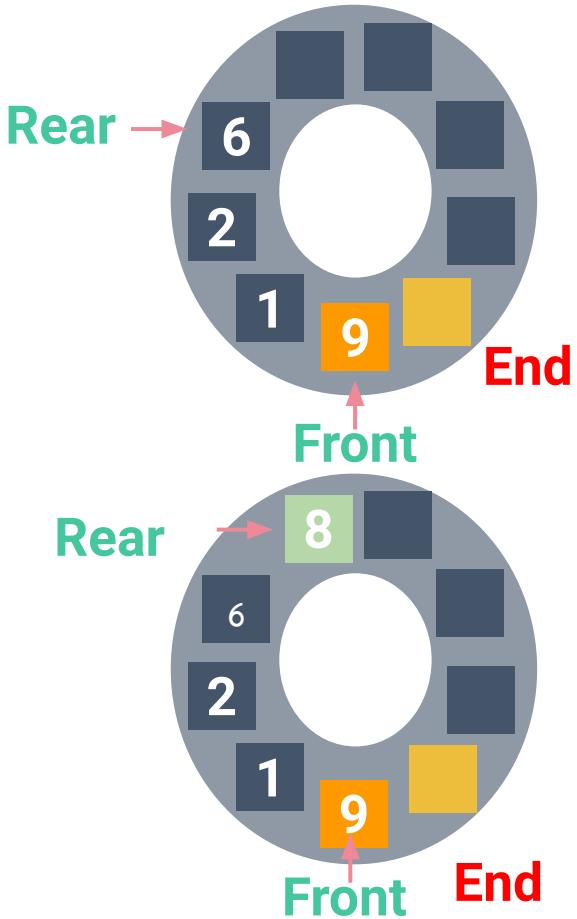


Circular Queue

Enqueue



Circular Queue in Python



When **enqueue**, the rear pointer is given by

$$r = (r + 1) \% N$$

where N is the array length.

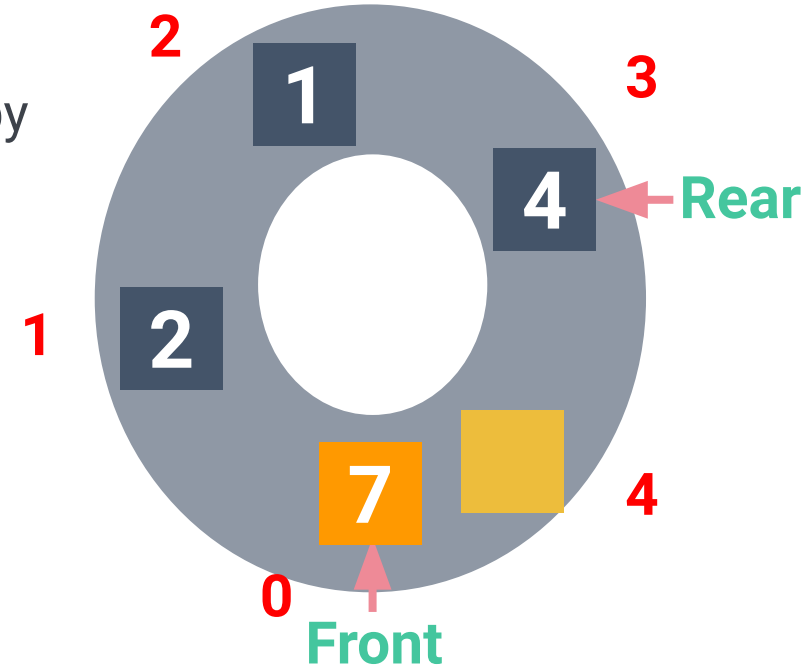
* Note: % denotes ***modulo*** operation in python which provides the remainder after division.

Circular Queue in Python

When **enqueue**, the rear pointer is given by

$$r = (r + 1) \% N$$

where N is the array length.

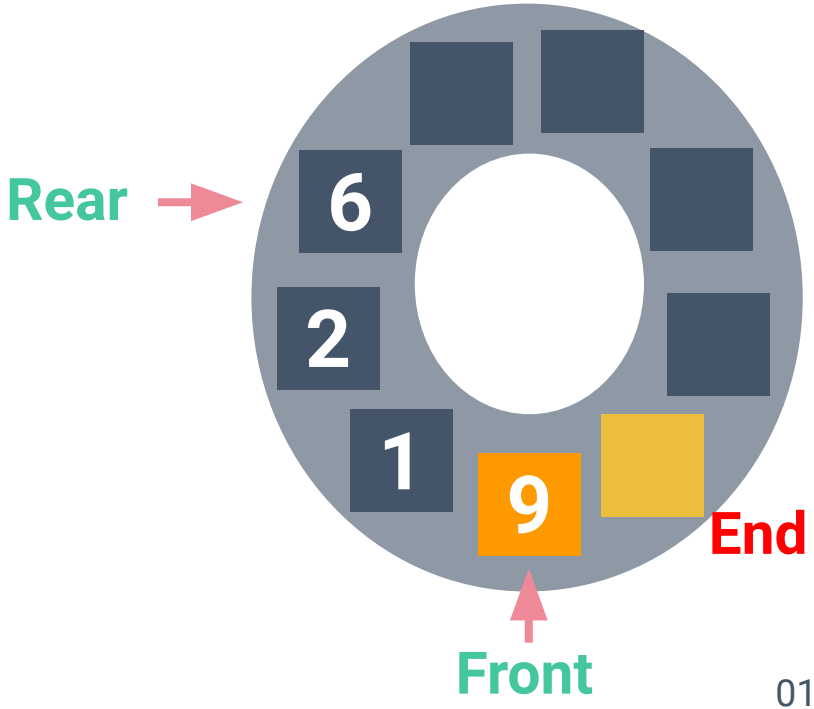


For example, given a list of length 5, current rear pointer is at 3.

When an enqueue is called, the new index for rear pointer is (3+1) % 5 = 4.

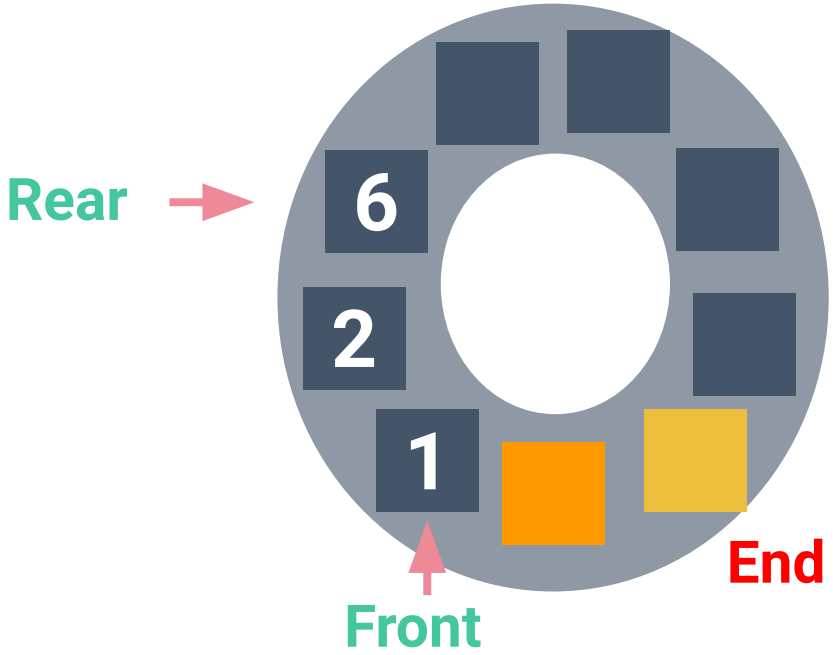
Circular Queue

Dequeue

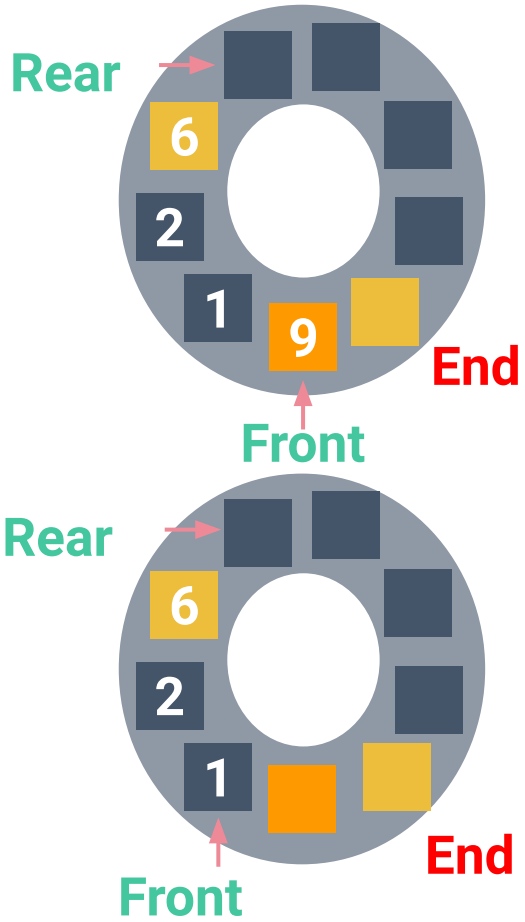


Circular Queue

Dequeue



Circular Queue in Python



When **dequeue**, the front pointer is given by

$$f = (f + 1) \% N$$

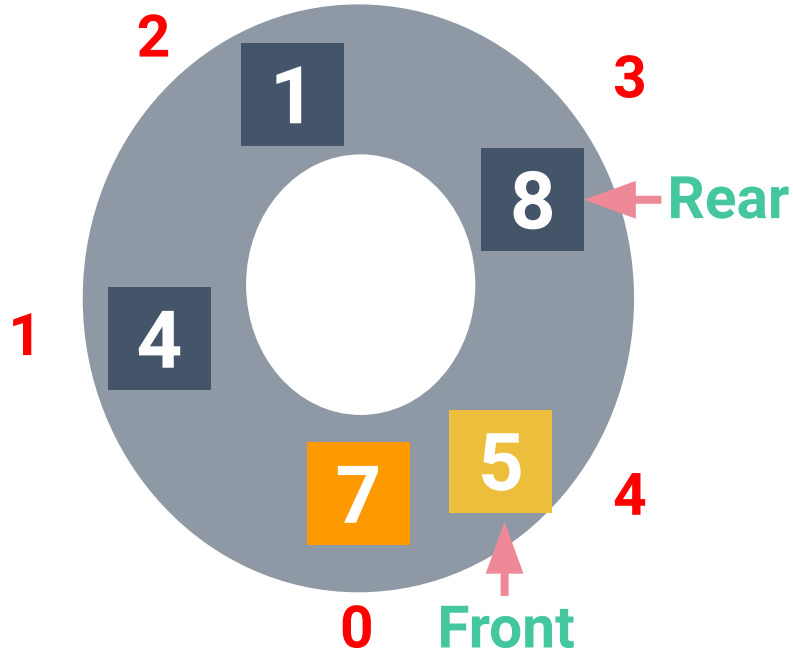
where N is the array length.

Circular Queue in Python

When **dequeue**, the front index is given by

$$f = (f + 1) \% N$$

where N is the array length.



For example, given a list of length 5, current front pointer is at 4.

When a dequeue is called, the new index for front pointer is (4+1) % 5 = 0.