

Algoritmos e Estruturas de Dados I

FILAS

Prof. Tiago Eugenio de Melo
tmelo@uea.edu.br

www.tiagodemelo.info

Observações

- O conteúdo dessa aula é parcialmente proveniente do Capítulo 6 do livro “*Data Structures and Algorithms in Python*”.
- As palavras com a fonte `Courier` indicam uma palavra-reservada da linguagem de programação.

Introdução

Introdução

- Coleção de objetos que são inseridos e removidos de acordo com o princípio **FIFO** (*first in, first out*).

Introdução

- Coleção de objetos que são inseridos e removidos de acordo com o princípio **FIFO** (*first in, first out*).
- Os elementos podem ser inseridos a qualquer momento, mas existe uma ordem na remoção dos elementos, pois sairá da fila o elemento mais antigo.

Introdução

- A noção de fila é usada no nosso cotidiano:
 - Lojas
 - Cinema
 - Restaurante
 - Computadores
 - Fila de impressão.
 - Servidor web.
 - Etc.

TAD Fila

TAD Fila

- TAD que representa uma fila mantém os elementos em uma sequência.

TAD Fila

- TAD que representa uma fila mantém os elementos em uma sequência.
- As operações de acesso e a remoção de elementos são restritas ao **primeiro** elemento.

TAD Fila

- TAD que representa uma fila mantém os elementos em uma sequência.
- As operações de acesso e a remoção de elementos são restritas ao **primeiro** elemento.
- A operação de inserção é restrita ao **final** da fila.

TAD Fila

- Uma fila **Q** deverá ter os seguintes métodos:

TAD Fila

- Uma fila **Q** deverá ter os seguintes métodos:
 - **Q.enqueue (e)**
 - Adiciona o elemento **e** no final da fila **Q**.

TAD Fila

- Uma fila **Q** deverá ter os seguintes métodos:
 - **Q.enqueue (e)**
 - Adiciona o elemento **e** no final da fila **Q**.
 - **Q.dequeue ()**
 - Remove e retorna o primeiro elemento da fila **Q**.
 - Ocorre um erro se a fila estiver vazia.

TAD Fila

- Outros métodos adicionais:

TAD Fila

- Outros métodos adicionais:
 - **Q.first ()**
 - Retorna o primeiro elemento da fila **Q**, mas sem removê-lo.
 - Ocorrerá um erro se a fila **Q** estiver vazia.

TAD Fila

- Outros métodos adicionais:
 - **Q.first ()**
 - Retorna o primeiro elemento da fila **Q**, mas sem removê-lo.
 - Ocorrerá um erro se a fila **Q** estiver vazia.
 - **Q.is_empty ()**
 - Retorna `True` se a fila **Q** estiver vazia.

TAD Fila

- Outros métodos adicionais:
 - **Q.first ()**
 - Retorna o primeiro elemento da fila **Q**, mas sem removê-lo.
 - Ocorrerá um erro se a fila **Q** estiver vazia.
 - **Q.is_empty ()**
 - Retorna `True` se a fila **Q** estiver vazia.
 - **len (Q)**
 - Retorna o número de elementos da fila **Q**.
 - Em Python, nós podemos implementar isto como um tipo especial com o método `__len__`.

TAD Fila (exemplo)

Operation	Return Value	first \leftarrow Q \leftarrow last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue()	5	[3]
Q.is_empty()	False	[3]
Q.dequeue()	3	[]
Q.is_empty()	True	[]

TAD Fila (exemplo)

Operation	Return Value	first \leftarrow Q \leftarrow last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue()	5	[3]
Q.is_empty()	False	[3]
Q.dequeue()	3	[]
Q.is_empty()	True	[]
Q.dequeue()	“error”	[]

TAD Fila (exemplo)

Operation	Return Value	first ← Q ← last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue()	5	[3]
Q.is_empty()	False	[3]
Q.dequeue()	3	[]
Q.is_empty()	True	[]
Q.dequeue()	“error”	[]
Q.enqueue(7)	–	[7]

TAD Fila (exemplo)

Operation	Return Value	first ← Q ← last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue()	5	[3]
Q.is_empty()	False	[3]
Q.dequeue()	3	[]
Q.is_empty()	True	[]
Q.dequeue()	“error”	[]
Q.enqueue(7)	–	[7]
Q.enqueue(9)	–	[7, 9]

TAD Fila (exemplo)

Operation	Return Value	first \leftarrow Q \leftarrow last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue()	5	[3]
Q.is_empty()	False	[3]
Q.dequeue()	3	[]
Q.is_empty()	True	[]
Q.dequeue()	“error”	[]
Q.enqueue(7)	–	[7]
Q.enqueue(9)	–	[7, 9]
Q.first()	7	[7, 9]

TAD Fila (exemplo)

Operation	Return Value	first \leftarrow Q \leftarrow last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue()	5	[3]
Q.is_empty()	False	[3]
Q.dequeue()	3	[]
Q.is_empty()	True	[]
Q.dequeue()	“error”	[]
Q.enqueue(7)	–	[7]
Q.enqueue(9)	–	[7, 9]
Q.first()	7	[7, 9]
Q.enqueue(4)	–	[7, 9, 4]

TAD Fila (exemplo)

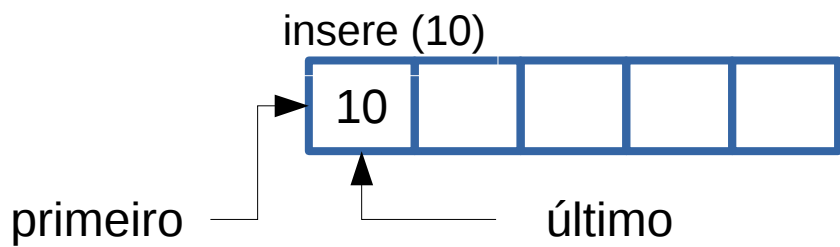
Operation	Return Value	first \leftarrow Q \leftarrow last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue()	5	[3]
Q.is_empty()	False	[3]
Q.dequeue()	3	[]
Q.is_empty()	True	[]
Q.dequeue()	“error”	[]
Q.enqueue(7)	–	[7]
Q.enqueue(9)	–	[7, 9]
Q.first()	7	[7, 9]
Q.enqueue(4)	–	[7, 9, 4]
len(Q)	3	[7, 9, 4]

TAD Fila (exemplo)

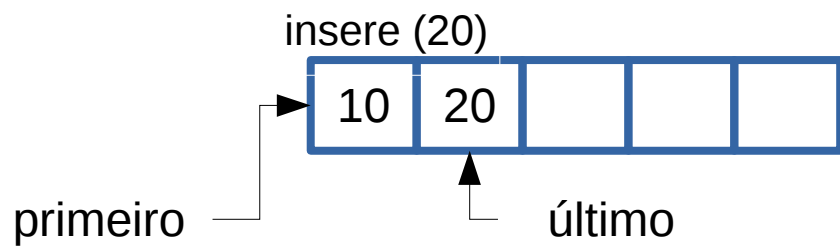
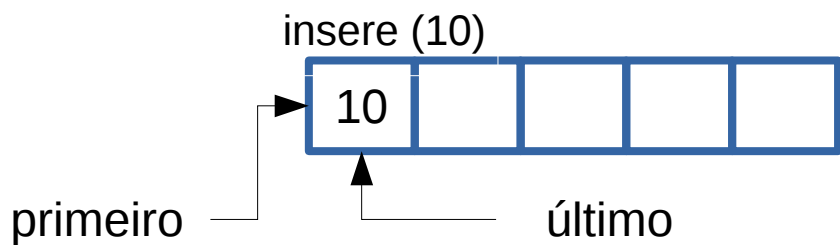
Operation	Return Value	first \leftarrow Q \leftarrow last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue()	5	[3]
Q.is_empty()	False	[3]
Q.dequeue()	3	[]
Q.is_empty()	True	[]
Q.dequeue()	“error”	[]
Q.enqueue(7)	–	[7]
Q.enqueue(9)	–	[7, 9]
Q.first()	7	[7, 9]
Q.enqueue(4)	–	[7, 9, 4]
len(Q)	3	[7, 9, 4]
Q.dequeue()	7	[9, 4]

Fila

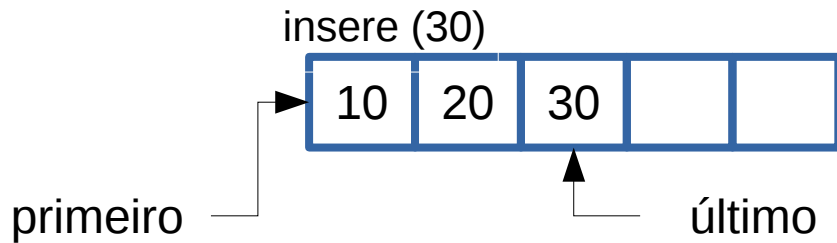
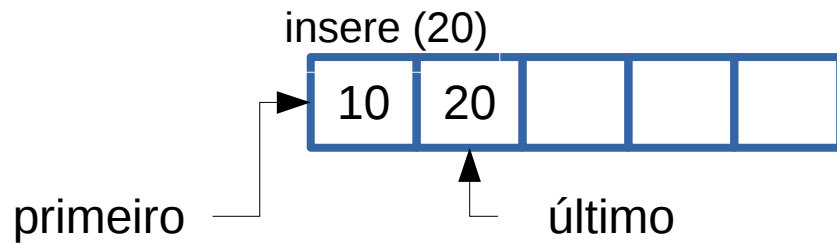
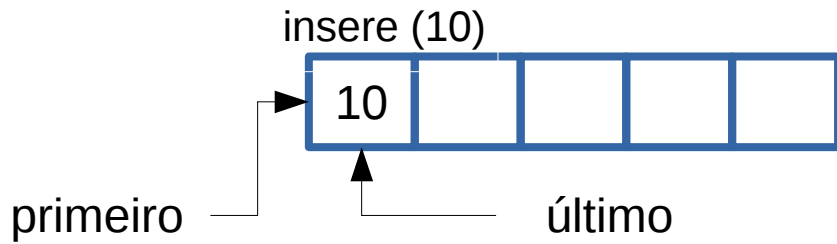
Fila



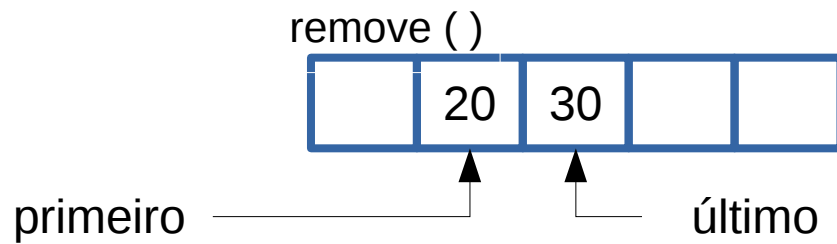
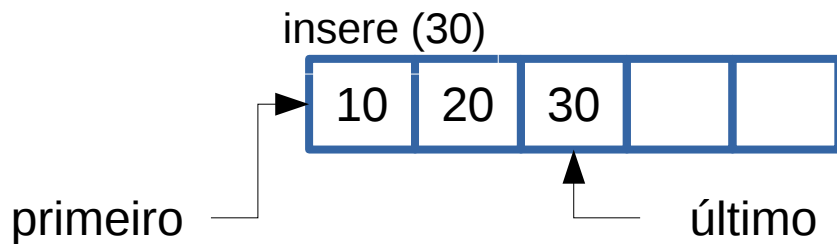
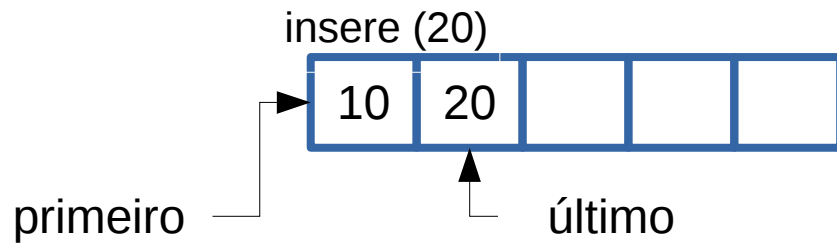
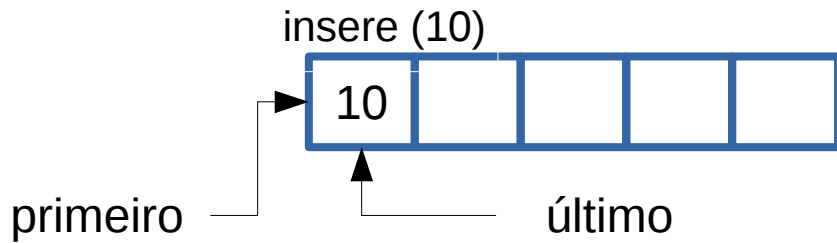
Fila



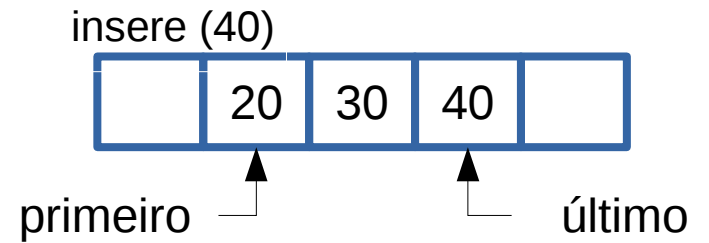
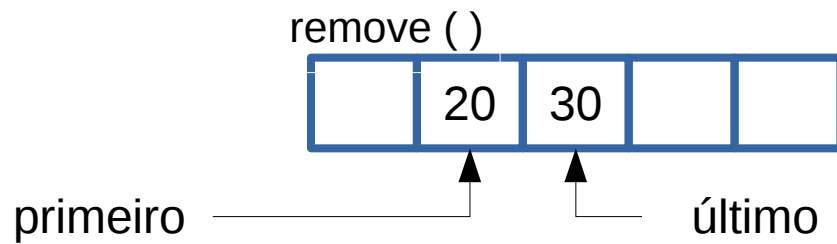
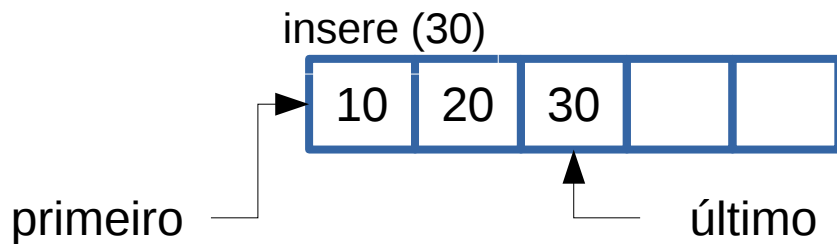
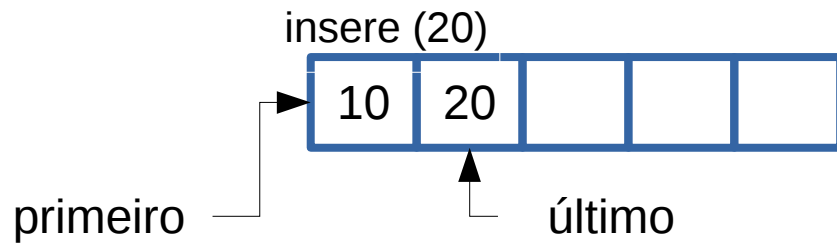
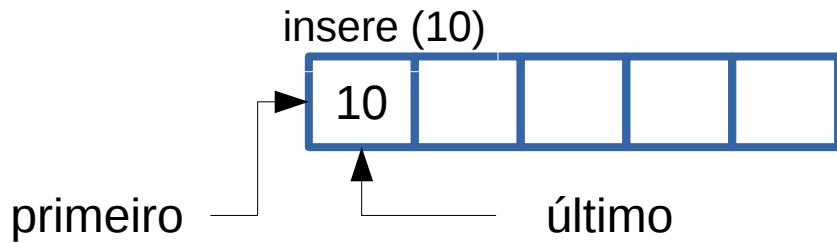
Fila



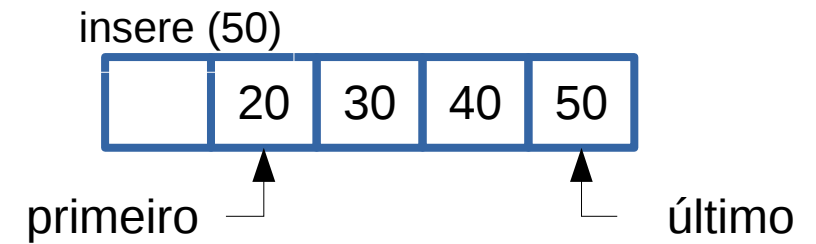
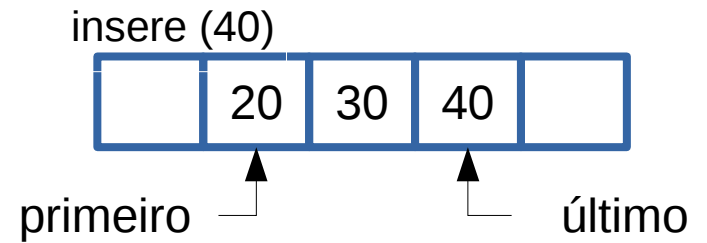
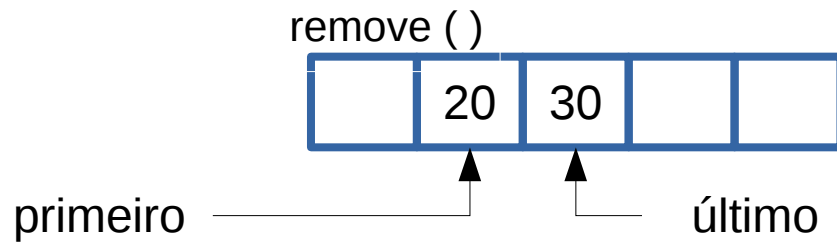
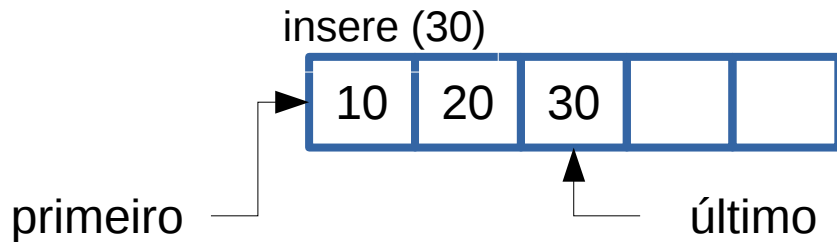
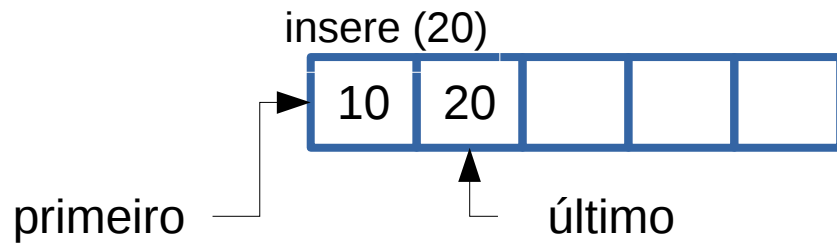
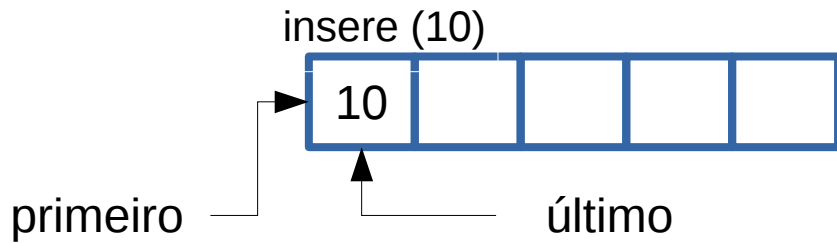
Fila



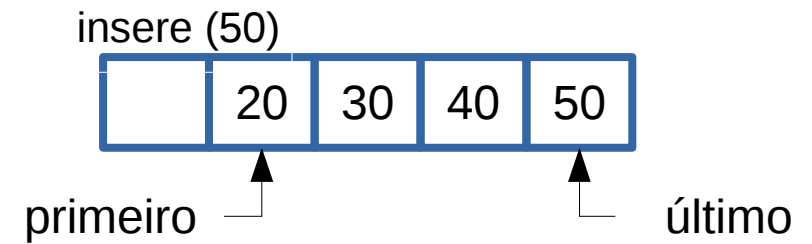
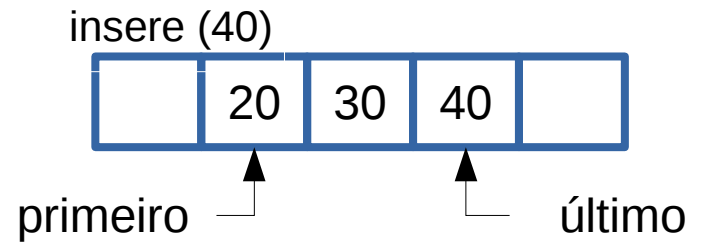
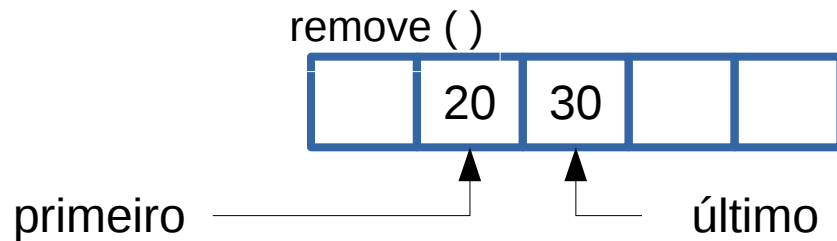
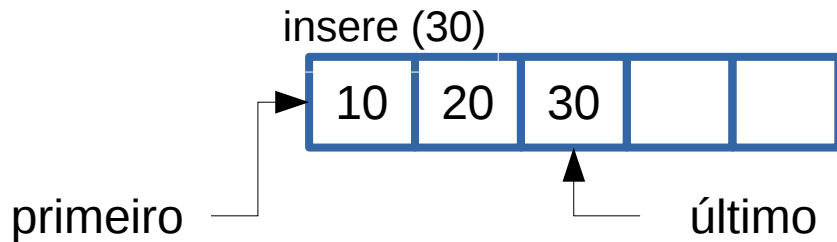
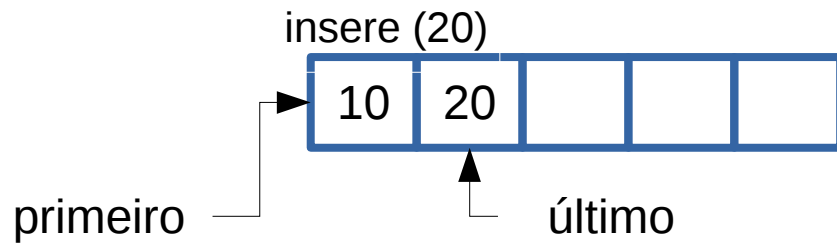
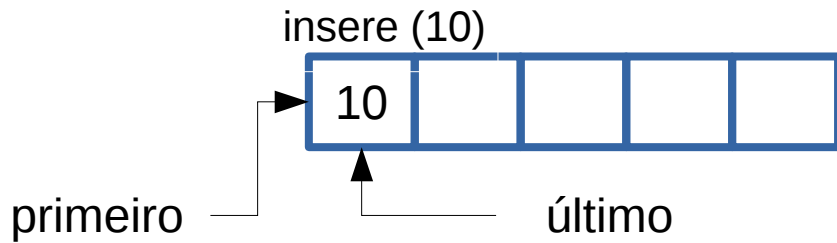
Fila



Fila



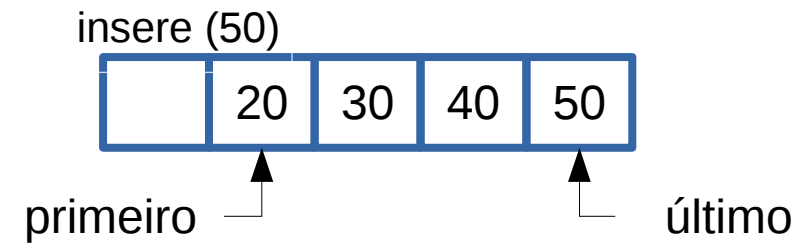
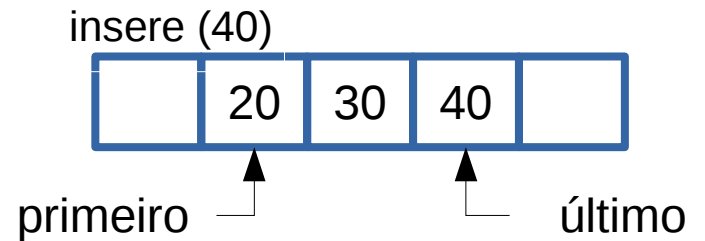
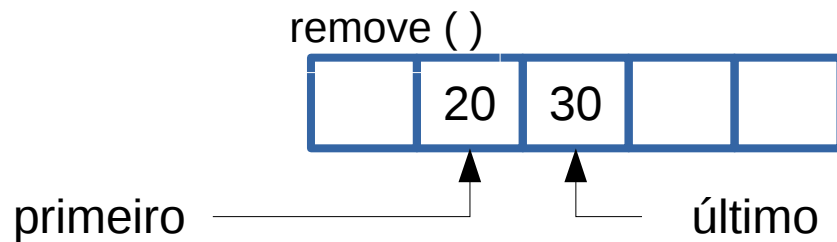
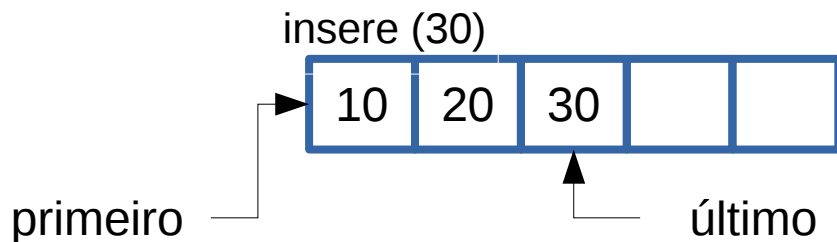
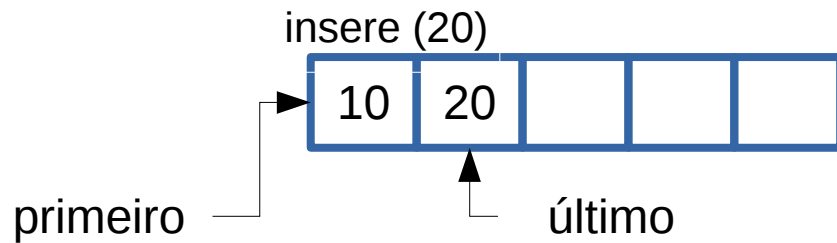
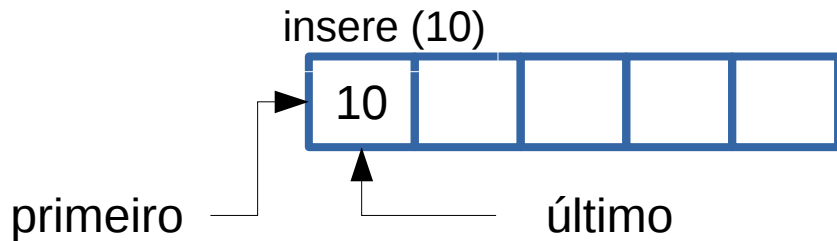
Fila



Como seria possível inserir um novo elemento?



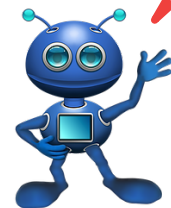
Fila



Como seria possível inserir um novo elemento?



Fila Circular!



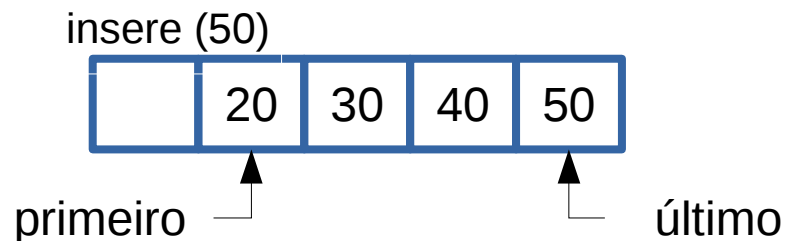
Fila Circular

Fila Circular

- Para evitar problemas de não ser capaz de inserir mais elementos na fila, mesmo quando ela não está cheia, as referências primeiro e último circundam até o início do vetor, resultando numa fila circular.

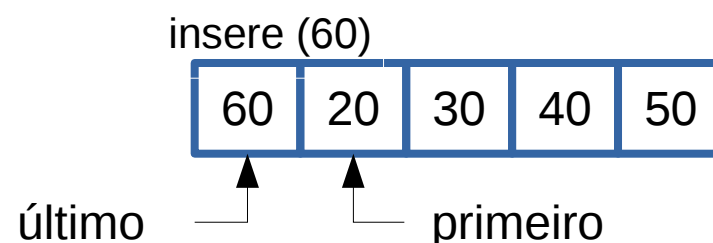
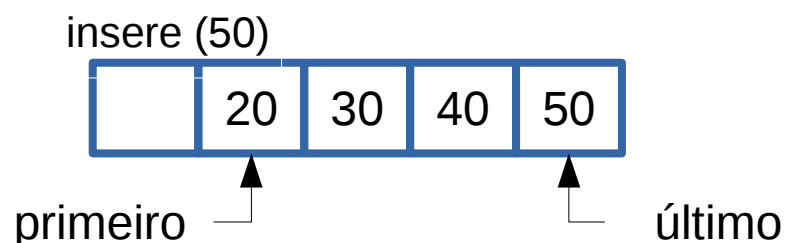
Fila Circular

- Para evitar problemas de não ser capaz de inserir mais elementos na fila, mesmo quando ela não está cheia, as referências primeiro e último circundam até o início do vetor, resultando numa fila circular.



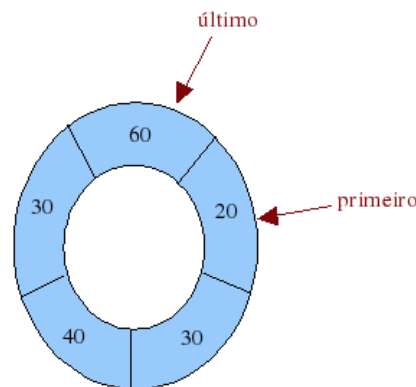
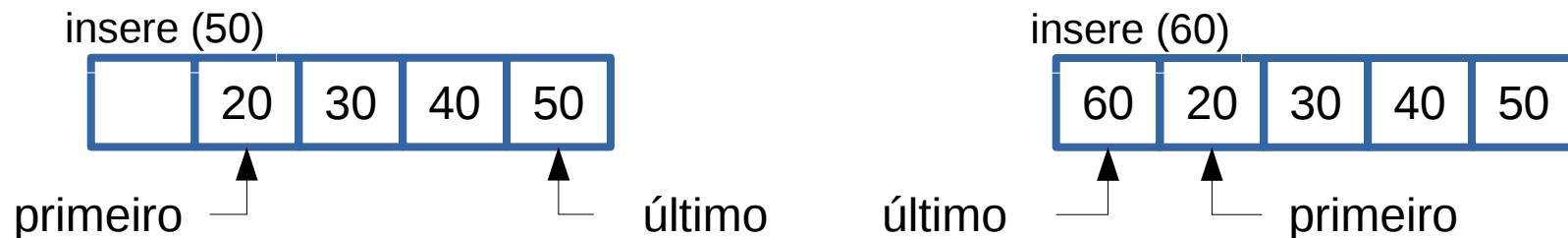
Fila Circular

- Para evitar problemas de não ser capaz de inserir mais elementos na fila, mesmo quando ela não está cheia, as referências primeiro e último circundam até o início do vetor, resultando numa fila circular.

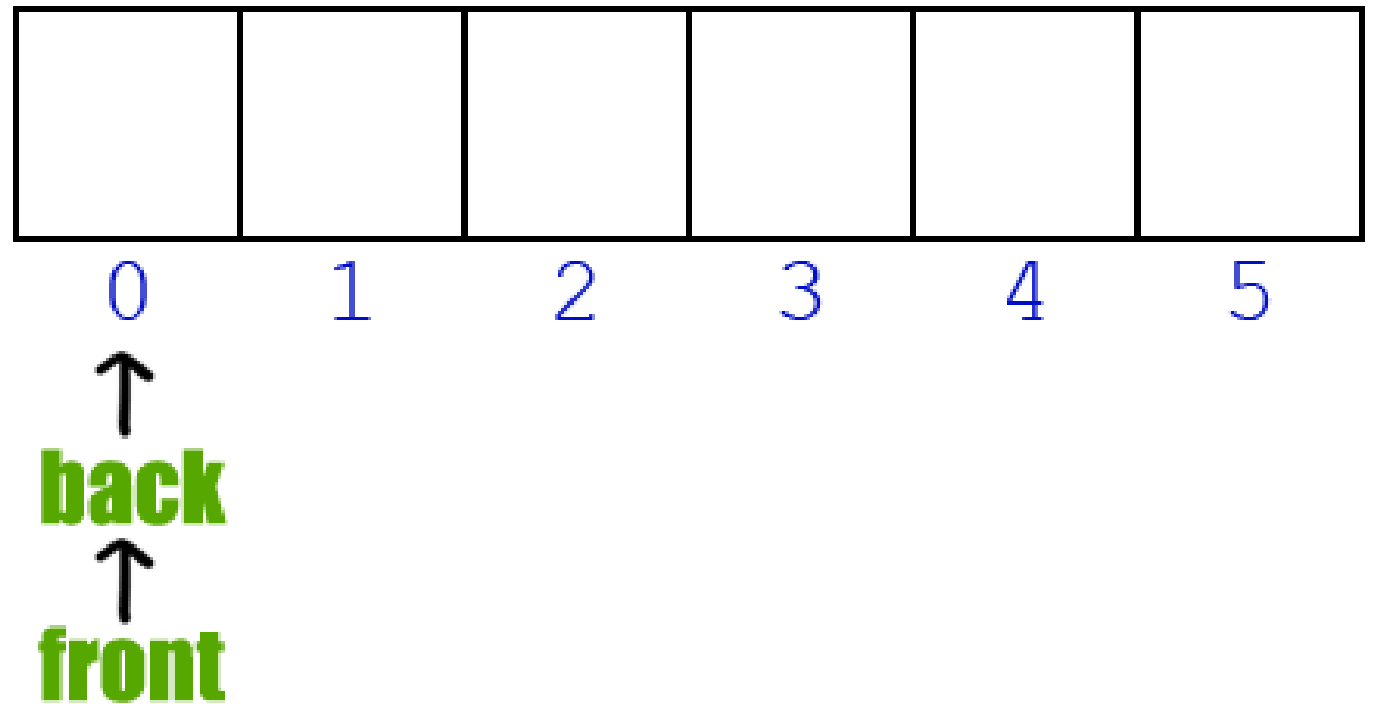


Fila Circular

- Para evitar problemas de não ser capaz de inserir mais elementos na fila, mesmo quando ela não está cheia, as referências primeiro e último circundam até o início do vetor, resultando numa fila circular.



Fila Circular



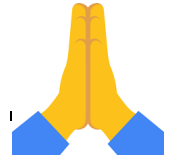
Fila Circular

Fila Circular

- Implementar uma fila circular **não** é difícil. 🙏

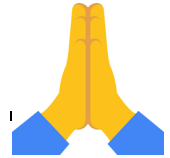
Fila Circular

- Implementar uma fila circular **não** é difícil.
- Basta usar a seguinte aritmética:



Fila Circular

- Implementar uma fila circular **não** é difícil.
- Basta usar a seguinte aritmética:
 - $f = (f + 1) \% N$




Fila Circular

- Implementar uma fila circular **não** é difícil. 🙏
- Basta usar a seguinte aritmética:
 - $f = (f + 1) \% N$
 - Exemplo:
 - Lista de tamanho (N) igual a 10 e índice do primeiro igual a 7.

Fila Circular

- Implementar uma fila circular **não** é difícil. 🙏
- Basta usar a seguinte aritmética:
 - $f = (f + 1) \% N$
 - Exemplo:
 - Lista de tamanho (N) igual a 10 e índice do primeiro igual a 7.
 - Para avançar, bastaria calcular $f = (7 + 1) \% 10$ que é igual a 8. O que seria natural.

Fila Circular

- Implementar uma fila circular **não** é difícil. 
- Basta usar a seguinte aritmética:
 - $f = (f + 1) \% N$
 - Exemplo:
 - Lista de tamanho (N) igual a 10 e índice do primeiro igual a 7.
 - Para avançar, bastaria calcular $f = (7 + 1) \% 10$ que é igual a 8. O que seria natural.
 - Porém, quando o índice for 9, nós podemos calcular $(9 + 1) \% 10$ que levará ao índice 0.

TAD Fila Circular



TAD Fila Circular

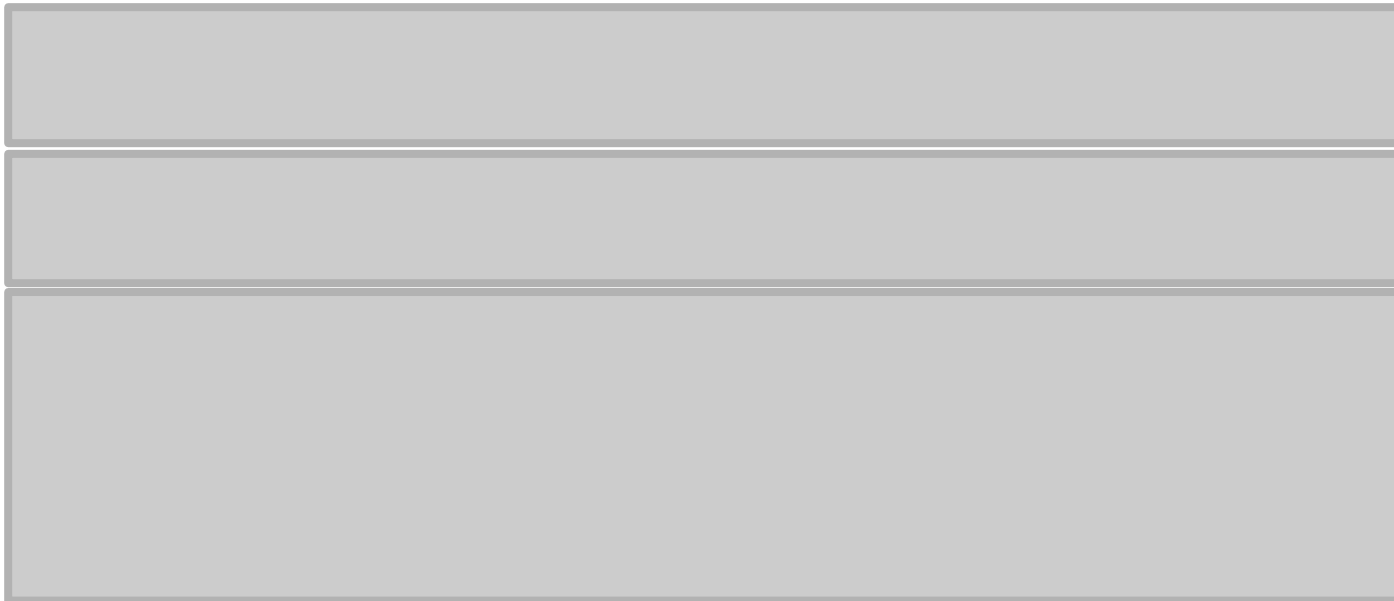
```
class ArrayQueue:
    """FIFO queue implementation using a Python list as underlying storage."""
    DEFAULT_CAPACITY = 10          # moderate capacity for all new queues
```



TAD Fila Circular

```
class ArrayQueue:
    """FIFO queue implementation using a Python list as underlying storage."""
    DEFAULT_CAPACITY = 10          # moderate capacity for all new queues

    def __init__(self):
        """Create an empty queue."""
        self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
        self._size = 0
        self._front = 0
```

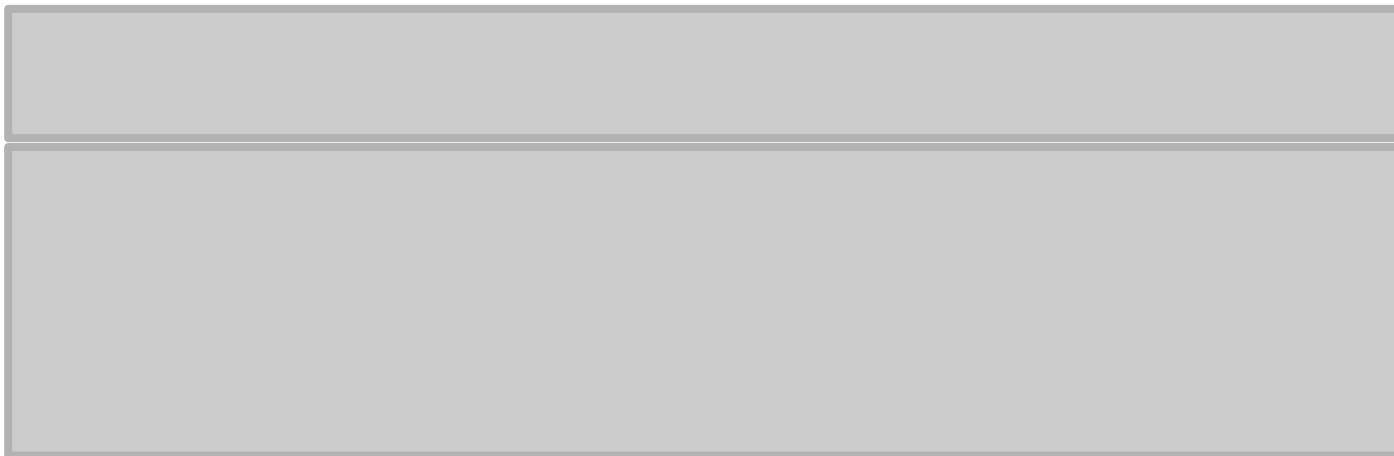


TAD Fila Circular

```
class ArrayQueue:
    """FIFO queue implementation using a Python list as underlying storage."""
    DEFAULT_CAPACITY = 10          # moderate capacity for all new queues

    def __init__(self):
        """Create an empty queue."""
        self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
        self._size = 0
        self._front = 0

    def __len__(self):
        """Return the number of elements in the queue."""
        return self._size
```



TAD Fila Circular

```
class ArrayQueue:
    """FIFO queue implementation using a Python list as underlying storage."""
    DEFAULT_CAPACITY = 10          # moderate capacity for all new queues

    def __init__(self):
        """Create an empty queue."""
        self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
        self._size = 0
        self._front = 0

    def __len__(self):
        """Return the number of elements in the queue."""
        return self._size

    def is_empty(self):
        """Return True if the queue is empty."""
        return self._size == 0
```

TAD Fila Circular

```
class ArrayQueue:
    """FIFO queue implementation using a Python list as underlying storage."""
    DEFAULT_CAPACITY = 10          # moderate capacity for all new queues

    def __init__(self):
        """Create an empty queue."""
        self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
        self._size = 0
        self._front = 0

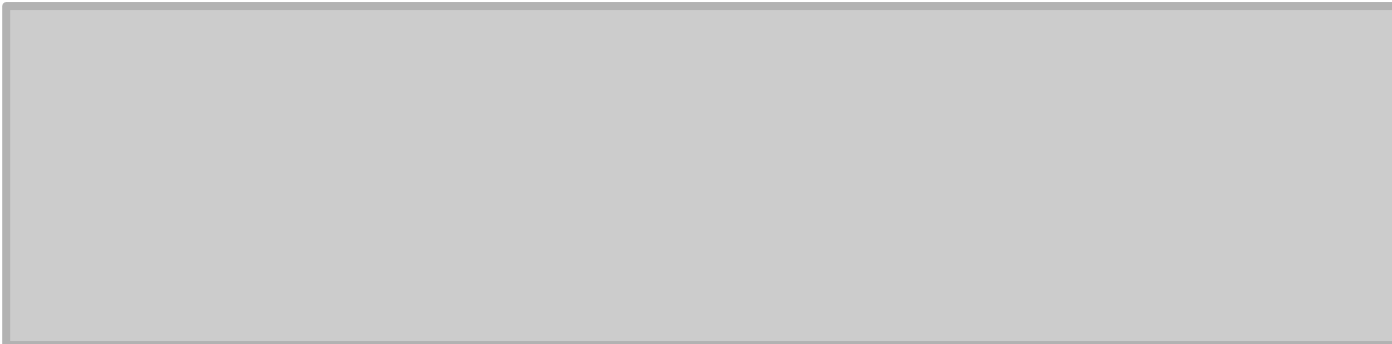
    def __len__(self):
        """Return the number of elements in the queue."""
        return self._size

    def is_empty(self):
        """Return True if the queue is empty."""
        return self._size == 0

    def first(self):
        """Return (but do not remove) the element at the front of the queue.

        Raise Empty exception if the queue is empty.
        """
        if self.is_empty():
            raise Empty('Queue is empty')
        return self._data[self._front]
```

TAD Fila Circular



TAD Fila Circular

```
def dequeue(self):  
    """Remove and return the first element of the queue (i.e., FIFO).  
  
    Raise Empty exception if the queue is empty.  
    """  
    if self.is_empty():  
        raise Empty('Queue is empty')  
    answer = self._data[self._front]  
    self._data[self._front] = None          # help garbage collection  
    self._front = (self._front + 1) % len(self._data)  
    self._size -= 1  
    return answer
```

TAD Fila Circular

```
def dequeue(self):  
    """Remove and return the first element of the queue (i.e., FIFO).  
  
    Raise Empty exception if the queue is empty.  
    """  
    if self.is_empty():  
        raise Empty('Queue is empty')  
    answer = self._data[self._front]  
    self._data[self._front] = None          # help garbage collection  
    self._front = (self._front + 1) % len(self._data)  
    self._size -= 1  
    return answer  
  
def enqueue(self, e):  
    """Add an element to the back of queue."""  
    if self._size == len(self._data):  
        self._resize(2 * len(self._data))    # double the array size  
    avail = (self._front + self._size) % len(self._data)  
    self._data[avail] = e  
    self._size += 1
```

TAD Fila Circular

```
def dequeue(self):  
    """Remove and return the first element of the queue (i.e., FIFO).  
  
    Raise Empty exception if the queue is empty.  
    """  
    if self.is_empty():  
        raise Empty('Queue is empty')  
    answer = self._data[self._front]  
    self._data[self._front] = None # help garbage collection  
    self._front = (self._front + 1) % len(self._data)  
    self._size -= 1  
    return answer  
  
def enqueue(self, e):  
    """Add an element to the back of queue."""  
    if self._size == len(self._data):  
        self._resize(2 * len(self._data)) # double the array size  
    avail = (self._front + self._size) % len(self._data)  
    self._data[avail] = e  
    self._size += 1  
  
def _resize(self, cap): # we assume cap >= len(self)  
    """Resize to a new list of capacity >= len(self)."""  
    old = self._data # keep track of existing list  
    self._data = [None] * cap # allocate list with new capacity  
    walk = self._front  
    for k in range(self._size): # only consider existing elements  
        self._data[k] = old[walk] # intentionally shift indices  
        walk = (1 + walk) % len(old) # use old size as modulus  
    self._front = 0 # front has been realigned
```