Data Science & AI
&
NIC - Param

Python-For Data Science

Stack and Queue

Lecture No.- 03

By- Pankaj Sharma Sir

Recap of Previous Lecture

Topic Stack and Queues Part- 02

Topics to be Covered

Topic Stack and Queues Part- 03

Topic : Stack and Queues

Queue

First In First Out

Insert ⇒ Enqueue

Deletion ⇒ Dequeue

isEmpty()

size()

Front() : —

Dequeue

Enqueue(10)

Enqueue(20)

Enqueue(30)

Enqueue(40)

Dequeue

isEmpty → False

10, 20, 30, 40 →

front() → 30

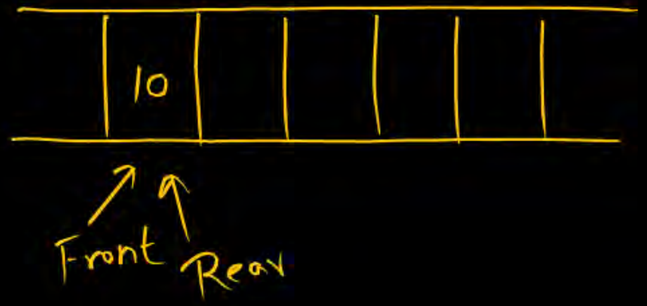
size() → 2

Dequeue() ⇒

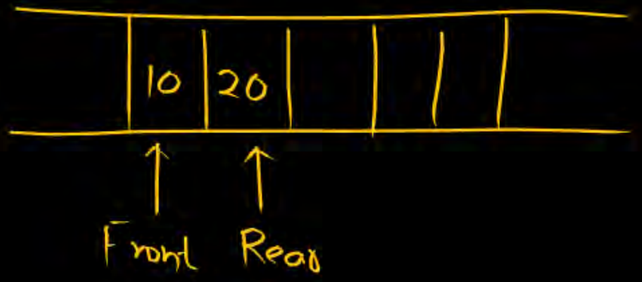
Front : → points to element that can be deleted

Rear : points to most recently added element

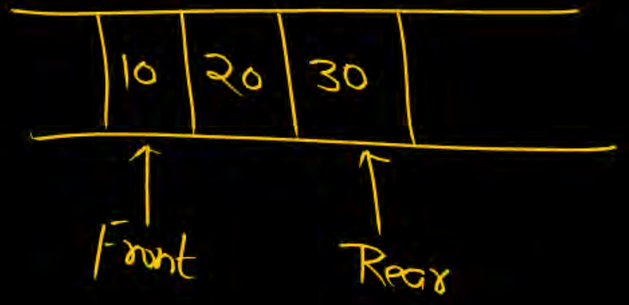
Enqueue(10)

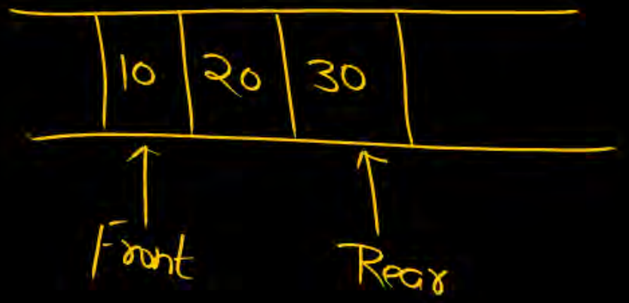


Enqueue(20)



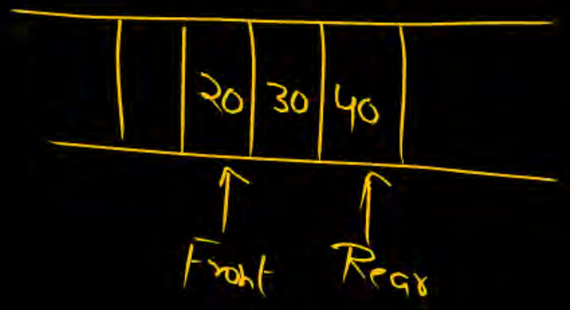
Enqueue(30)





Enqueue(40)

Dequeue( )



list/array

```
l = []
```

dequeue

↓

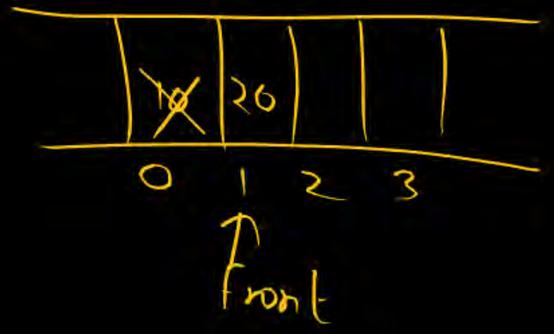
Front

Enqueue ⇒ append

10

Initially

Front = -1




```
class Queue:
    def __init__(self):
        self.__array = []
        self.__front = 0
        self.__count = 0

    def Enqueue(self, ele)
        self.__array.append(ele)
        self.__count = self.__count + 1

    def dequeue(self):
        if self.__count == 0
            return -1
        ele = self.__array[front]
        front = front + 1
        self.__count -= 1
        return ele
```

```
def Front(self):
    if self.__count == 0:
        return -1
    ele = self.__array[front]
    return ele
```


```
def size(self):
    return self.__count
def isEmpty(self):
    return self.size() == 0
```

T.C ⇒ O(1)

O(1)

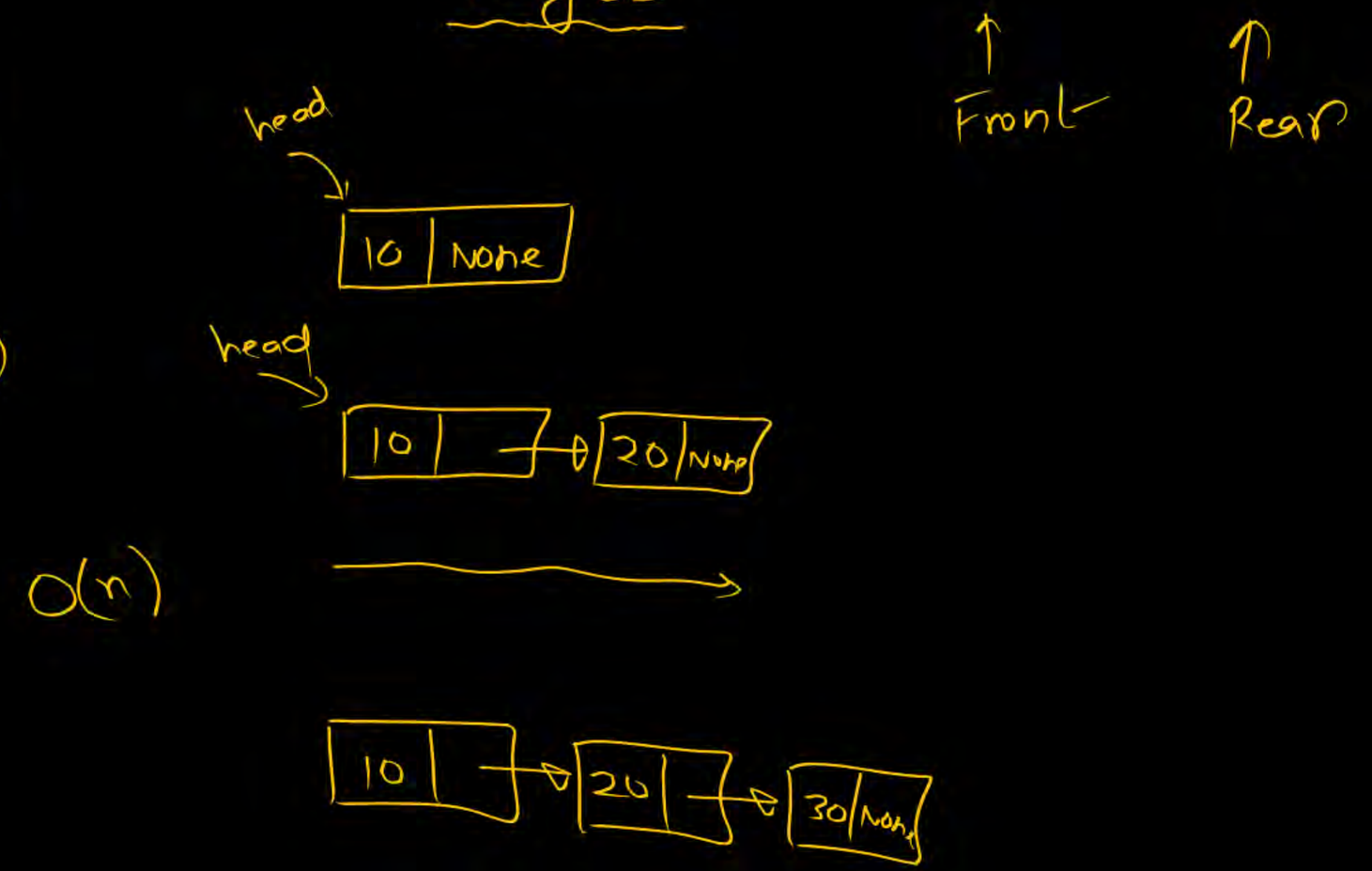
```
class Queue:
    def __init__(self):
        self.__array = []
        self.__front = 0
        self.__count = 0

def Enqueue(self, ele)
    self.__array.append(ele)
    self.__count = self.__count + 1

def dequeue(self):
    if self.__count == 0
        return -1
    ele = self.__array[self.__front]
    self.__front += 1
    self.__count -= 1
    return ele
```

Using LL

Enqueue(10)

Enqueue(20)

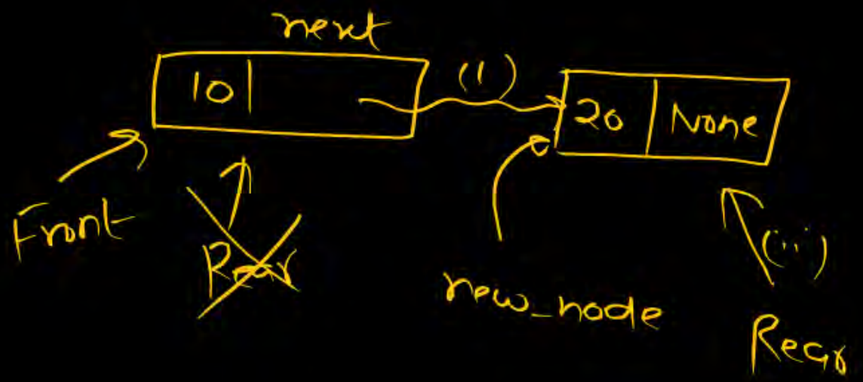
Enqueue(30)

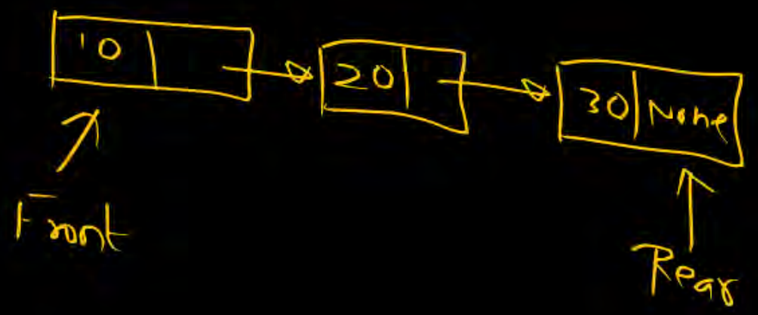


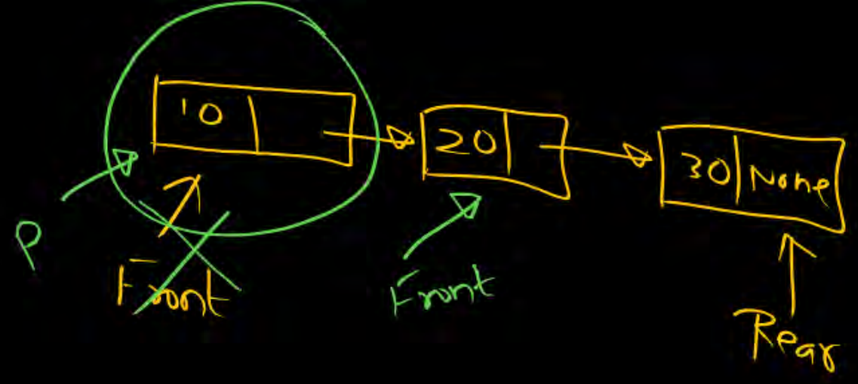
$O(1)$

(i) Rear.next = new_node

(ii) Rear = new_node







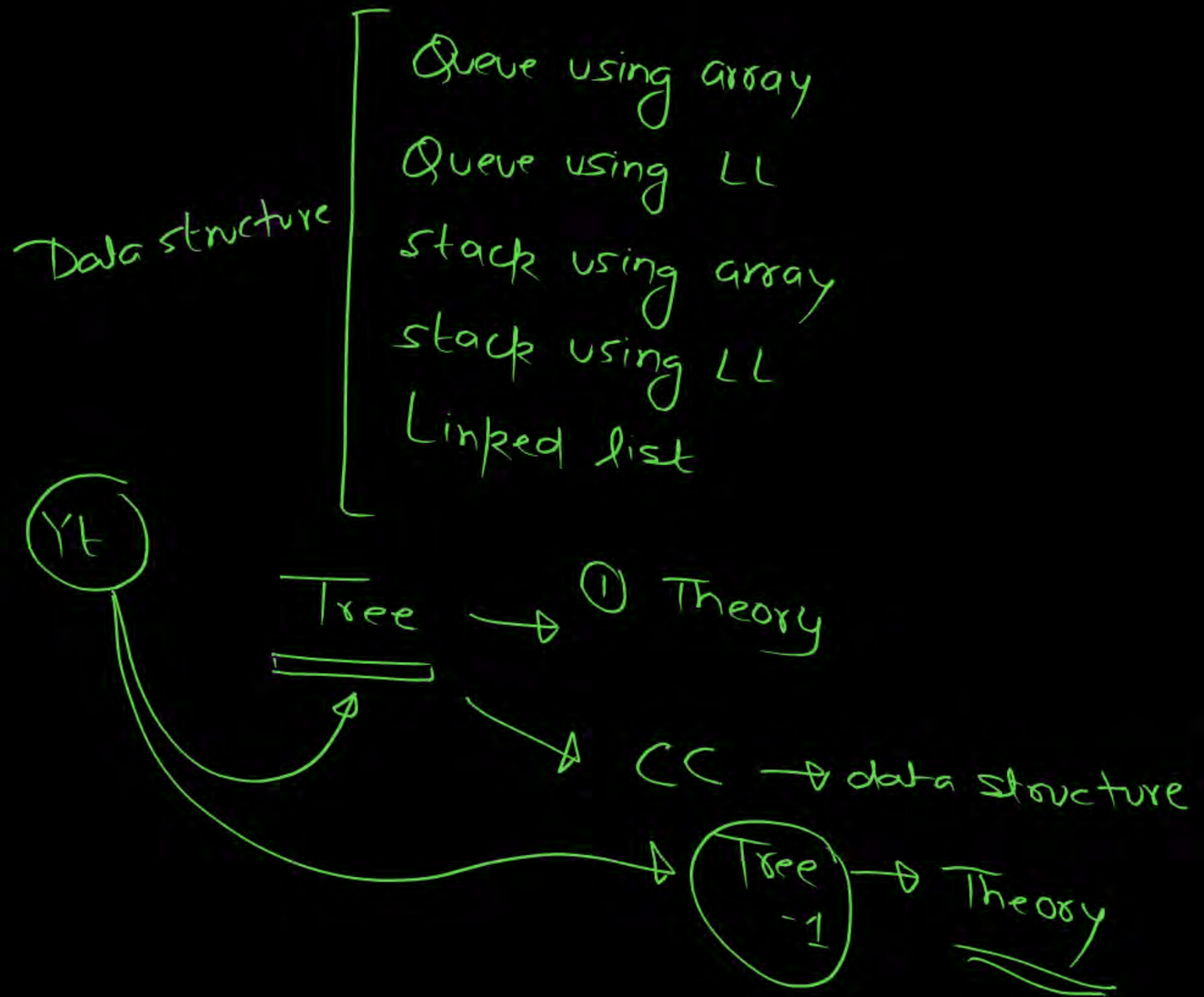
(i) Rear.next = new_node

(ii) Rear = new_node

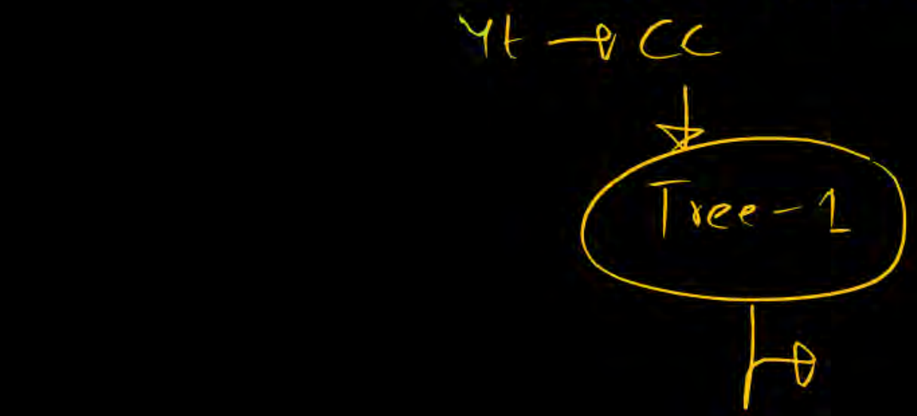
dequeue()

```
if front is None:
    ____
Front = Front.next
```

① Doubt ?


Data structure
Queue using array
Queue using LL
stack using array
stack using LL
Linked list
Yt
Tree
① Theory
CC → data structure
Tree -1
Theory

Data structure



t.me/PWpankajsir

Problem

del

In [1]:
```
class Queue:
    def __init__(self):
        self.__array=[]
        self.__count=0
        self.__front=0
    def enqueue(self,ele):
        self.__array.append(ele)
        self.__count+=1
    def dequeue(self):
        if self.__count==0:
            return -1
        ele=self.__array[self.__front]
        self.__front+=1
        self.__count-=1
        return ele
    def front(self):
        if self.__count==0:
            return -1
        ele=self.__array[self.__front]
        return ele
    def size(self):
        return self.__count
    def isempty(self):
        return self.size()==0
```

In [2]:
```
q=Queue()
```

In [3]:
```
q.enqueue(10)
```

In [4]:
```
q.enqueue(20)
```

In [5]:
```
q.enqueue(30)
```

In [6]:
```
q.dequeue()
```

Out[6]:
```
10
```

In [7]:
```
q.dequeue()
```

Out[7]:
```
20
```

In [8]:
```
q.dequeue()
```

Out[8]:
```
30
```

In [9]:
```
q.dequeue()
```

Out[9]:
```
-1
```

In [10]:
```
q.enqueue(10)
q.enqueue(20)
q.enqueue(30)
```

```
q.enqueue(40)
q.enqueue(50)
```

In [11]:
```
while q.isempty() is False :
    print(q.dequeue())
```

```
10
20
30
40
50
```

In [12]:
```
#inbuilt stack and queue
#stack===>list
import queue
```

In [13]:
```
q=queue.Queue()
```

In [14]:
```
q.put(10)
q.put(20)
q.put(30)
q.put(40)
q.put(50)
```

In [15]:
```
while q.empty() is False :
    print(q.get())
```

```
10
20
30
40
50
```

In [16]:
```
help(queue)
```

```
Help on module queue:

NAME
    queue - A multi-producer, multi-consumer queue.

MODULE REFERENCE
    https://docs.python.org/3.11/library/queue.html

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

CLASSES
    builtins.Exception(builtins.BaseException)
        _queue.Empty
        Full
    builtins.object
        _queue.SimpleQueue
        Queue
            LifoQueue
            PriorityQueue

    class Empty(builtins.Exception)
     |  Exception raised by Queue.get(block=0)/get_nowait().
     |
     |  Method resolution order:
     |      Empty
     |      builtins.Exception
     |      builtins.BaseException
     |      builtins.object
     |
     |  Data descriptors defined here:
     |
     |  __weakref__
     |      list of weak references to the object (if defined)
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from builtins.Exception:
     |
     |  __init__(self, /, *args, **kwargs)
     |      Initialize self.  See help(type(self)) for accurate signature.
     |
     |  ----------------------------------------------------------------------
     |  Static methods inherited from builtins.Exception:
     |
     |  __new__(*args, **kwargs) from builtins.type
     |      Create and return a new object.  See help(type) for accurate signature.
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from builtins.BaseException:
     |
     |  __delattr__(self, name, /)
     |      Implement delattr(self, name).
     |
     |  __getattribute__(self, name, /)
     |      Return getattr(self, name).
     |
     |  __reduce__(...)
```

```
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __setattr__(self, name, value, /)
 |      Implement setattr(self, name, value).
 |
 |  __setstate__(...)
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  add_note(...)
 |      Exception.add_note(note) --
 |      add a note to the exception
 |
 |  with_traceback(...)
 |      Exception.with_traceback(tb) --
 |      set self.__traceback__ to tb and return self.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from builtins.BaseException:
 |
 |  __cause__
 |      exception cause
 |
 |  __context__
 |      exception context
 |
 |  __dict__
 |
 |  __suppress_context__
 |
 |  __traceback__
 |
 |  args

class Full(builtins.Exception)
 |  Exception raised by Queue.put(block=0)/put_nowait().
 |
 |  Method resolution order:
 |      Full
 |      builtins.Exception
 |      builtins.BaseException
 |      builtins.object
 |
 |  Data descriptors defined here:
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from builtins.Exception:
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Static methods inherited from builtins.Exception:
```

```
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from builtins.BaseException:
 |
 |  __delattr__(self, name, /)
 |      Implement delattr(self, name).
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __setattr__(self, name, value, /)
 |      Implement setattr(self, name, value).
 |
 |  __setstate__(...)
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  add_note(...)
 |      Exception.add_note(note) --
 |      add a note to the exception
 |
 |  with_traceback(...)
 |      Exception.with_traceback(tb) --
 |      set self.__traceback__ to tb and return self.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from builtins.BaseException:
 |
 |  __cause__
 |      exception cause
 |
 |  __context__
 |      exception context
 |
 |  __dict__
 |
 |  __suppress_context__
 |
 |  __traceback__
 |
 |  args

class LifoQueue(Queue)
 |  LifoQueue(maxsize=0)
 |
 |  Variant of Queue that retrieves most recently added entries first.
 |
 |  Method resolution order:
 |      LifoQueue
 |      Queue
```

```
|      builtins.object
|
|  Methods inherited from Queue:
|
|  __init__(self, maxsize=0)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  empty(self)
|      Return True if the queue is empty, False otherwise (not reliable!).
|
|      This method is likely to be removed at some point.  Use qsize() == 0
|      as a direct substitute, but be aware that either approach risks a race
|      condition where a queue can grow before the result of empty() or
|      qsize() can be used.
|
|      To create code that needs to wait for all queued tasks to be
|      completed, the preferred technique is to use the join() method.
|
|  full(self)
|      Return True if the queue is full, False otherwise (not reliable!).
|
|      This method is likely to be removed at some point.  Use qsize() >= n
|      as a direct substitute, but be aware that either approach risks a race
|      condition where a queue can shrink before the result of full() or
|      qsize() can be used.
|
|  get(self, block=True, timeout=None)
|      Remove and return an item from the queue.
|
|      If optional args 'block' is true and 'timeout' is None (the default),
|      block if necessary until an item is available. If 'timeout' is
|      a non-negative number, it blocks at most 'timeout' seconds and raises
|      the Empty exception if no item was available within that time.
|      Otherwise ('block' is false), return an item if one is immediately
|      available, else raise the Empty exception ('timeout' is ignored
|      in that case).
|
|  get_nowait(self)
|      Remove and return an item from the queue without blocking.
|
|      Only get an item if one is immediately available. Otherwise
|      raise the Empty exception.
|
|  join(self)
|      Blocks until all items in the Queue have been gotten and processed.
|
|      The count of unfinished tasks goes up whenever an item is added to the
|      queue. The count goes down whenever a consumer thread calls task_done()
|      to indicate the item was retrieved and all work on it is complete.
|
|      When the count of unfinished tasks drops to zero, join() unblocks.
|
|  put(self, item, block=True, timeout=None)
|      Put an item into the queue.
|
|      If optional args 'block' is true and 'timeout' is None (the default),
|      block if necessary until a free slot is available. If 'timeout' is
|      a non-negative number, it blocks at most 'timeout' seconds and raises
|      the Full exception if no free slot was available within that time.
|      Otherwise ('block' is false), put an item on the queue if a free slot
```

```
 |      is immediately available, else raise the Full exception ('timeout'
 |      is ignored in that case).
 |
 |  put_nowait(self, item)
 |      Put an item into the queue without blocking.
 |
 |      Only enqueue the item if a free slot is immediately available.
 |      Otherwise raise the Full exception.
 |
 |  qsize(self)
 |      Return the approximate size of the queue (not reliable!).
 |
 |  task_done(self)
 |      Indicate that a formerly enqueued task is complete.
 |
 |      Used by Queue consumer threads.  For each get() used to fetch a task,
 |      a subsequent call to task_done() tells the queue that the processing
 |      on the task is complete.
 |
 |      If a join() is currently blocking, it will resume when all items
 |      have been processed (meaning that a task_done() call was received
 |      for every item that had been put() into the queue).
 |
 |      Raises a ValueError if called more times than there were items
 |      placed in the queue.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from Queue:
 |
 |  __class_getitem__ = GenericAlias(...) from builtins.type
 |      Represent a PEP 585 generic type
 |
 |      E.g. for t = list[int], t.__origin__ is list and t.__args__ is (int,).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Queue:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class PriorityQueue(Queue)
 |  PriorityQueue(maxsize=0)
 |
 |  Variant of Queue that retrieves open entries in priority order (lowest firs
t).
 |
 |  Entries are typically tuples of the form:  (priority number, data).
 |
 |  Method resolution order:
 |      PriorityQueue
 |      Queue
 |      builtins.object
 |
 |  Methods inherited from Queue:
 |
 |  __init__(self, maxsize=0)
 |      Initialize self.  See help(type(self)) for accurate signature.
```

```
|
|  empty(self)
|      Return True if the queue is empty, False otherwise (not reliable!).
|
|      This method is likely to be removed at some point.  Use qsize() == 0
|      as a direct substitute, but be aware that either approach risks a race
|      condition where a queue can grow before the result of empty() or
|      qsize() can be used.
|
|      To create code that needs to wait for all queued tasks to be
|      completed, the preferred technique is to use the join() method.
|
|  full(self)
|      Return True if the queue is full, False otherwise (not reliable!).
|
|      This method is likely to be removed at some point.  Use qsize() >= n
|      as a direct substitute, but be aware that either approach risks a race
|      condition where a queue can shrink before the result of full() or
|      qsize() can be used.
|
|  get(self, block=True, timeout=None)
|      Remove and return an item from the queue.
|
|      If optional args 'block' is true and 'timeout' is None (the default),
|      block if necessary until an item is available. If 'timeout' is
|      a non-negative number, it blocks at most 'timeout' seconds and raises
|      the Empty exception if no item was available within that time.
|      Otherwise ('block' is false), return an item if one is immediately
|      available, else raise the Empty exception ('timeout' is ignored
|      in that case).
|
|  get_nowait(self)
|      Remove and return an item from the queue without blocking.
|
|      Only get an item if one is immediately available. Otherwise
|      raise the Empty exception.
|
|  join(self)
|      Blocks until all items in the Queue have been gotten and processed.
|
|      The count of unfinished tasks goes up whenever an item is added to the
|      queue. The count goes down whenever a consumer thread calls task_done()
|      to indicate the item was retrieved and all work on it is complete.
|
|      When the count of unfinished tasks drops to zero, join() unblocks.
|
|  put(self, item, block=True, timeout=None)
|      Put an item into the queue.
|
|      If optional args 'block' is true and 'timeout' is None (the default),
|      block if necessary until a free slot is available. If 'timeout' is
|      a non-negative number, it blocks at most 'timeout' seconds and raises
|      the Full exception if no free slot was available within that time.
|      Otherwise ('block' is false), put an item on the queue if a free slot
|      is immediately available, else raise the Full exception ('timeout'
|      is ignored in that case).
|
|  put_nowait(self, item)
|      Put an item into the queue without blocking.
|
```

```
 |      Only enqueue the item if a free slot is immediately available.
 |      Otherwise raise the Full exception.
 |
 |  qsize(self)
 |      Return the approximate size of the queue (not reliable!).
 |
 |  task_done(self)
 |      Indicate that a formerly enqueued task is complete.
 |
 |      Used by Queue consumer threads.  For each get() used to fetch a task,
 |      a subsequent call to task_done() tells the queue that the processing
 |      on the task is complete.
 |
 |      If a join() is currently blocking, it will resume when all items
 |      have been processed (meaning that a task_done() call was received
 |      for every item that had been put() into the queue).
 |
 |      Raises a ValueError if called more times than there were items
 |      placed in the queue.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from Queue:
 |
 |  __class_getitem__ = GenericAlias(...) from builtins.type
 |      Represent a PEP 585 generic type
 |
 |      E.g. for t = list[int], t.__origin__ is list and t.__args__ is (int,).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Queue:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class Queue(builtins.object)
 |  Queue(maxsize=0)
 |
 |  Create a queue object with a given maximum size.
 |
 |  If maxsize is <= 0, the queue size is infinite.
 |
 |  Methods defined here:
 |
 |  __init__(self, maxsize=0)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  empty(self)
 |      Return True if the queue is empty, False otherwise (not reliable!).
 |
 |      This method is likely to be removed at some point.  Use qsize() == 0
 |      as a direct substitute, but be aware that either approach risks a race
 |      condition where a queue can grow before the result of empty() or
 |      qsize() can be used.
 |
 |      To create code that needs to wait for all queued tasks to be
 |      completed, the preferred technique is to use the join() method.
 |
```

```
|  full(self)
|      Return True if the queue is full, False otherwise (not reliable!).
|
|      This method is likely to be removed at some point.  Use qsize() >= n
|      as a direct substitute, but be aware that either approach risks a race
|      condition where a queue can shrink before the result of full() or
|      qsize() can be used.
|
|  get(self, block=True, timeout=None)
|      Remove and return an item from the queue.
|
|      If optional args 'block' is true and 'timeout' is None (the default),
|      block if necessary until an item is available. If 'timeout' is
|      a non-negative number, it blocks at most 'timeout' seconds and raises
|      the Empty exception if no item was available within that time.
|      Otherwise ('block' is false), return an item if one is immediately
|      available, else raise the Empty exception ('timeout' is ignored
|      in that case).
|
|  get_nowait(self)
|      Remove and return an item from the queue without blocking.
|
|      Only get an item if one is immediately available. Otherwise
|      raise the Empty exception.
|
|  join(self)
|      Blocks until all items in the Queue have been gotten and processed.
|
|      The count of unfinished tasks goes up whenever an item is added to the
|      queue. The count goes down whenever a consumer thread calls task_done()
|      to indicate the item was retrieved and all work on it is complete.
|
|      When the count of unfinished tasks drops to zero, join() unblocks.
|
|  put(self, item, block=True, timeout=None)
|      Put an item into the queue.
|
|      If optional args 'block' is true and 'timeout' is None (the default),
|      block if necessary until a free slot is available. If 'timeout' is
|      a non-negative number, it blocks at most 'timeout' seconds and raises
|      the Full exception if no free slot was available within that time.
|      Otherwise ('block' is false), put an item on the queue if a free slot
|      is immediately available, else raise the Full exception ('timeout'
|      is ignored in that case).
|
|  put_nowait(self, item)
|      Put an item into the queue without blocking.
|
|      Only enqueue the item if a free slot is immediately available.
|      Otherwise raise the Full exception.
|
|  qsize(self)
|      Return the approximate size of the queue (not reliable!).
|
|  task_done(self)
|      Indicate that a formerly enqueued task is complete.
|
|      Used by Queue consumer threads.  For each get() used to fetch a task,
|      a subsequent call to task_done() tells the queue that the processing
|      on the task is complete.
```

```
 |
 |      If a join() is currently blocking, it will resume when all items
 |      have been processed (meaning that a task_done() call was received
 |      for every item that had been put() into the queue).
 |
 |      Raises ValueError if called more times than there were items
 |      placed in the queue.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __class_getitem__ = GenericAlias(...) from builtins.type
 |      Represent a PEP 585 generic type
 |
 |      E.g. for t = list[int], t.__origin__ is list and t.__args__ is (int,).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class SimpleQueue(builtins.object)
 |  Simple, unbounded, reentrant FIFO queue.
 |
 |  Methods defined here:
 |
 |  empty(self, /)
 |      Return True if the queue is empty, False otherwise (not reliable!).
 |
 |  get(self, /, block=True, timeout=None)
 |      Remove and return an item from the queue.
 |
 |      If optional args 'block' is true and 'timeout' is None (the default),
 |      block if necessary until an item is available. If 'timeout' is
 |      a non-negative number, it blocks at most 'timeout' seconds and raises
 |      the Empty exception if no item was available within that time.
 |      Otherwise ('block' is false), return an item if one is immediately
 |      available, else raise the Empty exception ('timeout' is ignored
 |      in that case).
 |
 |  get_nowait(self, /)
 |      Remove and return an item from the queue without blocking.
 |
 |      Only get an item if one is immediately available. Otherwise
 |      raise the Empty exception.
 |
 |  put(self, /, item, block=True, timeout=None)
 |      Put the item on the queue.
 |
 |      The optional 'block' and 'timeout' arguments are ignored, as this method
 |      never blocks.  They are provided for compatibility with the Queue class.
 |
 |  put_nowait(self, /, item)
 |      Put an item into the queue without blocking.
 |
 |      This is exactly equivalent to `put(item)` and is only provided
```

```
|      for compatibility with the Queue class.
|
|  qsize(self, /)
|      Return the approximate size of the queue (not reliable!).
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __class_getitem__(...) from builtins.type
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.

DATA
    __all__ = ['Empty', 'Full', 'Queue', 'PriorityQueue', 'LifoQueue', 'Si...

FILE
    c:\programdata\anaconda3\lib\queue.py
```

In [17]:
```
s=queue.LifoQueue()
```

In [ ]:
```

```

THANK - YOU