Week-10

The First In First Out (Queue) Data structure – Example: Media player list, keyboard buffer queue, printer queue etc. (to be used to explain concept of FIFO). Implementing the Queue and its operations using Python List. Priority Queues, Implementation.

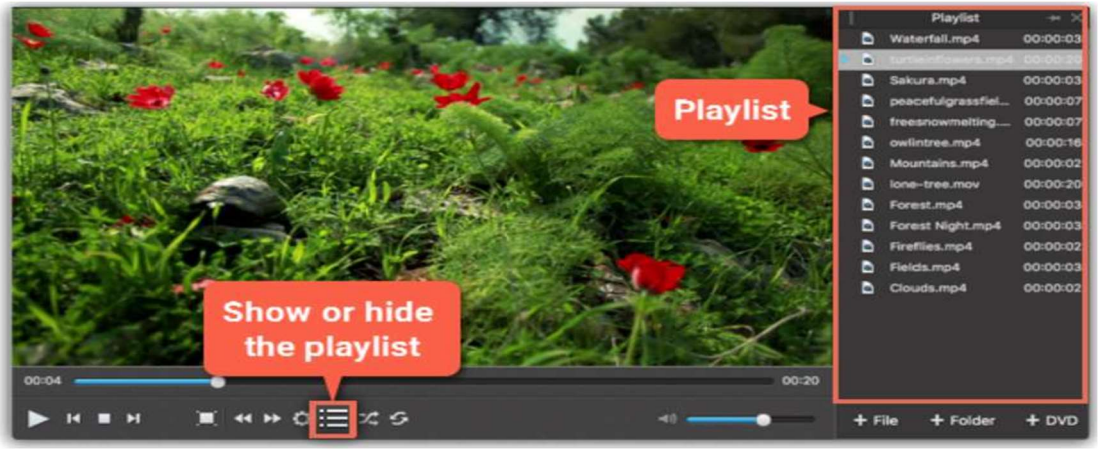
The First In First Out (Queue) Data structure:

- Queue: A Queue is special type of data structure where elements are inserted from one end called **rear** and elements are deleted from another end called **front**.
- Queue is a FIFO (First in First out) data structure.
- A queue can be illustrated with line of people waiting for the bus at a bus station, list of calls put on hold to be answered by a telephone operator is a queue, and a list of waiting jobs to be processed by a computer is a queue.
- The pictorial representation of the queue is given below.

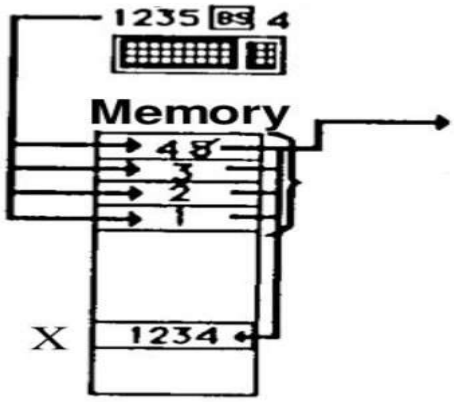
A queue data structure is well suited for problems in computer science that require data to be processed in the order in which it was received.

Examples:

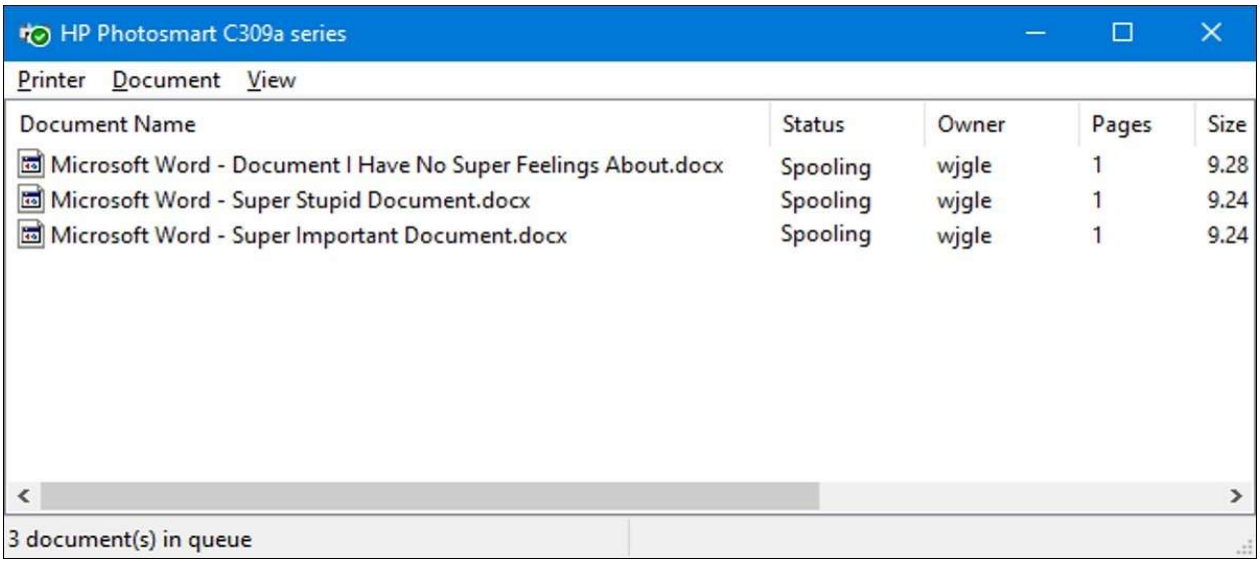
- ✓ **Media player list**: Music player software allows users the chance to add songs to a playlist. Upon hitting the play button, all the songs in the main playlist are played one after the other. The sequential playing of the songs can be implemented with queues because the first song to be queued is the first song that is played.



- ✓ **Keyboard buffer queue**: When users type in the characters using keyboard they stored in a queue in a keyboard buffer and given as input to the application.



- ✓ **Printer queue**: A job to be printed is stored on disk in a queue. The printer prints them in the order it receives - first come first print. This allows users to work on other tasks after sending the printing jobs to the queue.



Queue ADT:

- ➢ A queue is a data structure which consists of linear collection of items in which access is restricted to a first-in first-out basis.
- ➢ New items are inserted at the back and existing items are removed from the front.
- ➢ The items are maintained in the order in which they are added to the structure.
  - ✓ Queue(): Creates a new empty queue, which is a queue containing no items.
  - ✓ enqueue(): Adds the given item to the back of the queue.
  - ✓ dequeue(): Removes the front item from the queue. An item cannot be dequeued from an empty queue.
  - ✓ isEmpty(): Returns a Boolean value indicating whether the queue is empty.
  - ✓ length (): Returns the number of items currently in the queue.

Implementing the Queue and its operations using Python List:

Let us implement a very simple queue using Python's list class. The operations that must be performed on the queue are encapsulated in the Queue class:

```
class Queue:
   def__init_(self):
      self.items = []
```

In the initialization method __init_, the instance variable "**items"** is set to [], which means the queue is empty when created.

Enqueue operation:

- The enqueue operation is used to add items to the queue.
- In queue, elements are always inserted from **rear** end.
- The enqueue operation or method uses the **append** method of the list class to insert items (or data) at the rear of the list:
- The python code of enqueue operation is given below.

```
def enqueue(self,item):
        self.items.append(item)
```

The enqueue operation can be visualized as shown in the diagram below.

```
Q = Queue()
Q.enqueue(28)
Q.enqueue(19)
Q.enqueue(45)
Q.enqueue(13)
Q.enqueue(7)
```

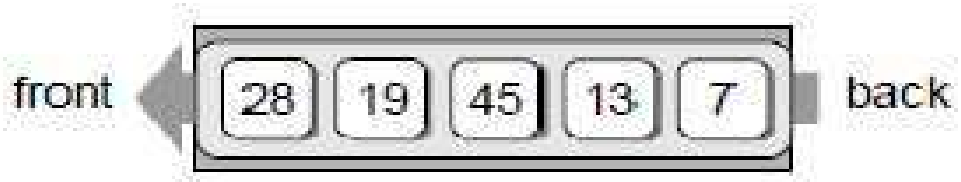
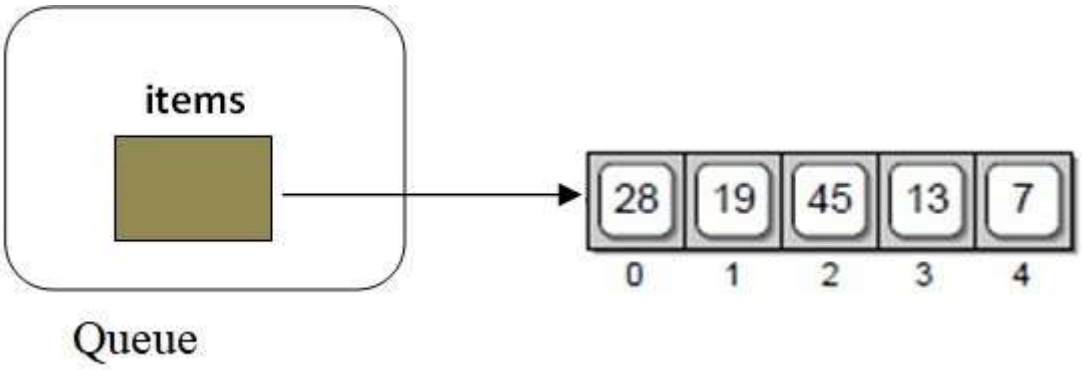


Fig. Abstract view of the queue after the above enqueue operations.

We first create the Queue object and enqueue 5 values into the queue in the order as shown in queue.



The above figure shows the abstract view of the object or instance of Queue ADT using python list.

Dequeue operation:

- The dequeue operation is used to remove items from the queue.
- In queue items are always deleted from the **front** end.
- The Python list class has a method called **pop(index)** which is used to remove item from the queue.
- We will use **pop(0)** to delete the front item from the list.
- The item in the list is popped and saved in the data variable.
- The python code of dequeue operation is given below.

```
def dequeue(self):
    if self.isEmpty():
        print("Queue is Empty cannot delete")
    else:
        item=self.items.pop(0)
        print("Deleted Item is:",item)
```

isEmpty() operation :

- This isEmpty() is a utility function which returns Boolean value **true** if list containing queue elements is empty and **false** if list has some elements.
- This function is used by other functions for example: dequeue operation will first make sure whether the queue is not empty before removing items from the queue.
- The python code of this operation is given below

```
def isEmpty(self):
    return len(self.items) == 0
```

The below diagram shows the enqueue and dequeue operations on the queue.

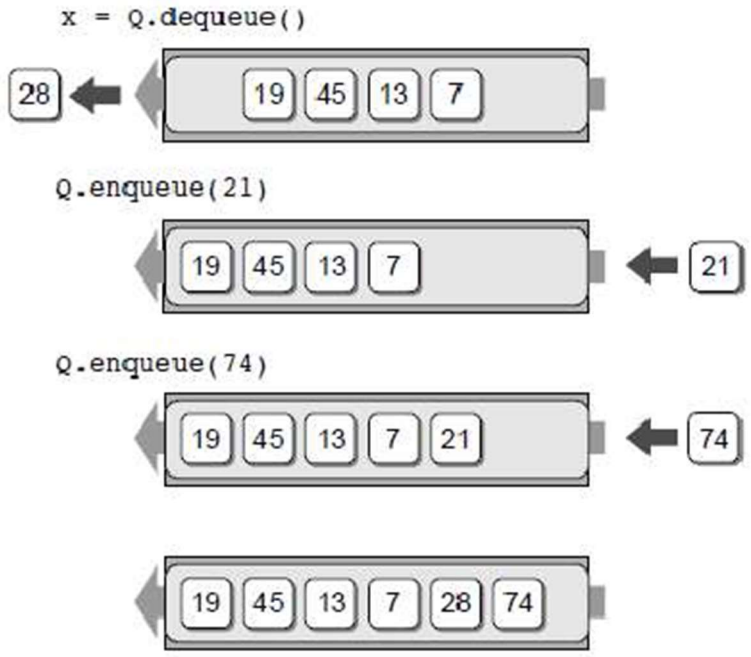


Fig. Abstract view of the queue after performing the additional operations.

Complete Implementation of Queue data structure:

```
class Queue:
    def__init_(self):
        self.items = []
    def enqueue(self,item):
        self.items.append(item)
    def dequeue(self):
        if self.isEmpty():
            print("Queue is Empty cannot delete")
        else:
            item=self.items.pop(0)
            print("Deleted Item is:",item)
    def display(self):
        if self.isEmpty():
            print("Queue is Empty")
        else:
            print(self.items)
    def isEmpty(self):
        return len(self.items) == 0
    def length(self):
        return len(self.items)
q = Queue()
while True:
    print("1:Enqueue 2:Dequeue 3:Display 4:Length 5:Exit")
    choice = int(input("Enter your choice:"))
    if choice==1:
        item=input("Enter the element:")
        q.enqueue(item)
    elif choice==2:
        q.dequeue()
    elif choice==3:
        q.display()
    elif choice==4:
        n = q.length()
        print("Length of the queue is ",n)
    elif choice==5:
        break
```

Priority Queues:

- A priority queue is a queue in which each item is assigned a priority and items with a higher priority are dequeued (removed) before those with a lower priority, irrespective of when they were added. However all the items with the same priority still obey FIFO principle.
- Integer values are used for the priorities with a smaller integer value having a higher priority.

- There are two basic types of priority queues:
    1. Bounded priority queue.
    2. Unbounded priority queue.
- **A bounded priority queue** restricts the priorities to the integer values between zero and a predefined upper limit **N** whereas an unbounded priority queue places no limits on the range of priorities.

The priority Queue ADT:

- ✓ PriorityQueue(): Creates a new empty unbounded priority queue.
- ✓ isEmpty(): Returns a boolean value indicating whether the queue is empty.
- ✓ length (): Returns the number of items currently in the queue.
- ✓ enqueue(item, priority): Adds the given item to the queue by inserting it in the proper position based on the given priority. The priority value must be within the legal range when using a bounded priority queue.
- ✓ Dequeue(): Removes and returns the item from the queue, which is the item with the highest priority. If two items have the same priority, then those items are removed in a FIFO order. An item cannot be deleted from an empty queue.

A note on priority queue item:

- When implementing the priority queue, the items cannot simply be added directly to the list, but instead we must have a way to associate a priority with each item.
- This can be done with a simple storage class containing two fields: one for the priority and one for the queue item as shown below.

```
class PriorityQueueEntry:
    def__init_(self,value,priority):
        self.v = value
        self.p = priority
```

How to organize items in the priority queue?

We can consider t
wo approaches, both of which satisfy the requirements of the priority queue:

1. Append new items to the end of the list.

- When a new item is enqueued, simply append a new instance of the storage class (containing the item and its priority) to the end of the list.
- When an item is dequeued, search the list for the item with the highest priority and remove it from the list.
- If more than one item has the same priority, the first one encountered during the search will be the first to be dequeued.

2. Keep the items sorted within the list based on their priority.

- When a new item is enqueued, find its proper position within the list based on its priority and insert an instance of the storage class at that point.
- If we order the items in the list from lowest priority at the front to highest at the end, then the dequeue operation simply requires the removal of the last item in the list.
- To maintain the proper ordering of items with equal priority, the enqueue operation must ensure newer items are inserted closer to the front of the list than the other items with the same priority.

- We have used first approach in our implementation below.

Enqueue Operation:

- We are using the first approach for enqueue operation.
- In this approach, we simply add the items into the priority queue using the **append** function on the items list.
- We first create a **PriorityQueueEntry** object and assign it with value and the priority fields and then **append** the oject to list.
- The python code of enqueue operation is given below

```
def enqueue(self,value,priority):
    item  = PriorityQueueEntry(value,priority)
    self.items.append(item)
```

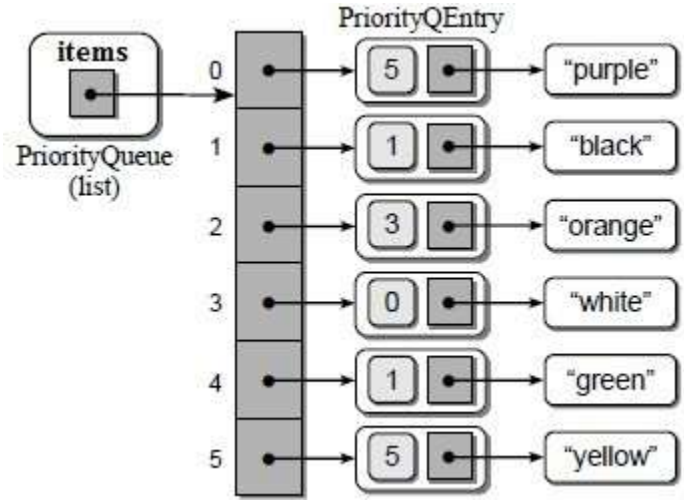


Fig. Abstract view of the priority queue object.

The above diagram shows the visualization of the priority queue after enqueue of 5 queue items. You can see that the items are simply appended as they are inserted.

Dequeue operation:

- The dequeue operation will remove the item with highest priority irrespective of when it was inserted into the queue.
- We will search for the item with highest priority in the queue and find its index and then use pop method provide with index of the highest priority item to delete the item from the queue.
- In our implementation we assume item at first index as highest priority at first and then loop through all items to find index of highest priority.
- **Note**: The highest priority item is one with lowest integer. In the above diagram white is highest priority item since it has 0 as priority value.
- The python code of dequeue operation is given below

```
def dequeue(self):
    if self.isEmpty():
       print("Queue is empty cannot delete")
    else:
```

```
        highest = self.items[0].p
        index = 0
        for i in range(0,self.length()):
            if highest > self.items[i].p:
                highest = self.items[i].p
                index = i
        del_item = self.items.pop(index)
        print("deleted item is ",del_item.v)
```

isEmpty() operation:

- This isEmpty() is a utility function which returns Boolean value **true** if list containing queue elements is empty and false if list has some elements.
- This function is used by other functions for example: dequeue operation will first make sure whether the queue is not empty before removing items from the queue.
- The python code of this operation is given below

```
def isEmpty(self):
    return len(self.items) == 0
```

Implementation of priority queue data structure:

```
class PriorityQueueEntry:
    def___init_(self,value,priority):
        self.v = value
        self.p = priority
class PriorityQueue:
    def___init_(self):
        self.items = []
    def isEmpty(self):
        return len(self.items) == 0
    def length(self):
        return len(self.items)
    def enqueue(self,v


    alue,priority):
        item  = PriorityQueueEntry(value,priority)
        self.items.append(item)
```

```
    def dequeue(self):
        if self.isEmpty():
            print("Queue is empty cannot delete")
        else:
            highest = self.items[0].p
            index = 0
            for i in range(0,self.length()):
                if highest > self.items[i].p:
                    highest = self.items[i].p
                    index = i
            del_item = self.items.pop(index)
            print("deleted item is ",del_item.v)




    def display(self):
        if self.isEmpty():
            print("Queue is empty")
        else:
            for x in range(0,self.length()):
                print(self.items[x].v,":",self.items[x].p)



pq = PriorityQueue()
while(True):
    print("1:Enqueue 2:Dequeue 3:Display 4:Length 5:Exit")
    choice = int(input("Enter your choice:"))
    if choice == 1:
        value = input("enter the item to insert:")
        priority = int(input("Enter the priority:"))
        pq.enqueue(value,priority)
    if choice == 2:
        pq.dequeue()
    if choice == 3:
        pq.display()
    if choice == 4:
        n = pq.length()
        print("length of queue is:",n)
    if choice == 5:
```

break