# Stack in Python

# A stack is a linear data structure that stores items in a Last-In/First-Out (LIFO) or First-In/Last-Out (FILO) manner.

# Stack in python

Using List as Stack

# Python code to demonstrate Implementing

# stack using list

stack = ["Amar", "Akbar", "Anthony"]

stack.append("Ram")

stack.append("Iqbal")

print(stack)

print(stack.pop())

print(stack)

print(stack.pop())

print(stack)

OUTPUT:

['Amar', 'Akbar', 'Anthony', 'Ram', 'Iqbal']

Iqbal

['Amar', 'Akbar', 'Anthony', 'Ram']

Ram

['Amar', 'Akbar', 'Anthony']

**Implementation of stack using Class/OOPS:**

class Stack:

    def \_\_init\_\_(self):

        self.items=[]

    def push(self,item):

        self.items.append(item)

    def pop(self):

        return self.items.pop()

    def getstack(self):

        return self.items

s=Stack()

s.push("A")

s.push("B")

s.push("C")

s.push("D")

print(s.getstack())

s.pop()

print(s.getstack())

OUTPUT:

['A', 'B', 'C', 'D']

['A', 'B', 'C']

class Stack:

    def \_\_init\_\_(self,size):

        self.items=[]

        self.size=size

    def push(self,item):

        if len(self.items)==self.size:

            print("stack is full")

        else:

            self.items.append(item)

    def pop(self):

        if self.items==[]:

            print("stack is empty")

        else:

            return self.items.pop()

    def getstack(self):

        return self.items

s=Stack(5)

s.push("A")

s.push("B")

s.push("C")

s.push("D")

s.push("E")

s.push("F")

print(s.getstack())

print(s.pop())

print(s.pop())

print(s.pop())

print(s.pop())

print(s.pop())

print(s.pop())

print(s.getstack())

OUTPUT:

stack is full

['A', 'B', 'C', 'D', 'E']

E

D

C

B

A

stack is empty

None

[]

1. **Stack using queue**

This module queue also provides LIFO Queue which technically works as a Stack.

import queue

L = queue.LifoQueue(maxsize=6)

# qsize() give the maxsize of the Queue

print(L.qsize())

# Data Inserted as 5->9->1->7, same as Queue

L.put(5)

L.put(9)

L.put(1)

L.put(7)

L.put(9)

L.put(10)

print("Full: ", L.full())

print("Size: ", L.qsize())

# Data will be accessed in the reverse order Reverse of that of Queue

print(L.get())

print(L.get())

print(L.get())

print(L.get())

print(L.get())

print("Empty: ", L.empty())

OUTPUT:

0

Full: True

Size: 6

10

9

7

1

9

Empty: False

# Queue in Python



* **Enqueue:** Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition – Time Complexity : O(1)
* **Dequeue:** Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition – Time Complexity : O(1)
* **Front:** Get the front item from queue – Time Complexity : O(1)
* **Rear:** Get the last item from queue – Time Complexity : O(1)

**Queue in Python can be implemented by the following ways:**

* list
* collections.deque
* queue.Queue

#### **Implementation using list**

# Initializing a queue

queue = []

# Adding elements to the queue

queue.append('a')

queue.append('b')

queue.append('c')

print("Initial queue")

print(queue)

# Removing elements from the queue

print("\nElements dequeued from queue")

print(queue.pop(0))

print(queue.pop(0))

print(queue.pop(0))

print("\nQueue after removing elements")

print(queue)

#### **Implementation using collections.deque**

from collections import deque

# Initializing a queue

q = deque()

# Adding elements to a queue

q.append('a')

q.append('b')

q.append('c')

print("Initial queue")

print(q)

# Removing elements from a queue

print("\nElements dequeued from the queue")

print(q.popleft())

print(q.popleft())

print(q.popleft())

print("\nQueue after removing elements")

print(q)

**Implementing queue**

Implementing queue is a bit different. Queue works on the principle of “First-in, first-out”.(FIFO)

# Python code to demonstrate Implementing

# Queue using deque and list

from collections import deque

queue = deque(["Ram", "Tarun", "Asif", "John"])

print(queue)

queue.append("Akbar")

print(queue)

queue.append("Birbal")

print(queue)

print(queue.popleft())

print(queue.popleft())

print(queue)

OUTPUT:

deque(['Ram', 'Tarun', 'Asif', 'John'])

deque(['Ram', 'Tarun', 'Asif', 'John', 'Akbar'])

deque(['Ram', 'Tarun', 'Asif', 'John', 'Akbar', 'Birbal'])

Ram

Tarun

deque(['Asif', 'John', 'Akbar', 'Birbal'])

#### **append()** :- This function is used to **insert**the value in its argument to the **right end** of deque.

from collections import deque

a=deque([1,2,3,4,5])

print(a)

a.append(6)

print(a)

OUTPUT:

deque([1, 2, 3, 4, 5])

deque([1, 2, 3, 4, 5, 6])

**2. appendleft()** :- This function is used to**insert** the value in its argument to the **left end** of deque.

from collections import deque

a=deque([1,2,3,4,5])

print(a)

a.appendleft(6)

print(a)

OUTPUT:

deque([1, 2, 3, 4, 5])

deque([6, 1, 2, 3, 4, 5])

**3. pop()** :- This function is used to**delete** an argument from the**right end** of deque.

**4. popleft()** :- This function is used to**delete** an argument from the**left end** of deque.

#### **5. insert(i, a)** :- This function **inserts the value** mentioned in arguments(a) **at index(i)** specified in arguments.

from collections import deque

a=deque([1,2,3,4,5])

print(a)

a.insert(2,6)

print(a)

OUTPUT:

deque([1, 2, 3, 4, 5])

deque([1, 2, 6, 3, 4, 5])

**6. index(ele, beg, end)** :- This function **returns the first index of the value** mentioned in arguments, **starting searching from beg till end** index.

from collections import deque

a=deque([1,2,3,4,2,5,2,68])

print(a)

print(a.index(2,5,68))

OUTPUT:

6

**7. remove()** :- This function**removes the first occurrence** of value mentioned in arguments.

**8. count()** :- This function **counts the number of occurrences** of value mentioned in arguments.

**9. reverse()**:- This function is used to**reverse order** of deque elements.

from collections import deque

a=deque([1,2,3,4,5,6,7])

print(a)

a.reverse()

print(a)

OUTPUT:

deque([1, 2, 3, 4, 5, 6, 7])

deque([7, 6, 5, 4, 3, 2, 1])

**10. rotate()**:- This function**rotates the deque** by the number specified in arguments.**If the number specified is negative, rotation occurs to left. Else rotation is to right.**

from collections import deque

a=deque([1,2,3,4,5,6,7])

print(a)

a.rotate(1)

print(a)

OUTPUT:

deque([1, 2, 3, 4, 5, 6, 7])

deque([7, 1, 2, 3, 4, 5, 6])

from collections import deque

a=deque([1,2,3,4,5,6,7])

print(a)

a.rotate(-1)

print(a)

OUTPUT:

deque([1, 2, 3, 4, 5, 6, 7])

deque([2, 3, 4, 5, 6, 7, 1])

#### **Implementation using collections.queue.Queue**

import queue

# From class queue, Queue is created as an object Now L is Queue of a maximum #capacity of 20

L = queue.Queue(maxsize=20)

# Data is inserted into Queue using put() Data is inserted at the end

L.put(5)

L.put(9)

L.put(1)

L.put(7)

# get() takes data out from the Queue from the head of the Queue

print(L.get())

print(L.get())

print(L.get())

print(L.get())

OUTPUT:

5

9

1

7

1. **Creating a FIFO Queue**

// Initialize queue

Syntax: queue.Queue(maxsize)

// Insert Element

Syntax: Queue.put(data)

// Get And remove the element

Syntax: Queue.get()

import queue

# From class queue, Queue is created as an object Now L is Queue of a maximum #capacity of 20

L = queue.Queue(maxsize=20)

# Data is inserted into Queue using put() Data is inserted at the end

L.put(5)

L.put(9)

L.put(1)

L.put(7)

# get() takes data out from the Queue from the head of the Queue

print(L.get())

print(L.get())

print(L.get())

print(L.get())

OUTPUT:

5

9

1

7

1. **UnderFlow and OverFlow**

import queue

L = queue.Queue(maxsize=6)

# qsize() give the maxsize of the Queue

print(L.qsize())

L.put(5)

L.put(9)

L.put(1)

L.put(7)

# Return Boolean for Full Queue

print("Full: ", L.full())

L.put(9)

L.put(10)

print("Full: ", L.full())

print(L.get())

print(L.get())

print(L.get())

# Return Boolean for Empty Queue

print("Empty: ", L.empty())

print(L.get())

print(L.get())

print(L.get())

print("Empty: ", L.empty())

print("Full: ", L.full())

# This would result into Infinite Loop as the Queue is empty.

#print(L.get())

OUTPUT:

0

Full: False

Full: True

5

9

1

Empty: False

7

9

10

Empty: True

Full: False

**Implementation of queue using Class:**

class Queue:

    def \_\_init\_\_(self,size):

        self.items=[]

        self.size=size

    def push(self,element):

        if len(self.items)==self.size:

            print("queue is full")

        else:

            self.items.append(element)

    def pop(self):

        if self.items==[]:

            print("queue is empty")

        else:

            return print(self.items.pop(0))

    def display(self):

        return print(self.items)

q=Queue(5)

q.push(1)

q.push(2)

q.push(3)

q.push(4)

q.push(5)

q.push(6)

q.display()

q.pop()

q.pop()

q.pop()

q.pop()

q.pop()

q.pop()

q.display()

# Class Queue to represent a queue

class Queue:

    # \_\_init\_\_ function

    def \_\_init\_\_(self, capacity):

        self.front = self.size = 0

        self.rear = capacity -1

        self.Q = [None]\*capacity

        self.capacity = capacity

    # Queue is full when size becomes equal to the capacity

    def isFull(self):

        return self.size == self.capacity

    # Queue is empty when size is 0

    def isEmpty(self):

        return self.size == 0

    # Function to add an item to the queue.It changes rear and size

    def EnQueue(self, item):

        if self.isFull():

            print("Full")

            return

        self.rear = (self.rear + 1) % (self.capacity)

        self.Q[self.rear] = item

        self.size = self.size + 1

        print("%s enqueued to queue"  %str(item))

    # Function to remove an item from queue. It changes front and size

    def DeQueue(self):

        if self.isEmpty():

            print("Empty")

            return

        print("%s dequeued from queue" %str(self.Q[self.front]))

        self.front = (self.front + 1) % (self.capacity)

        self.size = self.size -1

    # Function to get front of queue

    def que\_front(self):

        if self.isEmpty():

            print("Queue is empty")

        print("Front item is", self.Q[self.front])

    # Function to get rear of queue

    def que\_rear(self):

        if self.isEmpty():

            print("Queue is empty")

        print("Rear item is",  self.Q[self.rear])

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

    queue = Queue(30)

    queue.EnQueue(10)

    queue.EnQueue(20)

    queue.EnQueue(30)

    queue.EnQueue(40)

    queue.DeQueue()

    queue.que\_front()

    queue.que\_rear()

OUTPUT:

10 enqueued to queue

20 enqueued to queue

30 enqueued to queue

40 enqueued to queue

10 dequeued from queue

Front item is 20

Rear item is 40