25/Dec/22

Data structures (Python)

Data structures is a way to store and organize the data so that it can be accessed effectively.

Built_in data structure	User defined data structure
List	Stack
Tuple	Queue
set	Linked List
Dictionary	Tree
	graph

---

① List : [ ] → Mutable

Negative Indexing in List

10	20	30	40
-4	-3	-2	-1

Square braces technique : list_name = [group of values]

List constructor : list_name = list (range() / str / sequence)

1. Duplicates are allowed
2. guarantee the order in o/p.
3. Can store data of different data type
4. List can be Nested

---

② Tuple : ( ) → Immutable

Round braces technique : tuple_name = (group of values)

Tuple constructor : tuple_name = tuple (range () / str / sequence)

1. Duplicates are allowed
2. guarantee the order in o/p
3. Can store data of different type
4. Tuples can be Nested

---

③ Set : { } → Mutable

Flower braces technique : set_name = { group of values }

set Constructor : Set_name = set (range() / str / sequence)

① Duplicates are not allowed
② order is not guarenteed
③ set can't be Nested
④ set can store diff. datatype

---

④ Dictionaries : { } → Mutable

Flower braces technique : dict_name = {k1 : v1 , k2 : v2 , k3 : v3....}

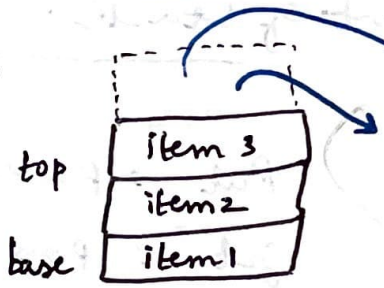
Dictionaries constructor : dict_name = dict (sequence)

① Key – Duplicates are not allowed (unique)
Values – Can be duplicated

② Order is Not Guaranteed

③ Dictionaries can be Nested.

Stack :



Last In First Out (LIFO)

operations : Push ----> append
Pop ----> Pop
top
isEmpty

* Stack is used to reverse a string.

① Implement stack using List :

```
Stack = [ ]
stack.append(10) // insert element in list
stack.pop() // remove last element from stack
len(stack) == 0 // check stack isEmpty (or) Not
Stack [-1] // Top of stack (last element in list)

n = Int(Input("Limit of stack"))
len(stack) == n // Then the list is full (overflow)
```

② Implement stack using Modules :

① Collections module (Deque class) → Double ended queue

```
import collections
stack = collections.deque()
stack.append(10)
stack.append(20)  // deque([10, 20])
stack.pop()  // remove 20 from list
not stack  // True, if the stack is Empty
stack[-1]  // check Top element of stack.
```

---

② Queue module (Lifo Queue class)

Put → push
get → pop

```
Import queue
stack = queue.LifoQueue()
stack.put(10)
stack.put(20)
stack.get()  // remove 20
```

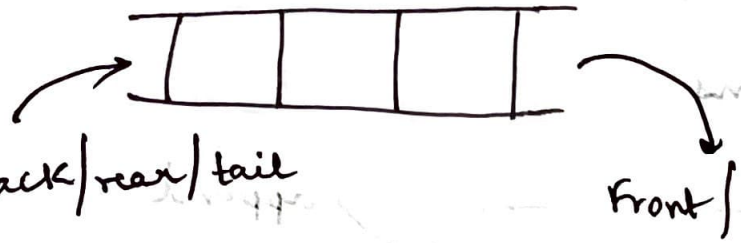
stack.put(20, timeout=1)

Timeout parameter is used, so that the program won't stop when stack is empty (or) when stack is full

↓

Timeout will display message like "stack is empty" (or) "stack is full" after 1 sec

Queue : FIFO → First In First out



* adding element (enqueue) → rear++ (→ append method)
* remove element (dequeue) → Front-- (→ Pop method, queue.pop(0))
* is Full
* is Empty (Not queue → True → Then queue is empty)

① Implement Queue using List :

```
queue = [ ]
queue.append (10)
queue.append (20)
queue.append (30)   // [10, 20, 30]
queue.Pop (0)       // Pop will takes place from front (10 will be removed).
queue.Pop (0)       // 20 will be removed
queue.Pop (0)       // 30 will be removed.
```

To check front element → queue [0]
To check rear element → queue [-1]

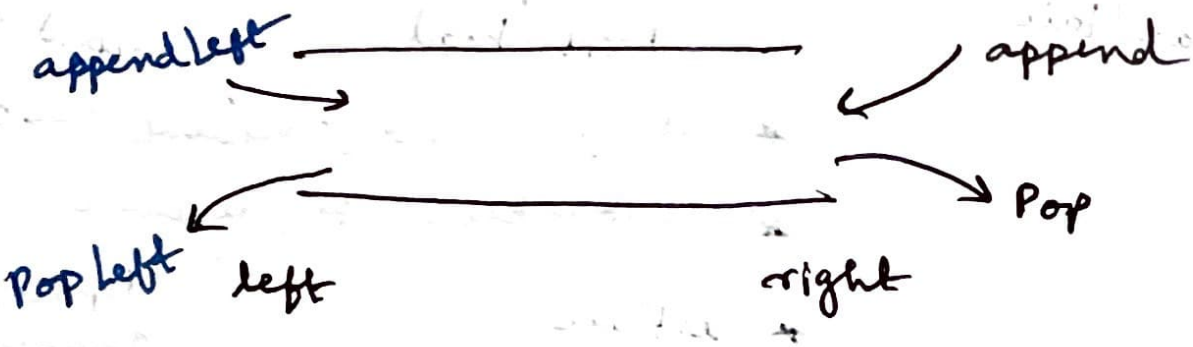
front ← rear

```
queue = [ ]
queue.Insert (0,10)
queue.Insert (0,20)
queue.Insert (0,30) // [30, 20, 10]
queue.Pop ()        // remove 10
queue.Pop ()        // remove 20
```

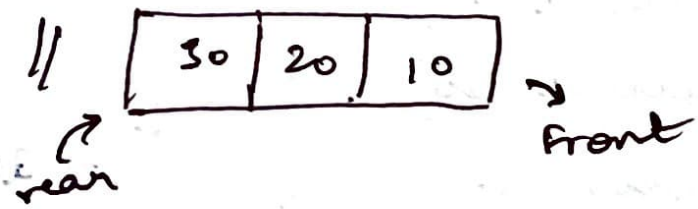
rear → Front

Queue Implementation using Classes

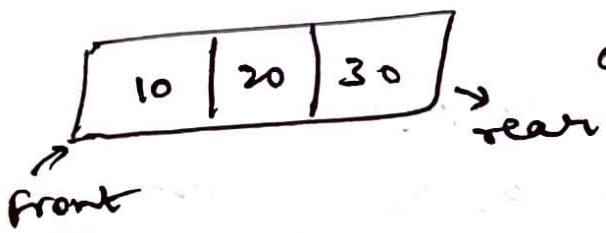
① deque → collections




```
import collections
q = collections.deque ()  // deque([]) empty queue
q.appendleft (10)
q.appendleft (20)
q.appenleft (30)
q.Pop ( ) // remove 10
q.Pop ( ) // remove 20
```

// 

```
import collections
q = collections.deque ( )
q.append (10)
q.append (20)
q.append (30)
q.Popleft ( ) // remove 10
q.Pop left ( ) // remove 20
```



deque([10, 20, 30])

② queue module → we have queue class

(i) Put (item, block = True, timeout) → To insert a element in queue

Put_nowait (item)

(ii) get (block = True, timeout = None) → Remove element from queue.

get_nowait ( )

```
Import queue
q = queue.Queue ( )
q.put (10)
q.put (50)
q.put (30) ||  < queue.Queue object at 0x000001E7CD38850 >
q.get ( ) || 10 will be removed
q.get ( ) || 50 will be removed next
```

Priority Queue → Implemented by using

① List!
```
q = [ ]
q.append (10)
q.append (40)
q.sort( ) || [10, 40]
```

② Priority Queue from Queue Module!
```
Import queue
q = queue.PriorityQueue ( )
q.put (10)
q.put (60)
q.put (20)
q.get ( ) || 10 removed
q.get ( ) || 20 removed
```

Queue (FIFO)

10	20	30	40	100

Priority queue

The elements are removed, based on "Priority"

① Lowest element → high priority → [10, 20, 30, 40, 100]

(or)

② highest element → high Priority → [100, 40, 30, 20, 10]

(use q.sort(reverse = True))