

New  
syllabus  
2021-22



Chapter 7  
Data-structures:  
lists, stack, queue

# Computer Science Class XII ( As per CBSE Board)

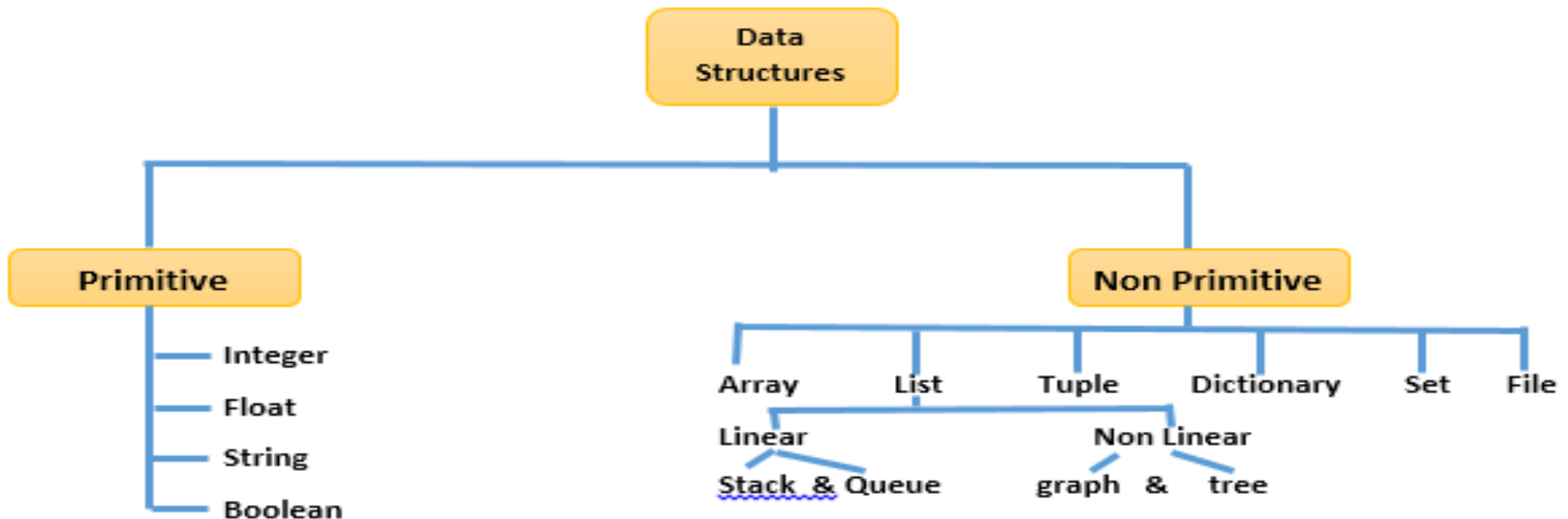
Visit : [python.mykvs.in](http://python.mykvs.in) for regular updates

# Data-structures



It a way of organizing and storing data in such a manner so that it can be accessed and work over it can be done efficiently and less resources are required. It define the relationship between the data and the operations over those data. There are many various types of data structures defined that make it easier for the computer programmer, to concentrate on the main problems rather than getting lost in the details of data description and access.

## Python Data Structure



# Data-structures



## List

It is a collection of items and each item has its own index value.

Index of first item is 0 and the last item is n-1. Here n is number of items in a list.

## Indexing of list

|    |    |    |    |    |                |
|----|----|----|----|----|----------------|
| 0  | 1  | 2  | 3  | 4  | index          |
| 80 | 60 | 70 | 85 | 75 | value          |
| -5 | -4 | -3 | -2 | -1 | Negative index |

## Creating a list

Lists are enclosed in square brackets [ ] and each item is separated by a comma.

e.g.

```
list1 = ['English', 'Hindi', 1997, 2000];
```

```
list2 = [11, 22, 33, 44, 55];
```

```
list3 = ["a", "b", "c", "d"];
```



## Access Items From A List

List items can be accessed using its index position.

e.g.

```
list =[3,5,9]
```

```
print(list[0])
```

```
print(list[1])
```

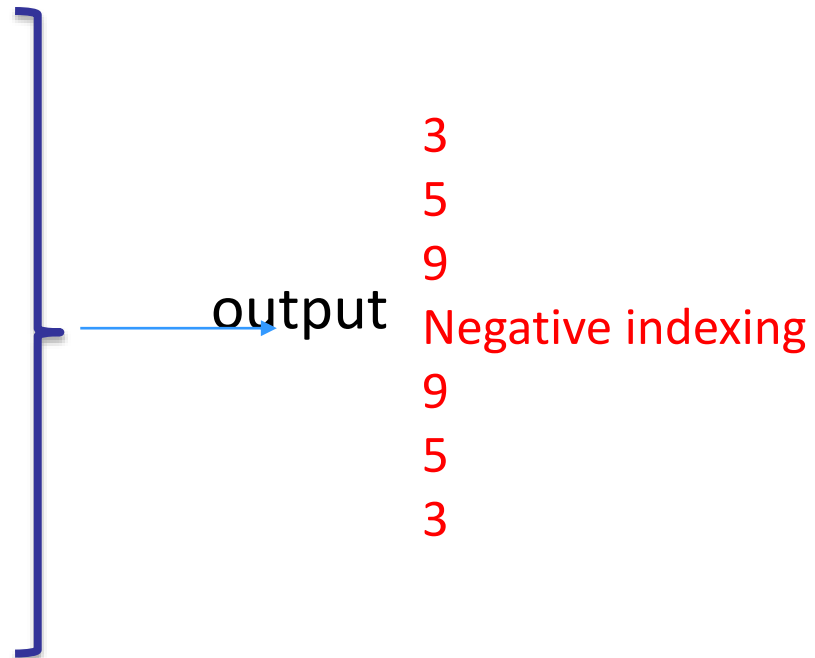
```
print(list[2])
```

```
print('Negative indexing')
```

```
print(list[-1])
```

```
print(list[-2])
```

```
print(list[-3])
```





## Iterating Through A List

List elements can be accessed using looping statement.

e.g.

```
list =[3,5,9]
for i in range(0, len(list)):
    print(list[i])
```

Output

```
3
5
9
```



## Important methods and functions of List

| Function                     | Description                                               |
|------------------------------|-----------------------------------------------------------|
| <code>list.append()</code>   | Add an Item at end of a list                              |
| <code>list.extend()</code>   | Add multiple Items at end of a list                       |
| <code>list.insert()</code>   | insert an Item at a defined index                         |
| <code>list.remove()</code>   | remove an Item from a list                                |
| <code>del list[index]</code> | Delete an Item from a list                                |
| <code>list.clear()</code>    | empty all the list                                        |
| <code>list.pop()</code>      | Remove an Item at a defined index                         |
| <code>list.index()</code>    | Return index of first matched item                        |
| <code>list.sort()</code>     | Sort the items of a list in ascending or descending order |
| <code>list.reverse()</code>  | Reverse the items of a list                               |
| <code>len(list)</code>       | Return total length of the list.                          |
| <code>max(list)</code>       | Return item with maximum value in the list.               |
| <code>min(list)</code>       | Return item with min value in the list.                   |
| <code>list(seq)</code>       | Converts a tuple, string, set, dictionary into list.      |

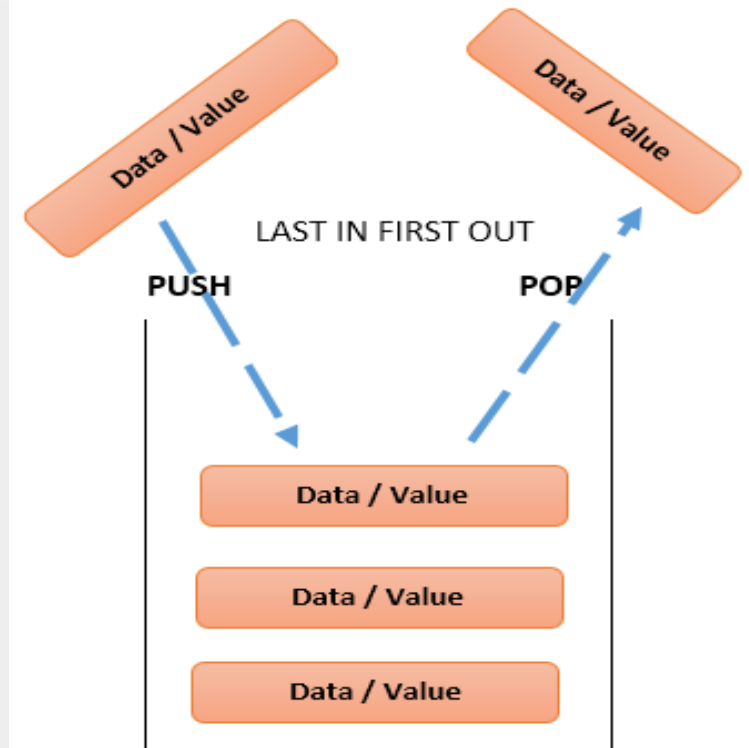
For detail on list [click here](#)

# Data-structures

## Stack:

A stack is a linear data structure in which all the insertion and deletion of data / values are done at one end only.

- It is type of linear data structure.
- It follows LIFO(Last In First Out) property.
- Insertion / Deletion in stack can only be done from top.
- Insertion in stack is also known as a PUSH operation.
- Deletion from stack is also known as POP operation in stack.





## Applications of Stack:

- **Expression Evaluation:** It is used to evaluate prefix, postfix and infix expressions.
- **Expression Conversion:** It can be used to convert one form of expression(prefix,postfix or infix) to one another.
- **Syntax Parsing:** Many compilers use a stack for parsing the syntax of expressions.
- **Backtracking:** It can be used for back traversal of steps in a problem solution.
- **Parenthesis Checking:** Stack is used to check the proper opening and closing of parenthesis.
- **String Reversal:** It can be used to reverse a string.
- **Function Call:** Stack is used to keep information about the active functions or subroutines.





## Using List as Stack in Python:

The concept of Stack implementation is easy in Python , because it support inbuilt functions (`append()` and `pop()`) for stack implementation.By Using these functions make the code short and simple for stack implementation.

To add an item to the top of the list, i.e., to push an item, we use `append()` function and to pop out an element we use `pop()` function. These functions work quiet efficiently and fast in end operations.

# Data-structures

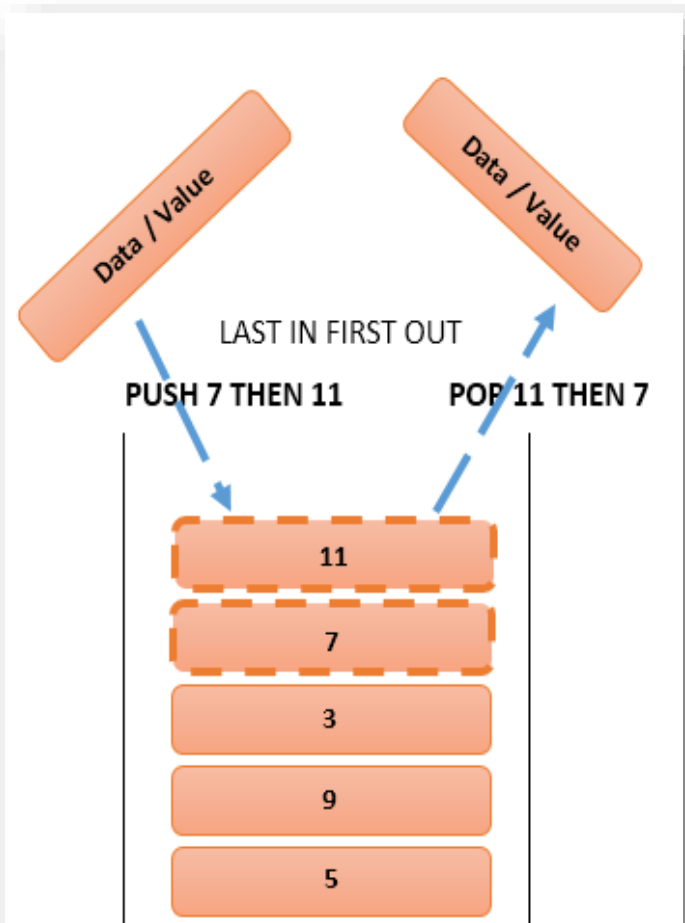


## Stack e.g. program:

```
stack = [5, 9, 3]
stack.append(7)
stack.append(11)
print(stack)
print(stack.pop())
print(stack)
print(stack.pop())
print(stack)
```

OUTPUT


→ [5, 9, 3, 7, 11]  
→ 11  
→ [5, 9, 3, 7]  
→ 7  
→ [5, 9, 3]



# Data-structures

---

## Stack interactive program:



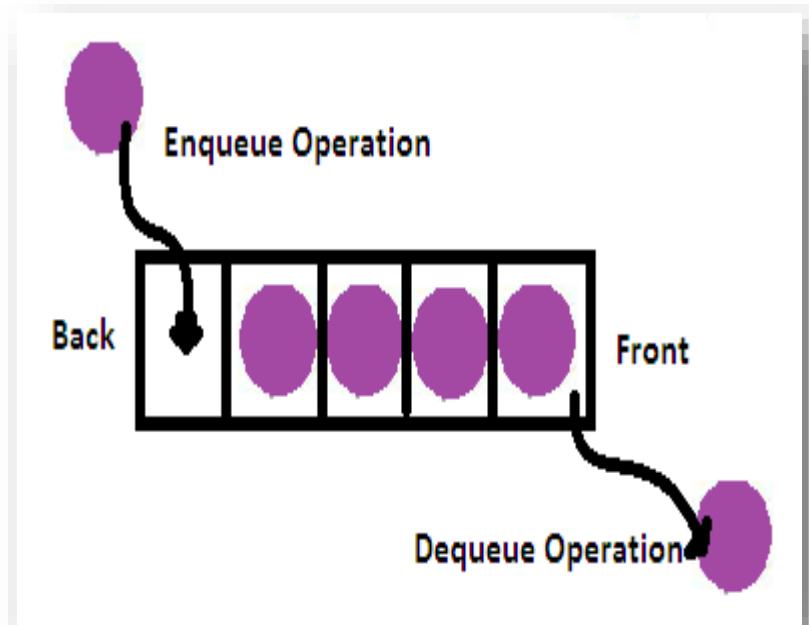
```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def push(self, data):
        self.items.append(data)
    def pop(self):
        return self.items.pop()
s = Stack()
while True:
    print('Press 1 for push')
    print('Press 2 for pop')
    print('Press 3 for quit')
    do = int(input('What would you like to do'))
    if do == 1:
        n=int(input("enter a number to push"))
        s.push(n)
    elif do == 2:
        if s.is_empty():
            print('Stack is empty.')
        else:
            print('Popped value: ', s.pop())
    elif operation == 3:
        break #Note :- Copy and paste above code in python file then execute that file
```

# Data-structures

## Queue:

Queue is a data structures that is based on First In First Out (FIFO) strategy ,i.e. the first element that is added to the queue is the first one to be removed.

- Queue follows the FIFO (First - In - First Out) structure.
- According to its FIFO structure, element inserted first will also be removed first.
- In a queue, one end is always used to insert data (enqueue) and the other is used to delete data (dequeue), because queue is open at both its ends.





## Applications of Queue:

**Synchronization** : When data are transferred to asynch devices then it is used to synchronized.

**Scheduling** : When a resource is shared among multiple consumers.

**Searching** : Like breadth first search in graph theory.

**Interrupt handling** : Handling of multiple interrupt as the order they arrive.



## Using List as Queue in Python:

The concept of Queue implementation is easy in Python , because it support inbuilt functions (insert() and pop()) for queue implementation.By Using these functions make the code short and simple for queue implementation.

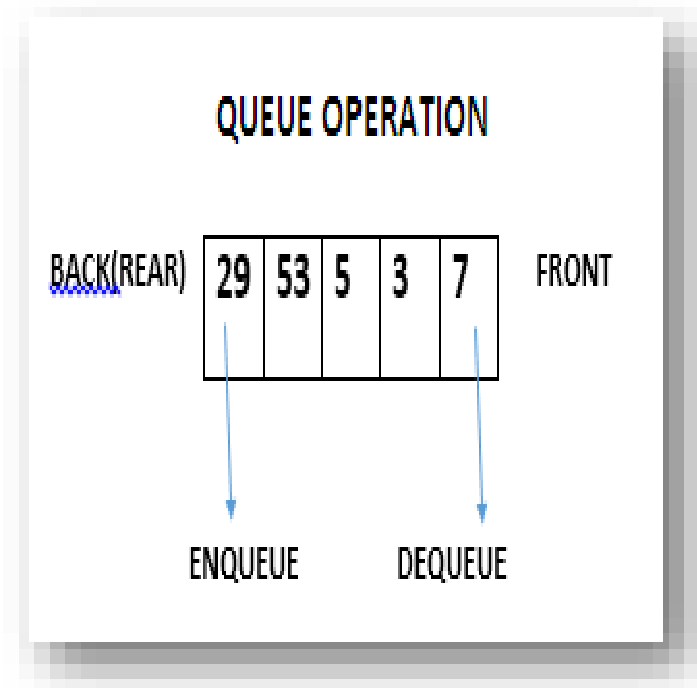
To add an item at front of the queue, i.e., to enqueue an item, we use insert() function and to dequeue an element we use pop() function. These functions work quiet efficiently and fast in end operations.

# Data-structures




Queue e.g. program:

```
queue = [5, 3, 7]           OUTPUT  
print(queue)               → [5, 3, 7]  
queue.insert(0,53)  
print(queue)               → [53, 5, 3, 7]  
queue.insert(0,29)  
print(queue)               → [29, 53, 5, 3, 7]  
print(queue.pop())         → 7  
print(queue.pop())         → 3  
print(queue)               → [29, 53, 5]
```



# Data-structures

## Queue Interactive program:



```
class Queue:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def enqueue(self, item):
        self.items.insert(0,item)
    def dequeue(self):
        return self.items.pop()
    def size(self):
        return len(self.items)
q = Queue()
while True:
    print('Press 1 for insert')
    print('Press 2 for delete')
    print('Press 3 for quit')
    do = int(input('What would you like to do'))
    if do == 1:
        n=int(input("enter a number to push"))
        q.enqueue(n)
    elif do == 2:
        if q.isEmpty():
            print('Queue is empty.')
        else:
            print('Deleted value: ', q.dequeue())
    elif operation == 3:
        break
```

#Note :- Copy and paste above code in python file then execute that file

Visit : [python.mykvs.in](http://python.mykvs.in) for regular updates