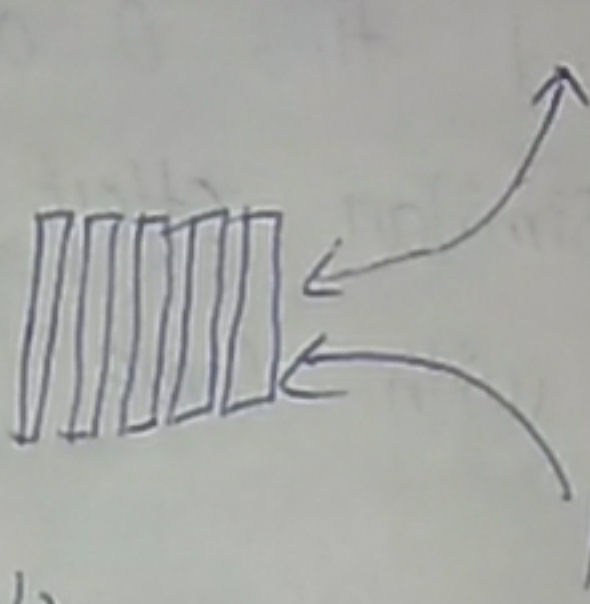
Stack

Day-4

Stack is a linear data structure which follows a particular order in which operations are performed. The order maybe (first in last) or (last in first out)



Stack (insertion and deletion happen on same end)

last in first out

Application of stack:

* Balancing of symbols
* infix to postfix / prefix Conversion
* Redo / undo features at many places like editors, photoshops.
* forward and backward feature in web browsers.
* used in many algorithms like tower of Hanoi, tree traversal

Operations are performed in Stack:-

* Push & Pop :- Adds an item in the stack. If stack is full, then it said to be an Overflow Condition.

* pop :- Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty. then it is said to be an underflow Condition.

peek or top :-

returns top element of Stack

is empty :-

Returns true if Stack is empty else. false.

Time Complexity of Operations on Stack :-

all above operations are take O(1) time.

Implementation :-

* Using array
* using linked list

Implementing Stack using Array :-

```
From SYS import maxsize
Class Stack :
    def __init__(self, Stack):
        SelF.Stack = []
        return Stack
    def isempty(Stack):
        return len(Stack) ==0
    def Push(Stack, item):
        self.Stack.append(item)
        return (item + " pushed to stack")
    def POP(Stack):
        if (isempty(Stack)):
            return str(-maxsize-1)
        return self.Stack.POP()
    def Peek(Stack):
        if (isempty(Stack)):
            return str(-maxsize-1)
        return Stack[len(Stack)-1]
```

Pros :- (using array) :-

* easy to implement. memory is saved as pointers. are not involved.

Cons :

* it is not dynamic, it doesn't grow and shrink depending on needs at runtime

Implementing Stack using linked list :-

```
Class classnode :-
    def __init__ (self, data)
        self.data = data
        self.next = None
class stack :
    def __init__ (self):
        self.root = None
    def isempty (self):
        return True if self.root is None
.
else False.
    def push (self, data)
        newnode = classnode (data)
        newnode.next = self.root
```

```
        self.root = newnode
        print "%d pushed to stack %(data)

def pop(self):
    if (self.isEmpty()):
        return float("-inf")
    temp = self.root
    self.root = self.root.next
    popped = temp.data
    return popped

def peek(self):
    if self.isEmpty():
        return float("-inf")
    return self.root.data
```

Pros :-

The linked list implementation of stack grow shrink according to the needs at runtime

Cons :-

Requires extra memory due to involvement of pointers.