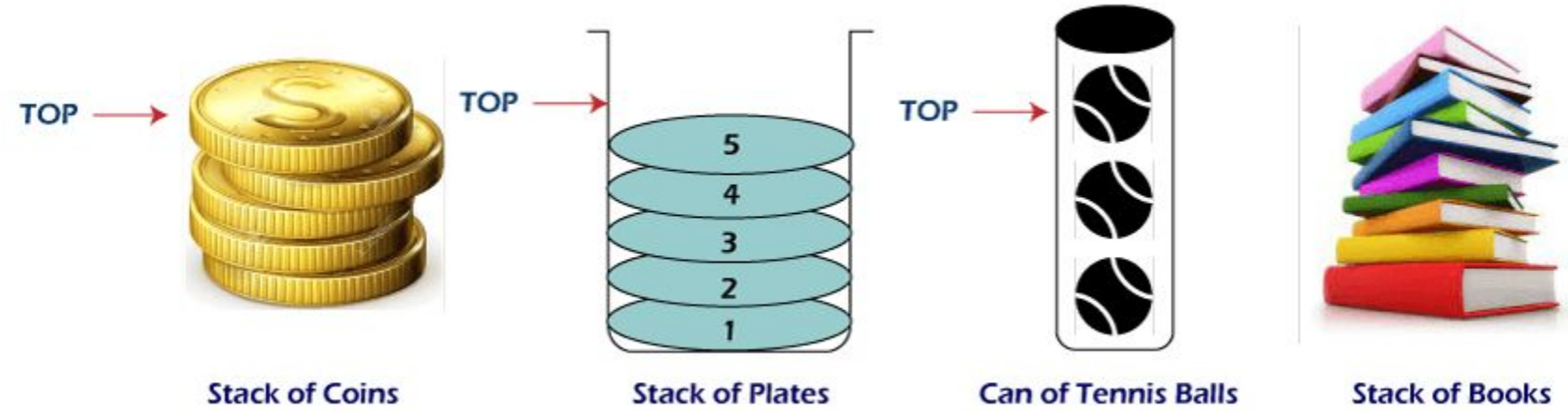
Data Structures

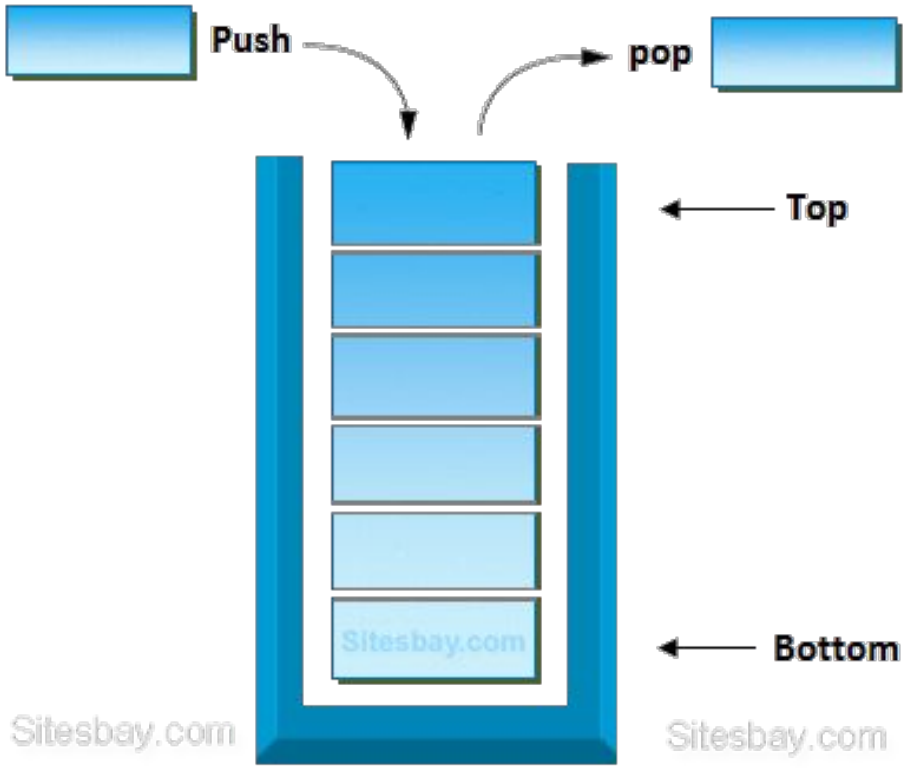
Lecture 5
Stack

Credit: Prantik Paul, Lecturer, BRAC University

Stack (In Real life)

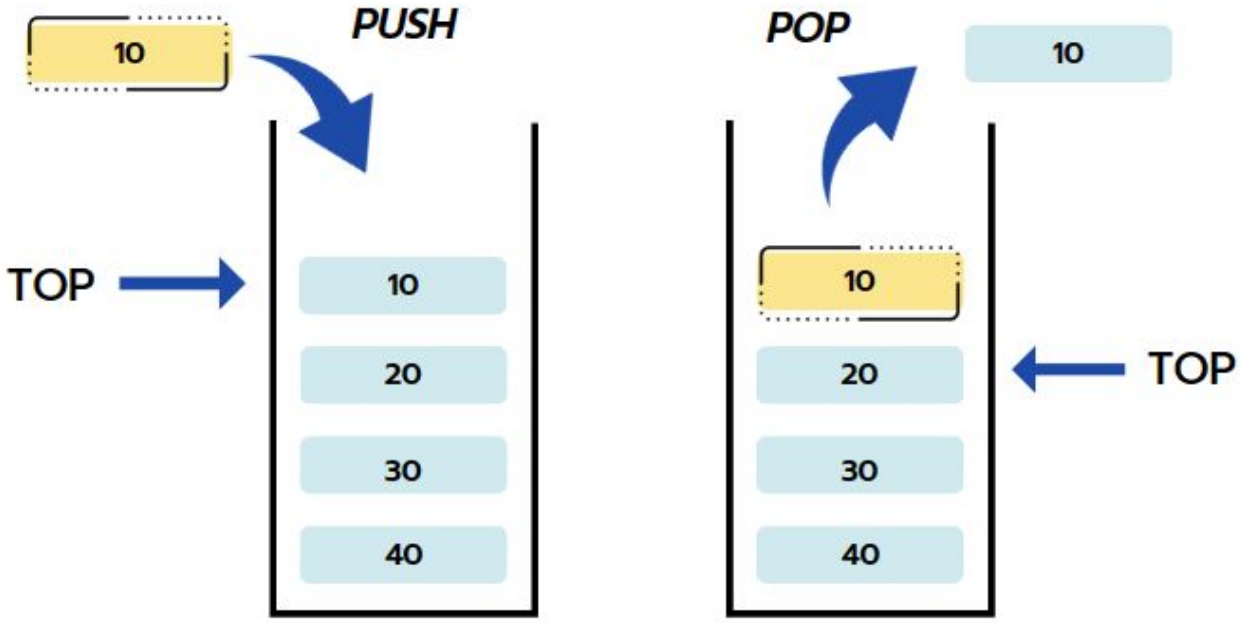


Stack (In Data Structure)



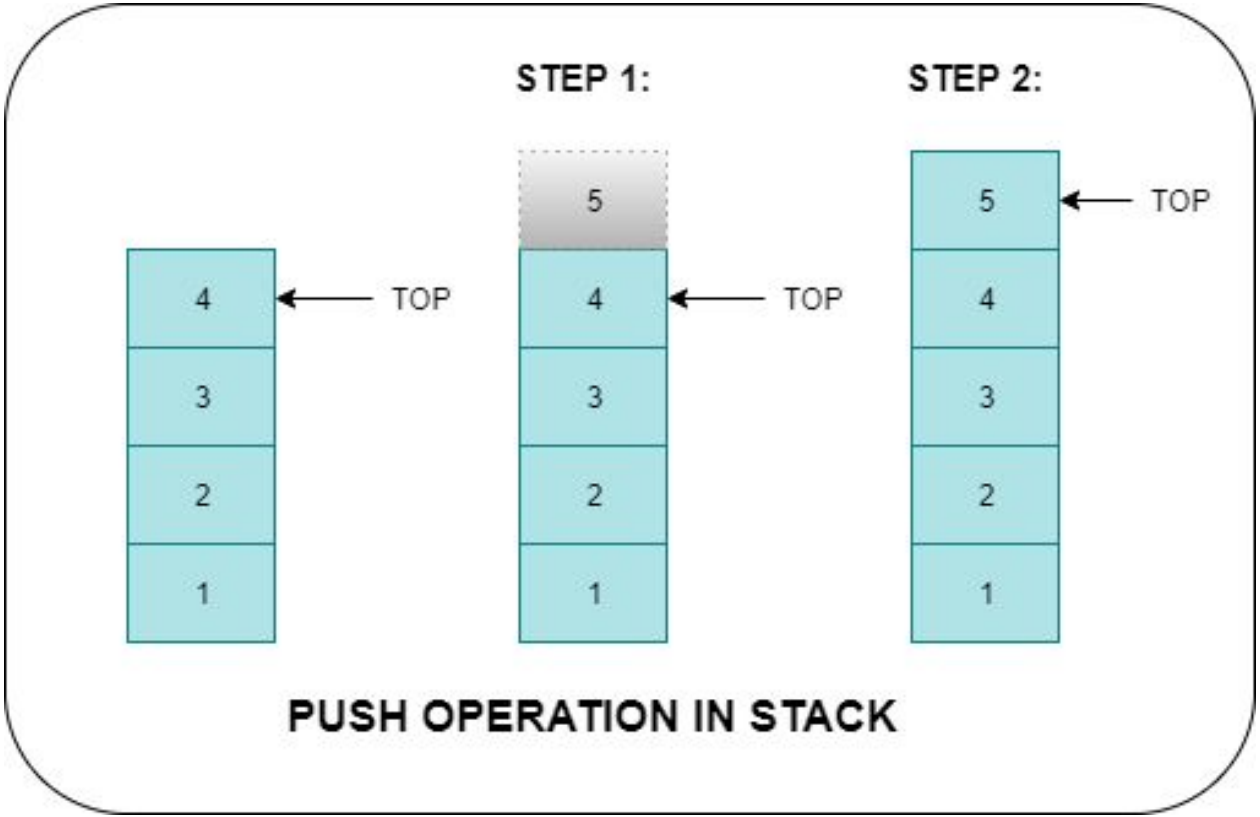
Stack (In CS Problems)

Stack (Basic Actions)

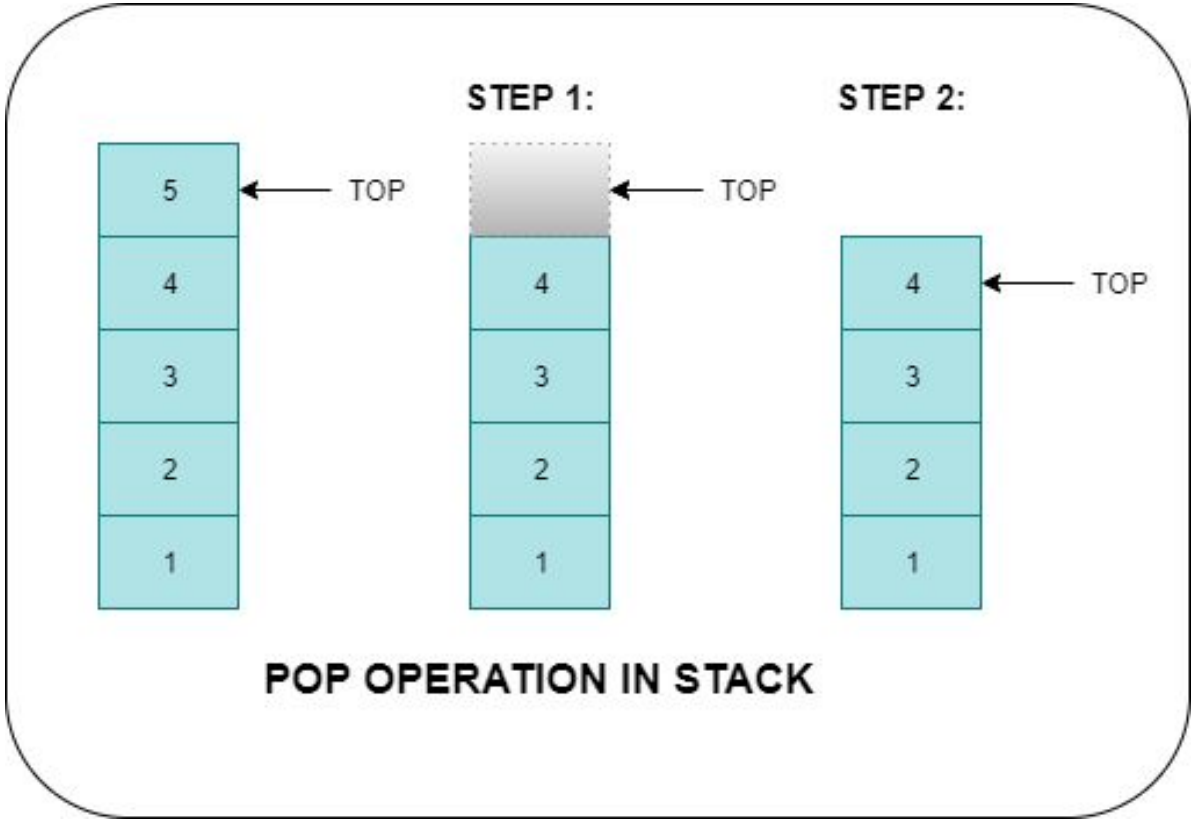


STACK DATA STRUCTURE

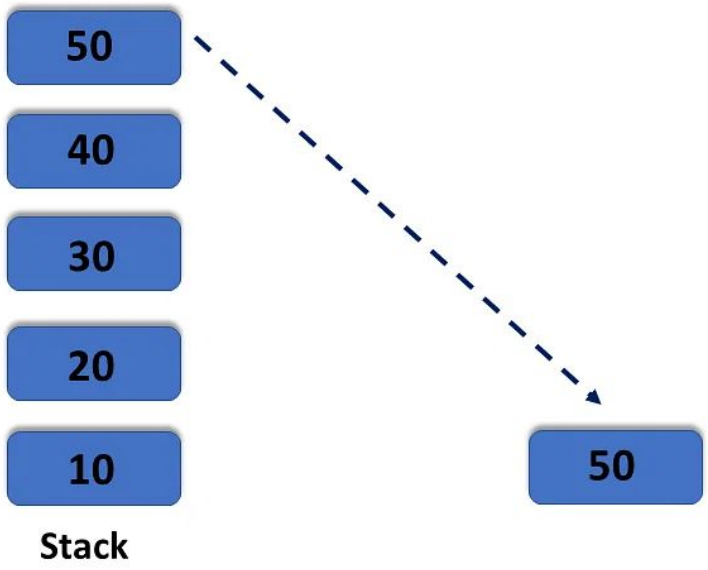
Stack (Push Operation)



Stack (Pop Operation)

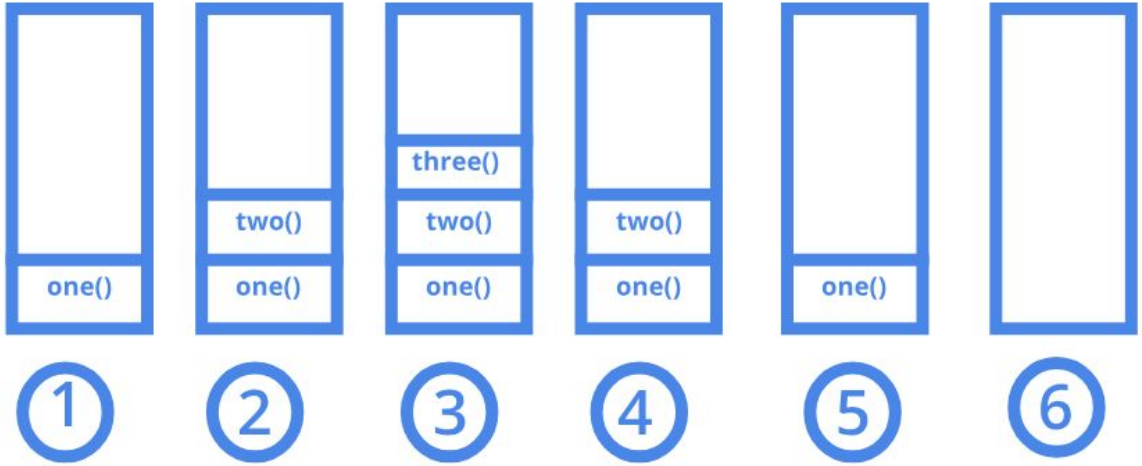


Stack (Peek Operation)

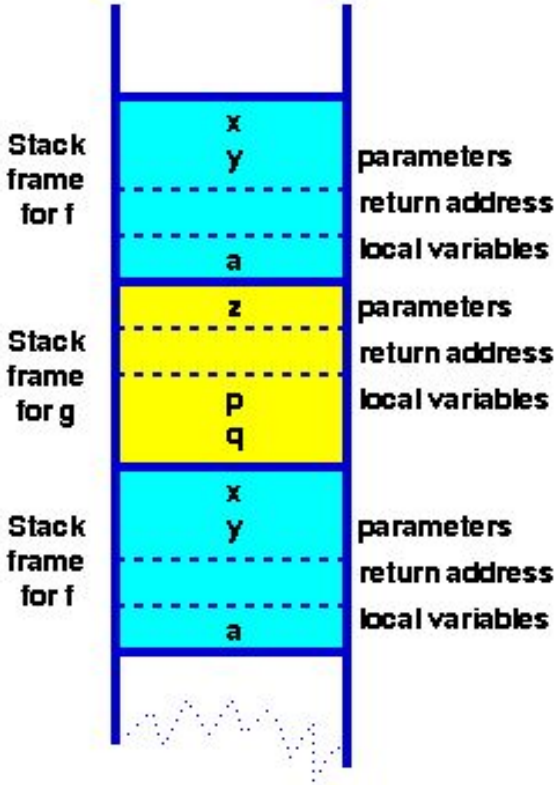


Stack Applications (Call Stack)

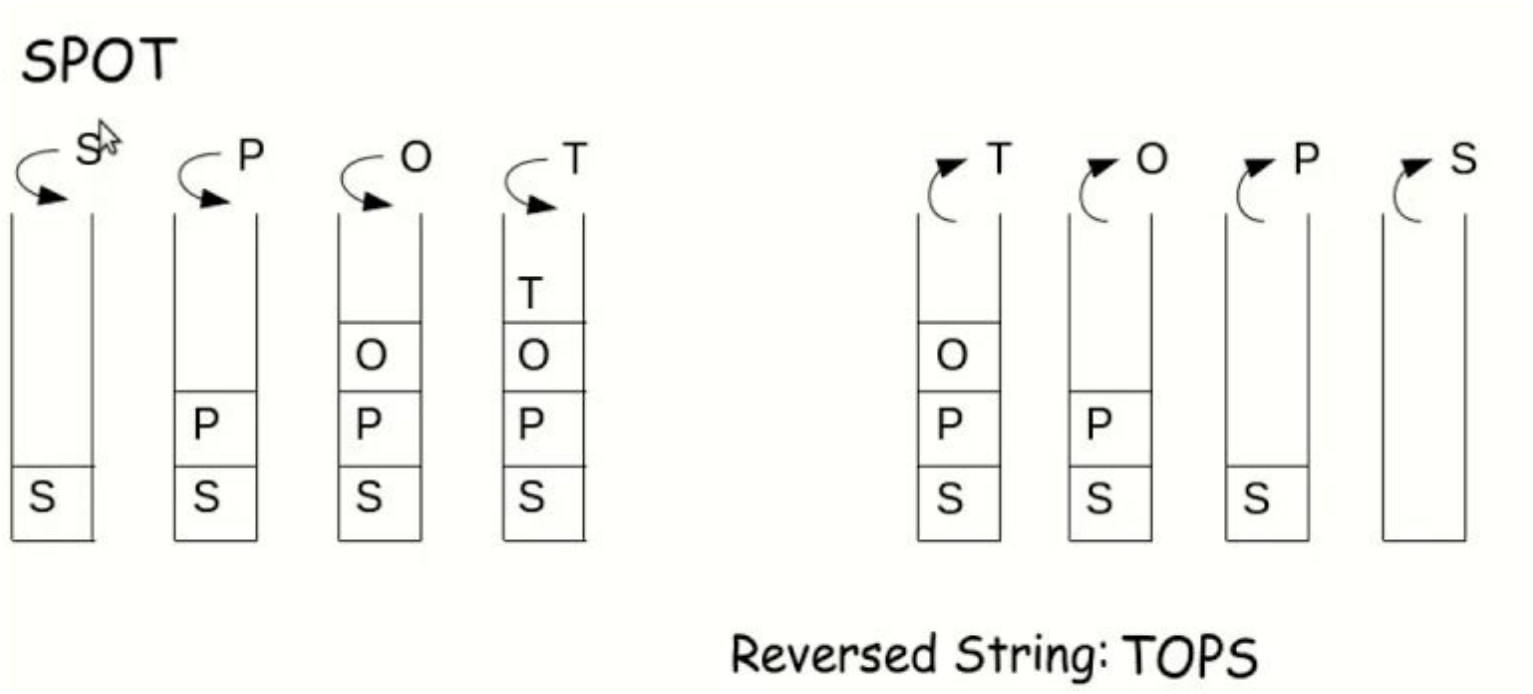
```
function one(){
  two();
}

function two() {
  three();
}

function three() {
  console.trace("Call
                Stack");
}
```



Stack Applications (Function Stack)



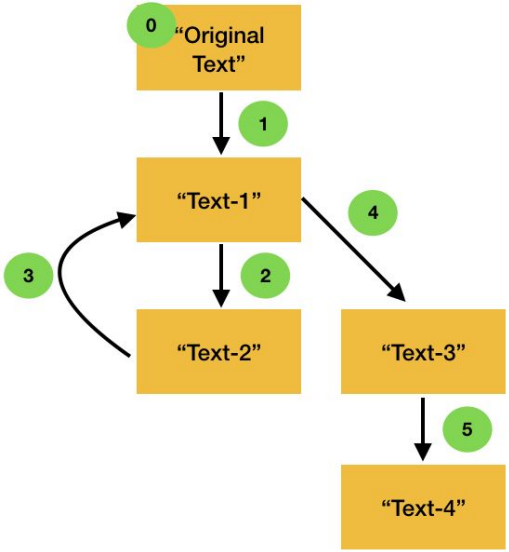
Stack Applications (Reverse String)

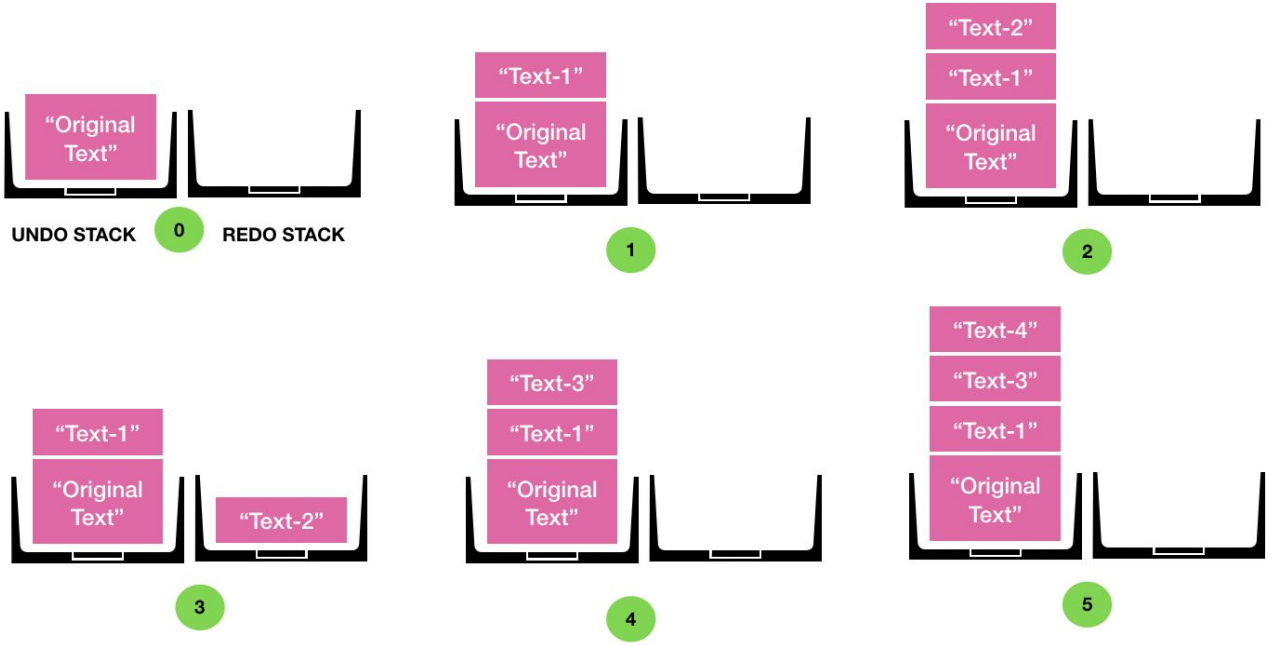


Stack Applications (Reverse String)

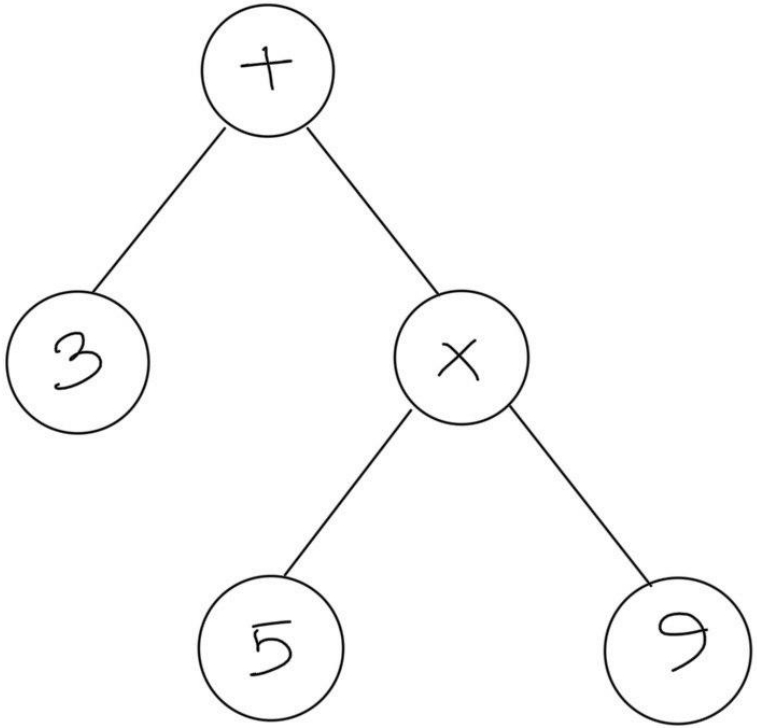
```
begin with an empty stack and an input stream.
while there is more characters to read, do:
        read the next input character;
        push it onto the stack;
end while;
while the stack is not empty, do:
        c = pop the stack;
        print c;
end while;
```

Stack Applications (Undo Operation)





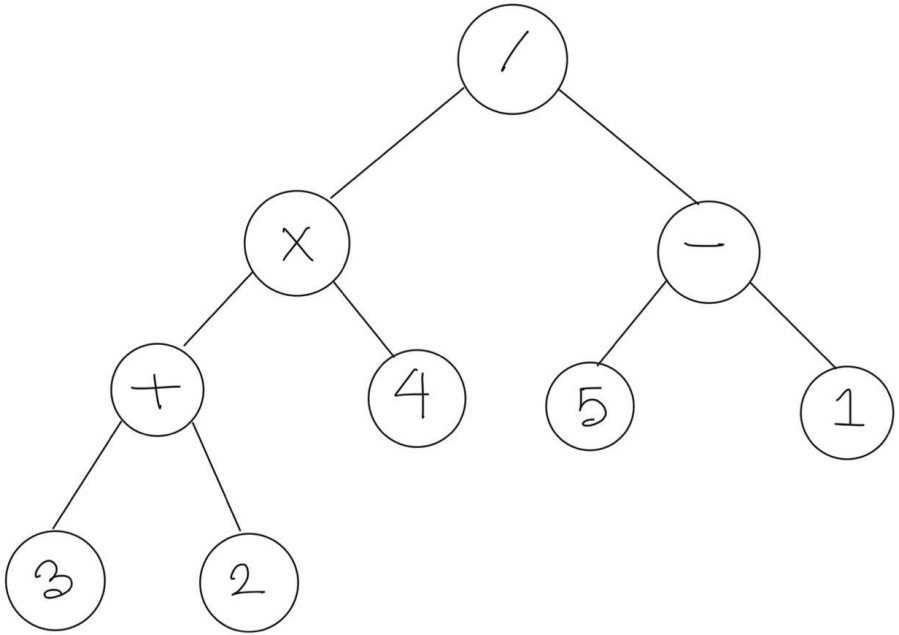
Stack Applications (Postfix Notation)



3 + 5 * 9

3 5 9 * +

Stack Applications (Postfix Notation)




```
((3 + 2) * 4) / (5 - 1)
```

```
3 2 + 4 * 5 1 - /
```

Stack Applications (Expression Evaluation)

`3 2 + 4 * 5 1 - /`

Stack	Expression
Empty initially	3 2 + 4 * 5 1 - /
2 3	+ 4 * 5 1 - /
4 5	* 5 1 - /
5 20	1 - /
4 20	/
3	2 + 4 * 5 1 - /
5	4 * 5 1 - /
20	5 1 - /
1 5 20	- /
5	Finished, and the result is at the top of the stack

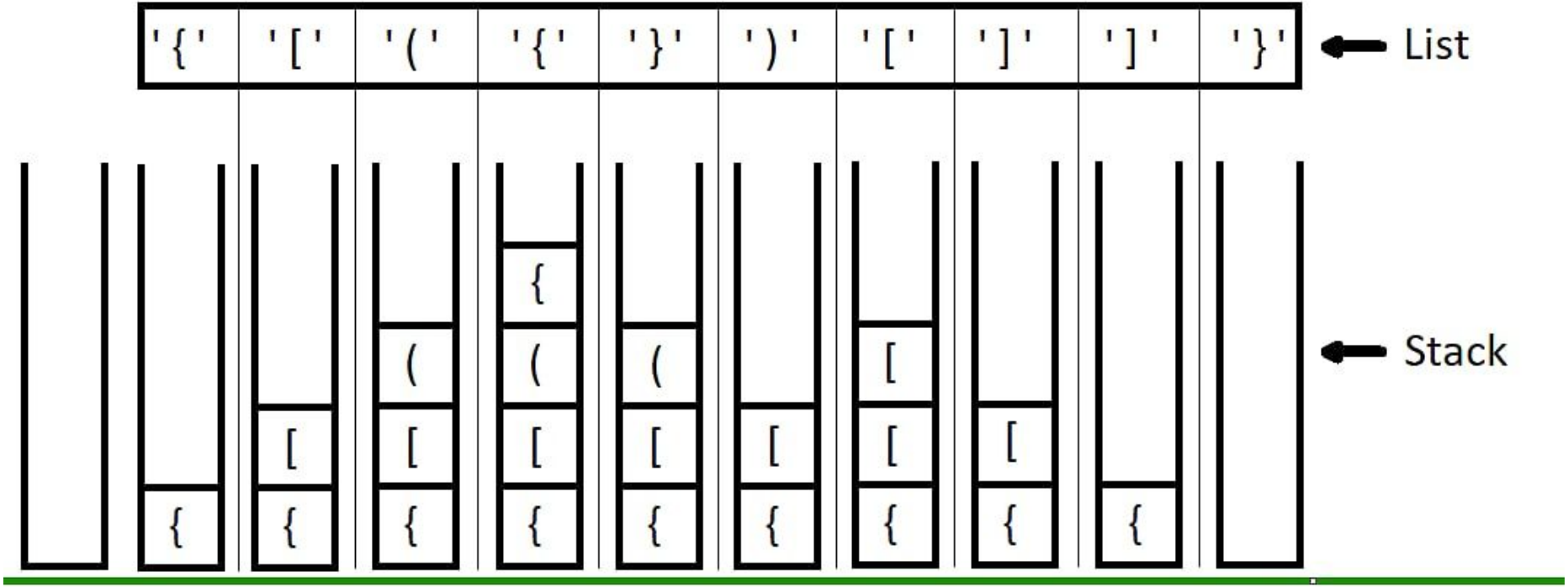
Stack Applications (Expression Evaluation)

```
begin with an empty stack and an input stream (for the expression).
while there is more input to read, do:
        read the next input symbol;
        if it's an operand,
            then push it onto the stack;
        if it's an operator
            then pop two operands from the stack;
            perform the operation on the operands;
            push the result;
end while;
// the answer of the expression is waiting for you in the
stack: pop the answer;
```

Stack Applications (Parenthesis Matching)

(a (b + c) + d)	(a (b + c) + d
[(a b) (c d)]	[(a b] (c d))
([a {x y} b])	([a {x y) b])
Correct	Incorrect

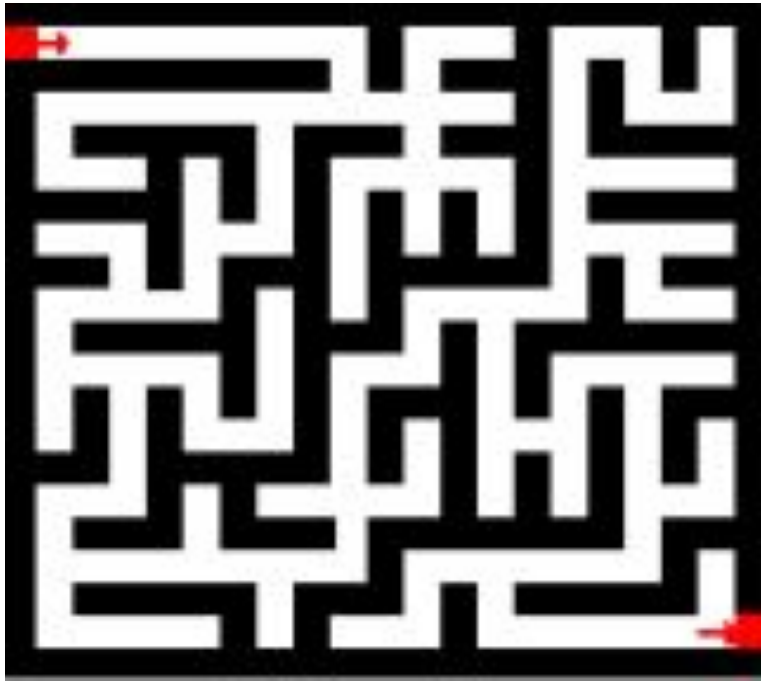
Stack Applications (Parenthesis Matching)



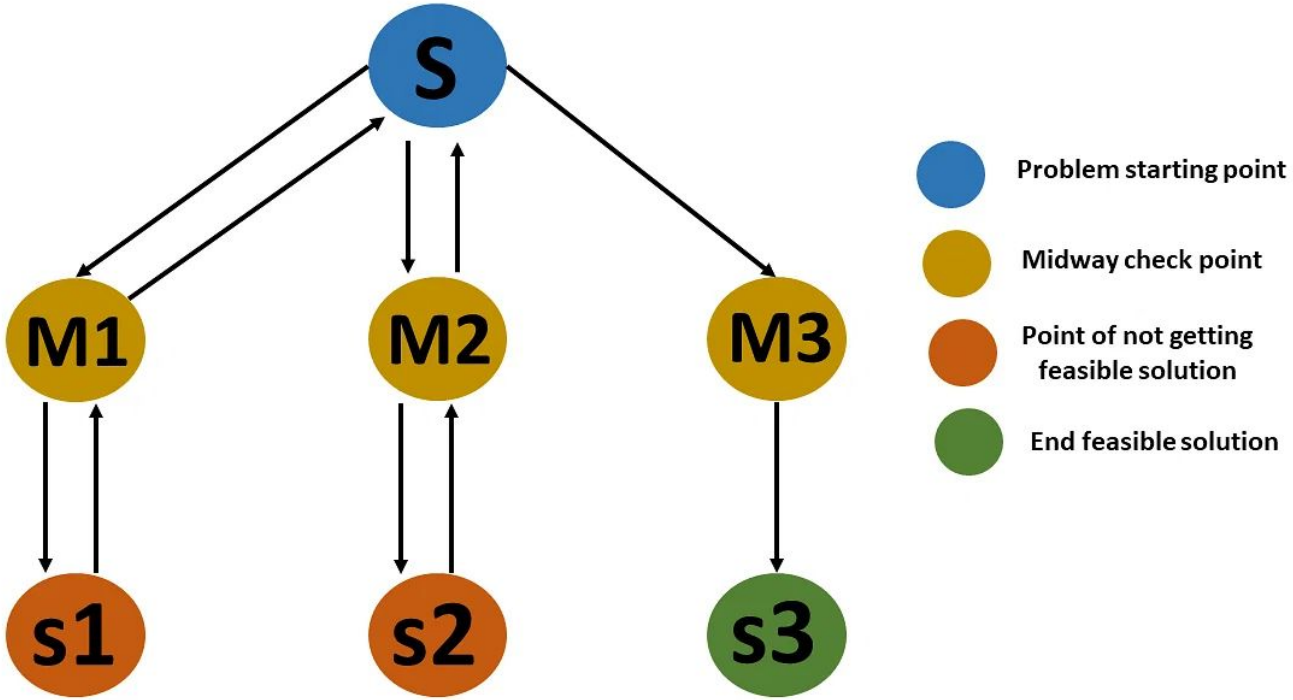
Stack Applications (Parenthesis Matching)

```
begin with an empty stack and an input stream (for the expression).
while there is more input to read, do:
      read the next input character;
      if it's an opening parenthesis/brace/bracket ("(" or "{" or "[")
         then push it onto the stack;
      if it's a closing parenthesis/brace/bracket (")" or "}" or "]")
         then pop the opening symbol from stack;
         compare the closing with opening symbol;
         if it matches
            then continue with next input character;
         if it does not match
            then return false;
end while;
// all matched, so return true
return true;
```

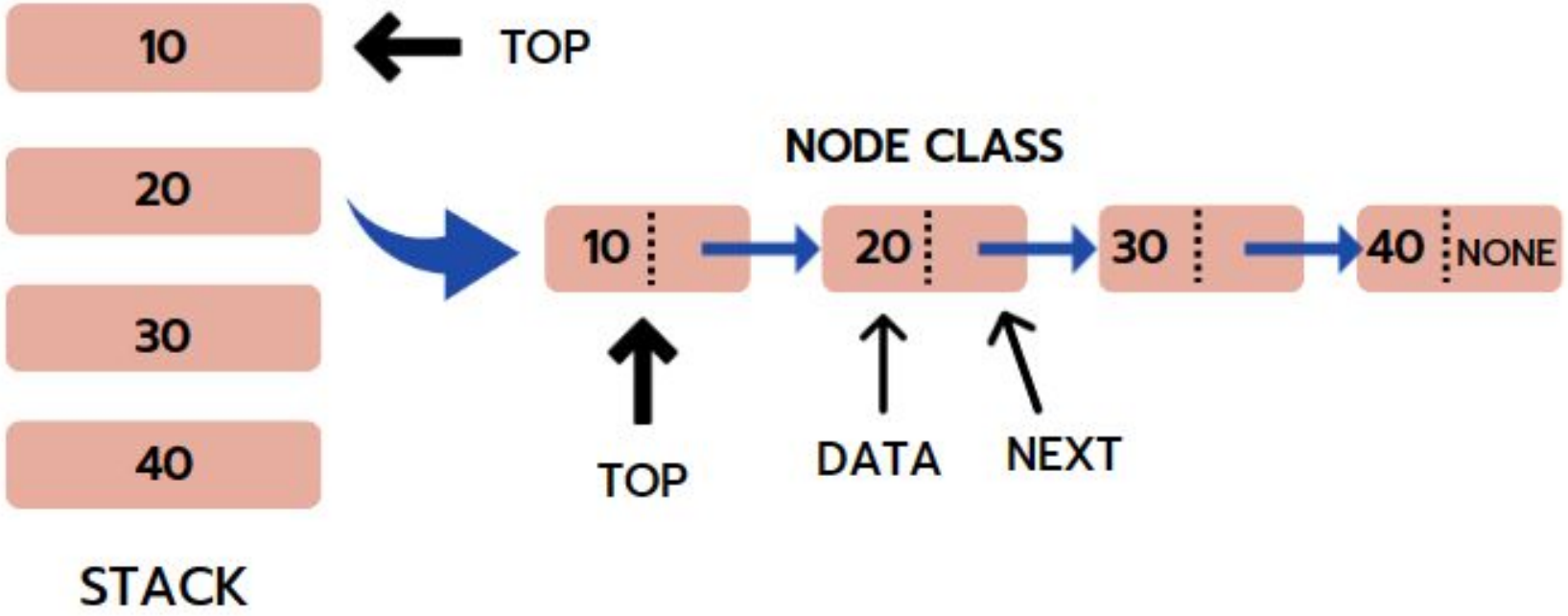
Stack Applications (BackTracking - Maze)



Stack Applications (BackTracking)



Stack Implementation (with Linked List)



Stack Implementation (Node Class)

```
class Node:
  def __init__(self,element,next):
    self.element = element
    self.next = next
```

Stack Implementation (Push Operation)

```
begin
    if stack is empty
        Create a node
        Assign the node to the top variable of the stack
    else
        Create a node and make the top of the stack its next node
        Assign the node to the top variable of the stack

end procedure
```

Stack Implementation (Push Operation)

```
class Stack:
  def __init__(self):
    self.top = None

  def push(self,elem):
    if self.top == None: #Stack is empty
      self.top = Node(elem,None)
    else:
      newNode = Node(elem,None)
      newNode.next = self.top
      self.top = newNode
```

Stack Implementation (Pop Operation)

```
begin
    if stack is empty
        return underflow exception
    else
        Save the reference of the top of the stack node in a variable
        Make the top node's next node the top of the stack
        Make the previously top node's element and next null using the saved variable

end procedure
```

Stack Implementation (Pop Operation)

```
def pop(self):
  if self.top == None:
    return None #Stack Underflow
  else:
    popped = self.top
    self.top = self.top.next
    popped.next = None
    return popped.element
```

Stack Implementation (Peek Operation)

```
begin
    if stack is empty
        return underflow exception
    else
        Return the reference of the top of the stack node
end procedure
```

Stack Implementation (Peek Operation)

```
def peek(self):
    if self.top == None:
        return None #Stack Underflow
    else:
        return self.top.element
```

Stack Problems

Count the elements of a stack

Stack Application

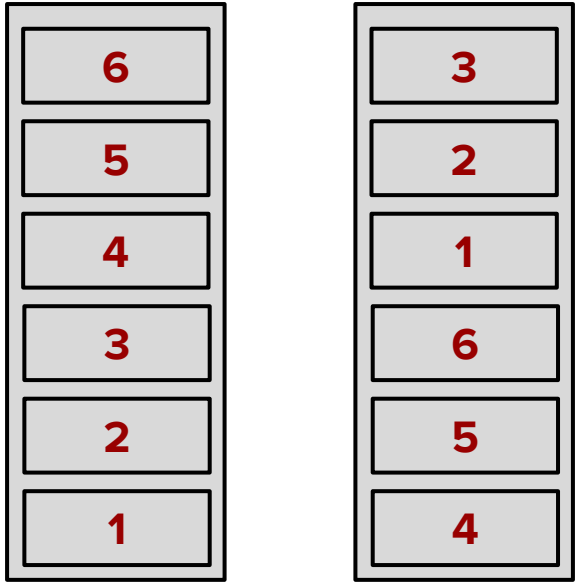
Reverse a Number using Stack

Stack Problems

Delete the third element from the stack

Stack Problems

Mid reversal of stack



Stack Implementation (with Array)