5 Stack

Last-In-First-Out (LIFO)

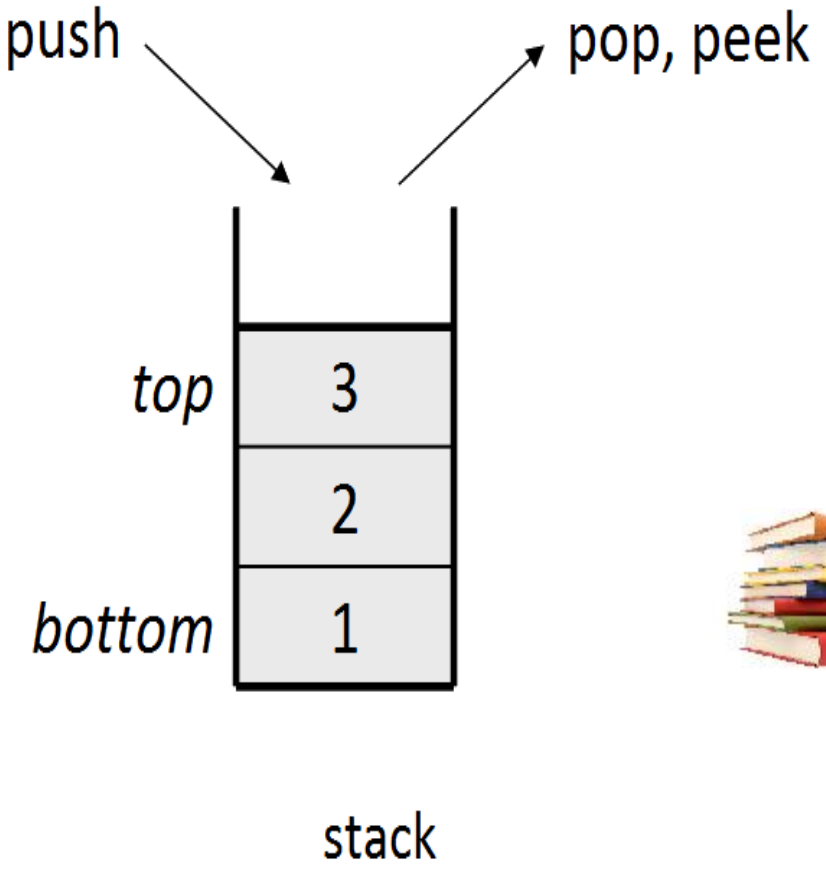
Dr. Nadia M. Mohammed

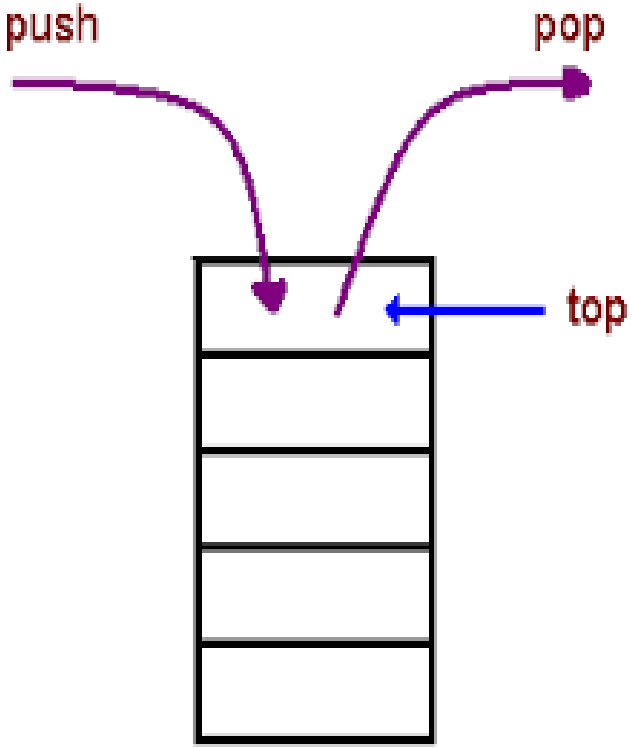
Stack

- **Stack**: A collection based on the principle of adding elements and retrieving them in the opposite order.
- –Last-In, First-Out ("LIFO")
- –Elements are stored in order of insertion.
- –Client can only add/remove/examine the last element added (the "top").

- basic stack operations:
- **push**: Add an element to the top.
- **pop**: Remove the top element.
- **peek**: Examine the top element.

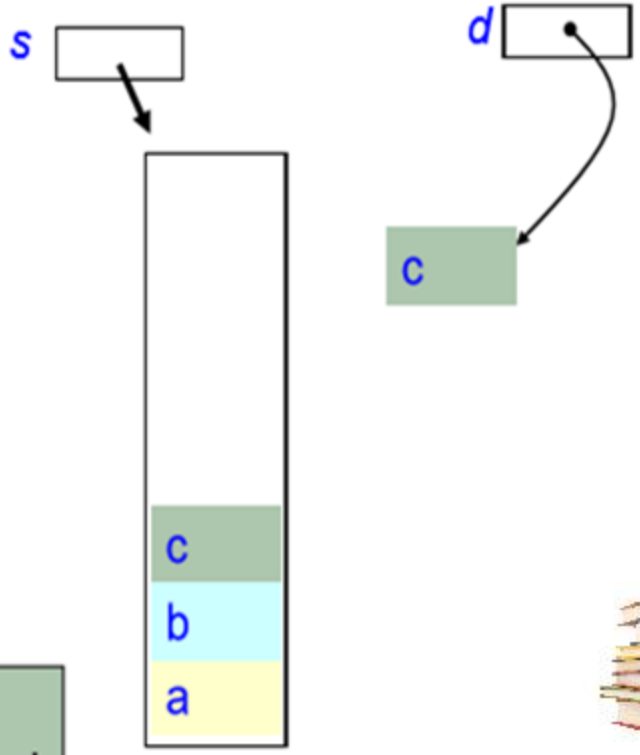
Stack





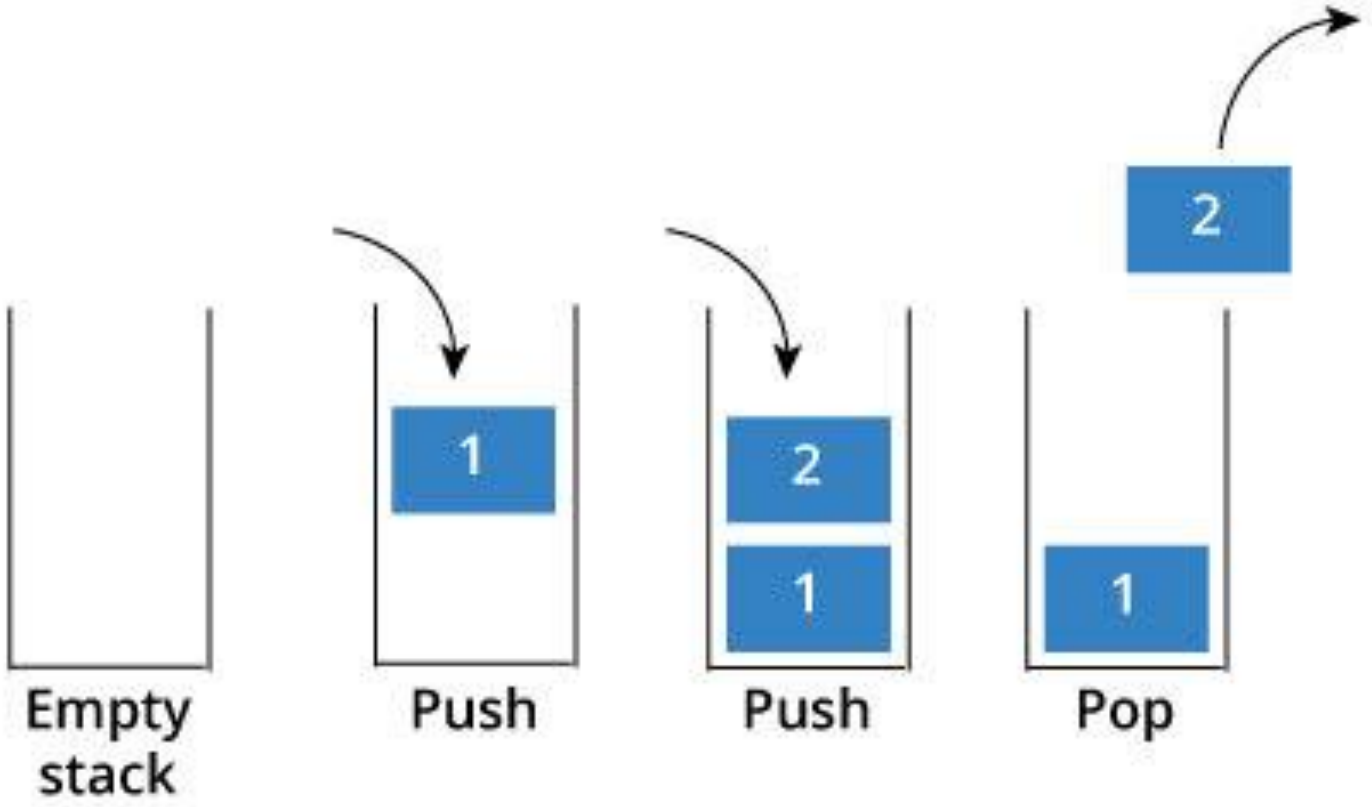
Stack: Usage

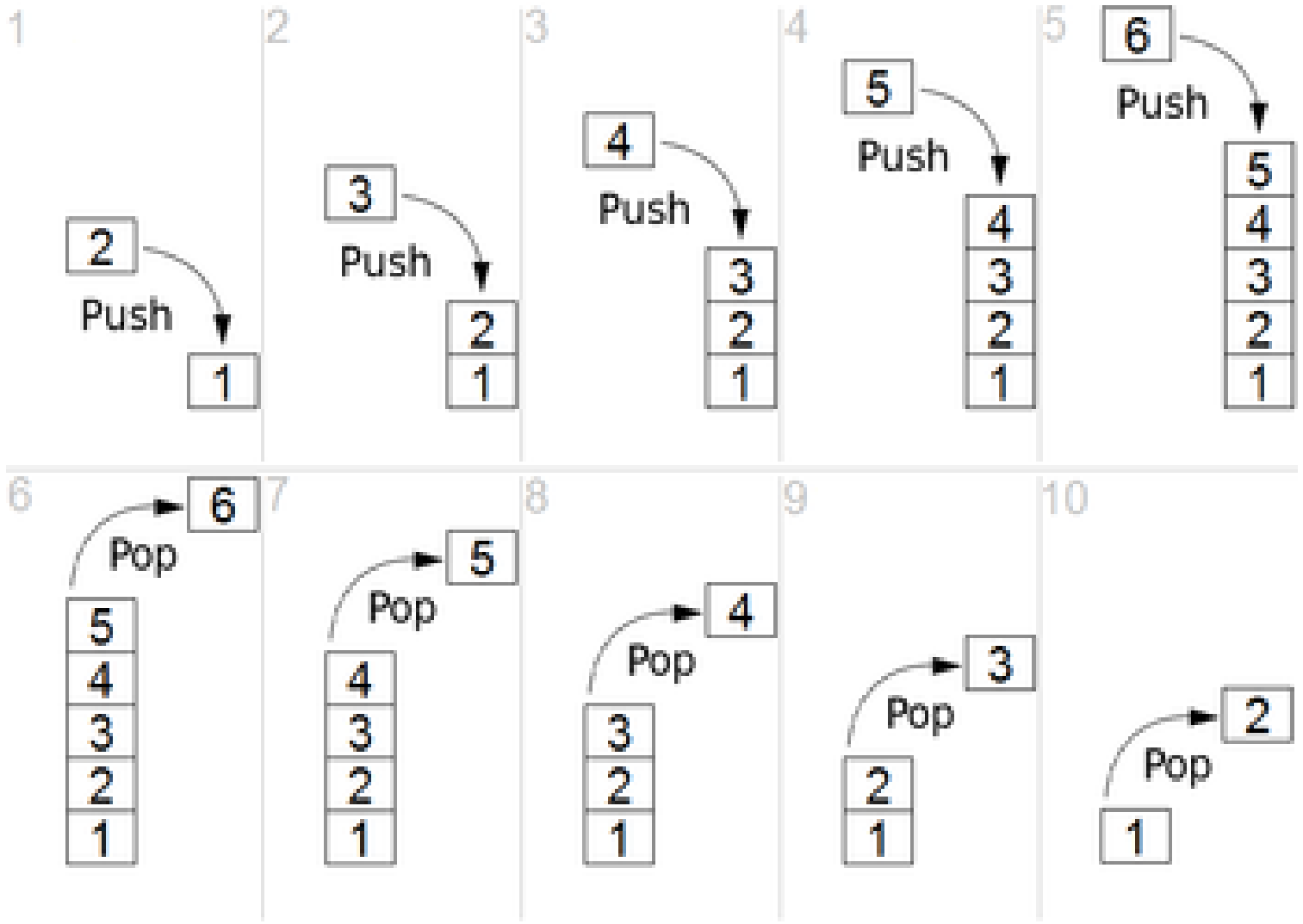
- Stack s = new Stack();
- s.push ("a");
- s.push ("b");
- s.push ("c");
- d = s.peek ();
- s.pop ();
- s.push ("e");
- s.pop ();



To be accurate, it is the references to "a", "b", "c", …, being pushed or popped.

Example




1
2
Push
1
2
3
Push
2
1
3
4
Push
3
2
1
4
5
Push
4
3
2
1
5
6
Push
5
4
3
2
1
6
6
Pop
5
4
3
2
1
7
5
Pop
4
3
2
1
8
4
Pop
3
2
1
9
3
Pop
2
1
10
2
Pop
1

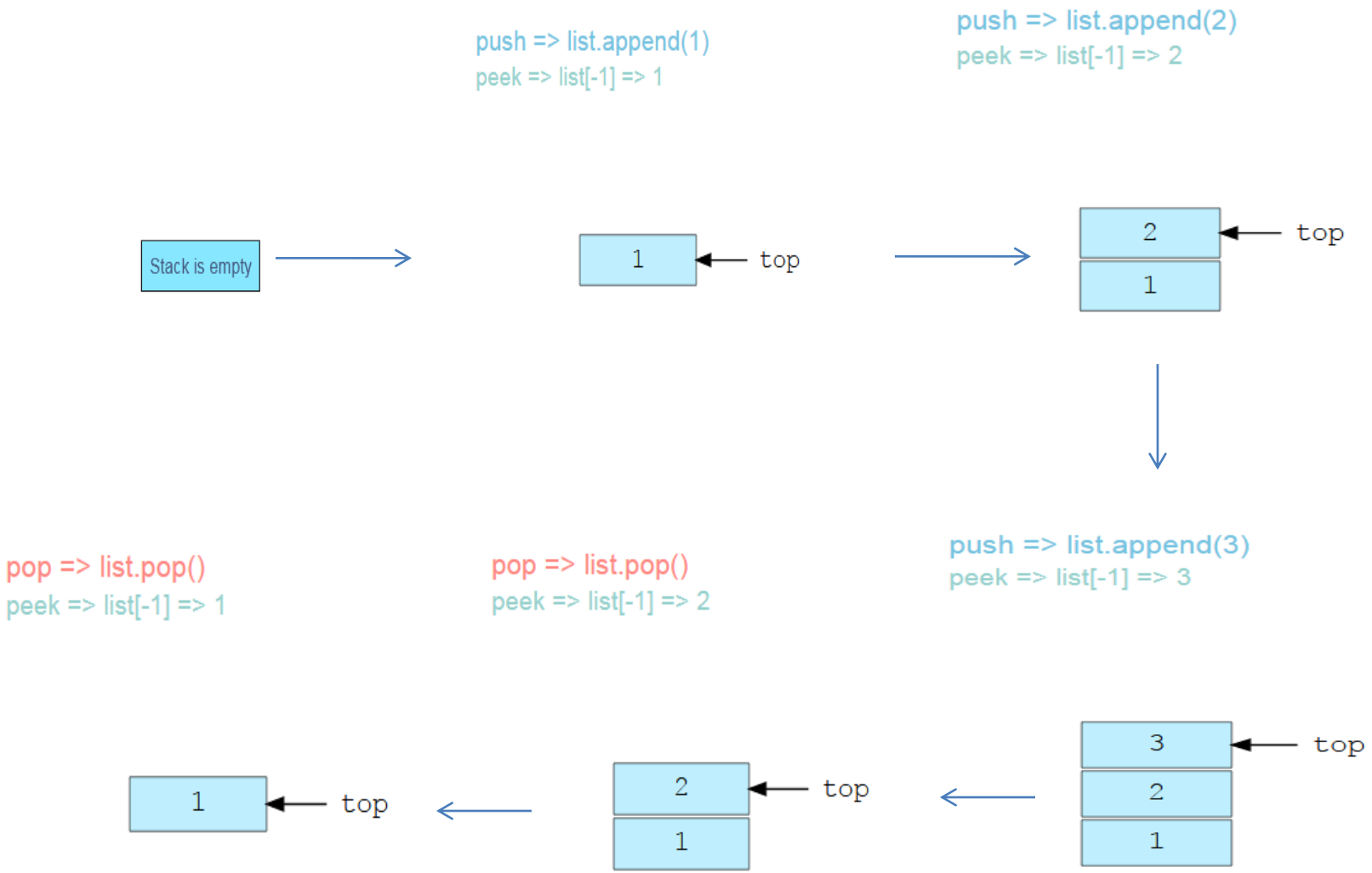
Stack

1 Stack ADT: Uses

- Calling a function
  - Before the call, the state of computation is saved on the stack so that we will know where to resume
- Recursion
- Matching parentheses
- Evaluating arithmetic expressions (e.g. a + b – c) :
  - postfix calculation
  - Infix to postfix conversion

Stack in Python

- In Python, a stack is implemented using a list object.
- To push an item in the stack, use the list function ***append*** list.append(item)
- To pop an item in the stack, use the list function ***pop*** list.pop()
- To get the top most item in the stack, write list[-1]
- The following illustration explains the concept:

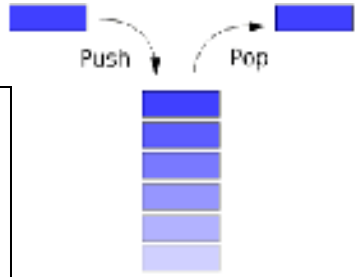

push => list.append(1)
peek => list[-1] => 1
push => list.append(2)
peek => list[-1] => 2
Stack is empty
1
top
2
top
1
push => list.append(3)
peek => list[-1] => 3
pop => list.pop()
peek => list[-1] => 2
pop => list.pop()
peek => list[-1] => 1
3
top
2
1
2
top
1
1
top

Stack Implementation in Python

```
class Stack_class


def__ ini__(self, size):
self.stack = []
self. size =  size
```

Push & Pop Methods



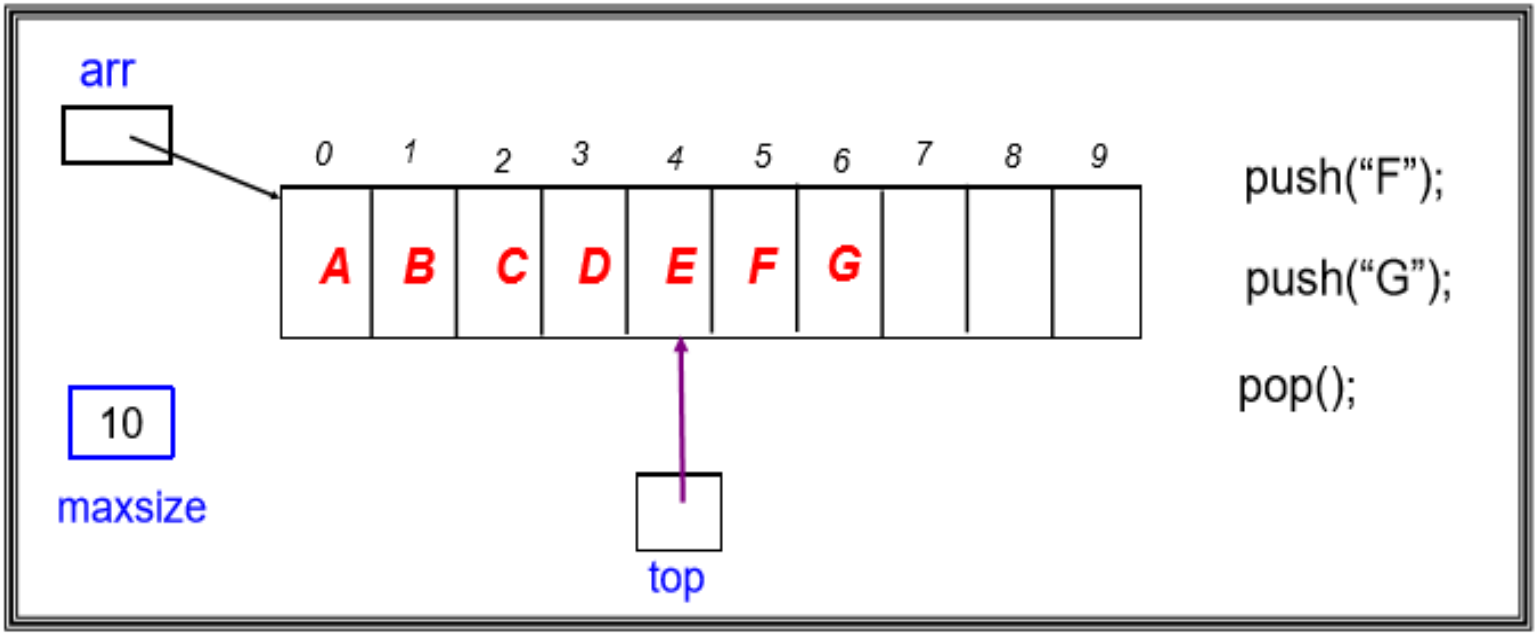

```
# The push method add an item to the stack
def push(self, item):
    if len(self.stack) == self.size:
            print("Stack is full!!")
    else:
            self.stack.append(item)
            print('stack size is: ', len(self.stack))
```

```
# The pop method delete an item from the stack
def pop(self):
    if self.stack == []:
        print("Stack is empty!")
        result = -1
    else:
        result = self.stack.pop()
    return result
```

2 Stack Implementation: Array (1/4)

- Use an Array with a top index pointer

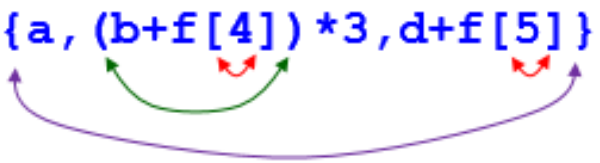
StackArr



3- Application 1: Bracket Matching (1/2)

- Ensures that pairs of brackets are properly matched

An example:



Incorrect examples:

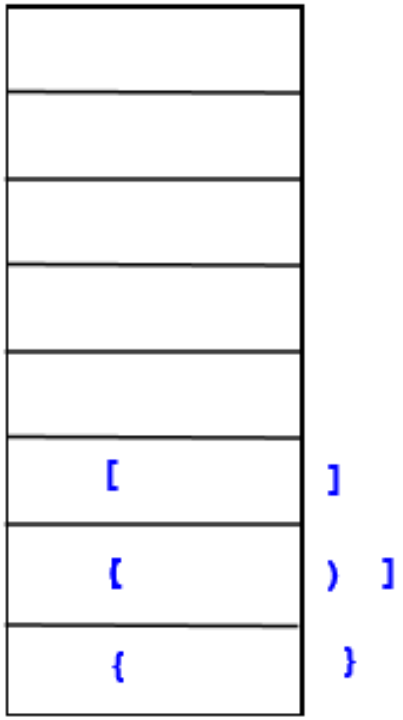
```
(..)..)        // too many close brackets
(..(..)        // too many open brackets
[..(..]..)     // mismatched brackets
```

3- Application 1: Bracket Matching (2/2)

```
create empty stack
for every char read
{
  if open bracket then
      push onto stack
  if close bracket, then
      pop from the stack
      if doesn't match or underflow then flag error
}
if stack is not empty then flag error
```

Q: What type of error does the last line test for?

A: too many closing brackets
B: too many opening brackets
C: bracket mismatch



Example

```
{ a-( b + f [ 4 ] ) * 3* d + f [ 5 ] }
```