

Stacks

Lab 12

What is a “stack”?

- A stack is an ordered collection of items where the **addition** of new items and the **removal** of existing items always takes place at the **same end**, referred to as the **top** of the stack.
 - i.e. add new elements to the top, remove existing elements from the top
- Last-in, first-out (LIFO) property
 - The last item placed on the stack will be the first item removed
- Example:
 - A stack of dishes in a cafeteria



Stack operations

- Data in stack is ordered by the insertion time: top elements is the most recently added elements, bottom element is the first element
- Access to the stack is limited to the top element and information about it's size:

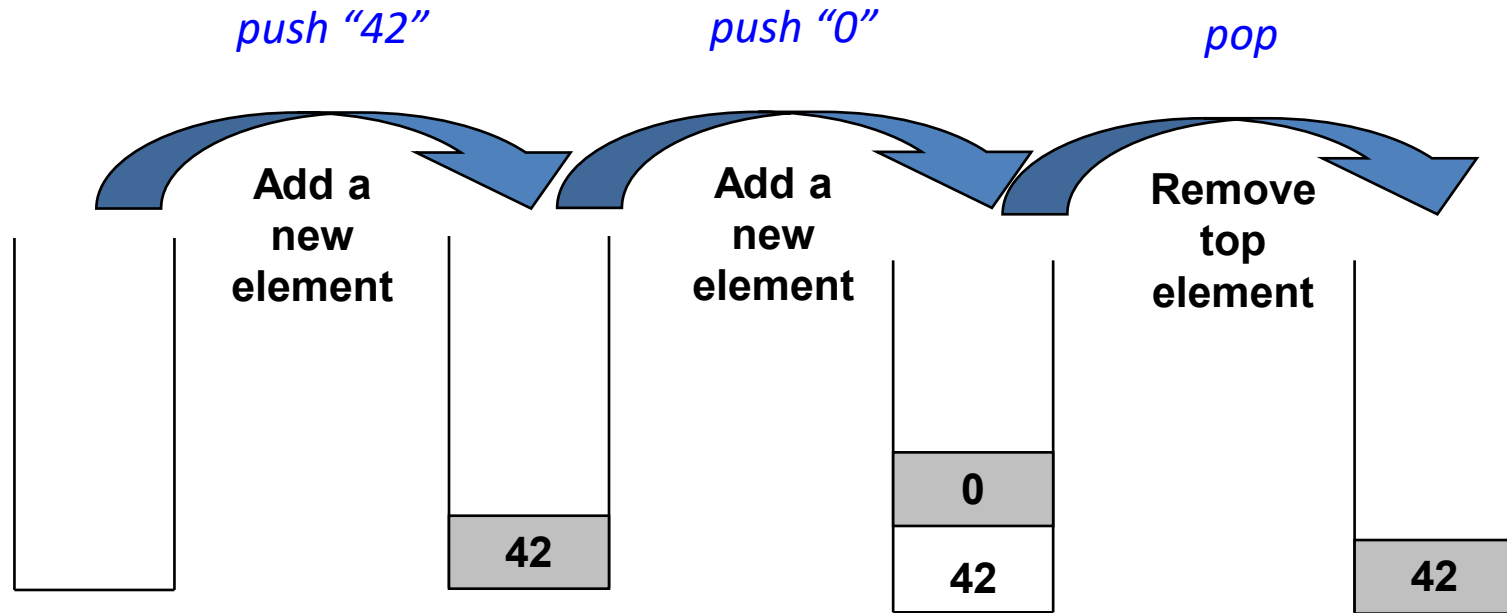


Operations:

- **create** a new empty stack (**Stack()**)
- determine whether a stack **is empty** (**is_empty()**)
- **add** a new element to the stack (**push**)
- **remove** the most recently added element from the stack (**pop**)
- **look at** (but don't remove) the most recently added element (**peek**)
- determine the **size** of a stack (how many elements) (**size**)

Example

- We add only to the **top** of a stack (called a “push”)
- We remove only from the **top** of the stack (called a “pop”)



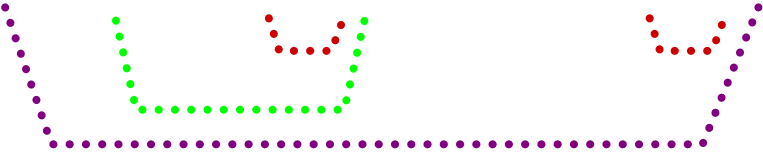
```
s = Stack()
s.push(42)
s.push(0)
print(s.peek())    // outputs 0
print(s.is_empty)  // outputs False
s.pop()
print(s.peek())    // outputs 42
print(s.size())    // outputs 1
```

Last In - First Out (LIFO)

Application 1 – Checking for balanced brackets

- We want to check whether an expression has balanced brackets

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



– Possible errors:

`(..) ..)` // too many closing brackets

`(..(..)` // too many open brackets

`[..(..)]..)` // mismatched brackets



Balanced brackets

- Algorithm:

initialise the stack to empty

for every char read

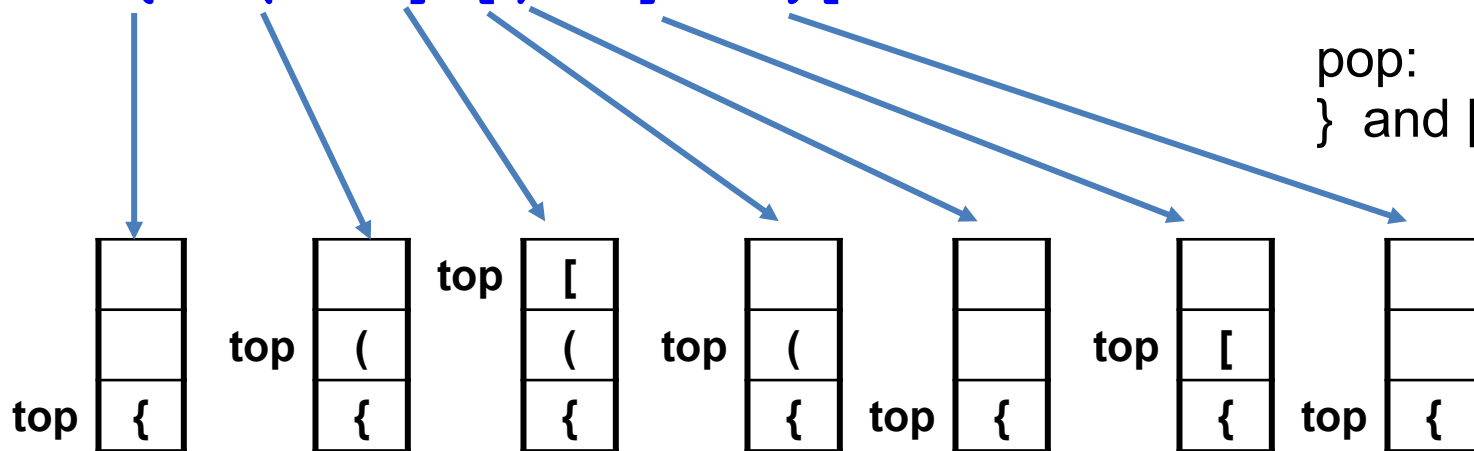
- if it is an open bracket then push onto stack
- if it is a close bracket, then
 - if the stack is empty, return ERROR
 - pop from the stack
 - if they don't match then return ERROR
- if it is a non-bracket, skip the character

if the stack is NON-EMPTY, ERROR

Balanced brackets

- Example:

$2 * \{ a * (b + f [4]) * f [5 + 4] \}$



Not
matched!!

Application 2 – Evaluating a postfix expression

- In postfix notation an operator follows its operands
- Was common in early calculators, e.g. 100+200 expressed as:

100 200 +



Early calculators evaluated postfix expressions, such as this HP 12C

Example

- Evaluate the following postfix expression:

4 4 5 3 - * + 3 -

4 4 2 * + 3 -

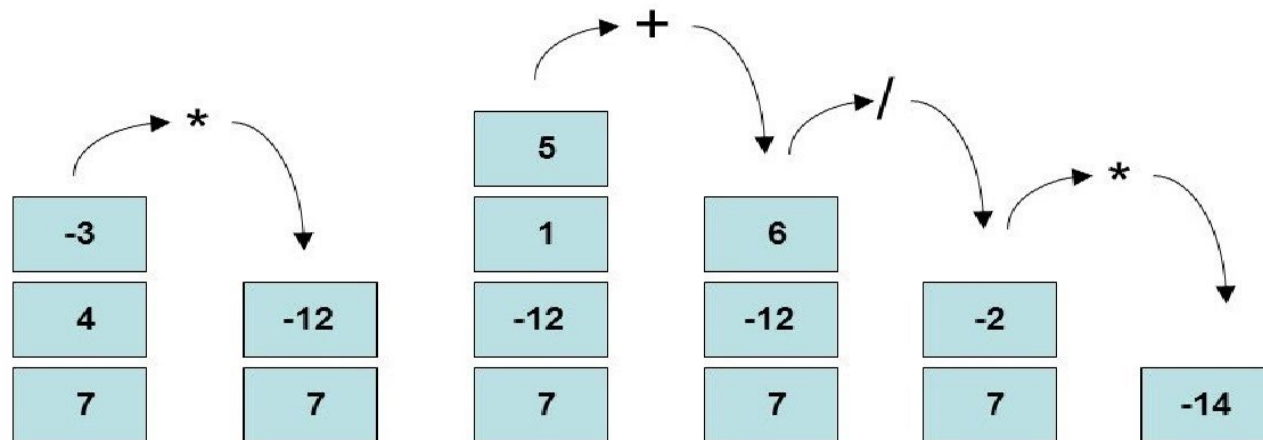
4 8 + 3 -

12 3 -

9

Evaluating Postfix Expressions

- Expression = 7 4 -3 * 1 5 + / *



Evaluating postfix expressions

- Algorithm:

When we reach an operand

- push the operand to the stack

When we reach an operator

- pop the top two operands from the stack
- perform the arithmetic using these operands
- push the resulting value back to the stack

Coderunner Tips!

- Q6 – Private fields are only private within each *class* – another Stack instance (other than self) will still be able to access its `__items` list
- Q8 – you will need some way of getting the corresponding bracket to either the open or close brackets to check if they match (a dictionary might help)
- Q8 – Think about what you need to check for at each step of the algorithm – does the stack have elements in it? Is it empty? Should it be?
- Q9 – you just need to write out the operations step by step for the expression in the question – you don't need to write any functions
- Q10 – You can edit the compute method – think about the order you pass numbers into this function for non-commutative expressions