# Stacks

# Abstract Data Type

- Model of a data structure
    - NOT the actual implementation
- Describes:
    - set of data values
    - the operations that can be performed
    - what the operations do (not how they do them...)
- Language independent
    - Helpful for approach/algorithm

# Stack

- Collection of objects
- Last in - first out (LIFO)
- Primary operations:
  - push (add to top)
  - pop (remove from top)

# Real life examples:

- Piles of items at grocery stores
- Shopping cart corral
- Stack of plates
- Pez dispenser

# When to use?

- Depends on the problem
- Not useful for all problems
- But -- Really useful for some problems
- Something to consider before you start coding
    - algorithm stage -- which ADT makes sense to use

# Application: Post-Fix Notation

- Infix:
  - what you're used to
  - <operand><operator><operand>
  - relies on order of operations and parentheses
  - Ex: 3 + 2 * 4
- Postfix:
  - <operator><operator><operand>
  - Ex: 3 2 4 * +

# Application: Post-Fix Notation

- Computer has to parse math expressions
- Postfix is easier
- How could we write parser to turn expression into code?

# Coding Applications - Others

- Reversing a string
- Back button in browser
- Undo/redo
- Balanced parentheses
- Maze solving
- Function call stack

# Impelemntations:

- Separate from the ADT
- The details of how we create the stack data structure
- Options:
    - Linked-list based
    - Array based (recall -- python lists are implemented on arrays)

# Linked List Based

- Store stack as linked list
- How could we best represent stack using linked list for operations to be as efficient as possible?
  - Can `push` be O(1)?
  - Can `pop` be O(1)?
  - Possible to implement so they are both O(1)?

# Linked List Based

- Think about what operations with linked list were O(1):
  - adding to start
  - adding to end (if there's a `tail`)
  - computing size (if `size` is stored as instance variable)
  - removing from start
- LIFO: want to remove the last one we added

# Array Based

- Store stack using an array (in python store using list)
- How can we represent to be as efficient as possible?
  - Can `push` be O(1)?
  - Can `pop` be O(1)?
  - Possible to implement so they are both O(1)?

# Array Based

- Just keep filling elements
    - Keep track of index representing `top`
    - Setting value of element is O(1)
- In Python don't need to worry about resizing (lists do this automatically)
- Conceptually though -- resizing is expensive
    - Don't resize everytime we add
    - Grab a chunk more (typically 2x) when we need to resize
    - O(n), but happens rarely