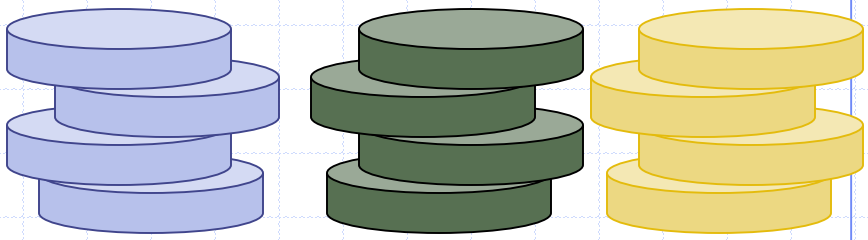
Stacks

LIFO : last in, first out
↳ Stack

FIFO : first in, first out
↳ Queue
↳ Taking turns

↳ Data Structure
↳ Stack of plates



Sareh Taebi
COP3410 – Florida Atlantic University

Abstract Data Types (ADTs)

create class of stacks

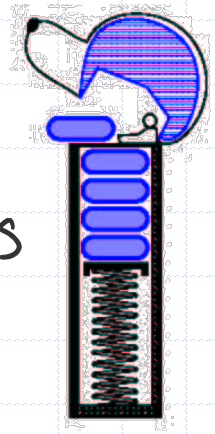
- An abstract data type (ADT) is an abstraction of a data structure
- An ADT specifies:
  - Data stored
  - Operations on the data
  - Error conditions associated with operations

Create class for stocks

- Example: ADT modeling a simple stock trading system
  - The data stored are buy/sell orders
  - The operations supported are
    - order buy(stock, shares, price)
    - order sell(stock, shares, price)
    - void cancel(order)
  - Error conditions:
    - Buy/sell a nonexistent stock
    - Cancel a nonexistent order

class should be able to provide useful error info.

The Stack ADT



- The Stack ADT stores arbitrary objects
- Insertions and deletions follow the last-in first-out (LIFO) scheme
- Think of a spring-loaded plate dispenser
- Main stack operations:
  - push(object): inserts an element
  - object pop(): removes and returns the last inserted element

create in class

- Auxiliary stack operations:
  - object top(): returns the last inserted element without removing it
  - integer len(): returns the number of elements stored
  - boolean is_empty(): indicates whether no elements are stored

Stacks:

↳ list → modify w/ operations → create stack

↳ Examples
  ↳ Back button
  ↳ undo

Applications of Stacks

- Direct applications
  - Page-visited history in a Web browser
  - Undo sequence in a text editor
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

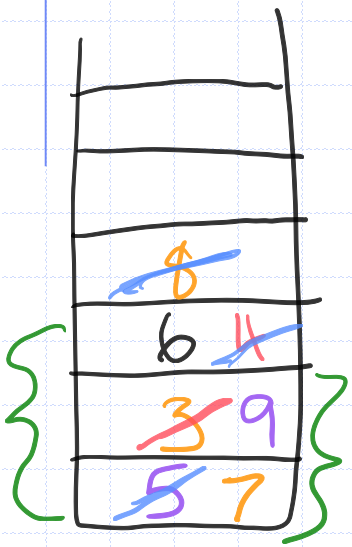
↳ Build Stack → use to build something more complex

Example

Assume S is an object that belongs to class stack

↳ Assume S is empty @ beginning

Stack starts empty



Operation	Return Value	Stack Contents
S.push(5)	–	[5]
S.push(3)	–	[5, 3]
len(S)	2	[5, 3]
S.pop( )	3	[5]
S.is_empty( )	False	[5]
S.pop( )	5	[ ]
S.is_empty( )	True	[ ]
S.pop( )	"error"	[ ]
S.push(7)	–	[7]
S.push(9)	–	[7, 9]
S.top( )	9	[7, 9]
S.push(4)	–	[7, 9, 4]
len(S)	3	[7, 9, 4]
S.pop( )	4	[7, 9]
S.push(6)	–	[7, 9, 6]
S.push(8)	–	[7, 9, 6, 8]
S.pop( )	8	[7, 9, 6]

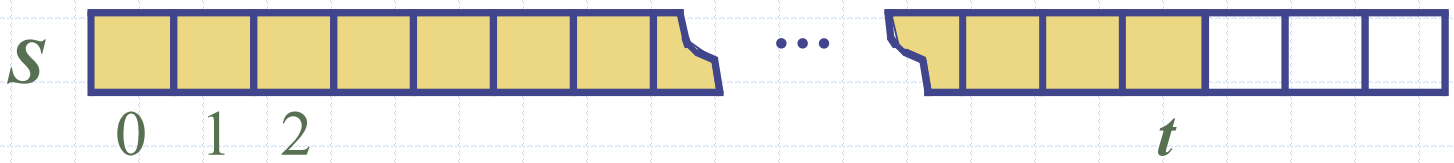
Operations on Lists

- `array.append(x)`
  - Append a new item with value x to the end of the array.
- `array.pop([i])`
  - Removes the item with the index i from the array and returns it. The optional argument defaults to -1, so that by default the last item is removed and returned.
- Although a list has operations relevant to a stack, some of it's behaviors break the abstraction of a stack

Stack Method	*Realization with Python list*
S.push(e)	L.append(e)
S.pop( )	L.pop( )
S.top( )	L[−1]
S.is_empty( )	len(L) == 0
len(S)	len(L)

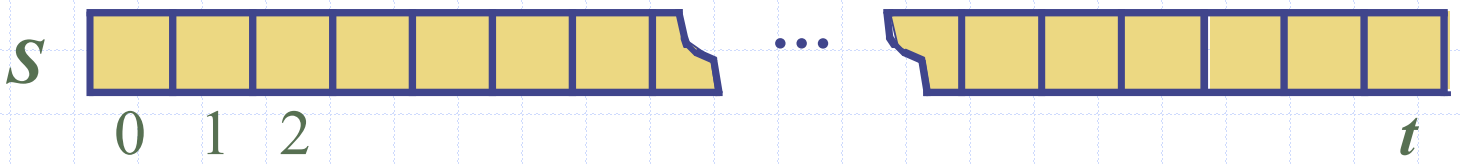
Array-based Stack

- A simple way of implementing the Stack ADT uses an array
- We add elements from left to right
- A variable keeps track of the index of the top element



Array-based Stack (cont.)

- The array storing the stack elements may become full
- A push operation will then need to grow the array and copy all the elements over.



Performance and Limitations

- Performance
  - Let n be the number of elements in the stack
  - The space used is $O(n)$
  - Each operation runs in time $O(1)$ (amortized in the case of a push)

Operation	Running Time
S.push(e)	$O(1)^*$
S.pop( )	$O(1)^*$
S.top( )	$O(1)$
S.is_empty( )	$O(1)$
len(S)	$O(1)$

*amortized

O(n)? grow/shrink

Array-based Stack in Python

```
1  class ArrayStack:
2    """LIFO Stack implementation using a Python list as underlying storage."""
3
4    def __init__(self):
5      """Create an empty stack."""
6      self._data = [ ]                       # nonpublic list instance
7
8    def __len__(self):
9      """Return the number of elements in the stack."""
10     return len(self._data)
11
12   def is_empty(self):
13     """Return True if the stack is empty."""
14     return len(self._data) == 0
15
16   def push(self, e):
17     """Add element e to the top of the stack."""
18     self._data.append(e)                   # new item stored at end of list
19
```

```
20   def top(self):
21     """Return (but do not remove) the element at the top of the stack.
22
23     Raise Empty exception if the stack is empty.
24     """
25     if self.is_empty( ):
26       raise Empty('Stack is empty')
27     return self._data[-1]                  # the last item in the list
28
29   def pop(self):
30     """Remove and return the element from the top of the stack (i.e., LIFO).
31
32     Raise Empty exception if the stack is empty.
33     """
34     if self.is_empty( ):
35       raise Empty('Stack is empty')
36     return self._data.pop( )               # remove last item from list
```

index error

Reversing Data Using Stack

- A stack can be used as a general tool to reverse a data sequence.
  - Print lines of data in reverse order

```
1  def reverse_file(filename):
2    """Overwrite given file with its contents line-by-line reversed."""
3    S = ArrayStack( )
4    original = open(filename)
5    for line in original:
6      S.push(line.rstrip('\n'))        # we will re-insert newlines when writing
7    original.close( )
8
9    # now we overwrite with contents in LIFO order
10   output = open(filename, 'w')      # reopening file overwrites original
11   while not S.is_empty( ):
12     output.write(S.pop( ) + '\n')  # re-insert newline characters
13   output.close( )
```

↳ only takes lines of data

↳ 1 string per line

↳ reverse order of lines

↳ removing \n