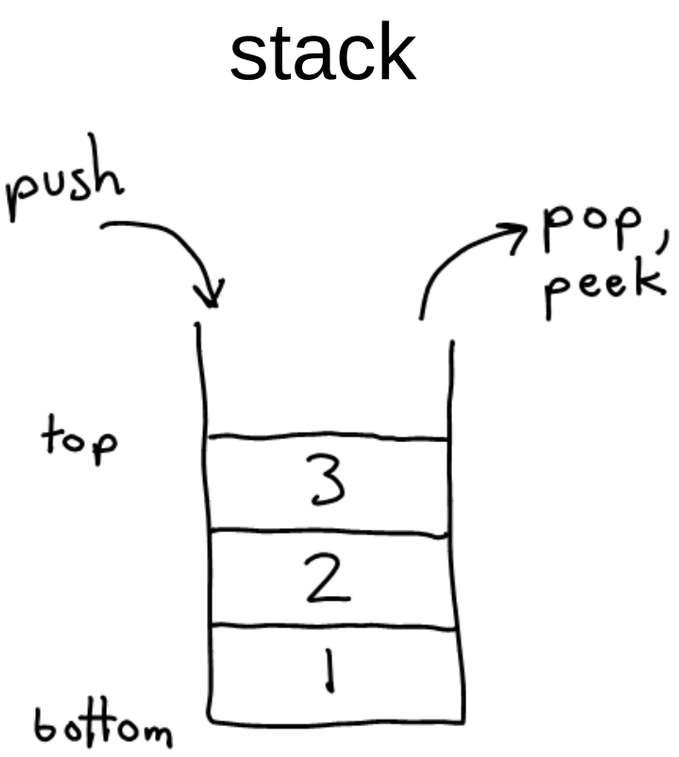
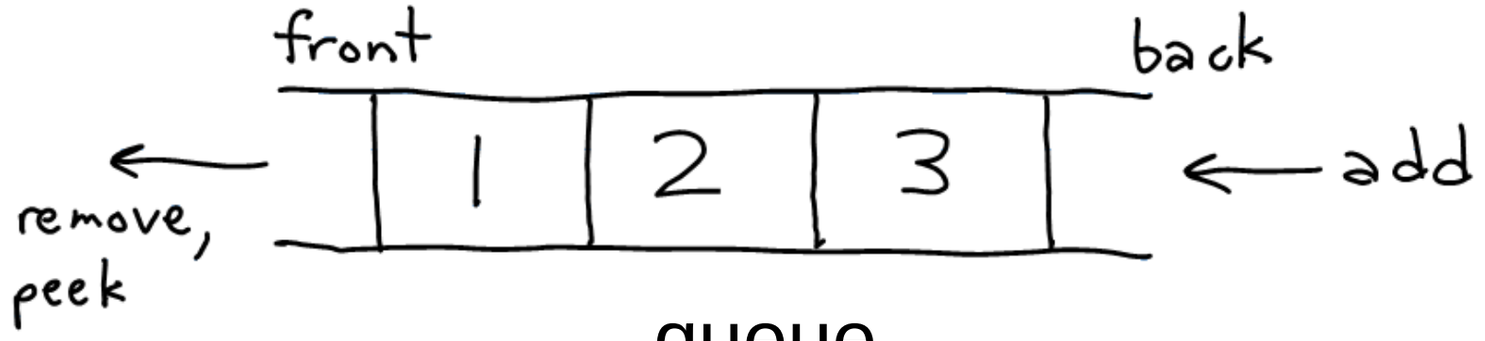
Stacks and queues

Stacks and queues

- Sometimes it is good to have a collection that is less powerful, but is optimized to perform certain operations very quickly.

- Today we will examine two specialty collections:
  - **stack**: Retrieves elements in the reverse of the order they were added.
  - **queue**: Retrieves elements in the same order they were added.



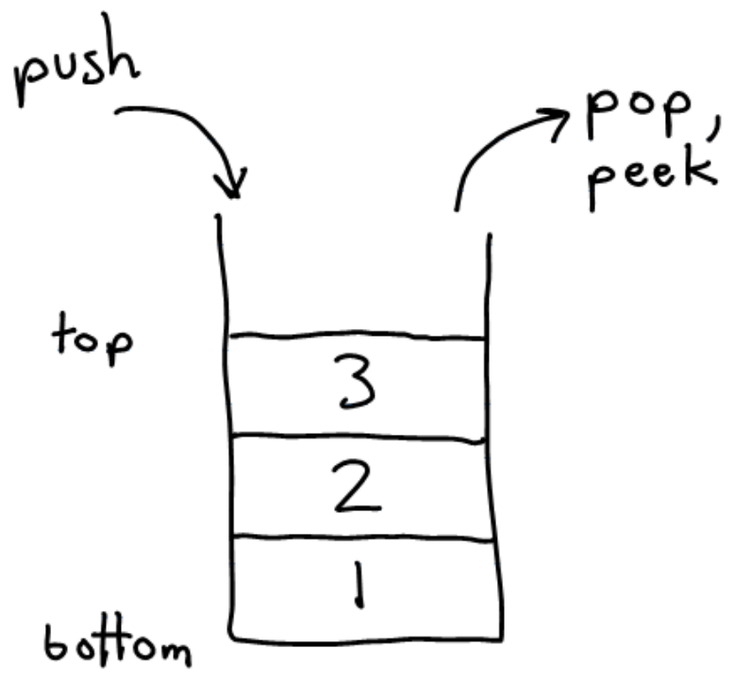


queue

Stack

- **stack**: A collection based on the principle of adding elements and retrieving them in the opposite order.
  - Last-In, First-Out ("LIFO")
  - The elements are stored in order of insertion, but we do not think of them as having indexes.
  - The 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.
  - **isEmpty**: Check whether the stack is empty.



Stack

- The Stack consists of two classes: the stack, which has a head element and the element, which has a next element.

```
class Element:
    """Element class"""
    def __init__(self, value, next):
        self.value = value
        self.next = next
```

```
class Stack:
    """Stack class"""
    def __init__(self):
        self.items = []
```

Stack: Push and Pop

- Push: To push a new item onto the stack, push appends it onto items list.

```
def push(self, item):
    """Function to push new items on to stack"""
    self.items.append(item)
```

- Pop: To pop an item off the stack, pop removes the item from the items list.

```
def pop(self):
    """Function to pop items off the stack"""
    return self.items.pop()
```

Stack: empty

- Empty: return true if the stack is empty, indicated by checking the items list.

```
def isempty(self):
    """Function to check stack empty"""
    return (self.items == [])
```

Stack: Example

- Main function using the Element and Stack class

```
if __name__ == "__main__":
    S = Stack()
    ELEMENTS = ["first", "second", "third", "fourth"]
    for e in ELEMENTS:
        S.push(e)
    RESULT = []
    while not S.isempty():
        RESULT.append(S.pop())
    assert RESULT == ["fourth", "third", "second", "first"]
```

Queues

Queue

- **queue**: Retrieves elements in the order they were added.
  - First-In, First-Out ("FIFO")
  - Elements are stored in order of insertion but don't have indexes.
  - Client can only add to the end of the queue, and can only examine/remove the front of the queue.

- basic queue operations:
  - **add** (enqueue): Add an element to the back.
  - **remove** (dequeue): Remove the front element.

Queues in Computer Science

- Operating systems:
  - queue of print jobs to send to the printer
  - queue of programs / processes to be run
  - queue of network data packets to send

- Programming:
  - modeling a line of customers or clients
  - storing a queue of computations to be performed in order

- Real world examples:
  - people on an escalator or waiting in a line
  - cars at a gas station (or on an assembly line)

Queue Operations

- The Queue is defined by the following operations:

__init__

✓ Initialize a new empty queue.

insert

✓ Add a new item to the queue.

remove

✓ Remove and return an item from the queue. The item that is returned is the first one that was added.

isempty

✓Check whether the queue is empty.

Queue Class

- The implementation of the Queue is called a **linked queue** because it is made up of linked Node objects.
- A Queue is empty when created ; thus the "head" node is None and the length is 0.

```
"""Queue Class"""
```

```
class Queue:
    """Contains the head and the length"""
    def __init__(self):
        self.length = 0
        self.head = None
```

Queue: Insert

- Inserting items into a queue is similar to inserting items in to a linked list at the tail.

Steps:

1. Create a node.
2. If the Queue is empty, set head to refer to the new node.
3. Else traverse the list to the last node and tack the new node on the end.
4. Increase the length of the list.

Queue: Insert

```
def insert(self, data):
    """Insert item at the end of list"""
    node = Node(data)  # create a Node
    node.next = None
    if self.head is None:
        # if list is empty the new node goes first
        self.head = node
    else:
        # find the last node in the list
        last = self.head
    while last.next:
        last = last.next
    # append the new node
    last.next = node
    self.length = self.length + 1
```

Queue: remove and isempty

- Removes the first item (head) from the queue and returns the removed item.
- Identical to removing items from head of Linked List.

```
def remove(self):
    """Removes head from list"""
    data = self.head.data
    self.head = self.head.next
    self.length = self.length - 1
    return data
```

- isempty checks if the queue is empty.
- Identical to the LinkedList method.

```
def isempty(self):
    """checks if the Queue is empty"""
    return (self.length == 0)
```

Python Queue Module

Queue Module

- Useful in threaded programming to exchange information among threads safely.

- The module implements three types of Queues.

✓ FIFO Queue: The first tasks added are the first to be retrieved.

✓ LIFO Queue: The most recently added entry is the first to be retrieved.

✓ Priority Queue: The entries are kept sorted and the lowest valued entry is retrieved first.

Classes

`class Queue.Queue(maxsize=0)`

- Constructor for a FIFO queue where maxsize is an integer that sets the upperbound limit on the number of items that can be placed in the queue.

Usage:

```
 >>> import Queue
 >>> myqueue = Queue.Queue(5)  # creates a queue of size 5
 >>> myqueue.put(100) # put, inserts 100 to the queue
 >>> myqueue.put(200) # put, inserts 200 to the queue
 >>> myqueue.qsize()
2
```

Classes

`class Queue.LifoQueue(maxsize=0)`

- Constructor for a LIFO queue where maxsize is an integer that sets the upperbound limit on the number of items that can be placed in the queue.

`class Queue.PriorityQueue(maxsize=0)`

- Constructor for a Priority queue where maxsize is an integer that sets the upperbound limit on the number of items that can be placed in the queue.

For methods within these classes

http://docs.python.org/2/library/queue.html

to be continued...