Week 9, Class 1

Stacks of Fun

CS 42: Principles & Practice of Computer Science

Melissa O’Neill
Fall, 2011
Recap

What are objects/classes for?
Recap (contd.)

class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy

    def __eq__(self, rhs):
        return self.x == rhs.x and self.y == rhs.y
Recap (contd.)

class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy

    def __eq__(self, rhs):
        return type(self) == type(rhs) and
                self.x == rhs.x and self.y == rhs.y
Recap (contd.)

def main():
    p1 = Point(0.0, 0.0)
    p2 = Point(0.0, 0.0)
    if p1 == (0.0, 0.0):
        print "Equally valid points!"
    else:
        print "They’re not equal!"
Suppose I want to get a packet from LA to Atlanta?
Recursion to the Rescue

def traverse(node):
    print node.name()
    for neighbor in node.connections():
        traverse(neighbor)

But I want the path from LA to Atlanta!

How much space?

No recursion allowed?
Stacks!
Stacks

What are the fundamental operations?

Does it matter how the stack works?

Rocks stack, and stacks rock!
Stack Interface

The key operations

- push
- pop
- empty

How can we use the stack interface to eliminate recursion in our Internet example?

Interface + Behavior (but not saying how) =

- **Abstract Data Type**
More Realistic
The Homework — A Demo

Let’s make it about spam!!??!
Spamsweeper!

A Boardcell
(boolean) mined
(boolean) exposed
(int) neighborcount

mined = False
neighborcount = 2

mined = True
neighborcount = 3

A Gameboard
**Automatic Exposure**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

- **neighbor count**: 3
- **exposed?**: true, false

If player chooses this, how much should be exposed?
Automatic Exposure

neighbor count

exposed?
true
false

6 7 8 1
5
4 3 2

0 0 1 1
0 0 1
1 1 2 1
3 2 2 0

SPAM

SPAM

SPAM
Stacks, Implementation Ideas?
S = Stack()
S.push("Sleep")
S.push("IWP")
S.push("CS42")
S.push("Dinner!")
S.pop()
S = Stack()
S.push("Sleep")
S.push("IWP")
S.push("CS42")
S.push("Dinner!")
S.pop()
class StackCell(object):
    def __init__(self, data):
        self.data = data
        self.next = None
    def __str__(self):
        return str(self.data)

class Stack(object):
    def __init__(self):
        self.top = None
class StackCell(object):
    def __init__(self, data):
        self.data = data
        self.next = None
    def __str__(self):
        return str(self.data)

class Stack(object):
    def __init__(self):
        self.top = None
Complete the linked-list implementation of a Stack.

```python
class Stack(object):
    def pop(self):
        write pop()

    def push(self, data):
        write push(Object data)

    def empty(self):
        write empty()
Interwebs, Revisited

LA to DC?
How will switching to a Queue change our search procedure?
Queues, Implementation
Queues

Three ways in the assignment

• Using Python lists
• Using Python’s deque class (provided by us!)
• Hand-implemented with a QCell class.
Queue vs Stack

Can we just reuse our design for the stack class?
Stacks, in code

class StackCell(object):
    def __init__(self, data):
        self.data = data
        self.next = None
    def __str__(self):
        return str(self.data)

class Stack(object):
    def __init__(self):
        self.top = None
Queues, in code

```python
class QCell(object):
    def __init__(self, data):
        self.data = data
        self.next = None
    def __str__(self):
        return str(self.data)

class Queue(object):
    def __init__(self):
        self.front = None
```
Queues, in code

class QCell(object):
    def __init__(self, data):
        self.data = data
        self.next = None
    def __str__(self):
        return str(self.data)

class Queue(object):
    def __init__(self):
        self.front = None
        self.back  = None
"Q"uiz

Draw the results of each of these lines of code...

Bert = Queue()
Bert.enqueue("U") (1)
Bert.enqueue("C") (2)
Bert.dequeue() (3)
Bert.dequeue() (4)

Here is a picture of a QCell...
class MazeCell(object):
    ''' A class to support Maze containing one cell of the maze '''
    def __init__(self, row, col, contents):
        self.row = row
        self.col = col
        self.contents = contents
        self.visited = False
        self.parent = None
    def __str__(self):
        return "[" + str(self.row) + "," +
            str(self.col) + "," + 
            self.contents + "]"

'S' = Start Spam Seeking  'D' = Delectible Dinner Destination  ' ' = Open Space
'*' = Wall!
Depth-first search (DFS)

Create an empty Stack
mark starting MazeCell as visited
push starting MazeCell onto our Stack

while ( the stack’s not empty )
current = pop the stack
for each of current’s neighbors
if ( it’s not visited or wall )
mark it (neighbor) as visited
set its parent to current MazeCell
push it onto the stack

'S' = Start Spam Seeking
'D' = Delectible Dinner Destination
A maze built by details?

```python
class Maze(object):
    # USE THESE RATHER THAN MAGIC VALUES!
    WALL = '*';
    EMPTY = ' ';  
    START = 'S';
    DESTINATION = 'D';
    FOOTSTEP = 'o';

    def __init__(self, filename):
        self.maze = None
        self.rows = 0
        self.columns = 0
        self.loadMazeFromFile(filename)

    def findMazeCell(self, charToFind):
        for r in range(self.rows):
            for c in range(self.columns):
                if self.maze[r][c].contents == charToFind:
                    return self.maze[r][c]

    def BFS(self, start, dest):
        # To be implemented by you!
```

Hw8, part 3: implement breadth-first search