Wrapping up Recursion
Announcements

• Lab 10 (recursion) posted
  - Due Saturday at midnight
Today’s plan

• Catalogue several different forms of recursion
• Recursion gotchas
• Recursive binary search
Recursing over ints

• Typical base case: \( n == 0 \) or \( n == 1 \)

• Typical general case: use \( fn(n-1) \) in solution for \( fn(n) \)

• Practice sheet: #1, #4
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
Recursing over lists

• Typical base case: \( \text{len}(L) == 0 \) or \( \text{len}(L) == 1 \)

• Typical general case: Use \( L[0] \) and \( \text{fn}(L[1:]) \) to solve \( \text{fn}(L) \)

• Practice sheet: #2, #3, #9, #10
def recursiveLinearSearch(x, L):
    if len(L) == 0:
        return False
    else:
        if L[0] == x:
            return True
        else:
            recursiveLinearSearch(x, L[1:])
Recursing over lists

def recursiveLinearSearch(x, L):
    if len(L) == 0:
        return False
    else:
        if L[0] == x:
            return True
        else:
            return recursiveLinearSearch(x, L[1:])
Recursing over strings

- Typical base case: `s == ''`
- Typical general case: use `s[0]` and `fn(s[1:])` to solve `fn(s)`
- Practice sheet: #6, #7, #8
Recursing over strings

def countLetter(s, l):
    if s == "":
        return 0
    elif s[0] == l:
        return 1 + countLetter(s[1:], l)
    else:
        return countLetter(s[1:], l)
How to approach recursion

1. Identify what we’re recursing over. For this example, let’s imagine it’s a string and our function is called foo(s).

2. Solve the base case, foo(“”).

3. Imagine you have a working version of foo. Ask yourself what foo(s[1:]) would return. Combine it with s[0] to figure out the return value for foo(s).

4. Don’t forget the return statements
Multiple general cases

• Often within the general case, we want to examine \( n, L[0], s[0], \) etc. in an if statement.
def countHeads(n):
    if n == 0:
        return 0
    else:
        flip = choice(['heads', 'tails'])
        if flip == 'heads':
            return 1 + countHeads(n-1)
        else:
            return countHeads(n)
Recursive graphics

- **Fractals** are self-repeating images. You can zoom in on a fractal and see a sub-image that closely resembles the original image.

- They appear in nature: trees, lightning, river tributaries…

- When we generate a fractal using computer graphics, it is natural to use recursion.
Multiple recursive calls

- Solve the problem with solutions to multiple smaller sub-problems:
  - Merge sort
  - Fractals
- Exponential growth
Returning new lists

def reverse(L):
    if len(L) == 1:
        return L
    else:
        return reverse(L[1:]) + L[:1]
Modifying a list in place

• Do the recursion over an integer that represents the index.

• The list and the index are both parameters.

• Use a wrapper function to avoid passing in the initial index.
Modifying lists in-place

```python
def squareOddIndicesH(L, index):
    if index == len(L):
        return
    else:
        if index % 2 == 1:
            L[index] = L[index]**2
        squareOddIndicesH(L, index+1)

def squareOddIndices(L):
    squareOddIndicesH(L, 0)
```
Recursion gotchas

• If you forget the base case, the function will continue calling itself indefinitely, until the stack reaches its maximum size. This also happens if your sub-problem is the same size as your original problem, e.g. foo(n) instead of foo(n-1).

  - `RuntimeError: maximum recursion depth exceeded`

• With functions that are called for their return value, it is easy to forget the ‘return’
Recursive binary search

- Pass ‘lo’ and ‘hi’ as additional parameters.
- Update the range of indices when you make the recursive call.
- Recursion makes sense here because binary search is repeatedly breaking the search down into a binary search on a smaller list.
def binarySearchH(x, L, lo, hi):
    if lo > hi:
        return False
    else:
        mid = (lo+hi)/2
        if L[mid] == x:
            return True
        elif L[mid] < x:
            return binarySearchH(x, L, mid+1, hi)
        else:
            return binarySearchH(x, L, lo, mid-1)

def binarySearch(x, L):
    return binarySearchH(x, L, 0, len(L)-1)
See you Wednesday!