Recursive Functions

CS 8: Introduction to Computer Science
Lecture #15

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

- 3 MORE CLASSES TO GO! ☺

M	T	W	Th	F
5/29	5/30 LECTURE 14	5/31 LAB 7 issued	**6/1** **LECTURE 15** HW8 issued	**6/2** ***Review session*** LAB 7 due
6/5	**6/6** **LECTURE 16** HW7 due	**6/7** Work on your Project2 in lab	**6/8** **REVIEW** HW8 due Project2 due	**6/9** ***Review session*** *Last day of Spring classes at UCSB*
6/12	**6/13**	**6/14**	**6/15** **FINAL EXAM at 4PM**	**6/16**

IMPORTANT NOTE!

NO assignment (hwk, lab, project) will be accepted
to be turned in **AFTER** the **LAST** lecture/class on
THURSDAY 6/8!

("late" assignments policy will not apply –
we simply will not accept them)

Review Sessions

- Review sessions next week with T.A. Sourav
  - See announcements on Piazza

- In-class review for the final exam next Thursday, 6/8.

Lecture Overview

- Recursion
- Classes and Objects

Starting chapter 9 (just covering through p. 315 though)

Recursive Functions

- **Recursive: (adj.) Repeating unto itself**
- **A recursive function contains a call to itself**

- When breaking a task into subtasks, it may be that the subtask is a smaller example of the same task

- For example: **Searching a large list for all occurrences**
  - Task can be divided into searching the 1st, then 2nd halves of array
  - Searching each half is a smaller version of searching the whole array
    - So, each half is then divided into 1st and 2nd halves, etc…

Simple Example: The Factorial Function

Recall factorials:

$2! = 1 * 2$, $3! = 1 * 2 * 3$, $4! = 1 * 2 * 3 * 4$, …

N! = 1 * 2 * … * (N-1) * N

There's some repetition here… We could think of it as a loop

(how would you write that?)

```
def factorial(n):
    f = 1
    for k in range(1, n+1):
        f = f * k
    return f
```

Recursive Functions

```
def factorial(n):                # return n! = n(n-1)(n-2)…1
    if n > 1:                    # recursive step: call self for n-1
        return n * factorial(n-1)
    else:                        # note: is the else really necessary??
        return 1                     # base case: stop recursion if n < 2
```

- Recursive functions should know **when to stop**
- There must be (at least) one ***base case***, and the recursive step must converge on a base case
    - Otherwise you get "infinite recursion"

Another Example: Mathematical Series

- Popular example: Fibonacci Series

F(n) = 1, 1, 2, 3, 5, 8, 13, …, *F(n-1) + F(n-2)*

- Again, there's some repetition here…
  We could think of it as a loop also
  (we did this for Project 1!!!)

- Or we could think of it as a recursive function!

Fibonacci Recursion

- What is/are the BASE CASE(S)?
- What is the recursive formula?

```
def fibo(n):
    if n == 0:
        return 1

    if n == 1:
        return 1

    return fibo(n-1) + fibo(n-2)
```

DEMO TIME!

File called:
recursive.py
now online

Other Examples

- Leibniz formula for π

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}.$$

DEMO TIME!

```
def series(n):
    if n == 0:
        return 4
    return 4* ( ((-1)**n)/(2*n + 1) ) + series(n-1)
```

Recursive Drawing Examples

- Listing 9.2 (also in recursive.py) – uses **drawSquare** function from chapter 2

```
def drawSquare(aTurtle,side):
    for i in range(4):
        aTurtle.forward(side)
        aTurtle.right(90)
```

```
def nestedBox(aTurtle,side):
    if side >= 1:                          # recursive step
        drawSquare(aTurtle, side)
        nestedBox(aTurtle, side - 5)
    # base case: do nothing (side will be < 1 and too small to draw)
```

Other Recursive Drawing Examples

- Other examples in the **recursive.py** file
  - Draw tick marks on a ruler

- Examples from the textbook and in other files
  - Listing 9.4 – draw nested triangles
  - In file **triangles.py**
  - Note demo introduces command line argument too

  - Listing 9.3 (and exercises 9.11-9.13) – draw tree
  - In file **trees.py**

</LECTURE>