Recursive Functions in Python

CS 8: Introduction to Computer Science, Winter 2018
Lecture #14

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

- Homework #8 is **DUE on Wed. (3/14)**
- Lab #6 due **Wed 3/14**
- Remaining on the calendar… ***This supersedes anything on the syllabus***

DATE	TOPIC	ASSIGNED	DUE
Mon. 3/5	File I/O ; Formats for Outputs	Hw #7 Lab #5	Hw #6 Lab #5
Wed. 3/7	Digital Images ; While-Loops		
Mon. 3/12	Recursive Functions	Hw #8 Lab #6	Hw #7, Hw #8 Lab #6 Proj #2
Wed. 3/14	Review for the Final Exam		

Administrative

- Turn in Homework #7
- Homework #8 is **DUE on WEDNESDAY (3/14)**
  - That's in 2 days…

- Lab #6 due **Wed 3/14**
- Project #2 due Fri 3/16

Preparation for the Final Exam

- We will have a review session in class on Wednesday
- I have put up Practice Questions for you
  - With answers!

Lecture Overview

Recursive Functions

See Ch. 9 (thru p. 315) in textbook

How *Do* Functions Work?

- Consider these 3 functions and tell me: what is **demo(-4)** ?

```
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x/2)

def g(x):
    return -1 * x
```

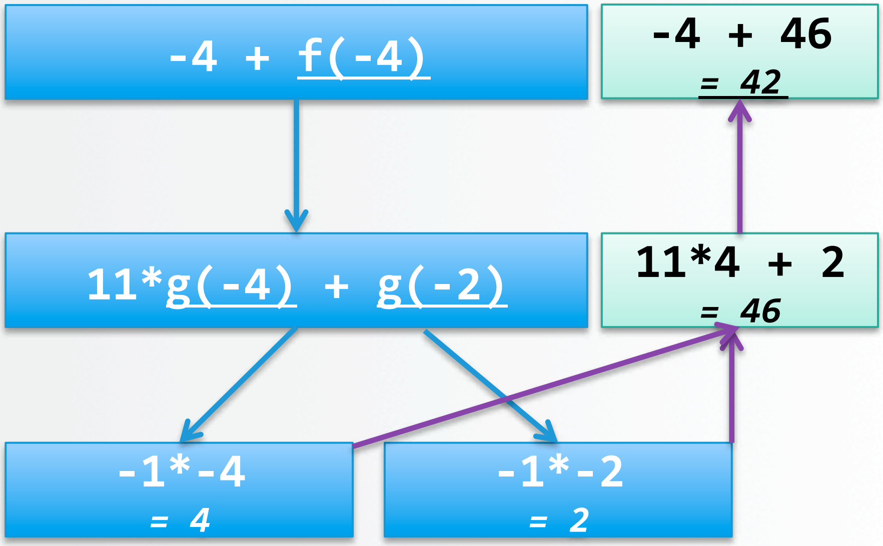
How *Do* Functions Work?

- Consider these 3 functions and tell me: what is **demo(-4)** ?

```
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x/2)

def g(x):
    return -1 * x
```



-4 + f(-4)

-4 + 46
= 42

11*g(-4) + g(-2)

11*4 + 2
= 46

-1*-4
= 4

-1*-2
= 2

What Keeps Track of All of This?!?

- Ans: **The Stack**

(1) keeps separate variables for each function call…

(2) remembers where to send results back to…

The stack is a special part of your computer's memory.

The compiler usually spells-out how the stack must be used with functions.

A child couldn't sleep,
so her mother told a story about a little frog,
who couldn't sleep,
so the frog's mother told a story about a little bear,
who couldn't sleep,
so bear's mother told a story about a little weasel
...who fell asleep.
...and the little bear fell asleep;
...and the little frog fell asleep;
...and the child fell asleep.

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

- Just like functions-calling-functions,
  recursive functions make use of the stack

Simple Example: Factorial Function

Recall factorials:

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

$$N! = 1 * 2 * \ldots * (N-1) * N$$

There's some repetition here… We could think of it as a loop
(how would you write that?)

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

Consider the Following…

```
def fac(N):
    return N * fac(N-1)   # Yes, this is legal!
```

What happens when `fac(4)` is called?

A. It returns the correct result (i.e. 24)

B. The execution never stops

C. It produces a return value that is incorrect

Just ‘Cause It’s Legal, Doesn’t Mean It’s Good Code!!!

```
def fac(N):
    return N * fac(N-1)    # Yes, this is legal!
```

This goes on and on into an infinite loop!

Q:Why?

A: It’s missing a “base case” (a.k.a a “stopping case”)

Q2: What’s a good “base case” here?

Base Case

```
def fac(N):
    if N <= 1:
        return 1
    else:
        return N * fac(N-1)
```

- 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***"

Under the Hood…

```
def fac(N):
    if N <= 1:
        return 1
    else:
        return N * fac(N-1)
```

>>> fac(1)

I get:

1 # easy-peasy

>>> fac(5) → 5 * fac(4)

→ 5 * (4 * fac(3))

→ 5 * (4 * (3 * fac(2)))

→ 5 * (4 * (3 * (2 * fac(1))))

→ 5 * (4 * (3 * (2 * 1))) = <u>120</u>

Every step, the new values are put into the STACK and kept track of by the computer

Exercise

- What does MyRecFun(3) do?

```
def MyRecFun(n):
    if n == 0:
        return 2
    else:
        return 2*MyRecFun(n-1)
```

Another Example: Mathematical Series

- Popular example: Fibonacci Series

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

- There's some repetition here…

  We could think of it as a loop also

- 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 0
   if n == 1:
      return 1
   else: # is this else necessary?
      return fibo(n-1) + fibo(n-2)
```

DEMO TIME!

File called:
recursive.py
now online

Example: Linear Number Series

- **Mathematical Linear Series**

Example:

S(n) = 0, 1, 4, 13, 40, … for n = 0 to ∞

What's the pattern?

Linear series: $S_{n+1} = \mathbf{A}.S_n + \mathbf{B}$ where A & B are constants

In the example above: A = 3 and B = 1

What is our base-case? What is our recursion?

Example: Linear Number Series

- **Mathematical Linear Series**

Example:

S(n) = 0, 1, 4, 13, 40, … for $n = 0$ to ∞

Linear series: $\underbrace{S_{n+1} = 3.S_n + 1}_{\textit{recursion}}$ and $\underbrace{S_0 = 0}_{\textit{base case}}$

```
def series(n):
    if n <= 0:
        return 0
    return (3*series(n-1) + 1)
```

Example: Reversing a String

- **Recursion in strings**

Example: Reverse a string

Given a string (e.g. “**hello**”), you would need to return “**olleh**”

What does a recursive algorithm look like? What is my base-case?

Hints: if `s = ‘hello’`, what is `s[1:]` ?

```
def revStr(s):
    if len(s) == 0:
        return s
    return revStr(s[1:]) + s[0]
```

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_draw.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**

DEMO TIME!

YOUR TO-DOs

□ Finish up all your assignments and by their due dates!

✓ Homework #8	**by Wednesday in class**
✓ Lab #6	**by Wednesday at 11:59 PM**
✓ Project #2	**by Friday at 11:59 PM**

□ Final Exam review in class on Wednesday

✓ Bring your questions! ☺

</LECTURE>