Recursive Functions

CS 8: Introduction to Computer Science, Winter 2019
Lecture #15

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

Administrative

- HW 7 due today!
- Left to-do:
  - HW 8 for Wednesday
  - Lab 7 for today (by midnite)
  - Project for Thursday

FINAL IS COMING!

- Material: ***Everything***!
- Homework, Labs, Lectures, Textbook
- **Wednesday, 3/20** in this classroom
- **Starts at 8:00 AM **SHARP****
- ***Bring your UCSB IDs and arrive 10-15 minutes early***
- Duration: **3 hours long** (but really designed for 1.5 – 2 hours)
- Closed book: no calculators, no phones, no computers
- Allowed: 1 sheet (***single***-sided) of written notes
  - Must be no bigger than 8.5” x 11”
  - **You have to turn it in with the exam**
- **You will write your answers on the exam sheet itself.**

STUDY GUIDE NOW ONLINE!

Lecture Outline

- Recursive Functions
- Exercises

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
```

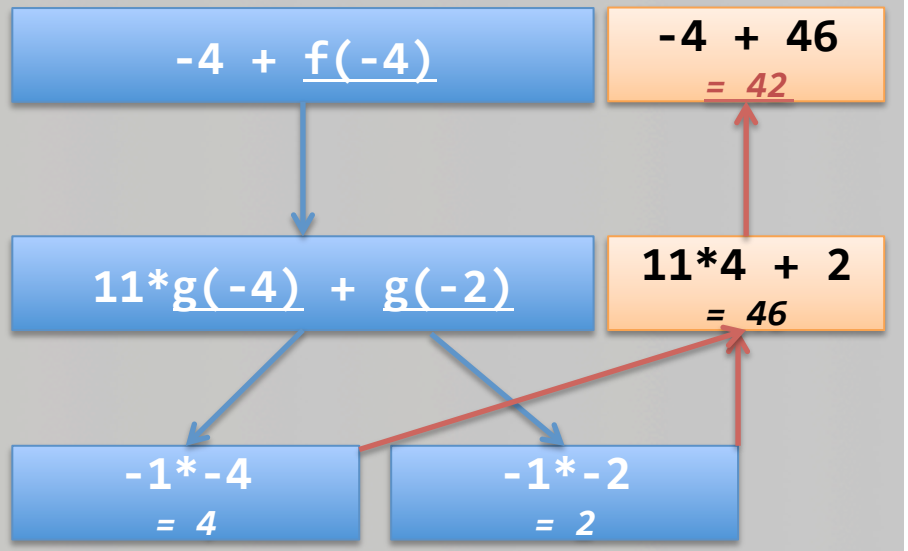
How *Do* Functions Work?

- 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
```



What Keeps Track of All of This?!?

Ans: **The Computer Memory 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 * … * (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!
print(fac(4))
```

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

A. It blows up! Does not compute! Does not compute!

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

C. The execution never stops (i.e. infinite loop) ANS

D. It produces a return value but that value is incorrect (i.e. not 24)

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 an "***infinite recursion***"

Under the Hood…

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

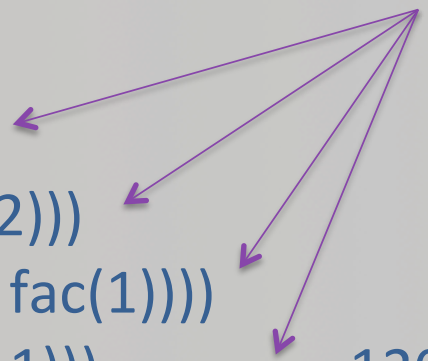
>>> fac(1)

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))) = 120



*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?

DEMO TIME!

```
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)
```

YOUR TO-DOs

- ❑ **HW8** (due on **Wednesday, 3/13**)
- ❑ **Lab7** (due on **Monday, 3/11**)
- ❑ **Project Assignment** (due on **Thursday, 3/14**)

</LECTURE>