UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Recursion II

1

Announcements

- Practice Midterm Setup will be out tonight or tomorrow
  - Practice with the exam tool (exam.cs61a.org)
  - Practice with creating a Zoom recording
  - You will submit the exam online, then submit a link to a recording.
  - Will be worth 2 of 56 points.
  - Should only take you 10 minutes. ☺
- Recursion is on the midterm, but not super advanced recursion.
- Reminder: No Live lecture on 10/7. Take a break or prep. :)

2

Computing In The News

Amsterdam and Helsinki launch algorithm registries to bring transparency to public deployments of AI

Venture Beat, Khari Johnson September 28, 2020

Amsterdam and Helsinki today launched AI registries to detail how each city government uses algorithms to deliver services, some of the first major cities in the world to do so. An AI Register for each city was introduced in beta today as part of the Next Generation Internet Policy Summit, organized in part by the European Commission and the city of Amsterdam. The Amsterdam registry currently features a handful of algorithms, but it will be extended to include all algorithms following the collection of feedback at the virtual conference to lay out a European vision of the future of the internet, according to a city official.

3

Learning Objectives

- Write a recursive function
- Understand the base case and a recursive case
- Work with printing in recursive functions
- Use recursion on lists

4

Why Recursion?

- Recursive structures exist (sometimes hidden) in nature and therefore in data!
- It's mentally and sometimes computationally more efficient to process recursive structures using recursion.
- Sometimes, the recursive definition is easier to understand or write, even if it is computationally slower.

5

The Recursive Process

- Recursive solutions involve two major parts:
  - Base case(s), the problem is simple enough to be solved directly
  - Recursive case(s). A recursive case has three components:
    - Divide the problem into one or more simpler or smaller parts
    - Invoke the function (recursively) on each part, and
    - Combine the solutions of the parts into a solution for the problem.

6

In words

- The sum of no numbers is zero
- The sum of 1^2 through n^2 is the
  - sum of 1^2 through $(n-1)^2$
  - plus n^2

```
def sum_of_squares(n):
    if n < 1:
        return 0
    else:
        return sum_of_squares(n-1) + n**2
```


7

How does it work?

- Each recursive call gets its own local variables
  - Just like any other function call
- Computes its result (possibly using additional calls)
  - Just like any other function call
- Returns its result and returns control to its caller
  - Just like any other function call
- The function that is called happens to be itself
  - Called on a simpler problem
  - Eventually stops on the simple base case

8

Recursion With Lists

```
def first(s):
    """Return the first element in a sequence."""
    return s[0]
def rest(s):
    """Return all elements in a sequence after the first"""
    return s[1:]

def min_r(s):
    """Return minimum value in a sequence."""
    if Base Case
    else:
        Recursive Case
```

- Recursion over sequence length, rather than number magnitude

9

Local variables

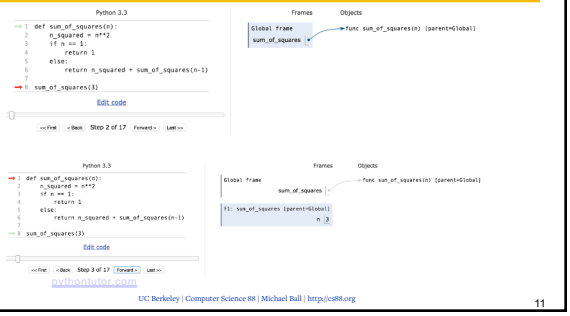
- Each call has its own "frame" of local variables
- Let's see the environment diagrams

```
def sum_of_squares(n):
    n_squared = n**2
    if n < 1:
        return 0
    else:
        return n_squared + sum_of_squares(n-1)
```

https://goo.gl/CiFaUJ

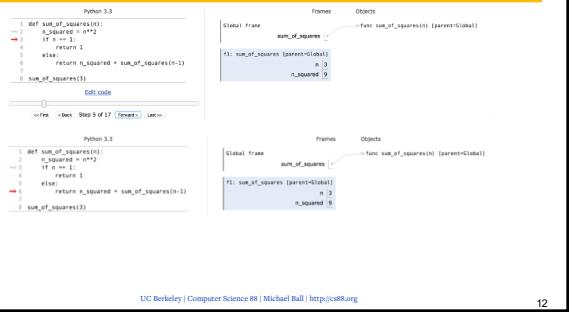
10

Environments Example

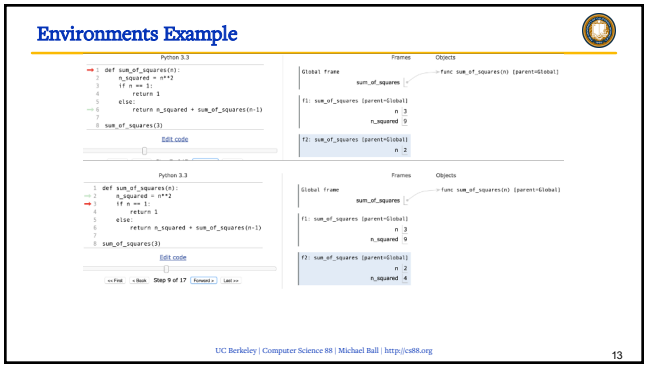


11

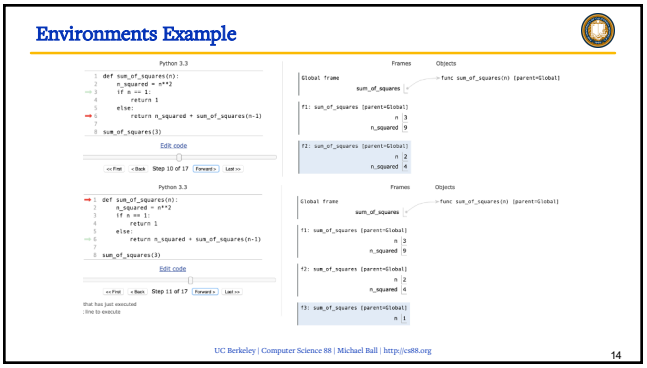
Environments Example



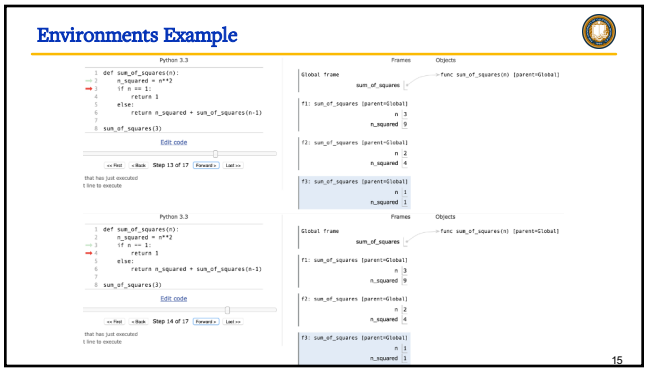
12



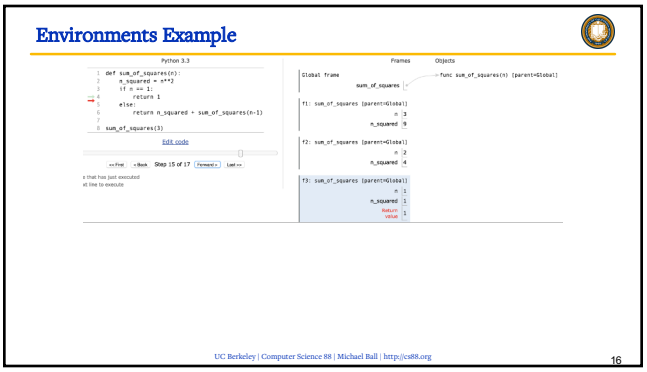
13



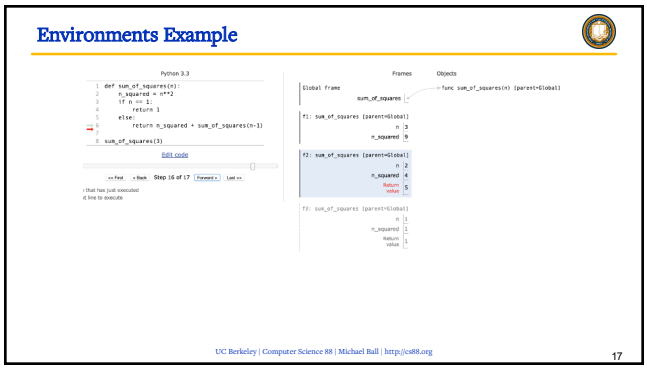
14



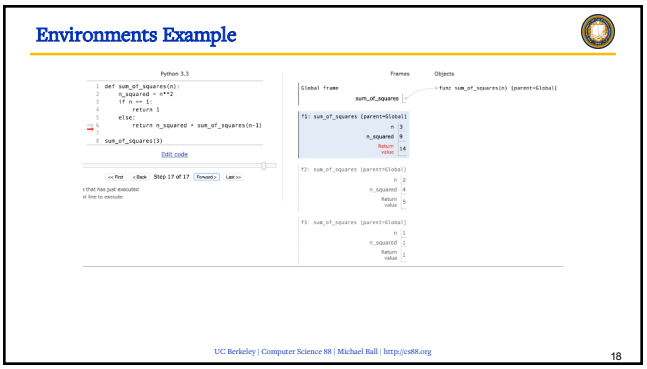
15



16



17



18

Extra Examples (We'll cover later, too.)

19

Tree Recursion

- Break the problem into multiple smaller sub-problems, and Solve them recursively

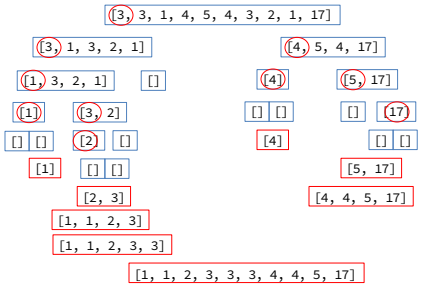
```
def split(x, s):
    return [i for i in s if i <= x], [i for i in s if i > x]

def quicksort(s):
    """Sort a sequence - split it by the first element,
    sort both parts and put them back together."""
    if not s:
        return []
    else:
        pivot = s[0]
        smaller, bigger = split(pivot, s[1:])
        return quicksort(smaller) + [pivot] + quicksort(bigger)

>>> qsort([3,3,1,4,5,4,3,2,1,17])
[1, 1, 2, 3, 3, 3, 4, 4, 5, 17]
```


20

QuickSort Example



21