

Recursion

David E. Culler CS8 – Computational Structures in Data Science http://inst.eecs.berkeley.edu/~cs88

Lecture 5

Feb 22, 2016

Computational Concepts Toolbox

- Data type: values, literals, operations,
 - e.g., int, float, string
- Expressions, Call expression
- Variables
- Assignment Statement
- Sequences: tuple, list
 _ indexing
- Data structures
- Tuple assignment
- Call Expressions
- Function Definition Statement
 - - Conditional Statement

- Iteration:
 - data-driven (list comprehension)
 - control-driven (for statement)
 - while statement
- Higher Order Functions
 - Functions as Values
 - Functions with functions as argument
 - Assignment of function values
- Higher order function patterns
 - Map, Filter, Reduce
- Function factories create and return functions

UCB CS88 Sp16 L4



Administrative Issues

- Windows conda install resolved ???
- Project 1 due Wednesday
- Tourney play to take place in stages
 - Early rounds prior to Monday 2/29
 - Final rounds in lab !!!
 - PreSeason games anyone?
- Midterm Friday 3/4 5-7 pm in 405 Soda
 - Review next week
- HW 03 out today



re.cur.sion

/riˈkərZHən/ 🐠

noun MATHEMATICS LINGUISTICS

the repeated application of a recursive procedure or definition.

 a recursive definition. plural noun: recursions

re·cur·sive

/riˈkərsiv/ Đ

adjective

characterized by recurrence or repetition, in particular

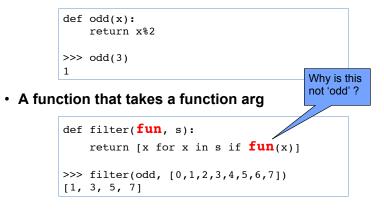
- MATHEMATICS LINGUISTICS
 relating to or involving the repeated application of a rule, definition, or procedure to
 successive results.
- COMPUTING
- relating to or involving a program or routine of which a part requires the application of the whole, so that its explicit interpretation requires in general many successive executions.
- · Recursive function calls itself, directly or indirectly

3

Review: Higher Order Functions



- Functions that operate on functions
- A function





UCB CS88 Sp16 L4

2/22/16

UCB CS88 Sp16 L4





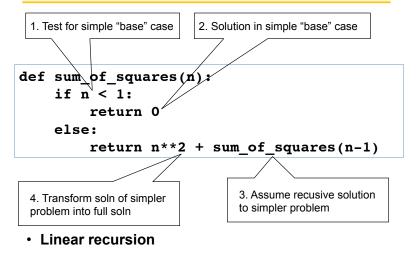
5

· What does this function do?

def split_fun(p, s):
 """ Returns <you fill this in>."""
 return [i for i in s if p(i)], [i for i in s if not p(i)]

>>> split_fun(leq_maker(3), [0,1,2,3,4,5,6])
([0, 1, 2, 3], [4, 5, 6])







A function that returns (makes) a function

def leq_maker(c):
 def leq(val):
 return val <= c
 return leq</pre>

>>> leq_maker(3)
<function leq_maker.<locals>.leq at 0x1019d8c80>

>>> leq_maker(3)(4)
False

```
>>> filter(leq_maker(3), [0,1,2,3,4,5,6,7])
[0, 1, 2, 3]
>>>
```

2/22/16

7

2/22/16

6

In words

- The sum of no numbers is zero
- The sum of 1² through n² is n² plus the sum of 1² through (n-1)²

Why does it work



sum_of_squares(3)

#	<pre>sum_of_squares(3) =></pre>	3**2 +	<pre>sum_of_squares(2)</pre>
#	=> 3**2	+ 2**2	+ sum_of_squares(1)
#	=> 3**2	+ 2**2	+ 1**2 + sum_of_squares(0)
#	=> 3**2	+ 2**2	+ 1 * * 2 + 0 = 14

def s	um_of_squares(n):
i	f n < 1:
	return O
е	lse:
	return n**2 + sum_of_squares(n-1)

2/22/16

UCB CS88 Sp16 L4

2/22/16

9

UCB CS88 Sp16 L4

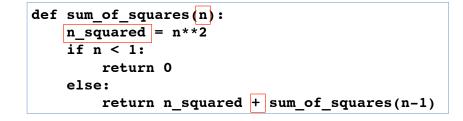


10

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 bottoms out on the simple base case
- · Reason about correctness "by induction"
 - Solve a base case
 - Assuming a solution to a smaller problem, extend it

Local variables

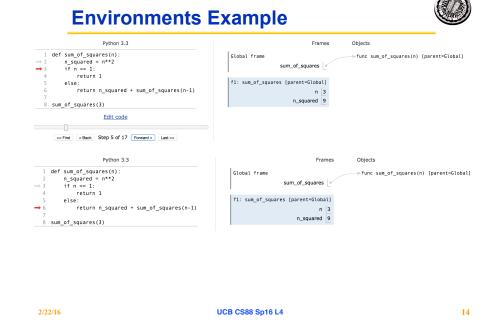


- · Each call has its own "frame" of local variables
- What about globals?
- · Let's see the environment diagrams

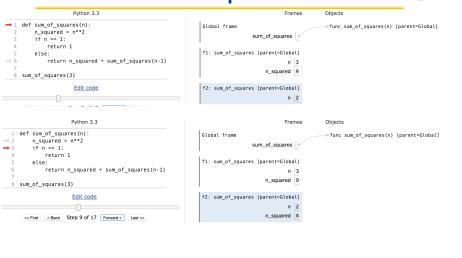
Environments Example







Environments Example





Python 3.3	Frames	Objects
1 def sum_of_squares(n): 2	Global frame sum_of_squares f1: sum_of_squares [parent=Global]	>func sum_of_squares(n) [parent=Global]
<pre>→ 6 return n_squared + sum_of_squares(n-1) 7 8 sum_of_squares(3)</pre>	n 3 n_squared 9	
Edit code	f2: sum_of_squares [parent=Global] n 2 n_squared 4	
Python 3.3	Frames	Objects
→ 1 def sum_of_squares(n): 2 n_squared = n**2 3 if n == 1: 4 return 1	Global frame sum_of_squares =	──>func sum_of_squares(n) [parent=Global]
5 else: → 6 return n_squared + sum_of_squares(n-1) 7 8 sum_of_squares(3)	f1: sum_of_squares [parent=Global] n 3 n_squared 9	
Edit code	f2: sum_of_squares [parent=Global] n 2 n_squared 4	
that has just executed : : line to execute	f3: sum_of_squares [parent=Global] n 1	

Environments Example



2 n_squared = n**2 3 if n = 1: 4 return 1 5 else: 6 return n_squared + sum_of_squares(n-1) 7 B 8 sum_of_squares(3) Edit code 6 return 1 6 return 1 7 B Bit code (expret < Back Step 13 of 17 Forwards) Last>) Int has just executed n=guared 1 Python 3.3 I def sum_of_squares(n): 1 A guared = n**2 3 if n = 1: 4 return 1 5 else: I def sum_of_squares(n): 1 Global frame Sum_of_squares [parent=Global] 1: sum_of_squares [parent=Global]		_
<pre>2 n_squared = n*2 3 if n = 1: 4 return 1 5 else: 6 return n_squared + sum_of_squares(n-1) 7 sum_of_squares(3) Edit code (cFrm_ clack Step 13 of 17 Forward> Lant> the to execute f1: sum_of_squares [parent=Global] f2: sum_of_squares [parent=Global] f3: sum_of_squares [parent=Global] f3: sum_of_squared [f3: sum_of_squares [parent=Global] f3: sum_of_squared [f3: sum_of_squares [parent=Global] f3: sum_of_squares [parent=Global] f3: sum_of_squares [parent=Global] f4: sum_of_squares [parent=Global] f5: sum_of_squares [parent=Global] f3: sum_of_squares [parent=Global] f4: sum_of_squares [parent=Global] f5: sum_of_squares [parent=Global</pre>	Python 3.3	Frames Objects
<pre>2 n_squared = n*2 3 if n = 1: 4 return 1 5 else: 6 return n_squared + sum_of_squares(n-1) 7 8 sum_of_squares(3) Edit code ceFret < Back Step 13 of 17 Forward> Lat> at has just executed n to execute 1 def sum_of_squares (n): 2 n_squared 1 (ceFret < Back Step 13 of 17 Forward> Lat> at has just executed n to execute 1 def sum_of_squares (n): 2 n_squared + sum_of_squares (n-1) 7 return n_squared + sum_of_squares (parent=Global) 7 return n_squared 4 7 return n_squared [7 return n_squared + sum_of_squares (parent=Global) 7 return n_squared function (parent 1 sum_of_squares (parent=Global) 1 return n_squared [7 return n_squared function (parent 1 return n_squared + sum_of_squares (parent=Global) 1 return n_squared [7 return n_squared [7 return n_squared function (parent 1 return n_squared + sum_of_squares (parent=Global) 1 return n_squared [7 return n_squared [7 return n_squares [7 return n</pre>	1 def sum_of_squares(n):	Global framefunc sum of squares(n) [paren
<pre>sum_of_squares [" sum_of_squares [" sum_of_</pre>	2 n squared = n**2	
4 return 1 5 else: 6 return n_squared + sum_of_squares(n-1) 7 sum_of_squares(3) Edit code f1: sum_of_squares [parent=Global] ide sum_of_squares(3) n ide sum_of_squares parent=Global] ide sum_of_squares n ide sum_of_squares(n:1) n idef sum_of_squares(n:1) n idef sum_of_squares(n:1) n if n = 1: return 1 idef sum_of_squares(3) f1: sum_of_squares [parent=Global] if n = 1: f1: sum_of_squares [parent=Global] if i return 1 sum_of_squares [parent=Global] if i return 1 sum_of_squares [parent=Global] if i sum_of_squares(3) n if i: sum_of_squares(3) n if i: sum_of_squares [parent=Global] n		sum_of_squares
5 else: 6 return n_squared + sum_of_squares(n-1) 7 sum_of_squares(3) fl: sum_of_squares [parent=Global] n_squared fl: sum_of_squares [parent=Global] return n_squared fl: sum_of_squares [parent=Global] return n_squared fl: sum_of_squares [parent=Global] return n_squared 1 fl: sum_of_squares [parent=Global] return n_squared 1 Python 3.3 Frames Objects fl: sum_of_squares [parent=Global] n_squared 1 return n_squared 1 return n_squared 1 fl: sum_of_squares [parent=Global] n_squared = **2 a fin == 1: fl: sum_of_squares [parent=Global] n_squared = **12 fl: sum_of_squares [parent=Global] n_squared = **2 fl: sum_of_squares [parent=Global] n_squared [s fl: sum_of_squares [parent=Global] <		
6 return n_squared + sum_of_squares(n-1) n 3 7 8 sum_of_squares(3) n_squared 9 8 sum_of_squares(3) 1 n_squared 9 1 1 n_squared 1 1 n_squared 1 n_squared 1 1 1 1 n_squared 1 1 return 1 1 1 1 2 n_squared + sum_of_squares(n-1) n 1 1 5 elsex elsex(1) n 3 1 1 def sum_of_squares(n): n_squared 1 1 1 2 n_squared = n**2 1 1 1 1 1 3 if n == 1: 1 </td <td></td> <td>fl: sum of squares [parent=Global]</td>		fl: sum of squares [parent=Global]
7 8 sum_of_squares(3) n_squared 8 sum_of_squares(3) n_squared n e f2: sum_of_squares [parent=Global] n 2 n_squared n 1 n_squared 1 f3: sum_of_squares(n): n 1 n_squared 1 1 def sum_of_squares(n): 2 n_squared *func sum_of_squares(n) [parent= 1 def sum_of_squares(n): 2 *func sum_of_squares(n) [parent= 2 n_squared *func sum_of_squares(n) [parent= 4 return 1 sum_of_squares [parent=Global] n_squared 6 return n_squared + sum_of_squares(n-1) n 3 n_squared 9 1 Edit.code f2: sum_of_squares [parent=Global] n_squared 9 1 2		
8 sum_of_squared 9 Edit code f2: sum_of_squares (parent=Global) ccFrat nb just executed n_squared 1 nb just executed f3: sum_of_squares (parent=Global) nb just executed f1: sum_of_squares (parent=Global) f1 def sum_of_squares (n): f1: sum_of_squares (parent=Global) f2 elit code f1: sum_of_squares (parent=Global) f2 elit code f2: sum_of_squares (parent=Global) f2 sum_of_squares (3) n_squared [parent=Global] f2: sum_of_squares (parent=Global) n_squared [parent=Global] f2: sum_of_squares (parent=Global) n_squared [parent=Global] f2: sum_of_squares (parent=Global) n_squared [parent=Global] f3: sum_of_squares [parent=Global] f3: sum_of_squares [parent=Global] f3: sum_of_squares [parent=Global] n_squared [parent=Global]		n 3
Edit code f2: sum_of_squares [parent=Global] c <frail 13="" 17="" [fewmed="" cback="" of="" step=""> Latt>) n</frail>	,	n_squared 9
<	8 sum_ot_squares(3)	
ceFmt eBack Step 13 of 17 Forward > Lat> at has just executed ine to execute n_squared 1 f3: sum_of_squares (parent=Global] n_squared n f4: sum_of_squares (parent=Global] n_squared n f1: sum_of_squares (n): 2 n_squared 1 f2: sum_of_squares (parent=Global] n_squared f1: sum_of_squares (n) (parent= sum_of_squares (parent=Global] n_squared f2: sum_of_squares (parent=Global] n_squared n 3 f2: sum_of_squares (parent=Global] n_squared 1 f2: sum_of_squares (parent=Global] n_squared 1 f2: sum_of_squares (parent=Global] n_squared 1 f3: sum_of_squares (parent=Global] n_squared 1	Edit code	f2: sum_of_squares [parent=Global]
c <fph< td=""> cBack Step 13 of 17 Forwards Latt> ut has just executed ine to execute n_squared 1 f3: sum_of_squares (parent=Global)] n_squared n f4: sum_of_squares (n): 2 n_squared 1 f3: sum_of_squares (n): 2 f3: sum_of_squares (n): 3 f1: f4: return 1 f3: sum_of_squares (parent=Global)] f1: f6: return n_squared + sum_of_squares (n-1) n 3 n f6: sum_of_squares (3) n 3 n_squared [9] f2: sum_of_squares (parent=Global) n 2 f2: sum_of_squares (parent=Global) n 2 f3: sum_of_squares (parent=Global) n 1 screet f1: sum_of_squares (parent=Global) 1</fph<>	0	n 2
at has just executed F3: sum_of_squares [parent=6lobal] ine to execute f3: sum_of_squares [parent=6lobal] 1 n 1 def sum_of_squares(n): 2 n.squared 3 if n == 1: 4 return 1 5 else: 6 return n_squared + sum_of_squares(n-1) 7 sum_of_squares (parent=6lobal] 6 return n_squared + sum_of_squares(n-1) 7 sum_of_squares (parent=6lobal] 6 return n_squared = n*2 8 sum_of_squares (parent=6lobal] 6 return n_squared + sum_of_squares(n-1) 7 sum_of_squares (parent=6lobal] 6 return n_squared = n 2 sum_of_squares (parent=6lobal] 6 return n_squared = n 1 n_squared = n 1 n_squared = n 1 n_squares (parent=6lobal] 6 return n_squares (parent=6lobal] 1 n_squares [parent=6lobal] 1 n_squares [parent=6lobal] 1 n_squares [parent=6lobal] 1 n_squar	U	
Python 3.3 rate Python 3.3 Frames 1 def sum_of_squares(n): Prames 2 n_squared = n*2 fill *3 tf n == 1: func sum_of_squares(n) (parent= *4 return 1 selse: 6 return n_squared + sum_of_squares(n-1) n 7 sum_of_squares(3) 1 Edit code f2: sum_of_squares (parent=Global) n_squared n_squared 1	<< First < Back Step 13 of 17 Forward > Last >>	n_squared 4
Python 3.3 Frames Objects 1 def sum_of_squares(n): 2 n_squared = n*2 3 if n == 1: Image: Sum_of_squares in the sum_of		f3: sum of squares [parent=Global]
Python 3.3 Frames Objects 1 def sum_of_squares(n): 2 n_squared = n*2 2 n_squared = n*2 at hs just executed ne to execute f1: sum_of_squares [parent=Global] f1: sum_of_squares [parent=Global] 6 return n_squared + sum_of_squares(n-1) n_squared g 7 sum_of_squares(3) n_squared g f2: sum_of_squares [parent=Global] ((seriet (slask: Step 14 of 17 Forward) (at hs just executed ne to execute	ine to execute	
Python 3.3 Frames Objects 1 def sum_of_squares(n): 2 n_squared = n*2 3 if n = 1: 4 return 1 5 else: 1 1 fsquares(n): 1 fsquares(n): 1 1 fsquares(n): 1 fsquares(n): fsquares(n): 1 fsquares(n): fsquar		
1 def sum_of_squares(n): 2		n_squared 1
2 n_squared = n*2 3 if n = 1: 4 return 1 5 else: 6 return n_squared + sum_of_squares(n-1) 7 n_squared = 1 8 sum_of_squares (3) Edit code (ceFrent (Back Step 14 of 17 Forward) Last>) at has just executed ne to execute	Python 3.3	Frames Objects
2 n_squared = n*2 3 if n = 1: 4 return 1 5 else: 6 return n_squared + sum_of_squares(n-1) 7 n_squared i 8 sum_of_squares (3) Edit code 6 f1: sum_of_squares [parent=Global] n_squared i else: Edit code f2: sum_of_squares [parent=Global] n_squared i (sefent i @Back Step 14 of 17 Forward>) at has just executed n i n i n i	1 def sum of squares(n):	
a if n == 1: sum_of_squares [* 4 return 1 fil: sum_of_squares [parent=Global] 5 else: n 6 return n_squared + sum_of_squares (n-1) n 7 sum_of_squares (3) n_squared Edit code f2: sum_of_squares [parent=Global] <		Global Trame Tunc Sum_ot_Squares(n) [parent
* 4 return 1 5 return n_squared + sum_of_squares(n-1) 6 return n_squared + sum_of_squares(n-1) 7 n_squared 9 8 sum_of_squares(3) Edit code f2: sum_of_squares [parent=Global] excPent Bask Step 14 of 17 reworded f3: sum_of_squares [parent=Global] e to execute f3: sum_of_squares [parent=Global]		sum_of_squares
5 else: 6 return n_squared + sum_of_squares(n-1) 7 sum_of_squares(3) 8 sum_of_squares(3) 6 f2: 6 return n_squared = 7 f2: 6 return n_squared = 6 return n_squared = 6 return n_squared = 6 return n_squared = 72: sum_of_squares [parent=Global] 0 n_squared = 0 n_squared = 1 1		
6 return n_squared + sum_of_squares(n-1) n 3 7 n_squared 9 8 sum_of_squares(3) n_squared Edit code 6 f2: sum_of_squares n a this just executed n n 1 (setPati (Back Step 14 of 17 Forward> 1 (setPati (Back (setPati (Back (Back (SetPati (Back (SetPati (Back		

Environments Example

Python 3.3	Frames Objects
1 def sum_of_squares(n): 2 n_squared = n**2 3 if n == 1:	Global frame sum_of_squares func sum_of_squares(n) [parent:
4 return 1 5 else: 6 return n_squared + sum_of_squares(n-1)	fl: sum_of_squares [parent=Global]
7 8 sum_of_squares(3)	n 3 n_squared 9
Edit code	f2: sum_of_squares [parent=Global]
<pre><</pre> << First < Back Step 15 of 17 Forward > Last >>	n 2 n_squared 4
hat has just executed	f3: sum_of_squares [parent=Global]
	n 1 n_squared 1
	Return value

2/22/16

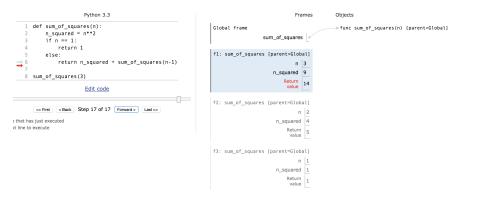
UCB CS88 Sp16 L4

18

Environments Example

Python 3.3	Frames Objects
<pre>1 def sum_of_squares(n): 2 n_squared = n**2 3 if n == 1: 4 return 1</pre>	Global frame
<pre>5 else: 6 return n_squared + sum_of_squares(n-1) 7 8 sum_of_squares(3)</pre>	f1: sum_of_squares [parent=6lobal] n 3 n_squared 9
Edit code	f2: sum_of_squares [parent=6lobal] n 2 n_squared 4 Return 5
	f3: sum_of_squares [parent=Global] n 1 squared 1 Rturn 1 value 1

Environments Example



Questions



- In what order do we sum the squares ?
- How does this compare to iterative approach ?

```
def sum_of_squares(n):
    accum = 0
    for i in range(1,n+1):
        accum = accum + i*i
    return accum
```



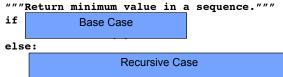
```
def first(s):
```

"""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):
```



Recursion over sequence length, rather than number magnitude

2/22/16	UCB CS88 Sp16 L4	21	2/22/16	UCB CS88 Sp16 L4	22





In [104]:	<pre>def min_r(s):</pre>
	<pre>print('min_r:', s)</pre>
	if len(s) == 1:
	<pre>return first(s)</pre>
	else:
	<pre>result = min(first(s), min_r(rest(s)))</pre>
	<pre>print('min_r:', s," => ", result)</pre>
	return result

In [105]: min_r([3,4,2,5,11])

```
min_r: [3, 4, 2, 5, 11]
min_r: [4, 2, 5, 11]
min_r: [5, 11]
min_r: [5, 11]
min_r: [11]
min_r: [5, 11] => 5
min_r: [2, 5, 11] => 2
min_r: [4, 2, 5, 11] => 2
min_r: [3, 4, 2, 5, 11] => 2
```

- What about sum?
- Don't confuse print with return value

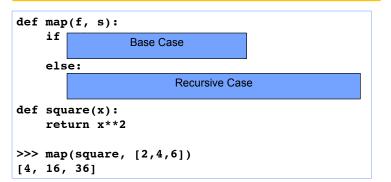
2/22/16

UCB CS88 Sp16 L4

23

2/22/16

Recursion with Higher Order Fun



• Divide and conquer



Trust ...



• The recursive "leap of faith" works as long as we hit the base case eventually

How much ???

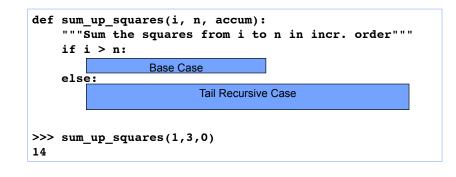
- Time is required to compute sum of squares(n)?
 - Recursively?
 - Recursively
 Iteratively ?
- Space is required to compute sum_of_squares(n)?
 - Recursively?
 - Iteratively ?
- Count the frames...
- · Recursive is linear, iterative is constant !
- · And what about the order of evaluation ?

2/22/16	UCB CS88 Sp16 L4	25	2/22/16	UCB CS88 Sp16 L4	26

Tail Recursion



- All the work happens on the way down the recursion
- On the way back up, just return



Using HOF to preserve interface

def sum_upper(i, accum):
if i > n:
return accum
else:
return sum upper(i+1, accum + i*i)

- What are the globals and locals in a call to sum_upper?
 Try python tutor
- · Lexical (static) nesting of function def within def vs
- Dynamic nesting of function call within call

2/22/16



Linear

proportional to n

cn for some c

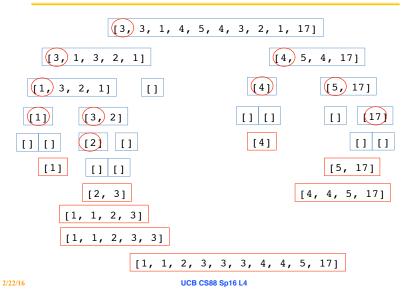
Tree Recursion



• Break the problem into multiple smaller subproblems, 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 qsort(s):
    """Sort a sequence - split it by the first element,
    sort both parts and put them back together."""
    if not s:
        return []
    else:
        pivot = first(s)
        lessor, more = split(pivot, rest(s))
        return qsort(lessor) + [pivot] + qsort(more)
>>> qsort([3,3,1,4,5,4,3,2,1,17])
[1, 1, 2, 3, 3, 3, 4, 4, 5, 17]
```

QuickSort Example



30



UCB CS88 Sp16 L4

29

4

Tree Recursion with HOF



```
def qsort(s):
    """Sort a sequence - split it by the first element,
    sort both parts and put them back together."""
    if not s:
        return []
    else:
        pivot = first(s)
        lessor, more = split_fun(leq_maker(pivot), rest(s))
        return qsort(lessor) + [pivot] + qsort(more)
>>> qsort([3,3,1,4,5,4,3,2,1,17])
[1, 1, 2, 3, 3, 3, 4, 4, 5, 17]
```