Lecture 6: Environment Diagrams, Recursion Review, Midterm Review

March 4, 2019

http://inst.eecs.berkeley.edu/~cs88

UC Berkeley EECS
Adj. Ass. Prof.
Dr. Gerald Friedland
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

- Recursion
- Environment Diagrams
Recursion Key concepts – by example

1. Test for simple “base” case

2. Solution in simple “base” case

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

3. Assume recursive solution to simpler problem

4. Transform solution of simpler problem into full solution
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 $n^2$

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

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

- Each call has its own “frame” of local variables
- What about globals?
- Let’s see the environment diagrams

https://goo.gl/CiFaUJ
Environments Example

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

sum_of_squares(3)
```

Edit code
Environments Example

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

sum_of_squares(3)
```

**Frames**

Global frame

- `sum_of_squares`

**Objects**

- `sum_of_squares(n) [parent=Global]`

- `f1: sum_of_squares [parent=Global]`
  - `n 3`
  - `n_squared 9`

3/04/19

UCB CS88 Sp19 L6
Environments Example

Python 3.3

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

    sum_of_squares(3)
```

Frames

_objects_

Objects

Global frame

```
func sum_of_squares(n) [parent=Global]
    sum_of_squares
```

f1: sum_of_squares [parent=Global]

```
n 3
n_squared 9
```

f2: sum_of_squares [parent=Global]

```
n 2
```

Frames

_objects_

Objects

Global frame

```
func sum_of_squares(n) [parent=Global]
    sum_of_squares
```

f1: sum_of_squares [parent=Global]

```
n 3
n_squared 9
```

f2: sum_of_squares [parent=Global]

```
n 2
n_squared 4
```
Environments Example

Python 3.3

```python
1 def sum_of_squares(n):
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)
```

Frames

Objects

Global frame

```text
 func sum_of_squares(n) [parent=Global]
 sum_of_squares
```

f1: sum_of_squares [parent=Global]

```text
    n
    3
    n_squared
    9
```

f2: sum_of_squares [parent=Global]

```text
    n
    2
    n_squared
    4
```

f3: sum_of_squares [parent=Global]

```text
    n
    1
```
Environments Example

Python 3.3

```python
1 def sum_of_squares(n):
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)
```

that has just executed
that line to execute

Frames

Objects

Global frame

```
  func sum_of_squares(n) [parent=Global]

  sum_of_squares
```

f1: sum_of_squares [parent=Global]

```
  n 3

  n_squared 9
```

f2: sum_of_squares [parent=Global]

```
  n 2

  n_squared 4
```

f3: sum_of_squares [parent=Global]

```
  n 1

  n_squared 1
```
Environments Example

Python 3.3

1 def sum_of_squares(n):
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)

Frames

Global frame

- func sum_of_squares(n) [parent=Global]
  - sum_of_squares

f1: sum_of_squares [parent=Global]

- n 3
- n_squared 9

f2: sum_of_squares [parent=Global]

- n 2
- n_squared 4

f3: sum_of_squares [parent=Global]

- n 1
- n_squared 1
- Return value 1

Edit code

的气息 has just executed
next line to execute
Environments Example

```
Python 3.3

1 def sum_of_squares(n):
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)
```

Frames | Objects
--- | ---
Global frame | func sum_of_squares(n) [parent=Global]

f1: sum_of_squares [parent=Global]
- n: 3
- n_squared: 9

f2: sum_of_squares [parent=Global]
- n: 2
- n_squared: 4
- Return value: 5

f3: sum_of_squares [parent=Global]
- n: 1
- n_squared: 1
- Return value: 1
Environments Example

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

sum_of_squares(3)
```

Frames

<table>
<thead>
<tr>
<th>Global frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum_of_squares</td>
</tr>
</tbody>
</table>

Objects

<table>
<thead>
<tr>
<th>f1: sum_of_squares [parent=Global]</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
</tr>
<tr>
<td>n_squared</td>
</tr>
<tr>
<td>Return value</td>
</tr>
</tbody>
</table>

f2: sum_of_squares [parent=Global]

<table>
<thead>
<tr>
<th>f3: sum_of_squares [parent=Global]</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
</tr>
<tr>
<td>n_squared</td>
</tr>
<tr>
<td>Return value</td>
</tr>
</tbody>
</table>

that has just executed

d line to execute
How much ???

• Time is required to compute \texttt{sum_of_squares}(n) ?
  – Recursively?
  – Iteratively ?

• Space is required to compute \texttt{sum_of_squares}(n) ?
  – Recursively?
  – Iteratively ?

• Count the frames…

• Recursive is linear, iterative is constant!
Tail Recursion

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

```python
def sum_up_squares(i, n, accum):
    """Sum the squares from i to n in incr. order""
    if i > n:
        Base Case
    else:
        Tail Recursive Case

>>> sum_up_squares(1,3,0)
14
```
Tree Recursion

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

```python
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

\[\begin{array}{l}
[3, 3, 1, 4, 5, 4, 3, 2, 1, 17] \\
[3, 1, 3, 2, 1] \\
[1, 3, 2, 1] \\
[1] \\
[1, 3, 2, 1] \\
[1] \\
[1, 2, 3] \\
[1, 1, 2, 3] \\
[1, 1, 2, 3, 3] \\
[1, 1, 2, 3, 3, 4, 4, 5, 17]
\end{array}\]
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]
On Computer Science Exams

In computer science exams, we try to assess the student’s understanding of concepts and his or her ability to practically apply these.

- In CS, we do not:
  - require extensive memorization (e.g. we allow cheat sheet)
  - require a lot of reading
  - require essay writing skills

In CS, we do:
- require the ability to translate a given textual problem into programming code
- require you to be able to read other people’s code
- value solutions that are almost right over no solution
- accept solutions we did not think about if they work
- prioritize math (logic) and science (experiment) over opinion or authority
How to prepare for a CS exam

• Explain the content of the computational concepts toolbox to somebody else
  • Describe the concept
  • What is an example of using it?
  • When does it not work? Corner cases?
  • Why does it exist?

• Practice programming:
  – Play around with the examples from lecture, lab, homework
  – Think about your own similar examples

• In the exam:
  – Make sure you understand the question: What is the given input?
    What is the required output?
  – Think of easy cases first (e.g. n=1?).
  – What is the iteration/recursion doing (e.g. i=i+1)?
  – What are corner cases that need explicit handling (e.g. division by zero, negative numbers, empty list)?
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
- Recursion
- Environment Diagrams
The computer choses a random element $x$ of the list generated by $\text{range}(0,n)$. What is the smallest amount of iteration/recursion steps the best algorithm needs to guess $x$?

$\log_2 n$

How would the algorithm look like?

Guess the binary digits of $x$ starting with the highest significant digit. This is, ask questions of the form “smaller than $2^{n-1}$?” (yes => 0…), “smaller than $2^{n-2}$?” (no => 0 1…), “smaller than $2^{n-2}+2^{n-3}$?”, …

This method is also called: binary search

Quantum physics: Allow less than $\log_2 n$ guesses.