Lecture 15: Recursion (Sections 5.8-5.10)

CS 1110

Introduction to Computing Using Python

http://www.cs.cornell.edu/courses/cs1110/2018sp
Announcements

Prelim 1

- Graded. 2 Emails went out (1 from gradescope, 1 from CMS) and some #s released
- **Regrade requests**: we are processing them
- **Q3**: we are re-evaluating them

Lab 8: alive!

(There was no Lab 7.)
Recursion

• Recursive Function:
  A function that calls itself (directly or indirectly)
Examples

- Blast-off
- Gift in gift
- Family Trees
- Towers of Hanoi
- Deblanking
Tower of Hanoi

- Three towers: *left*, *middle*, and *right*
- *n* disks of unique sizes on *left*
- **Goal**: move all disks from *left* to *right*
- Cannot put a larger disk on top of a smaller disk
1 Disc: Easy!

1. Move from *left* to *right*
2 Discs: Step 1

1. Move from *left* to *middle*
2 Discs: Step 2

1. Move from *left* to *middle*
2. Move from *left* to *right*
2 Discs: Step 3 (final)

1. Move from *left* to *middle*
2. Move from *left* to *right*
3. Move from *middle* to *right*
3 Discs: Step 1

1. Move from \textit{left} to \textit{right}
3 Discs: Step 2

1. Move from *left* to *right*
2. Move from *left* to *middle*
3 Discs: Step 3

1. Move from *left* to *right*
2. Move from *left* to *middle*
3. Move from *right* to *middle*
3 Discs: Step 4

1. Move from left to right
2. Move from left to middle
3. Move from right to middle
4. Move from left to right
3 Discs: Step 5

1. Move from left to right
2. Move from left to middle
3. Move from right to middle
4. Move from left to right
5. Move from middle to left
3 Discs: Step 6

1. Move from left to right
2. Move from left to middle
3. Move from right to middle
4. Move from left to right
5. Move from middle to left
6. Move from middle to right
3 Discs: Step 7 (final)

1. Move from *left* to *right*
2. Move from *left* to *middle*
3. Move from *right* to *middle*
4. Move from *left* to *right*
5. Move from *middle* to *left*
6. Move from *middle* to *right*
7. Move from *left* to *right*
4 Discs: Oh, boy...

BUT, we already know how to solve Hanoi for 3 towers.

Sooo....
Divide and Conquer

**Goal**: Solve really big problem P

**Idea**: Split into smaller problems, solve, combine

### 3 Steps:
1. Decide what to do for simple cases
2. Decide how to break up the task
3. Decide how to combine your work
Decide what to do for simple cases

Move from *left* to *right*
Decide how to break up the task

Simpler than the original task, slowly becoming the “simple case”

Simple case
Decide how to combine your work

\[ \text{Hanoi}(4) \]

\( \rightarrow \)

\[ \text{Hanoi}(3) \]

(uncover the big one)

\[ \text{move the big one} \]

\[ \text{Hanoi}(3) \]

(cover the big one)
4 Discs: High-level Idea

1. Move top three disks from \textit{left} to \textit{middle}
2. Move largest disk from \textit{left} to \textit{right}
3. Move top three disks from \textit{middle} to \textit{right}
solve_hanoi(source, target, other, n_disks)

"""Prints instructions for how to move n_disks (sorted small to large, going down) from the source peg to the target peg, using the other peg if needed. """

if (n==1):
    print("move from "+source+" to "+target)
else:
    # need to move top n_disks-1 disks from source to
    # other... luckily, I have a function that does that!
    solve_hanoi(source, other, target, n_disks-1)

    # move the last disk to the target, that's easy!
    print("move from "+source+" to "+target)

    # now put everything back on the last disk at target
    solve_hanoi(other, target, source, n_disks-1)
Recursion vs Iteration

• **Recursion** is *provably equivalent* to iteration
  - Iteration includes **for-loop** and **while-loop** (later)
  - Anything can do in one, can do in the other

• But some things are easier with recursion
  - And some things are easier with iteration

• Will **not** teach you when to choose recursion

• We just want you to **understand the technique**
Examples

• Blast-off
• Gift in gift
• Family Trees
• Towers of Hanoi
• Deblanking
Divide and Conquer

**Goal**: Solve problem P on a piece of data
Divide and Conquer

**Goal:** Solve problem P on a piece of data

**Idea:** Split data into two parts and solve problem

- data
- data 1
- data 2

Solve Problem P  Solve Problem P
Divide and Conquer

**Goal**: Solve problem P on a piece of data

**Idea**: Split data into two parts and solve problem

Combine Answer!
Exercise: Remove Blanks from a String

```python
def deblank(s):
    """Returns: s but with its blanks removed"""

1. Decide what to do on “small” data
   - If it is the empty string, nothing to do
     ```python
     if s == "":
         return s
     ```
   - If it is a single character, delete it if a blank
     ```python
     if s == ' ':
         # There is a space here
         return ''
     # Empty string
     else:
         return s
     ```
```
Exercise: Remove Blanks from a String

```python
def deblank(s):
    """Returns: s but with its blanks removed"""

2. Decide how to break it up

    left = deblank(s[0])  # A string with no blanks
    right = deblank(s[1:])  # A string with no blanks

3. Decide how to combine the answer

    return left+right  # String concatenation
```
def deblank(s):
    """Returns: s w/o blanks"""
    if s == '':
        return s
    elif len(s) == 1:
        if s[0] == ' ':
            return ''
        else:
            return s
    left = deblank(s[0])
    right = deblank(s[1:])
    return left + right
def deblank(s):
    """Returns: s w/o blanks"""
    if s == "":
        return s
    elif len(s) == 1:
        if s[0] == ' ':
            return ''
        else:
            return s
    left = deblank(s[0])
    right = deblank(s[1:])
    return left+right
Following the Recursion

debblank

a  b  c

stop (base case)

debblank

a  b  c

debblank

a

stop (base case)

debblank

b  c

...
Breaking it up (1)

deblanks

\[
\begin{array}{ccc}
a & b & c \\
\end{array}
\]

\[
\begin{array}{ccc}
a & b & c \\
\end{array}
\]
Breaking it up (2)
Breaking it up (3)
Breaking it up (4)

deblank

```
  a b c
```

deblank

```
  a b c
```

```
a  b c
```

```
b  c
```

```
c
```
Breaking it up (5)

deblank

a  b  c

deblank

a  b  c

da  deblank

b  c

deblank

b  c

b  deblank

c

deblank

c
Breaking it up (6)
Combining Left+Right (1)

deblank

\[
\begin{array}{ccc}
 a & b & c \\
 a & b & c \\
 a & b & c \\
 a & b & c \\
 a & b & c \\
 b & b & c \\
 b & b & c \\
 b & b & c \\
 b & b & c \\
 b & b & c \\
 c & c & c \\
 c & c & c \\
 c & c & c \\
 c & c & c \\

deblank
\end{array}
\]
Combining Left+Right (2)
Combining Left+Right (3)
Combining Left+Right (4)
Combining Left+Right (5)
Combining Left+Right (6)

deblank

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\times & \text{deblank} & \text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\times & \text{deblank} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\times & \text{deblank} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{c} \\
\end{array}
\]
Combining Left+Right (7)

deblank

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]