# Lecture 6: Subroutines & Recursion

**CS105: Great Insights in Computer Science**  
*Michael L. Littman, Fall 2006*

## Recap: Reduction

<table>
<thead>
<tr>
<th>Level</th>
<th>Examples</th>
<th>Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>software libraries</td>
<td>graphics, animation, robotics</td>
<td>networking, security, mathematics</td>
</tr>
<tr>
<td>high-level language</td>
<td>Python</td>
<td>C, Java, C++, Logo, LISP, Fortran, ML</td>
</tr>
<tr>
<td>machine language</td>
<td>ML³</td>
<td>x86, CARDIAC, Z80</td>
</tr>
<tr>
<td>logic gates</td>
<td>equal, ifthenelse, add</td>
<td>memlookup, memwrite</td>
</tr>
<tr>
<td>basic logic gates</td>
<td>and, or, not</td>
<td>nor, nand, xor</td>
</tr>
<tr>
<td>physical bits</td>
<td>0,1 via high/low voltage</td>
<td>water pressure, kinetic energy</td>
</tr>
</tbody>
</table>
Does It End There?

- Of course, not.
- We can continue to build sophisticated programs out of simpler programs.
- The idea of subroutines (procedures, functions) makes this work.

Subroutines

- A lot of research in computer science is about designing and creating just the right set of subroutines, sometimes called libraries.
- You don’t have enough background yet to weigh in on these problems.
- But, there is an analogous set of problems where you are already an expert...
Sing-A-Long Programs

Gilligan’s Island Theme

Just sit right back and you'll hear a tale, a tale of a fateful trip.
That started from this tropic port, aboard this tiny ship.
The mate was a mighty sailin' man, the skipper brave and sure.
Five passengers set sail that day, for a three hour tour, a three hour tour...
The weather started getting rough, the tiny ship was tossed.
If not for the courage of the fearless crew,
the Minnow would be lost; the Minnow would be lost.
The ship took ground on the shore of this uncharted desert isle,
with Gilligan, the Skipper too, the Millionaire, and his Wife,
the Movie Star, the Professor and Mary Ann, here on Gilligan's Isle.

Python Code

def GilligansIsland():
    print "Just sit right back and you'll hear a tale, a tale of a fateful trip."
    print "That started from this tropic port, aboard this tiny ship."
    print "The mate was a mighty sailin' man, the skipper brave and sure."
    print "Five passengers set sail that day, for a three hour tour, a three hour tour."
    print "The weather started getting rough, the tiny ship was tossed."
    print "If not for the courage of the fearless crew,"
    print "the Minnow would be lost; the Minnow would be lost."
    print "The ship took ground on the shore of this uncharted desert isle,"
    print "with Gilligan, the Skipper too, the Millionaire, and his Wife,"
    print "the Movie Star, the Professor and Mary Ann, here on Gilligan's Isle."

GilligansIsland()
Dreadful sorry, Clementine
Thou art lost and gone forever
Oh my darling, Clementine!
Oh my darling, oh my darling,
Sandals were for Clementine.
Herring boxes, without topses,
And her shoes were number nine,
Light she was and like a fairy,
Dreadful sorry, Clementine.
Thou art lost and gone forever
Oh my darling, Clementine!
Oh my darling, oh my darling,
And his daughter Clementine.
In a cavern, in a canyon,
Clementine
Drove she ducklings to the water
Ev'ry morning just at nine,
Hit her foot against a splinter,
Fell into the foaming brine.
Oh my darling, oh my darling,
Oh my darling, Clementine!
Thou art lost and gone forever
Dreadful sorry, Clementine.
Ruby lips above the water,
Blowing bubbles, soft and fine,
But, alas, I was no swimmer,
So I lost my Clementine.
Oh my darling, oh my darling,
Oh my darling, Clementine!
Thou art lost and gone forever
Dreadful sorry, Clementine.

```python
from Clementine():
    print "In a cavern, in a canyon,"
    print "Excavating for a mine,"
    print "Dwelt a miner forty niner,"
    print "And his daughter Clementine."
    print
    print "Oh my darling, oh my darling,"
    print "Oh my darling, Clementine!"
    print "Thou art lost and gone forever"
    print "Dreadful sorry, Clementine."
    print
    print "Ruby lips above the water,"
    print "Blowing bubbles, soft and fine,"
    print "But, alas, I was no swimmer,"
    print "So I lost my Clementine."
    print
    print "How I missed her! How I missed her,"
    print "How I missed my Clementine,"
    print "But I kissed her little sister,"
    print "I forgot my Clementine."
    print
    print "Oh my darling, oh my darling,"
    print "Oh my darling, Clementine!"
    print "Thou art lost and gone forever"
    print "Dreadful sorry, Clementine."
Clementine()
```
Using A Subroutine

def ClementineChorus():
    print "Ruby lips above the water,"
    print "Blowing bubbles, soft and fine,"
    print "But, alas, I was no swimmer,"
    print "So I lost my Clementine."
    ClementineChorus()
    print "How I missed her! How I missed her,"
    print "But I kissed her little sister,"
    print "I forgot my Clementine."
    ClementineChorus()
    print
    print "Oh my darling, oh my darling,"
    print "Oh my darling, ... then the program resumes from where the call took place.

• Simplifies the program, easier to maintain, extend, fix.

Now, “ClementineChorus()” is its own little program, or a new statement in the language, that we can call whenever we need it.

Calling the subroutine does all the steps, then the program resumes from where the call took place.

• Simplifies the program, easier to maintain, extend, fix.

Shared Structure

All My Loving
Lennon/McCartney

Close your eyes and I'll kiss you,
Tomorrow I'll miss you,
Remember I'll always be true,
And then while I'm away,
I'll write home every day,
And I'll send all my loving to you.

I'll pretend that I'm kissing,
The lips I am missing,
And hope that my dreams will come true,
And then while I'm away,
I'll write home every day,
And I'll send all my loving to you.

All my loving, I will send to you,
All my loving, darling, I'll be true.
Close your eyes and I'll kiss you,
Tomorrow I'll miss you,
Remember I'll always be true,
And then while I'm away,
I'll write home every day,
And I'll send all my loving to you.

What subroutines would you define?

What subroutines would you define?
def A	heway():
print "And then while I'm away"
print "I'll write home every day"
print "And I'll send all my loving to you"
def closeYourEyes():
print "Close your eyes and I'll kiss you"
print "Tomorrow I'll miss you"
print "Remember I'll always be true"
def allMyLoving():
closeYourEyes()
print "I'll pretend that I'm kissing"
print "The lips I am missing"
print "And hope that my dreams will come true"
def Away():
closeYourEyes()
def Away():
print "And then while I'm away"
print "I'll write home every day"
print "And I'll send all my loving to you"
def closeYourEyes():
print "Close your eyes and I'll kiss you"
print "Tomorrow I'll miss you"
print "Remember I'll always be true"
print "And then while I'm away"
print "I'll write home every day"
print "And I'll send all my loving to you"
print "All my loving, I will send to you"
print "All my loving, darling I'll be true"
print "All my loving, I will send to you"
print "All my loving, darling I'll be true"
print "All my loving, All my loving"
print "Woo, all my loving, I will send to you"

Deeper Nesting?

All My Loving
Lennon/McCartney

Close your eyes and I'll kiss you,
Remember I'll always be true,
And then while I'm away,
I'll write home every day,
And I'll send all my loving to you,
I'll pretend I am kissing,
The lips I am missing,
And hope that my dreams will come true,
And then while I'm away,
I'll write home every day,
And I'll send all my loving to you,
All my loving, I will send to you,
All my loving, darling, I'll be true,
Close your eyes and I'll kiss you,
Tomorrow I'll miss you,
Remember I'll always be true,
And then while I'm away,
I'll write home every day,
And I'll send all my loving to you,
All my loving, I will send to you,
All my loving, darling, I'll be true,
All my loving, I will send to you.

This subroutine called its own subroutine.

What subroutines would you define?

More Structure

Eight Days a Week (Beatles)

Hold me, love me,
Hold me, love me,
Ain't got nothin' but love babe,
Eight days a week.

R
B
R
Y
R
G
B

Hold me, love me,
Hold me, love me,
Ain't got nothin' but love babe,
Eight days a week.

Eight days a week I love you,
Eight days a week is not enough to show I care.

R
G
B
R
G
B

Ooh I need your love babe, guess you know it's true,
Hope you need my love 'just like I need you,

R
B
R
Y
R
G
B

Love you ev'ry day girl, always on my mind,
One thing I can say girl, love you all the time,

R
B
R
Y
R
G
B

Hold me, love me,
Hold me, love me,
Ain't got nothin' but love babe,
Eight days a week.

Eight days a week I love you,
Eight days a week is not enough to show I care.

R
G
B
R
G
B

Ooh I need your love babe, guess you know it's true,
Hope you need my love 'just like I need you,
def B():
    print "Ooh I need your love babe"
    print "Guess you know it's true"
    print "Hope you need my love babe"
    print "Just like I need you"
def R():
    print "Hold me, love me"
    print "Hold me, love me"
    print "Ain't got nothin' but love babe"
    print "Eight days a week"
def Y():
    print "Love you every day girl"
    print "Always on my mind"
    print "One thing I can say girl"
    print "Love you all the time"
def G():
    print "Eight days a week"
    print "I love you"
    print "Eight days a week"
    print "Is not enough to show I care"
def eightDaysAWeek():
    B(); R(); Y(); R(); G()
    B(); R(); G(); Y(); R()
    print "Eight days a week"
    print "Eight days a week"
def OldMacdonald():
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a cow, E-I-E-I-O'
    moo()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a pig, E-I-E-I-O'
    snort()
    moo()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a horse, E-I-E-I-O'
    neigh()
    snort()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a chick, E-I-E-I-O'
    cluck()
    neigh()
    snort()

moo():
    print 'With a "moo-moo" here and a "moo-moo" there'
    print 'Here a "moo" there a "moo"'
    print 'Everywhere a "moo-moo"'

def moo():
    print 'Wit a "moo-moo" here and a "moo-moo" there'
    print 'Here a "moo" there a "moo"'
    print 'Everywhere a "moo-moo"'

def snort():
    print 'With a (snort) here and a (snort) there'
    print 'Here a (snort) there a (snort)'
    print 'Everywhere a (snort-snort)'

def neigh():
    print 'With a "neigh, neigh" here and a "neigh, neigh" there'
    print 'Here a "neigh" there a "neigh"'
    print 'Everywhere a "neigh-neigh"'

def neigh():
    print 'With a "neigh, neigh" here and a "neigh, neigh" there'
    print 'Here a "neigh" there a "neigh"'
    print 'Everywhere a "neigh-neigh"'

def cluck():
    print 'With a "cluck, cluck" here and a "cluck, cluck" there'
    print 'Here a "cluck" there a "cluck"'
    print 'Everywhere a "cluck-cluck"'

def cluck():
    print 'With a "cluck, cluck" here and a "cluck, cluck" there'
    print 'Here a "cluck" there a "cluck"'
    print 'Everywhere a "cluck-cluck"'

But Notice...

• Each time we “snort”, we follow it with “moo”.
• Each time we “neigh”, we follow it with “snort”.
• Each time we “cluck”, we follow it with “neigh”.
• Let’s move the earlier noises into the later subroutines...
def OldMacdonald():
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a cow, E-I-E-I-O'
    moo()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a pig, E-I-E-I-O'
    snort()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a horse, E-I-E-I-O'
    neigh()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a chick, E-I-E-I-O'
    cluck()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print "

def moo():
    print 'With a "moo-moo" here and a "moo-moo" there'
    print 'Here a "moo" there a "moo"'
    print 'Everywhere a "moo-moo"'

def snort():
    print 'With a (snort) here and a (snort) there'
    print 'Here a (snort) there a (snort)'
    print 'Everywhere a (snort-snort)'
    moo()

def neigh():
    print 'With a "neigh, neigh" here and a "neigh, neigh" there'
    print 'Here a "neigh" there a "neigh"'
    print 'Everywhere a "neigh-neigh"'
    snort()

def cluck():
    print 'With a "cluck, cluck" here and a "cluck, cluck" there'
    print 'Here a "cluck" there a "cluck"'
    print 'Everywhere a "cluck-cluck"'
    neigh()
Subroutine Stack

```python
def OldMacdonald():
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a cow, E-I-E-I-O'
    print 'And on his farm he had a chick, E-I-E-I-O'
    cluck()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print ''
```

Subroutine Stack

```python
def moo():
    print 'With a "moo-moo" here'
    print '  and a "moo-moo" there'
    print 'Here a "moo" there a "moo"'
    print 'Everywhere a "moo-moo"'
```

Subroutine Stack

```python
def neigh():
    print 'With a "neigh, neigh" here'
    print '     and a "neigh, neigh" there'
    print 'Here a "neigh" there a "neigh"'
    print 'Everywhere a "neigh-neigh"'
    snort()```

Subroutine Stack

```python
def snort():
    print 'With a "snort" here and a "snort" there'
    print 'Here a "snort" there a "snort"'
    print 'Everywhere a "snort-snort"'
```
Subroutine Stack

```
def OldMacdonald():
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print 'And on his farm he had a cow, E-I-E-I-O'
    print 'And on his farm he had a chick, E-I-E-I-O'
    cluck()
    print 'Old Macdonald had a farm, E-I-E-I-O'
    print ''

def neigh():
    print 'With a "neigh, neigh" here'
    print '     and a "neigh, neigh" there'
    print 'Here a "neigh" there a "neigh"
    print 'Everywhere a "neigh-neigh"
    snort()

def moo():
    print 'With a "moo-moo" here'
    print '  and a "moo-moo" there'
    print 'Here a "moo" there a "moo"
    print 'Everywhere a "moo-moo"

def cluck():
    print 'With a "cluck, cluck" here'
    print '  and a "cluck, cluck" there'
    print 'Here a "cluck" there a "cluck"
    print 'Everywhere a "cluck-cluck"

def snort():
    print 'With a (snort) here and a (snort) there'
    print 'Here a (snort) there a (snort)
    print 'Everywhere a (snort-snort)'
```

Subroutine Stack
Subroutines So Far

- Can be used to gather up repeated code in one place.
- Can be deeply nested to capture structure when necessary.
- Next, parameters greatly increase the power of the subroutine idea...

Near Miss...

Magical Mystery Tour (Beatles)

Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
The magical mystery tour is to take you away,
to take you away.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
The magical mystery tour is to take you away,
to take you away.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
Roll up, roll up for the mystery tour.
The magical mystery tour is to take you away,
to take you away.

- Verse structure is nearly repetitive. Can’t quite define a single subroutine that covers all three.
- If we could fill in the blank, we could reuse the same routine...
def mysteryVerse(x):
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "The magical mystery tour is "
    print " waiting to take you away, "
    print " waiting to take you away."
    print

• x is called the formal parameter, “waiting” is the actual parameter.

Naive Python Code

def mysteryVerseWaiting():
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "The magical mystery tour is "
    print " waiting to take you away, "
    print " waiting to take you away."
    print

def mysteryVerseComing():
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "The magical mystery tour is "
    print " coming to take you away, "
    print " coming to take you away."
    print

def magicalMysteryTour():
    mysteryVerseWaiting()
    mysteryVerseComing()
    print "The magical mystery tour is "
    print " coming to take you away, "
    print " coming to take you away."

Formal Parameters

def mysteryVerse(x):
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "Roll up, roll up for the mystery tour."
    print "The magical mystery tour is "
    print " + x + " to take you away, "
    print " + x + " to take you away."
    print

• When mysteryVerse("waiting") is executed, the caller temporarily assigns
  the variable “x” the value “waiting”.

• The subroutine does its thing, using the value “waiting” whenever
  “x” appears. When finished, it returns to the caller, and “x” is
  forgotten.

• x is called the formal parameter, “waiting” is the actual parameter.
Macdonald #3

def OldMacdonald():
moo(True)
snort(True)
neigh(True)
cluck(True)
def doIntro(animal, noise, intro):
    if intro:
        print 'Old Macdonald had a farm, E-I-E-I-O'
        print 'And on his farm he had a ' + animal + ', E-I-E-I-O'
makeNoise(noise)
    if intro:
        print 'Old Macdonald had a farm, E-I-E-I-O'
        print ''
def makeNoise(noise):
    print 'With a "' + noise + '-' + noise + '  here and a '
    print '  "' + noise + '-' + noise + '  there'
    print 'Here a "' + noise + '+' + ' there a "' + noise + '+'
    print 'Everywhere a "' + noise + '+' + ' there a "' + noise + '+'
def moo(intro):
doIntro("cow", "moo", intro)
def snort(intro):
doIntro("pig", "(snort)", intro)
moo(False)
def neigh(intro):
doIntro("horse", "neigh", intro)
norst(False)
def cluck(intro):
doIntro("chick", "cluck", intro)
negh(False)

Understand Inside Out

def makeNoise(noise):
    print 'With a "' + noise + '+' + ' here and a '
    print '  "' + noise + '+' + ' there'
    print 'Here a "' + noise + '+' + ' there a "' + noise + '+'
    print 'Everywhere a "' + noise + '+' + ' there a "' + noise + '+'

def doIntro(animal, noise, intro):
    if intro:
        print 'Old Macdonald had a farm, E-I-E-I-O'
        print 'And on his farm he had a ' + animal + ', E-I-E-I-O'
makeNoise(noise)
    if intro:
        print 'Old Macdonald had a farm, E-I-E-I-O'
        print ''

    Writes out the “here-a-there-a” part with the given noise.

    If “intro” is False, just makes the given animal noise. If “intro” is True, also includes that animal’s introduction (“And on this farm...”).
The Verses

def moo(intro):
    doIntro("cow", "moo", intro)

def snort(intro):
    doIntro("pig", "(snort)", intro)
    moo(False)

def neigh(intro):
    doIntro("horse", "neigh", intro)
    snort(False)

def cluck(intro):
    doIntro("chick", "cluck", intro)
    neigh(False)

• If "intro" is True, sings the cow verse, otherwise, just makes the cow noise.
• If "intro" is True, sings the pig verse (including the cow noises), otherwise, just makes the pig noise, and the cow noise.
• If "intro" is True, sings the horse verse (including the pig/cow noises), otherwise, just makes the horse/pig/cow noises.
• If "intro" is True, sings the chick verse (including all noises), otherwise, just makes all noises.

The Song

def OldMacdonald():
    moo(True)
    snort(True)
    neigh(True)
    cluck(True)

• Sings all four complete verses.

• Important thing to notice:
  • snort(True) means “set intro to True and begin executing the snort subroutine”.
  • While executing snort(True), it makes a call to moo(False)
  • moo(False) means “set intro to False and begin executing the moo subroutine.
  • No confusion---each subroutine has its own copy of “intro”!
Local Variables

- Unless otherwise specified, variable names refer to their values within the subroutine itself only.
- Changes to these local variables do not impact any of the calling subroutines.
- “What happens in LV stays in LV.”
Recursion...

- Since each subroutine call has its own context (local variables, “red arrow”), a subroutine can call itself.
- This is called recursion.
- First, some song examples, then more general...

Recursive Algorithm

def bottlesOfBeer(n):
    if n > 0:
        print str(n) + " bottles of beer on the wall."
        print str(n) + " bottles of beer."
        print "If one of those bottles should happen to fall."
        print str(n-1) + " bottles of beer on the wall."
        print bottlesOfBeer(n-1)

- bottlesOfBeer(99) prints a message, then calls
- bottlesOfBeer(98), which prints a message, then calls
- bottlesOfBeer(97), which prints a message, then calls...
- bottlesOfBeer(0), which does nothing.
Some people think recursion is scary, but it’s all around us: language, nature, toys, music, family trees, mathematical expressions.

If you like self-reference, you’ll love recursion!

---

def songNeverEnds():
    print "This is 'The Song That Doesn't End"
    print "Yes it goes on, and on my friend"
    print "Some people, started singing it"
    print "not knowing what it was"
    print "And they’ll just keep on singing it"
    print "forever just because..."
    songNeverEnds()
I’m now going to switch gears into a topic that bridges the gap between subroutines and compilers.

It will also give us a different view of logical expressions.

Expression Trees
Expression Trees: Bigger

- (True and True)

Expression Trees: Combined

- ((True and True) and (not True or False))
Trees and Subtrees

- Just as we can build more complex expressions out of simpler ones, we can build more complex trees out of simpler ones.
- An expression tree is:

```
ET =
False True
not and
ET
or or
ET ET ET ET
or or
```

Circular Definition?

- In a sense, this definition looks broken because it is defining an expression tree in terms of expression trees.
- This circularity is **safe**, because the definition also provides us a way to stop (True/False).
- It’s also **necessary**, because there’s an infinitely large set of possible expressions.
Some Tree Terminology

- **root**: The top node of the tree ("True", "False", "not", "and", or "or").
- **subtree**: A tree underneath the root.
- **left subtree**: The subtree to the left.
- **right subtree**: The subtree to the right.
- **leaf**: A tree with no subtrees of its own.
- **depth**: number of nodes between the root and the farthest leaf.

Evaluate Bottom Up

- Evaluate a tree whenever all its subtrees are evaluated.
Evaluate Top Down

- Start at the root.
- Ask a friend to evaluate the subtrees.
- Do the root.

Not That Different, Really

- In a sense, you have to start at the bottom.
- But, what recursion (self delegation?) does is let you focus on what happens at the top and the lower-down stuff just works itself out.
- Can make for much cleaner code.
evaluateTree Function

- Takes a tree as input, returns True/False.
- In some sense, very literal!
- But, uses recursion to handle the messy lower-level stuff.
- Somehow, extremely natural and extremely mind bending.

```python
def evaluateTree(tree):
    if root(tree) == 'True':
        return True
    if root(tree) == 'False':
        return False
    if root(tree) == 'not':
        val = evaluateTree(subtree(tree))
        return not val
    if root(tree) == 'and':
        v1 = evaluateTree(leftSubtree(tree))
        v2 = evaluateTree(rightSubtree(tree))
        return v1 and v2
    if root(tree) == 'or':
        v1 = evaluateTree(leftSubtree(tree))
        v2 = evaluateTree(rightSubtree(tree))
        return v1 or v2
```

Next Time

- Computers don’t understand high-level languages directly.
- How does a high-level language get executed by a machine language?
- We’ll convert an expression tree to ML$^3$ instructions, recursively!
- Hillis Ch. 3 Sections 4-5.