Lecture #3: Recursion

Go watch Inception! (Movie about recursion)

September 9, 2016
CS88 news

- Homework will have “Challenge problems”

- Project 1 coming soon!
  Site to know: [www.stackoverflow.com](http://www.stackoverflow.com)

- Enrollment up to about 50. We might open a 3rd section.
Computational Concepts today

- Variable Scope (also: see reading)
- Recursion
Remember: Functions

- Generalizes an expression or set of statements to apply to lots of instances of the problem
- A function should do one thing well

```python
def concat(str1, str2):
    return str1+str2;

concat(“Hello”, “World”)
```
Variable Scope

When an input is passed to a function, what does the function actually get?

– Internal variables get a copy of input values, with the exception of mutable objects

Local variables only exist within the function in which they are defined

– The variables cease to exist when the function ends
– The scope of a variable is the part(s) of code where that variable name binding is valid (i.e. where it exists)
Variable Scope (Python)
Variable Scope: Example I

```
i = 1

def foo():
    i = 5
    print(i, 'in foo()')

print(i, '=global')

foo()
```

Output?

1=global
5 in foo()
a_var = 'global value'

def a_func():
    global a_var
    a_var = 'local value'
    print(a_var, '[ a_var inside a_func() ]')

print(a_var, '[ a_var outside a_func() ]')
a_func()
print(a_var, '[ a_var outside a_func() ]')

Output?

global value [ a_var outside a_func() ]
local value [ a_var inside a_func() ]
local value [ a_var outside a_func() ]
Recursion

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/

adjective

characterized by recurrence or repetition, in particular.

- 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
Reminder: Iteration

<initialization statements>
for <variables> in <sequence expression>:
  <body statements>

<rest of the program>

<initialization statements>
while <predicate expression>:
  <body statements>

<rest of the program>

[ <expr with loop var> for <loop var> in <sequence expr > ]
Iteration vs Recursion

For loop:
```python
def sum(n):
    s=0
    for i in range(0,n+1):
        s=s+i
    return s
```
Iteration vs Recursion

While loop:

```python
def sum(n):
    s=0
    i=0
    while i<n:
        i=i+1
        s=s+i
    return s
```
Recursion:

```python
def sum(n):
    if n==0:
        return 0
    return n+sum(n-1)
```
Recursion: Pattern

1. Test for simple “base” case
2. Solution in simple “base” case
3. Assume recursive solution to simpler problem
4. Transform sol’n of simpler problem into full sol’n

• Linear recursion

```python
def sum(n):
    if n == 0:
        return 0
    return n + sum(n-1)
```
Why does it work?

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

# \text{sum}(3) \Rightarrow 3 + \text{sum}(2)

# \Rightarrow 3 + \text{sum}(2) + \text{sum}(1)

# \Rightarrow 3 + \text{sum}(2) + \text{sum}(1) + \text{sum}(0)

# \Rightarrow 3 + \text{sum}(2) + \text{sum}(1) + 0

# \Rightarrow 3 + \text{sum}(2) + 1

# \Rightarrow 3 + 3

# \Rightarrow 6
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 mathematical 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

```python
def sum(n):
    if n==0:
        return 0
    return n+sum(n-1)
```
Sanity Check…

• Recursion is more powerful than Iteration (i.e., loops)

  a) more powerful than
  b) just as powerful as
  c) less powerful than
Why Recursion?

• “After Abstraction, Recursion is probably the 2\textsuperscript{nd} biggest idea in this course”
• “It’s tremendously useful when the problem is self-similar”
• “It’s no more powerful than iteration, but often leads to more concise & better code”
• “It’s more ‘mathematical’”
• “It embodies the beauty and joy of computing”
• …
Why Recursion? More Reason

- 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.
Recursion (unwanted)
List all items on your hard disk

- Files
- Folders contain
  - Files
  - Folders

Recursion!
def listfiles(directory):
    content = [os.path.join(directory, x) for x in os.listdir(directory)]

dirs = sorted([x for x in content if os.path.isdir(x)])
files = sorted([x for x in content if os.path.isfile(x)])

for d in dirs:
    print d
    listfiles(d)

for f in files:
    print f

Iterative version about twice as much code and much harder to think about.
Example II

Sort the numbers in a list.

Hidden recursive structure: Decision tree!
Tree Recursion makes Sorting Efficient

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

Initial array: [3, 3, 1, 4, 5, 4, 3, 2, 1, 17]

Steps of QuickSort:

1. [3, 1, 3, 2, 1]
   - [3, 1, 3, 2]
   - [1]
2. [1, 3, 2, 1]
   - [1, 3, 2]
   - []
3. [1, 1, 2, 3]
   - [1, 1, 2]
   - [3]
4. [1, 1, 2, 3, 3]
   - [1, 1, 2, 3]
   - [3, 3]
5. [1, 1, 2, 3, 3, 4, 4, 5, 17]
Questions?

There's a little green house
And inside the little green house
There is a little brown house
And inside the little brown house
There is a little yellow house
And inside the little yellow house
There is a little white house
And inside the little white house
There is a little red heart
Wrong and looking.

4! = n • (n - 1)!

Mother Goose Rhyme:
Myself
As I walked by myself
And talked to myself,
Myself said unto me:
"Look to thyself,
For no body cares for thee.
I answered myself,
And said to myself
In the stillness reported:
"Look to thyself,
Or not look to thyself,
The same thing will be."