Lecture #4: Higher Order Functions

Hackers steal medical data of US Olympic stars

Administrative issues

• Concurrent Enrollment: Assume you are in and work on the class!

• Data 8 is a requirement. You need to have taken c8 or do it concurrently.

• If you can’t get into data8, try CS10.
Computational Concepts today

• More on Recursion
• Runtime

• 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
More on Recursion

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

- The sum of no numbers is zero
- The sum of $1^2$ through $n^2$ is $n^2$ plus the sum of $1^2$ through $(n-1)^2$
Recap: 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:
        return accum  # Base Case
    else:
        return sum_up_squares(i+1, n, accum+i**2)  # Tail Recursive Case

>>> sum_up_squares(1,3,0)
14
```
How much ???

- Time is required to compute \( \text{sum\_of\_squares}(n) \)?
  - Recursively?
  - Iteratively?

- Space is required to compute \( \text{sum\_of\_squares}(n) \)?
  - Recursively?
  - Iteratively?

- Count the frames...
- Recursive is linear, iterative is constant!
- And what about the order of evaluation?

Linear proportional to \( cn \) for some \( c \)
Recap: Defining Functions

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

```
def <function name> (<argument list>) :
   expression
   return
```
Recap: Data or Code?
Higher Order Functions

• Functions that operate on functions
• A function

```python
def odd(x):
    return (x%2==1)

>>> odd(3)
True
```

• A function that takes a function arg

```python
def filter(fun, s):
    return [x for x in s if fun(x)]

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

Why is this not ‘odd’?
Higher Order Functions (cont)

- A function that returns (makes) a function

```python
def leq_maker(c):
    def leq(val):
        return val <= c
    return leq

>>> 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]
>>>
One more example

• 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])
map(function_to_apply, list_of_inputs)
Applies function to each element of the list

filter(condition, list_of_inputs)
Returns a list of elements for which the condition is true

reduce(function, list_of_inputs)
Reduces the list to a result, given the function
Recursion with Higher Order Fun

\[
\text{def } \text{map}(f, s):
\]

\[
\text{if } \not\text{ s}:
\]

\[
\text{else:}
\]

Base Case

Recursive Case

\[
\text{def } \text{square}(x):
\]

\[
\text{return } x**2
\]

>>> \text{map} (\text{square}, [2,4,6])
[4, 16, 36]

- Divide and conquer
Using HOF to preserve interface

```python
def sum_of_squares(n):
    def sum_upper(i, accum):
        if i > n:
            return accum
        else:
            return sum_upper(i+1, accum + i*i)
    return sum_upper(1,0)
```

- 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
Recap: Quicksort

• 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]
```
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]
How much ???

• Time is required to compute quicksort(s)?

• Space is required?

• Name of this recursion scheme?
  – Tree recursion

Logarithmic to len(s)
c*\log(len(s)) for some c
Questions?