CS61A Lecture 4

Higher Order Functions

Tom Magrino and Jon Kotker
UC Berkeley EECS
June 21, 2012
AI System Automates Consumer Gripees

By Ben Rooney

An artificial intelligence service that automatically resolves consumer disputes and can even engage in negotiations was displayed at Le Web on Wednesday.

Cognicor, a spinoff company from the Barcelona Institute of Artificial Intelligence, is aimed at businesses that get a huge volume of low-value complaints that need to be resolved.

Write the function `filtered_count` that takes a number, `n`, and a function, `pred*`, and prints all the numbers 1 to `n` for which `pred` returns True.

```python
def filtered_count(n, pred):
    k = 1
    while k <= n:
        if pred(k):
            print(k)
        k = k + 1
```

*`pred` stands for *predicate*, a function that only returns True or False.*
Write the function `filtered_count` that takes a number, `n`, and a function, `pred*`, and prints all the numbers 1 to `n` for which `pred` returns `True`.

```python
def filtered_count(n, pred):
    k = 1
    while k <= n:
        if pred(k):
            print(k)
        k = k + 1
```

*`pred` stands for *predicate*, a function that only returns `True` or `False`.\*
TODAY

Defining functions inside functions.
Returning functions from functions.
A Friendly Greek Letter.
Python allows us to make functions *inside* other functions. This is useful for making helper functions.

$$f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c}$$

```python
def funny_function(a, b, c):
    a_funny_b = (a * b) + (b / a) + (a ** b)
    a_funny_c = (a * c) + (c / a) + (a ** c)
    return a_funny_b / a_funny_c
```
Python allows us to make functions inside other functions. This is useful for making “helper” functions help us avoid repetition!

\[
f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c}
\]

```python
def funny_function(a, b, c):
    def a_funny(b):
        return (a * b) + (b / a) + (a ** b)
    return a_funny(b) / a_funny(c)
```
LOCAL (HELPER) FUNCTIONS

How does this work?

\[
f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c}
\]

def funny_function(a, b, c):
    def a_funny(b):
        return (a * b) + (b / a) + (a ** b)
    return a_funny(b) / a_funny(c)

funny_function(2, 4, 6)
LOCAL (HELPER) FUNCTIONS

How does this work?

\[
f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c}
\]

def funny_function(a, b, c):
    def a_funny(b):
        return (a * b) + (b / a) + (a ** b)
    return a_funny(b) / a_funny(c)

funny_function(2, 4, 6)
LOCAL (HELPER) FUNCTIONS

How does this work?

\[
f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c}
\]

def funny_function(\texttt{2, b, c}):
    def a_funny(b):
        return (\texttt{2 * b} + (b / \texttt{2}) + (\texttt{2}** b)
    return a_funny(b) / a_funny(c)

funny_function(\texttt{2, 4, 6})
How does this work?

```
def funny_function(a, b, c):
    def a_funny(b):
        return (2 * b) + (b / 2) + (2 ** b)
    return a_funny(b) / a_funny(c)

funny_function(2, 4, 6)
```
**LOCAL (HELPER) FUNCTIONS**

How does this work?

\[
f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c}
\]

```python
def funny_function(2, 4, 6):
    def a_funny(b):
        return (2 * b) + (b / 2) + (2 ** b)
    return a_funny(4) / a_funny(6)

funny_function(2, 4, 6)
```
LOCAL (HELPER) FUNCTIONS

How does this work?

\[ f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c} \]

def funny_function(2, 4, 6):
    def a_funny(b):
        return (2 * b) + (b / 2) + (2 ** b)
    return a_funny(4) / a_funny(6)

funny_function(2, 4, 6)
LOCAL (HELPER) FUNCTIONS

How does this work?

\[ f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c} \]

```python
def funny_function(2, 4, 6):
    def a_funny(b):
        return (2 * b) + (b / 2) + (2 ** b)
    return a_funny(4) / a_funny(6)

funny_function(2, 4, 6)
```
LOCAL (HELPER) FUNCTIONS

How does this work?

\[ f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c} \]

```python
def funny_function(a, b, c):
    def a_funny(c):
        return (2 * c) + (c / 2) + (2 ** c)
    return 26 / a_funny(c)
funny_function(2, 4, 6)
```
LOCAL (HELPER) FUNCTIONS

How does this work?

\[
f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c}
\]

def funny_function(2, 4, 6):
    def a_funny(b):
        return (2 * b) + (b / 2) + (2 ** b)
    return 26 / a_funny(6)

funny_function(2, 4, 6)
LOCAL (HELPER) FUNCTIONS

How does this work?

$$f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c}$$

def funny_function(2, 4, 6):
    def a_funny(6):
        return (2 * 6) + (6 / 2) + (2 ** 6)
    return a_funny(4) / a_funny(6)

funny_function(2, 4, 6)
LOCAL (HELPER) FUNCTIONS

How does this work?

\[ f(a, b, c) = \frac{ab + \left(\frac{b}{a}\right) + a^b}{ac + \left(\frac{c}{a}\right) + a^c} \]

def funny_function(2, 4, 6):
    def a_funny(b):
        return (2 * b) + (b / 2) + (2 ** b)
    return 26 / 79

funny_function(2, 4, 6) → 0.329...
ANNOUNCEMENTS

• You will be placed into study/exam groups in discussion section today.
  – Email your TA ASAP if you can’t make section so they can assign you a group.

• Homework 1 is due tomorrow night.

• Homework 2 is out and due Tuesday night.

• Project 1 is out and due the night of 6/29.
PROBLEM: MAKING ADDERS

def add_1_to_twice(n):
    return (2*n)+1

def add_2_to_twice(n):
    return (2*n)+2

def add_3_to_twice(n):
    return (2*n)+3

...

def add_65536_to_twice(n):
    return (2*n)+65536

def add_65537_to_twice(n):
    return (2*n)+65537

...
**PROBLEM: MAKING ADDERS**

```python
def add_1_to_twice(n):
    return (2*n)+1

def add_2_to_twice(n):
    return (2*n)+2

def add_3_to_twice(n):
    return (2*n)+3

... def add_65536_to_twice(n):
    return (2*n)+65536

def add_65537_to_twice(n):
    return (2*n)+65537
...```

*There has to be a better way!*
**Problem: Making Adders**

```python
def add_1_to_twice(n):
    return (2*n)+1

def add_2_to_twice(n):
    return (2*n)+2

def add_3_to_twice(n):
    return (2*n)+3

... def add_65536_to_twice(n):
    return (2*n)+65536

def add_65537_to_twice(n):
    return (2*n)+65537
...```

Can we generalize?
PROBLEM: MAKING ADDERS

http://farm8.staticflickr.com/7014/6766279501_4ddeb9193c_z.jpg
**Problem: Making Adders**

```python
def make_adder_for(m):
    def add_m_to_twice(n):
        return (2*n)+m
    return add_m_to_twice

add_1_to_twice = make_adder_for(1)
add_2_to_twice = make_adder_for(2)
add_3_to_twice = make_adder_for(3)
...
add_65536_to_twice = make_adder_for(65536)
add_65537_to_twice = make_adder_for(65537)
...
```
PROBLEM: MAKING ADDERS

```python
def make_adder_for(m):
    def add_m_to_twice(n):
        return (2 * n) + m
    return add_m_to_twice
```
PRACTICE: MAKING SUMMATION...ERS

I want to make specialized sums for later use. Complete the definition below:

```python
def make_summationer(term):
    def summation(n):
        k, sum = 1, 0
        while k <= n:
            sum = sum + term(k)
            k = k + 1
        return sum
    return summation

return summation
```
I want to make specialized sums for later use. Complete the definition below:

```python
def make_summationer(term):
    def summation(n):
        k, sum = 1, 0
        while k <= n:
            sum = sum + term(k)
            k = k + 1
        return sum
    return summation
```

**Practice: Making Summation...ers**
**Practice: Making Adders for Summations**

Say I wanted to get the sum of the numbers 4 through 17. How can I do this in one line using summation and make_adder?

```python
def make_adder(n):
    def add_n(m):
        return n + m
    return add_n
def summation(n, term):
    sum, k = 0, 1
    while k <= n:
        sum = sum + term(k)
        k = k + 1
    return sum

>>> summation(14, make_adder(3))
147
```

147
PRACTICE: MAKING ADDERS FOR SUMMATIONS

Say I wanted to get the sum of the numbers 4 through 17. How can I do this in one line using summation and make_adder?

```python
def make_adder(n):
    def add_n(m):
        return n + m
    return add_n

def summation(n, term):
    sum, k = 0, 1
    while k <= n:
        sum = sum + term(k)
        k = k + 1
    return sum

>>> summation(14, make_adder(3))
147
```
What if we don’t want to give a name to our function? Seems silly that we have to come up with a name that we’re only using once in our program (in the return statement).

```python
def make_adder(n):
    def add_n(m):
        return n + m
    return add_n
```

Can I do the same thing without providing a name?

– Yes!
In Computer Science, we traditionally call anonymous function values “lambda functions.”

In Python a lambda function has the form:

\[
\text{lambda } \langle \text{arguments} \rangle: \langle \text{expression} \rangle
\]

For example we could have done:

```python
def make_adder(n):
    return lambda m: n + m
```
ANONYMOUS FUNCTIONS

Translating between lambdas and defined functions.

\[
<\text{name}> = \text{lambda } <\text{arguments}> : <\text{expression}>
\]

\[
\text{def } <\text{name}>(<\text{arguments}>): \\
\text{return } <\text{expression}>
\]
ANONYMOUS FUNCTIONS

Translating between lambdas and defined functions.

\[ \text{square} = \lambda x: x \cdot x \]

\[ \text{def square}(x): \]
\[ \quad \text{return } x^2 \]
Anonymous Functions

Translating between lambdas and defined functions.

\[
\text{foo} = \lambda a, x, b: (a \times x) + b
\]

```python
def foo(a, x, b):
    return (a * x) + b
```
ANONYMOUS FUNCTIONS

Translating between lambdas and defined functions.

```python
def make_adder(n):
    return lambda m: m + n

def make_adder(n):
    def add_n(m):
        return m + n
    return add_n
```

35
Key points about lambda in Python:

– A lambda in Python *is an expression*, so you can use them anywhere you can use any other expression in Python.

– You can only use a single *expression* in the body of a lambda function.

– A lambda function always *returns* the value of the body expression.

– Lambdas should be used *sparingly*, only for very simple functions. Otherwise the code can quickly become *unreadable*. 
PRACTICE: ANONYMOUS FUNCTIONS

What would the following expression evaluate to?

```python
>>> summation(3, lambda x: x + 5)
```

21
PRACTICE: ANONYMOUS FUNCTIONS

What would the following expression evaluate to?

```python
>>> summation(3, lambda x: x + 5)
21
```
CONCLUSION

• Functions can be defined inside other functions!
• Functions can return functions.
• Python has lambda functions for creating anonymous functions.
• Next Time: More practice with HOFs, and practical applications!
EXTRAS: MAKING MY ADDER AND USING IT TOO!

We can add 3 and 5, using only the values make_adder, 3, and 5.

\[
\text{make\_adder}(3) \ ( \ 5 \ )
\]

Where \( n \) is equal to 3