CS61A Lecture 3
Higher Order Functions

Jon Kotker and Tom Magrino
UC Berkeley EECS
June 20, 2012

Computer Science in the News
The (Literal) Evolution of Music

- Scientists from Imperial College London created a computer program powered by Darwinian natural selection.
  Theory that cultural changes in language, art, and music evolve like living things do, since consumers *choose* what is "popular".
- Program would produce loops of random sounds and analyze opinions of musical consumers. The top loops were then "mated".
- "DarwinTunes" (darwintunes.org) has evolved through at least 2513 generations.

Source: http://www3.imperial.ac.uk/newsandeventspggrp/imperialcollege/newssummary/news_19-6-2012-9-59-1

Review

```
def foo(n):
    k = 2
    while k < n:
        if n % k == 0:
            return False
        k = k + 1
    return True

foo(7)? foo(9)? foo(1)?
```

TRIVIA

foo is a *metasyntactic variable*, or a commonly used nonsense name to denote an arbitary or temporary thing.

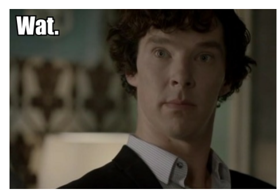
Other examples you may see are bar, garply, and baz. Sometimes, om and nom. Or even herp and derp.

This deserves a better name though. Any suggestions?

Today

Domain and range of functions

... and feeding functions to functions



Domain and Range

Domain is the *set of values* that the function is *defined for*.

```
square(n):
    return n*n
```

Range (or image) is the *set of values* that the function *returns*.

Domain of `square`: All real *numbers*.
Range of `square`: All non-negative real *numbers*.

Domain and Range

Remember the domain and range of the functions that you are writing and using!

Many bugs arise because programmers forget what a function can and cannot work with.

The domain and range are *especially important* when working with higher order functions.

Problem: Finding Squares

```
>>> square_of_0 = 0*0
>>> square_of_1 = 1*1
>>> square_of_2 = 2*2
>>> square_of_3 = 3*3
...
>>> square_of_65536 = 65536*65536
>>> square_of_65537 = 65537*65537
...
```


Problem: Finding Squares

```
>>> square_of_0 = 0*0
>>> square_of_1 = 1*1
>>> square_of_2 = 2*2
>>> square_of_3 = 3*3
...
>>> square_of_65536 = 65536*65536
>>> square_of_65537 = 65537*65537
...
```

There has to be a better way!

Problem: Finding Squares

```
>>> square_of_0 = 0*0
>>> square_of_1 = 1*1
>>> square_of_2 = 2*2
>>> square_of_3 = 3*3
...
>>> square_of_65536 = 65536*65536
>>> square_of_65537 = 65537*65537
...
```

Can we generalize?

Finding Squares: Generalization

THE SQUARE OF A NUMBER

```
def square(n):
    return n * n
```

IS THE NUMBER MULTIPLIED BY ITSELF

Problem: Sums of Series

```
def sum_of_n_squares(n):
    '''Returns 1² + 2² + 3² + ... + n².'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + square(k), k + 1
    return sum

def sum_of_n_cubes(n):
    '''Returns 1³ + 2³ + 3³ + ... + n³.'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + cube(k), k + 1
    return sum
```


Problem: Sums of Series

```
from math import sin, sqrt
def sum_of_n_sines(n):
    '''Returns sin(1) + sin(2) + sin(3) + ... + sin(n).'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + sin(k), k + 1
    return sum

def sum_of_n_sqrts(n):
    '''Returns sqrt(1) + sqrt(2) + ... + sqrt(n).'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + sqrt(k), k + 1
    return sum
```

and so on…

Problem: Sums of Series

```
from math import sin, sqrt
def sum_of_n_sines(n):
    '''Returns sin(1) + sin(2) + sin(3) + ... + sin(n).'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + sin(k), ...
    return sum

de..._...ts(n):
    '''Returns sqrt(1) + sqrt(2) + ... + sqrt(n).'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + sqrt(k), k + 1
    return sum
```

There has to be a better way!

and so on…

13

Problem: Sums of Series

```
def sum_of_n_squares(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + square(k)
        k = k + 1
    return sum

def sum_of_n_sines(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + sin(k)
        k = k + 1
    return sum

def sum_of_n_cubes(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + cube(k)
        k = k + 1
    return sum

def sum_of_n_sqrts(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + sqrt(k)
        k = k + 1
    return sum
```

Can we generalize?

14

Announcements

- HW2 is released, due **Tuesday, June 26**.
- Discussions earlier held in 320 Soda will now be held in **310 Soda** for the next six weeks. We will move back to 320 Soda for the last two weeks.
- Groups for studying and midterms will be assembled on **Thursday, June 21** in discussion section.

15

Evaluation of Primitive Expressions

```
>>> 3
3
>>> x = 5
>>> x
5
>>> 'CS61A'
'CS61A'
```

evaluates to

Statement

evaluates to

evaluates to

16

Function Names

```
>>> square
<function square at 0x0000000002279648>
```

evaluates to

Address where the function is *stored* in the computer. It is not fixed.

Python's representation of

```
square(n):
    return n*n
```

square is the name of the function, or "machine", that squares its input

17

Function Names

```
>>> square
<function square at 0x0000000002279648>
>>> cube
<function cube at 0x00000000022796C8>
>>> max
<built-in function max>
```

Function names are *primitive expressions* that *evaluate* to the corresponding "machines"

18

Using Primitive Expressions

We have seen primitive expressions used as arguments to functions:

```
>>> square(3)
9
>>> x = 4
>>> square(x)
16
```

Can we then use function names as arguments?

19

http://boredommagnified.files.wordpress.com/2011/07/challenge-accepted.jpg

20

Problem: Sums of Series

```
def sum_of_n_squares(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + square(k)
        k = k + 1
    return sum

def sum_of_n_sines(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + sin(k)
        k = k + 1
    return sum

def sum_of_n_cubes(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + cube(k)
        k = k + 1
    return sum

def sum_of_n_sqrts(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + sqrt(k)
        k = k + 1
    return sum
```

21

Functions as Arguments

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

def sum_of_n_squares(n):
    return summation(n, square)
```

term(1) + term(2) + ... + term(n)

22

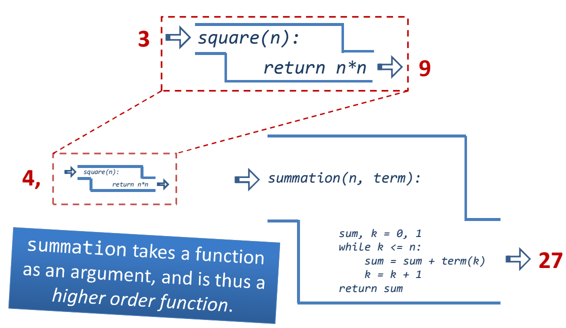
Functions as Arguments

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

def sum_of_n_squares(n):
    return summation(n, square)
```

23

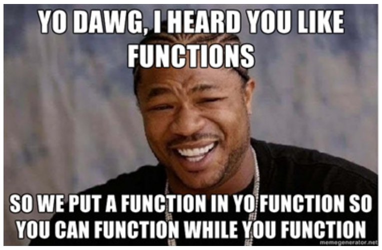
Functions as Arguments



summation takes a function as an argument, and is thus a higher order function.

24

Higher Order Functions



http://cdn.memegenerator.net/instances/400x/13712469.jpg

Higher Order Functions: Practice

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

def sum_of_n_cubes(n):
    return summation(n, ______________)

def sum_of_n_sines(n):
    return summation(n, ______________)

def sum_of_n_positive_ints(n):
    return summation(n, ______________)
```


Higher Order Functions: Practice

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

def sum_of_n_cubes(n):
    return summation(n, lambda x: x ** 3)

def sum_of_n_sines(n):
    return summation(n, sin)

def sum_of_n_positive_ints(n):
    return summation(n, lambda x: x)
```


Higher Order Functions

Functions that can take other functions as arguments are considered *higher order functions*.

The *domains* of these functions now include *other functions*.

Functions can be treated as ***data***!

Higher Order Functions: Aside

Why "higher order"?

First-order functions only take non-function values, like numbers and strings, as arguments.

Second-order functions can take first-order functions as arguments.

Third-order functions can take second-order functions as arguments.

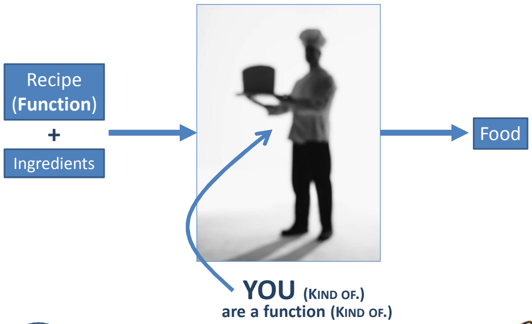
… and so on.

HOFs Elsewhere

$$\int_a^b f(x)dx$$

Integral $\int$ is a **function** that takes three arguments: lower limit *a*, upper limit *b*, and a **function** *f*

HOFs Elsewhere



HOFs Elsewhere

Basketball

Playing

NOUN
(Data)

VERB
(Function)

Interests: Playing Basketball

NOUN (“Gerund”)
(Function as data)

Functions as Arguments

```
def summation2(n, term, next):
    sum, k = 0, 1
    while k <= n:
        sum = sum + term(k)
        k = next(k)
    return sum
```


Functions as Arguments: Practice

```
def summation2(n, term, next):
    sum, k = 0, 1
    while k <= n:
        sum = sum + term(k)
        k = next(k)
    return sum

def summation(n, term):
    return summation2(n, term,
                      _______________)
```


Functions as Arguments: Practice

```
def summation2(n, term, next):
    sum, k = 0, 1
    while k <= n:
        sum = sum + term(k)
        k = next(k)
    return sum

def summation(n, term):
    return summation2(n, term,
                      lambda x: x + 1)
```


Higher Order Functions: Practice

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

def sum_of_n_even(n):
    ‘‘‘Returns the sum of the first n even numbers.’’’
    return summation(n, ________________)

def sum_of_n_odd(n):
    ‘‘‘Returns the sum of the first n odd numbers.’’’
    return summation(n, ________________)

def sum_of_n_starting_from_m(m, n):
    ‘‘‘Returns the sum of the first n numbers starting from m.’’’
    return summation(n, ________________)
```

Higher Order Functions: Practice

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

def sum_of_n_even(n):
    '''Returns the sum of the first n even numbers.'''
    return summation(n, lambda x: 2 * x)

def sum_of_n_odd(n):
    '''Returns the sum of the first n odd numbers.'''
    return summation(n, lambda x: 2 * x + 1)

def sum_of_n_starting_from_m(m, n):
    '''Returns the sum of the first n numbers starting from m.'''
    return summation(n, lambda x: x + m)
```

37

Higher Order Functions: Practice

The value of the *derivative* of a function f at a point x can be approximated as

$$\frac{f(x + \Delta x) - f(x)}{\Delta x}$$

if Δx is really small.

Write the function `derivative` that takes as arguments a function `f`, a point `x`, and a really small value `delta_x`, and returns the approximate value of the derivative at x.

```
def deriv(f, x, delta_x):
    return _______________
```

38

Higher Order Functions: Practice

The value of the *derivative* of a function f at a point x can be approximated as

$$\frac{f(x + \Delta x) - f(x)}{\Delta x}$$

if Δx is really small.

Write the function `derivative` that takes as arguments a function `f`, a point `x`, and a really small value `delta_x`, and returns the approximate value of the derivative at x.

```
def deriv(f, x, delta_x):
    return (f(x + delta_x) - f(x)) / delta_x
```

39

Conclusion

- It is important to remember the domain and range of functions!
- Functions can take functions as arguments.
- ***Preview***: Functions can also *return* functions.

40