61A Lecture 6

Friday, September 7

Lambda Expressions

```
>>> ten = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

```
>>> square = lambda x: x * x
```

Notice: no "return"

A function

with formal parameter x

and body "return x * x"

Must be a single expression

```
>>> square(4)
16
```

Lambda expressions are rare in Python, but important in general

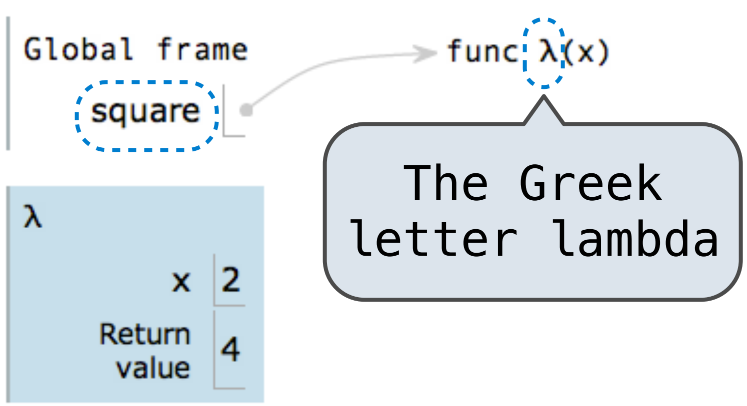
Lambda Expressions Versus Def Statements

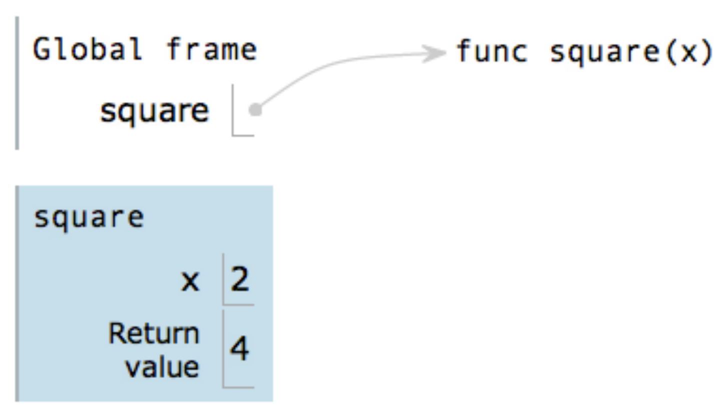
```
square = lambda x: x * x
```

VS

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

- Both create a function with the same arguments & behavior
- Both of those functions are associated with the environment in which they are defined
- Both bind that function to the name "square"
- Only the def statement gives the function an intrinsic name





Function Currying

```
def make_adder(n):
    return lambda k: n + k
```

```
>>> make_adder(2)(3)
5
>>> add(2, 3)
5
```

There's a general relationship between these functions

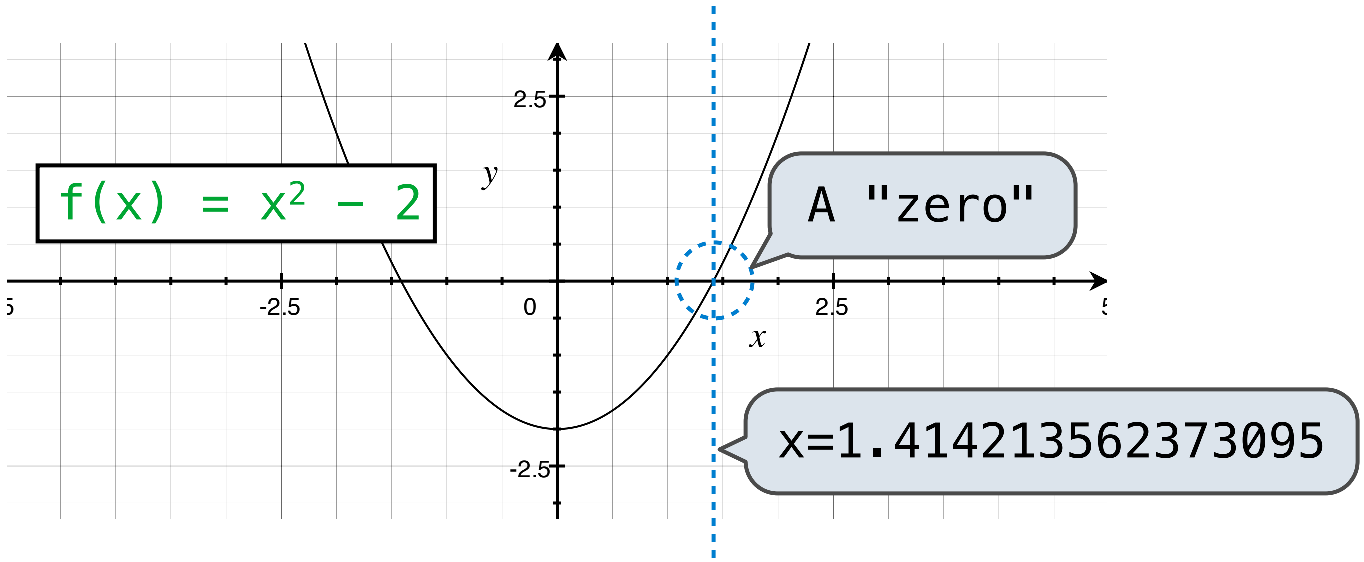
Currying: Transforming a multi-argument function into a single-argument, higher-order function.

Fun Fact: Currying was discovered by Moses Schönfinkel and later re-discovered by Haskell Curry.

Schönfinkeling?

Newton's Method Background

Finds approximations to zeroes of differentiable functions



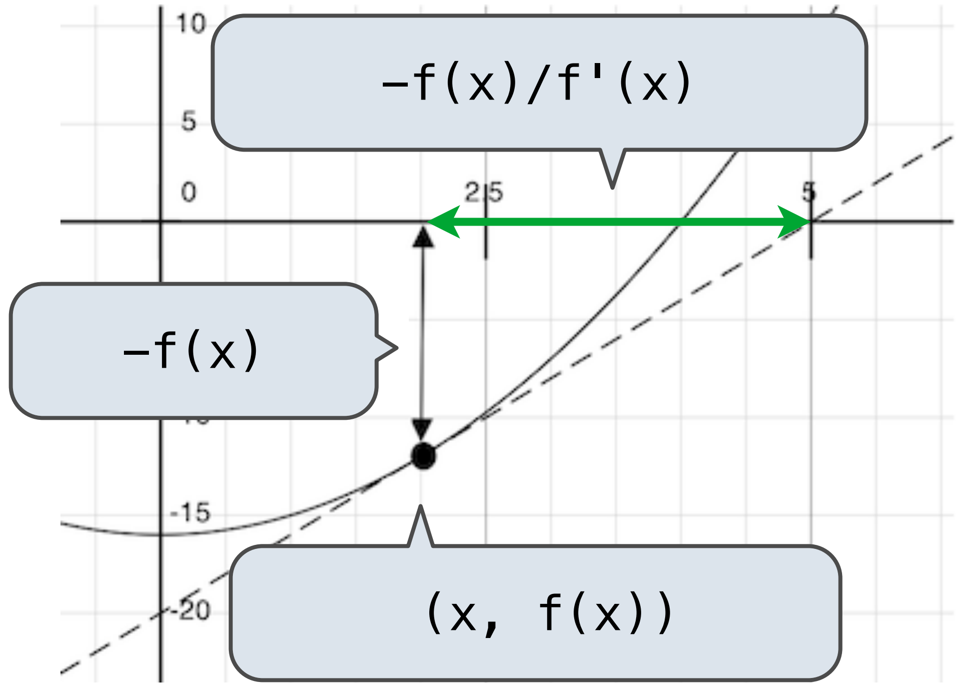
Application: a method for (approximately) computing square roots, using only basic arithmetic.

The positive zero of $f(x) = x^2 - a$ is $\sqrt{a}$

Newton's Method

Begin with a function f and

an initial guess x



1. Compute the value of f at the guess: f(x)

2. Compute the derivative of f at the guess: f'(x)

3. Update guess to be: $x - \frac{f(x)}{f'(x)}$

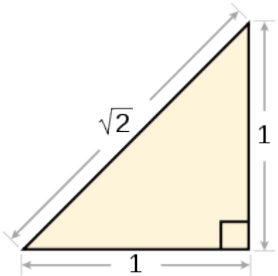
Visualization of Newton's Method

(Demo)

http://en.wikipedia.org/wiki/File:NewtonIteration_Ani.gif

Using Newton's Method

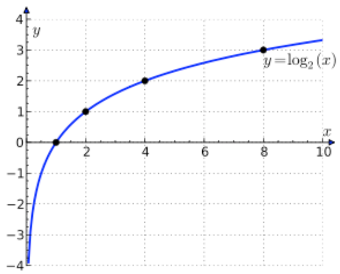
How to find the **square root** of 2?




```
>>> f = lambda x: x*x - 2
>>> find_zero(f)
1.4142135623730951
```

$f(x) = x^2 - 2$

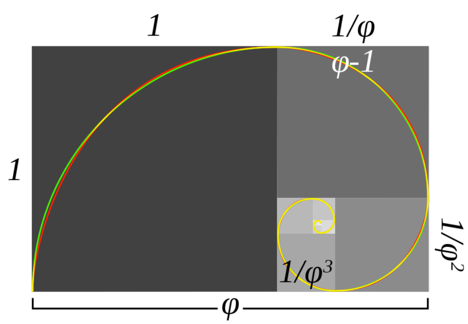
How to find the **log base 2** of 1024?




```
>>> g = lambda x: pow(2, x) - 1024
>>> find_zero(g)
10.0
```

$g(x) = 2^x - 1024$

What number is one less than its square?




```
>>> h = lambda x: x*x - (x+1)
>>> find_zero(h)
1.618033988749895
```

$h(x) = x^2 - (x+1)$

Special Case: Square Roots

How to compute square_root(a)

Idea: Iteratively refine a guess x about the square root of a

Update: $$x = \frac{x + \frac{a}{x}}{2}$$

Babylonian Method

Implementation questions:

What *guess* should start the computation?

How do we know when we are finished?

Special Case: Cube Roots

How to compute cube_root(a)

Idea: Iteratively refine a guess x about the cube root of a

Update:

$$x = \frac{2 \cdot x + \frac{a}{x^2}}{3}$$

Implementation questions:

What *guess* should start the computation?

How do we know when we are finished?

Iterative Improvement

(Demo)

```
def golden_update(guess):
    return 1/guess + 1
```

```
def golden_test(guess):
    return guess * guess == guess + 1
```

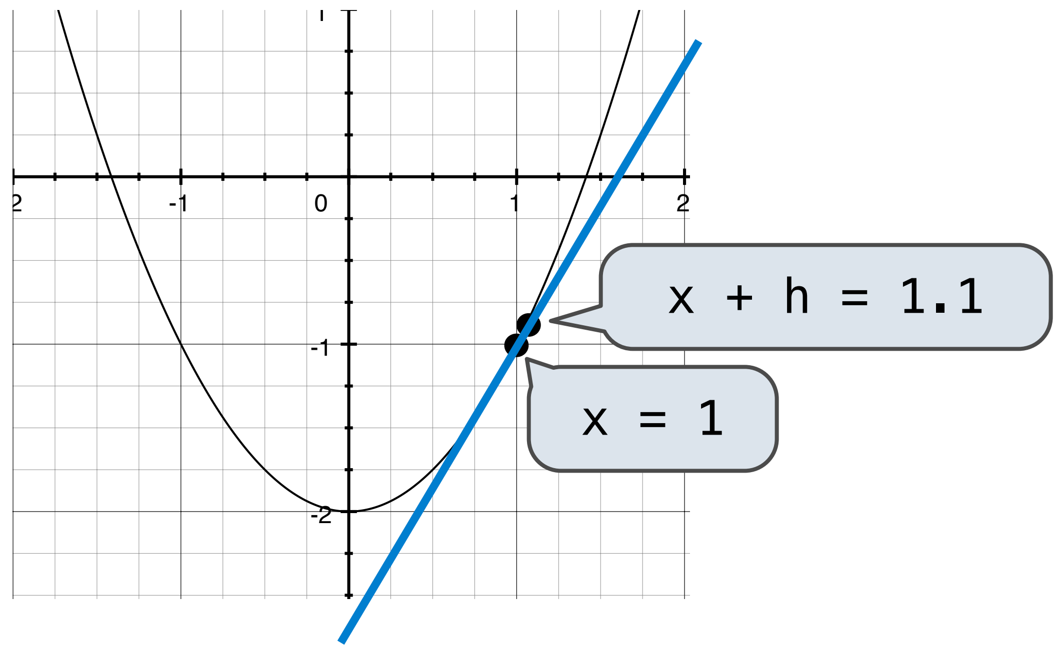
```
def iter_improve(update, done, guess=1, max_updates=1000):
    """Iteratively improve guess with update until done returns a true value.

    guess -- An initial guess
    update -- A function from guesses to guesses; updates the guess
    done -- A function from guesses to boolean values; tests if guess is good

    >>> iter_improve(golden_update, golden_test)
    1.618033988749895
    """
    k = 0
    while not done(guess) and k < max_updates:
        guess = update(guess)
        k = k + 1
    return guess
```

Derivatives of Single-Argument Functions

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



(Demo)

http://en.wikipedia.org/wiki/File:Graph_of_sliding_derivative_line.gif

Approximating Derivatives

(Demo)

Implementing Newton's Method

```
def newton_update(f):
    """Return an update function for f using Newton's method."""
    def update(x):
        return x - f(x) / approx_derivative(f, x)
    return update
```

Could be replaced with the exact derivative

```
def approx_derivative(f, x, delta=1e-5):
    """Return an approximation of the derivative of f at x."""
    df = f(x + delta) - f(x)
    return df/delta
```

Limit approximated by a small value

```
def find_root(f, guess=1):
    """Return a guess of a zero of the function f, near guess.

    >>> from math import sin
    >>> find_root(lambda y: sin(y), 3)
    3.141592653589793
    """
    return iter_improve(newton_update(f), lambda x: f(x) == 0, guess)
```

Definition of a function zero