Lambda Expressions

An expression: this one evaluates to a number

Also an expression: evaluates to a function

A function with formal parameter x and body "return x * x" must be a single expression

Lambda expressions are rare in Python, but important in general

Lambda Expressions Versus Def Statements

\[
\begin{align*}
\text{square} &= \lambda x: x \times x \\
\text{def square}(x): &\quad \text{return } x \times x
\end{align*}
\]

- Both create a function with the same arguments & body
- 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

\[
\text{def make_adder}(n): \quad \text{return } \lambda k: n + k
\]

\[
\begin{align*}
\text{>>> make_adder}(2)(3) &\quad 5 \\
\text{>>> add}(2, 3) &\quad 5
\end{align*}
\]

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.

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 \( y = x^2 - a \) is \( \sqrt{a} \)

Newton’s Method

Begin with a function \( f \) and an initial guess \( x \)

1. Compute the value of \( f \) at guess: \( f(x) \)
2. Compute the derivative of \( f \) at guess: \( f'(x) \)
3. Update guess \( x \) to be: \( x - \frac{f(x)}{f'(x)} \)
Visualization of Newton's Method

(Animation)


Using Newton's Method

How to find the square root of 2?

```python
>>> f = lambda x: x**2 - 2
>>> find_root(f, 1)
```

How to find the log base 2 of 1024?

```python
>>> g = lambda x: pow(2, x) - 1024
>>> find_root(g, 1)
```

What number is one less than its square?

```python
>>> h = lambda x: x**2 - (x+1)
>>> find_root(h, 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: Square 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**

```python
def golden_update(guess):
    return guess * guess + guess

def golden_text(guess):
    return "guess + guess"

def iter_improve(update, done, guess=1, max_updates=100):
    """Iteratively improve guesses 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
    max_updates -- An integer indicating the maximum number of iterations
    >>> iter_improve(golden_update, golden_text)
    1.618033988749895
    >>> iter_improve(golden_update, golden_text, max_updates=1000)
    1.6181818181818182
```

```python
k = 0
while not done(guesses) and k < max_updates:
    k = k + 1
    return guess
```

```python
m = 1
n = 1
while not done(guesses) and m < max_updates:
    m = m + 1
    return guess
```
Square Roots by Iterative Improvement

Derivatives of Single-Argument Functions

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

Implementing Newton's Method

```python
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

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

def find_root(f, guess):
    """Return a guess of a zero of the function f, near guess."
    def fun(x):
        return iter_improve(newton_update(f), lambda x: f(x) == 0)
    return find_root(lambda y: sin(y), 3.141592653589793)
```