How to Draw an Environment Diagram

When defining a function:
1. Create a function value with signature `<name>(<formal parameters>)`
2. For nested definitions, label the parent as the first frame of the current environment
3. Bind `<name>` to the function value in the first frame of the current environment

When calling a function:
1. Add a local frame labeled with the `<name>` of the function
2. If the function has a parent label, copy it to this frame
3. Bind the `<formal parameters>` to the arguments in this frame
4. Execute the body of the function in the environment that starts with this frame

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"

Lambda expressions are rare in Python, but important in general

Evaluation of Lambda vs. Def

```python
lambda x: x * x
```

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

Execution procedure for def statements:
1. Create a function value with signature `<name>(<formal parameters>)` and the current frame as parent
2. Bind `<name>` to that value in the current frame

Evaluation procedure for lambda expressions:
1. Create a function value with signature `{<formal parameters>}` and the current frame as parent
2. Evaluate to that value
Lambda vs. Def Statements

\[
\text{square = lambda x: x * x} \\
\text{def square(x): return x * x}
\]

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

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 \( \sqrt{a} \)

Special Case: Square Roots

How to compute \( \text{square_root}(a) \)

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

\[
x = \frac{x - f(x)/f'(x)}{2}
\]

Implementation questions:
What guess should start the computation?
How do we know when we are finished?

Special Case: Cube Roots

How to compute \( \text{cube_root}(a) \)

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

\[
x = \frac{2x + \frac{a}{x^2}}{3}
\]

Implementation questions:
What guess should start the computation?
How do we know when we are finished?
Iterative Improvement

First, identify common structure.
Then define a function that generalizes the procedure.

def iter_improve(update, done, guess=1, max_updates=1000):
    """Iteratively improve guess with update until done returns a true value."
    k = 0
    while not done(guess) and k < max_updates:
        guess = update(guess)
        k = k + 1
    return guess

Newton’s Method for nth Roots

def nth_root_func_and_derivative(n, a):
    def root_func(x):
        return pow(x, n) - a
    def derivative(x):
        return n * pow(x, n-1)
    return root_func, derivative

def nth_root_newton(a, n):
    """Return the nth root of a."
    root_func, deriv = nth_root_func_and_derivative(n, a)
    def update(x):
        return x - root_func(x) / deriv(x)
    def done(x):
        return root_func(x) == 0
    return iter_improve(update, done)