Higher-Order Functions

Slides adapted from Berkeley CS61a

Higher-Order Functions

Functions are first-class, meaning they can be manipulated as values

A higher-order function is:

A function that takes a function as an argument

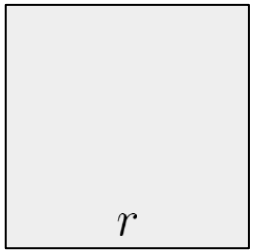
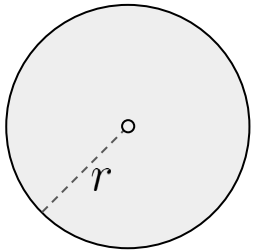
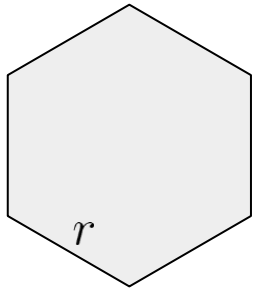
and/or

A function that returns a function as a return value

Generalization

Generalizing Patterns with Arguments

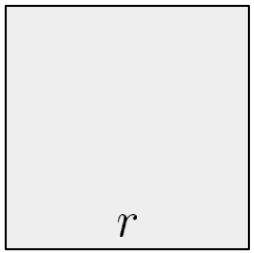
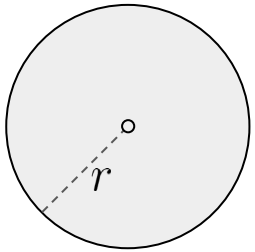
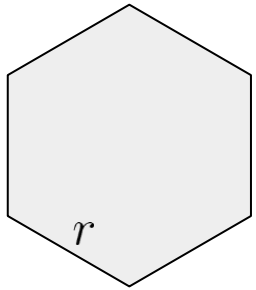
Regular geometric shapes relate length and area.

Shape:			
Area:	r^2	$\pi \cdot r^2$	$\frac{3\sqrt{3}}{2} \cdot r^2$

Demo_1

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:			
Area:	$1 \cdot r^2$	$\pi \cdot r^2$	$\frac{3\sqrt{3}}{2} \cdot r^2$

Finding common structure allows for shared implementation

Demo_1

Higher-Order Functions

Generalizing Over Computational Processes

The common structure among functions may be a computational process, rather than a number.

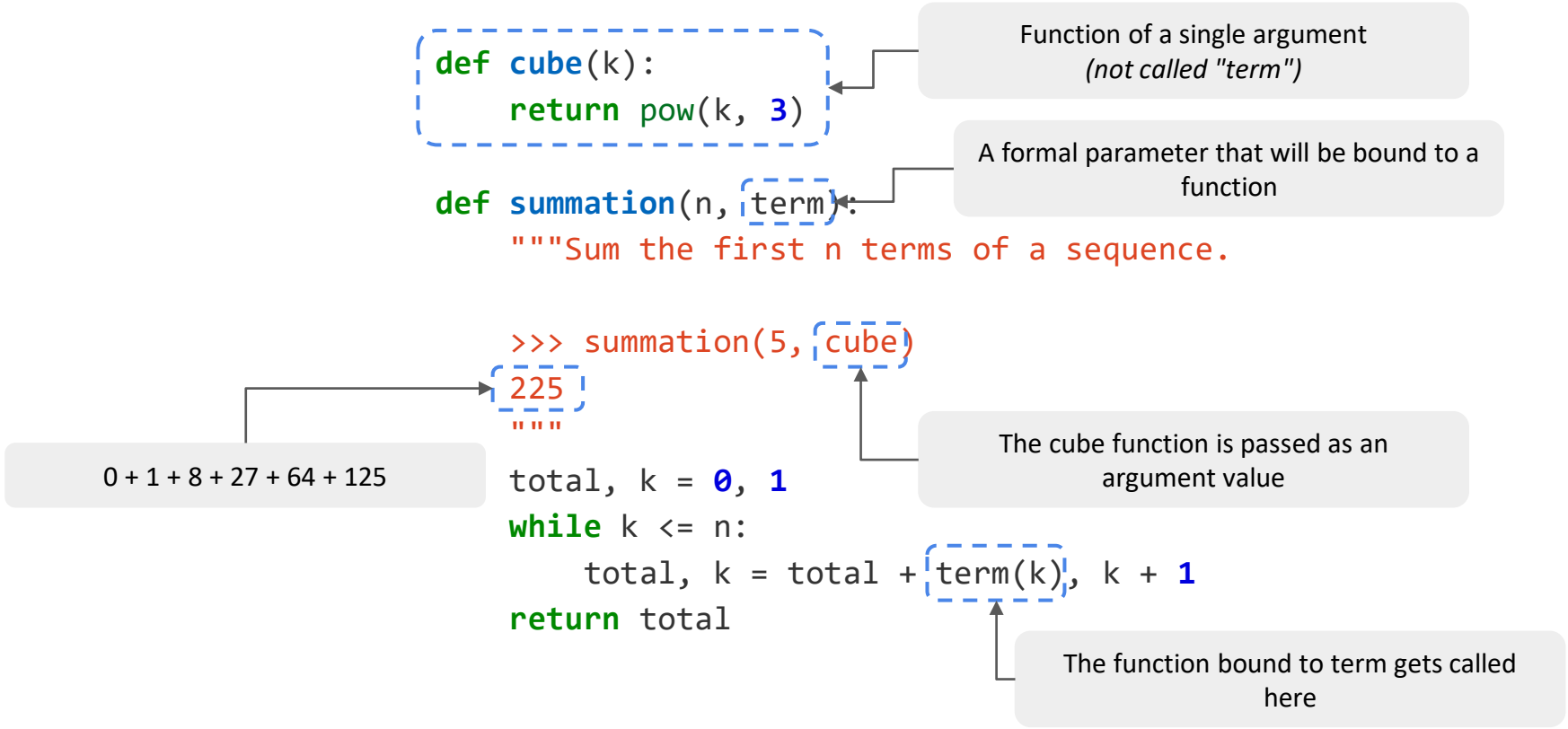
$$\sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15$$

$$\sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225$$

$$\sum_{k=1}^{5} \frac{8}{(4k-3)\cdot(4k-1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04$$

Demo_2

Summation Example

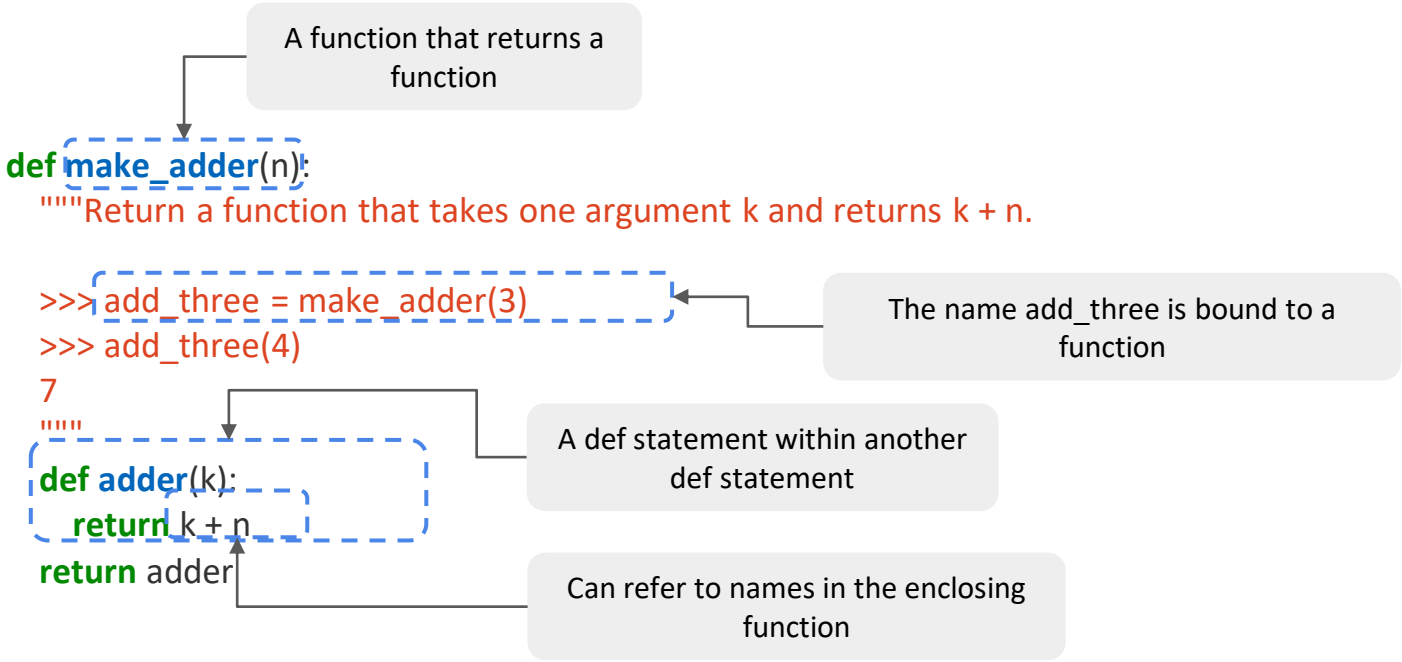


Functions as Return Values

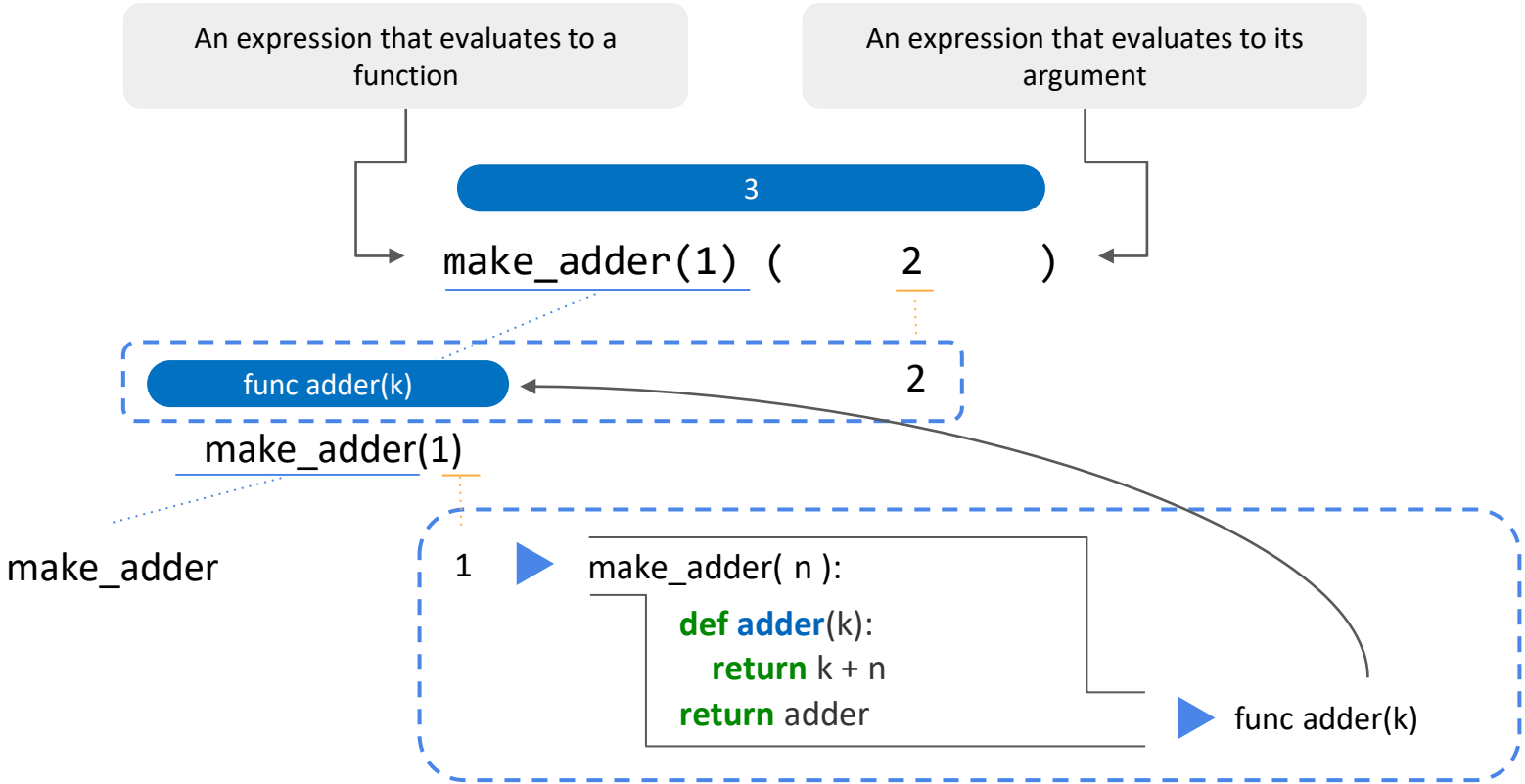
Demo_3

Locally Defined Functions

Functions defined within other function bodies are bound to names in a local frame



Call Expressions as Operator Expressions



Summary

- Higher-order function: any function that either accepts a function as an argument and/or returns a function
- Why are these useful?
    - Generalize over different form of computation
    - Helps remove repetitive segments of code
- We saw nested functions (closures) can access variables in outer function through static scoping.

A More Complex Example

```
def make_adder(n):
    """Return a function that takes one argument k and returns k + n.

    >>> add_three = make_adder(3)
    >>> add_three(4)

    """
    def adder(k):
        return k + n
    return adder

def square(x):
    return x * x

def compose1(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
compose1(square, make_adder(2))(3)
```