Higher-Order Functions

9 / 30 / 2019

Slides adapted from Berkeley CS61a

Higher-Order Functions

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

A higher-order function is:

1. A function that takes a function as an argument

   and/or

1. A function that returns a function as a return value

Designing Functions

Describing Functions

```
def square(x):
    """Return X * X"""
```

A function's *domain* is the set of all inputs it might possibly take as arguments.

x is a number

A function's range is the set of output values it might possibly return.

square returns a non-negative real number

A pure function's behavior is the relationship it creates between input and output.

square returns the square of x

A Guide to Designing Function

Give each function exactly one job, but make it apply to many related situations

```
>>> round(1.23)
1
```

```
>>> round(1.23, 1)
1.2
```

```
>>> round(1.23, 0)
1
```

```
>>> round(1.23, 5)
1.23
```

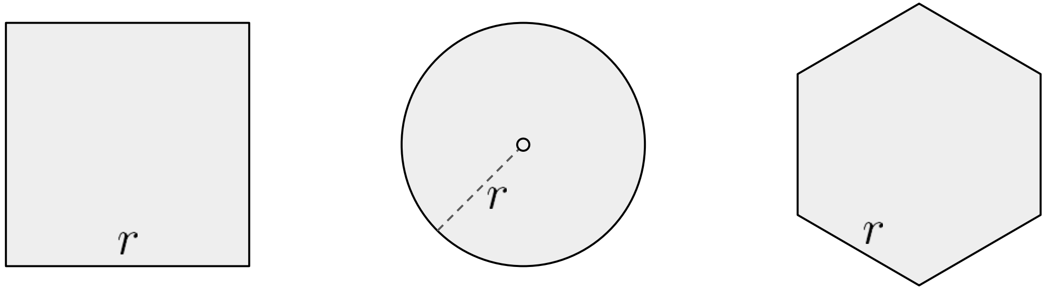
Don't repeat yourself (DRY). Implement a process just once, but execute it many times.

Generalization

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:



Area:

$\boxed{}\ r^2$ $\qquad$ $\boxed{\pi} \cdot r^2$ $\qquad$ $\boxed{\frac{3\sqrt{3}}{2}} \cdot r^2$

Finding common structure allows for shared implementation

Demo

Higher-Order Functions

Generalizing Over Computational Processes

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

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

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

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

Demo

Summation Example

```python
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence.

    >>> summation(5, cube)
    225
    """
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total
```

Function of a single argument
(not called "term")

A formal parameter that will
be bound to a function

0 + 1 + 8 + 27 + 64 + 125

The cube function is passed
as an argument value

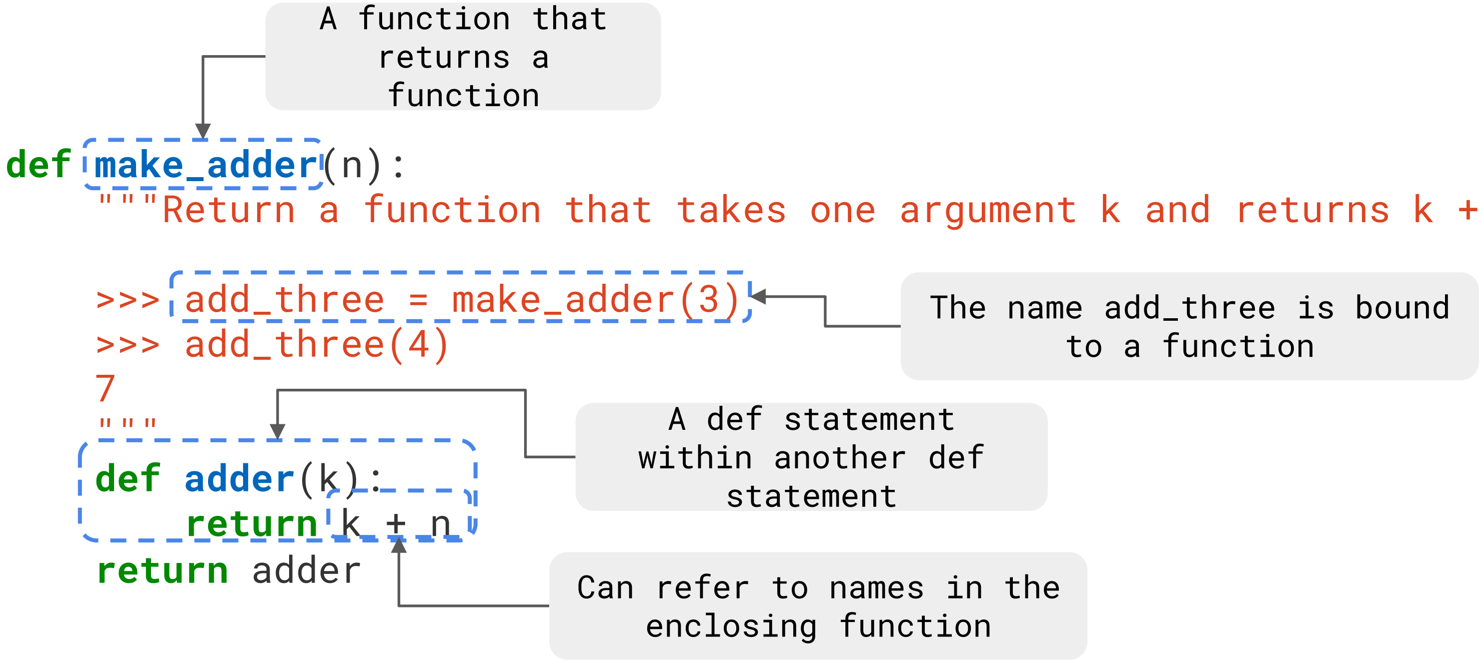
The function bound to term
gets called here

Functions as Return Values

Demo

Locally Defined Functions

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

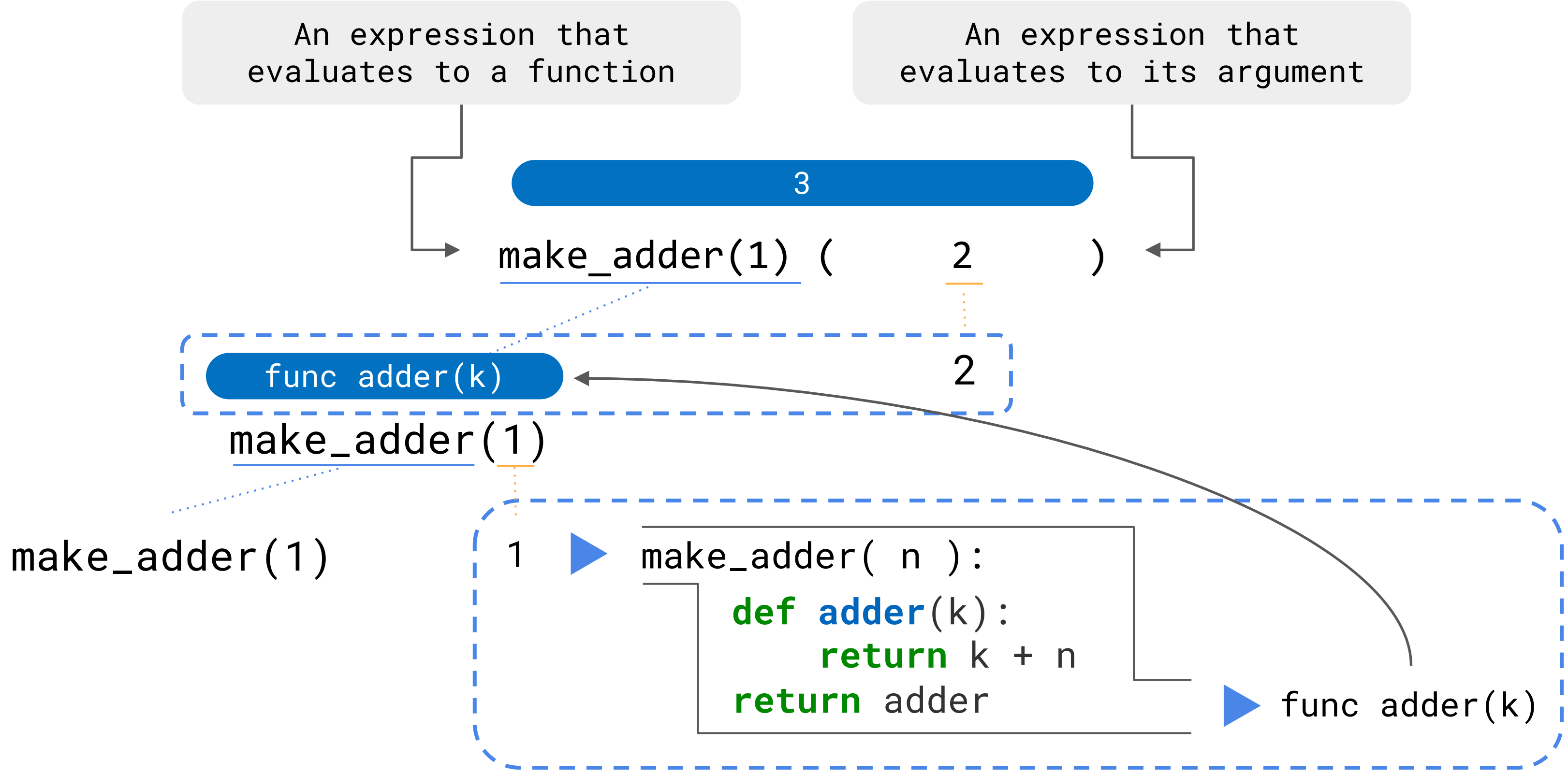


```python
def make_adder(n):
    """Return a function that takes one argument k and returns k + n.

    >>> add_three = make_adder(3)
    >>> add_three(4)
    7
    """
    def adder(k):
        return k + n
    return adder
```

A function that returns a function
The name add_three is bound to a function
A def statement within another def statement
Can refer to names in the enclosing function

Call Expressions as Operator Expressions



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)
```

Self Reference

Returning a Function Using Its Own Name

```python
def print_sums(n):
    print(n)
    def next_sum(k):
        return print_sums(n + k)
    return next_sum

print_sums(1)(3)(5)
```

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
- One use case: summation
  - We generalized over the computation of each term
- We saw nested functions can access variables in outer function (adder) as well as the outer function itself (print_sums)