Higher-Order Functions
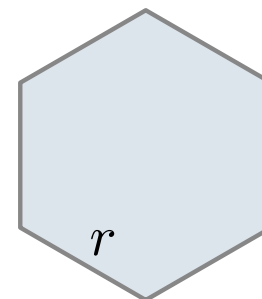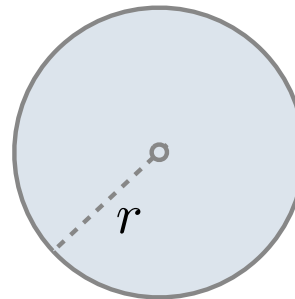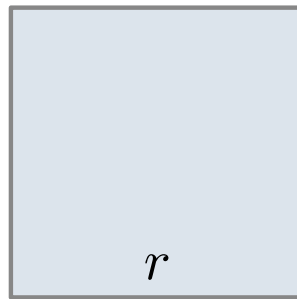
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

Area: $\boxed{1} \cdot r^2$ $\boxed{\pi} \cdot r^2$ $\boxed{\dfrac{3\sqrt{3}}{2}} \cdot r^2$

Finding common structure allows for shared implementation

(Demo)

Generalizing Over Computational Processes

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

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

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

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

(Demo)

Summation Example

```
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

The cube function is passed as an argument value

0 + 1 + 8 + 27 + 64 + 125

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

A function that
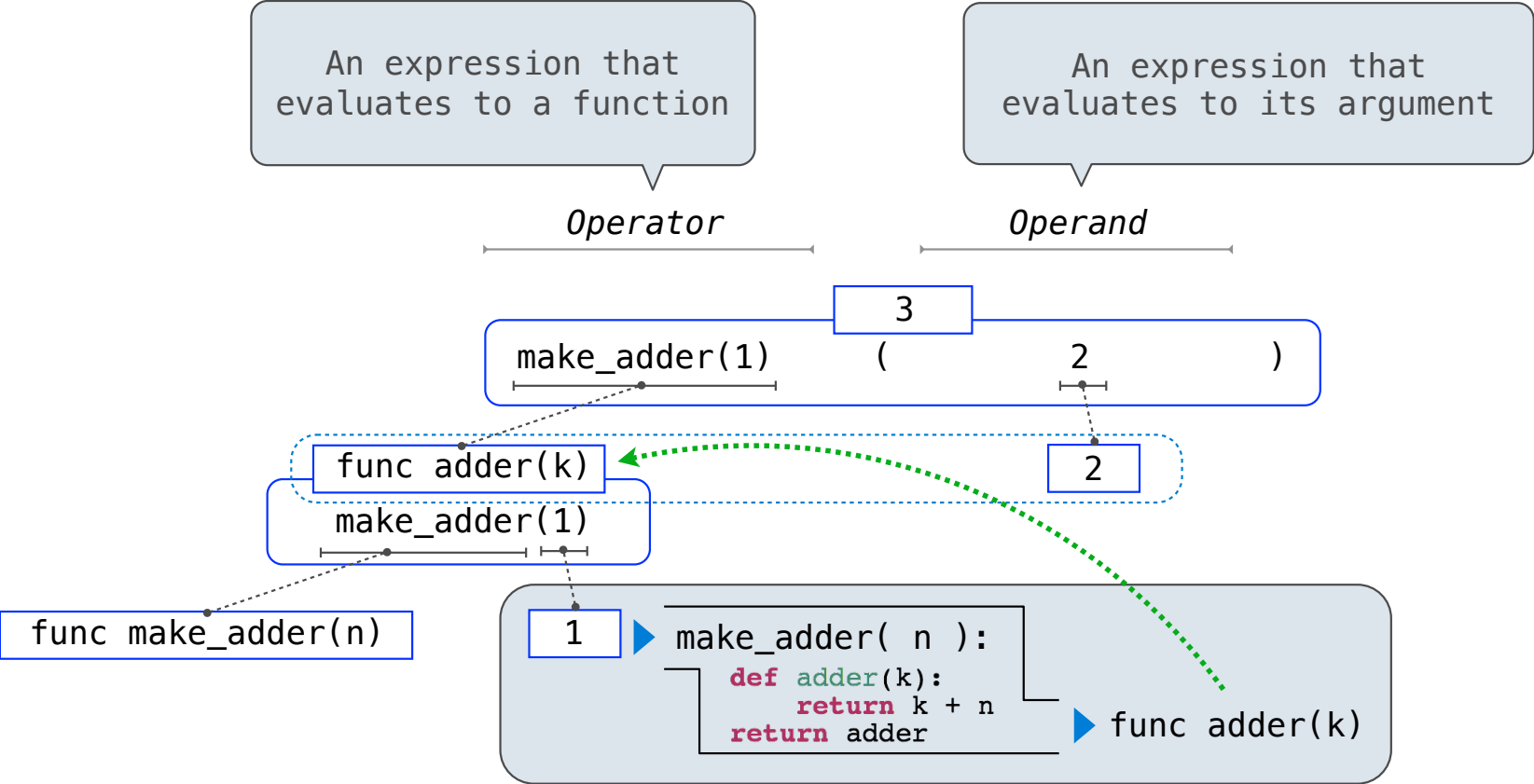returns a function

```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
```

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

An expression that evaluates to a function

An expression that evaluates to its argument

Operator

Operand

3

make_adder(1)   (2   )

func adder(k)

make_adder(1)

2

func make_adder(n)

1   make_adder(n):
         def adder(k):
             return k + n
         return adder   func adder(k)
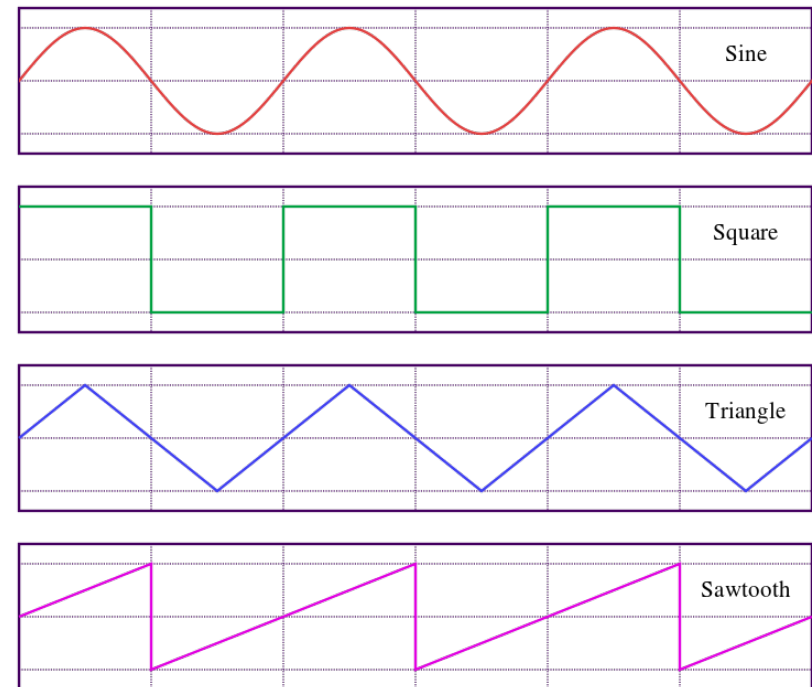
Function Example: Sounds

WAV Files

The Waveform Audio File Format
encodes a sampled sound wave
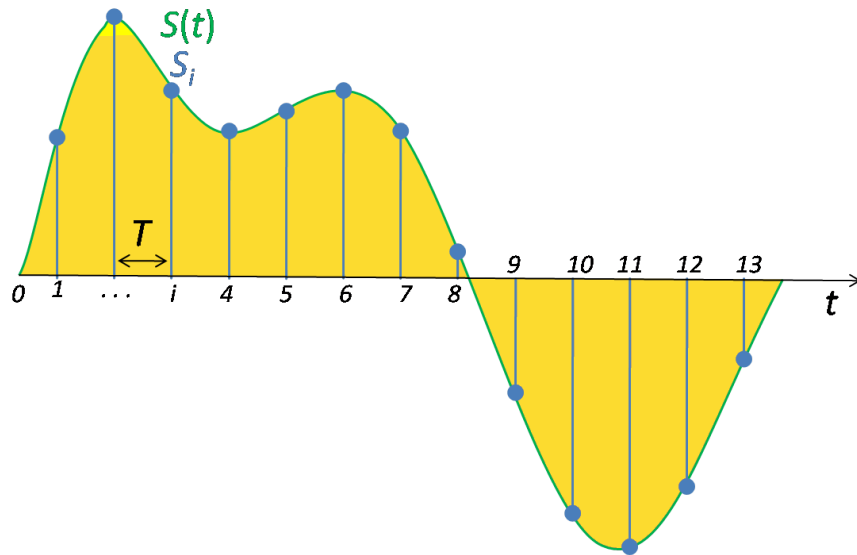
A triangle wave is the simple wave
form with the most pleasing sound

(Demo)

https://en.wikipedia.org/wiki/Triangle_wave
https://en.wikipedia.org/wiki/Sampling_(signal_processing)

Function Composition
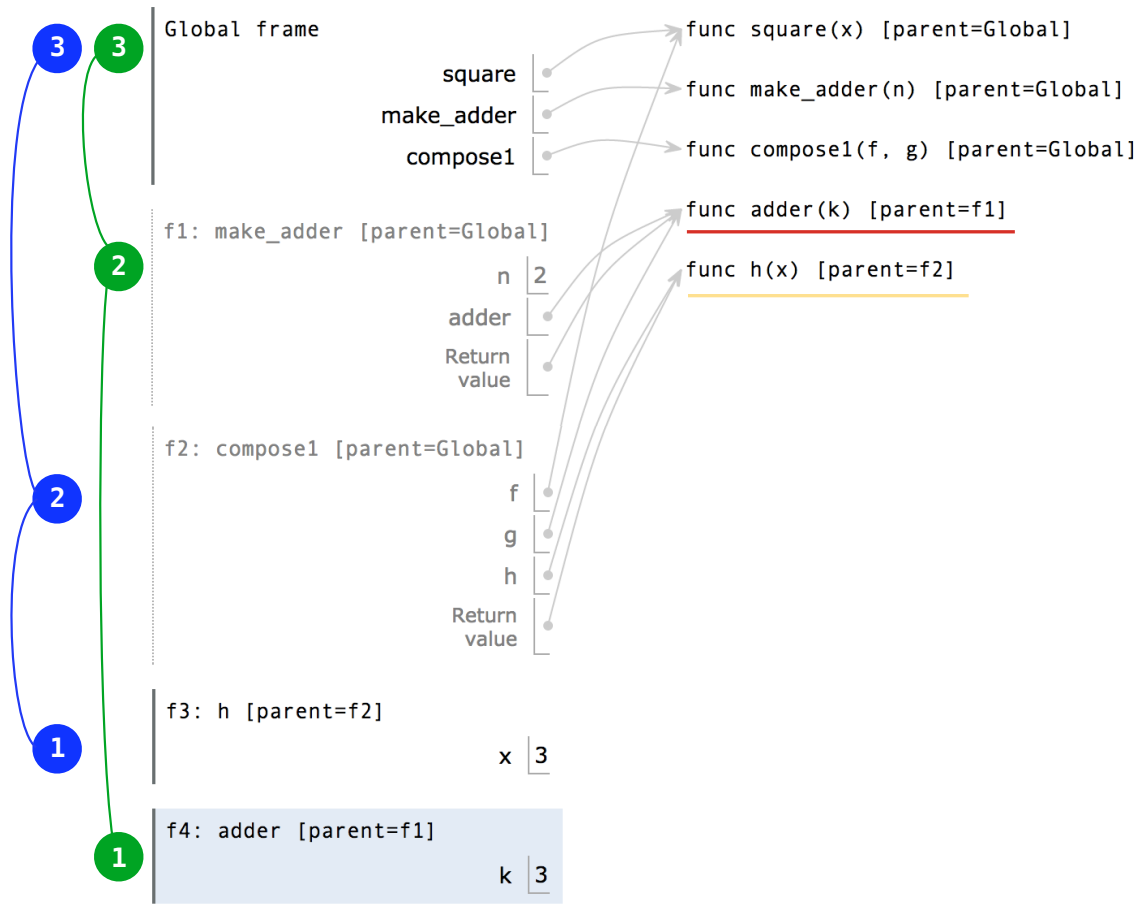
(Demo)

The Environment Diagram for Function Composition

```
1   def square(x):
2       return x * x
3
4   def make_adder(n):
5       def adder(k):
6           return k + n
7       return adder
8
9   def compose1(f, g):
10      def h(x):
11          return f(g(x))
12      return h
13
14  compose1(square, make_adder(2))(3)
```

> Return value of make_adder is an argument to compose1

Global frame

square	→ func square(x) [parent=Global]
make_adder	→ func make_adder(n) [parent=Global]
compose1	→ func compose1(f, g) [parent=Global]

func adder(k) [parent=f1]

func h(x) [parent=f2]

f1: make_adder [parent=Global]

n	2
adder	
Return value	

f2: compose1 [parent=Global]

f	
g	
h	
Return value	

f3: h [parent=f2]

x	3

f4: adder [parent=f1]

k	3

http://pythontutor.com/composingprograms.html#code=def%20square%28x%29%3A%0A%20%20%20%20return%20x%20*%20x%0A%0Adef%20make_adder%28n%29%3A%0A%20%20%20%20def%20adder%28k%29%3A%0A%20%20%20%20%20%20%20%20return%20k%20%2B%20n%0A%20%20%20%20return%20adder%0A%20%20%20%20%0Adef%20compose1%28f,%20g%29%3A%0A%20%20%20%20def%20h%28x%29%3A%0A%20%20%20%20%20%20%20%20return%20f%28g%28x%29%29%0A%20%20%20%20return%20h%0A%0Acompose1%28square,%20make_adder%282%29%29%283%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

15

Abstraction

Functional Abstractions

```
def square(x):                          def sum_squares(x, y):
    return mul(x, x)                         return square(x) + square(y)
```

What does sum_squares need to know about square?

• Square takes one argument.   **Yes**

• Square has the intrinsic name square.   **No**

• Square computes the square of a number.   **Yes**

• Square computes the square by calling mul.   **No**

```
def square(x):                          def square(x):
    return pow(x, 2)                         return mul(x, x-1) + x
```

> If the name "square" were bound to a built-in function,
> sum_squares would still work identically.

Choosing Names

Names typically don't matter for correctness

but

they matter a lot for composition

From:	To:
true_false	rolled_a_one
d	dice
helper	take_turn
my_int	num_rolls
l, I, O	k, i, m

Names should convey the meaning or purpose of the values to which they are bound.

The type of value bound to the name is best documented in a function's docstring.

Function names typically convey their effect (**print**), their behavior (**triple**), or the value returned (**abs**).
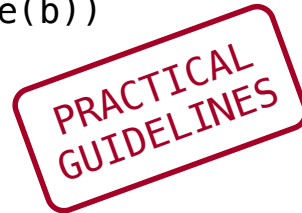
Which Values Deserve a Name

Reasons to add a new name

Repeated compound expressions:

```
if sqrt(square(a) + square(b)) > 1:
    x = x + sqrt(square(a) + square(b))
```

▽

```
hypotenuse = sqrt(square(a) + square(b))
if hypotenuse > 1:
    x = x + hypotenuse
```

Meaningful parts of complex expressions:

```
x1 = (−b + sqrt(square(b) − 4 * a * c)) / (2 * a)
```

▽

```
discriminant = square(b) − 4 * a * c
x1 = (−b + sqrt(discriminant)) / (2 * a)
```

PRACTICAL GUIDELINES

More Naming Tips

- Names can be long if they help document your code:

```
average_age = average(age, students)
```

is preferable to

```
# Compute average age of students
aa = avg(a, st)
```

- Names can be short if they represent generic quantities: counts, arbitrary functions, arguments to mathematical operations, etc.

```
n, k, i − Usually integers
x, y, z − Usually real numbers
f, g, h − Usually functions
```