

61A Lecture 4

Monday, September 8

Announcements

Announcements

- Homework 1 due Wednesday 9/10 at 2pm. Late homework is not accepted!

Announcements

- Homework 1 due Wednesday 9/10 at 2pm. Late homework is not accepted!
- Homework parties on Monday 9/8 (**Today!**)

Announcements

- Homework 1 due Wednesday 9/10 at 2pm. Late homework is not accepted!
- Homework parties on Monday 9/8 (**Today!**)
 - 3pm–4pm in Wozniak Lounge in Soda Hall (100 person capacity)

Announcements

- Homework 1 due Wednesday 9/10 at 2pm. Late homework is not accepted!
- Homework parties on Monday 9/8 (**Today!**)
 - 3pm–4pm in Wozniak Lounge in Soda Hall (100 person capacity)
 - 6pm–8pm in 2050 Valley Life Sciences Building (408 person capacity)

Announcements

- Homework 1 due Wednesday 9/10 at 2pm. Late homework is not accepted!
- Homework parties on Monday 9/8 (**Today!**)
 - 3pm–4pm in Wozniak Lounge in Soda Hall (100 person capacity)
 - 6pm–8pm in 2050 Valley Life Sciences Building (408 person capacity)
- More sections for students without prior programming experience! <http://cs61a.org>

Announcements

- Homework 1 due Wednesday 9/10 at 2pm. Late homework is not accepted!
- Homework parties on Monday 9/8 (**Today!**)
 - 3pm–4pm in Wozniak Lounge in Soda Hall (100 person capacity)
 - 6pm–8pm in 2050 Valley Life Sciences Building (408 person capacity)
- More sections for students without prior programming experience! <http://cs61a.org>
- Take-home quiz 1 starts Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm

Announcements

- Homework 1 due Wednesday 9/10 at 2pm. Late homework is not accepted!
- Homework parties on Monday 9/8 (**Today!**)
 - 3pm–4pm in Wozniak Lounge in Soda Hall (100 person capacity)
 - 6pm–8pm in 2050 Valley Life Sciences Building (408 person capacity)
- More sections for students without prior programming experience! <http://cs61a.org>
- Take-home quiz 1 starts Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
 - Open-computer, but no external resources or friends

Announcements

- Homework 1 due Wednesday 9/10 at 2pm. Late homework is not accepted!
- Homework parties on Monday 9/8 (**Today!**)
 - 3pm–4pm in Wozniak Lounge in Soda Hall (100 person capacity)
 - 6pm–8pm in 2050 Valley Life Sciences Building (408 person capacity)
- More sections for students without prior programming experience! <http://cs61a.org>
- Take-home quiz 1 starts Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
 - Open-computer, but no external resources or friends
 - Content Covered: Lectures through last Friday 9/5 (same topics as Homework 1)

Announcements

- Homework 1 due Wednesday 9/10 at 2pm. Late homework is not accepted!
- Homework parties on Monday 9/8 (**Today!**)
 - 3pm–4pm in Wozniak Lounge in Soda Hall (100 person capacity)
 - 6pm–8pm in 2050 Valley Life Sciences Building (408 person capacity)
- More sections for students without prior programming experience! <http://cs61a.org>
- Take-home quiz 1 starts Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
 - Open-computer, but no external resources or friends
 - Content Covered: Lectures through last Friday 9/5 (same topics as Homework 1)
- Project 1 due next Wednesday 9/17 at 11:59pm

Iteration Example

The Fibonacci Sequence

The Fibonacci Sequence



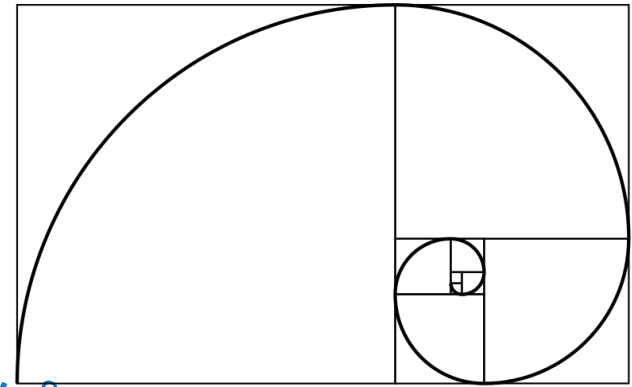
The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987



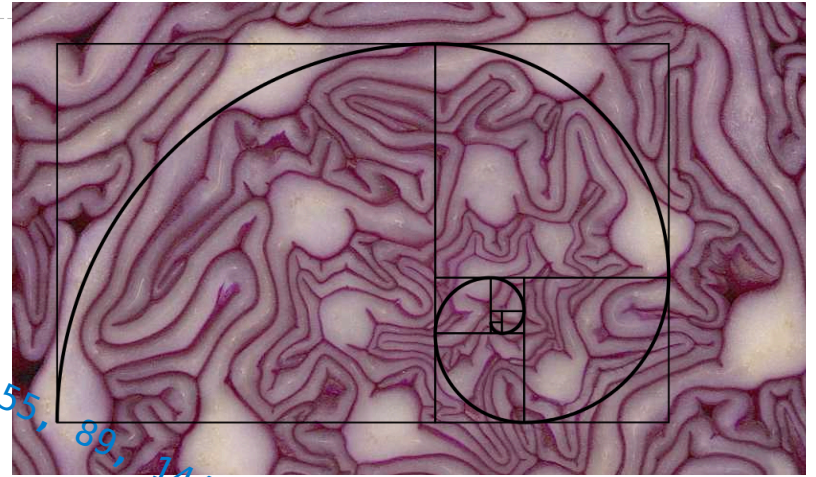
The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987



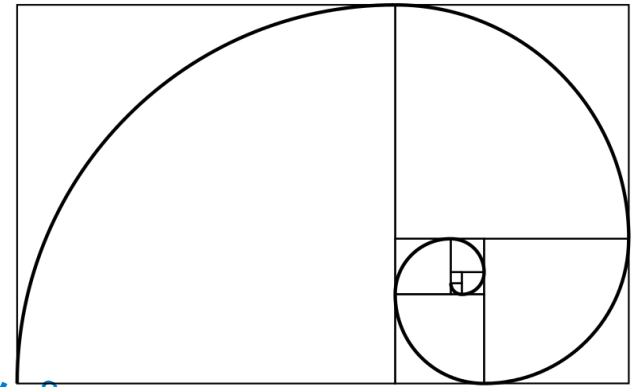
The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987



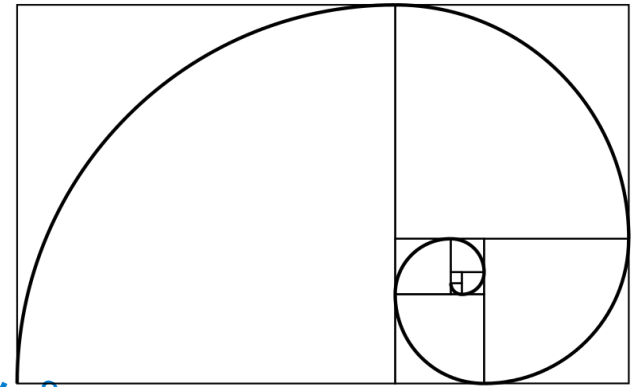
The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987



The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987

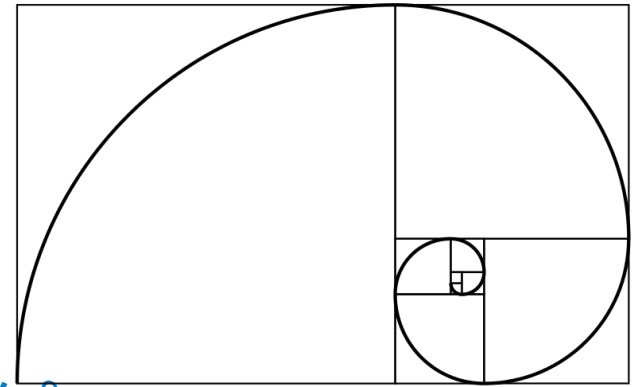


```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 1                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```



The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987



```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 1                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

The next Fibonacci number is the sum of the current one and its predecessor

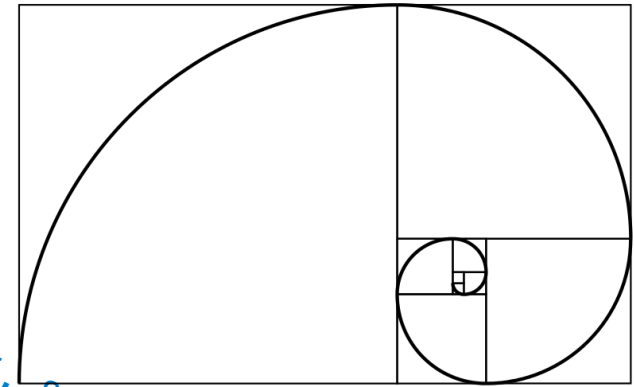


The Fibonacci Sequence

fib

n	
predecessor	
current	
k	

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987



```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 1                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

The next Fibonacci number is the sum of the current one and its predecessor



The Fibonacci Sequence

fib

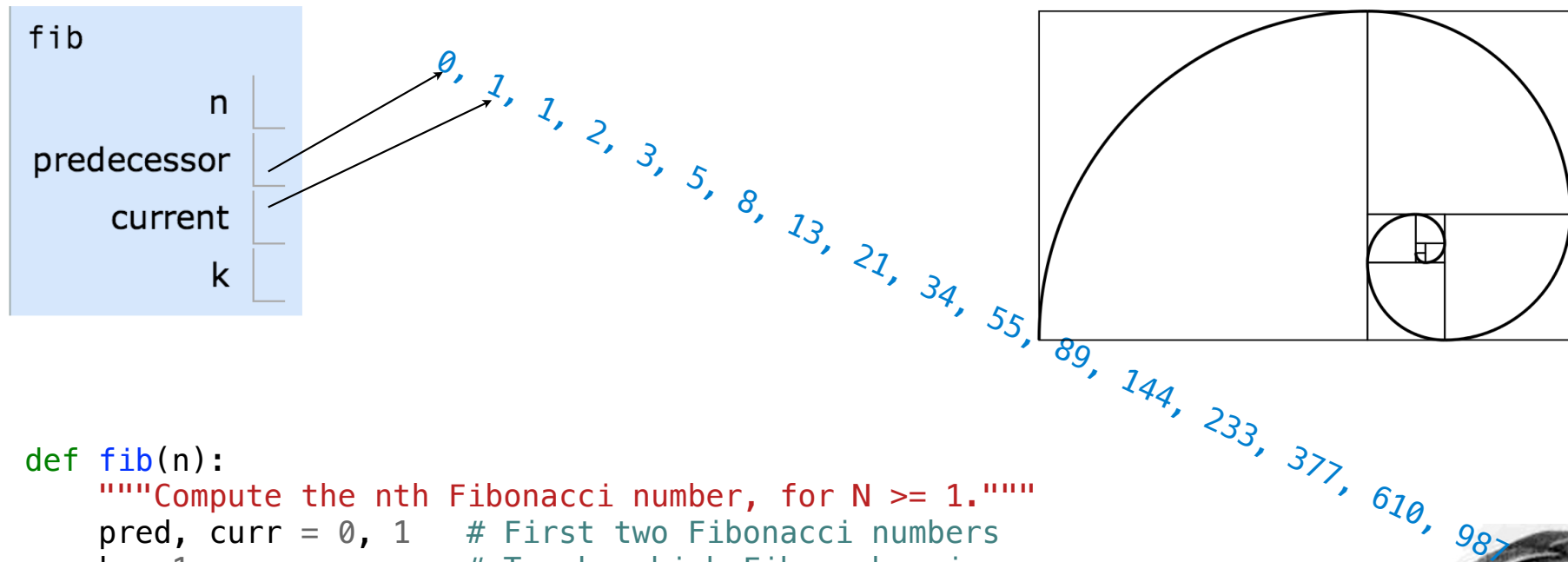
n	
predecessor	
current	
k	

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987

```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 1                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

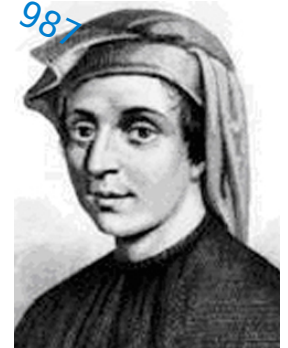
The next Fibonacci number is the sum of the current one and its predecessor

The Fibonacci Sequence

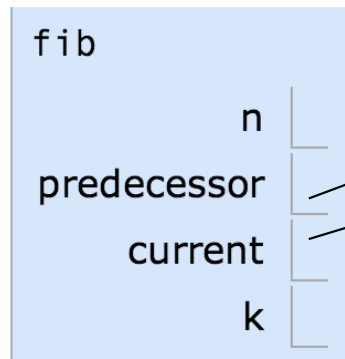


```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 1                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

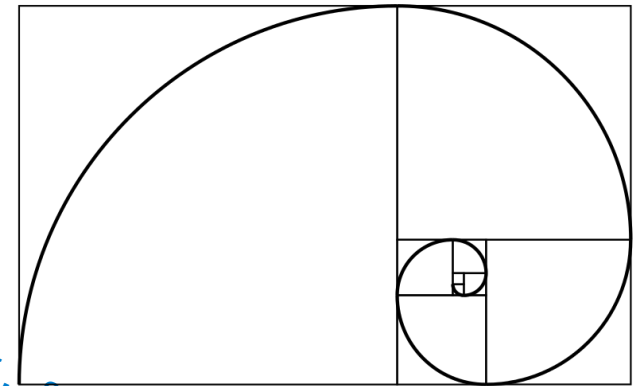
The next Fibonacci number is the sum of the current one and its predecessor



The Fibonacci Sequence

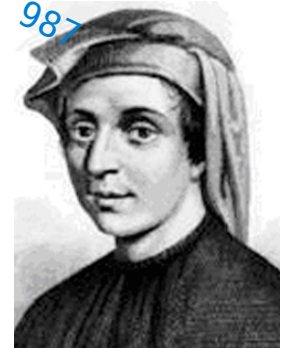


0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987

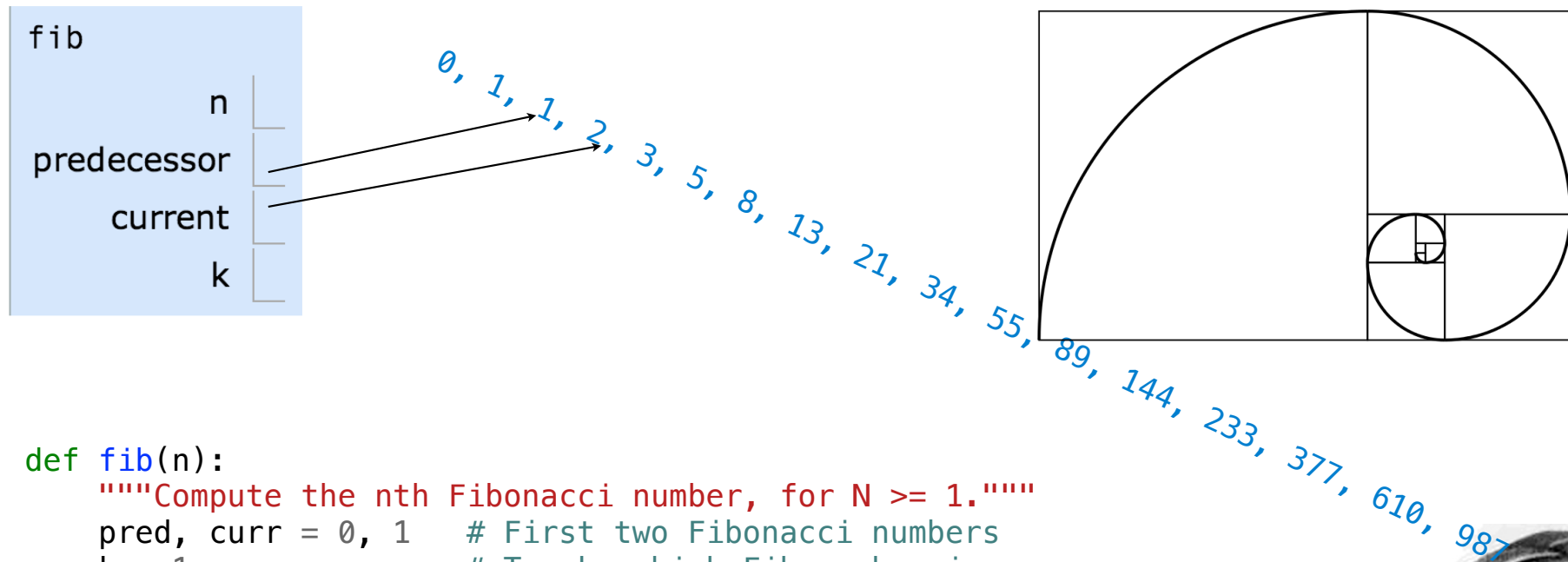


```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 1                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

The next Fibonacci number is the sum of the current one and its predecessor

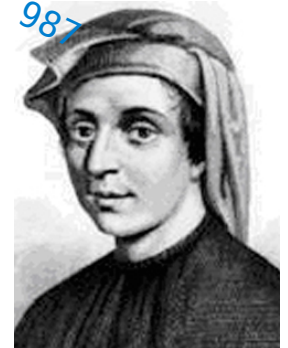


The Fibonacci Sequence

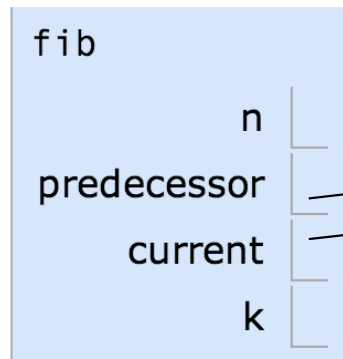


```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 1                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

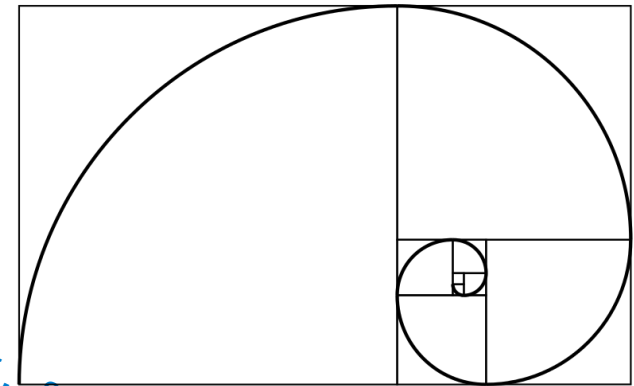
The next Fibonacci number is the sum of the current one and its predecessor



The Fibonacci Sequence



0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987

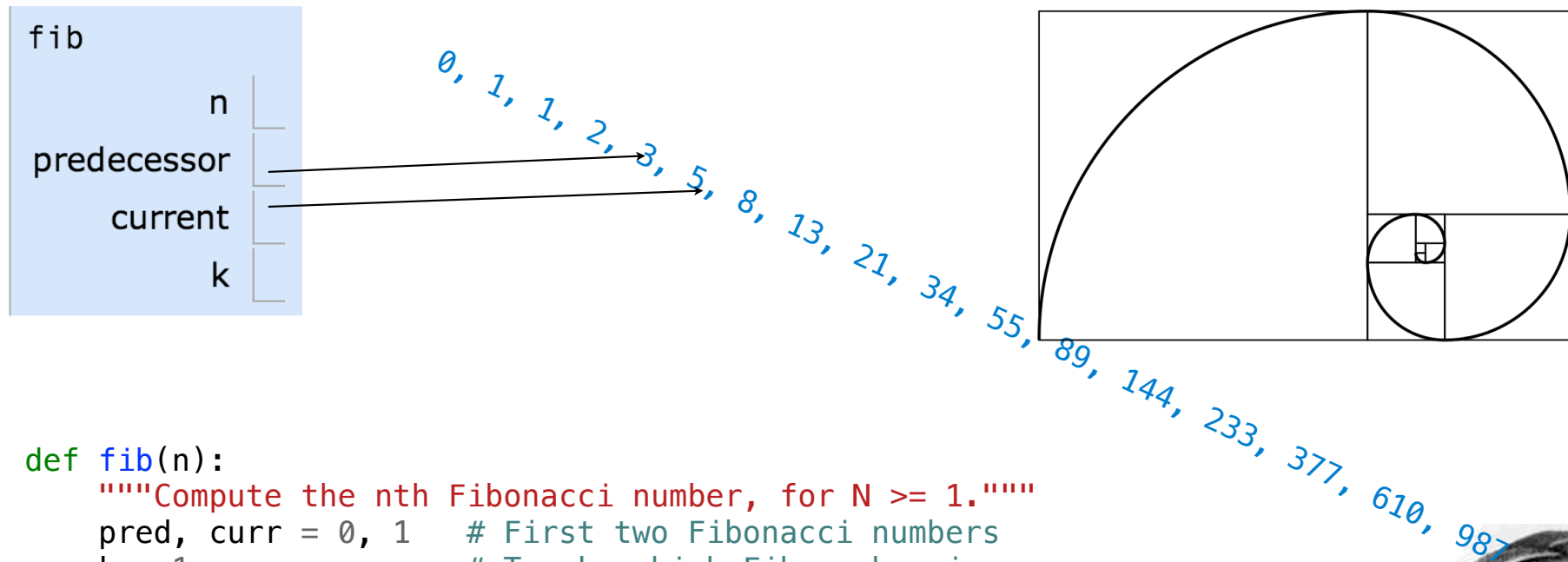


```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 1                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

The next Fibonacci number is the sum of the current one and its predecessor



The Fibonacci Sequence



```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 1                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

The next Fibonacci number is the sum of the current one and its predecessor



Discussion Question 1

What does pyramid compute?



Discussion Question 1

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



$$n^2 + 1$$



$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



$$n^2 + 1$$



$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```

a

b



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



$$n^2 + 1$$



$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```

a

b



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



$$n^2 + 1$$



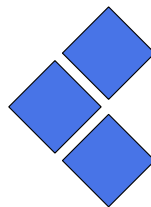
$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```

a

b



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



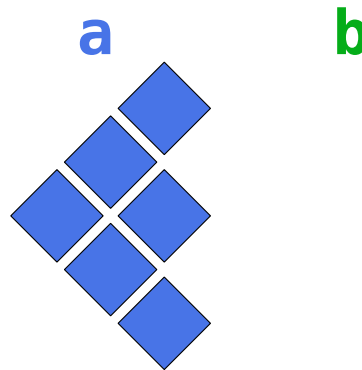
$$n^2 + 1$$



$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



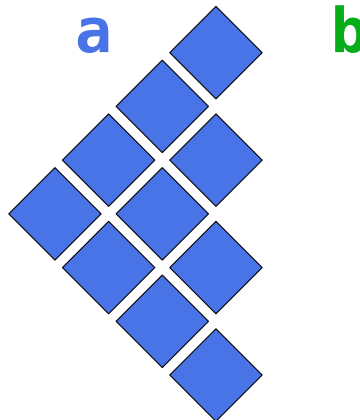
$$n^2 + 1$$



$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



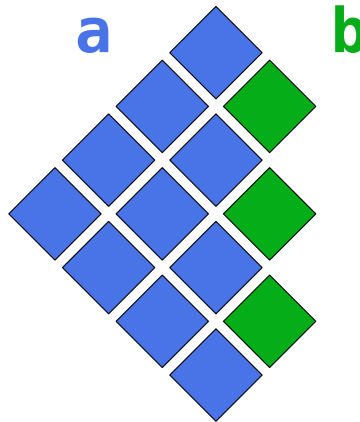
$$n^2 + 1$$



$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



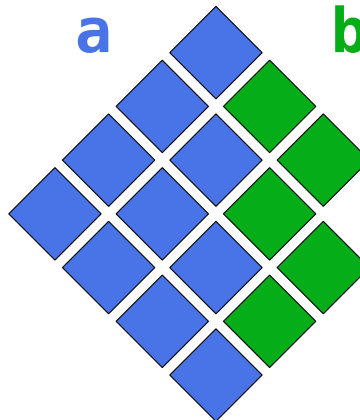
$$n^2 + 1$$



$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



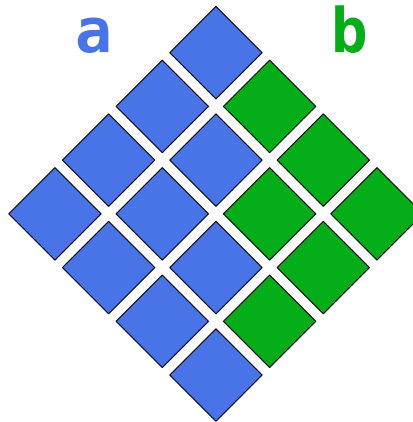
$$n^2 + 1$$



$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



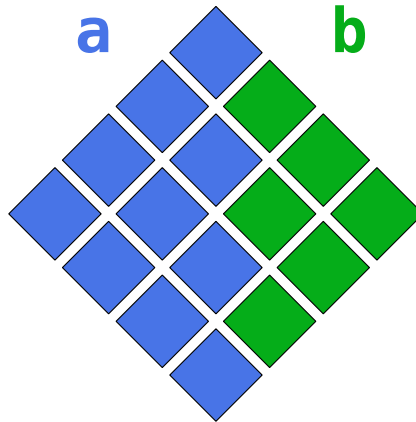
$$n^2 + 1$$



$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



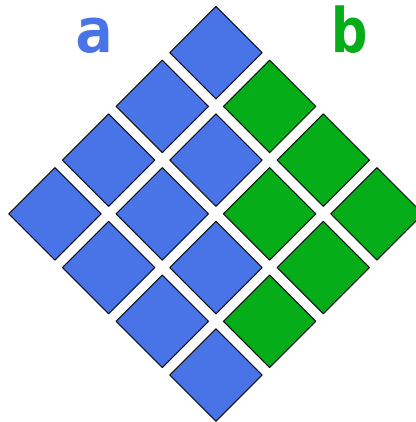
$$n^2 + 1$$



$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```



Discussion Question 1



$$n^2$$



$$(n + 1)^2$$



$$2 \cdot (n + 1)$$



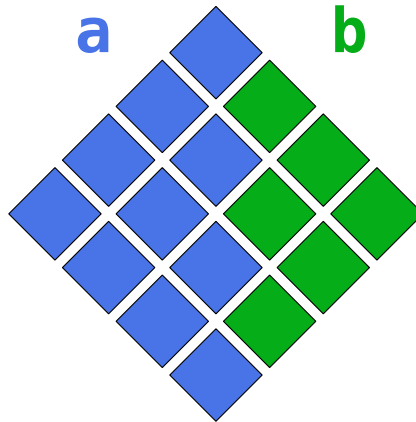
$$n^2 + 1$$



$$n \cdot (n + 1)$$

What does pyramid compute?

```
def pyramid(n):  
    a, b, total = 0, n, 0  
    while b:  
        a, b = a+1, b-1  
        total = total + a + b  
    return total
```



I'm still here



Designing Functions

Characteristics of Functions

Characteristics of Functions

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

Characteristics of Functions

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

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

Characteristics of Functions

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

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

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

Characteristics of Functions

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

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

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

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

Characteristics of Functions

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

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

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

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

```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""
```


Characteristics of 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 real number

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

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

```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""
```

Characteristics of 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 real number

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

*returns a non-negative
real number*

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

```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""
```

Characteristics of 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 real number

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

*returns a non-negative
real number*

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

*return value is the
square of the input*

```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""
```

Characteristics of 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 real number

```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""
```

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

*returns a non-negative
real number*

n is an integer greater than or equal to 1

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

*return value is the
square of the input*

Characteristics of 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 real number

```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""
```

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

*returns a non-negative
real number*

n is an integer greater than or equal to 1

returns a Fibonacci number

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

*return value is the
square of the input*

Characteristics of 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 real number

```
def fib(n):  
    """Compute the nth Fibonacci number, for N >= 1."""
```

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

*returns a non-negative
real number*

n is an integer greater than or equal to 1

returns a Fibonacci number

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

*return value is the
square of the input*

return value is the nth Fibonacci number

A Guide to Designing Function

A Guide to Designing Function

Give each function exactly one job.

A Guide to Designing Function

Give each function exactly one job.

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

A Guide to Designing Function

Give each function exactly one job.

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

Define functions generally.

A Guide to Designing Function

Give each function exactly one job.



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

Define functions generally.

A Guide to Designing Function

Give each function exactly one job.



not



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

Define functions generally.

A Guide to Designing Function

Give each function exactly one job.



not



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



Define functions generally.

A Guide to Designing Function

Give each function exactly one job.



not



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



Define functions generally.



Generalization

Generalizing Patterns with Arguments

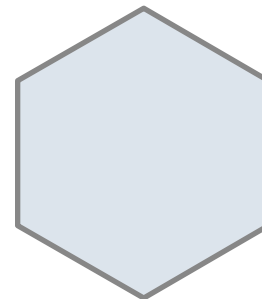
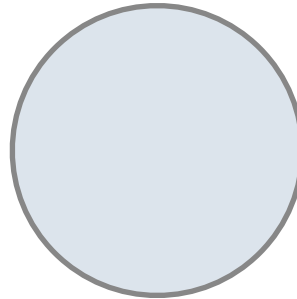
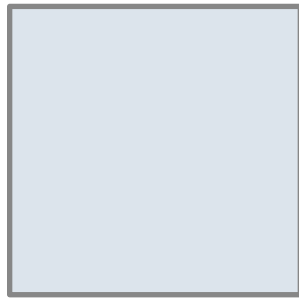
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

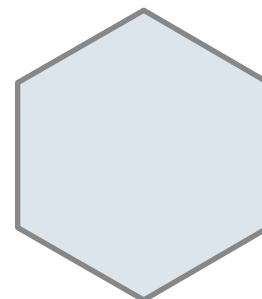
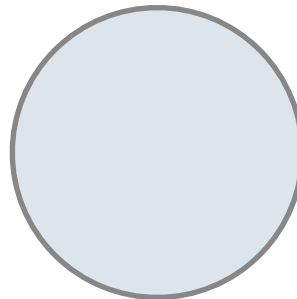
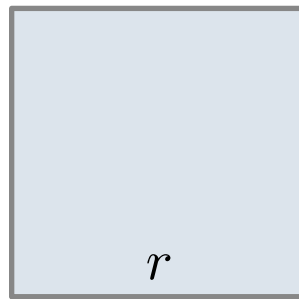
Shape:



Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

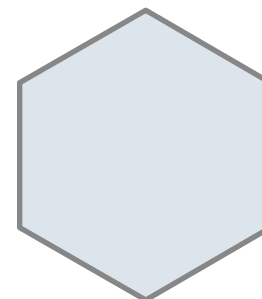
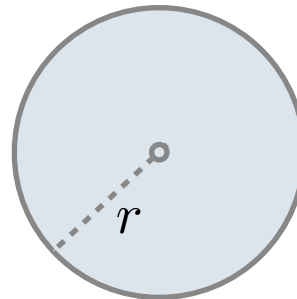
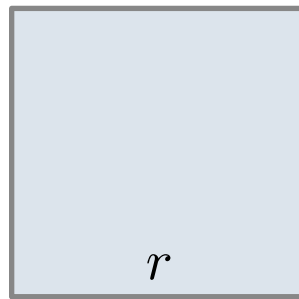
Shape:



Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

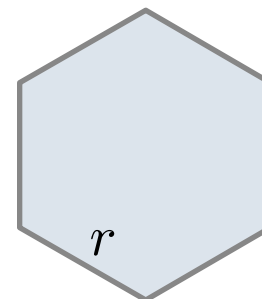
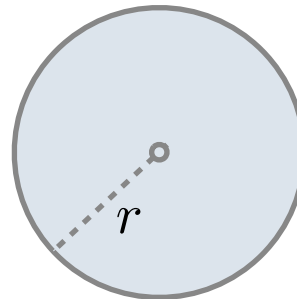
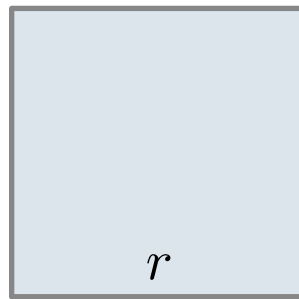
Shape:



Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

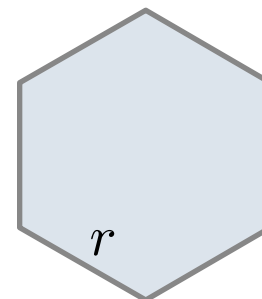
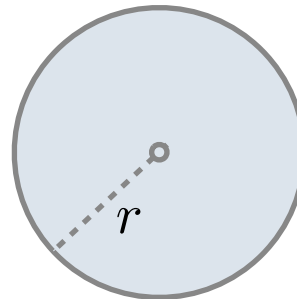
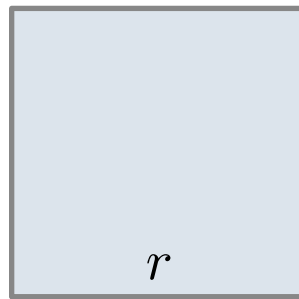
Shape:



Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

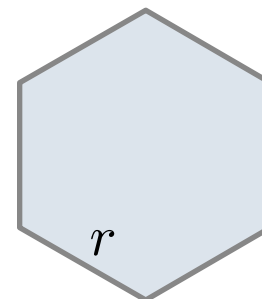
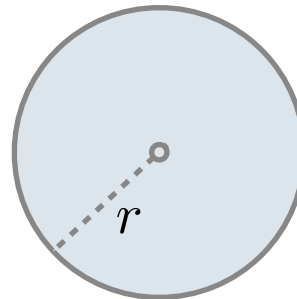
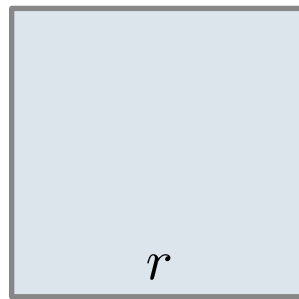


Area:

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:



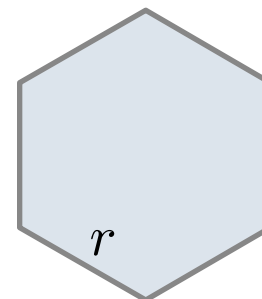
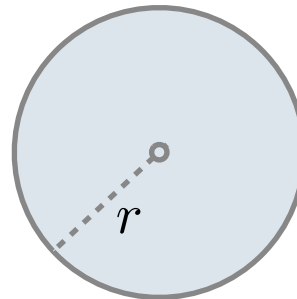
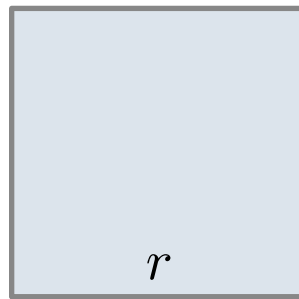
Area:

$$r^2$$

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:



Area:

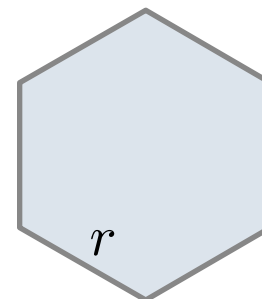
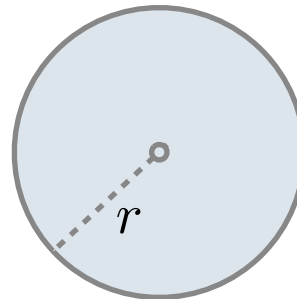
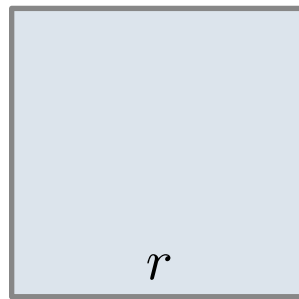
$$r^2$$

$$\pi \cdot r^2$$

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:



Area:

$$r^2$$

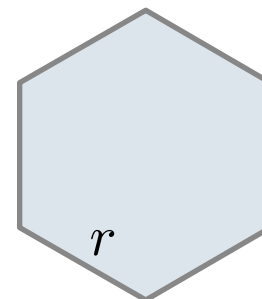
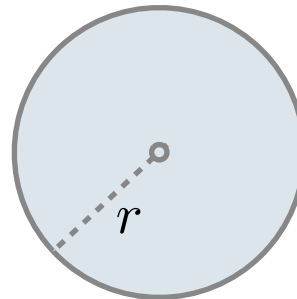
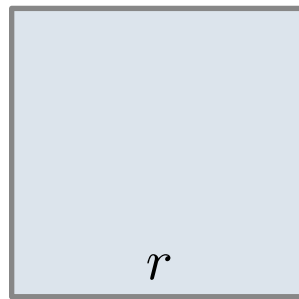
$$\pi \cdot r^2$$

$$\frac{3\sqrt{3}}{2} \cdot r^2$$

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:



Area:

$$1 \cdot r^2$$

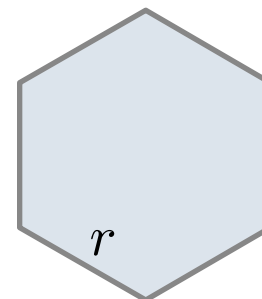
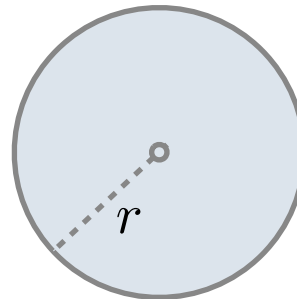
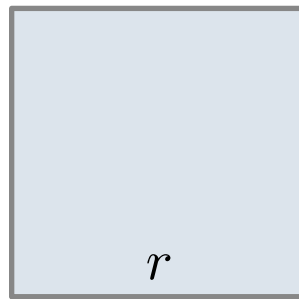
$$\pi \cdot r^2$$

$$\frac{3\sqrt{3}}{2} \cdot r^2$$

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:



Area:

$$1 \cdot r^2$$

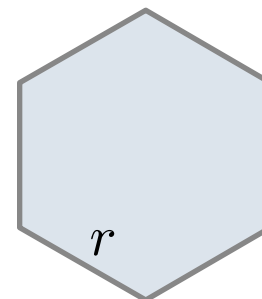
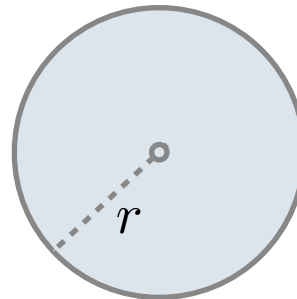
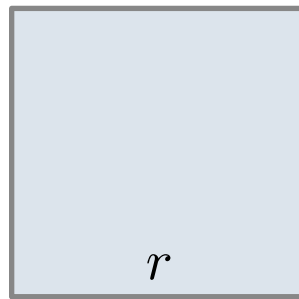
$$\pi \cdot r^2$$

$$\frac{3\sqrt{3}}{2} \cdot r^2$$

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:



Area:

$$1 \cdot r^2$$

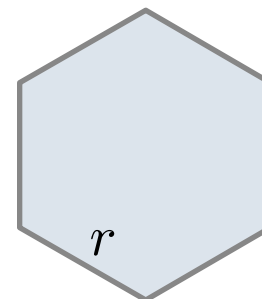
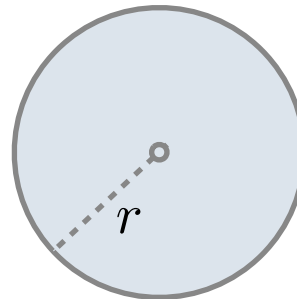
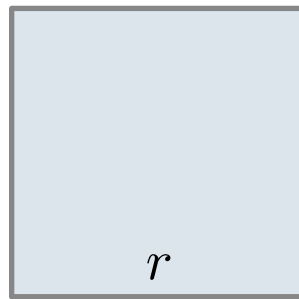
$$\pi \cdot r^2$$

$$\frac{3\sqrt{3}}{2} \cdot r^2$$

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:



Area:

$$1 \cdot r^2$$

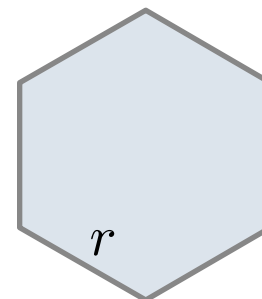
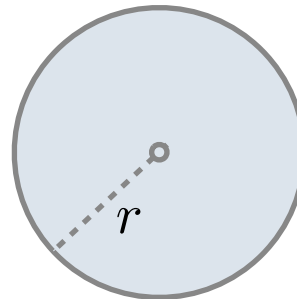
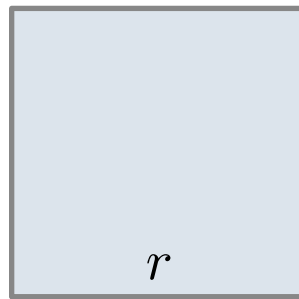
$$\pi \cdot r^2$$

$$\frac{3\sqrt{3}}{2} \cdot r^2$$

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:



Area:

$$1 \cdot r^2$$

$$\pi \cdot r^2$$

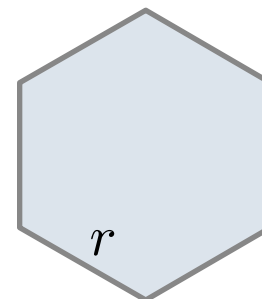
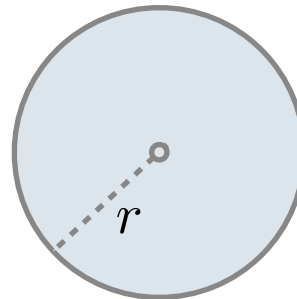
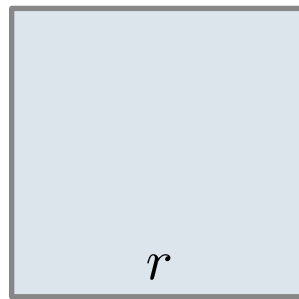
$$\frac{3\sqrt{3}}{2} \cdot r^2$$

Finding common structure allows for shared implementation

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)

Higher-Order Functions

Generalizing Over Computational Processes

Generalizing Over Computational Processes

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

Generalizing Over Computational Processes

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

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

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

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

Generalizing Over Computational Processes

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

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

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

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

Generalizing Over Computational Processes

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

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

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

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

Generalizing Over Computational Processes

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

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

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

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

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)

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
```


Summation Example

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

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

```
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
```

Summation Example

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

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

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

A formal parameter that will
be bound to a function

```
>>> summation(5, cube)  
225  
"""
```

```
total, k = 0, 1  
while k <= n:  
    total, k = total + term(k), k + 1  
return total
```

Summation Example

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

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

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

A formal parameter that will
be bound to a function

```
>>> summation(5, cube)  
225  
"""
```

```
    total, k = 0, 1  
    while k <= n:  
        total, k = total + term(k), k + 1  
    return total
```

The function bound to term
gets called here

Summation Example

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

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

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

A formal parameter that will
be bound to a function

```
>>> summation(5, cube)  
225  
"""
```

The cube function is passed
as an argument value

```
    total, k = 0, 1  
    while k <= n:  
        total, k = total + term(k), k + 1  
    return total
```

The function bound to term
gets called here

Summation Example

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

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

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

A formal parameter that will
be bound to a function

```
>>> summation(5, cube)
```

```
225
```

```
"""
```

```
    total, k = 0, 1
```

```
    while k <= n:
```

```
        total, k = total + term(k), k + 1
```

```
    return total
```

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

Locally Defined Functions

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

Locally Defined Functions

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

```
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
```

Locally Defined Functions

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

A function that
returns a function

```
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
```

Locally Defined Functions

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

A function that
returns a function

```
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

Locally Defined Functions

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

A function that
returns a function

```
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

Locally Defined Functions

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

A function that
returns a function

```
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

Call Expressions as Operator Expressions

`make_adder(1) (2)`

Call Expressions as Operator Expressions

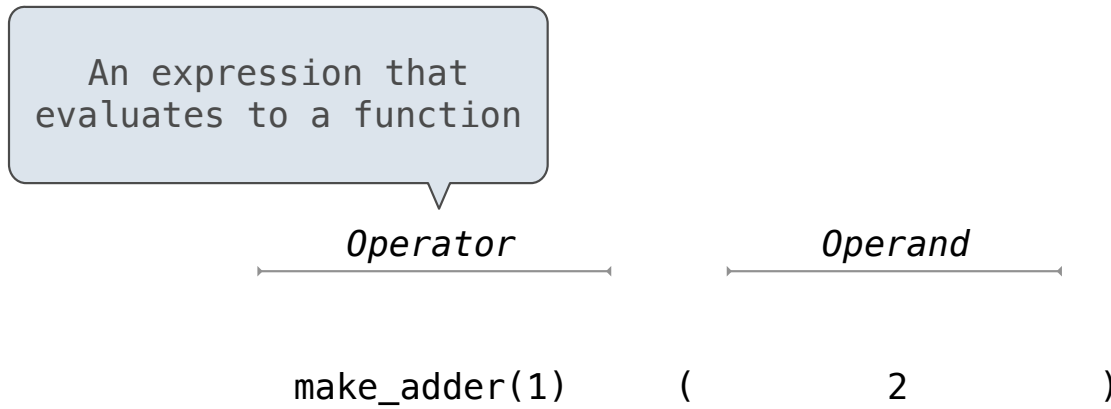
Operator

make_adder(1) (2)

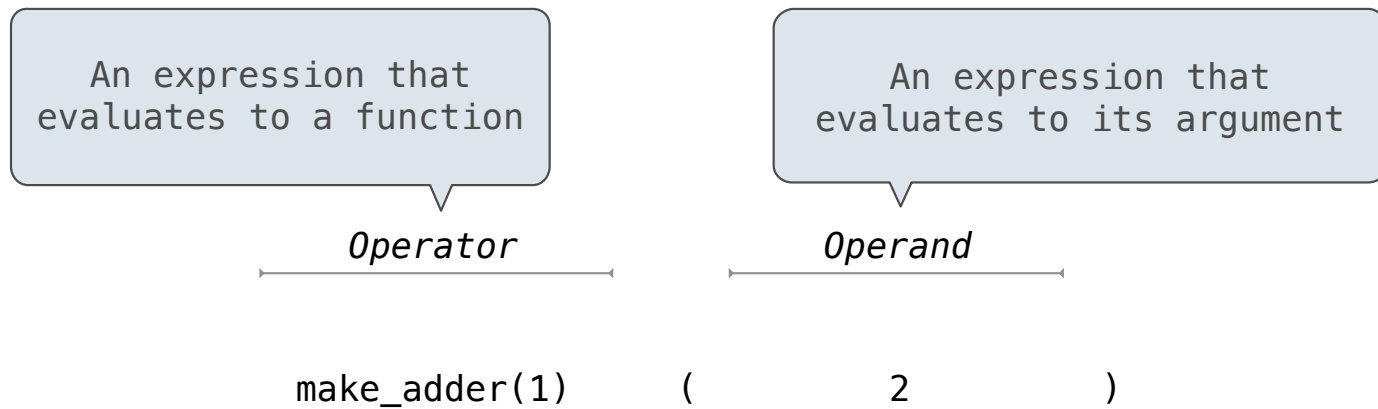
Call Expressions as Operator Expressions

<i>Operator</i>	<i>Operand</i>
<code>make_adder(1)</code>	<code>(2)</code>

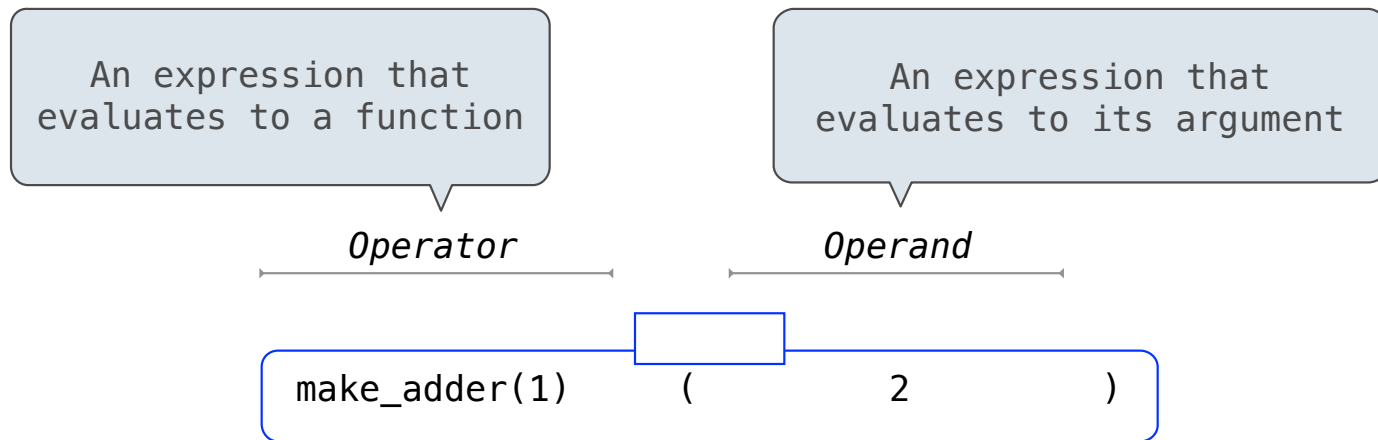
Call Expressions as Operator Expressions



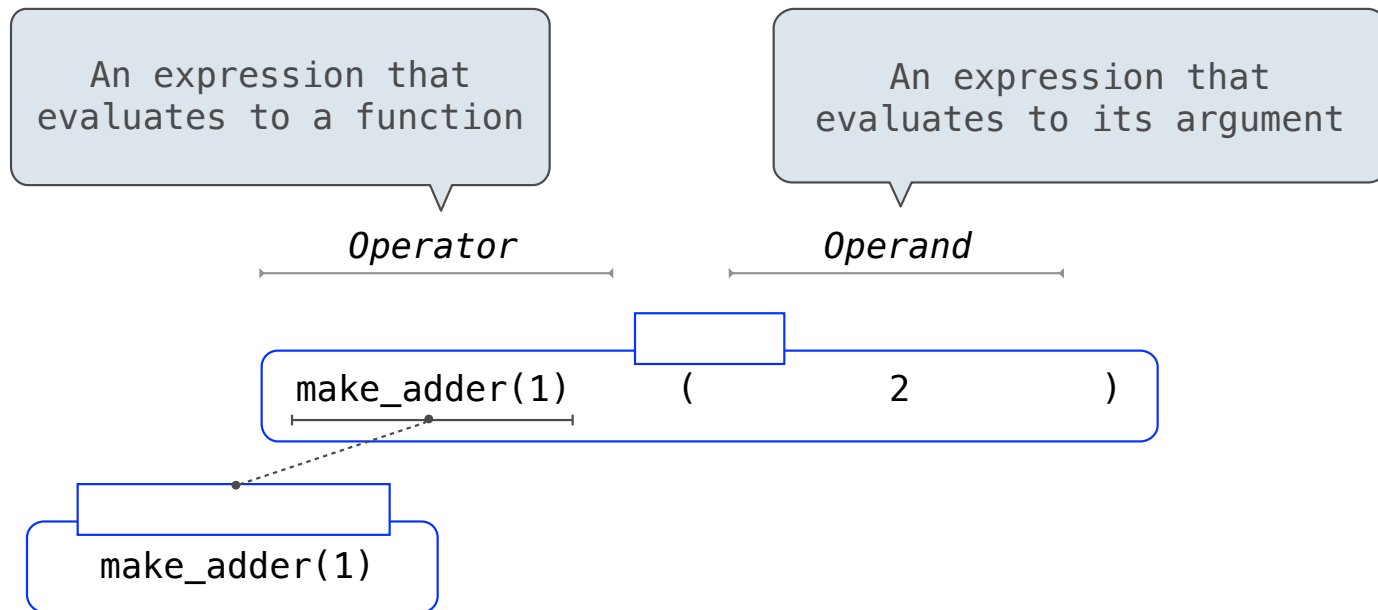
Call Expressions as Operator Expressions



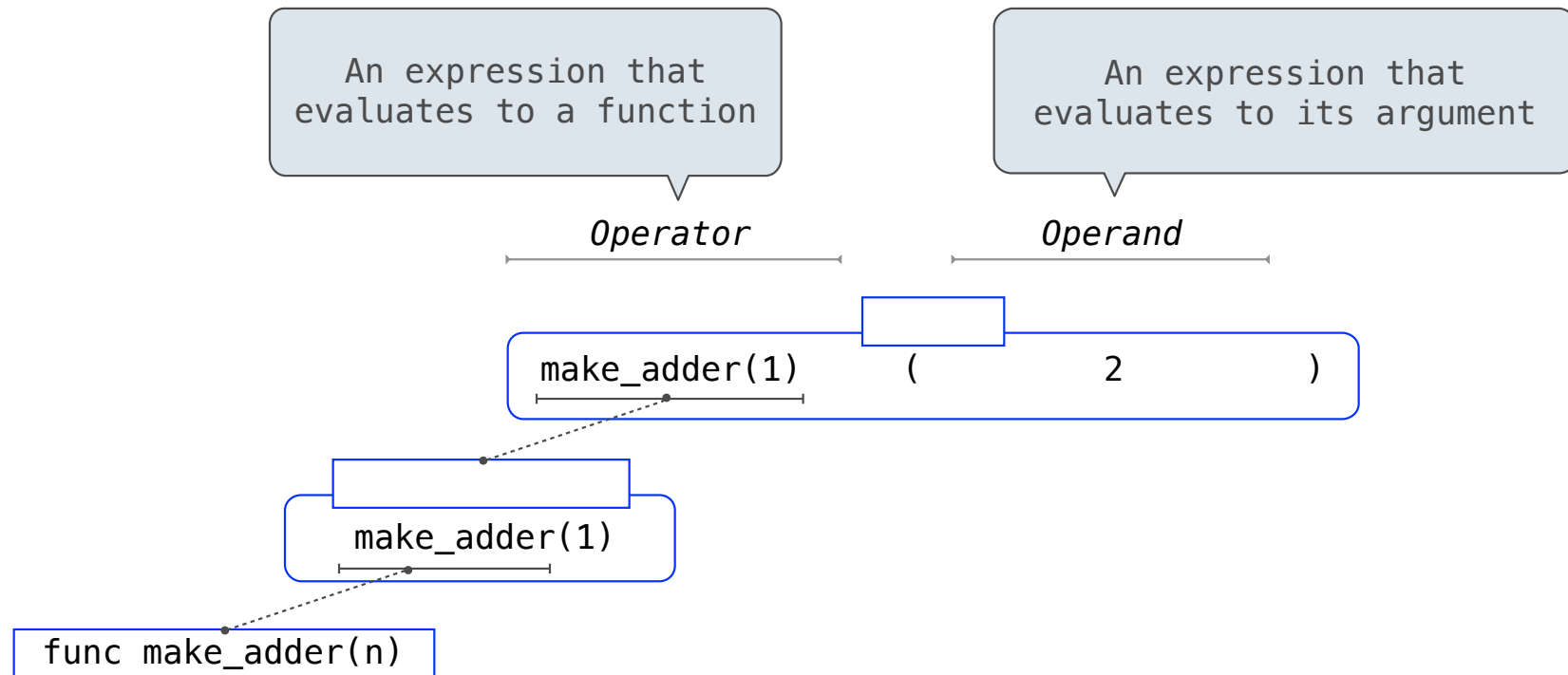
Call Expressions as Operator Expressions



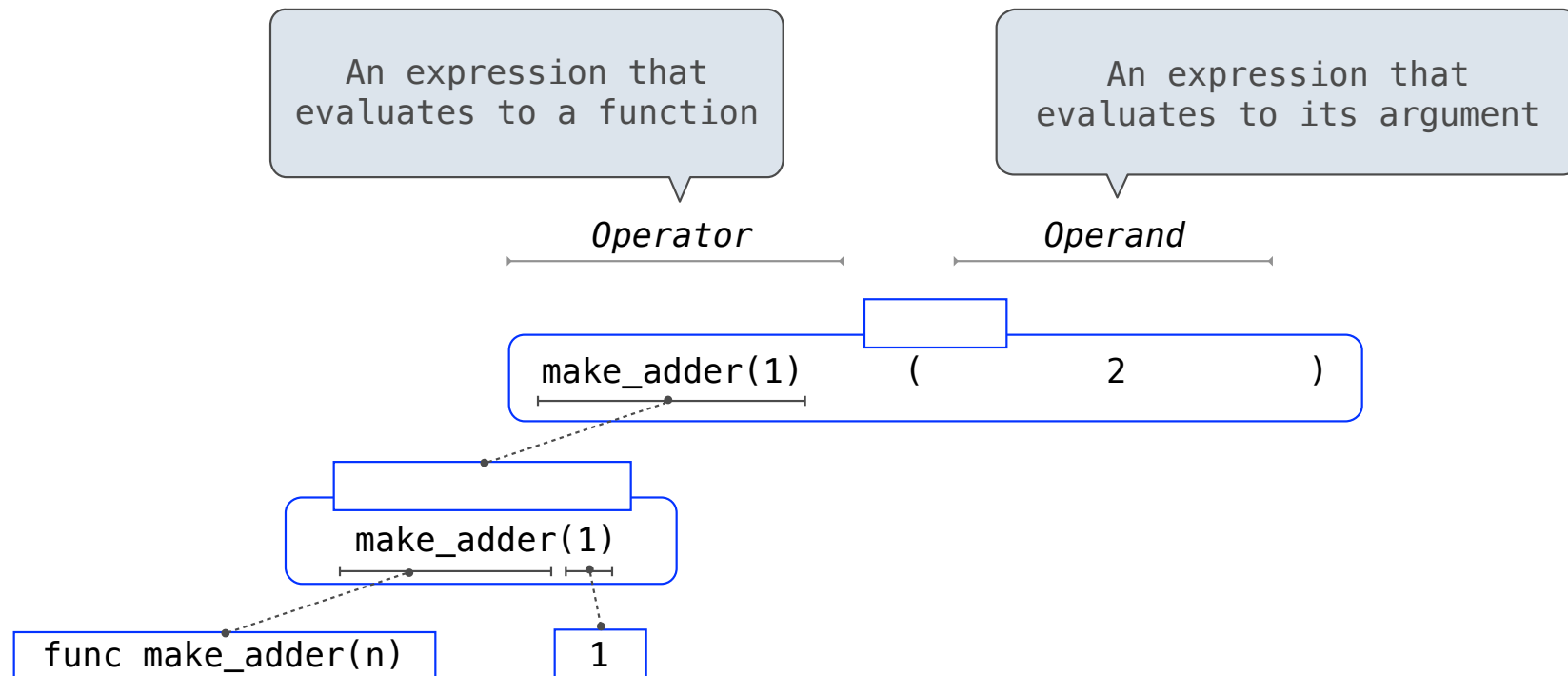
Call Expressions as Operator Expressions



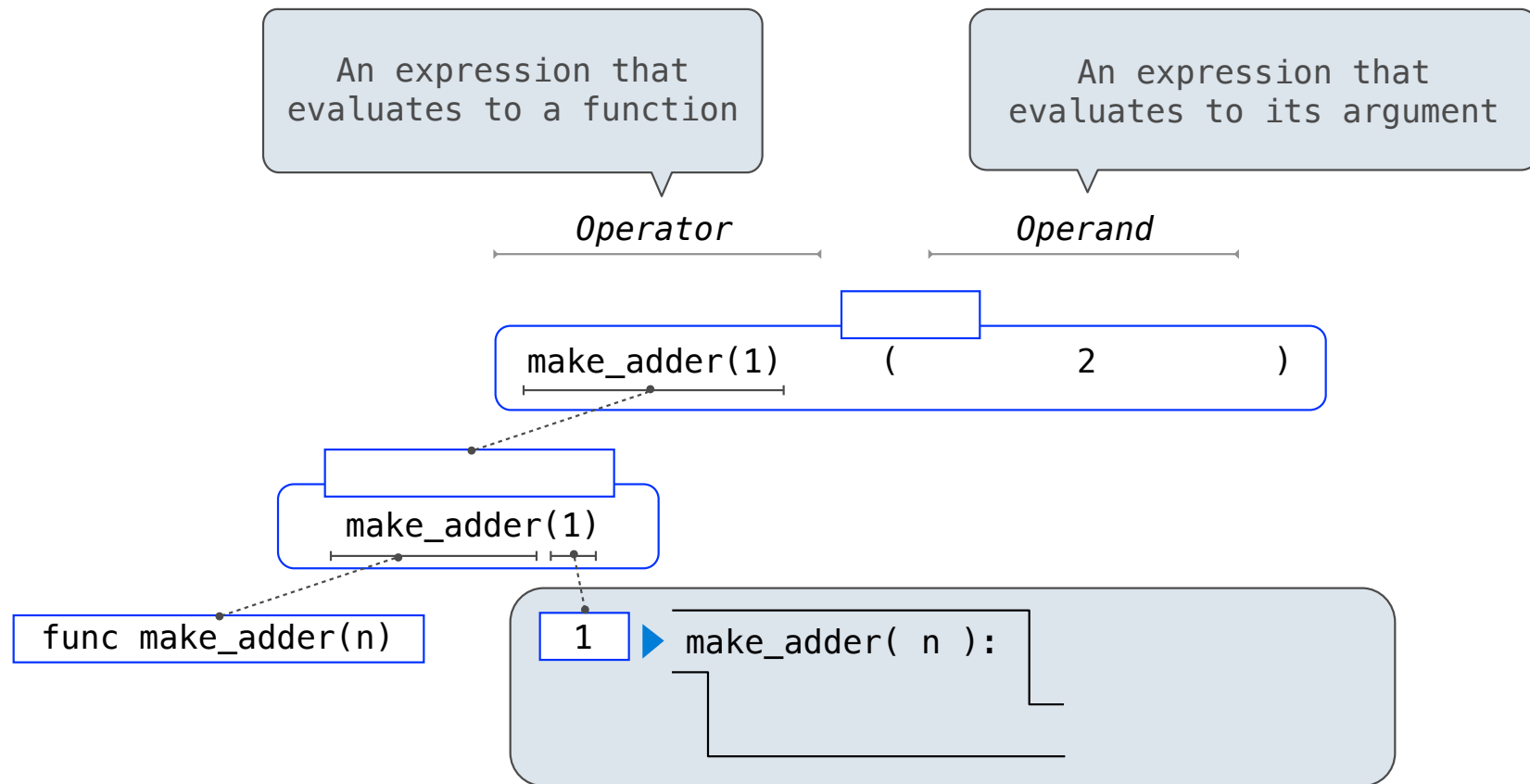
Call Expressions as Operator Expressions



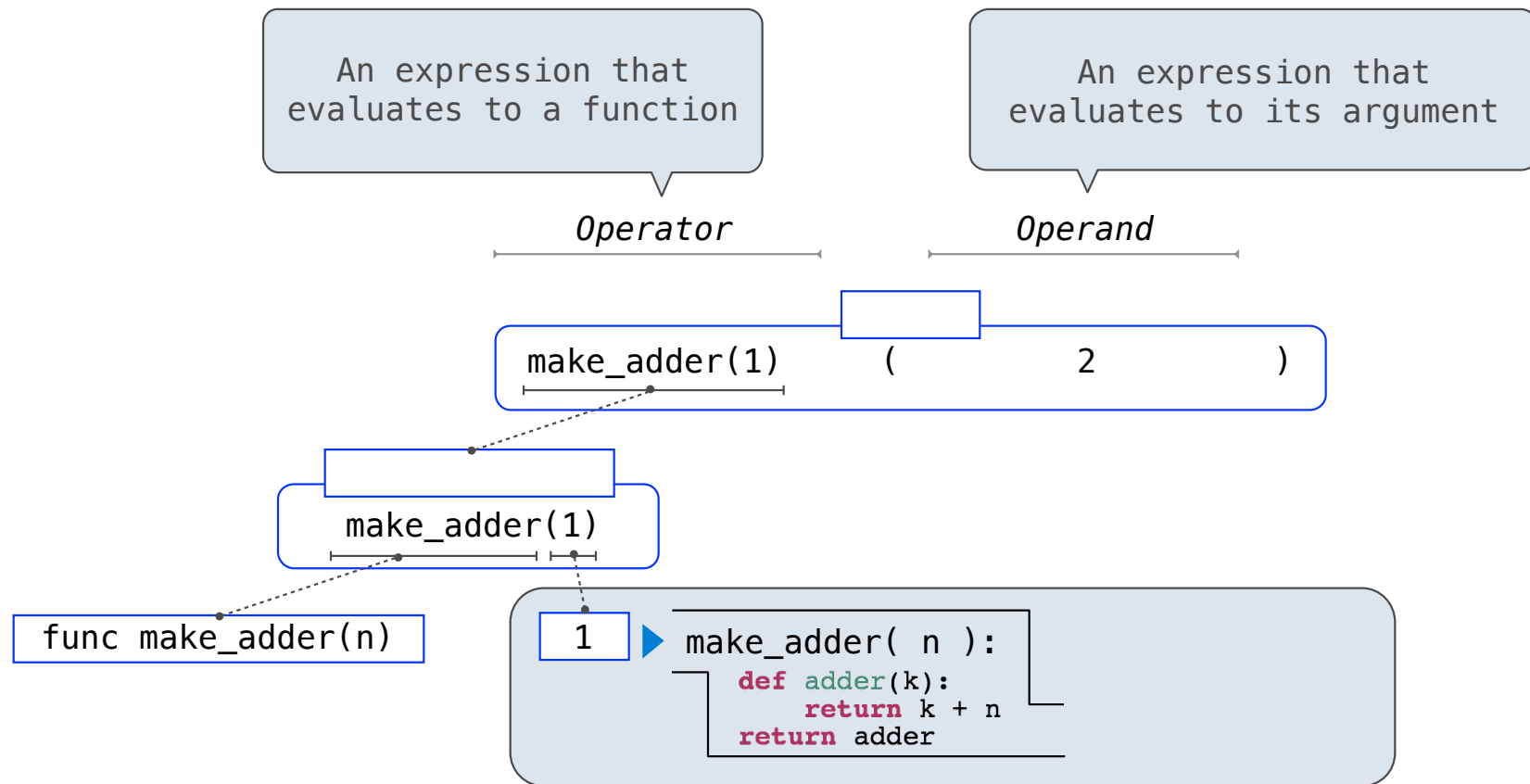
Call Expressions as Operator Expressions



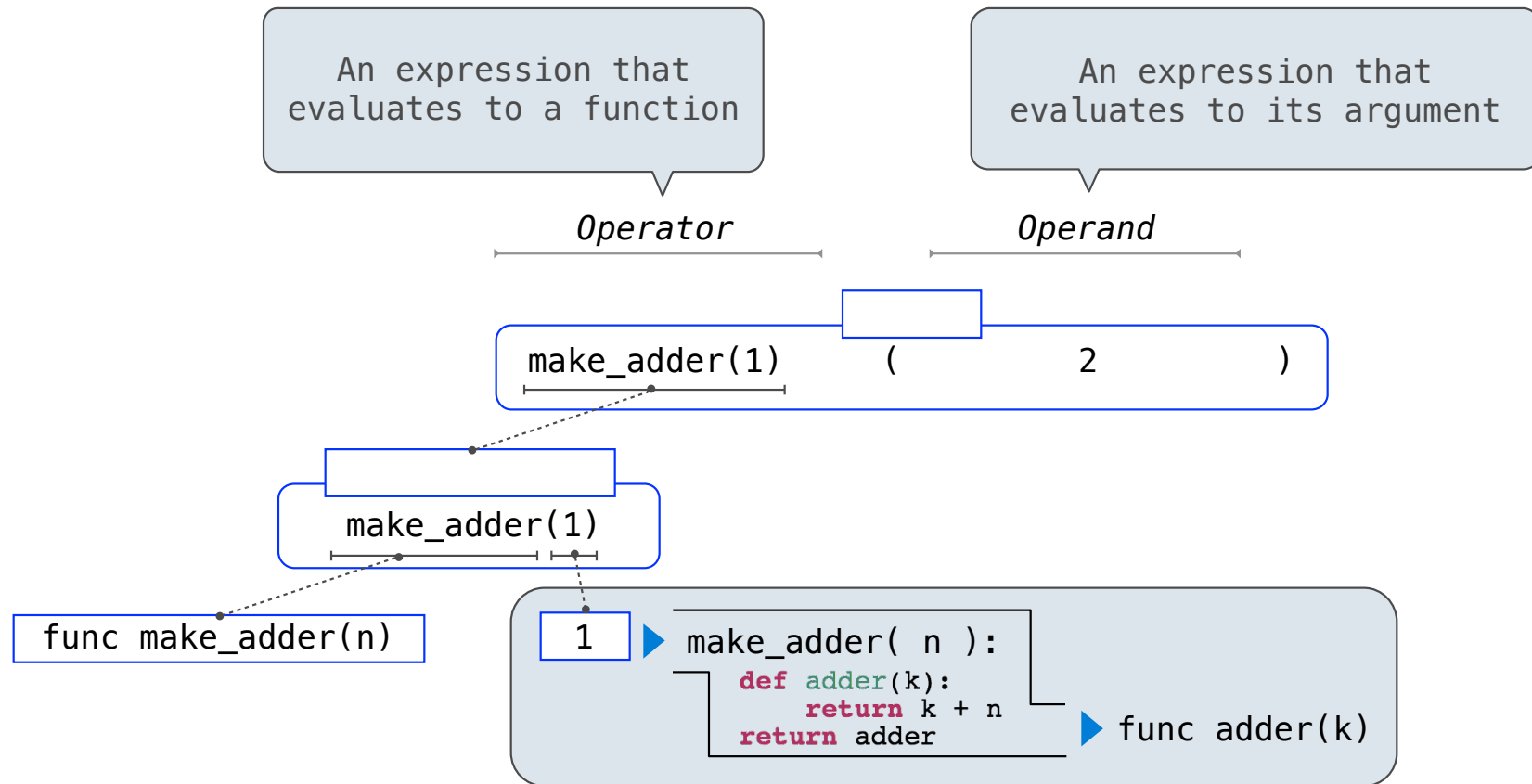
Call Expressions as Operator Expressions



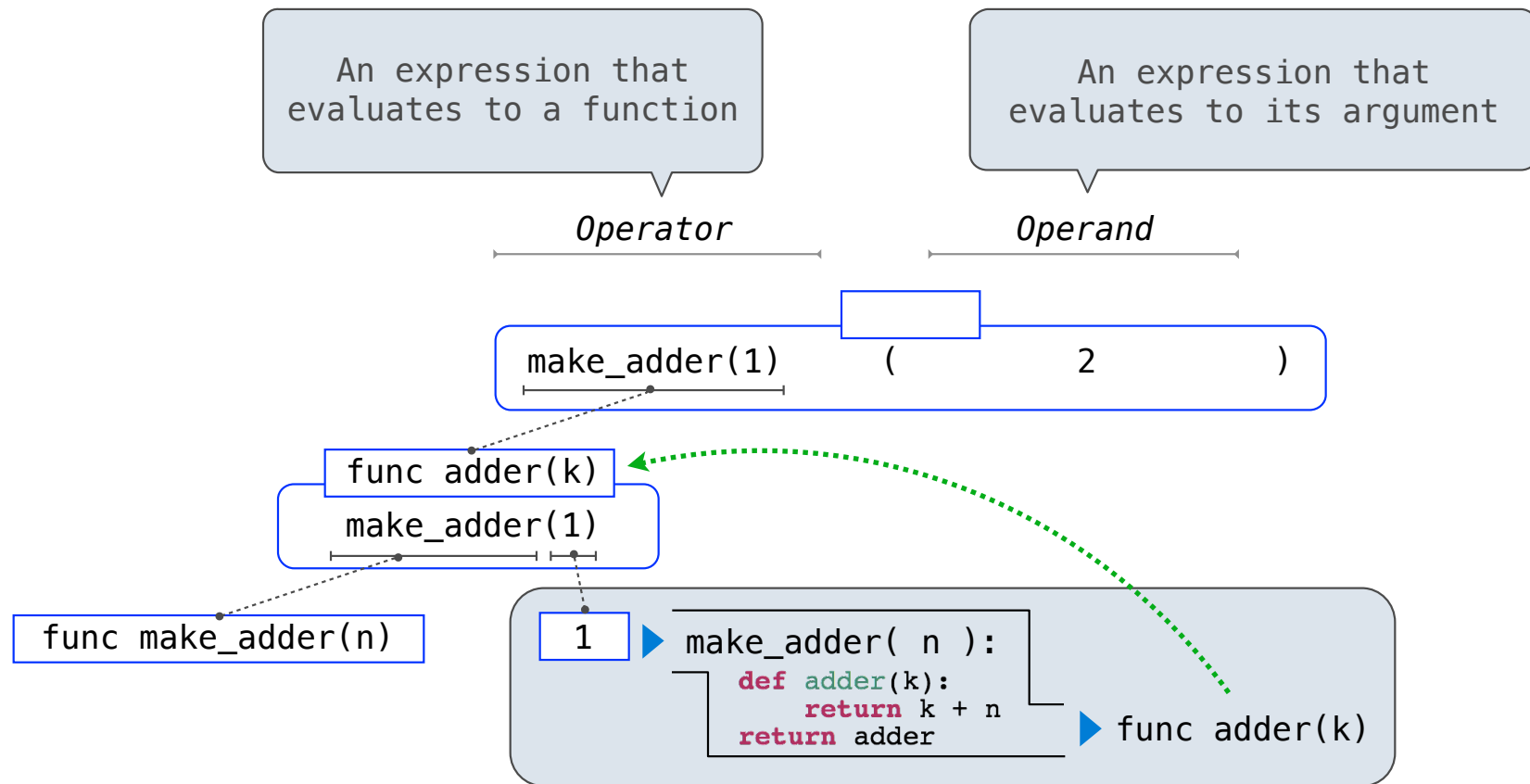
Call Expressions as Operator Expressions



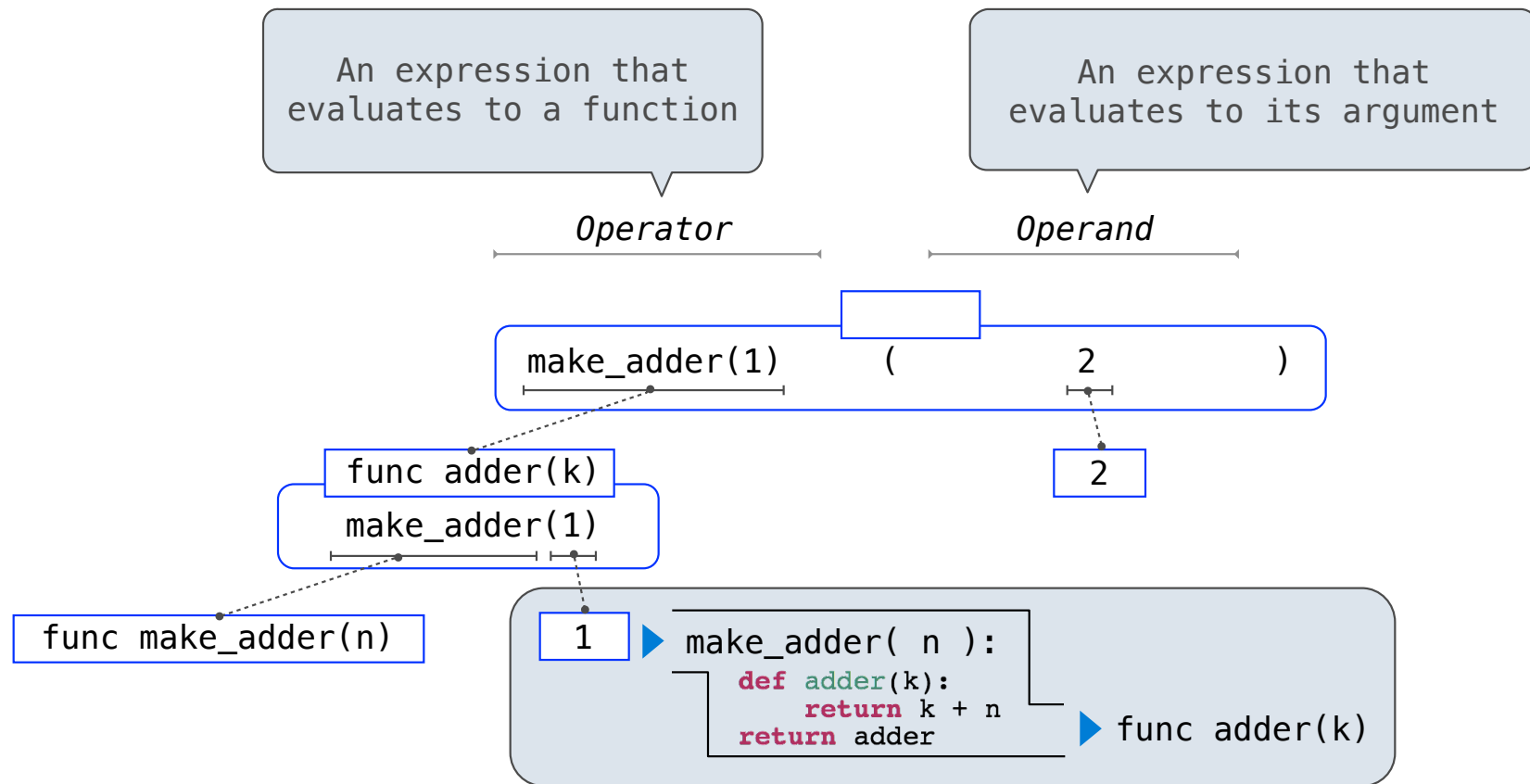
Call Expressions as Operator Expressions



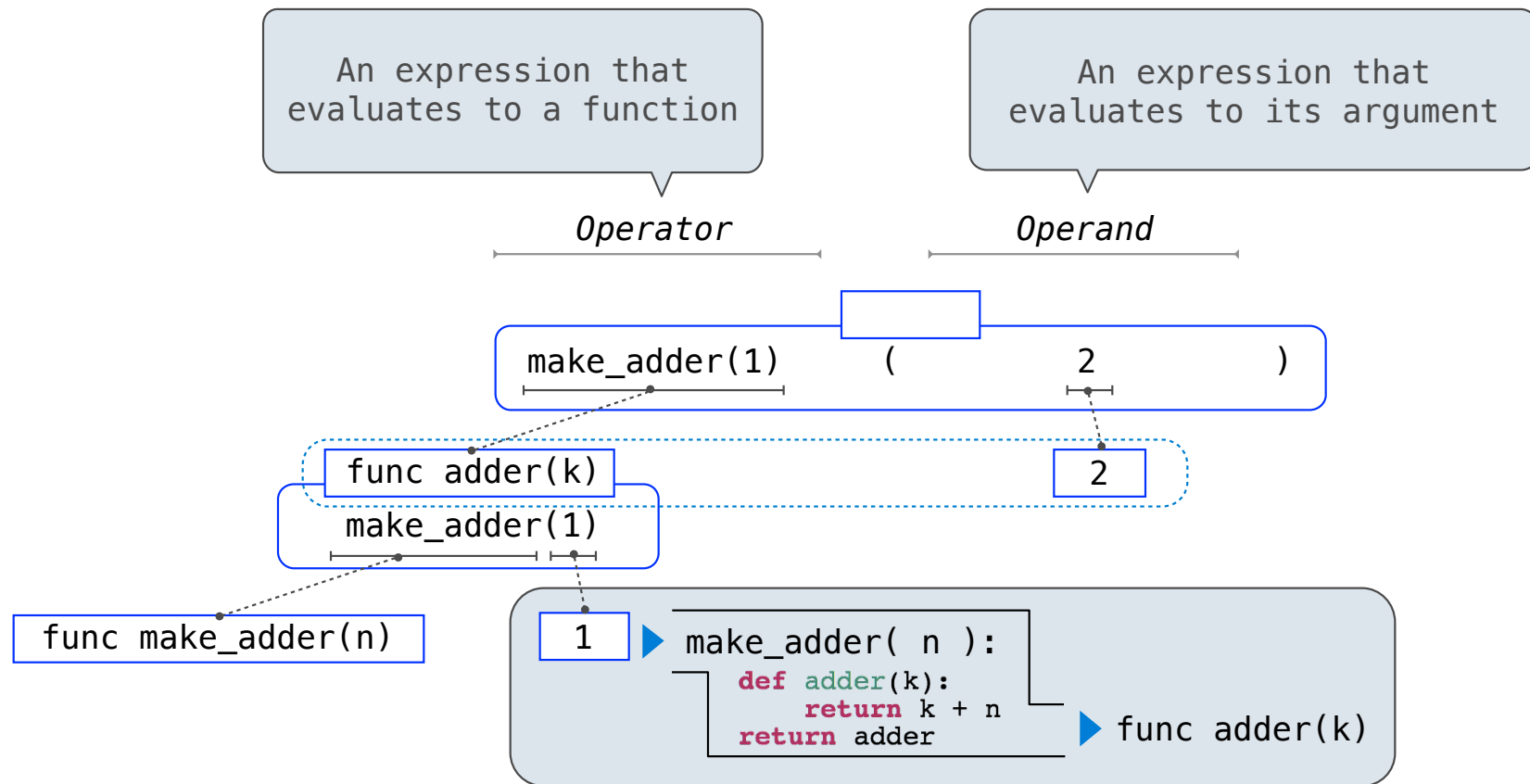
Call Expressions as Operator Expressions



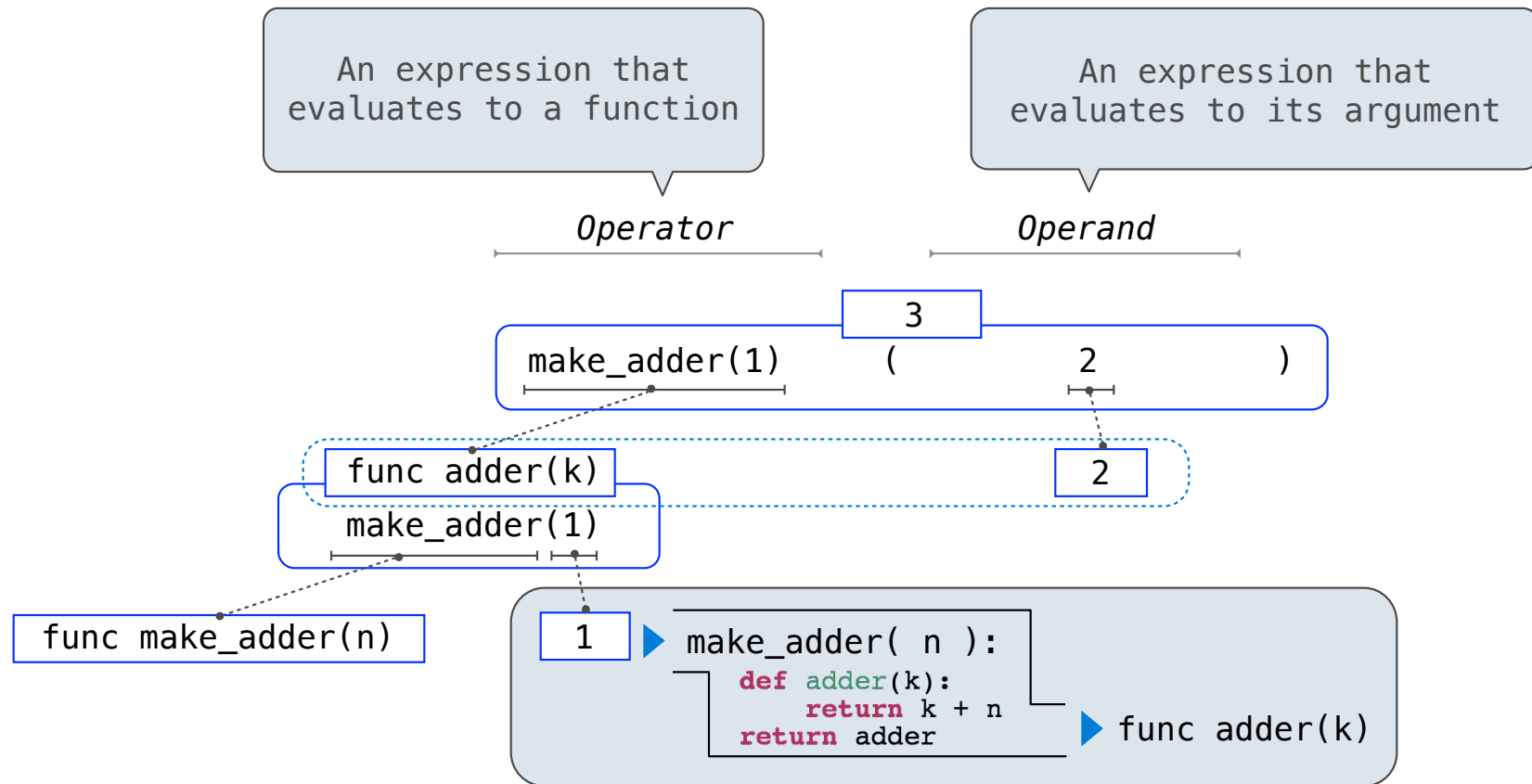
Call Expressions as Operator Expressions



Call Expressions as Operator Expressions



Call Expressions as Operator Expressions



The Purpose of Higher-Order Functions

The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

Higher-order function: A function that takes a function as an argument value or returns a function as a return value

The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

Higher-order function: A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:

The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

Higher-order function: A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:

- Express general methods of computation

The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

Higher-order function: A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:

- Express general methods of computation
- Remove repetition from programs

The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

Higher-order function: A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:

- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions

The Game of Hog

(Demo)