61A Lecture 9

Friday, September 20

Announcements

Announcements

- Midterm 1 is on Monday 9/23 from 7pm to 9pm

Announcements

- Midterm 1 is on Monday 9/23 from 7pm to 9pm
    - 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel

Announcements

- Midterm 1 is on Monday 9/23 from 7pm to 9pm
  - 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
  - HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB

Announcements

- Midterm 1 is on Monday 9/23 from 7pm to 9pm
  - 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
  - HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
  - Extra weekend office hours announced on Piazza

Announcements

- Midterm 1 is on Monday 9/23 from 7pm to 9pm
  - 2 review sessions on Saturday 9/21 2pm-4pm and 4pm-6pm in 1 Pimentel
  - HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
  - Extra weekend office hours announced on Piazza
  - Cannot attend? Fill out the conflict form by Friday 9/20 @ 11:59pm!

Announcements

- Midterm 1 is on Monday 9/23 from 7pm to 9pm
  - 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
  - HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
  - Extra weekend office hours announced on Piazza
  - Cannot attend? Fill out the conflict form by Friday 9/20 @ 11:59pm!
- No lab next week: Monday 9/23, Tuesday 9/24, or Wednesday 9/25

Announcements

- Midterm 1 is on Monday 9/23 from 7pm to 9pm
  - 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
  - HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
  - Extra weekend office hours announced on Piazza
  - Cannot attend? Fill out the conflict form by Friday 9/20 @ 11:59pm!
- No lab next week: Monday 9/23, Tuesday 9/24, or Wednesday 9/25
- Homework 3 due Tuesday 10/1 @ 11:59pm

Announcements

- Midterm 1 is on Monday 9/23 from 7pm to 9pm
  - 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
  - HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
  - Extra weekend office hours announced on Piazza
  - Cannot attend? Fill out the conflict form by Friday 9/20 @ 11:59pm!
- No lab next week: Monday 9/23, Tuesday 9/24, or Wednesday 9/25
- Homework 3 due Tuesday 10/1 @ 11:59pm
- Optional Hog strategy contest ends Thursday 10/3 @ 11:59pm

Abstraction

Functional Abstractions

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

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

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

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

What does sum_squares need to know about square?

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.

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

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

What does sum_squares need to know about square?

- Square takes one argument. **Yes**

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

```
def sum_squares(x, y):
    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.

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

```
def sum_squares(x, y):
    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**

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

```
def sum_squares(x, y):
    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.

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**

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

```
def sum_squares(x, y):
    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.

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

```
def sum_squares(x, y):
    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**

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

```
def sum_squares(x, y):
    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):
    return pow(x, 2)
```

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

```
def sum_squares(x, y):
    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):
    return pow(x, 2)
```

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

Functional Abstractions

```
def square(x):
    return mul(x, x)
```

```
def sum_squares(x, y):
    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):
    return pow(x, 2)
```

```
def square(x):
    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

Choosing Names

Names typically *don’t* matter for correctness

but

they matter a lot for composition

Choosing Names

Names typically *don’t* matter for correctness

but

they matter a lot for composition

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

Choosing Names

Names typically *don’t* matter for correctness

but

they matter a lot for composition

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.

Choosing Names

Names typically *don’t* matter for correctness

but

they matter a lot for composition

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).

Choosing Names

Names typically *don’t* matter for correctness

but

they matter a lot for composition

From:	**To:**

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).

Choosing Names

Names typically *don't* matter for correctness

but

they matter a lot for composition

From:	**To:**
true_false	rolled_a_one

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).

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

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).

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
play_helper	take_turn

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).

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
play_helper	take_turn
my_int	num_rolls

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).

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
play_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

Which Values Deserve a Name

Repeated compound expressions:

Which Values Deserve a Name

Repeated compound expressions:

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

Which Values Deserve a 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
```

Which Values Deserve a 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:

Which Values Deserve a 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:

```
x = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
```

Which Values Deserve a 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:

```
x = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
```

```
discriminant = sqrt(square(b) - 4 * a * c)
x = (-b + discriminant) / (2 * a)
```

Which Values Deserve a 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:

```
x = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
```

```
discriminant = sqrt(square(b) - 4 * a * c)
x = (-b + discriminant) / (2 * a)
```

More Naming Tips

Which Values Deserve a 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:

```
x = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
```

```
discriminant = sqrt(square(b) - 4 * a * c)
x = (-b + discriminant) / (2 * a)
```

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)
  ```

Which Values Deserve a 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:

```
x = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
```

```
discriminant = sqrt(square(b) - 4 * a * c)
x = (-b + discriminant) / (2 * a)
```

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
```

Which Values Deserve a 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
```

PRACTICAL GUIDELINES

Meaningful parts of complex expressions:

```
x = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
```

```
discriminant = sqrt(square(b) - 4 * a * c)
x = (-b + discriminant) / (2 * a)
```

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
```

Testing

Test-Driven Development

Test-Driven Development

Write the test of a function before you write the function.

Test-Driven Development

Write the test of a function before you write the function.

A test will clarify the domain, range, & behavior of a function.

Test-Driven Development

Write the test of a function before you write the function.

A test will clarify the domain, range, & behavior of a function.

Tests can help identify tricky edge cases.

Test-Driven Development

Write the test of a function before you write the function.

A test will clarify the domain, range, & behavior of a function.

Tests can help identify tricky edge cases.

Develop incrementally and test each piece before moving on.

Test-Driven Development

Write the test of a function before you write the function.

A test will clarify the domain, range, & behavior of a function.

Tests can help identify tricky edge cases.

Develop incrementally and test each piece before moving on.

You can't depend upon code that hasn't been tested.

Test-Driven Development

Write the test of a function before you write the function.

A test will clarify the domain, range, & behavior of a function.

Tests can help identify tricky edge cases.

Develop incrementally and test each piece before moving on.

You can't depend upon code that hasn't been tested.

Run your old tests again after you make new changes.

Test-Driven Development

Write the test of a function before you write the function.

A test will clarify the domain, range, & behavior of a function.

Tests can help identify tricky edge cases.

Develop incrementally and test each piece before moving on.

You can't depend upon code that hasn't been tested.

Run your old tests again after you make new changes.

Run your code interactively.

Test-Driven Development

Write the test of a function before you write the function.

A test will clarify the domain, range, & behavior of a function.

Tests can help identify tricky edge cases.

Develop incrementally and test each piece before moving on.

You can't depend upon code that hasn't been tested.

Run your old tests again after you make new changes.

Run your code interactively.

Don't be afraid to experiment with a function after you write it.

Test-Driven Development

Write the test of a function before you write the function.

A test will clarify the domain, range, & behavior of a function.

Tests can help identify tricky edge cases.

Develop incrementally and test each piece before moving on.

You can't depend upon code that hasn't been tested.

Run your old tests again after you make new changes.

Run your code interactively.

Don't be afraid to experiment with a function after you write it.

Interactive sessions can become doctests. Just copy and paste.

Test-Driven Development

Write the test of a function before you write the function.

A test will clarify the domain, range, & behavior of a function.

Tests can help identify tricky edge cases.

Develop incrementally and test each piece before moving on.

You can't depend upon code that hasn't been tested.

Run your old tests again after you make new changes.

Run your code interactively.

Don't be afraid to experiment with a function after you write it.

Interactive sessions can become doctests. Just copy and paste.

(Demo)

Decorators

Function Decorators

(demo)

Function Decorators

(demo)

```
@trace1
def triple(x):
    return 3 * x
```

Function Decorators

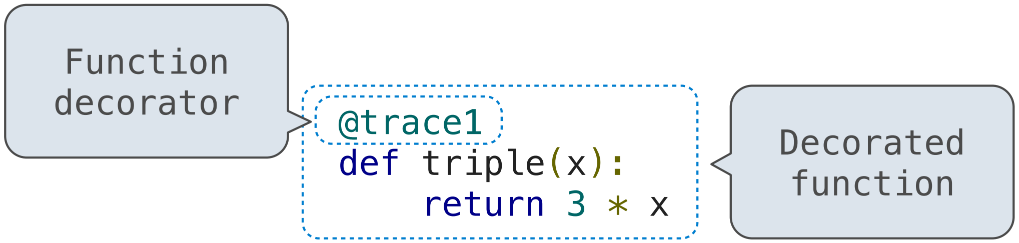
(demo)

Function decorator

```
@trace1
def triple(x):
    return 3 * x
```

Function Decorators

(demo)



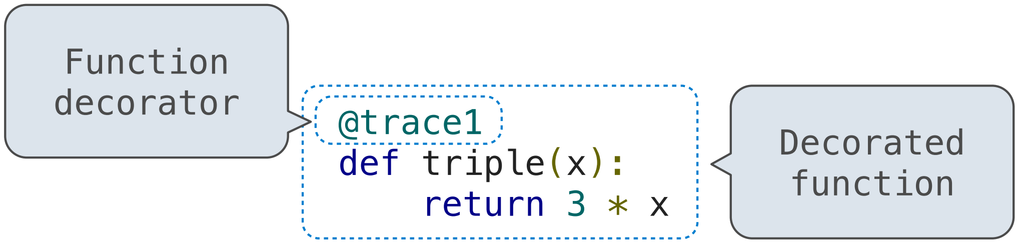
Function decorator

```
@trace1
def triple(x):
    return 3 * x
```

Decorated function

Function Decorators

(demo)



Function decorator

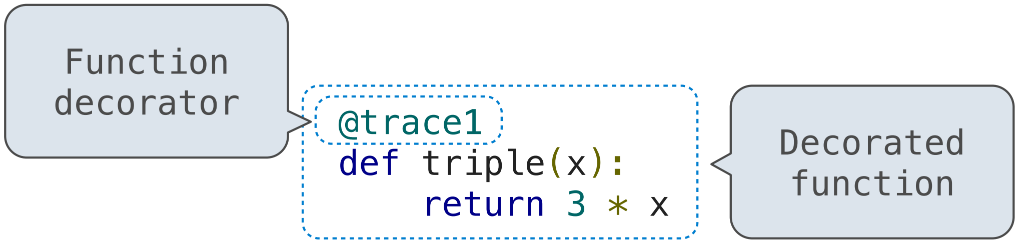
```
@trace1
def triple(x):
    return 3 * x
```

Decorated function

is identical to

Function Decorators

(demo)



Function decorator

```
@trace1
def triple(x):
    return 3 * x
```

Decorated function

is identical to

```
def triple(x):
    return 3 * x
triple = trace1(triple)
```

Function Decorators

(demo)

Function decorator

```
@trace1
def triple(x):
    return 3 * x
```

Decorated function

is identical to

Why not just use this?

```
def triple(x):
    return 3 * x
triple = trace1(triple)
```

Review

What Would Python Print?

What Would Python Print?

```
The print function returns None.  It also displays its arguments
(separated by spaces) when it is called.
```

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)		

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)	None	

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)	None	5

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
print(add(3, 4), print(5))		

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
print($\underset{7}{\underline{\texttt{add(3, 4)}}}$, print(5))		

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$		

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$		5

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$	None	5

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$	None	5 7 None

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{\texttt{7}}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$	None	5 7 None

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$	None	5 7 None
delay(delay)()(6)()		

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$	None	5 7 None
delay(delay)()(6)()		

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{\texttt{7}}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$	None	5 7 None
delay(delay)()(6)()		

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

> A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

> Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$	None	5 7 None
$\underline{\texttt{delay(delay)()(6)}}\texttt{()}$		

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
print($\underbrace{\texttt{add(3, 4)}}_{7}$, $\underbrace{\texttt{print(5)}}_{\texttt{None}}$)	None	5 7 None
$\underline{\underline{\texttt{delay(delay)()}}\texttt{(6)}}\texttt{()}$		

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$	None	5 7 None
delay(delay)()(6)()		

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$	None	5 7 None
delay(delay)()(6)()		

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
print(add(3, 4), print(5)) 7 None	None	5 7 None
delay(delay)()(6)()		delayed

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\text{None}}\texttt{)}$	None	5 7 None
delay(delay)()(6)()		delayed delayed

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
`5`	5	
`print(5)`	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$	None	5 7 None
`delay(delay)()(6)()`	6	delayed delayed

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
`5`	5	
`print(5)`	None	5
`print(add(3, 4), print(5))` 7 None	None	5 7 None
`delay(delay)()(6)()`	6	delayed delayed
`print(delay(print)()(4))`		

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
print(add(3, 4), print(5)) 7 None	None	5 7 None
delay(delay)()(6)()	6	delayed delayed
print(delay(print)()(4))		delayed

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\texttt{None}}\texttt{)}$	None	5 7 None
delay(delay)()(6)()	6	delayed delayed
print(delay(print)()(4))		delayed 4

What Would Python Print?

```
The print function returns None.  It also displays its arguments
(separated by spaces) when it is called.
```

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
`5`	5	
`print(5)`	None	5
$\texttt{print(}\underbrace{\texttt{add(3, 4)}}_{7}\texttt{, }\underbrace{\texttt{print(5)}}_{\text{None}}\texttt{)}$	None	5 7 None
`delay(delay)()(6)()`	6	delayed delayed
`print(delay(print)()(4))`		delayed 4 None

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	And prints
5	5	
print(5)	None	5
print(add(3, 4), print(5)) 7 None	None	5 7 None
delay(delay)()(6)()	6	delayed delayed
print(delay(print)()(4))	None	delayed 4 None

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints

Example: http://goo.gl/NdrVqr

What Would Python Print?

```
The print function returns None.  It also displays its arguments
(separated by spaces) when it is called.
```

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1)		

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1)		

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1)		

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1)		

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1)		Matey

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1)		Matey

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1) *func square(x)*		Matey

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	**Evaluates to**	**And prints**
add(pirate(3)(square)(4), 1)		Matey
func square(x)		

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1)		Matey
func square(x)		
16		

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1)	17	Matey
func square(x)		
16		

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1)	17	Matey
func square(x)		
16		
pirate(pirate(pirate))(5)(7)		

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1) *func square(x)* *16*	17	Matey
pirate(pirate(pirate))(5)(7)		

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1) *func square(x)* *16*	17	Matey
pirate(pirate(pirate))(5)(7) *Identity function*		

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1) *func square(x)* *16*	17	Matey
pirate(pirate(pirate))(5)(7) *Identity function*		Matey Matey

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1) *func square(x)* *16*	17	Matey
pirate(pirate(pirate))(5)(7) *Identity function*		Matey Matey

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1)	17	Matey
func square(x)		
16		
pirate(pirate(pirate))(5)(7)		Matey Matey
Identity function		
5		

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

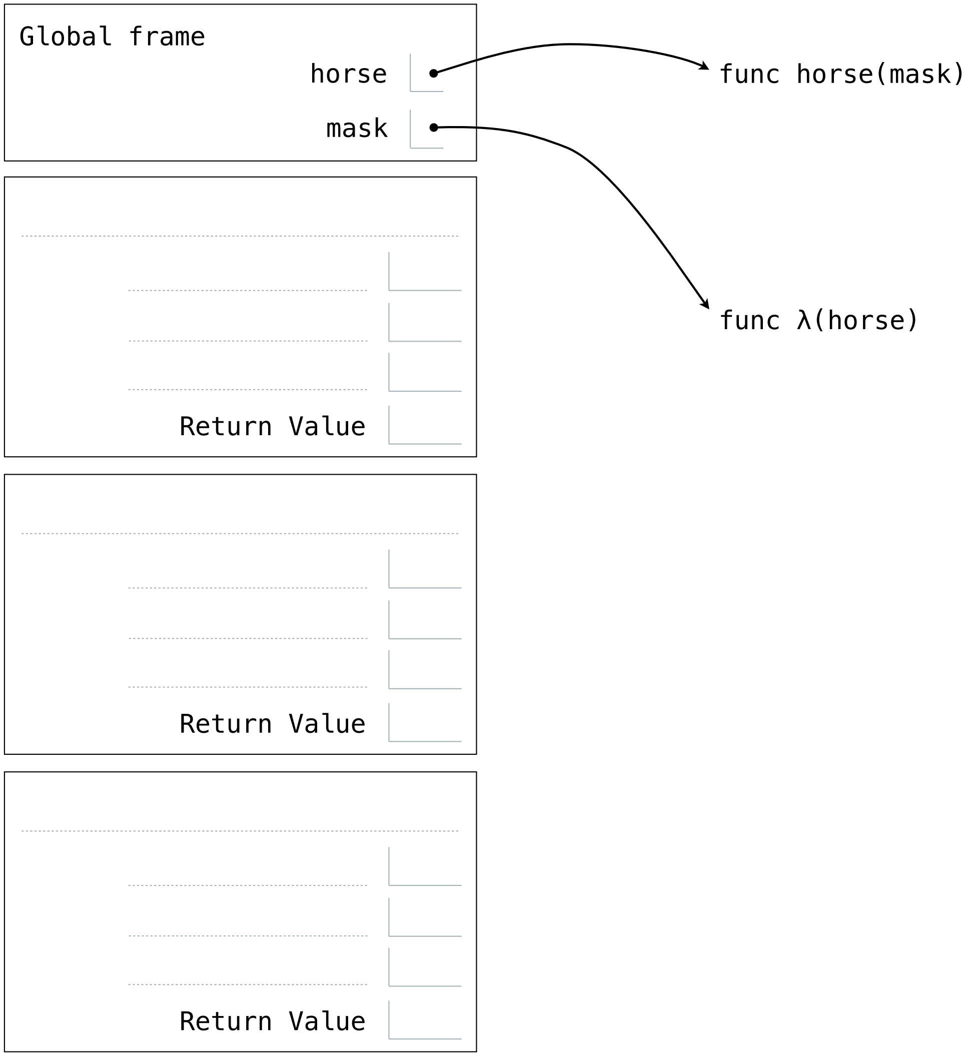
```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	And prints
add(pirate(3)(square)(4), 1) *func square(x)* *16*	17	Matey
pirate(pirate(pirate))(5)(7) *Identity function* *5*	Error	Matey Matey

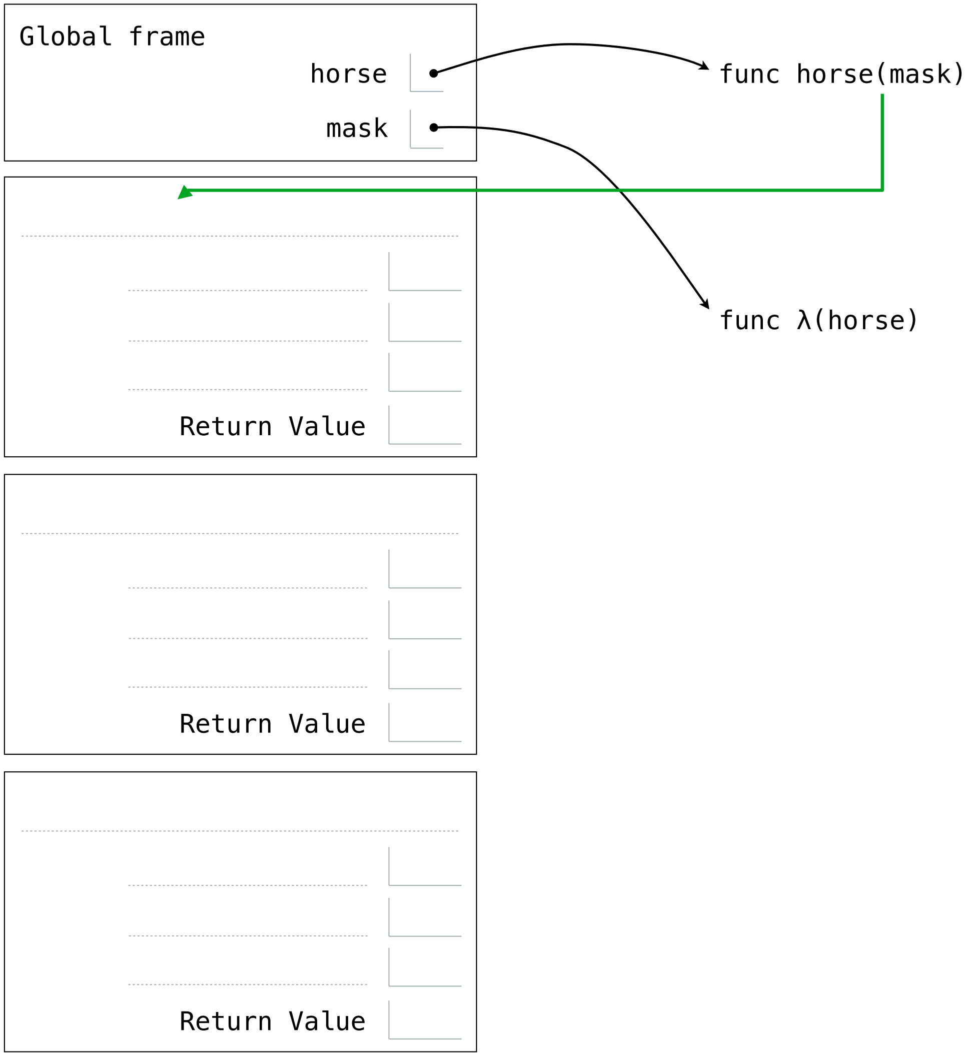
A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr

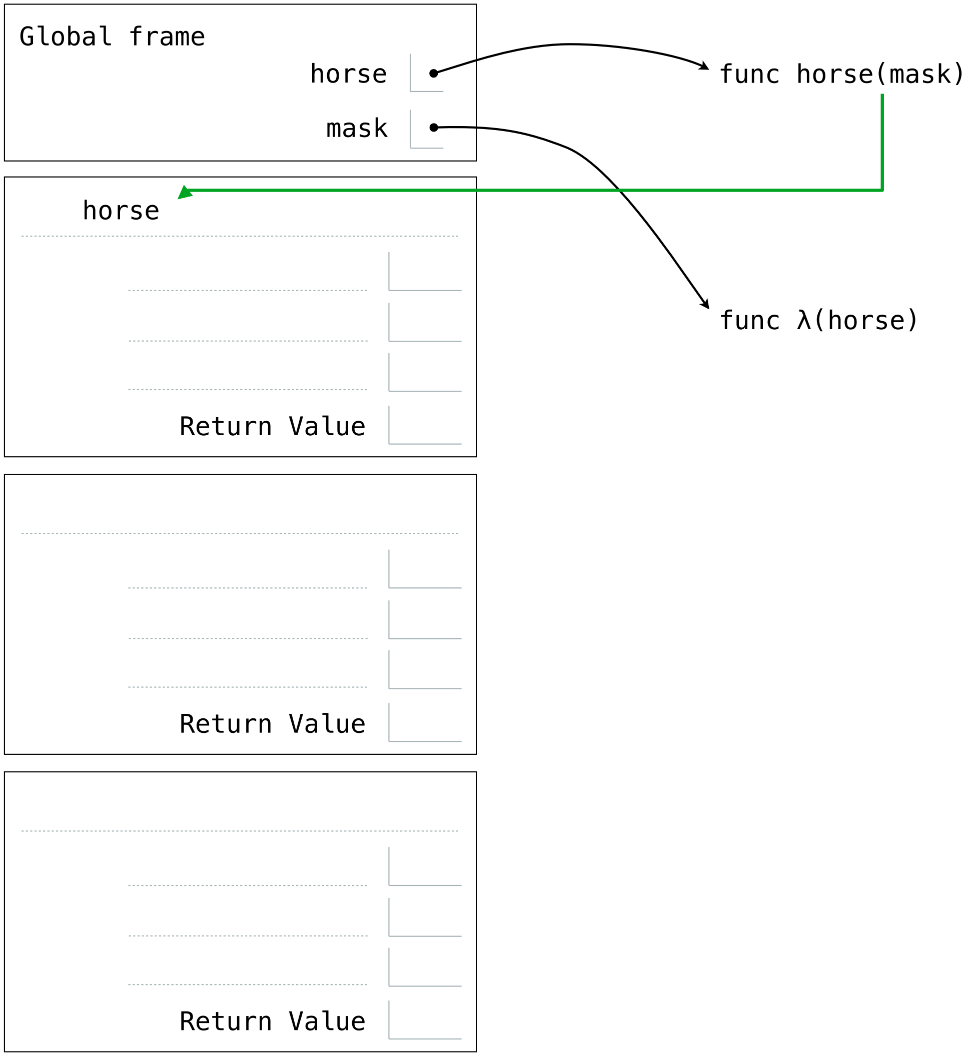
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



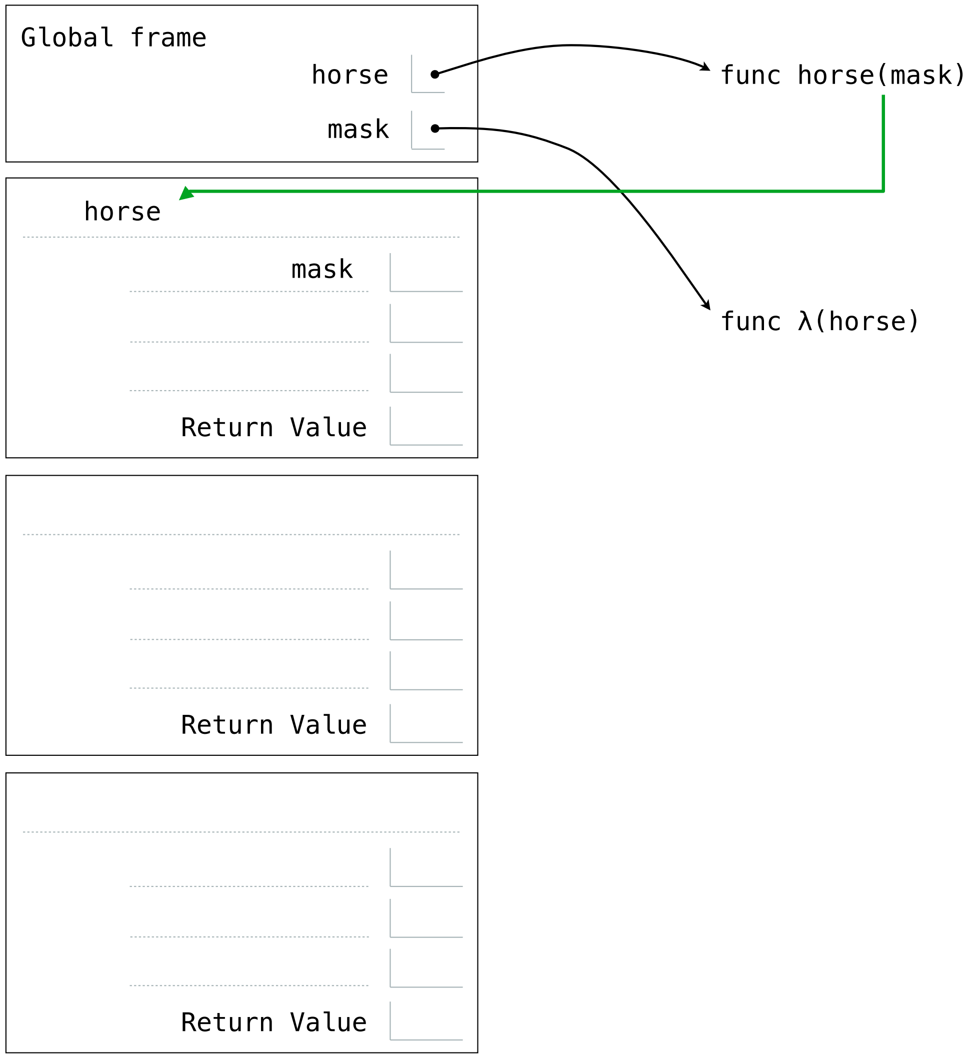
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



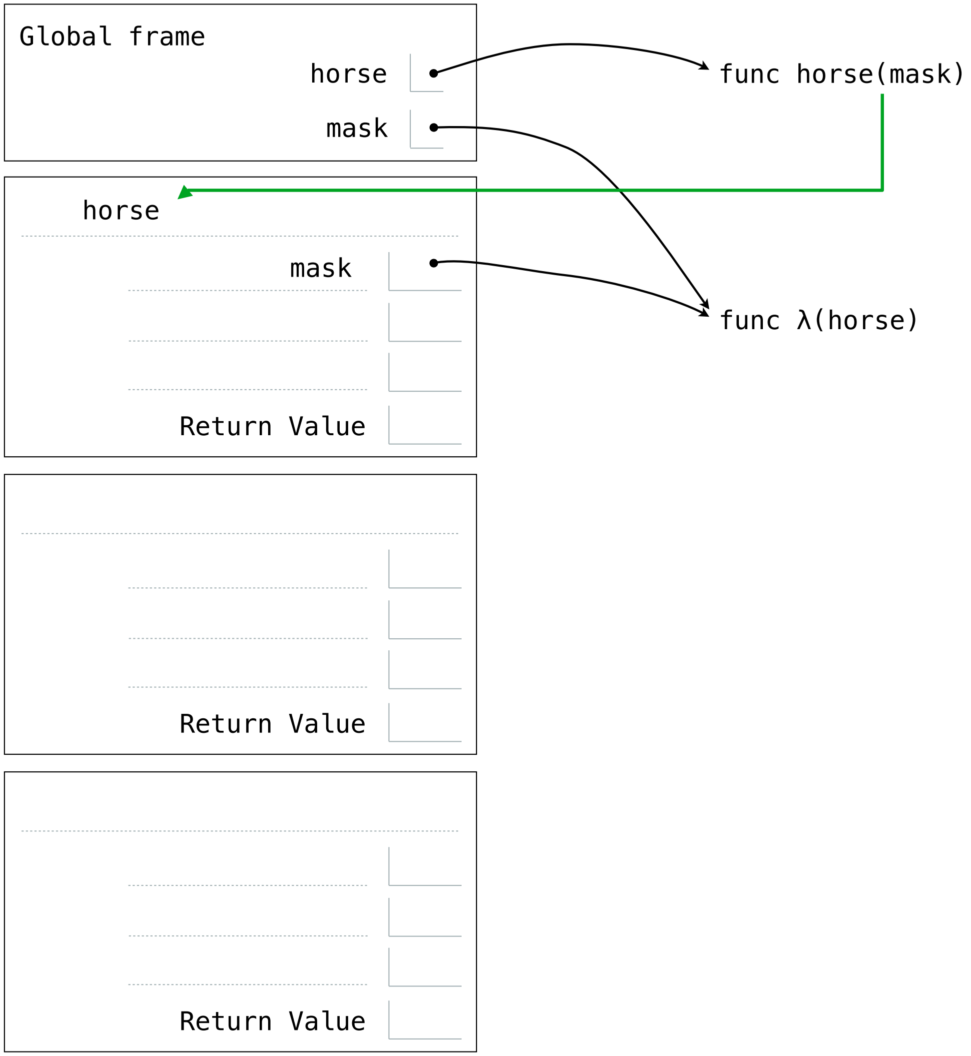
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



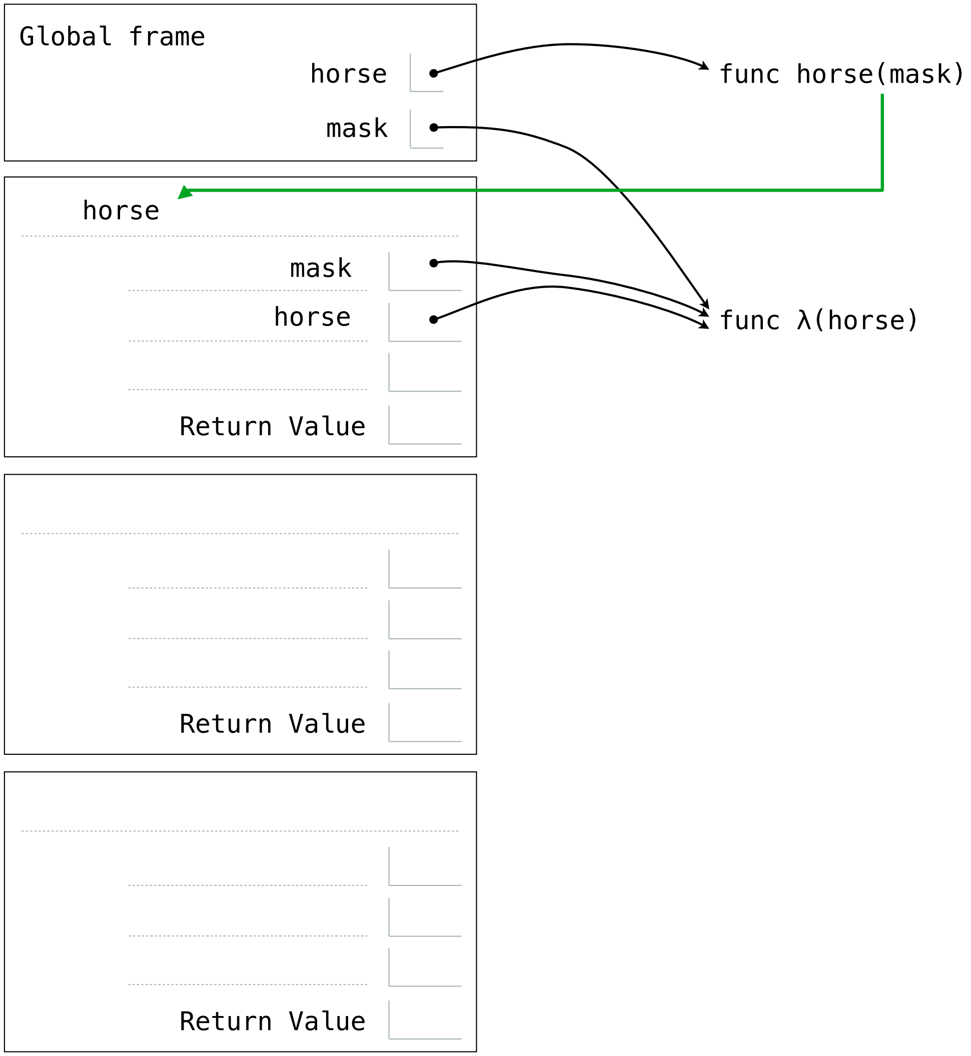
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



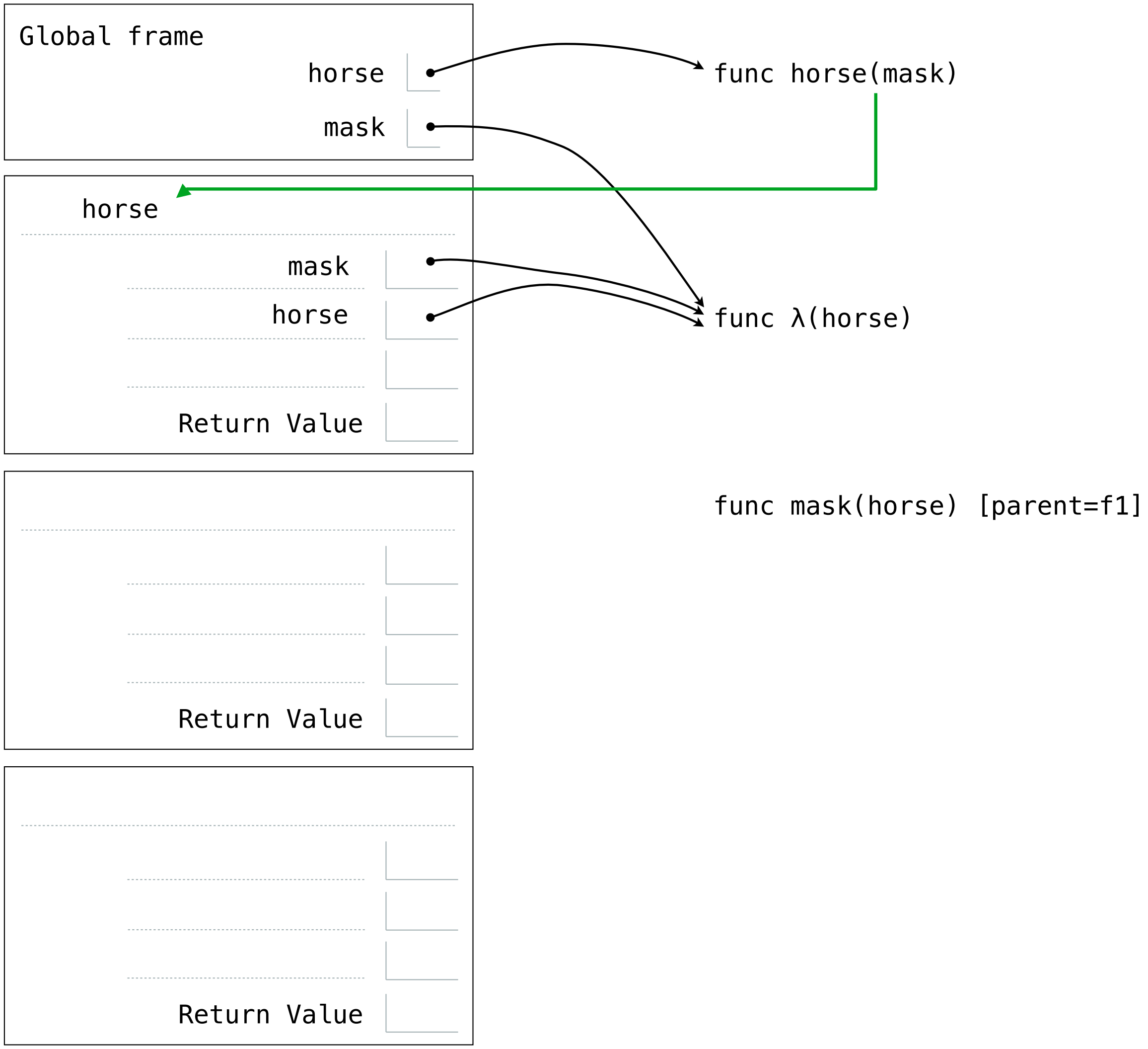
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

horse

mask

horse

Return Value

Return Value

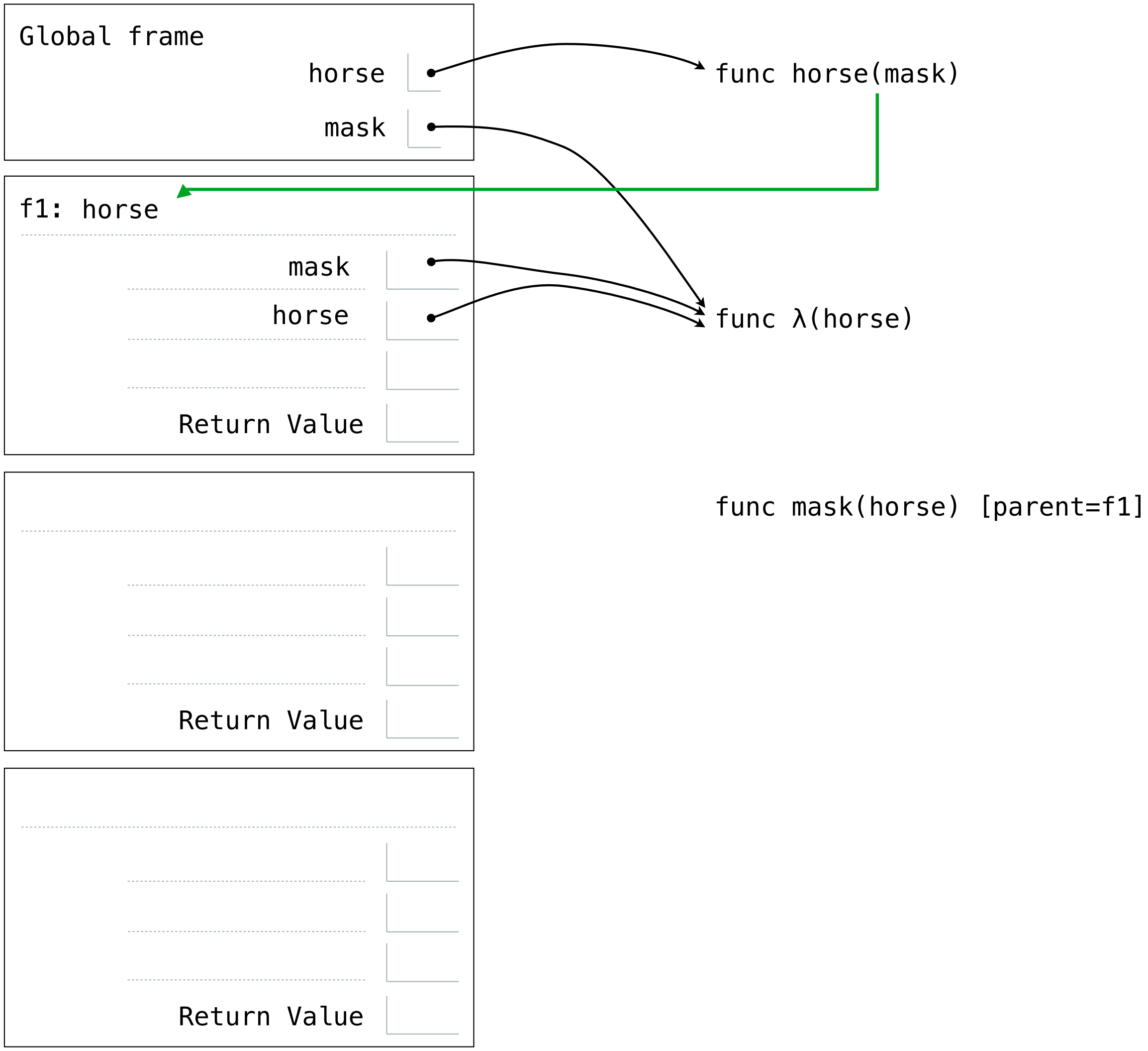
Return Value

func horse(mask)

func λ(horse)

func mask(horse) [parent=f1]

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

func horse(mask)

f1: horse

mask

horse

Return Value

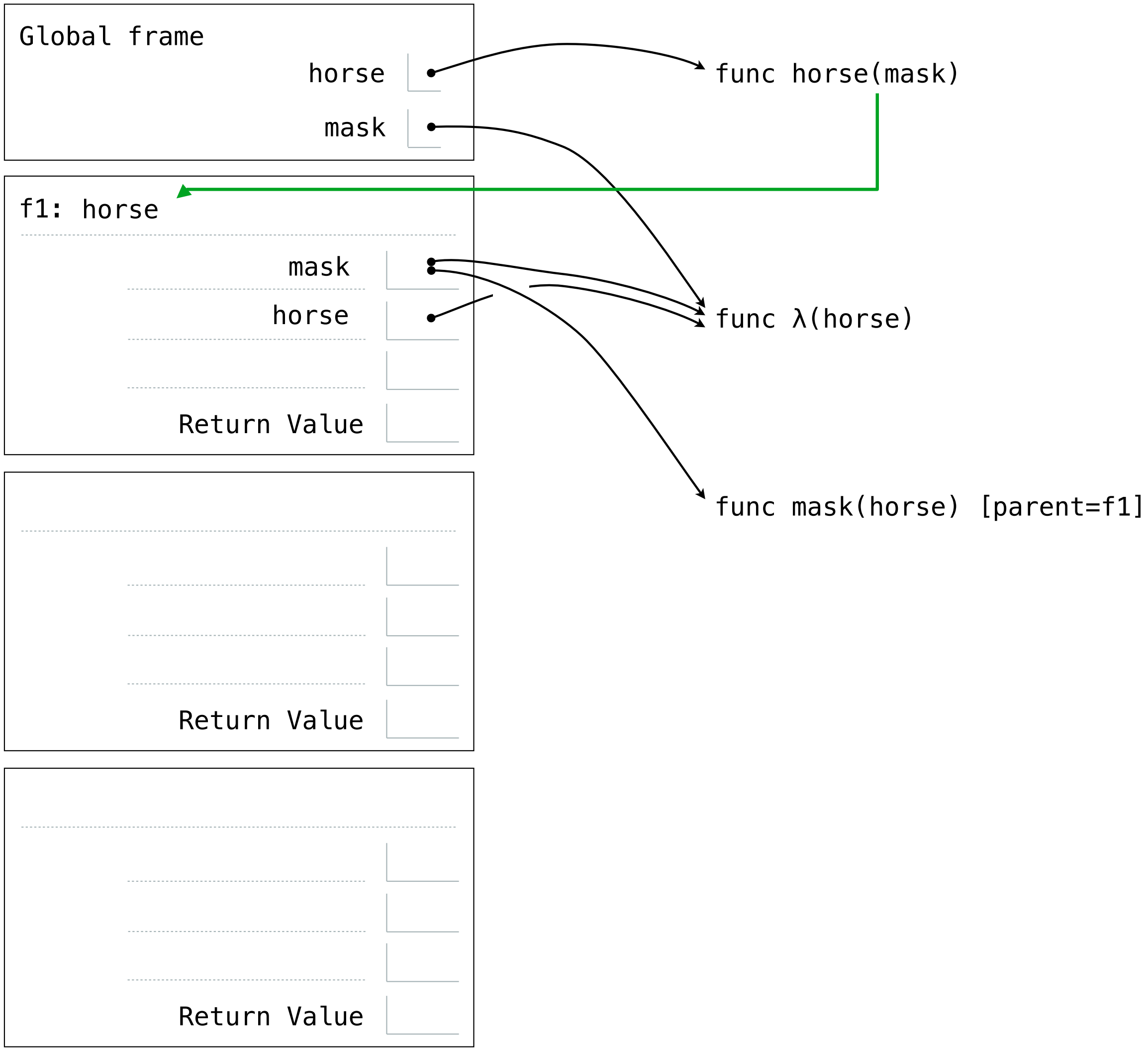
func λ(horse)

Return Value

func mask(horse) [parent=f1]

Return Value

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

func horse(mask)

f1: horse

mask

horse

Return Value

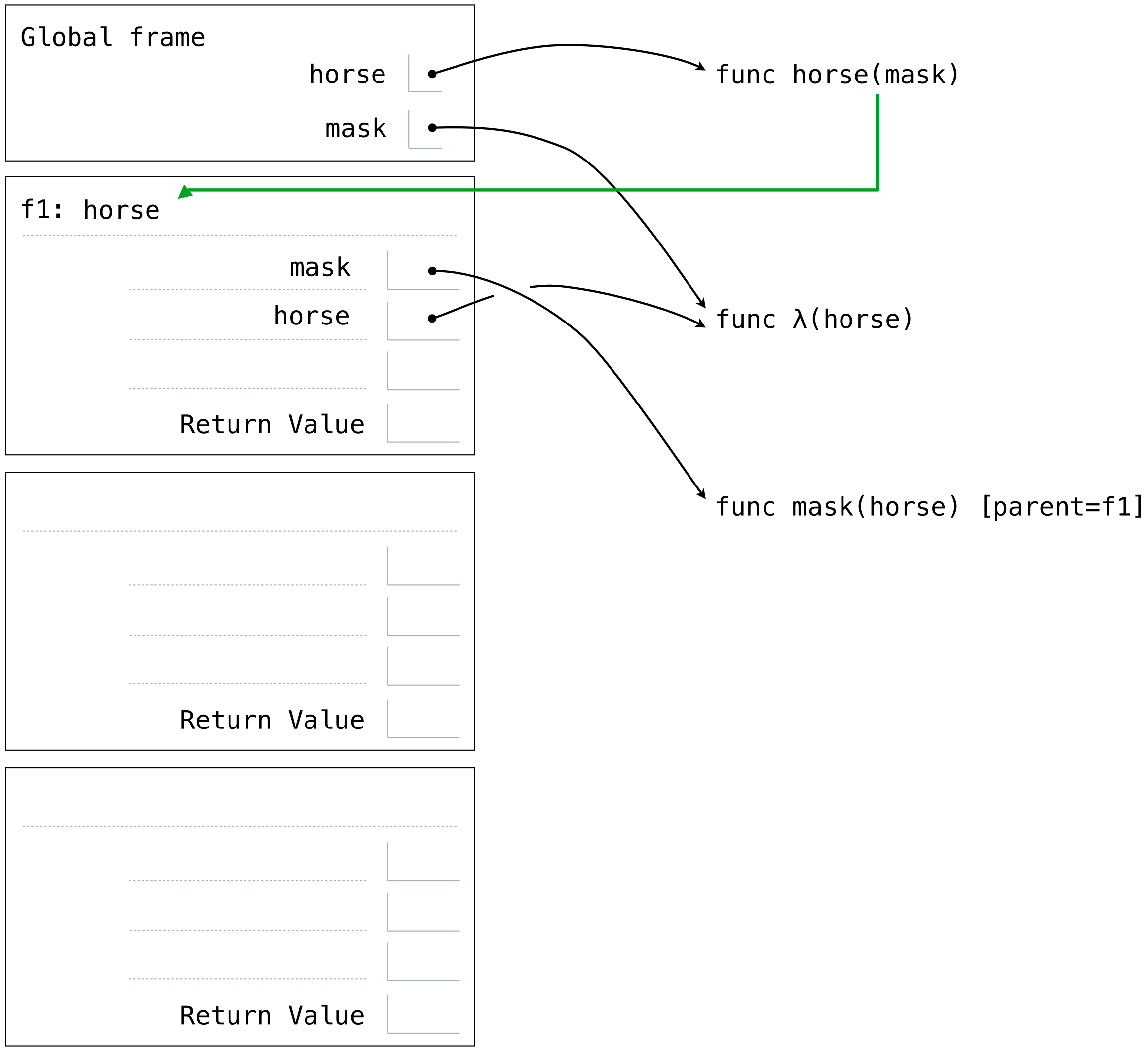
func λ(horse)

func mask(horse) [parent=f1]

Return Value

Return Value

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

func horse(mask)

f1: horse

mask

horse

Return Value

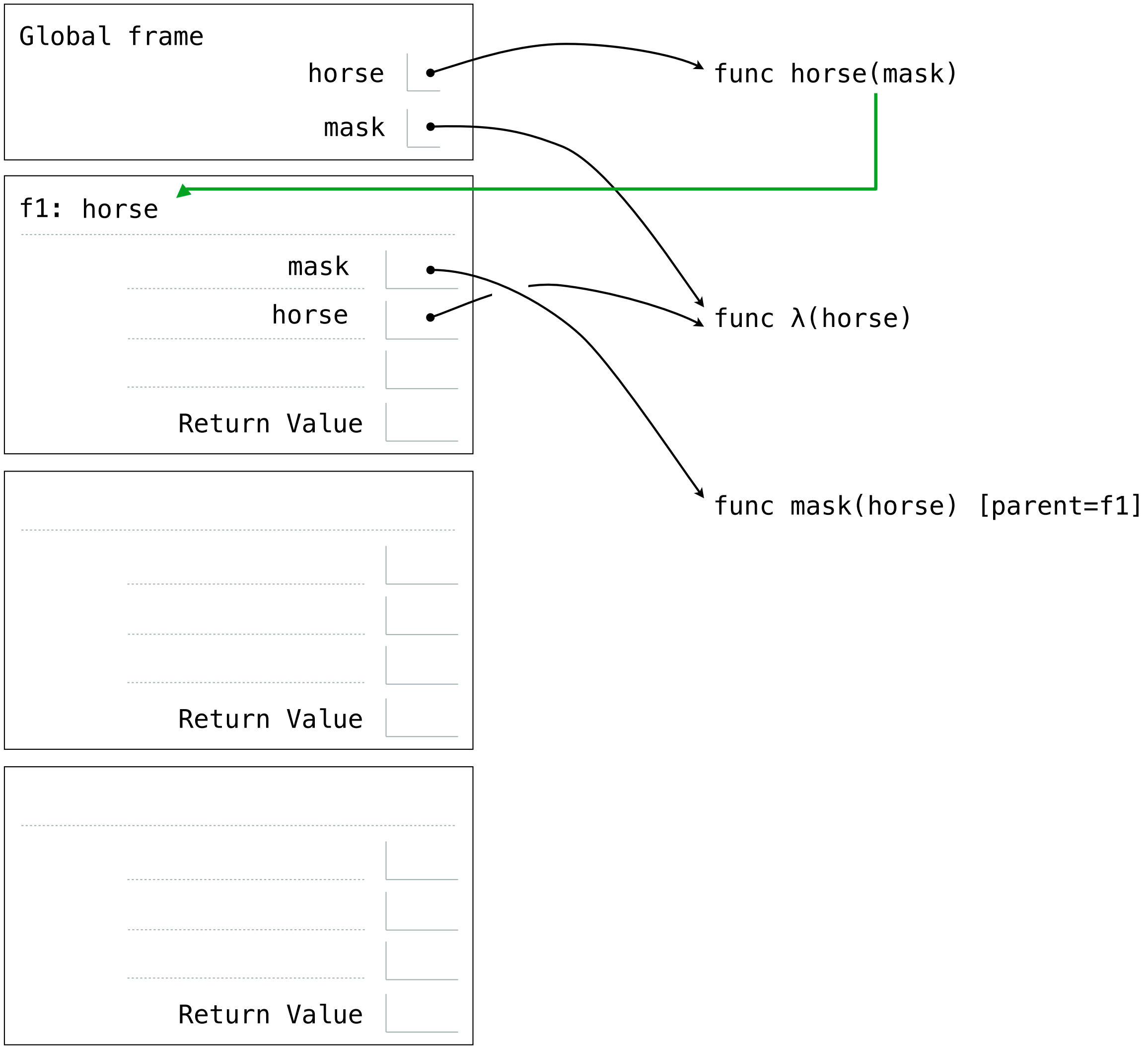
func λ(horse)

func mask(horse) [parent=f1]

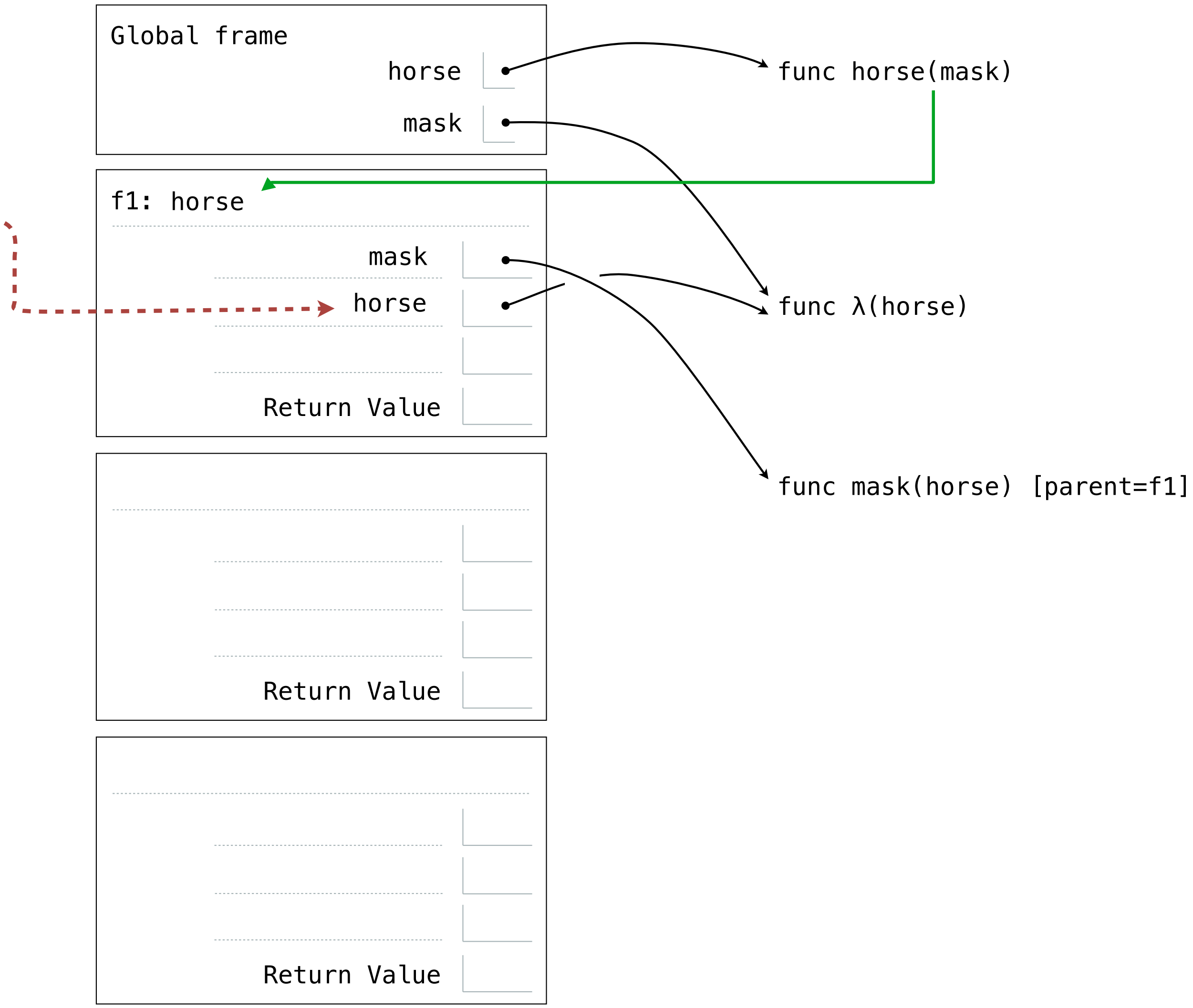
Return Value

Return Value

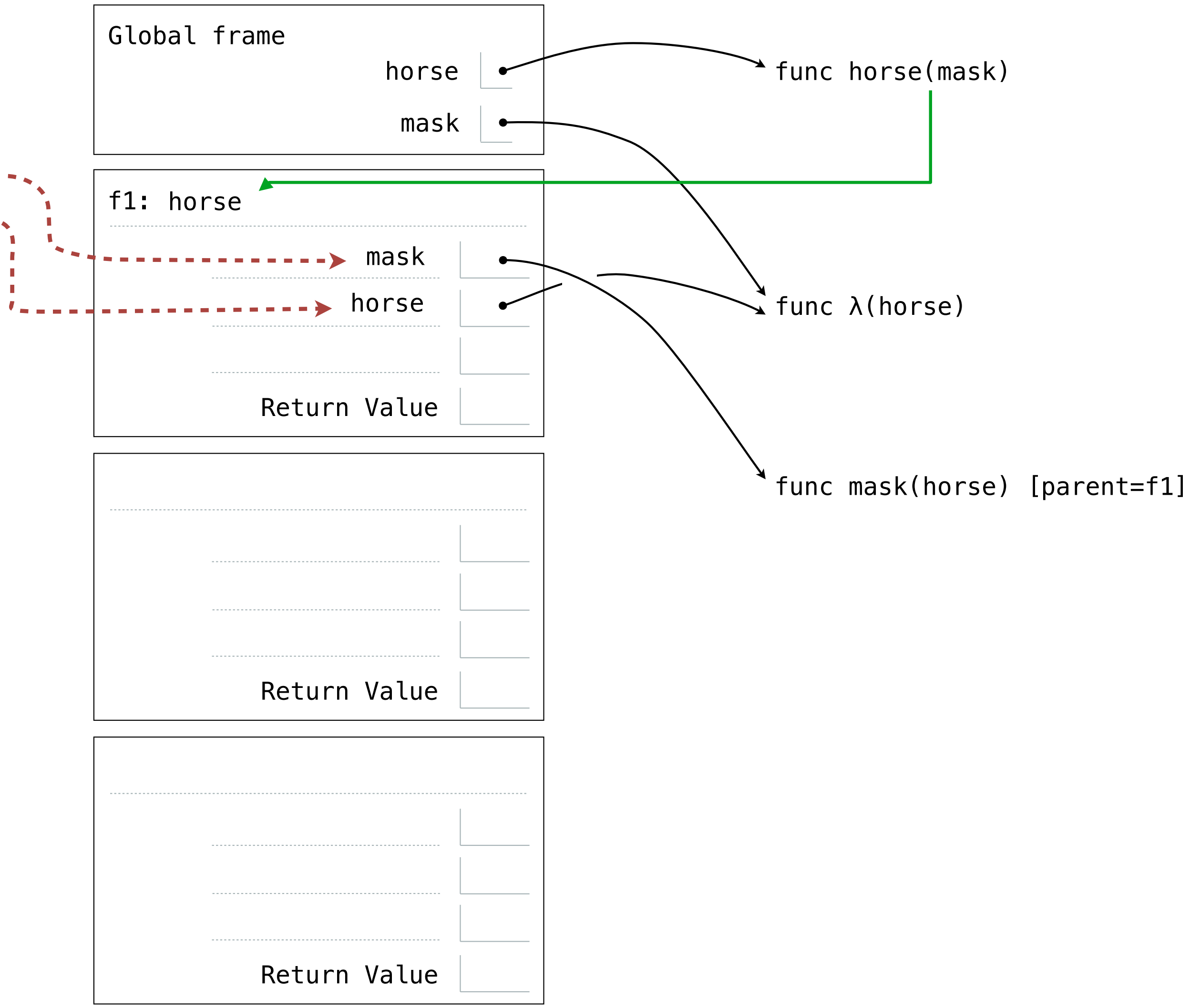
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



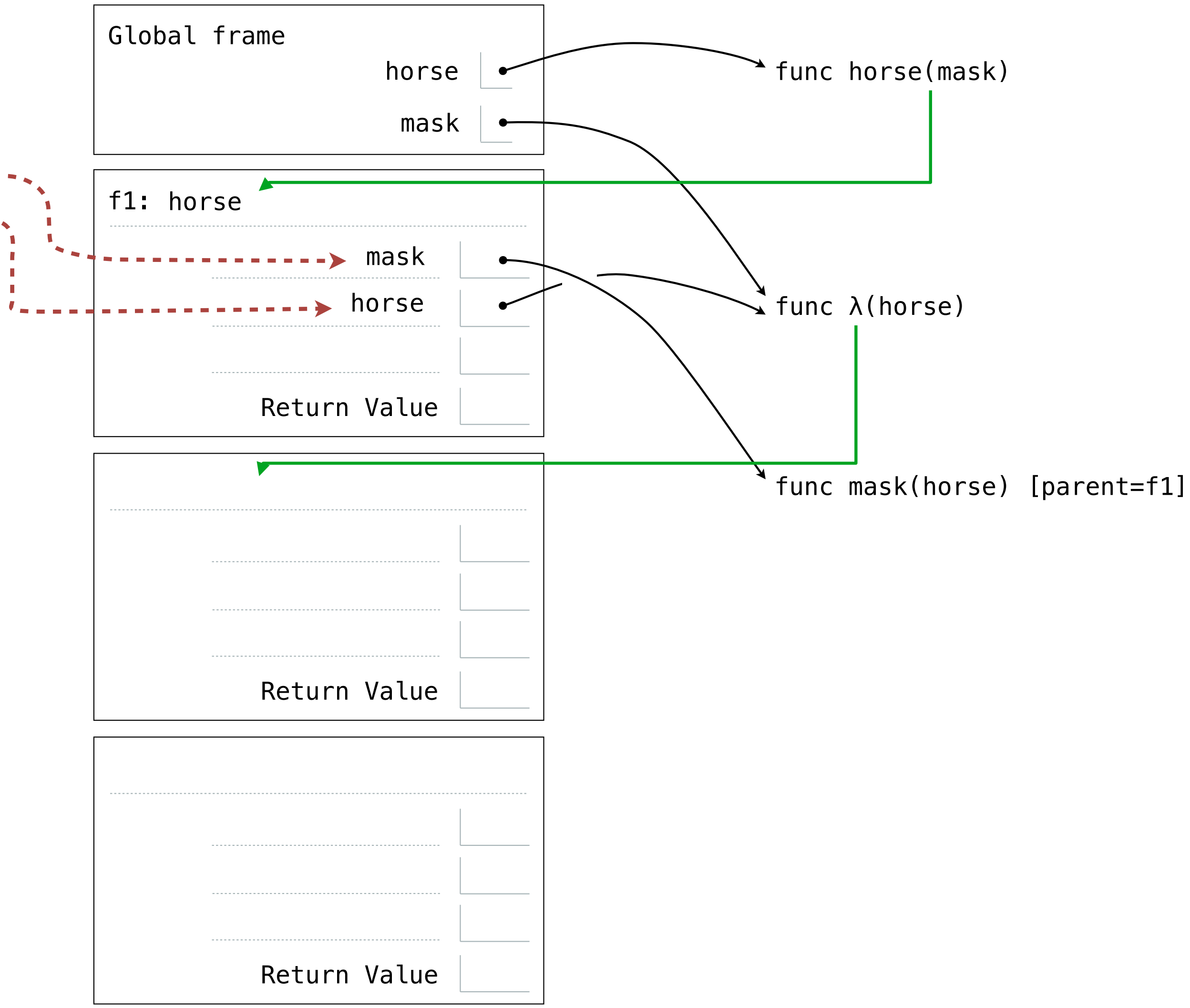
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

func horse(mask)

f1: horse

mask

horse

Return Value

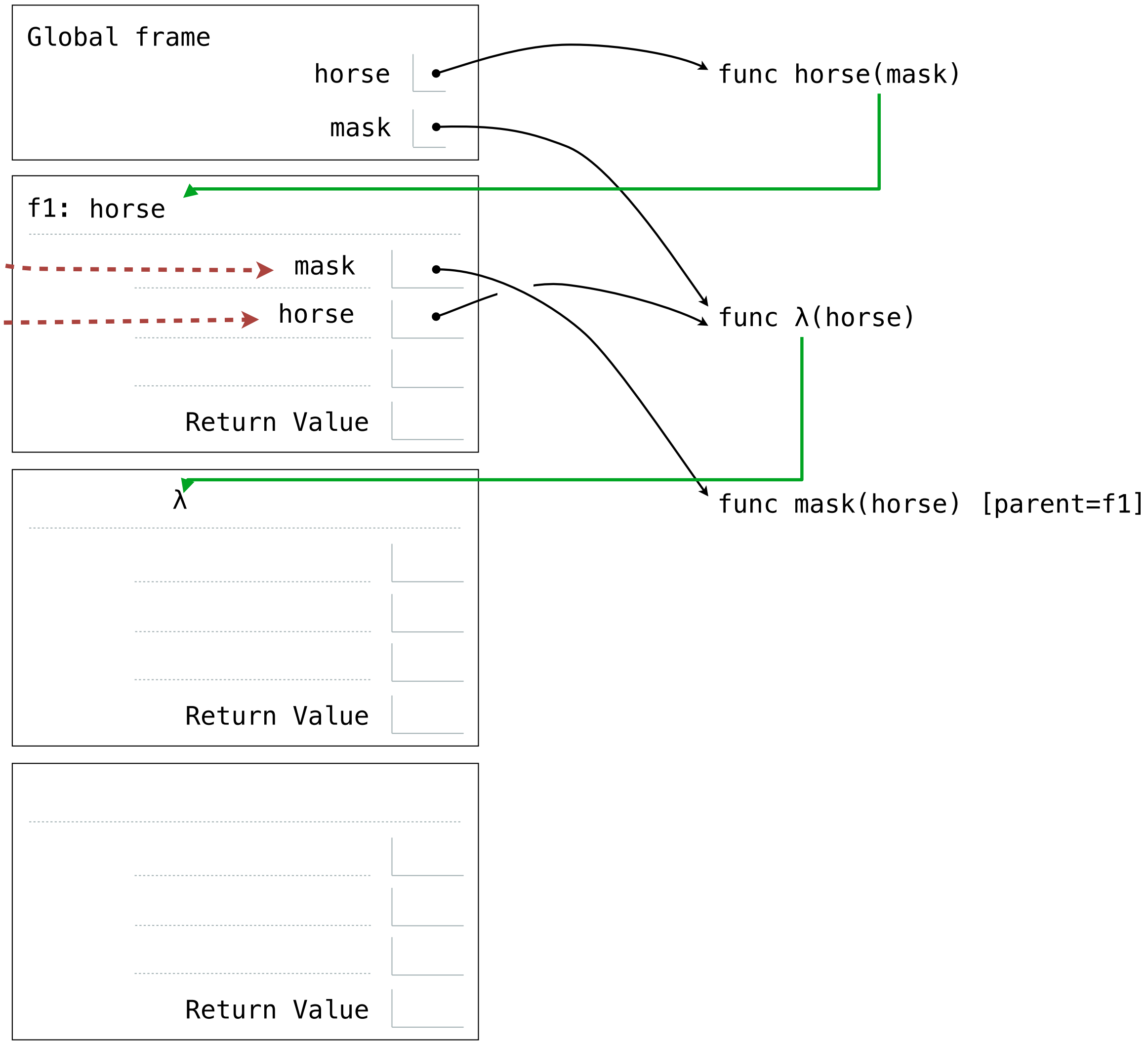
func λ(horse)

func mask(horse) [parent=f1]

Return Value

Return Value

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

func horse(mask)

f1: horse

mask

horse

Return Value

func λ(horse)

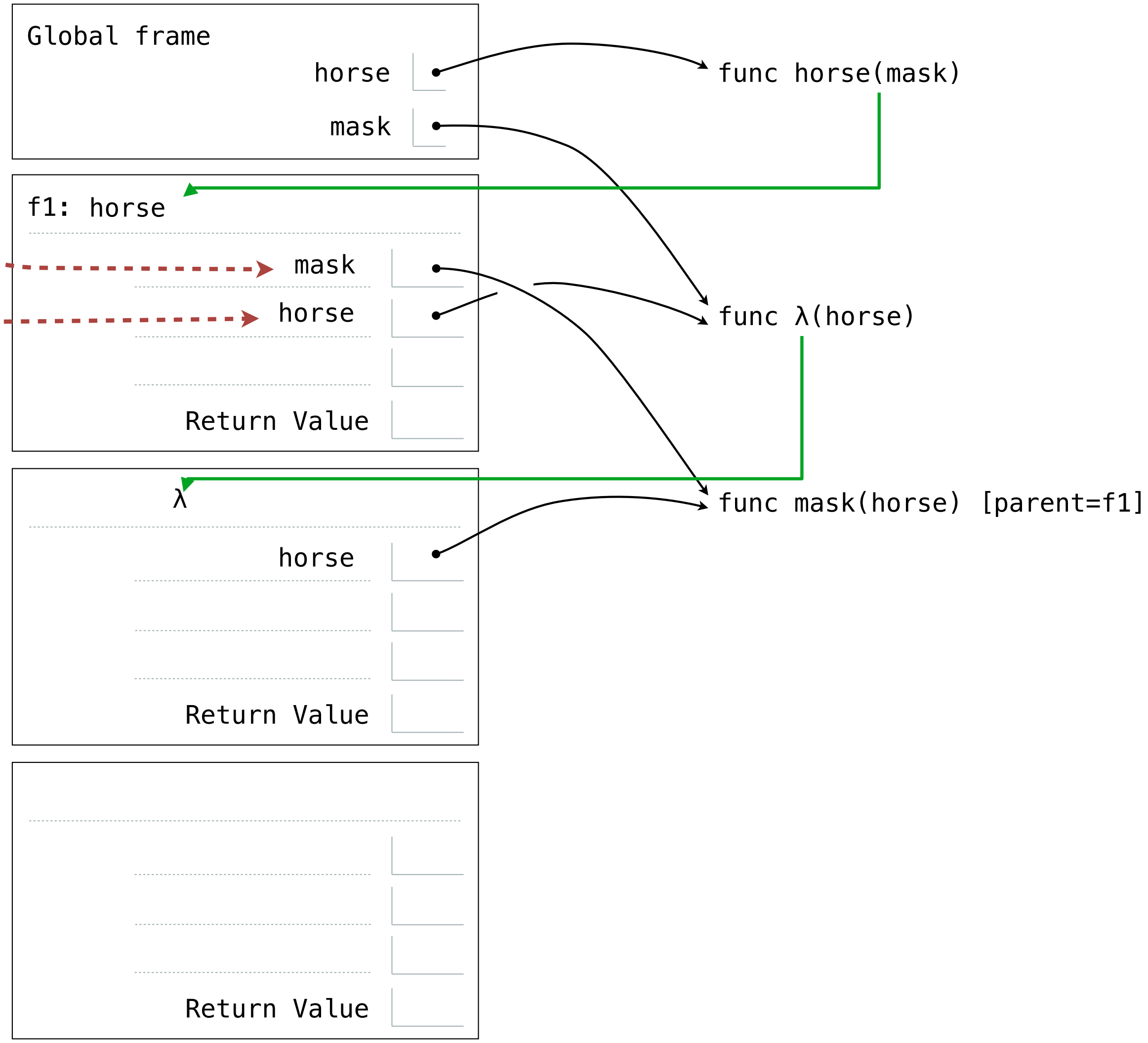
λ

Return Value

func mask(horse) [parent=f1]

Return Value

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

f1: horse

mask

horse

Return Value

λ

horse

Return Value

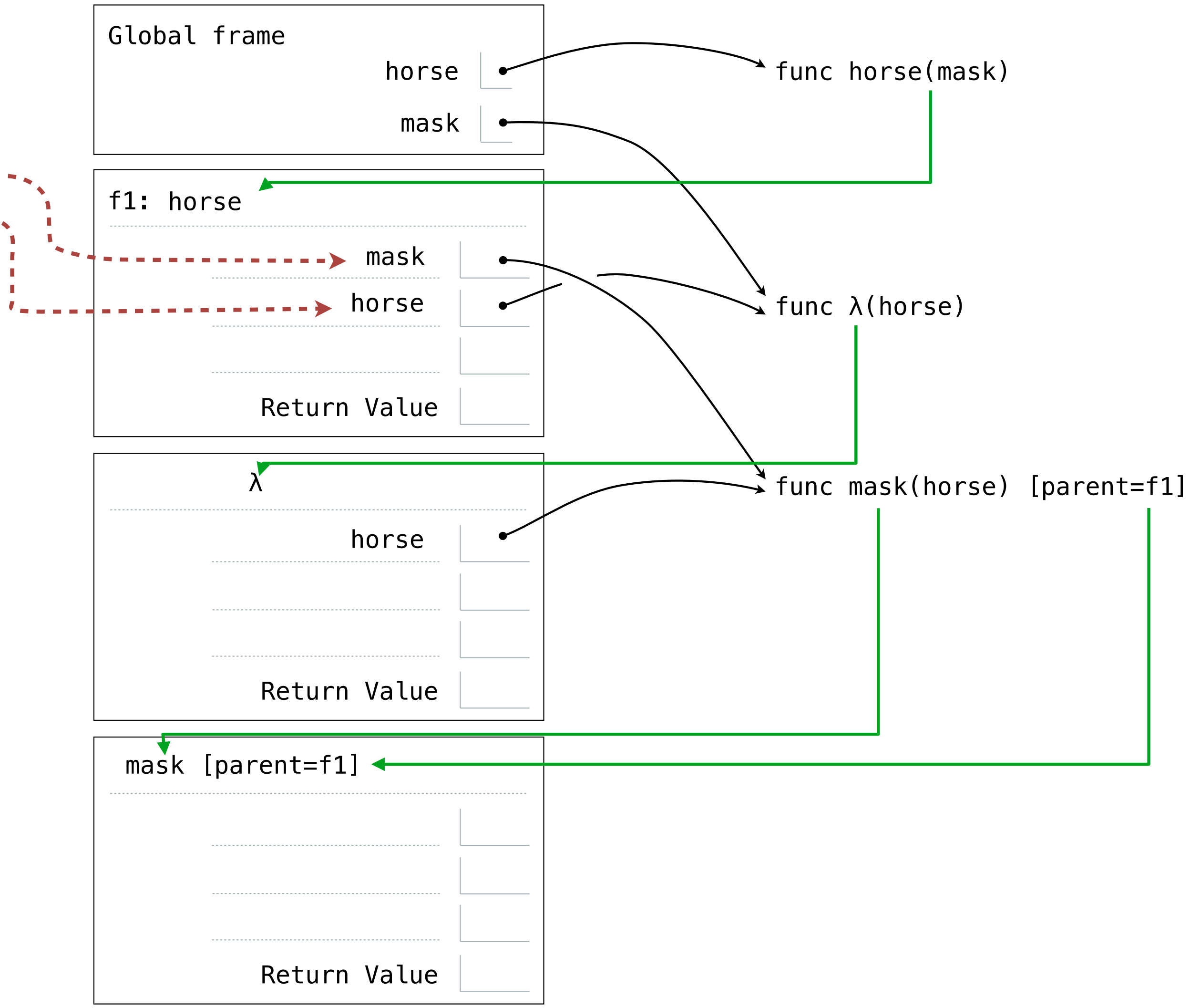
Return Value

func horse(mask)

func λ(horse)

func mask(horse) [parent=f1]

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

f1: horse

mask

horse

Return Value

λ

horse

Return Value

mask [parent=f1]

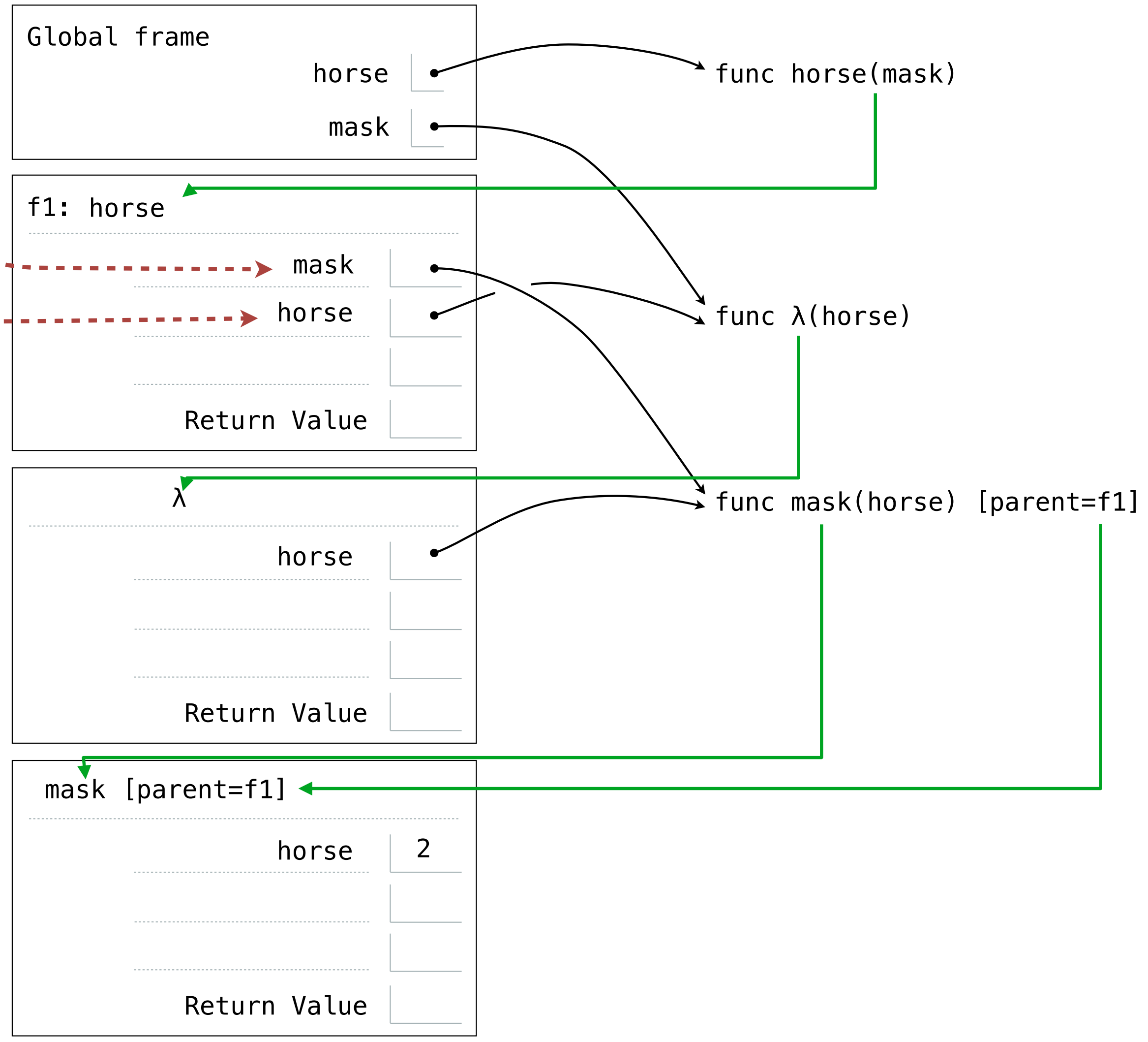
Return Value

func horse(mask)

func λ(horse)

func mask(horse) [parent=f1]

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

func horse(mask)

f1: horse

mask

horse

Return Value

func λ(horse)

λ

horse

Return Value

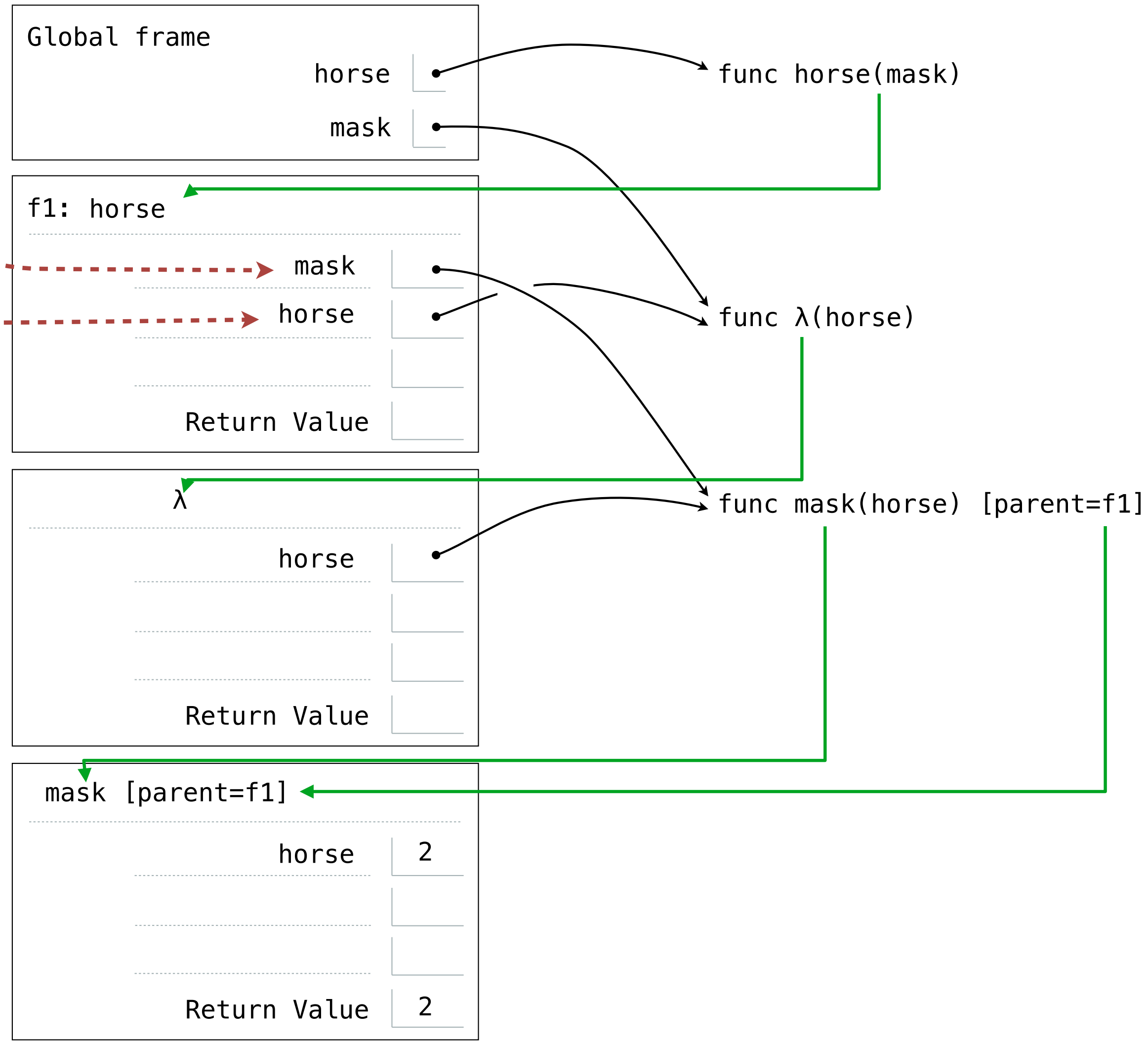
func mask(horse) [parent=f1]

mask [parent=f1]

horse 2

Return Value

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

func horse(mask)

f1: horse

mask

horse

Return Value

func λ(horse)

λ

horse

Return Value

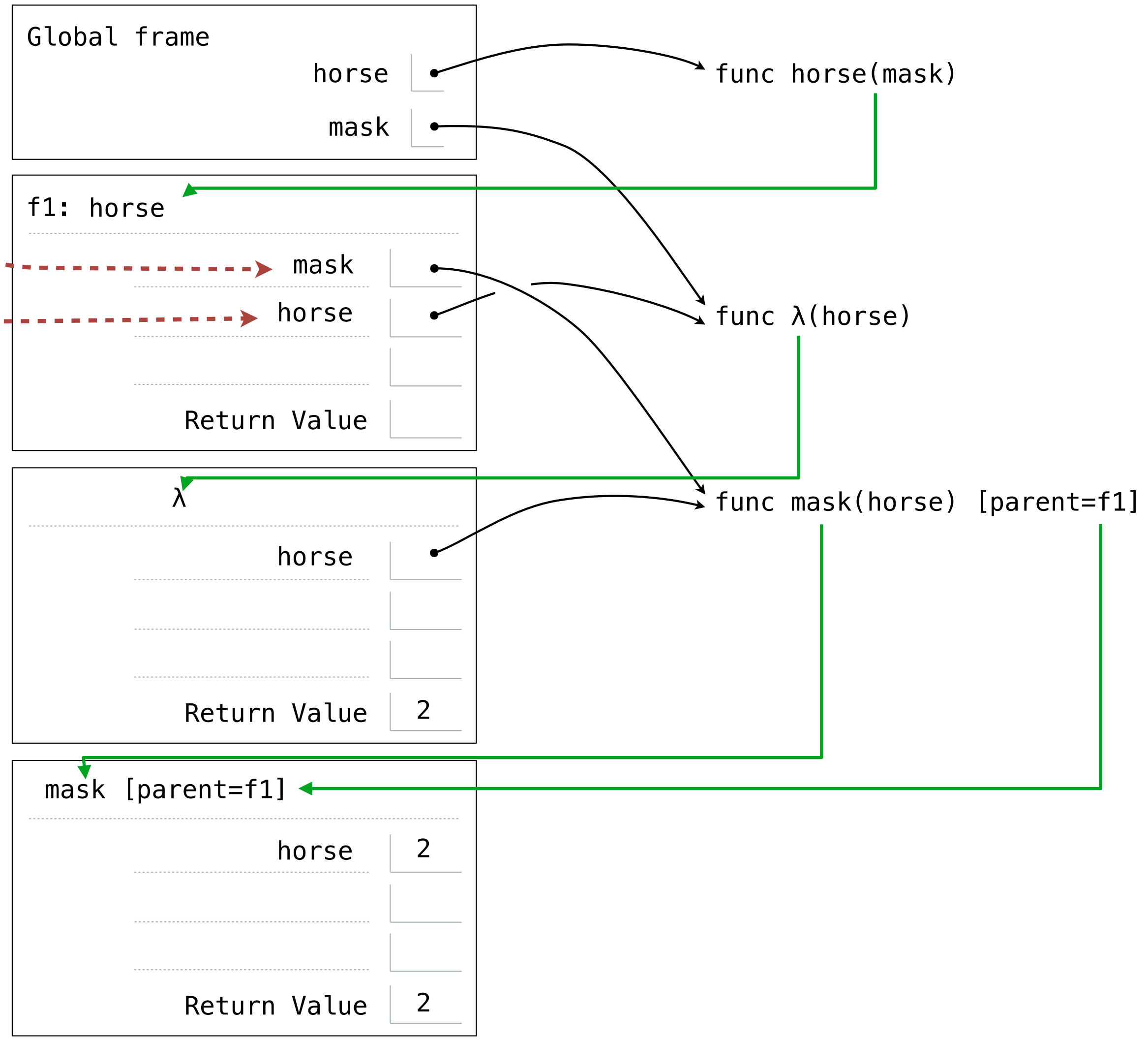
func mask(horse) [parent=f1]

mask [parent=f1]

horse 2

Return Value 2

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

func horse(mask)

f1: horse

mask

horse

Return Value

func λ(horse)

λ

horse

Return Value 2

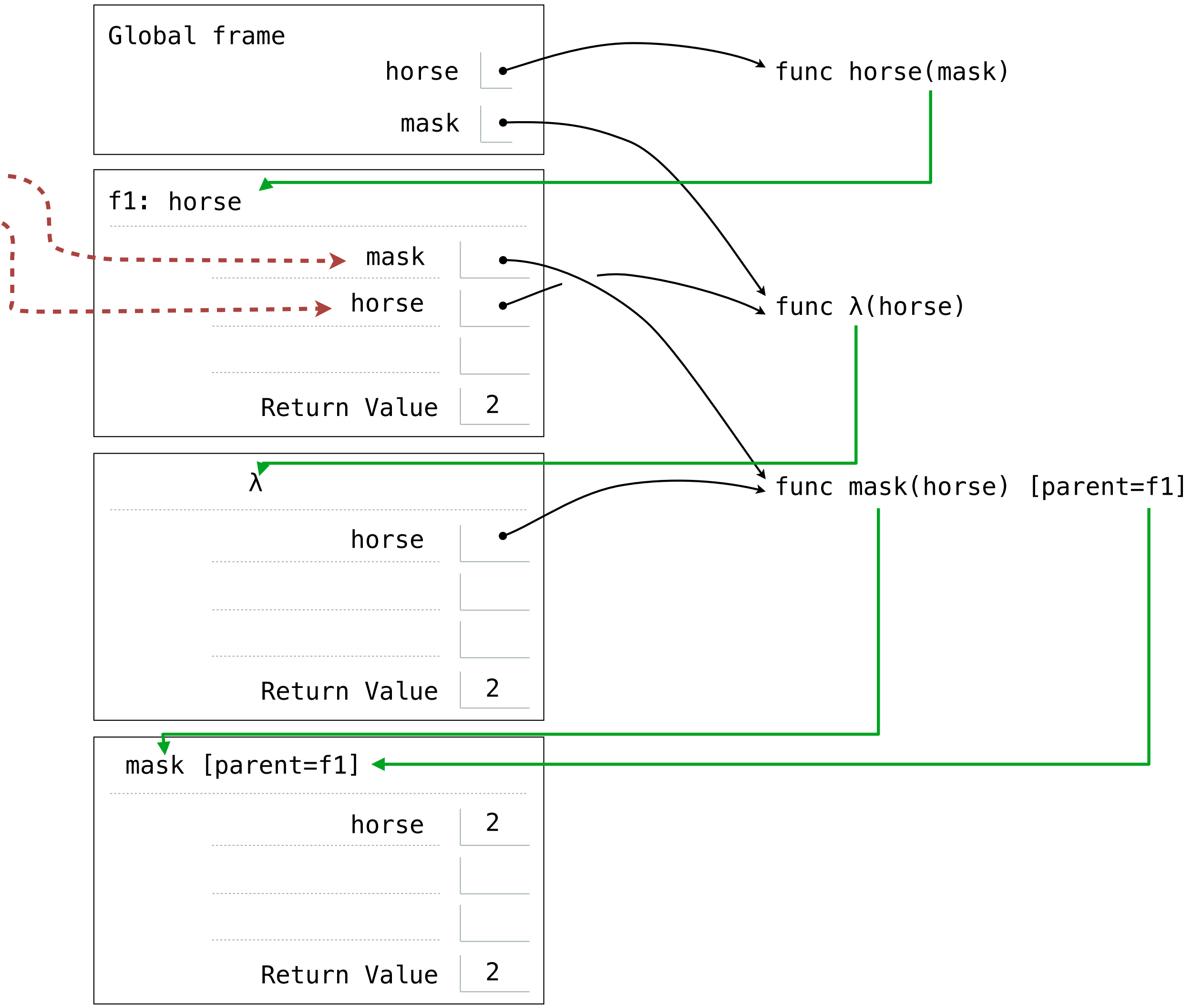
func mask(horse) [parent=f1]

mask [parent=f1]

horse 2

Return Value 2

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



Global frame

horse

mask

func horse(mask)

f1: horse

mask

horse

Return Value 2

func λ(horse)

λ

horse

Return Value 2

func mask(horse) [parent=f1]

mask [parent=f1]

horse 2

Return Value 2