при поддержке
{а}вижн

Лекция 8

Перевод и адаптация материалов Джона ДеНиро (John DeNero). Используется с разрешения автора.

Абстракция

Функциональные абстракции

Функциональные абстракции

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

Функциональные абстракции

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

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

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

- «square» принимает один аргумент?

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

- «square» принимает один аргумент? **Да**

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

- «square» принимает один аргумент? **Да**
- «square» имеет внутреннее имя «square»?

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

- «square» принимает один аргумент? **Да**
- «square» имеет внутреннее имя «square»? **Нет**

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

- «square» принимает один аргумент? **Да**
- «square» имеет внутреннее имя «square»? **Нет**
- «square» вычисляет квадрат аргумента?

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

- «square» принимает один аргумент? **Да**
- «square» имеет внутреннее имя «square»? **Нет**
- «square» вычисляет квадрат аргумента? **Да**

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

- «square» принимает один аргумент? **Да**
- «square» имеет внутреннее имя «square»? **Нет**
- «square» вычисляет квадрат аргумента? **Да**
- «square» вычисляет квадрат, используя «mul»?

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

- «square» принимает один аргумент? **Да**
- «square» имеет внутреннее имя «square»? **Нет**
- «square» вычисляет квадрат аргумента? **Да**
- «square» вычисляет квадрат, используя «mul»? **Нет**

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

- «square» принимает один аргумент? **Да**
- «square» имеет внутреннее имя «square»? **Нет**
- «square» вычисляет квадрат аргумента? **Да**
- «square» вычисляет квадрат, используя «mul»? **Нет**

```
def square(x):
    return pow(x, 2)
```

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

- «square» принимает один аргумент? **Да**
- «square» имеет внутреннее имя «square»? **Нет**
- «square» вычисляет квадрат аргумента? **Да**
- «square» вычисляет квадрат, используя «mul»? **Нет**

```
def square(x):
    return pow(x, 2)
```

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

Функциональные абстракции

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

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

Что «sum_squares» должна знать про «square»?

- «square» принимает один аргумент? **Да**
- «square» имеет внутреннее имя «square»? **Нет**
- «square» вычисляет квадрат аргумента? **Да**
- «square» вычисляет квадрат, используя «mul»? **Нет**

```
def square(x):
    return pow(x, 2)
```

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

Независимо от реализации функции «square», функция «sum_squares» продолжит работать так же хорошо.

Выбор имен

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Имена должны отражать смысл или назначение связанных с ними значений.

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Имена должны отражать смысл или назначение связанных с ними значений.

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Имена должны отражать смысл или назначение связанных с ними значений.

Тип значения, связанного с именем, лучше описывать в докстринге функции.

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Имена должны отражать смысл или назначение связанных с ними значений.

Тип значения, связанного с именем, лучше описывать в докстринге функции.

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Имена должны отражать смысл или назначение связанных с ними значений.

Тип значения, связанного с именем, лучше описывать в докстринге функции.

Имена функций обычно отражают их действие (print), их поведение (triple), или возвращаемое значение (abs).

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Плохо:	**Хорошо:**

Имена должны отражать смысл или назначение связанных с ними значений.

Тип значения, связанного с именем, лучше описывать в докстринге функции.

Имена функций обычно отражают их действие (print), их поведение (triple), или возвращаемое значение (abs).

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Плохо:	**Хорошо:**
`true_false`	`rolled_a_one`

Имена должны отражать смысл или назначение связанных с ними значений.

Тип значения, связанного с именем, лучше описывать в докстринге функции.

Имена функций обычно отражают их действие (`print`), их поведение (`triple`), или возвращаемое значение (`abs`).

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Плохо:	**Хорошо:**
`true_false`	`rolled_a_one`
`d`	`dice`

Имена должны отражать смысл или назначение связанных с ними значений.

Тип значения, связанного с именем, лучше описывать в докстринге функции.

Имена функций обычно отражают их действие (`print`), их поведение (`triple`), или возвращаемое значение (`abs`).

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Плохо:	**Хорошо:**
`true_false`	`rolled_a_one`
`d`	`dice`
`play_helper`	`take_turn`

Имена должны отражать смысл или назначение связанных с ними значений.

Тип значения, связанного с именем, лучше описывать в докстринге функции.

Имена функций обычно отражают их действие (`print`), их поведение (`triple`), или возвращаемое значение (`abs`).

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Плохо:	**Хорошо:**
`true_false`	`rolled_a_one`
`d`	`dice`
`play_helper`	`take_turn`
`my_int`	`num_rolls`

Имена должны отражать смысл или назначение связанных с ними значений.

Тип значения, связанного с именем, лучше описывать в докстринге функции.

Имена функций обычно отражают их действие (`print`), их поведение (`triple`), или возвращаемое значение (`abs`).

Выбор имен

Обычно имена не влияют на корректность программы,

но

от них сильно зависит читаемость!

Плохо:	**Хорошо:**
`true_false`	`rolled_a_one`
`d`	`dice`
`play_helper`	`take_turn`
`my_int`	`num_rolls`
`l, I, O`	`k, i, m`

Имена должны отражать смысл или назначение связанных с ними значений.

Тип значения, связанного с именем, лучше описывать в докстринге функции.

Имена функций обычно отражают их действие (`print`), их поведение (`triple`), или возвращаемое значение (`abs`).

Кто заслуживает собственное имя?

Причины добавления нового имени

Кто заслуживает собственное имя?

Причины добавления нового имени

Повторение составных выражений:

Кто заслуживает собственное имя?

Причины добавления нового имени

Повторение составных выражений:

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

Кто заслуживает собственное имя?

Причины добавления нового имени

Повторение составных выражений:

```
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
```

Кто заслуживает собственное имя?

Причины добавления нового имени

Повторение составных выражений:

```
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
```

Смысловые части сложных выражений:

Кто заслуживает собственное имя?

Причины добавления нового имени

Повторение составных выражений:

```
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
```

Смысловые части сложных выражений:

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

Кто заслуживает собственное имя?

Причины добавления нового имени

Повторение составных выражений:

```
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
```

Смысловые части сложных выражений:

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

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

Кто заслуживает собственное имя?

Причины добавления нового имени

Повторение составных выражений:

```
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
```

Смысловые части сложных выражений:

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

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

Советы по именованию:

Кто заслуживает собственное имя?

Причины добавления нового имени

Повторение составных выражений:

```
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
```

Смысловые части сложных выражений:

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

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

Советы по именованию:

- Имена могут быть длинными, если они помогают документировать код:

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

лучше, чем

```
# Вычисляем средний возраст студентов
aa = avg(a, st)
```

Кто заслуживает собственное имя?

Причины добавления нового имени

Повторение составных выражений:

```
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
```

Смысловые части сложных выражений:

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

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

Советы по именованию:

- Имена могут быть длинными, если они помогают документировать код:

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

лучше, чем

```
# Вычисляем средний возраст студентов
aa = avg(a, st)
```

Кто заслуживает собственное имя?

Причины добавления нового имени

Повторение составных выражений:

```
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
```

Смысловые части сложных выражений:

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

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

Советы по именованию:

- Имена могут быть длинными, если они помогают документировать код:

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

лучше, чем

```
# Вычисляем средний возраст студентов
aa = avg(a, st)
```

- Имена могут быть короткими, если они связаны с очевидными величинами: счётчики, произвольные функции, аргументы математических действий и так далее.

```
n, k, i – обычно целые
x, y, z – обычно действительные
f, g, h – обычно функции
```

Кто заслуживает собственное имя?

Причины добавления нового имени

Повторение составных выражений:

```
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
```

ПРАКТИЧЕСКИЕ РЕКОМЕНДАЦИИ

Смысловые части сложных выражений:

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

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

Советы по именованию:

- Имена могут быть длинными, если они помогают документировать код:

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

лучше, чем

```
# Вычисляем средний возраст студентов
aa = avg(a, st)
```

- Имена могут быть короткими, если они связаны с очевидными величинами: счётчики, произвольные функции, аргументы математических действий и так далее.

```
n, k, i – обычно целые
x, y, z – обычно действительные
f, g, h – обычно функции
```

Тестирование

Разработка основанная на тестировании (TDD)

Разработка основанная на тестировании (TDD)

Пиши тест для функции до написания самой функции.

Разработка основанная на тестировании (TDD)

Пиши тест для функции до написания самой функции.

Тест прояснит поведение, область определения и область значений функции.

Разработка основанная на тестировании (TDD)

Пиши тест для функции до написания самой функции.

Тест прояснит поведение, область определения и область значений функции.

Тесты помогают отыскать «хитрые» граничные случаи.

Разработка основанная на тестировании (TDD)

Пиши тест для функции до написания самой функции.

Тест прояснит поведение, область определения и область значений функции.

Тесты помогают отыскать «хитрые» граничные случаи.

Разрабатывай небольшими шагами и тестируй результат каждого шага.

Разработка основанная на тестировании (TDD)

Пиши тест для функции до написания самой функции.

Тест прояснит поведение, область определения и область значений функции.

Тесты помогают отыскать «хитрые» граничные случаи.

Разрабатывай небольшими шагами и тестируй результат каждого шага.

Не полагайся на код, который не протестирован.

Разработка основанная на тестировании (TDD)

Пиши тест для функции до написания самой функции.

Тест прояснит поведение, область определения и область значений функции.

Тесты помогают отыскать «хитрые» граничные случаи.

Разрабатывай небольшими шагами и тестируй результат каждого шага.

Не полагайся на код, который не протестирован.

Всегда запускай все тесты при внесении изменений.

Разработка основанная на тестировании (TDD)

Пиши тест для функции до написания самой функции.

Тест прояснит поведение, область определения и область значений функции.

Тесты помогают отыскать «хитрые» граничные случаи.

Разрабатывай небольшими шагами и тестируй результат каждого шага.

Не полагайся на код, который не протестирован.

Всегда запускай все тесты при внесении изменений.

Дополнительная идея: запускай код в интерактивном режиме.

Разработка основанная на тестировании (TDD)

Пиши тест для функции до написания самой функции.

Тест прояснит поведение, область определения и область значений функции.

Тесты помогают отыскать «хитрые» граничные случаи.

Разрабатывай небольшими шагами и тестируй результат каждого шага.

Не полагайся на код, который не протестирован.

Всегда запускай все тесты при внесении изменений.

Дополнительная идея: запускай код в интерактивном режиме.

Не бойся экспериментировать с написаной функцией.

Разработка основанная на тестировании (TDD)

Пиши тест для функции до написания самой функции.

Тест прояснит поведение, область определения и область значений функции.

Тесты помогают отыскать «хитрые» граничные случаи.

Разрабатывай небольшими шагами и тестируй результат каждого шага.

Не полагайся на код, который не протестирован.

Всегда запускай все тесты при внесении изменений.

Дополнительная идея: запускай код в интерактивном режиме.

Не бойся экспериментировать с написаной функцией.

Интерактивные сессии могут превращаться в доктесты. Просто скопируй и вставь.

Разработка основанная на тестировании (TDD)

Пиши тест для функции до написания самой функции.

Тест прояснит поведение, область определения и область значений функции.

Тесты помогают отыскать «хитрые» граничные случаи.

Разрабатывай небольшими шагами и тестируй результат каждого шага.

Не полагайся на код, который не протестирован.

Всегда запускай все тесты при внесении изменений.

(Пример)

Дополнительная идея: запускай код в интерактивном режиме.

Не бойся экспериментировать с написаной функцией.

Интерактивные сессии могут превращаться в доктесты. Просто скопируй и вставь.

Декораторы

Декораторы функций

(Пример)

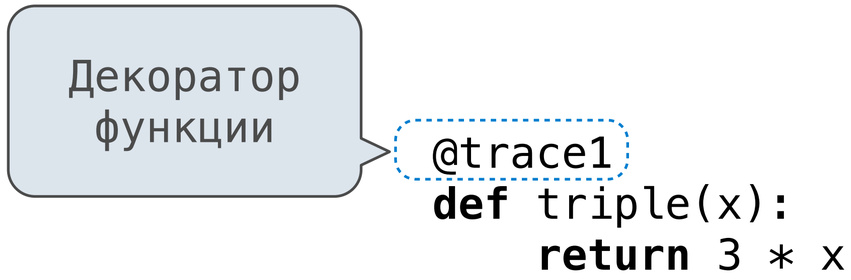
Декораторы функций

(Пример)

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

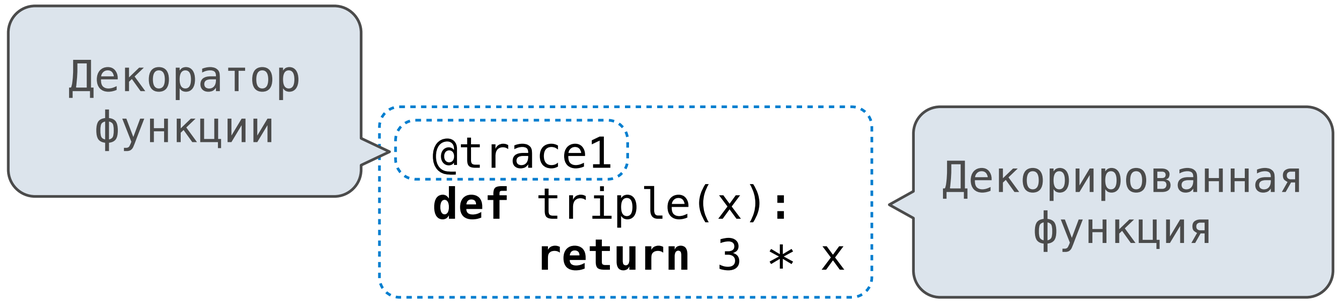
Декораторы функций

(Пример)



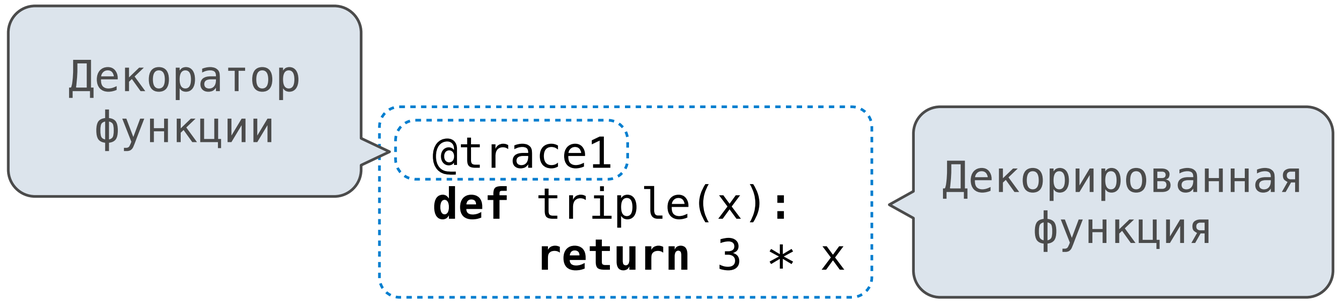
Декораторы функций

(Пример)



Декораторы функций

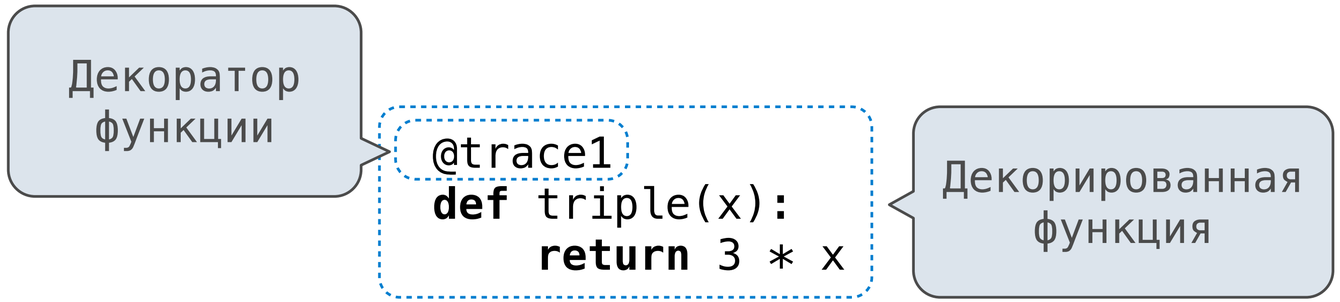
(Пример)



является идентичной с

Декораторы функций

(Пример)



является идентичной с

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

Декораторы функций

(Пример)

Декоратор функции

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

Декорированная функция

является идентичной с

Почему не делать так?

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

Примеры задач

Представь себя «пайтоном» (ПСП)

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
5	5	

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
5	5	5

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
5	5	5
print(5)		

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
5	5	5
print(5)	None	

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
5	5	5
print(5)	None	5

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`		

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))` `None`	`None`	`5` `None`

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
5	5	5
print(5)	None	5
print(print(5))	None	5 None

None

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

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

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

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`
`delay(delay)()(6)()`		

None

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`
`delay(delay)()(6)()`		

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
5	5	5
print(5)	None	5
print(print(5))	None	5 None
delay(delay)()(6)()		

None

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
5	5	5
print(5)	None	5
print(print(5))	None	5 None
delay(delay)()(6)()		

None

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
5	5	5
print(5)	None	5
print(print(5))	None	5 None
delay(delay)()(6)()		

None

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
5	5	5
print(5)	None	5
print(print(5))	None	5 None
delay(delay)()(6)()		

None

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
5	5	5
print(5)	None	5
print(print(5))	None	5 None

None

```
delay(delay)()(6)()
```

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))` `None`	`None`	`5` `None`
`delay(delay)()(6)()`		`задержка`

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`
`delay(delay)()(6)()`		`задержка` `задержка`

None

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`

`delay(delay)()(6)()`		`задержка`
		`задержка`
		`6`

None

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`

`delay(delay)()(6)()`	`6`	`задержка`
		`задержка`
		`6`

None

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`

`delay(delay)()(6)()`	`6`	`задержка`
		`задержка`
		`6`

`print(delay(print)()(4))`		

None

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`

`delay(delay)()(6)()`	`6`	`задержка`
		`задержка`
		`6`

`print(delay(print)()(4))`		`задержка`

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`

`delay(delay)()(6)()`	`6`	`задержка`
		`задержка`
		`6`

`print(delay(print)()(4))`		`задержка` `4`

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`

`delay(delay)()(6)()`	`6`	`задержка`
		`задержка`
		`6`

`print(delay(print)()(4))`		`задержка`
		`4`
		`None`

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, принимающая любой аргумент и возвращающая функцию, которая возвращает этот аргумент

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

Имена во вложеной инструкции «def» могут быть определены во внешней

Выражение	Значение	Интерактивный вывод
`5`	`5`	`5`
`print(5)`	`None`	`5`
`print(print(5))`	`None`	`5` `None`

`delay(delay)()(6)()`	`6`	`задержка`
		`задержка`
		`6`

`print(delay(print)()(4))`	`None`	`задержка`
		`4`
		`None`

None

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`		

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Выражение	Значение	Интерактивный вывод
add(pirate(3)(square)(4), 1)		

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`		

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`		

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`		йо-хо-хо

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`		йо-хо-хо

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`		`йо-хо-хо`
`func square(x)`		

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`		йо-хо-хо
func square(x)		

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`		
func square(x)		
16		

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`		йо-хо-хо
func square(x)		17
16		

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`	17	йо-хо-хо 17

func square(x)

16

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`	17	йо-хо-хо 17
func square(x)		
16		
`pirate(pirate(pirate))(5)(7)`		

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`	17	йо-хо-хо 17
func square(x)		
16		
`pirate(pirate(pirate))(5)(7)`		

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`	17	йо-хо-хо 17
func square(x)		
16		
`pirate(pirate(pirate))(5)(7)`		
Тождественная функция		

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)`	17	йо-хо-хо 17
func square(x)		
16		
`pirate(pirate(pirate))(5)(7)`		йо-хо-хо
Тождественная функция		

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)` *func square(x)* *16*	17	йо-хо-хо 17
`pirate(pirate(pirate))(5)(7)` *Тождественная функция*		йо-хо-хо йо-хо-хо

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)` *func square(x)* *16*	17	йо-хо-хо 17
`pirate(pirate(pirate))(5)(7)` *Тождественная функция*		йо-хо-хо йо-хо-хо

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)` *func square(x)* *16*	17	йо-хо-хо 17
`pirate(pirate(pirate))(5)(7)` *Тождественная функция* *5*		йо-хо-хо йо-хо-хо

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)` *func square(x)* *16*	17	йо-хо-хо 17
`pirate(pirate(pirate))(5)(7)` *Тождественная функция* *5*		йо-хо-хо йо-хо-хо Error

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

Представь себя «пайтоном» (ПСП)

Функция «print» возвращает «None». При вызове она также выводит на экран аргументы (разделенные пробелами).

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

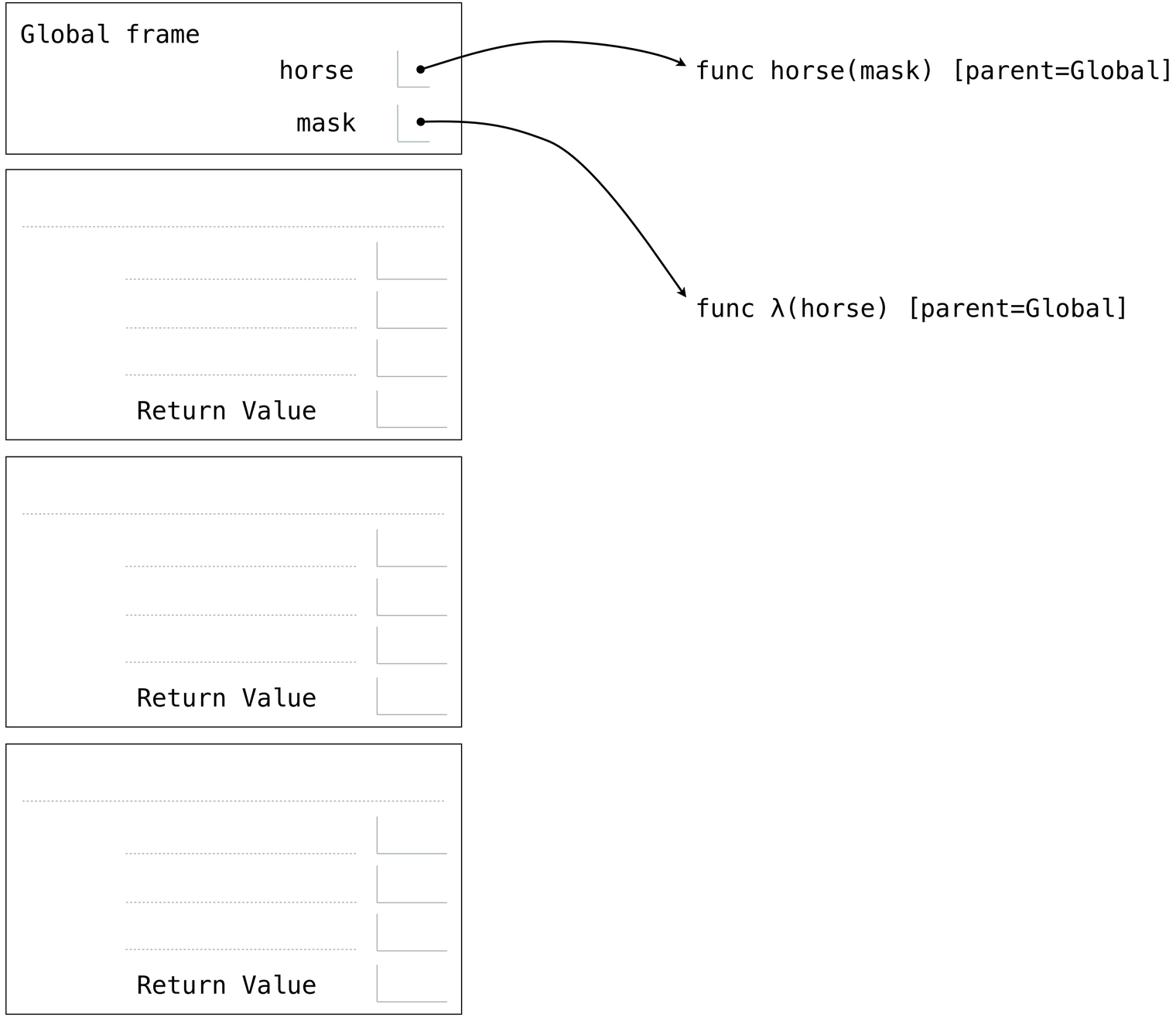
Функция, которая всегда возвращает тождественную функцию

```
def pirate(arggg):
    print('йо-хо-хо')
    def plunder(arggg):
        return arggg
    return plunder
```

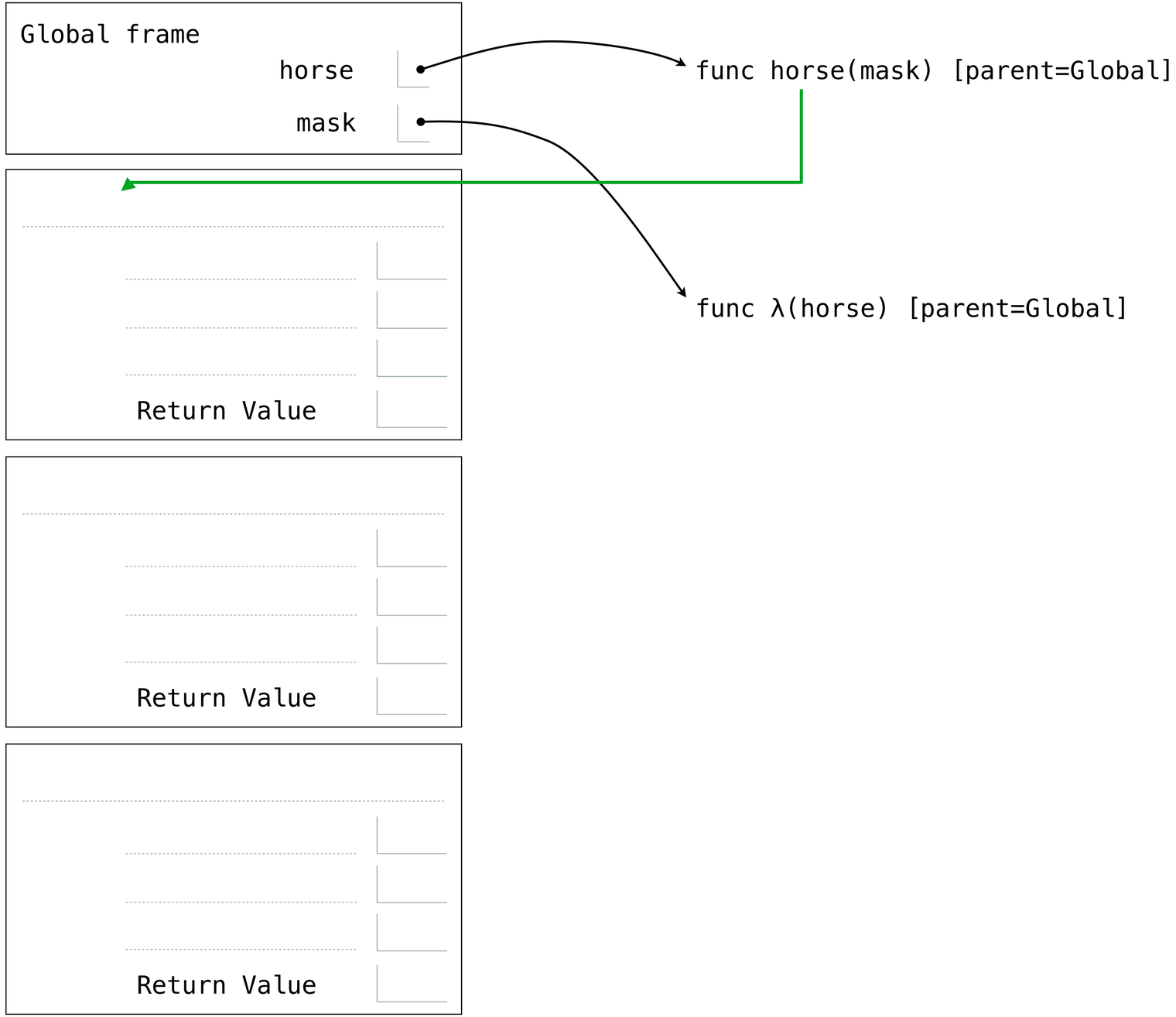
Выражение	Значение	Интерактивный вывод
`add(pirate(3)(square)(4), 1)` *func square(x)* *16*	17	йо-хо-хо 17
`pirate(pirate(pirate))(5)(7)` *Тождественная функция* *5*	Error	йо-хо-хо йо-хо-хо Error

Значение связанное с именем берётся из наиболее свежего фрейма текущего окружения, в котором оно определено.

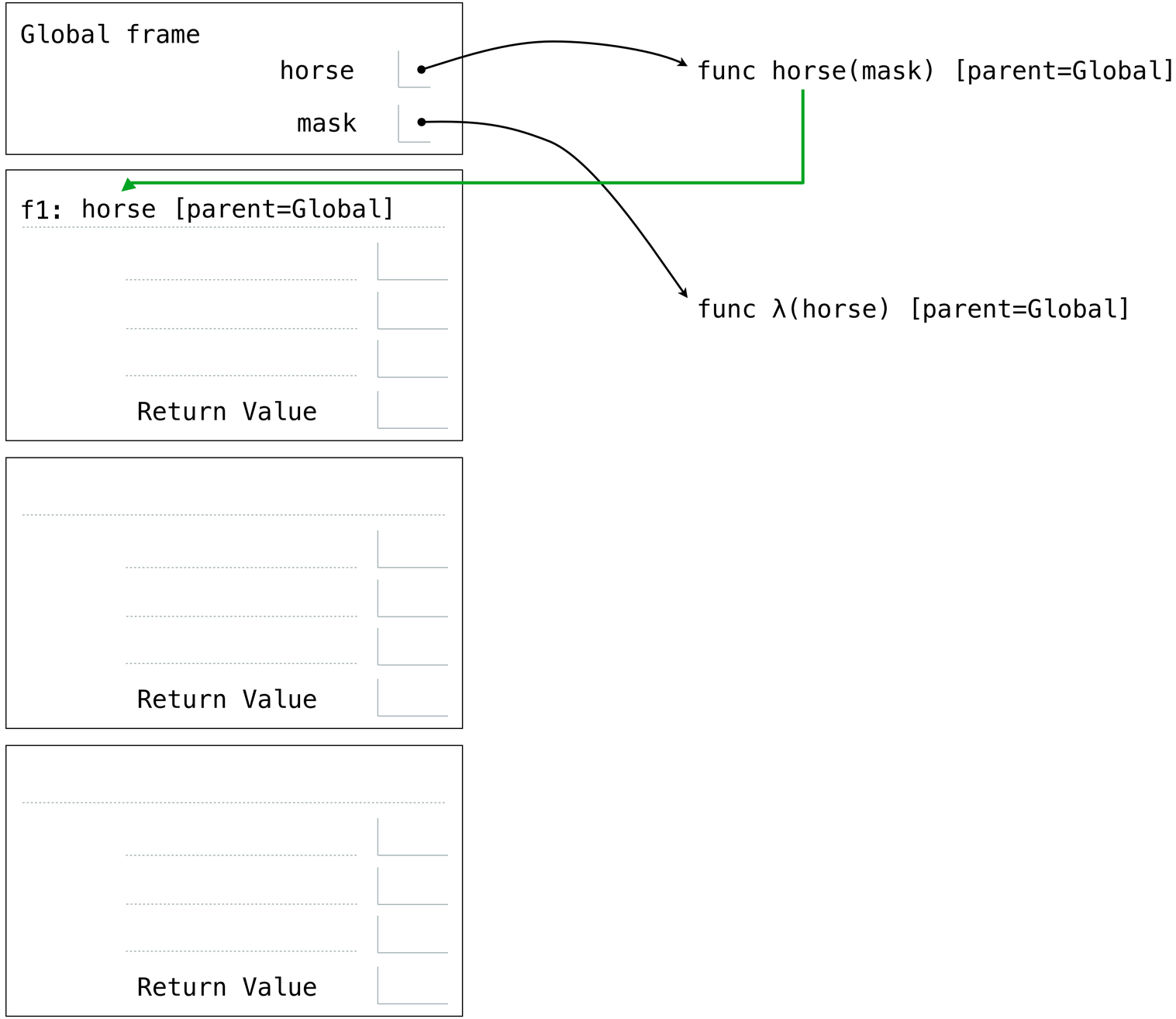
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



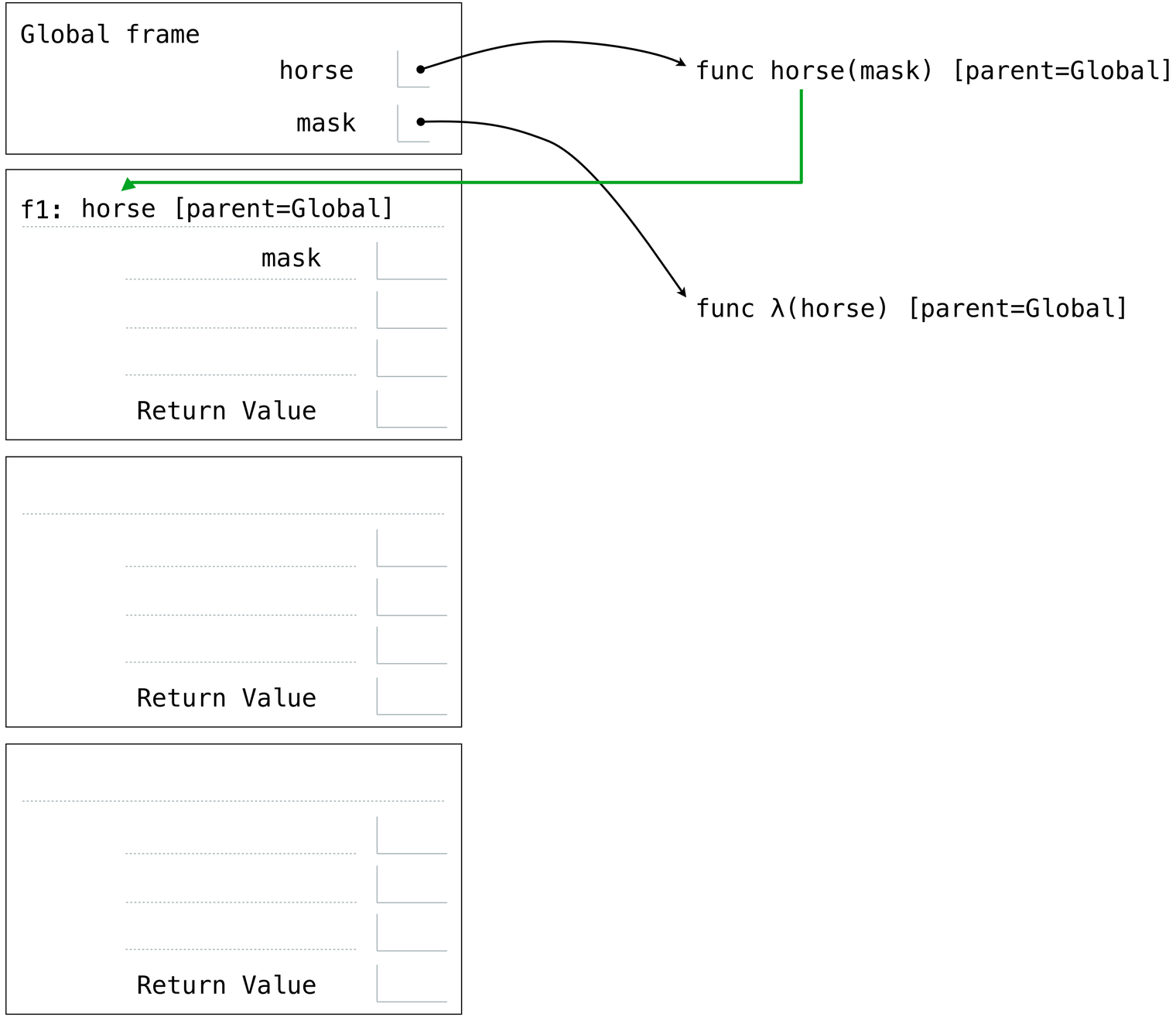
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



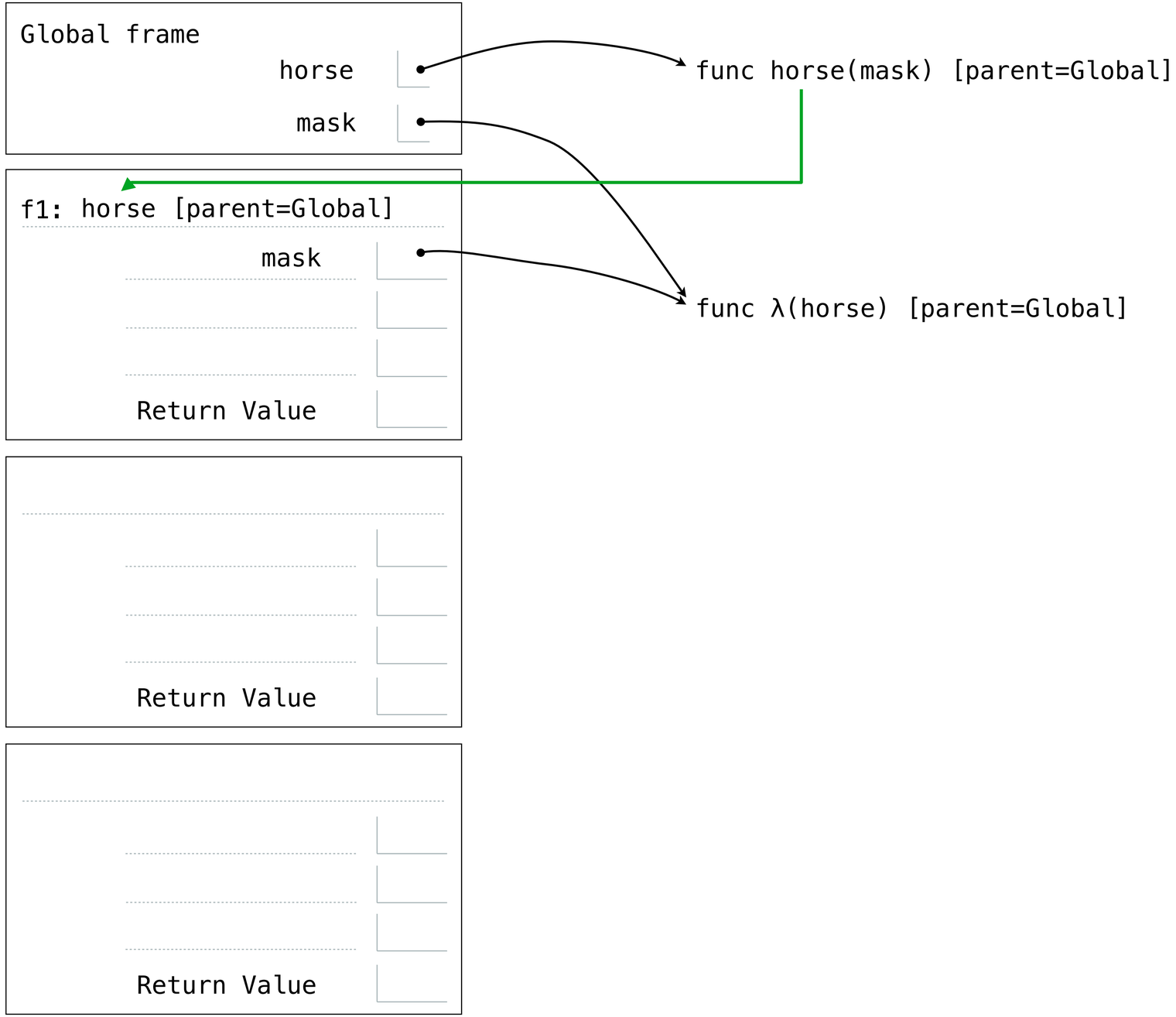
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



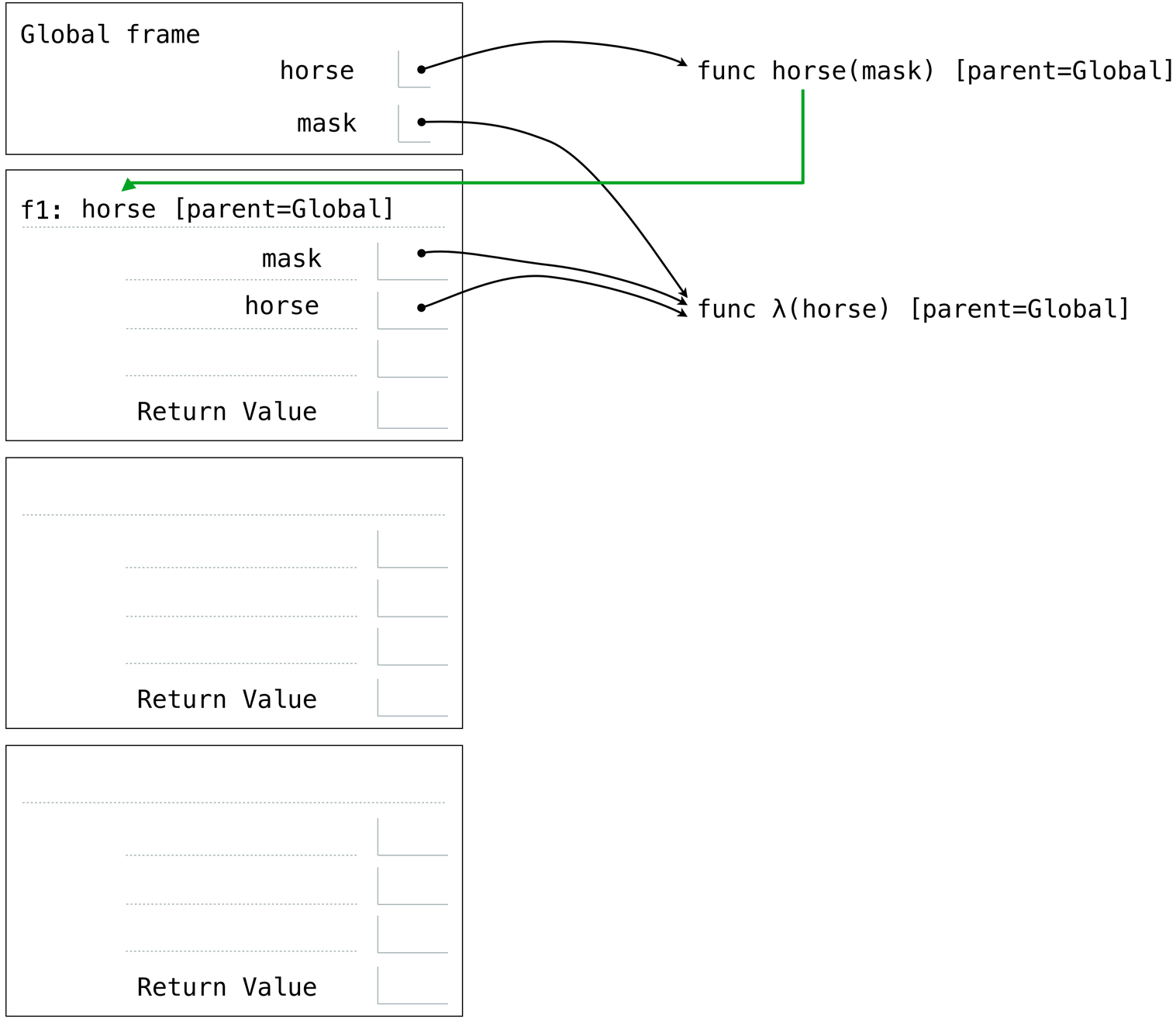
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



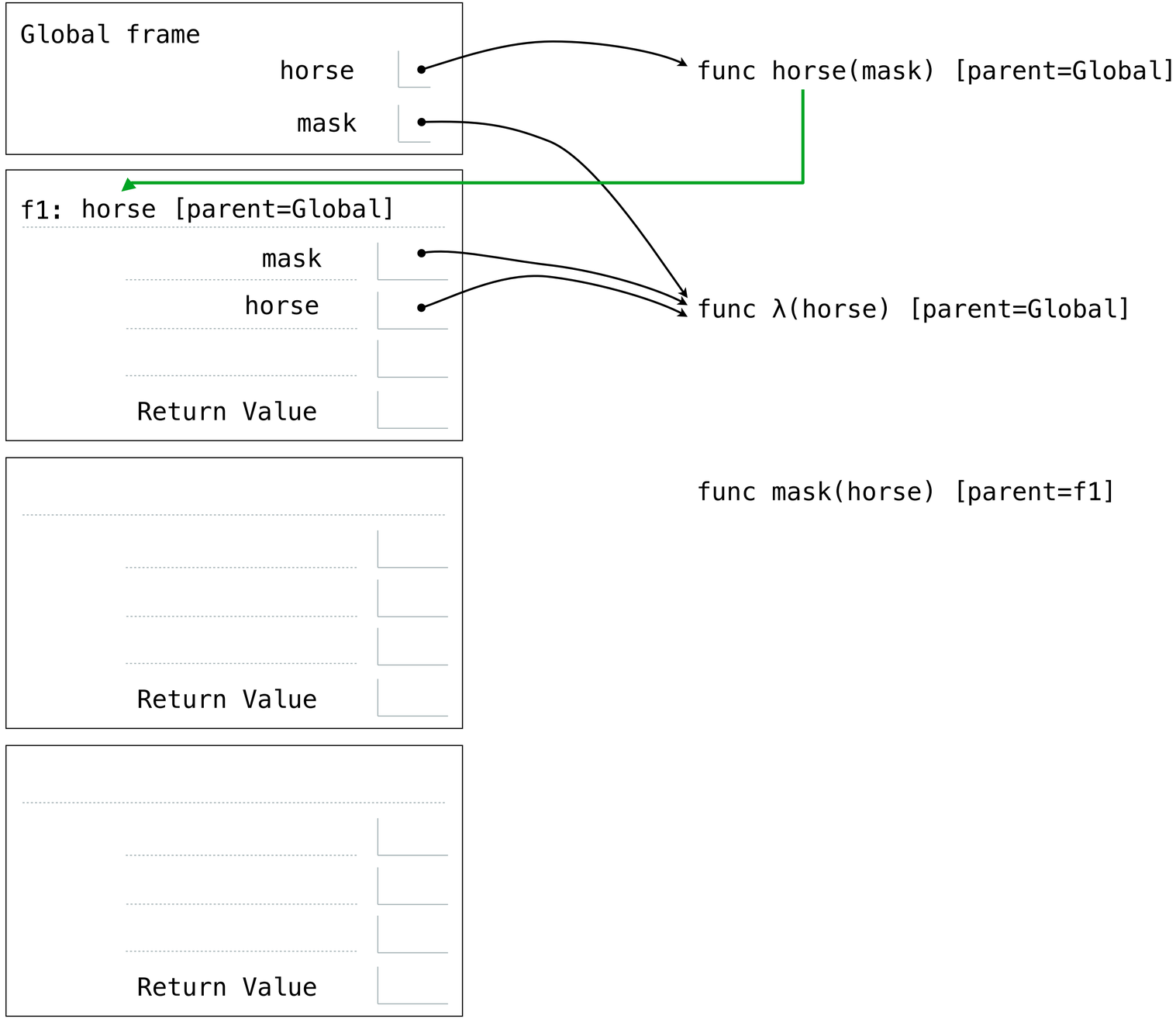
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



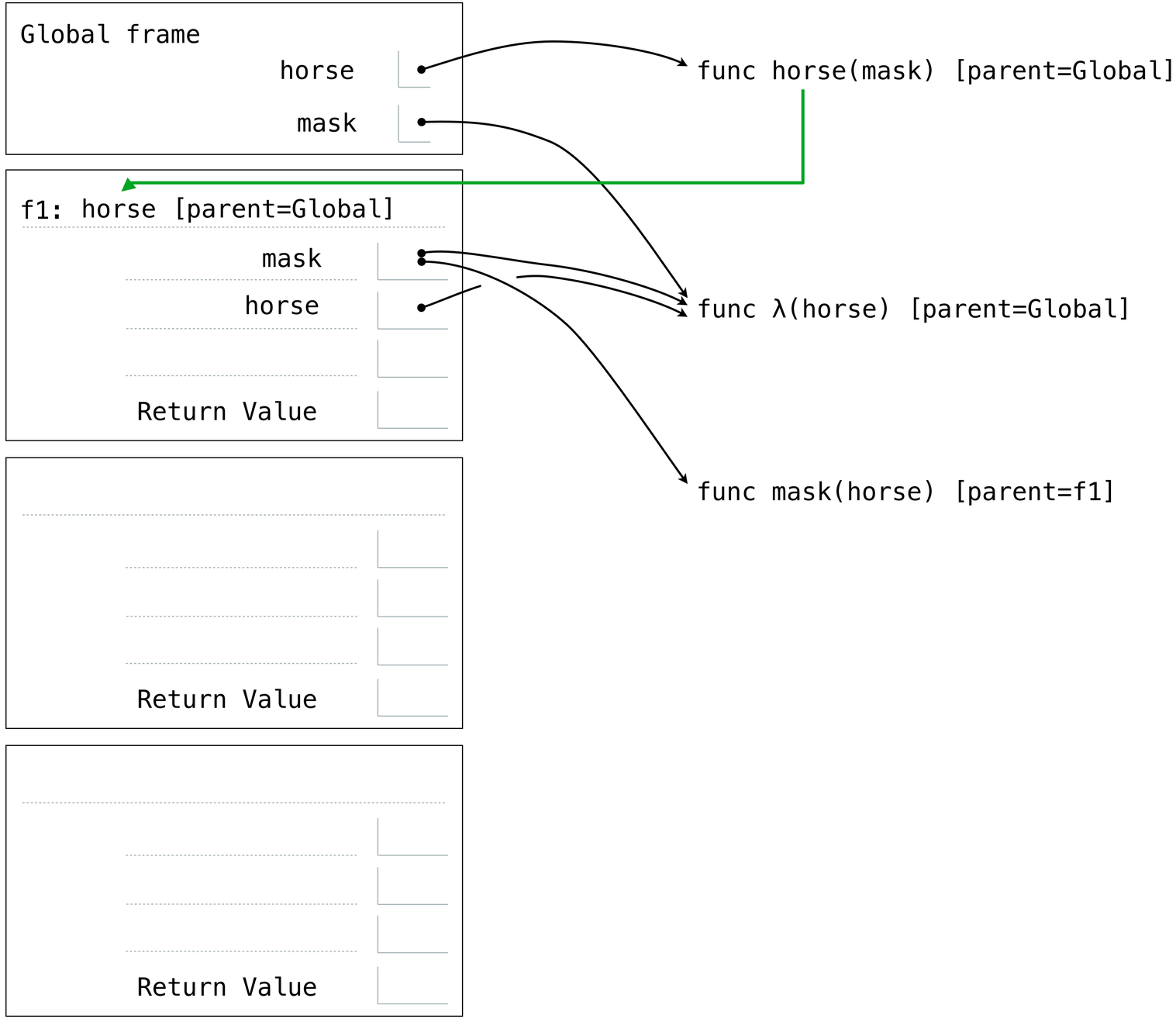
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



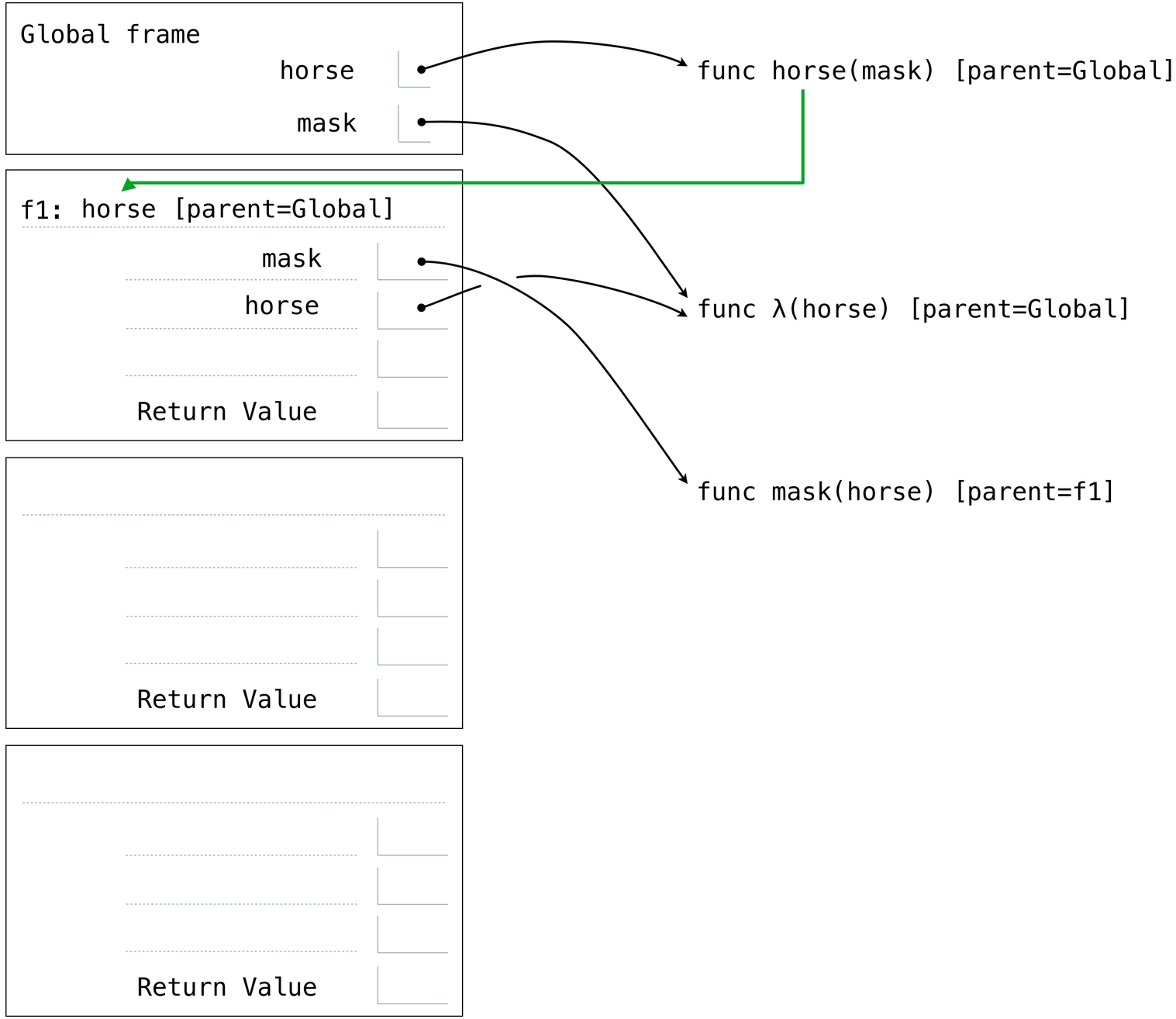
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



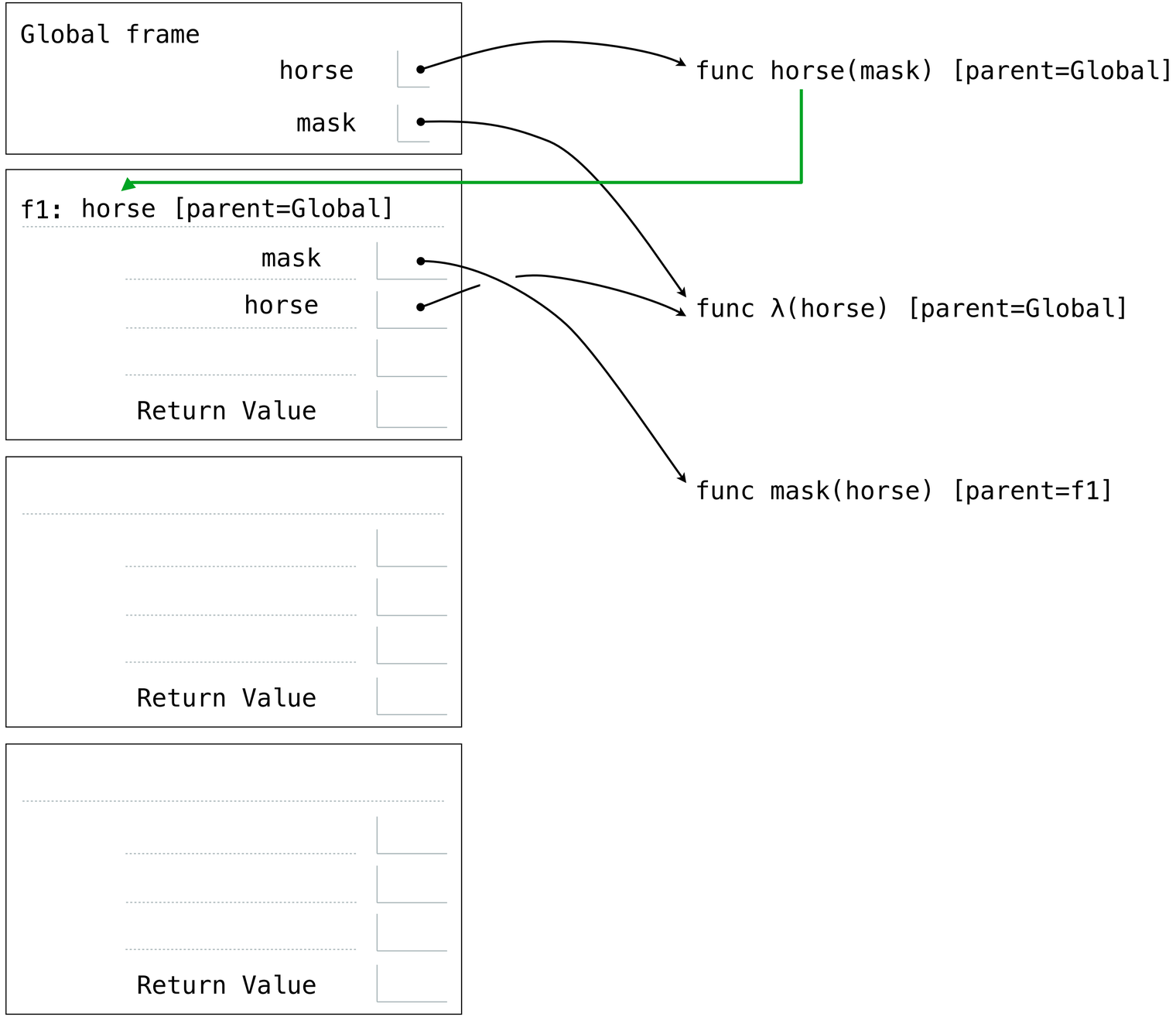
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



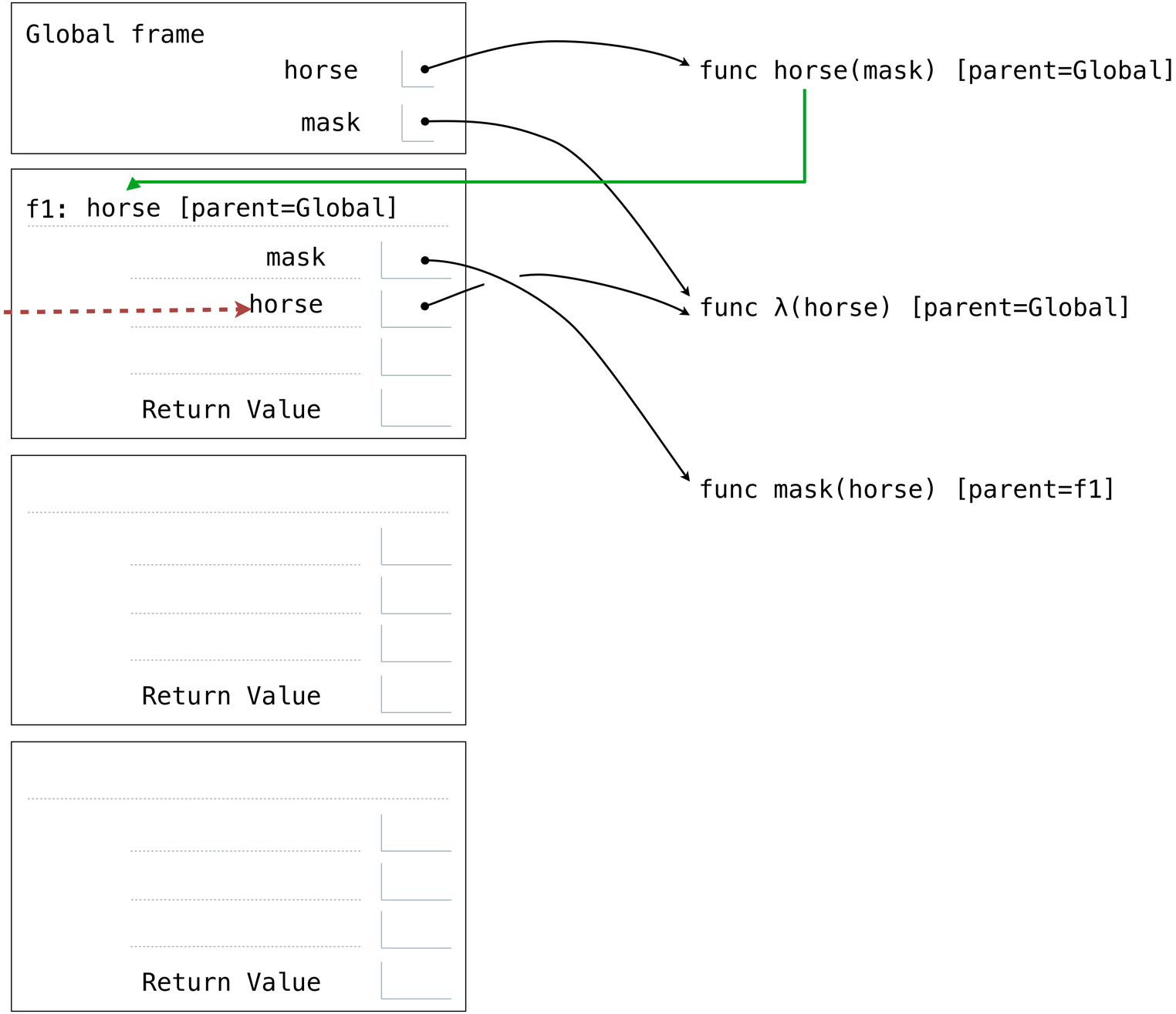
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



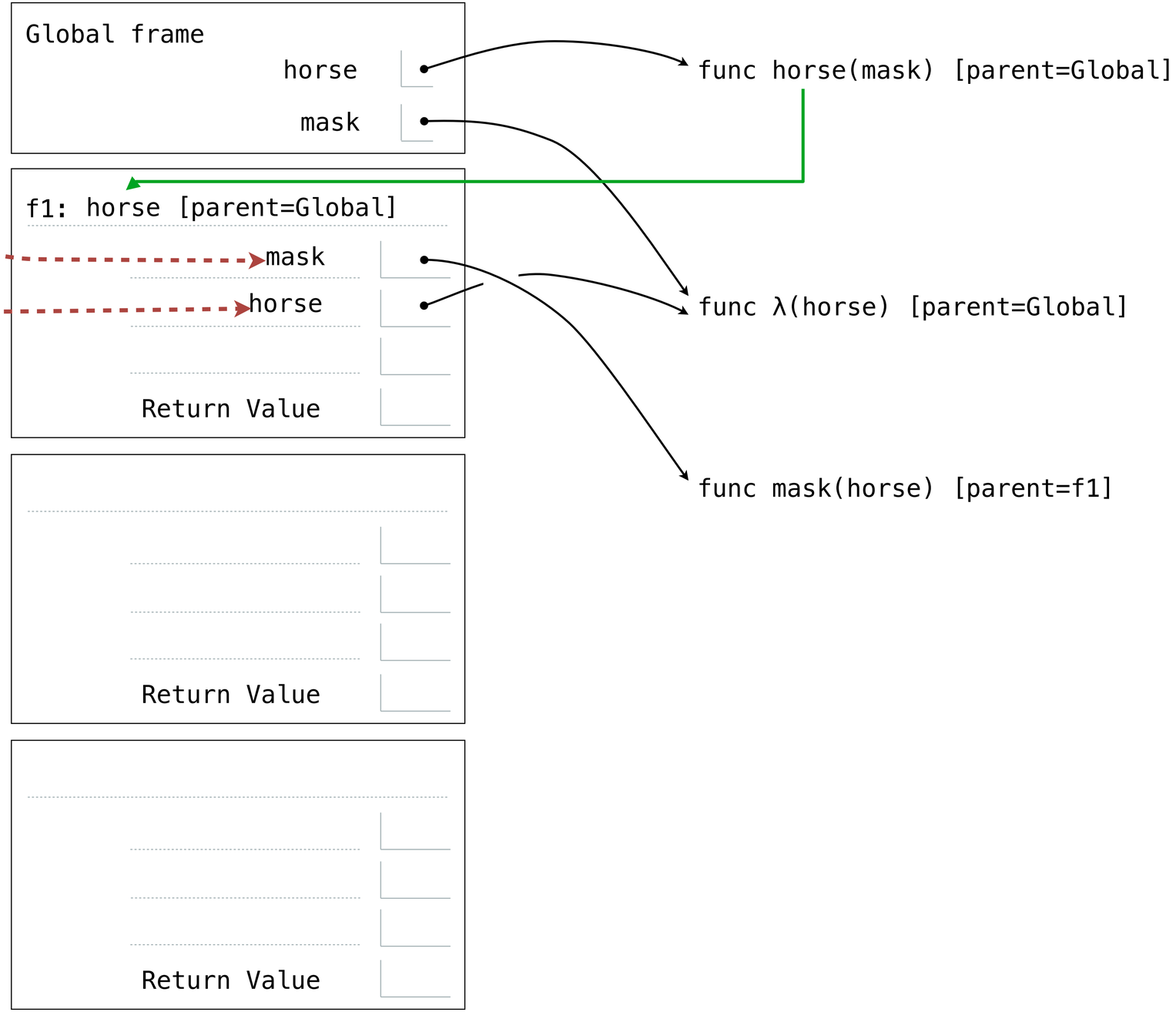
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```

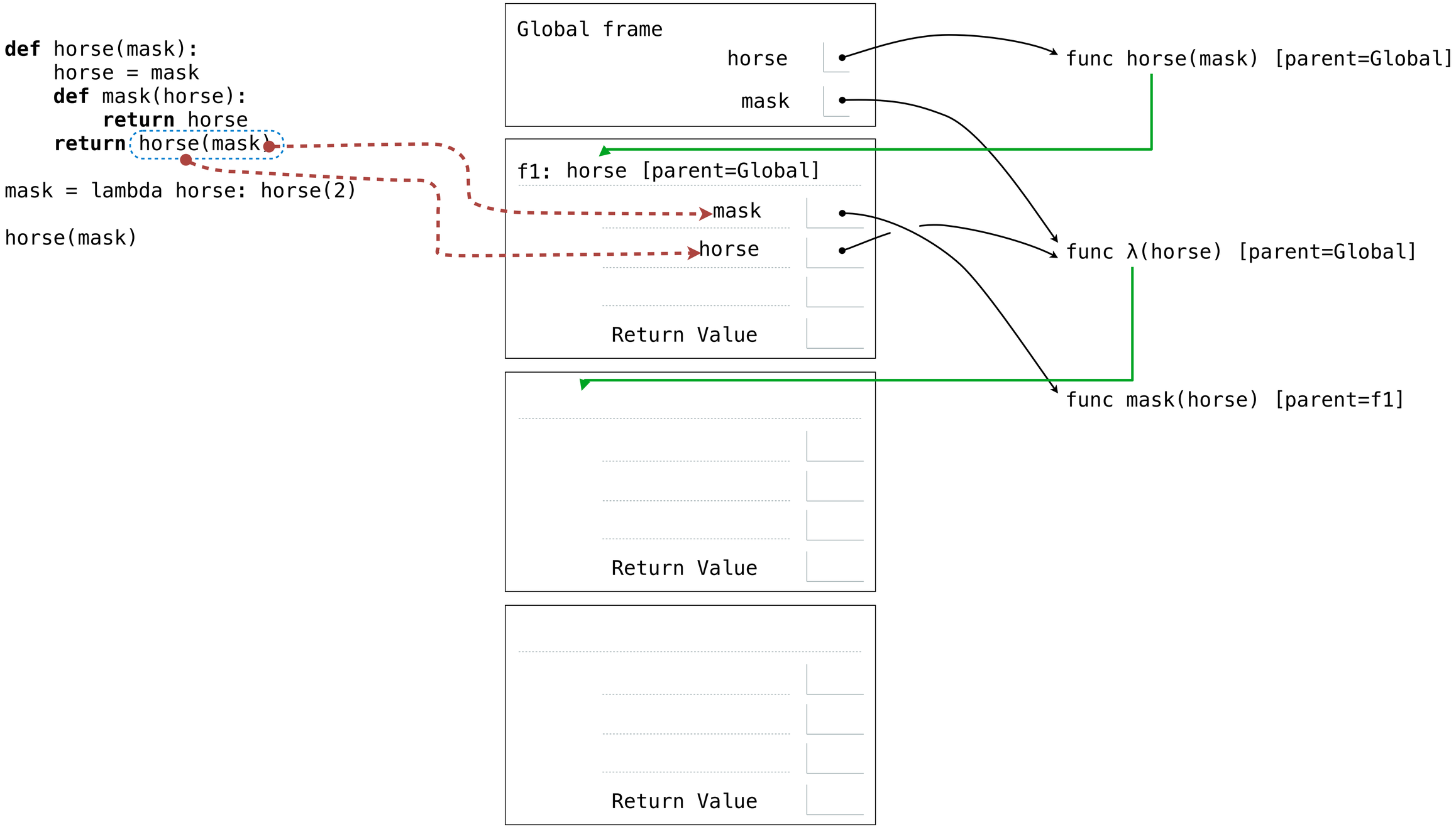


```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```

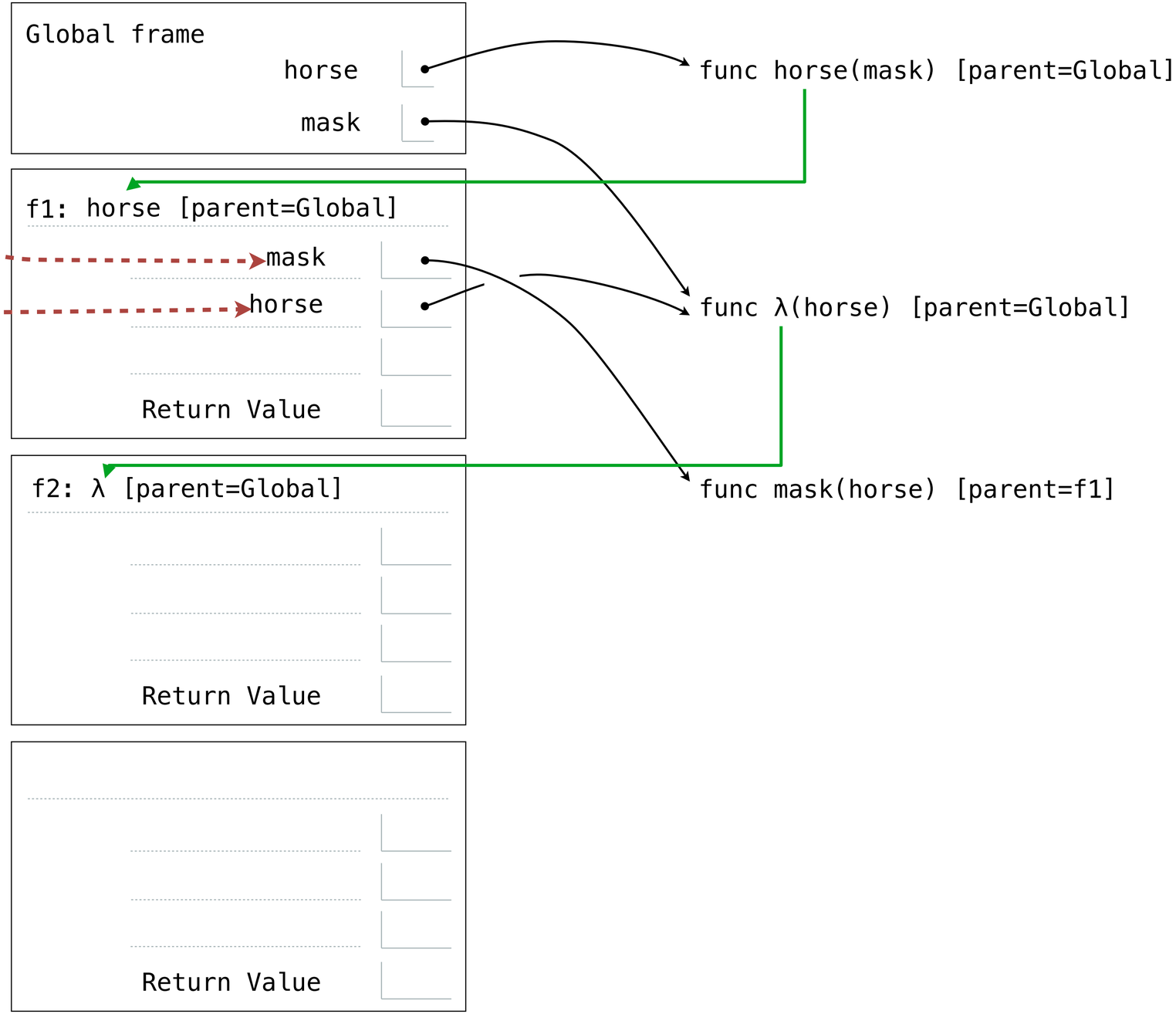


```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```

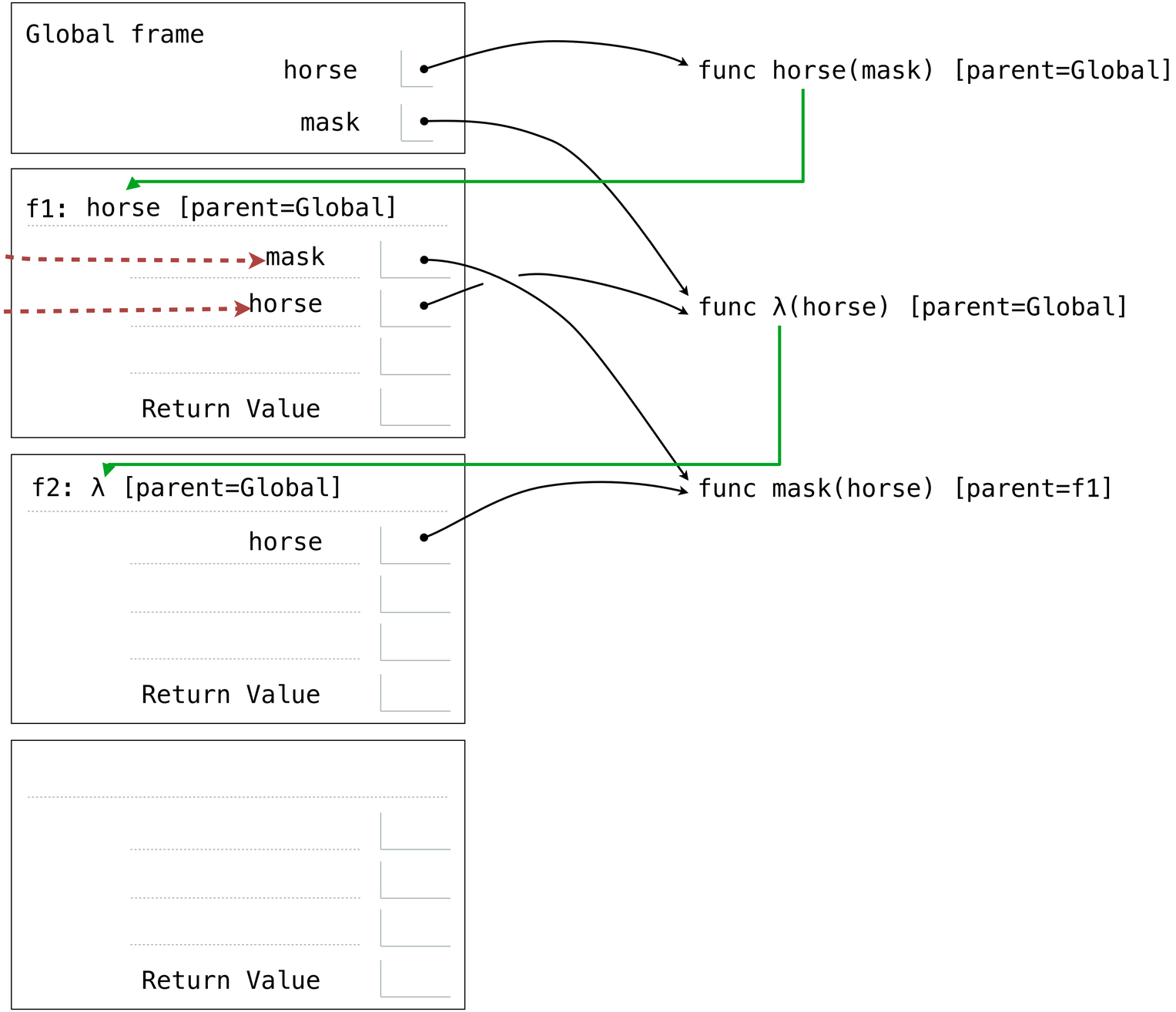



def horse(mask):
horse = mask
def mask(horse):
return horse
return horse(mask)
mask = lambda horse: horse(2)
horse(mask)
Global frame
horse
mask
func horse(mask) [parent=Global]
f1: horse [parent=Global]
mask
horse
Return Value
func λ(horse) [parent=Global]
func mask(horse) [parent=f1]
Return Value
Return Value

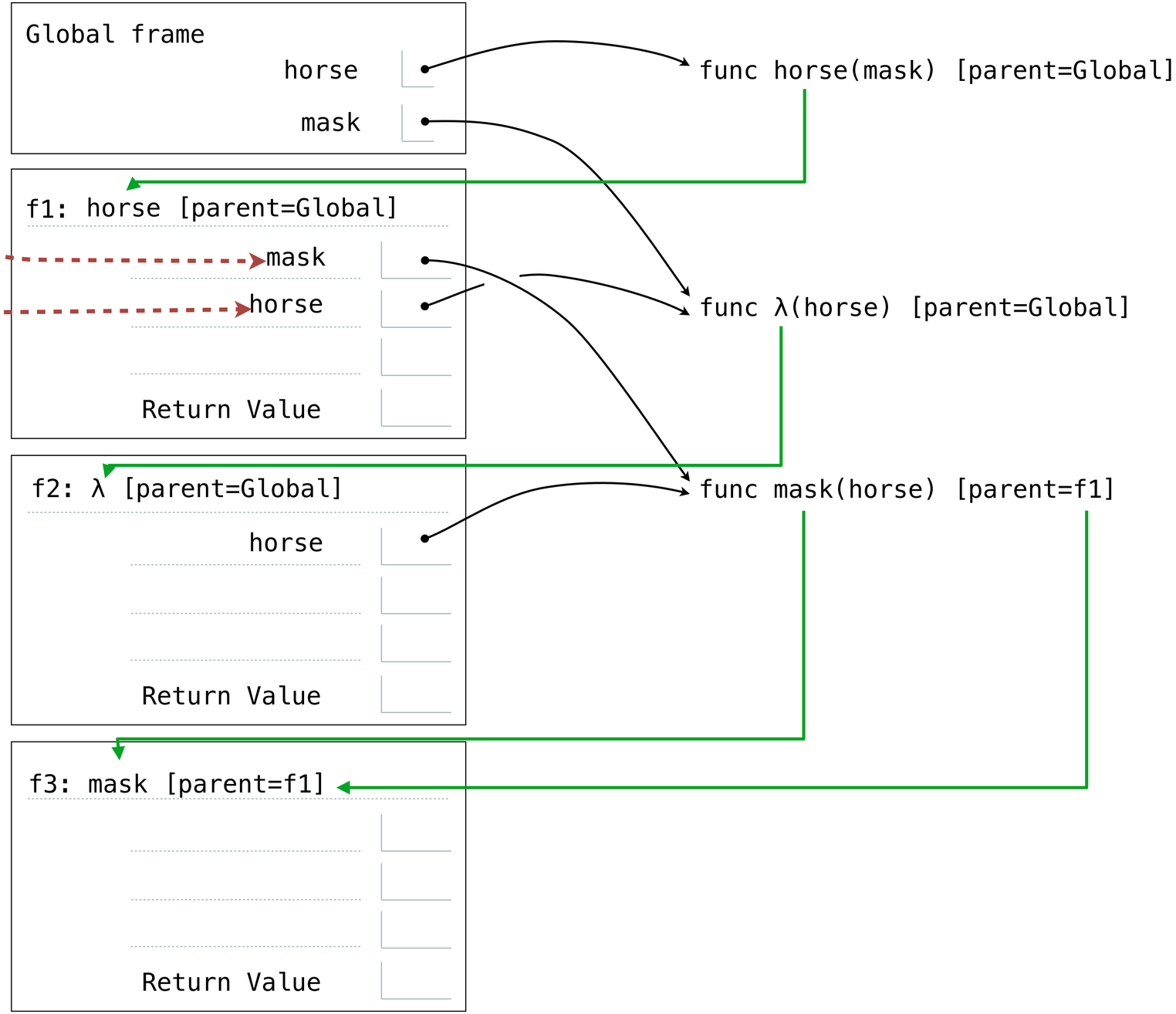
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



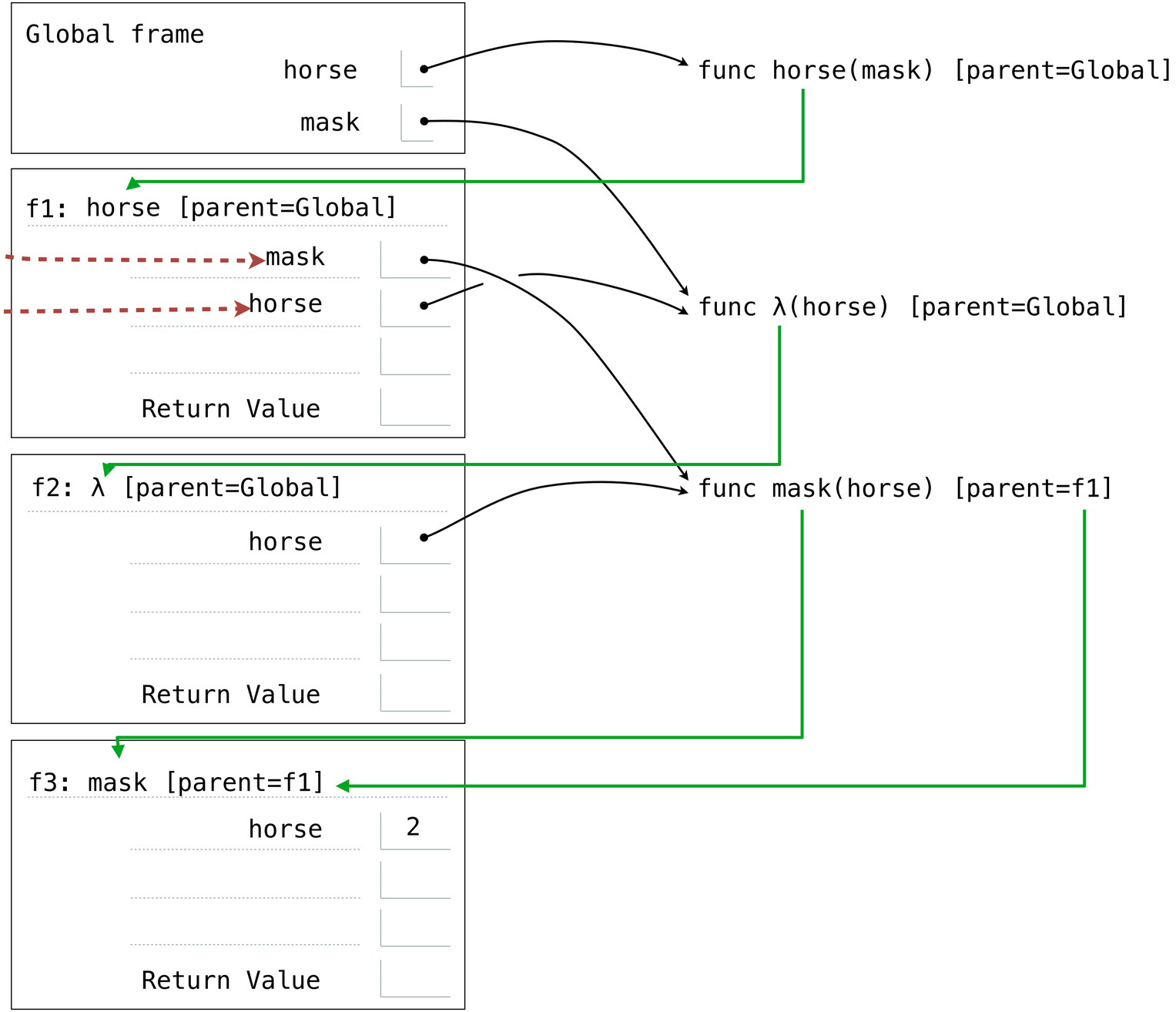
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



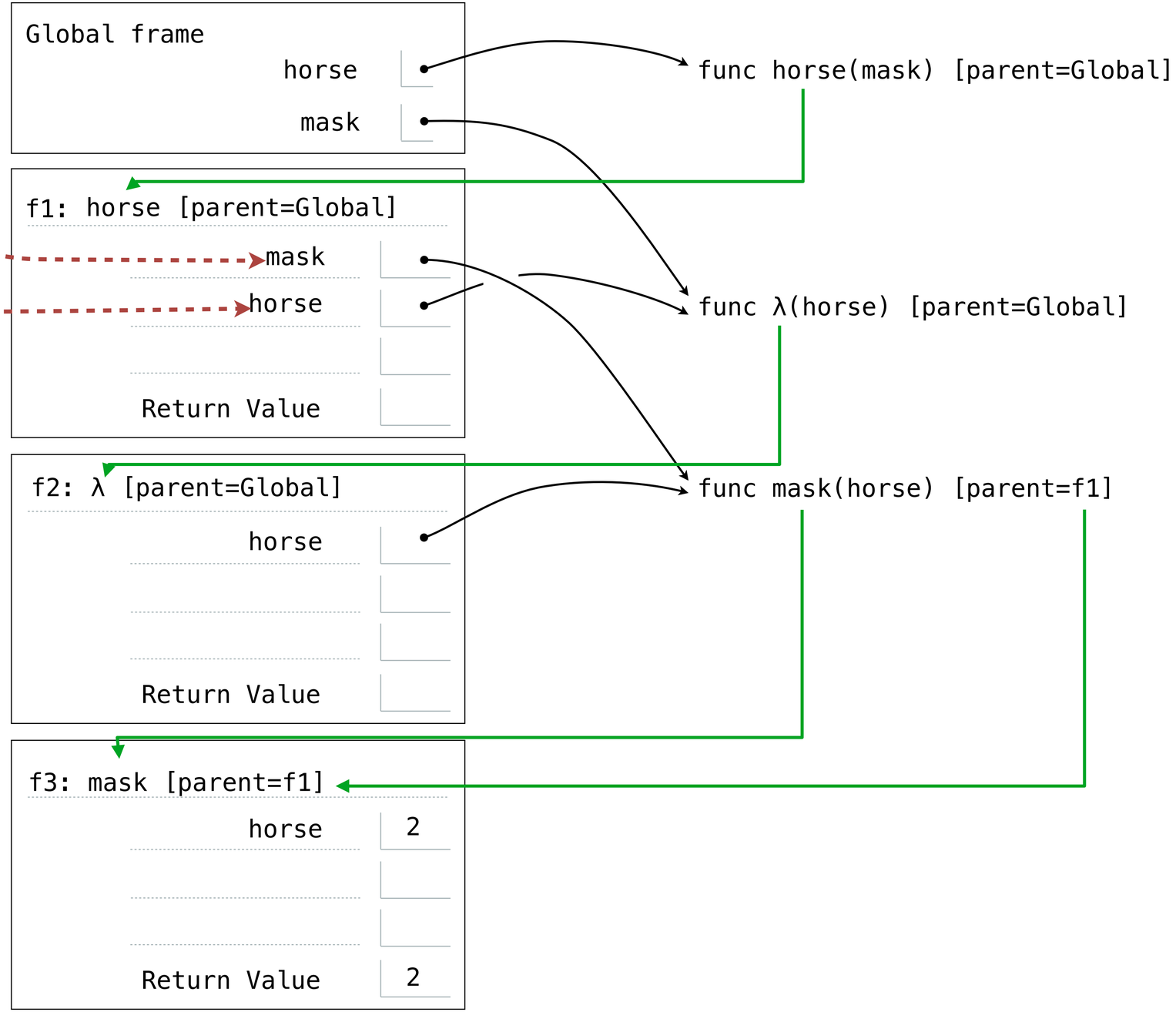
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



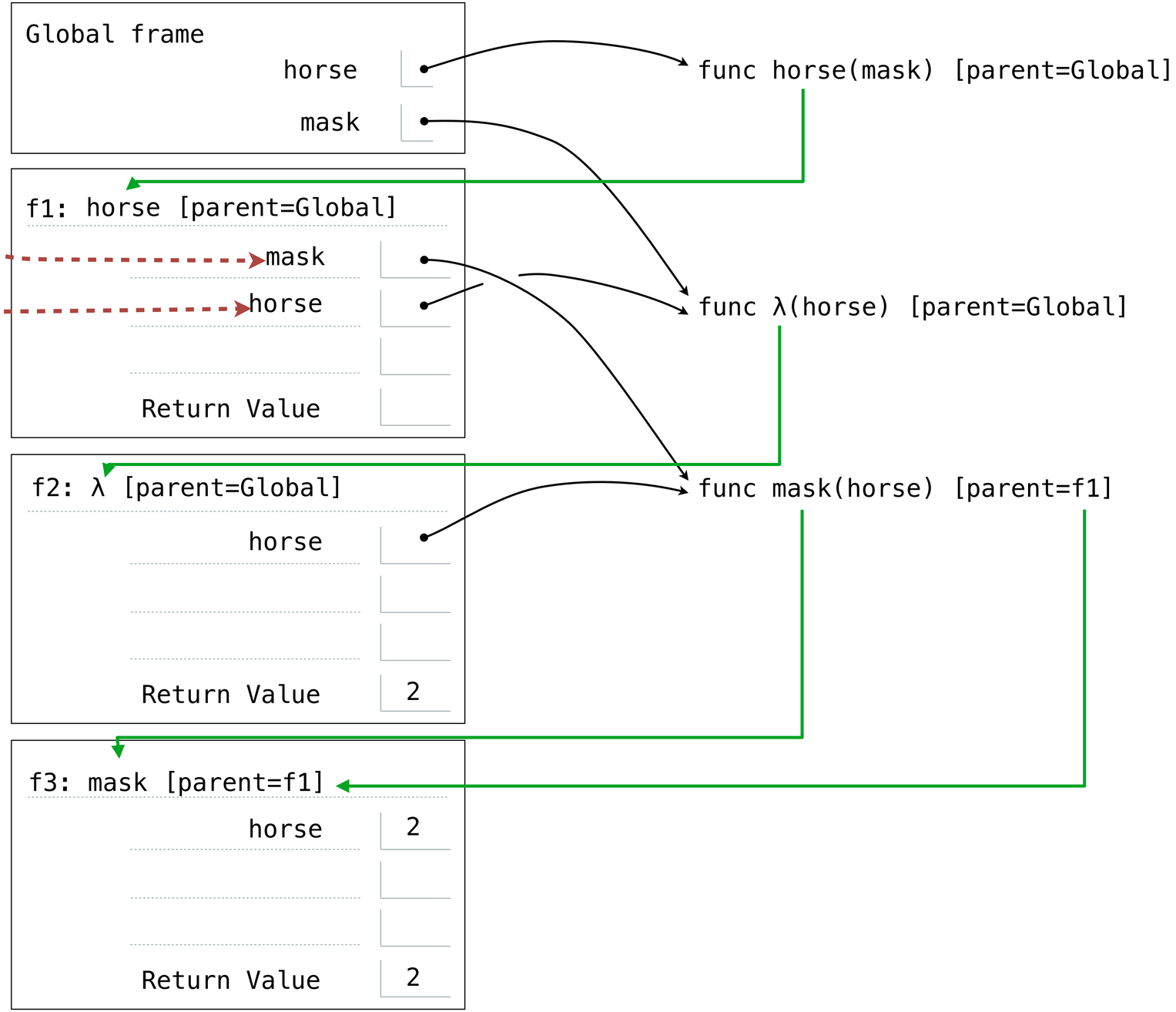
```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```



```
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)
```