при поддержке

{а}вижн

Лекция 3

в которой будет показано как окружения влияют на выражения и
что придумал Джордж Буль

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

Множество окружений

Жизненный цикл пользовательской функции

Инструкция def:

Вызывающее выражение:

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:

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

Вызывающее выражение:

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:

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

Вызывающее выражение:

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:

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

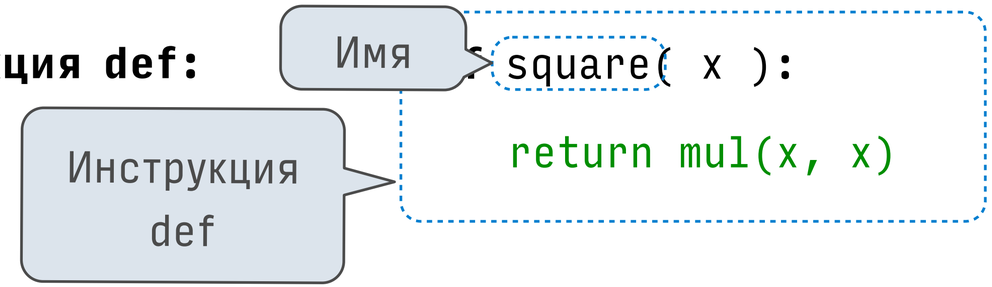
Инструкция def

Вызывающее выражение:

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:

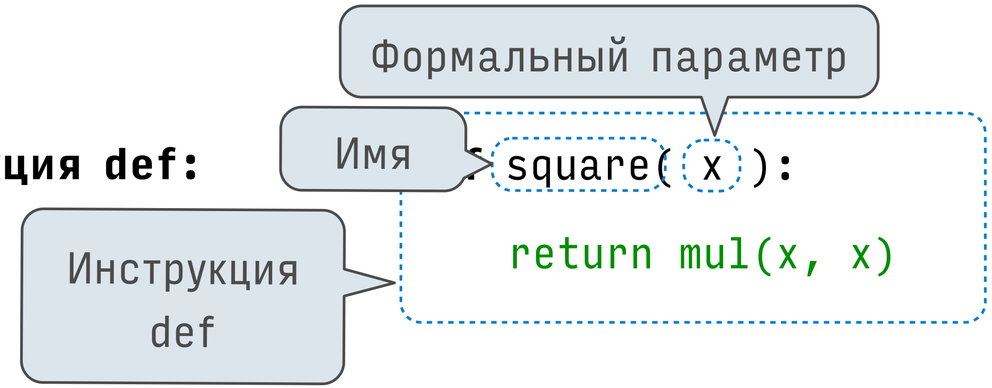


Вызывающее выражение:

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:

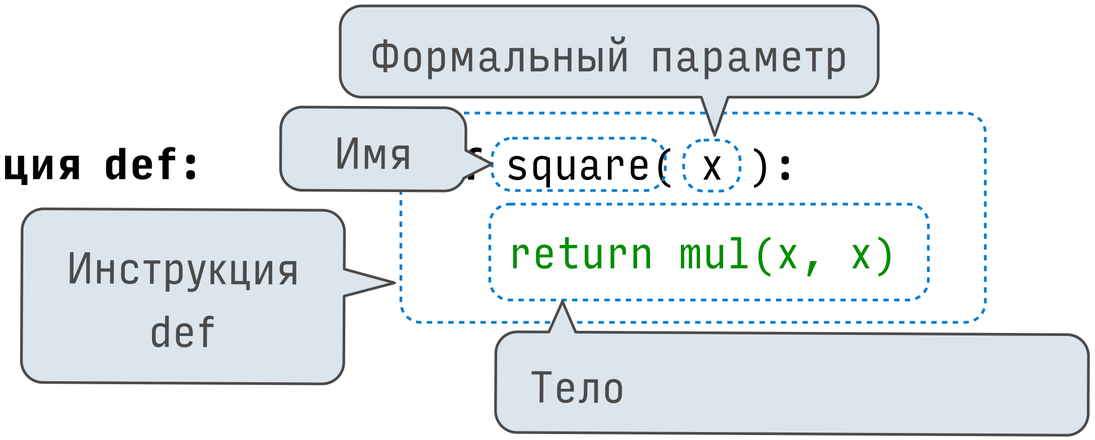


Вызывающее выражение:

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:

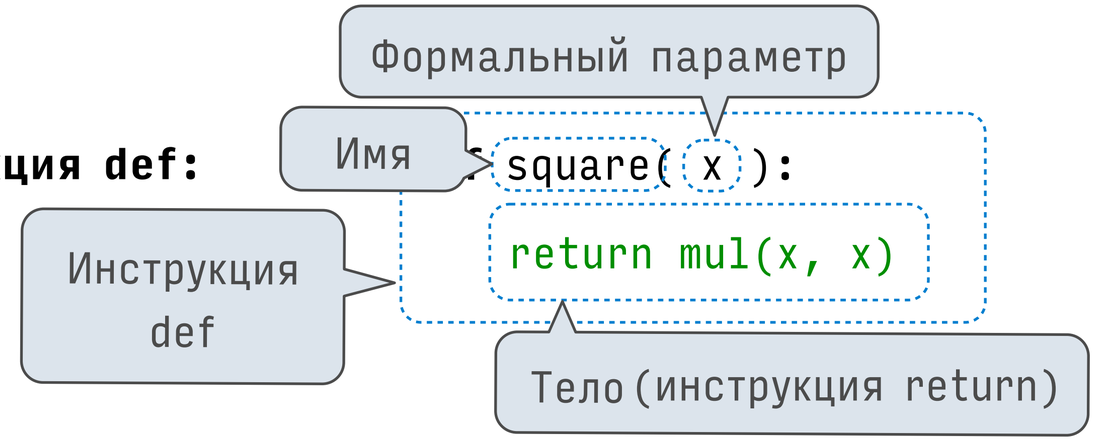


Вызывающее выражение:

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:

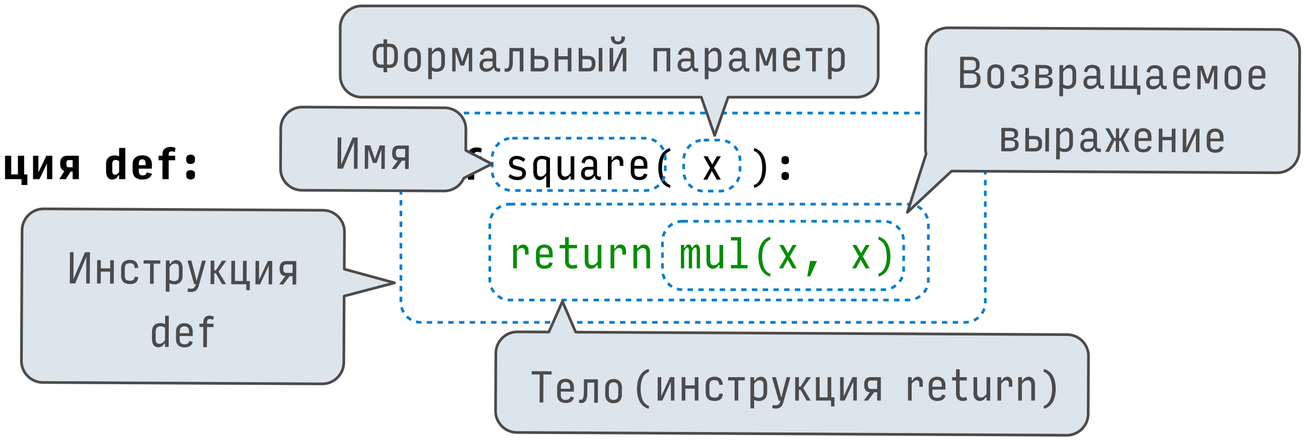


Вызывающее выражение:

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:

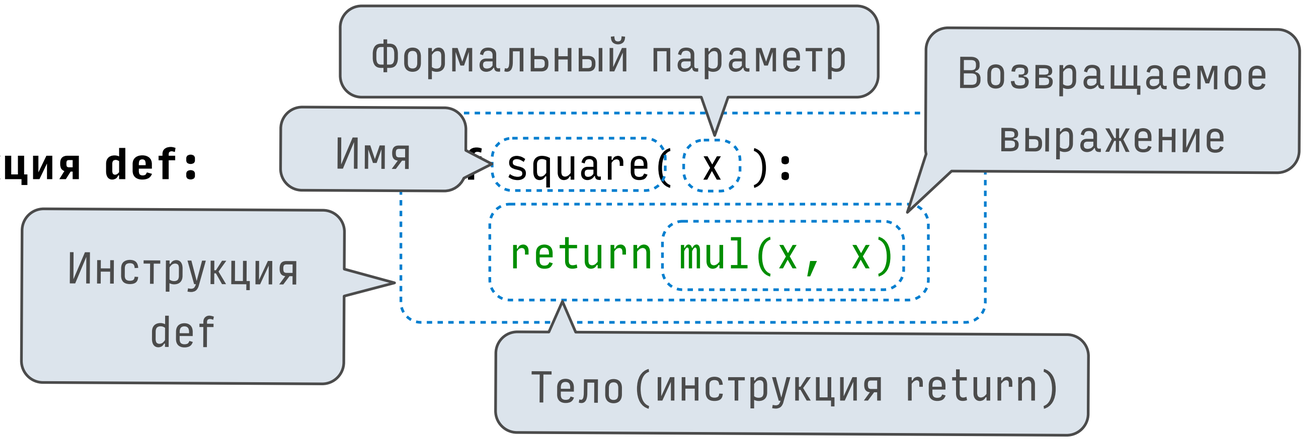


Вызывающее выражение:

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:



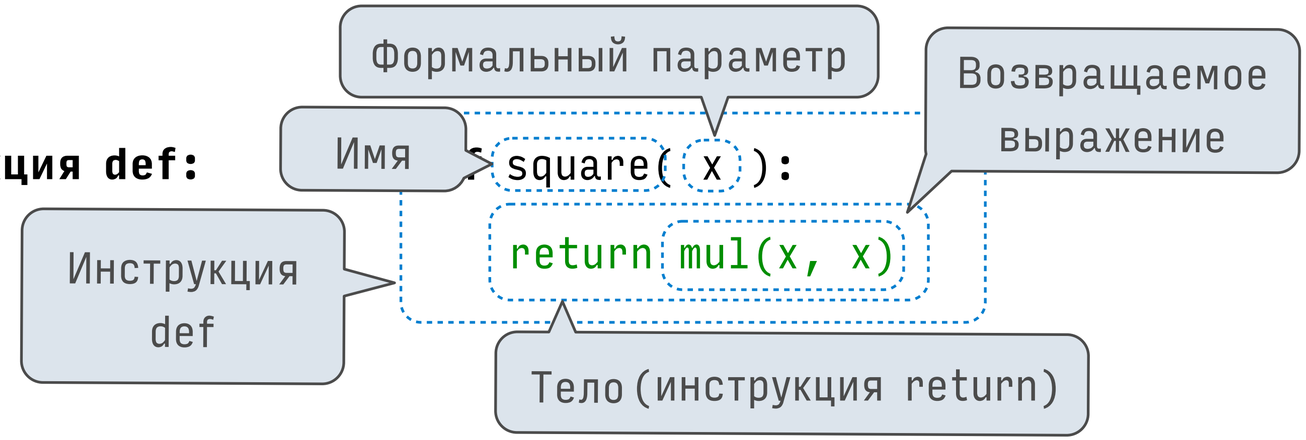
Создается новая функция!

Вызывающее выражение:

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:



`

Создается новая функция!

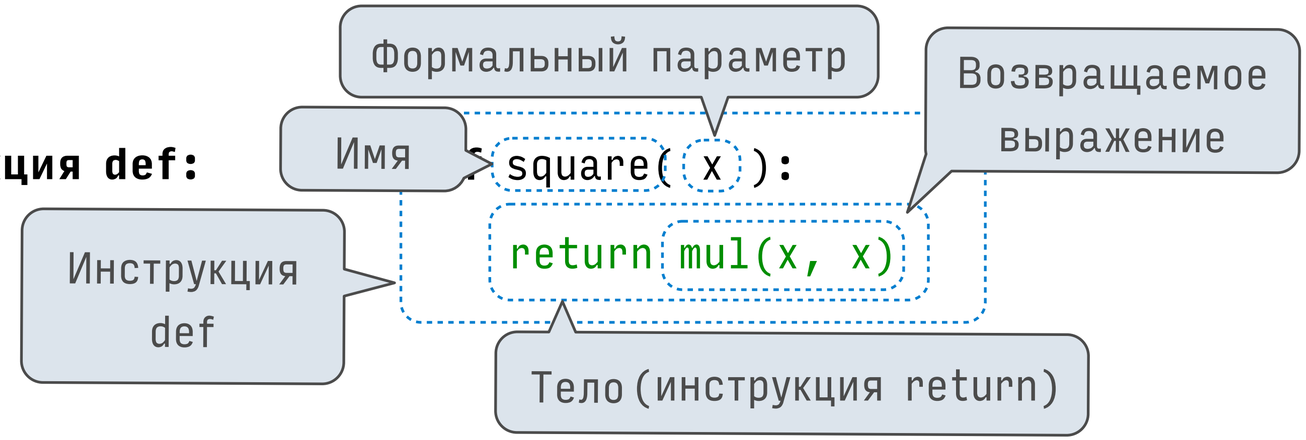
В текущем фрейме имя связывается с функцией.

Вызывающее выражение:

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:



`

Создается новая функция!

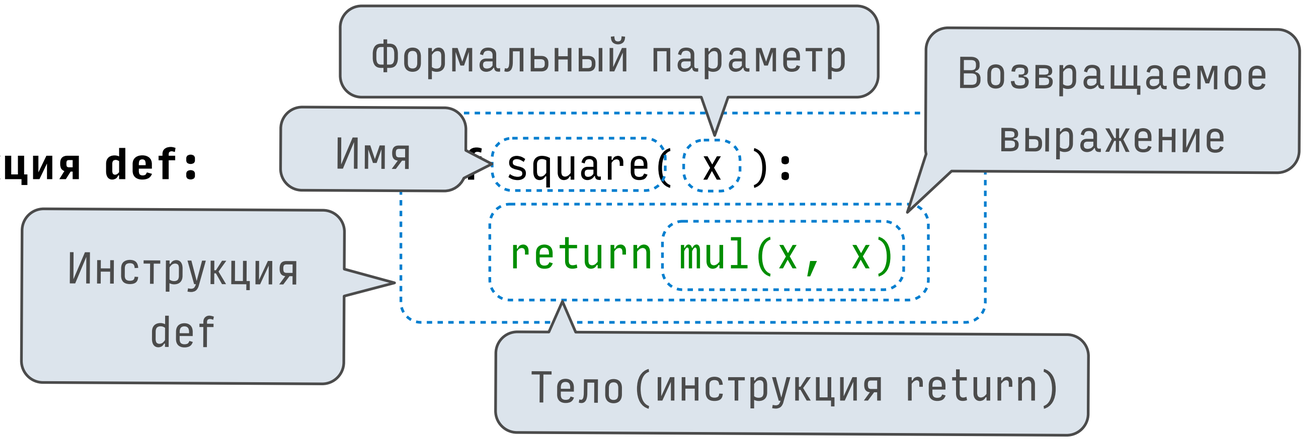
В текущем фрейме имя связывается с функцией.

Вызывающее выражение: square(2+2)

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:



`

Создается новая функция!

В текущем фрейме имя связывается с функцией.

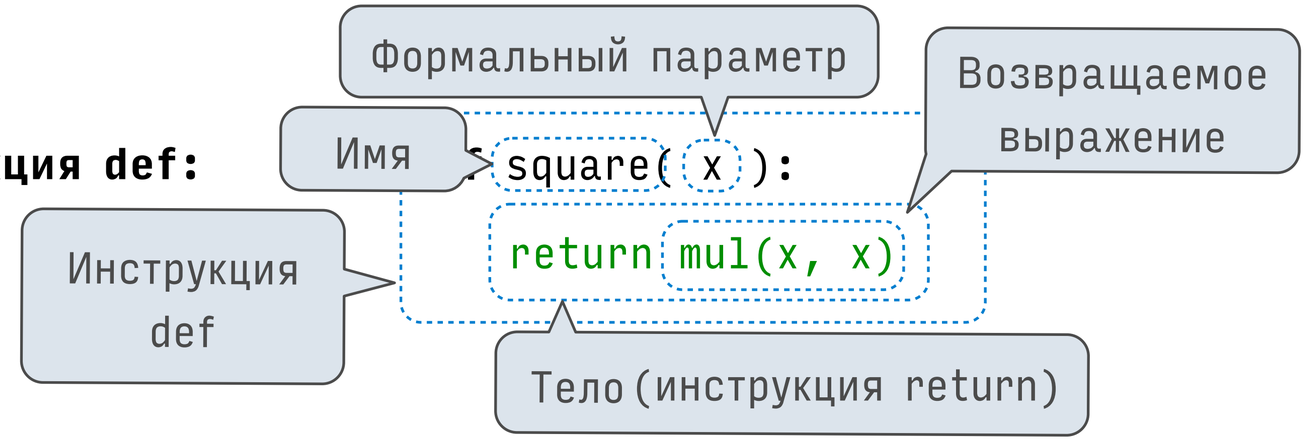
Вызывающее выражение: square(2+2)

оператор: square
функция: *func square(x)*

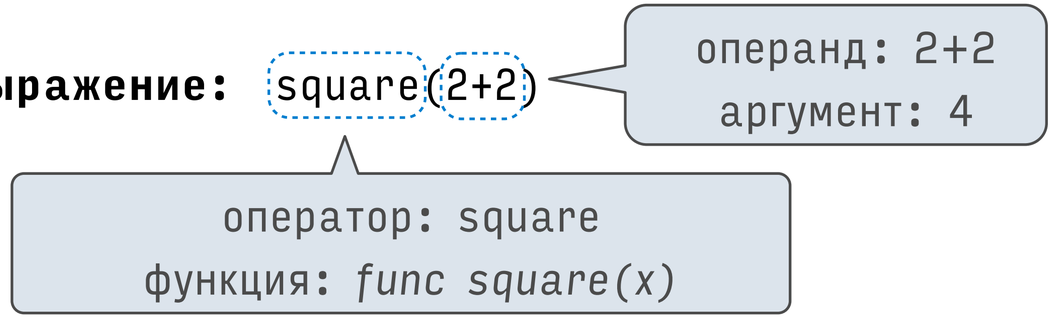
Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:



Вызывающее выражение:



Вызов/Применение:

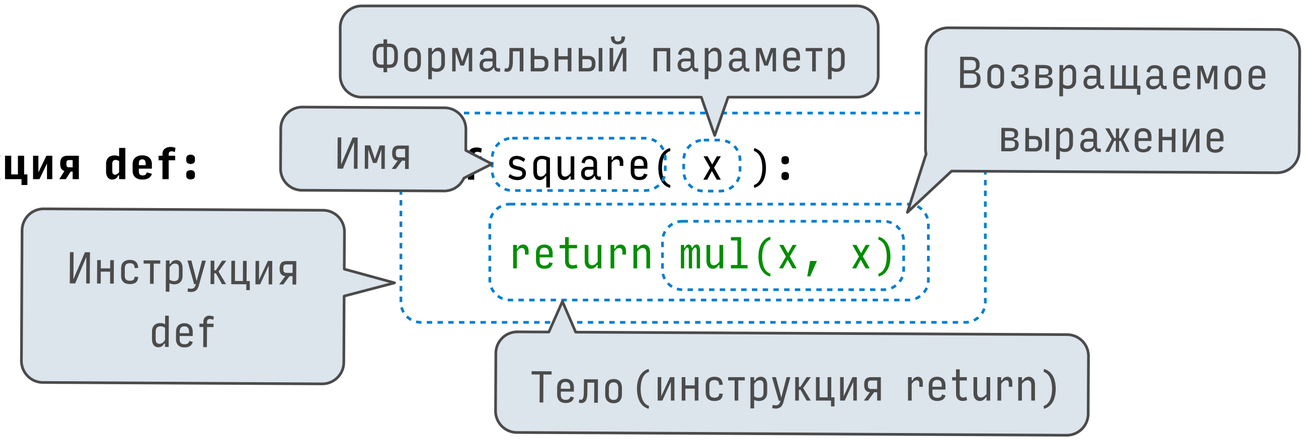
`

Создается новая функция!

В текущем фрейме имя связывается с функцией.

Жизненный цикл пользовательской функции

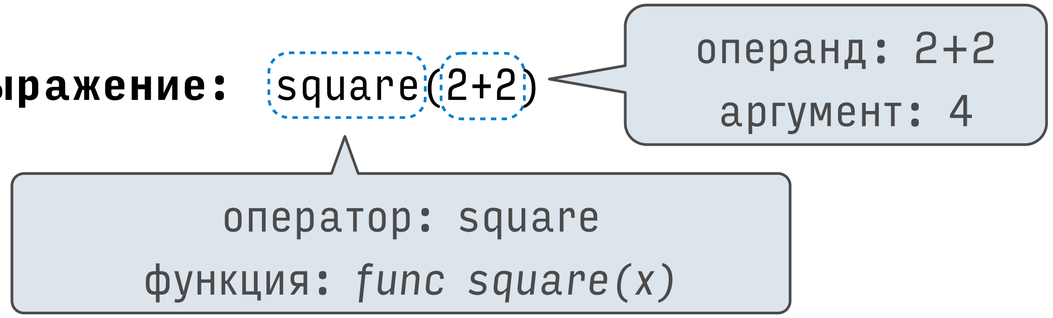
Инструкция def:



Создается новая функция!

В текущем фрейме имя связывается с функцией.

Вызывающее выражение:

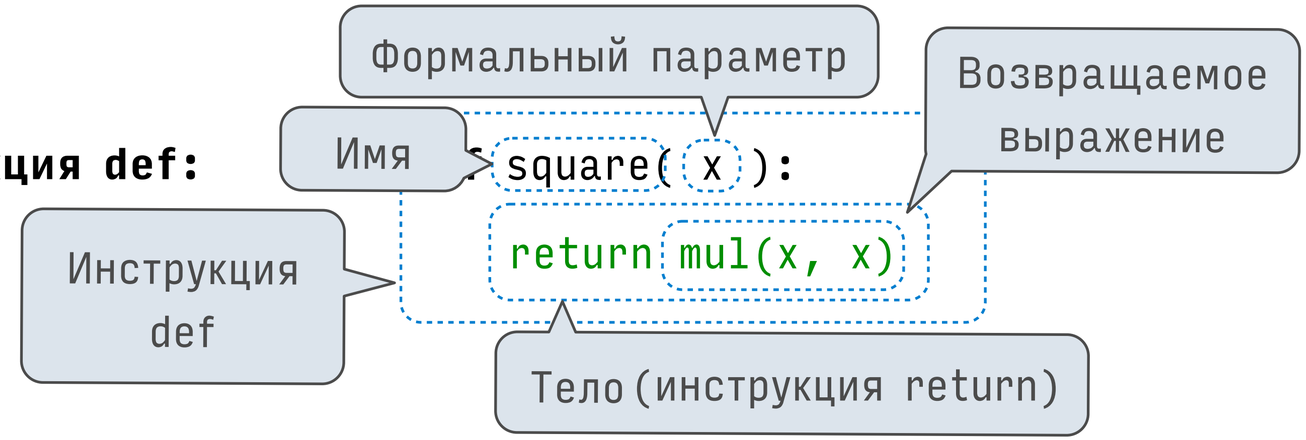


Вычисляются оператор и операнды.

Вызов/Применение:

Жизненный цикл пользовательской функции

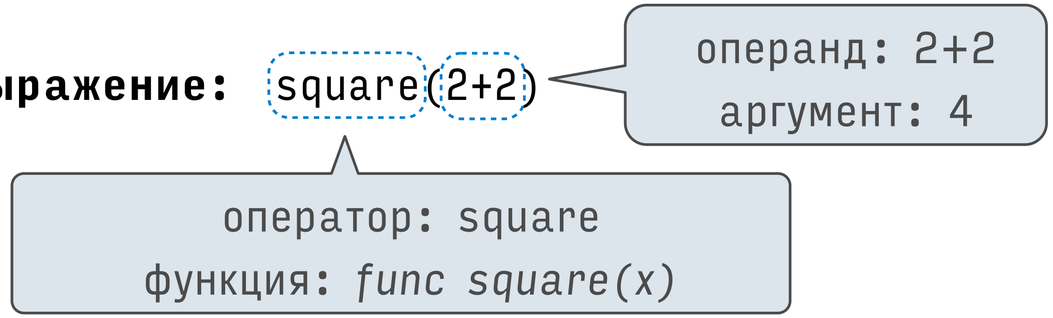
Инструкция def:



Создается новая функция!

В текущем фрейме имя связывается с функцией.

Вызывающее выражение:



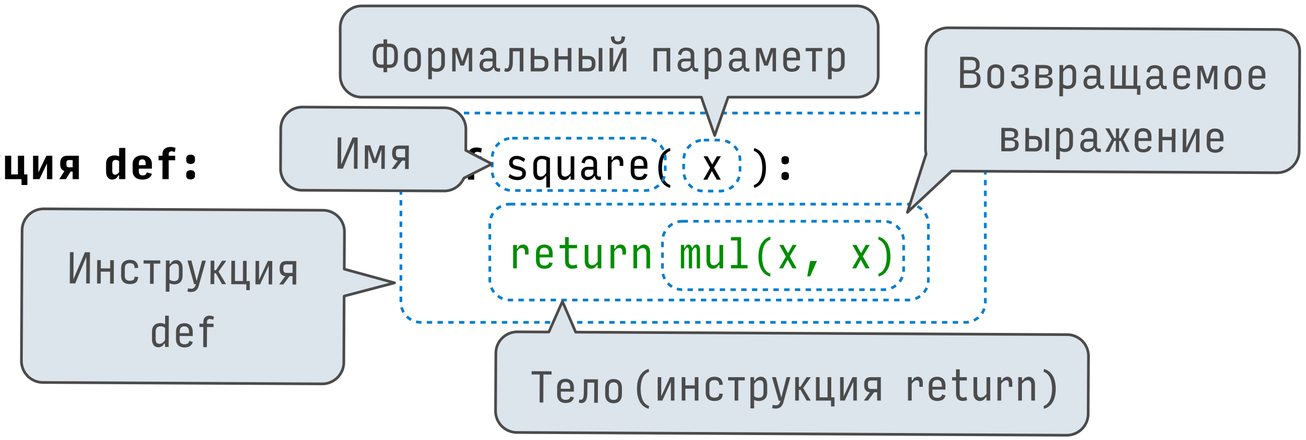
Вычисляются оператор и операнды.

Функция (значение оператора) вызывается с аргументами (значения операндов).

Вызов/Применение:

Жизненный цикл пользовательской функции

Инструкция def:

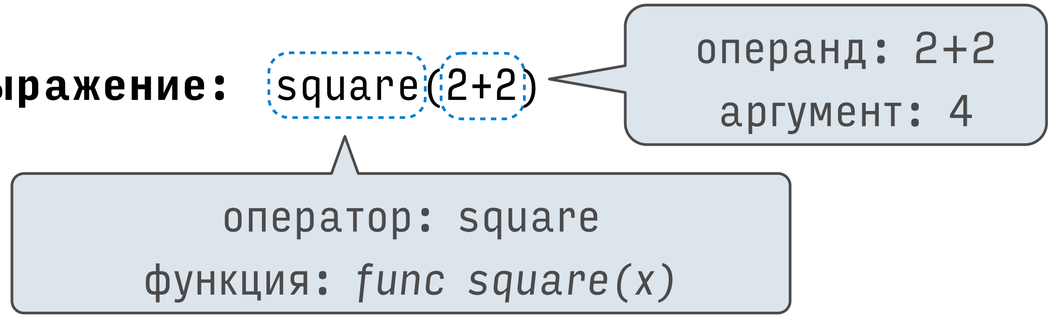


`

Создается новая функция!

В текущем фрейме имя связывается с функцией.

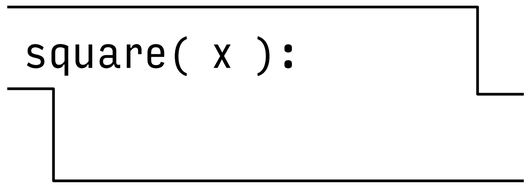
Вызывающее выражение:



Вычисляются оператор и операнды.

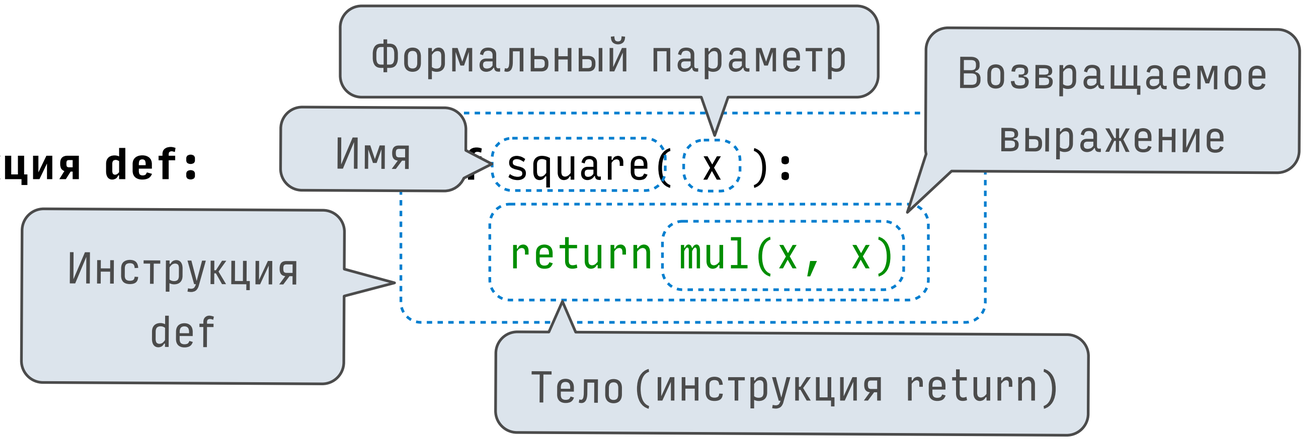
Функция (значение оператора) вызывается с аргументами (значения операндов).

Вызов/Применение:



Жизненный цикл пользовательской функции

Инструкция def:

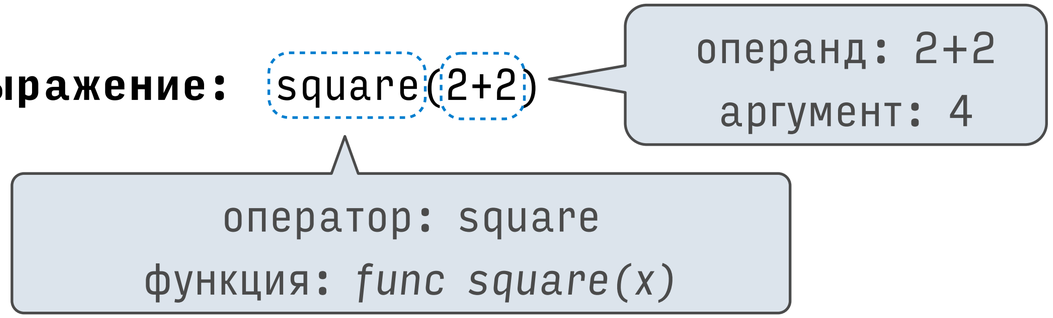


`

Создается новая функция!

В текущем фрейме имя связывается с функцией.

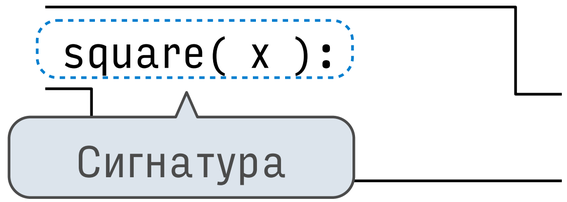
Вызывающее выражение:



Вычисляются оператор и операнды.

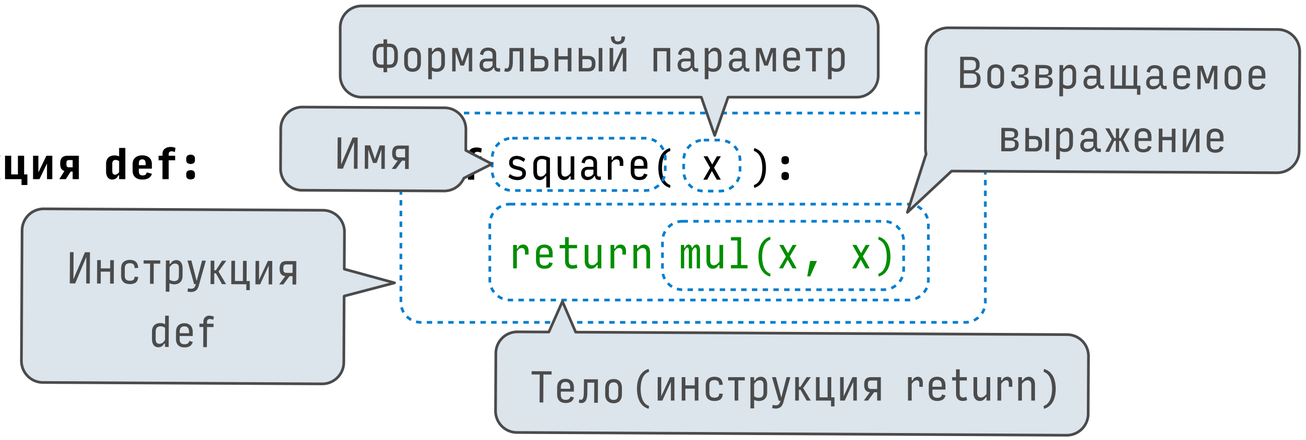
Функция (значение оператора) вызывается с аргументами (значения операндов).

Вызов/Применение:



Жизненный цикл пользовательской функции

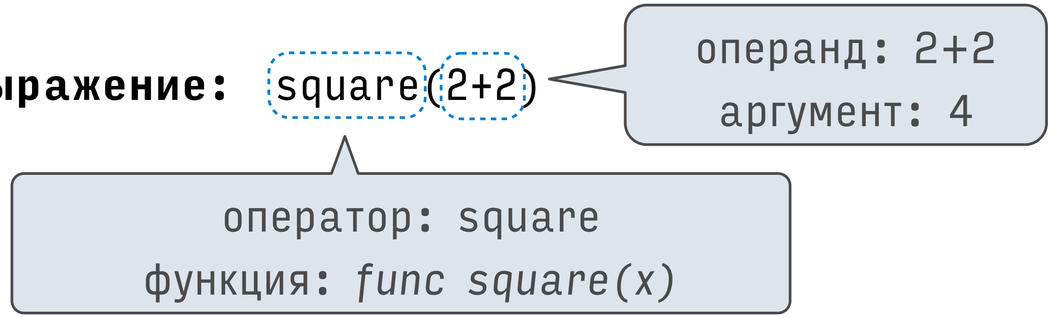
Инструкция def:



Создается новая функция!

В текущем фрейме имя связывается с функцией.

Вызывающее выражение:



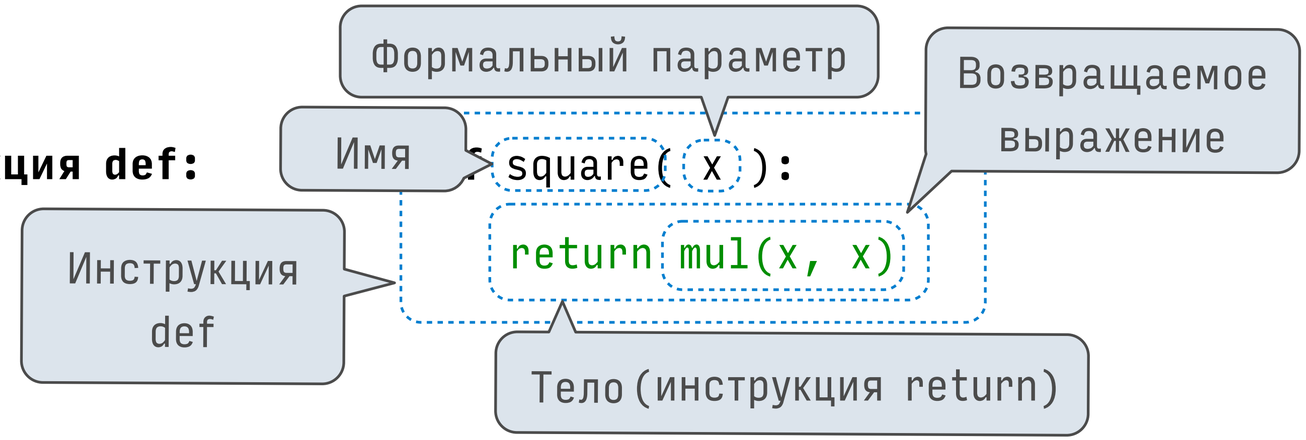
Вычисляются оператор и операнды.

Функция (значение оператора) вызывается с аргументами (значения операндов).

Вызов/Применение:

Жизненный цикл пользовательской функции

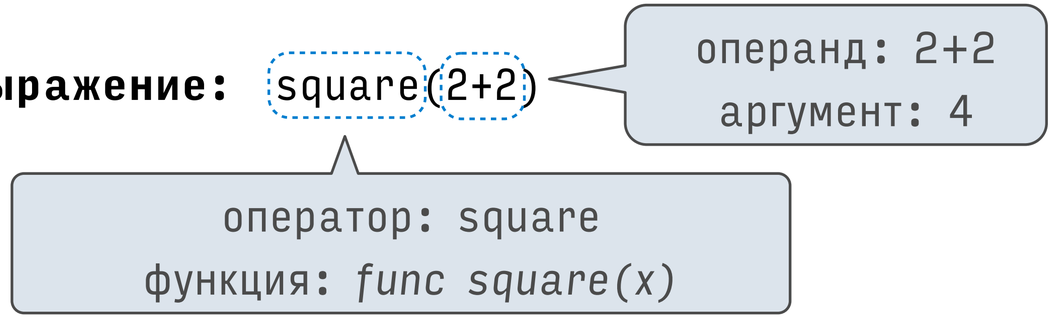
Инструкция def:



Создается новая функция!

В текущем фрейме имя связывается с функцией.

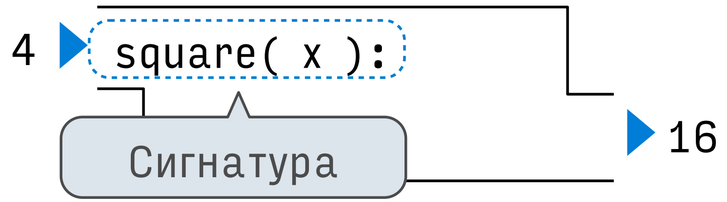
Вызывающее выражение:



Вычисляются оператор и операнды.

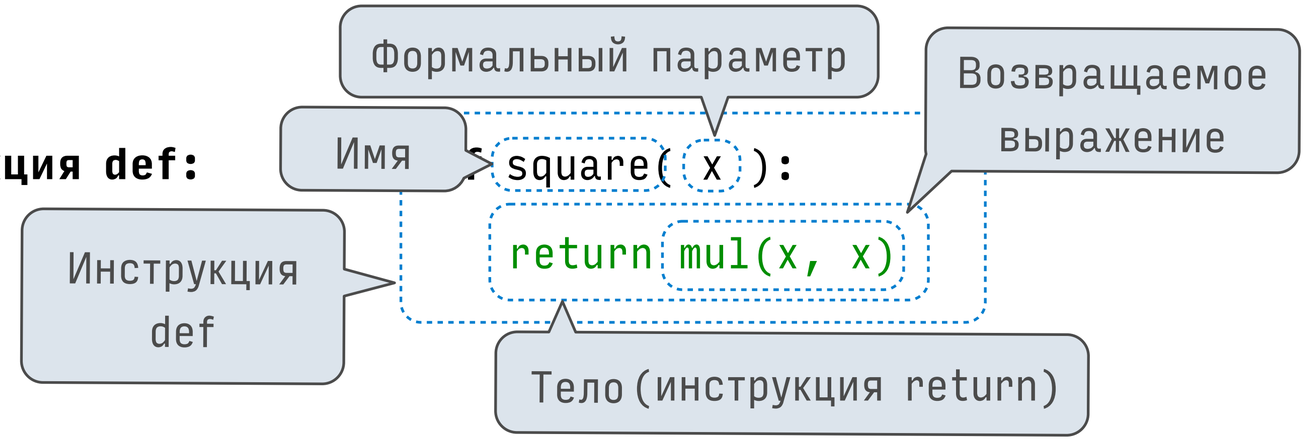
Функция (значение оператора) вызывается с аргументами (значения операндов).

Вызов/Применение:



Жизненный цикл пользовательской функции

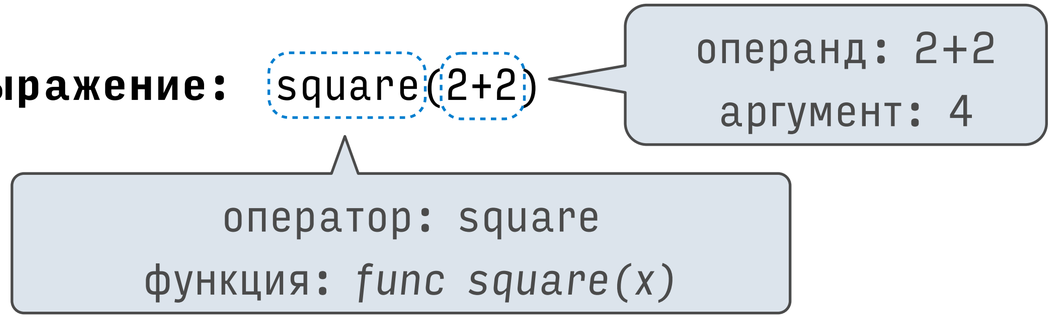
Инструкция def:



Создается новая функция!

В текущем фрейме имя связывается с функцией.

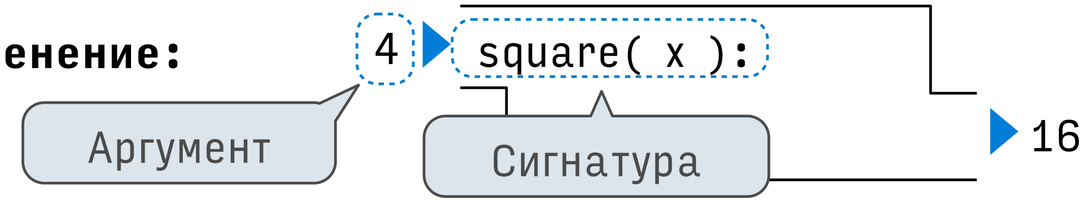
Вызывающее выражение:



Вычисляются оператор и операнды.

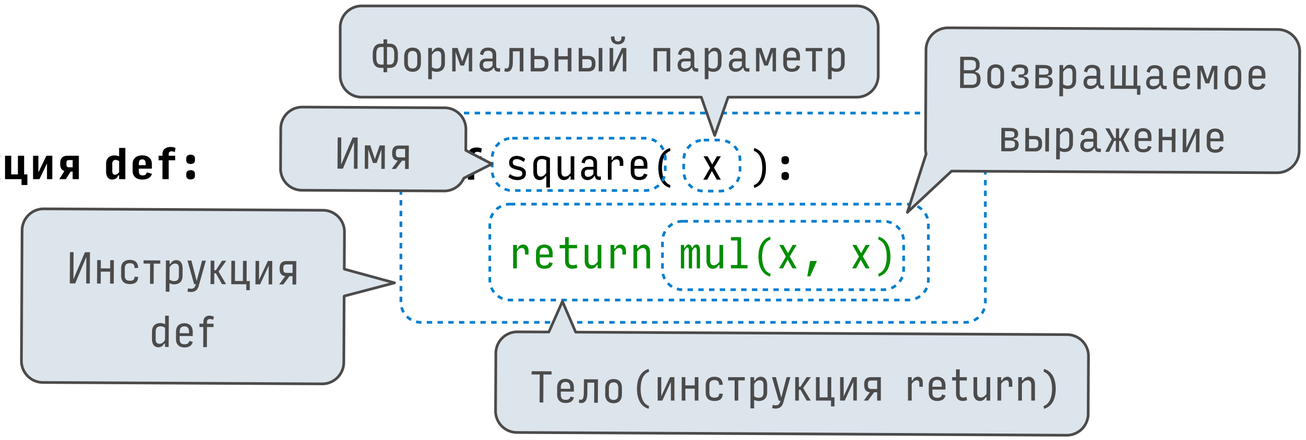
Функция (значение оператора) вызывается с аргументами (значения операндов).

Вызов/Применение:



Жизненный цикл пользовательской функции

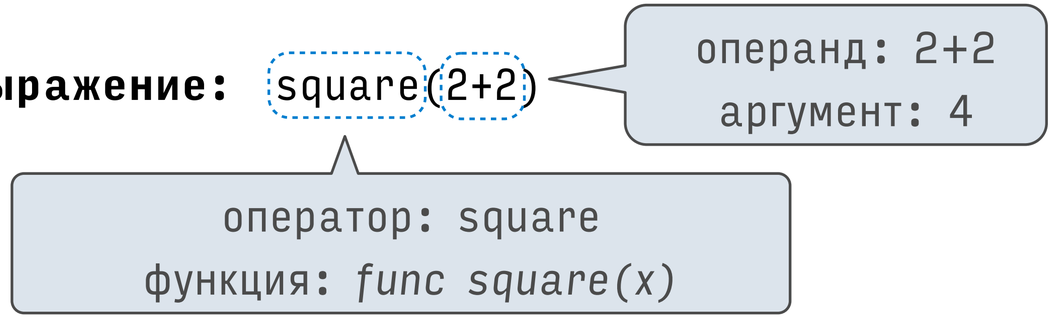
Инструкция def:



Создается новая функция!

В текущем фрейме имя связывается с функцией.

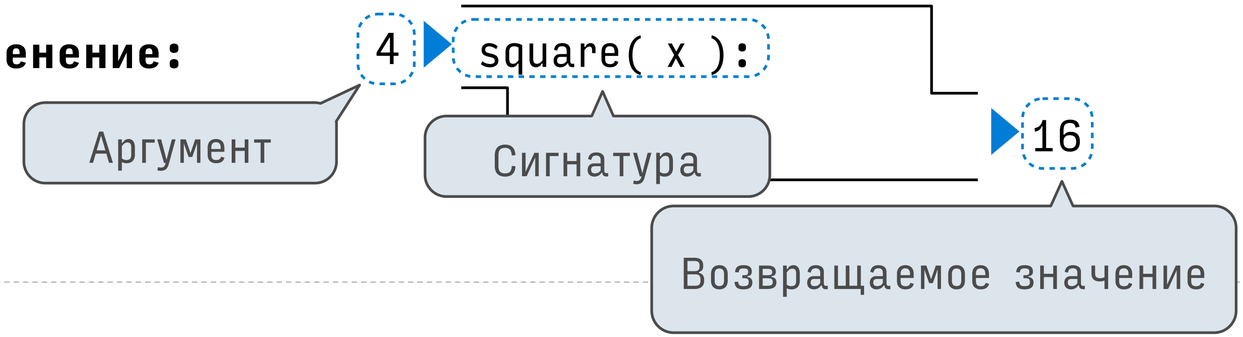
Вызывающее выражение:



Вычисляются оператор и операнды.

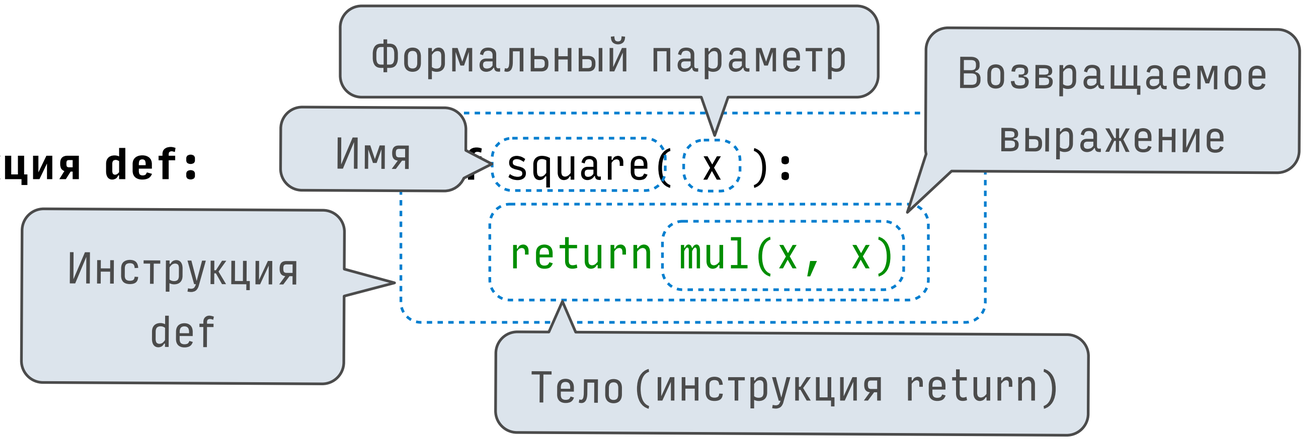
Функция (значение оператора) вызывается с аргументами (значения операндов).

Вызов/Применение:



Жизненный цикл пользовательской функции

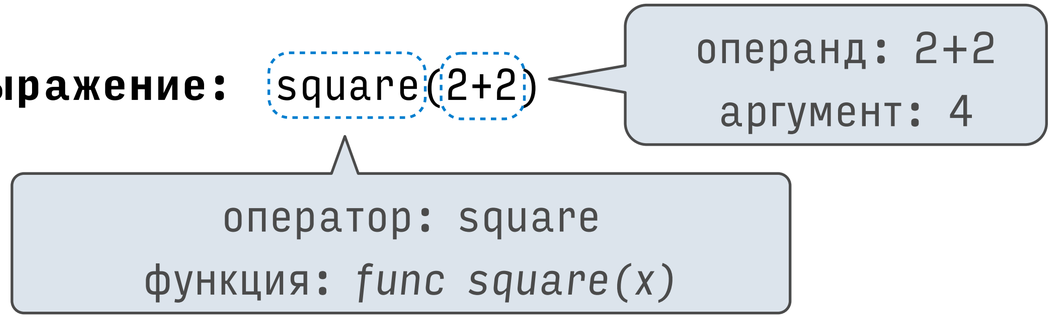
Инструкция def:



Создается новая функция!

В текущем фрейме имя связывается с функцией.

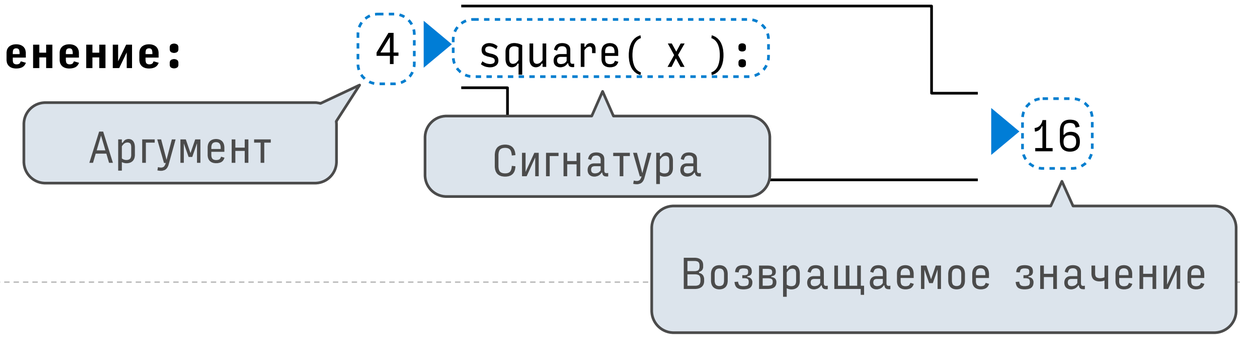
Вызывающее выражение:



Вычисляются оператор и операнды.

Функция (значение оператора) вызывается с аргументами (значения операндов).

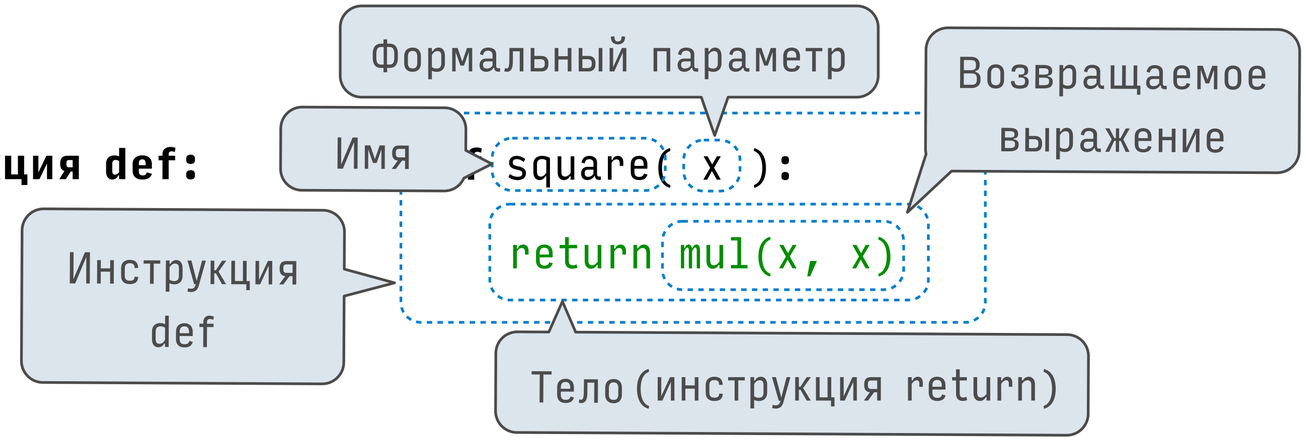
Вызов/Применение:



Создается новый фрейм!

Жизненный цикл пользовательской функции

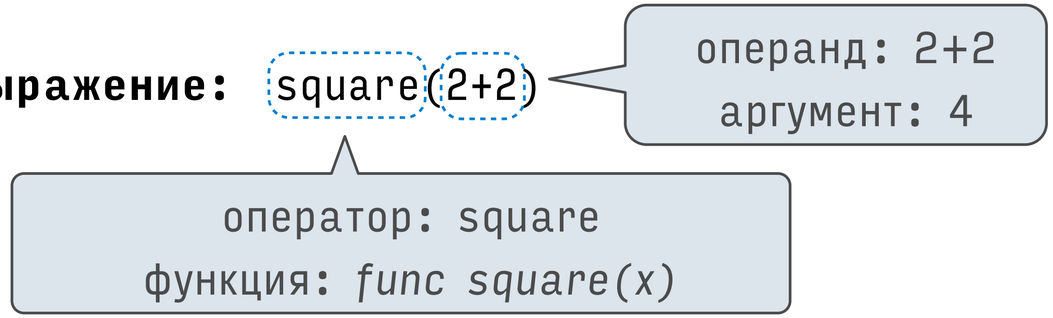
Инструкция def:



Создается новая функция!

В текущем фрейме имя связывается с функцией.

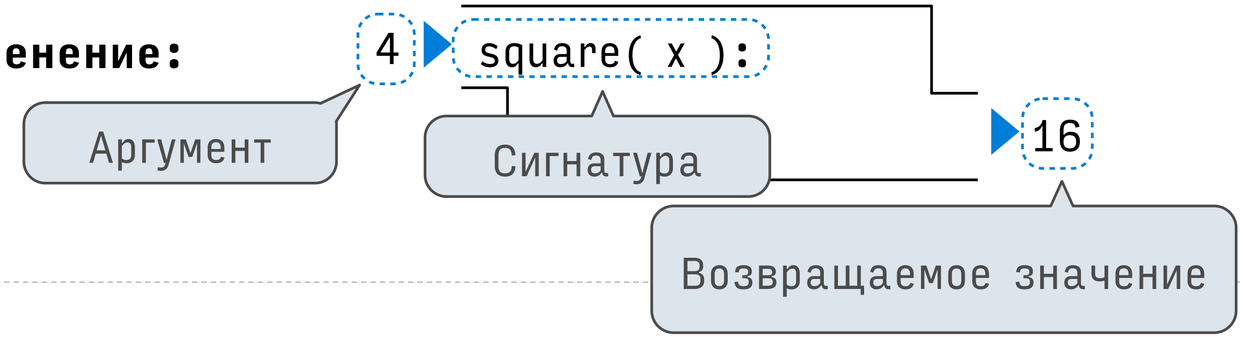
Вызывающее выражение:



Вычисляются оператор и операнды.

Функция (значение оператора) вызывается с аргументами (значения операндов).

Вызов/Применение:

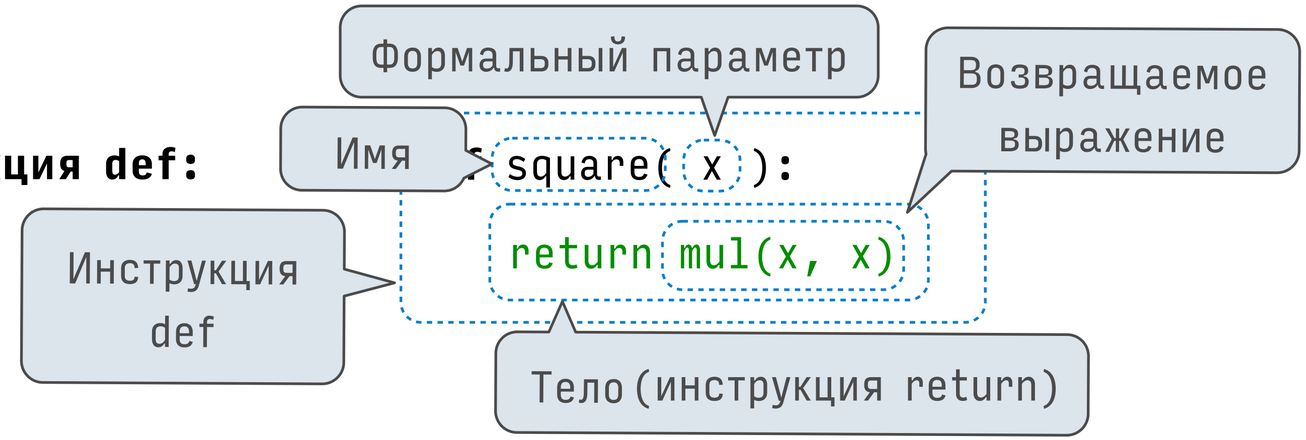


Создается новый фрейм!

Имена параметров связываются с аргументами.

Жизненный цикл пользовательской функции

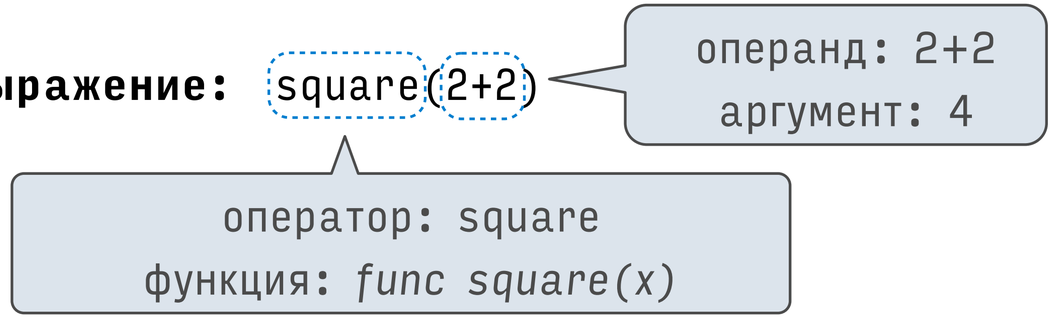
Инструкция def:



Создается новая функция!

В текущем фрейме имя связывается с функцией.

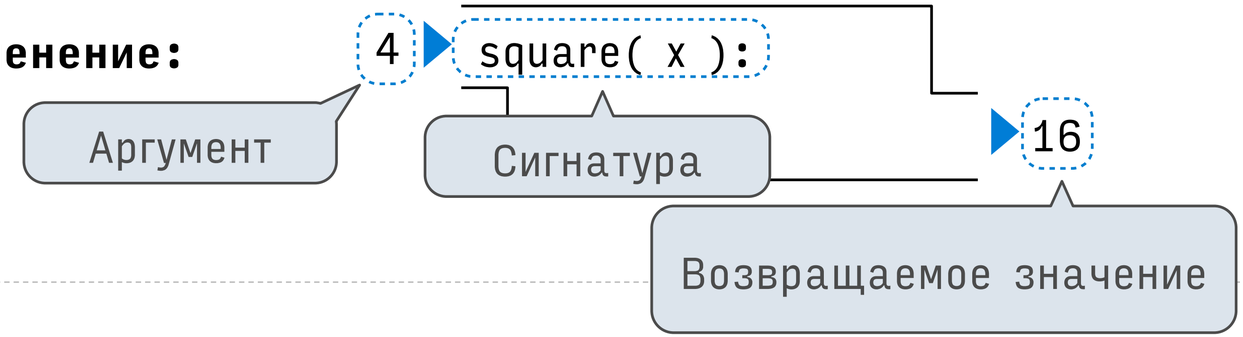
Вызывающее выражение:



Вычисляются оператор и операнды.

Функция (значение оператора) вызывается с аргументами (значения операндов).

Вызов/Применение:



Создается новый фрейм!

Имена параметров связываются с аргументами.

В этом новом окружении исполняется тело функции.

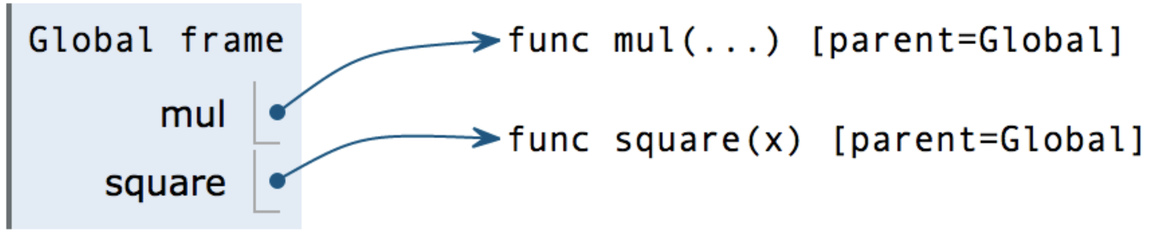
Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```

Интерактивная диаграмма

Несколько окружений на одной диаграмме!

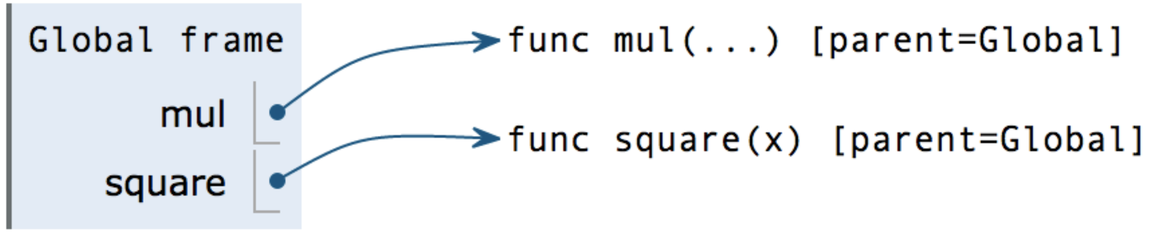
```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```



Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

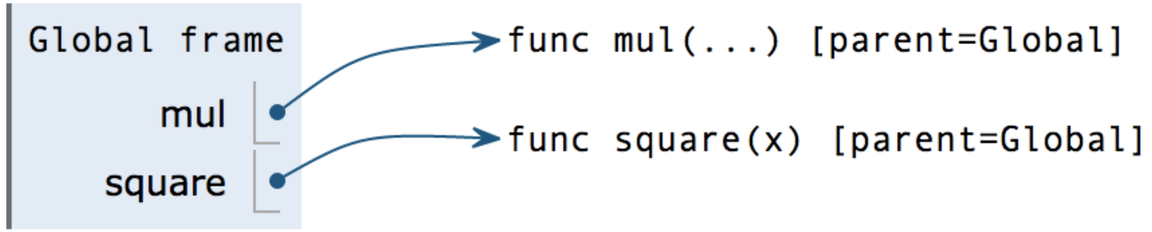


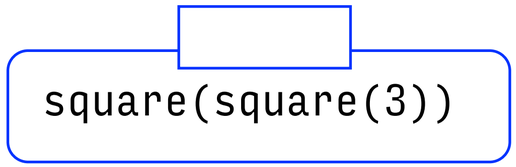

```
square(square(3))
```

Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```

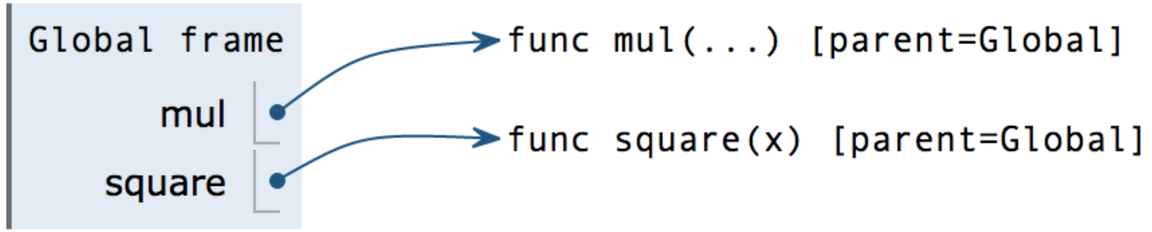


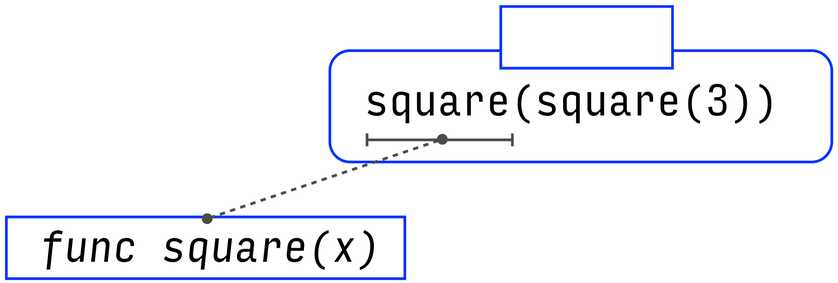


Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

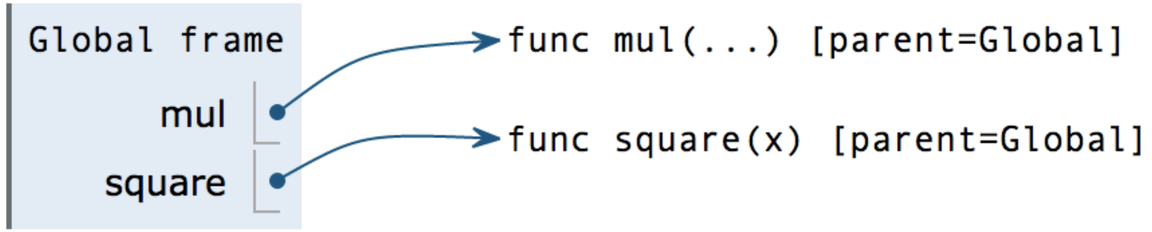


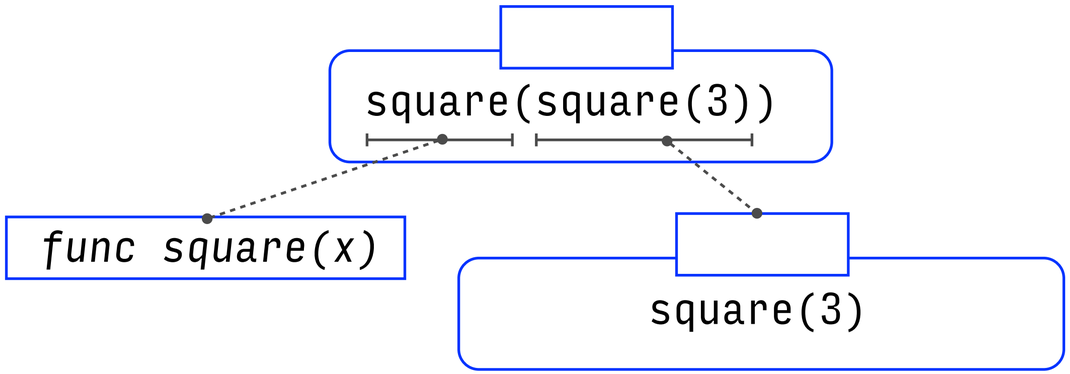


Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

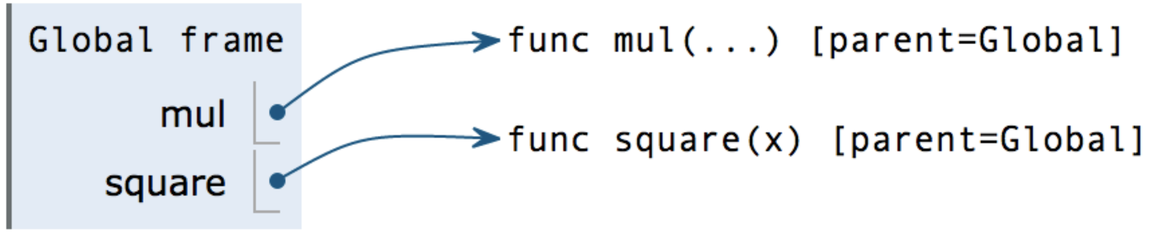


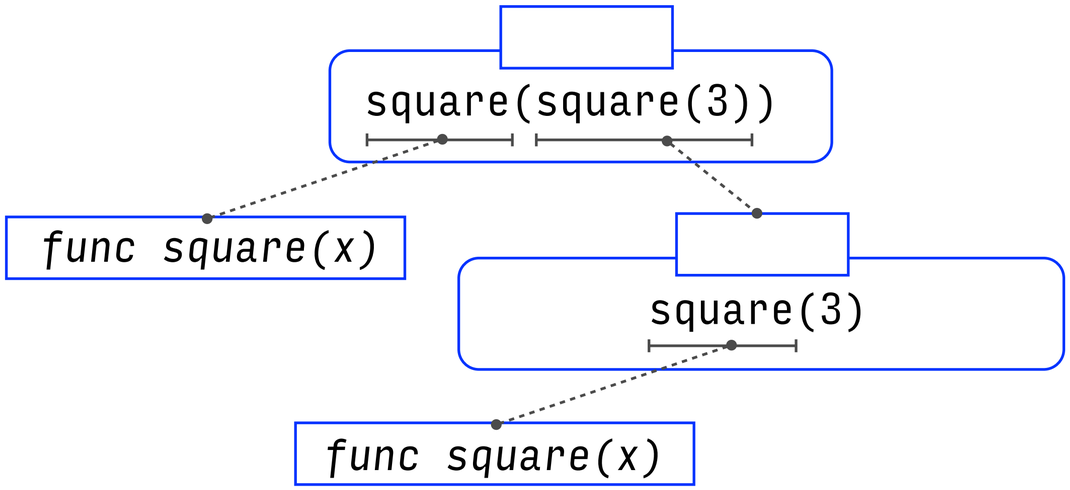


Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

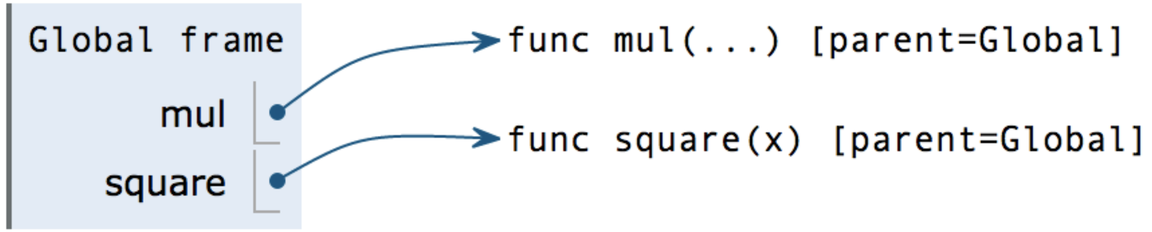


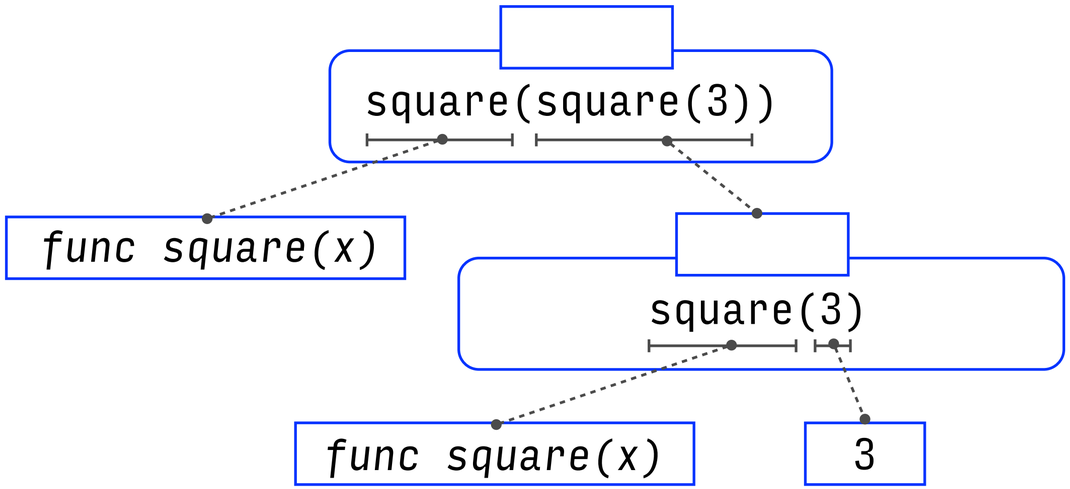


Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

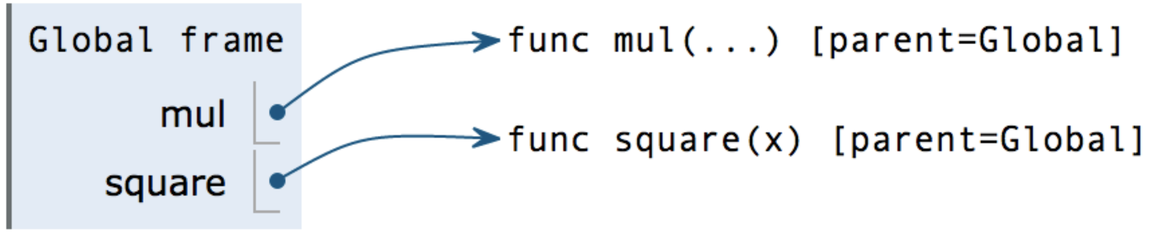


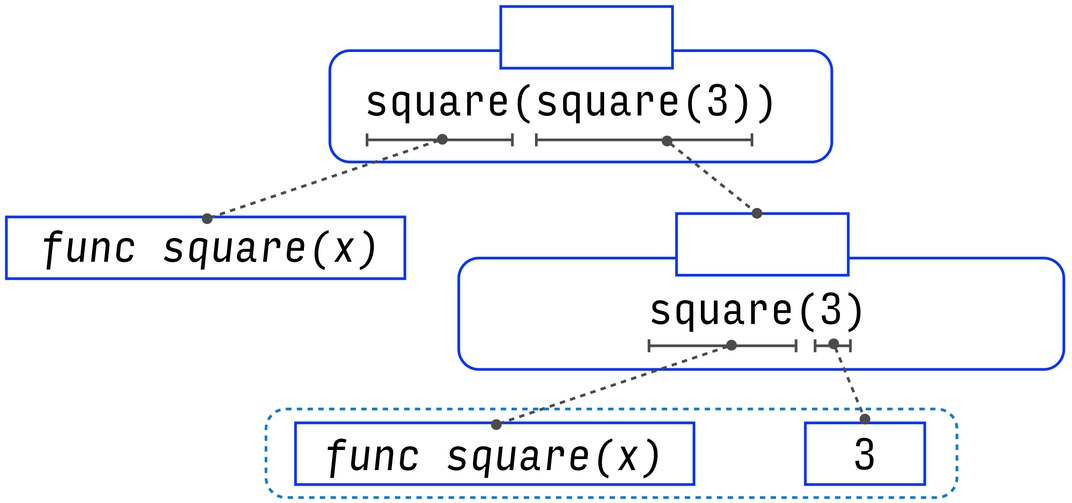


Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

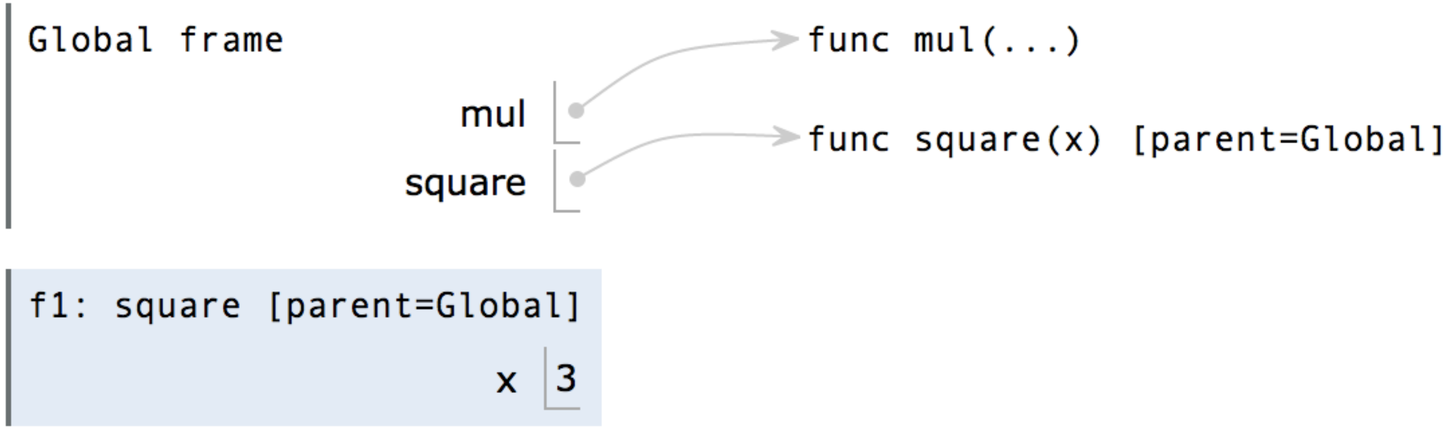


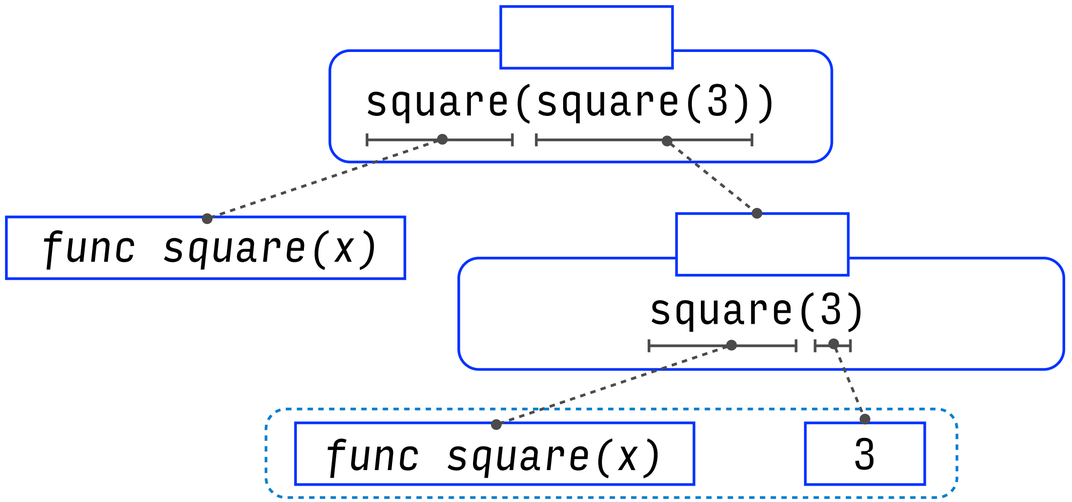


Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```

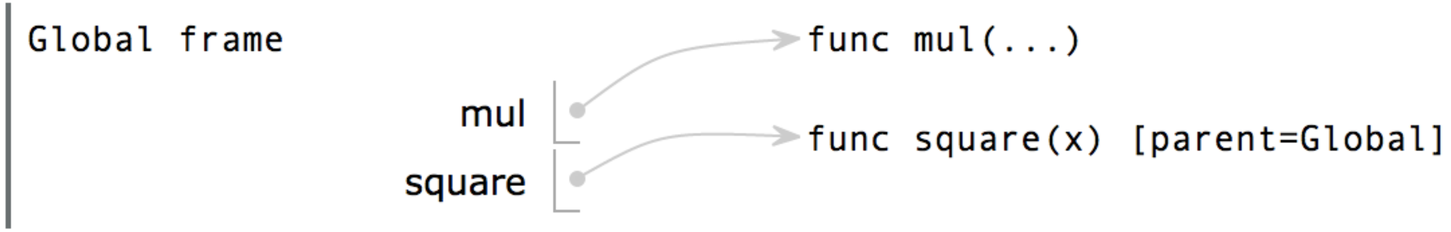


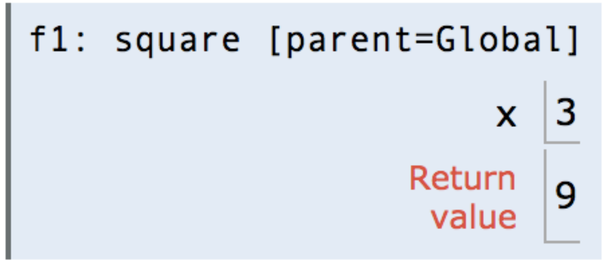


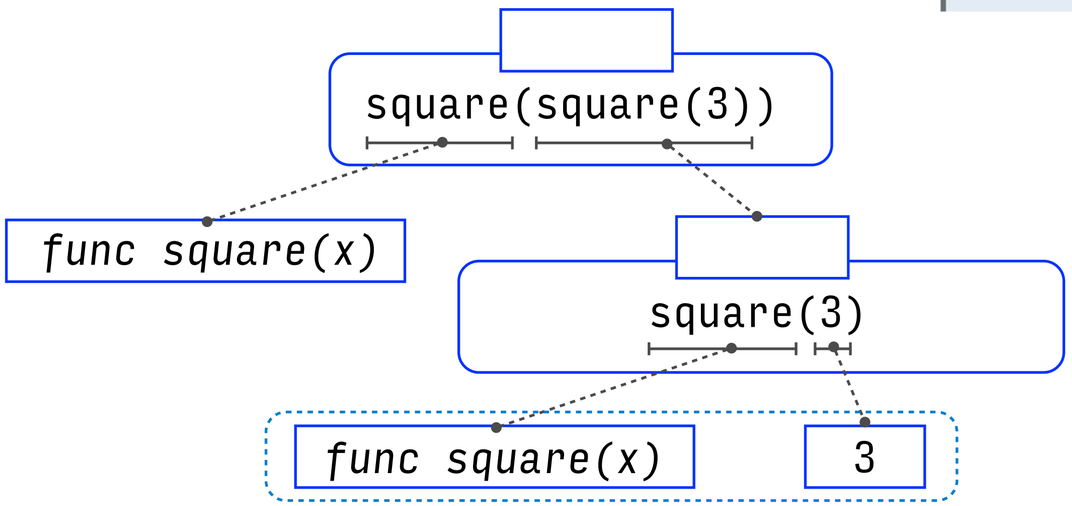
Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```



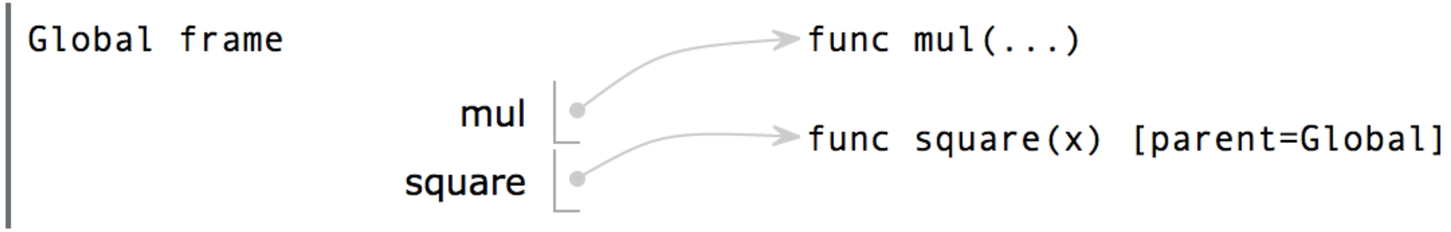


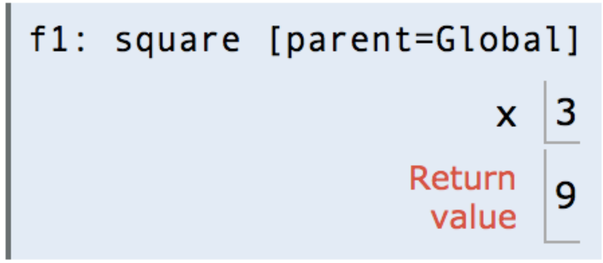


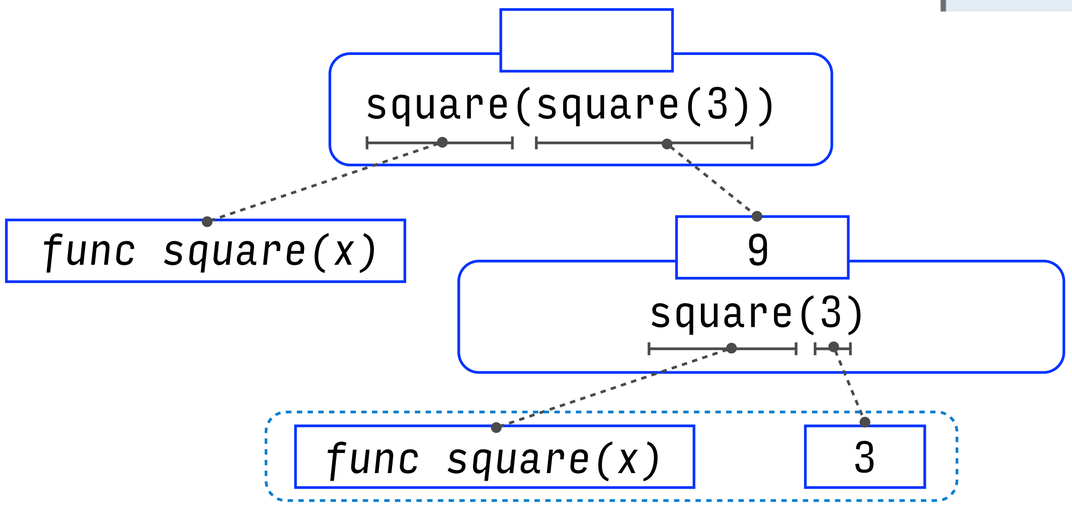
Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```



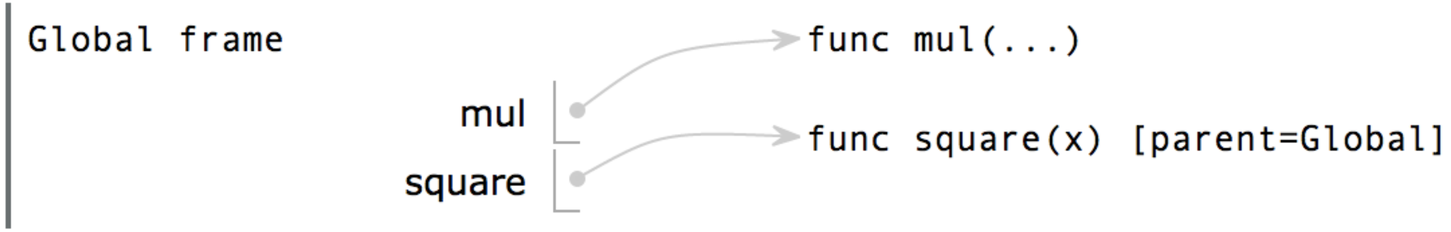


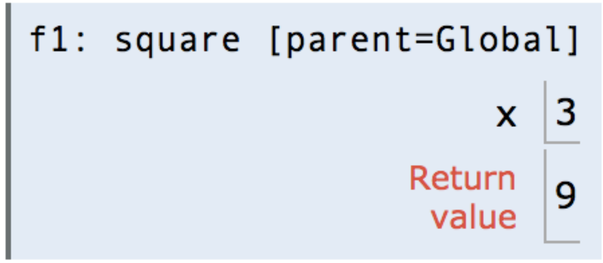


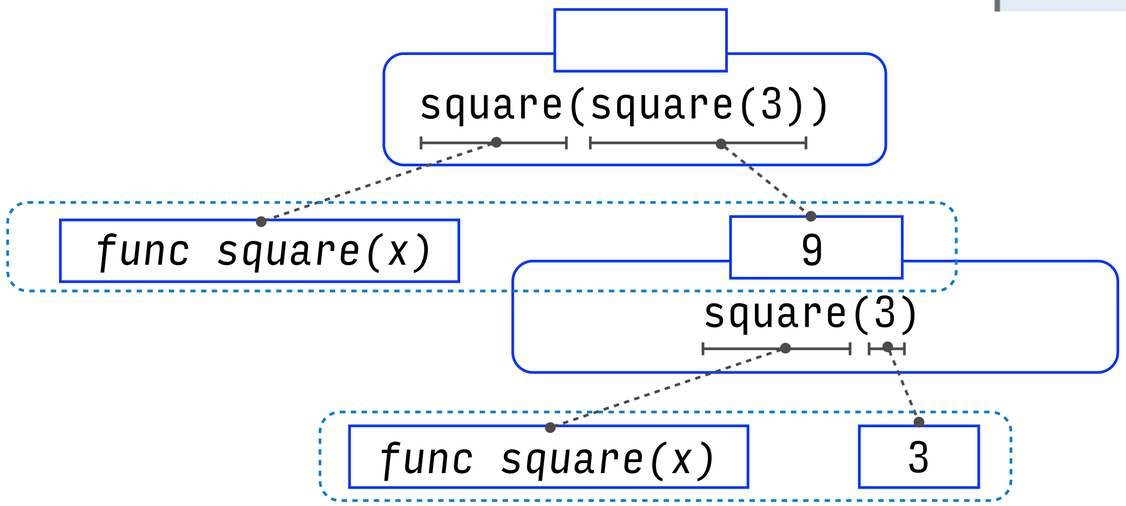
Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```



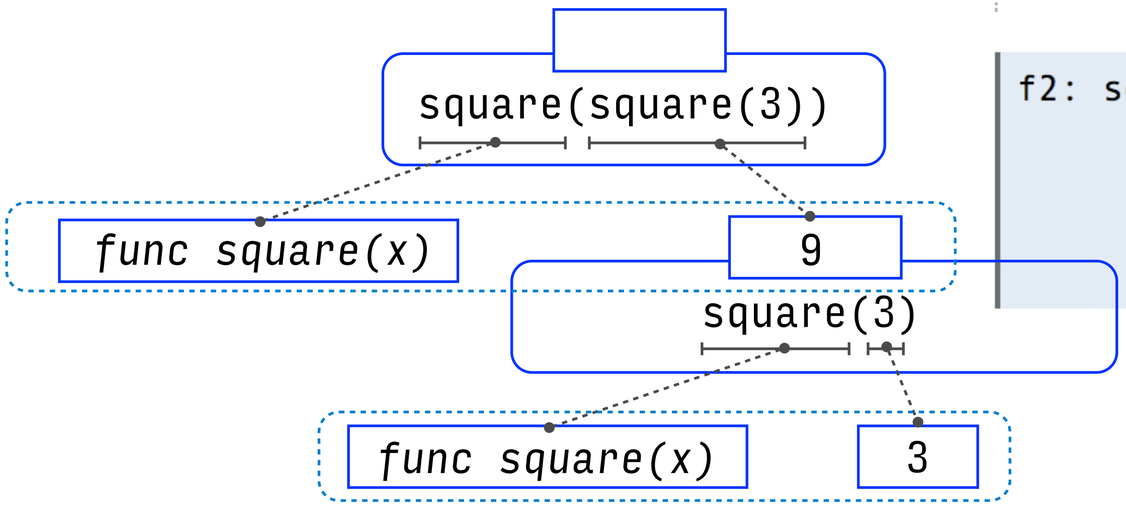


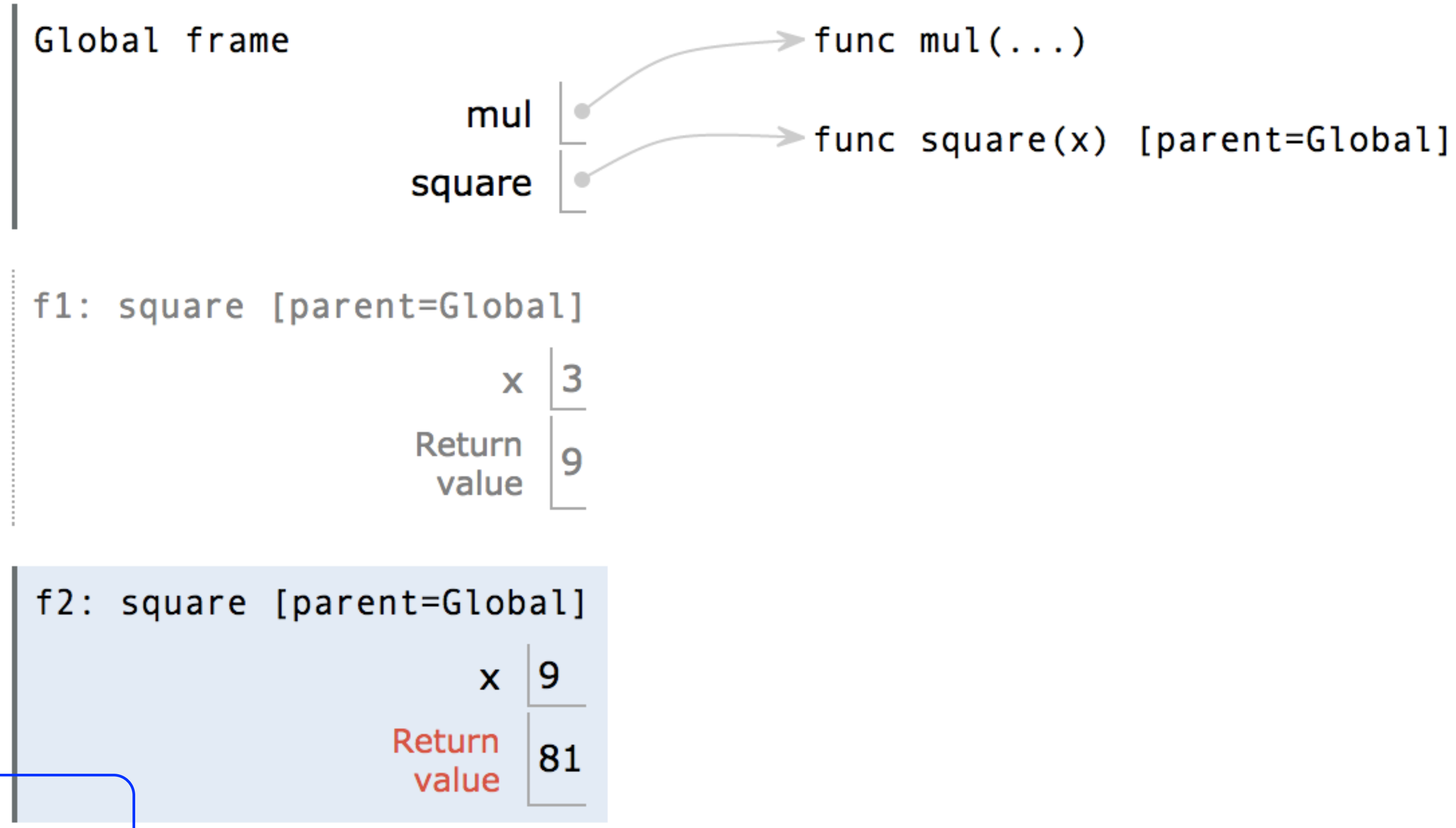


Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```

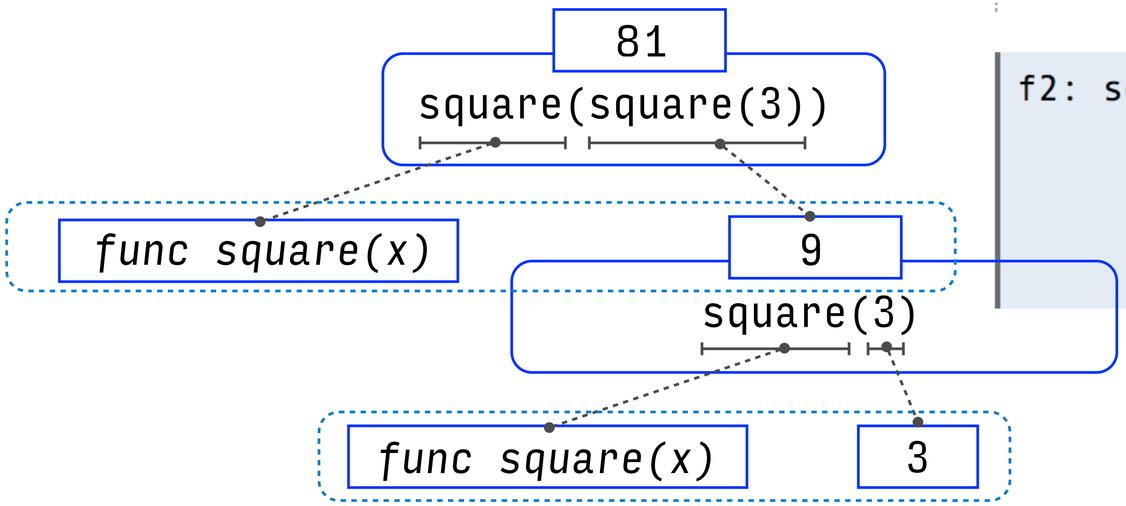


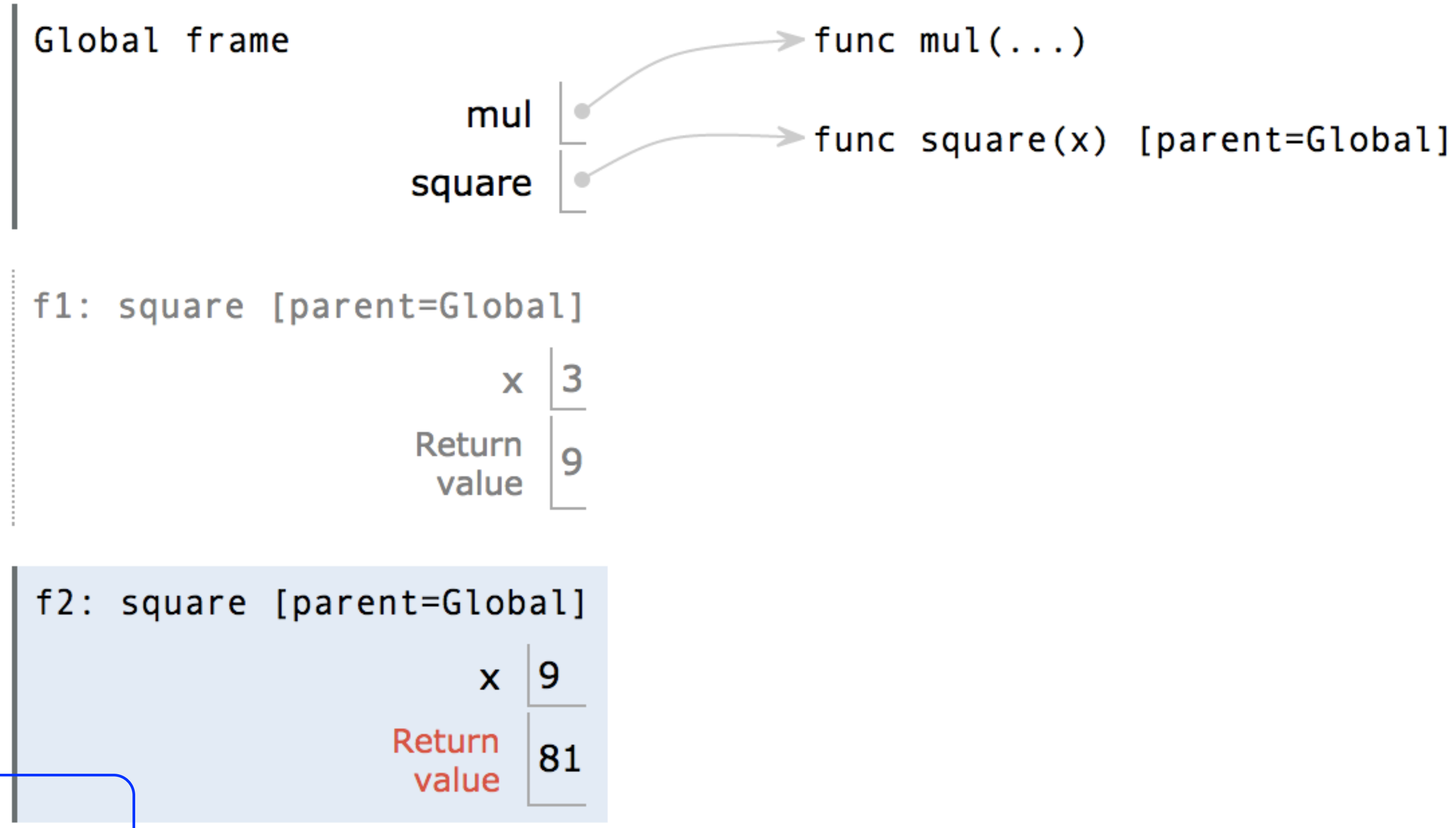


Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```

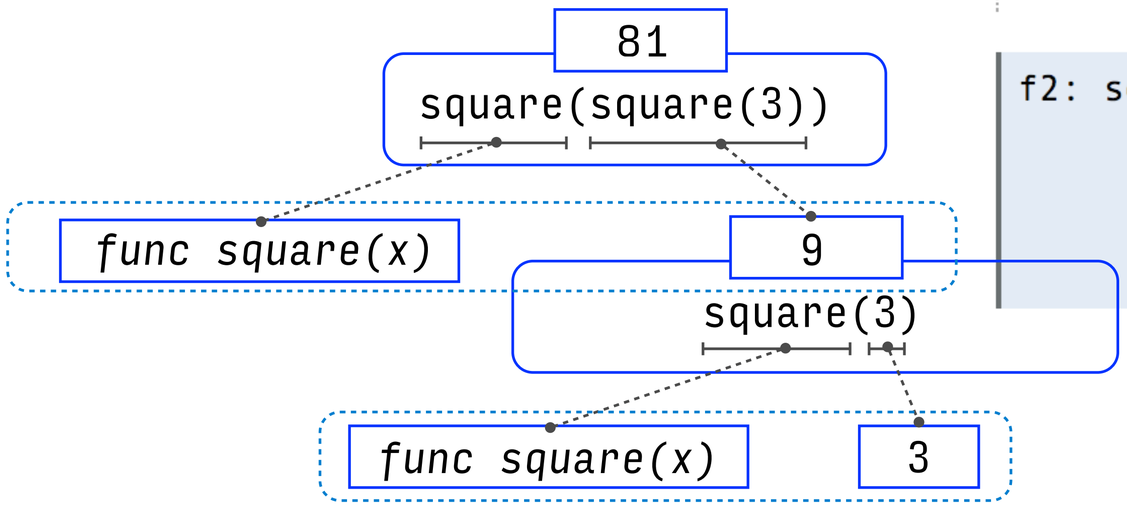


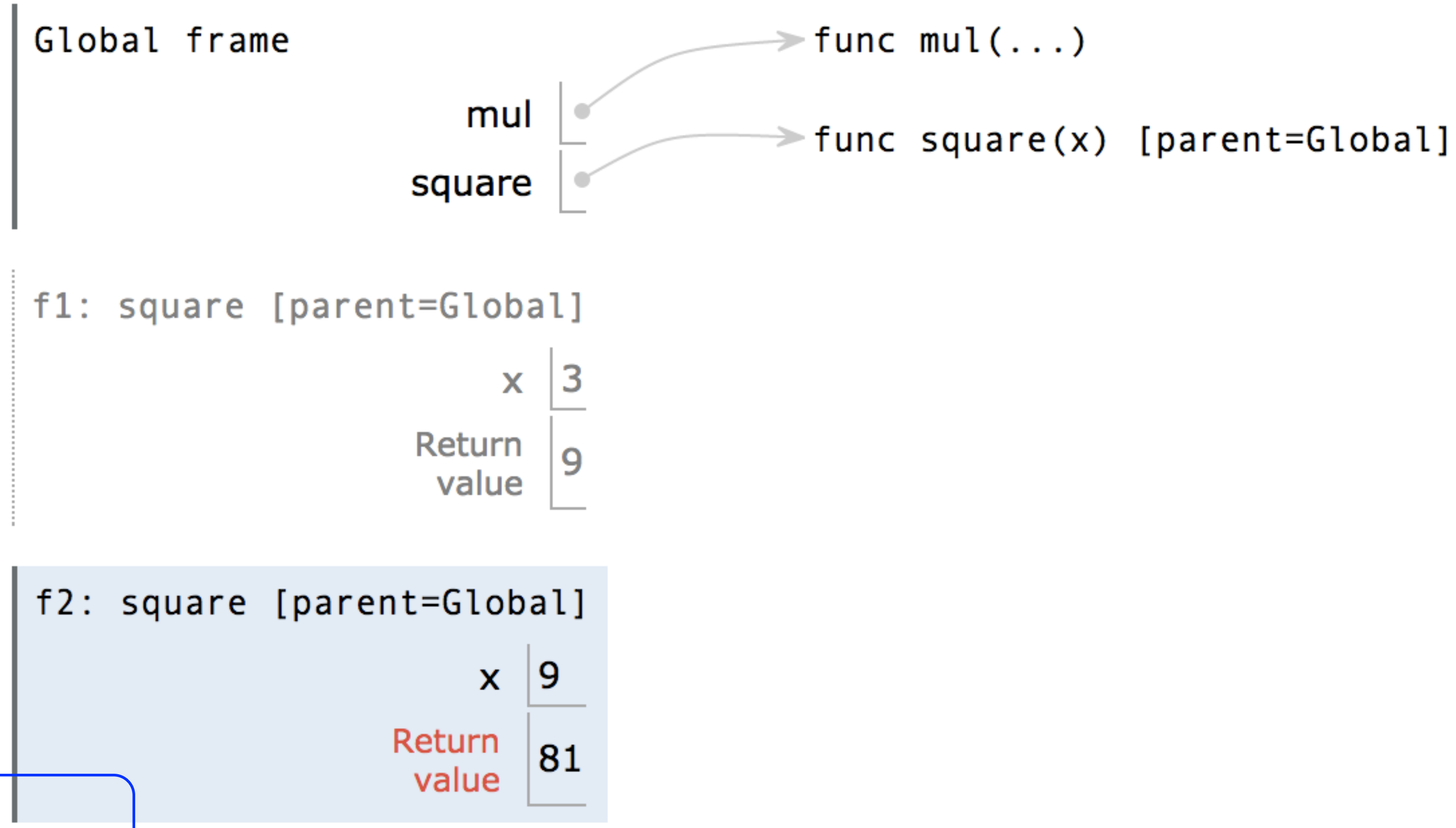


Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```

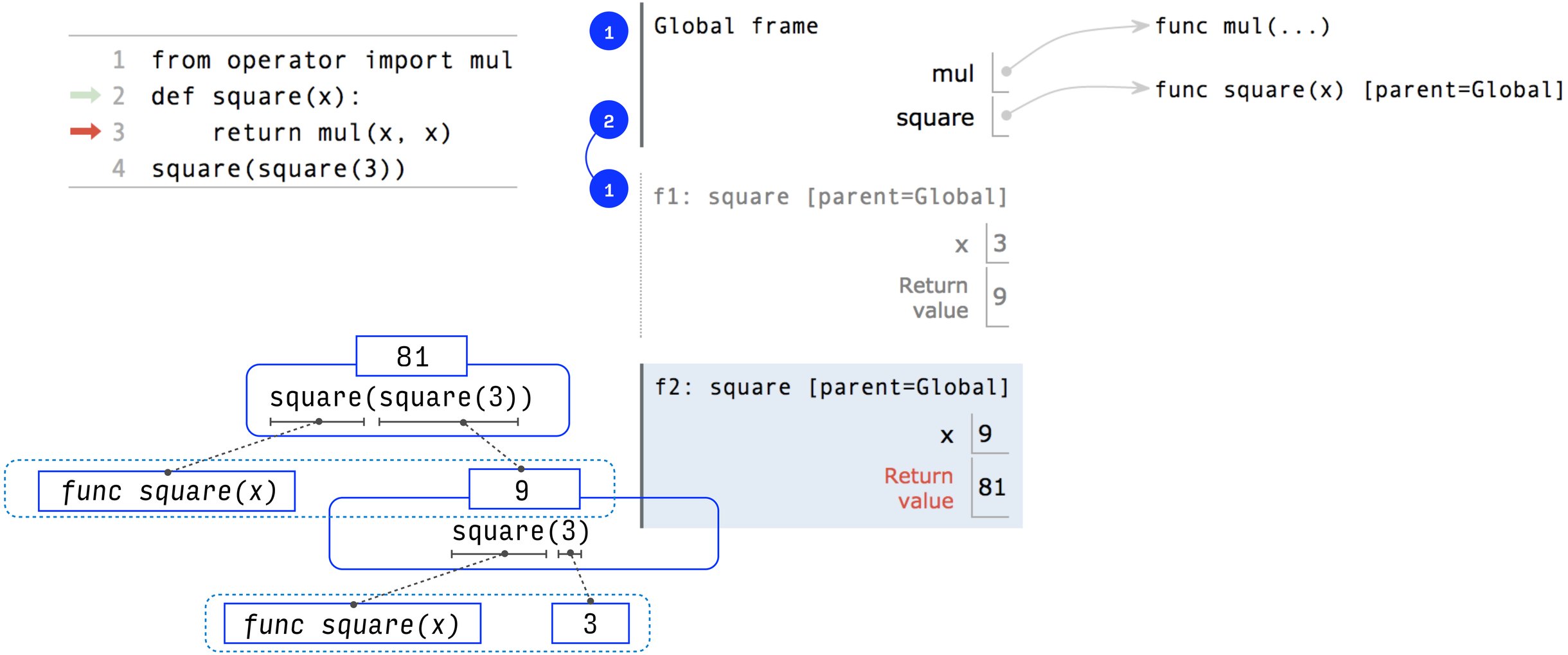




Интерактивная диаграмма

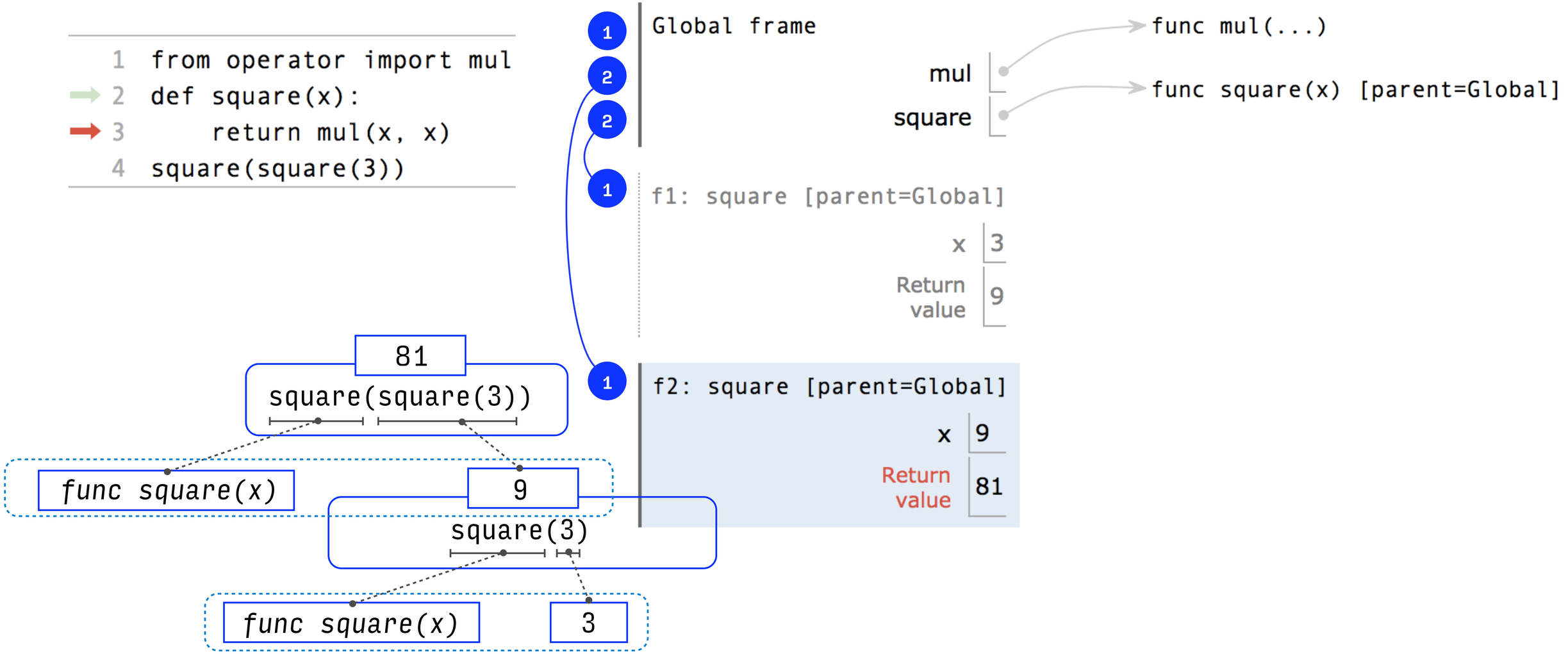
Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```



Интерактивная диаграмма

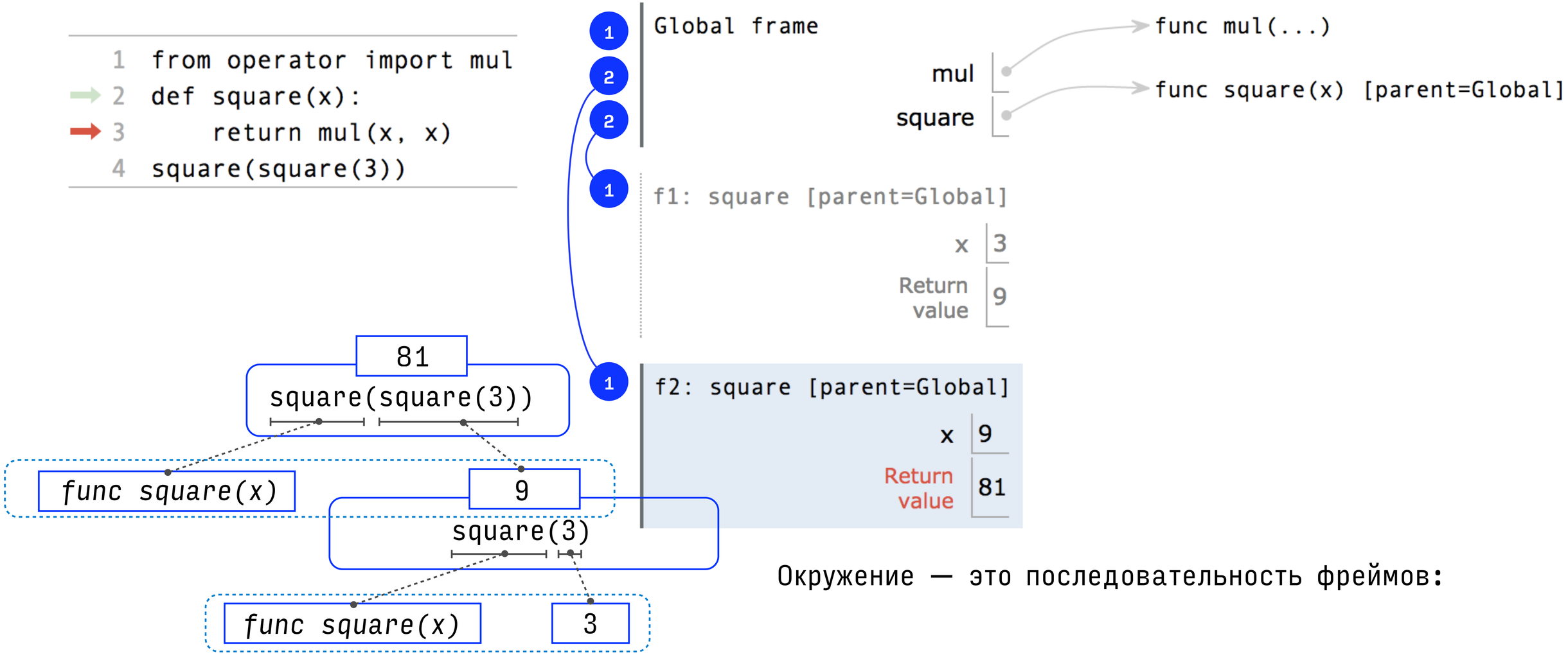
Несколько окружений на одной диаграмме!



Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```

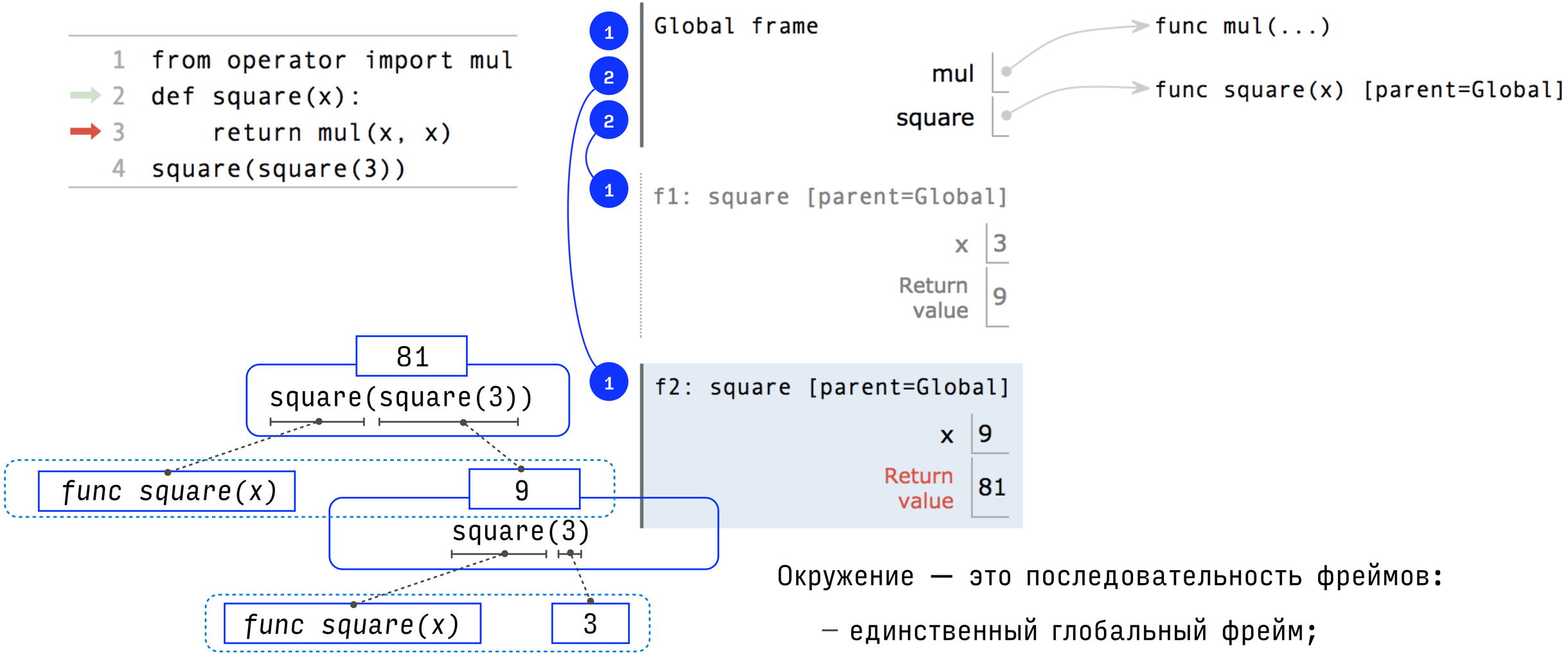


Окружение — это последовательность фреймов:

Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```



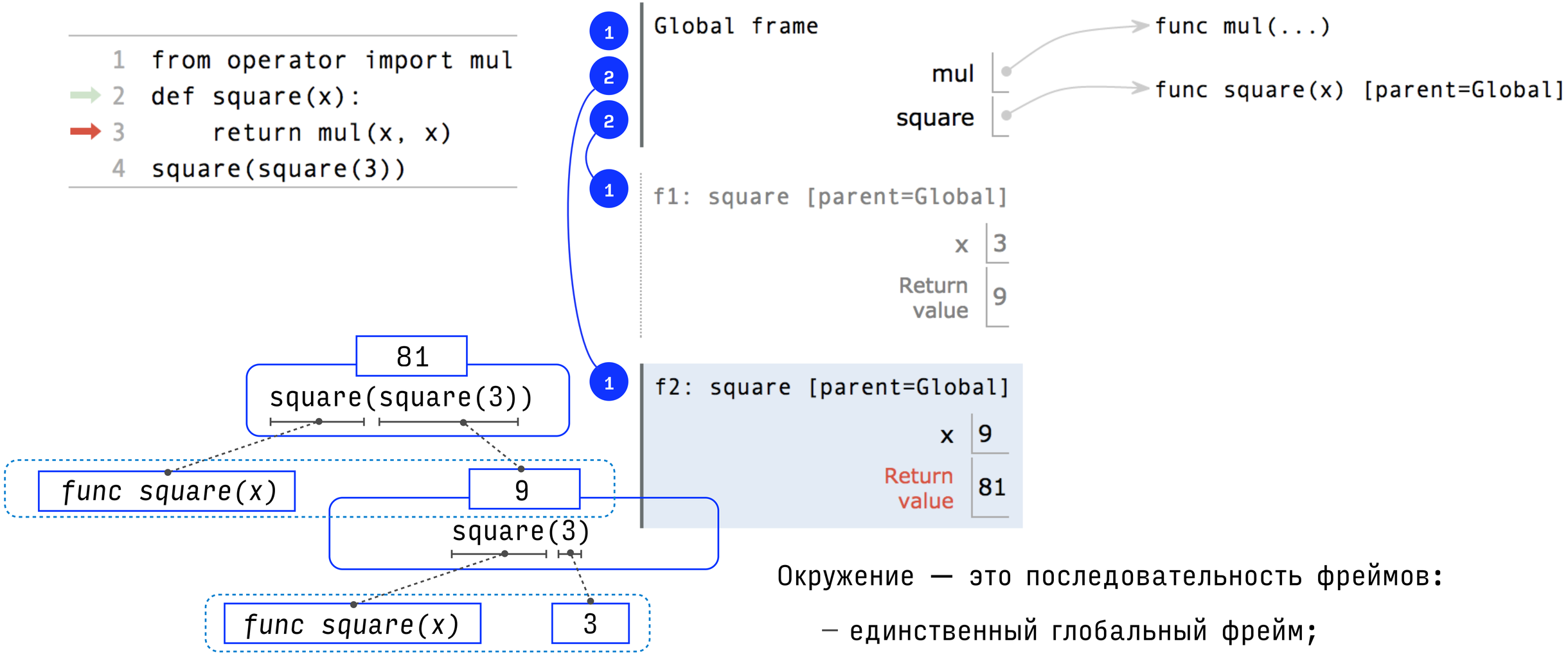
Окружение — это последовательность фреймов:

— единственный глобальный фрейм;

Интерактивная диаграмма

Несколько окружений на одной диаграмме!

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```



Окружение — это последовательность фреймов:

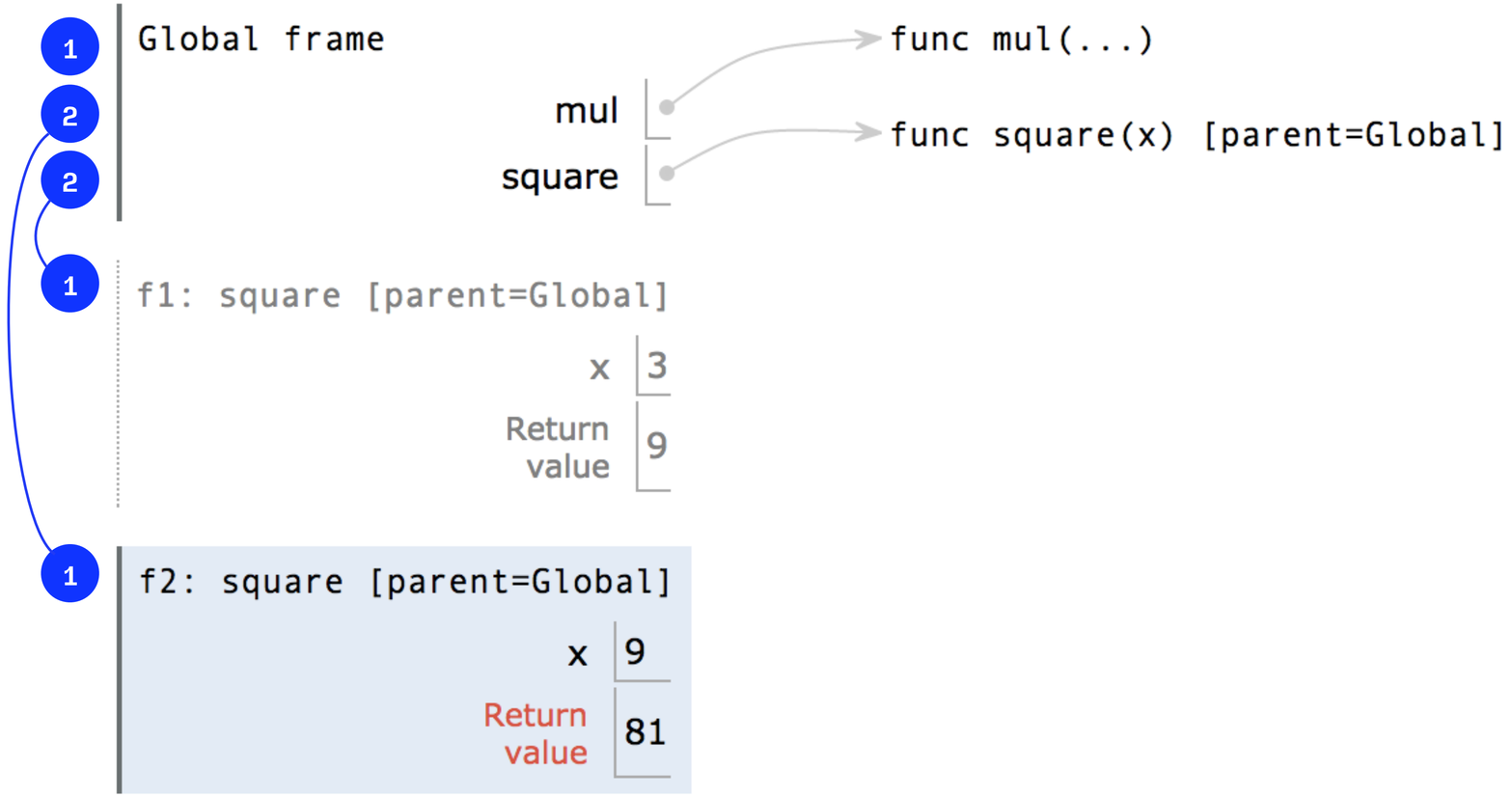
— единственный глобальный фрейм;

— локальный, затем глобальный фреймы.

Интерактивная диаграмма

Имена не имеют смысла без окружений

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```



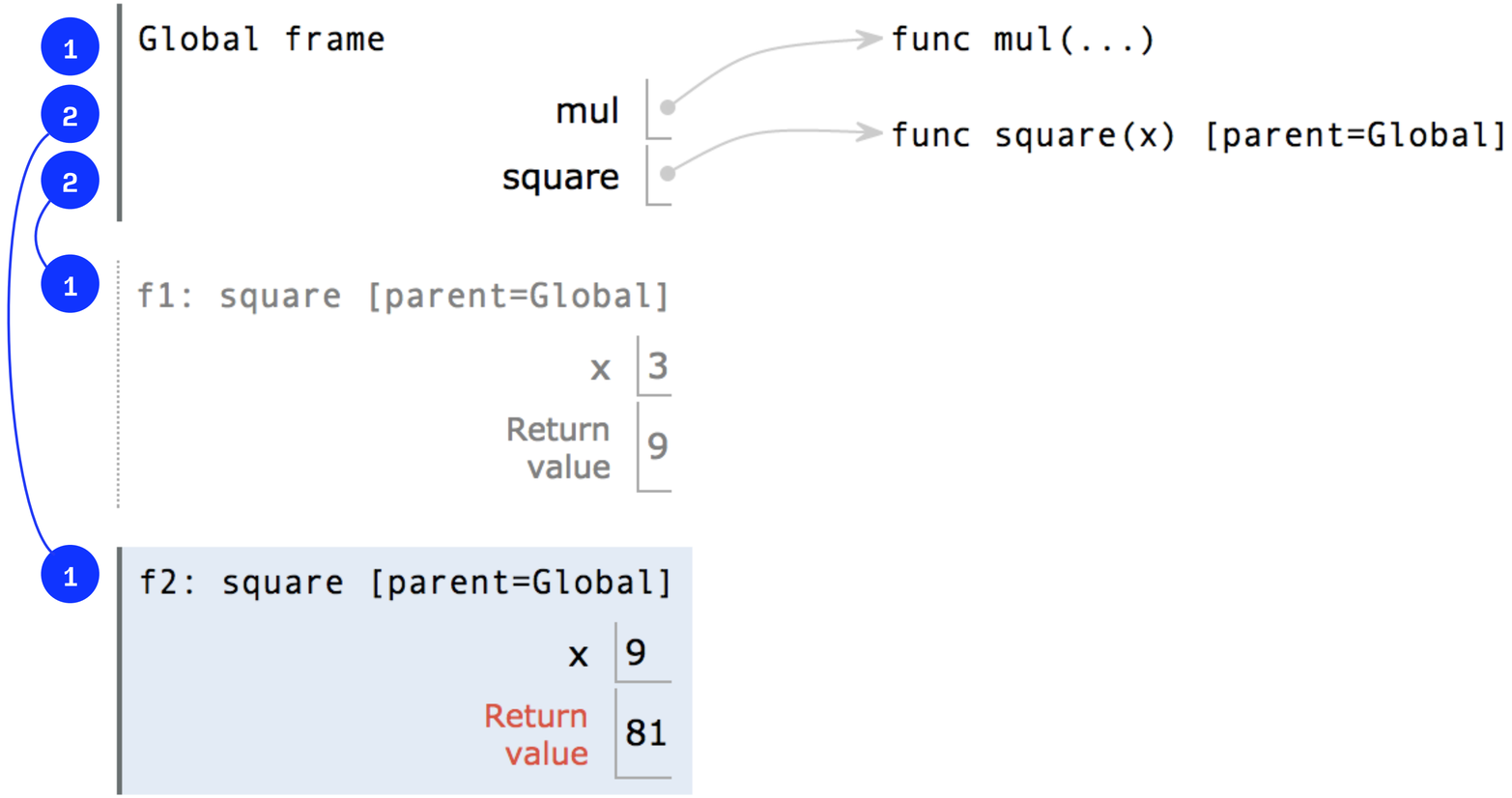
Окружение — это последовательность фреймов:

- единственный глобальный фрейм;
- локальный, затем глобальный фреймы.

Интерактивная диаграмма

Имена не имеют смысла без окружений

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```



Каждое выражение вычисляется в контексте окружения.

Окружение — это последовательность фреймов:

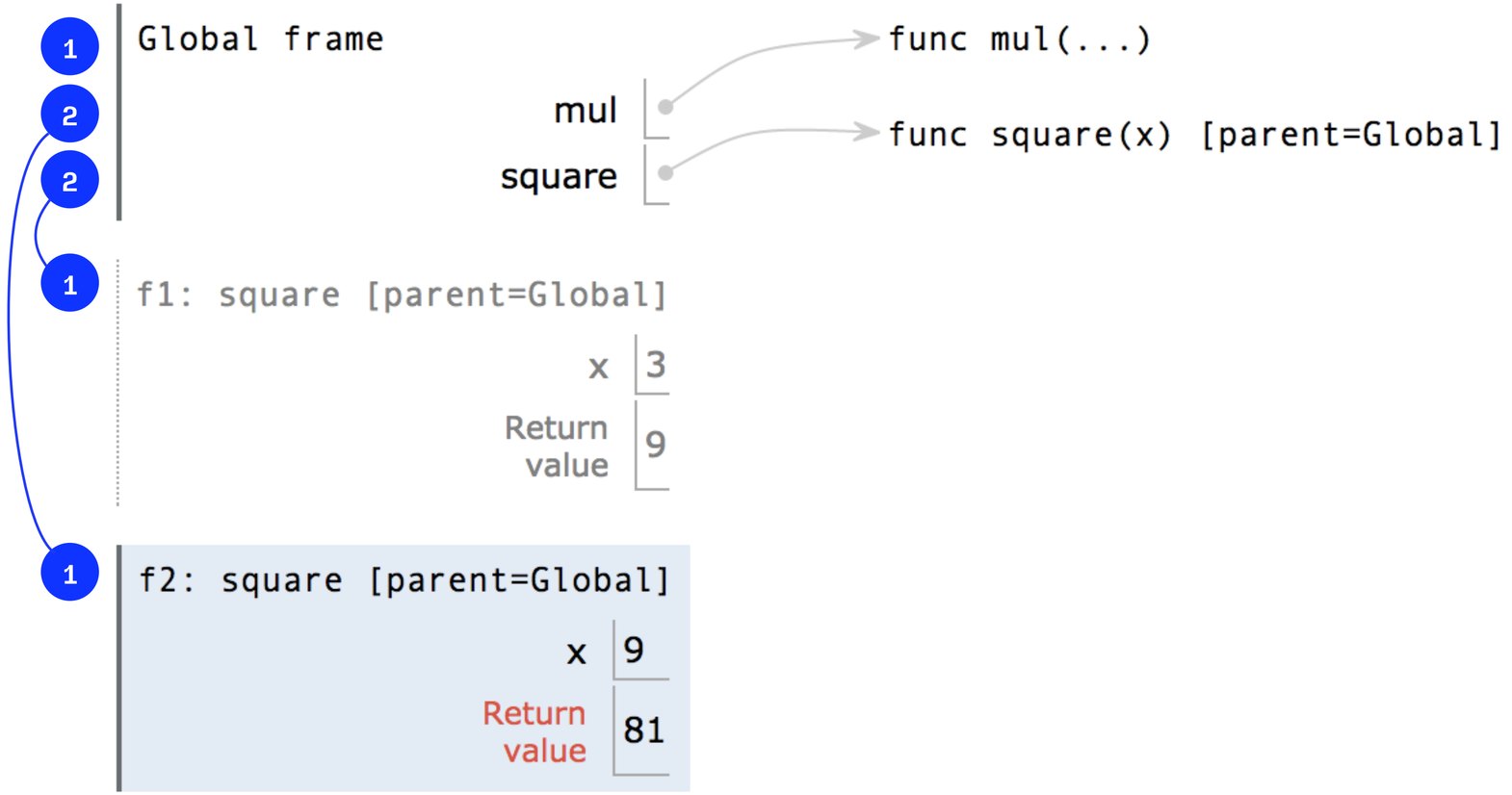
— единственный глобальный фрейм;

— локальный, затем глобальный фреймы.

Интерактивная диаграмма

Имена не имеют смысла без окружений

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```



Каждое выражение вычисляется в контексте окружения.

Значение имени в текущем окружении берётся из первого фрейма, в котором это имя присутствует.

Окружение — это последовательность фреймов:

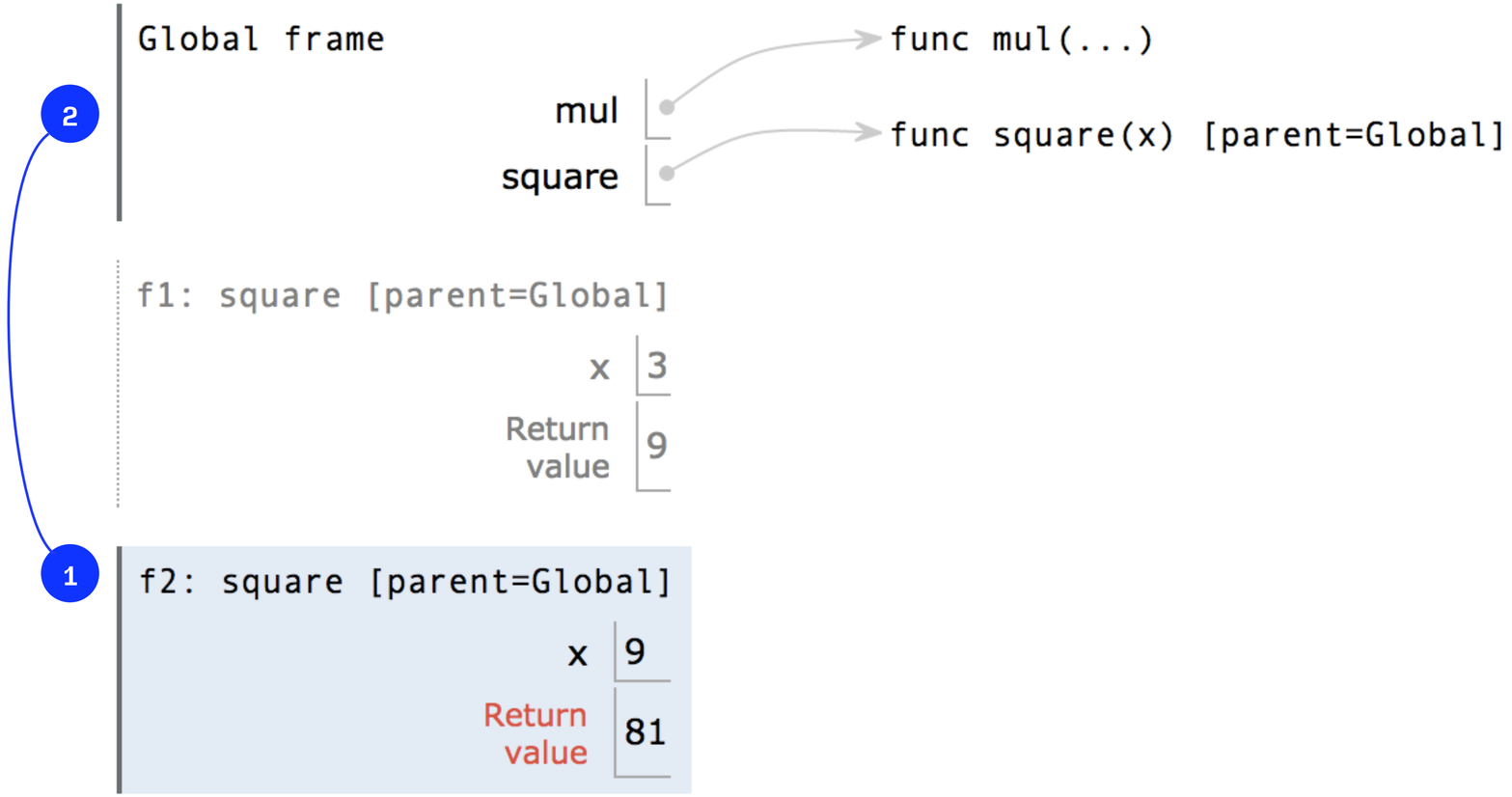
— единственный глобальный фрейм;

— локальный, затем глобальный фреймы.

Интерактивная диаграмма

Имена не имеют смысла без окружений

```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```



Каждое выражение вычисляется в контексте окружения.

Значение имени в текущем окружении берётся из первого фрейма, в котором это имя присутствует.

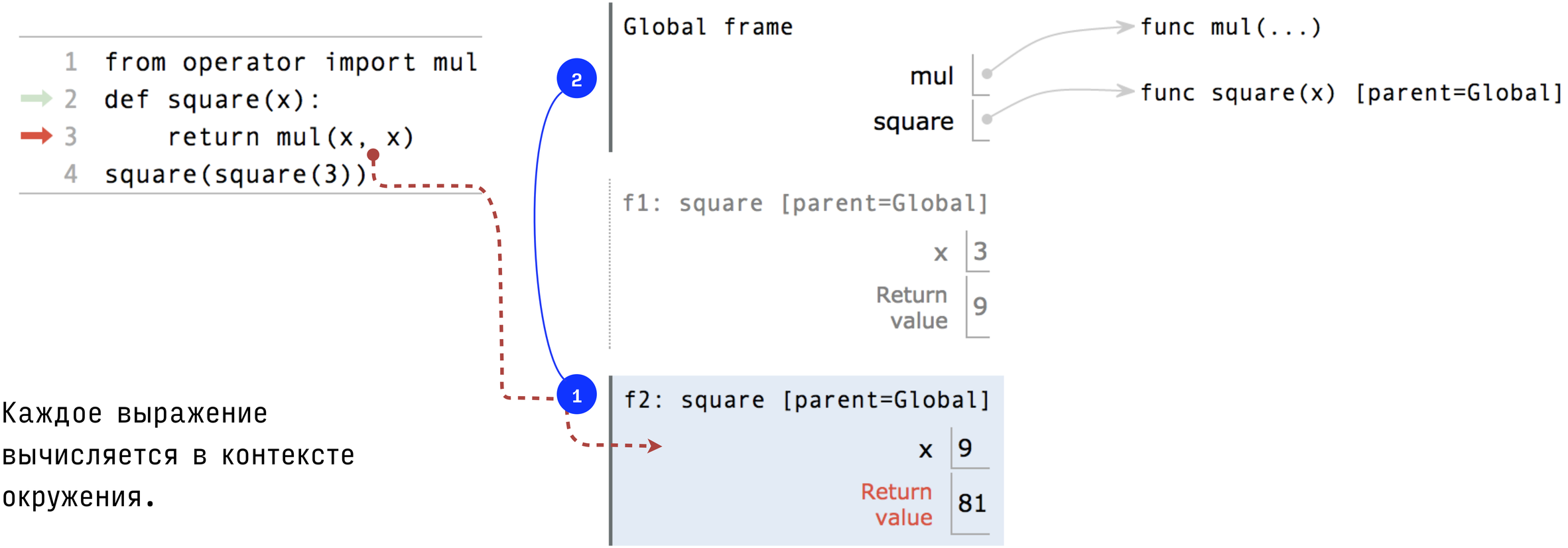
Окружение — это последовательность фреймов:

— единственный глобальный фрейм;

— локальный, затем глобальный фреймы.

Интерактивная диаграмма

Имена не имеют смысла без окружений



Каждое выражение вычисляется в контексте окружения.

Значение имени в текущем окружении берётся из первого фрейма, в котором это имя присутствует.

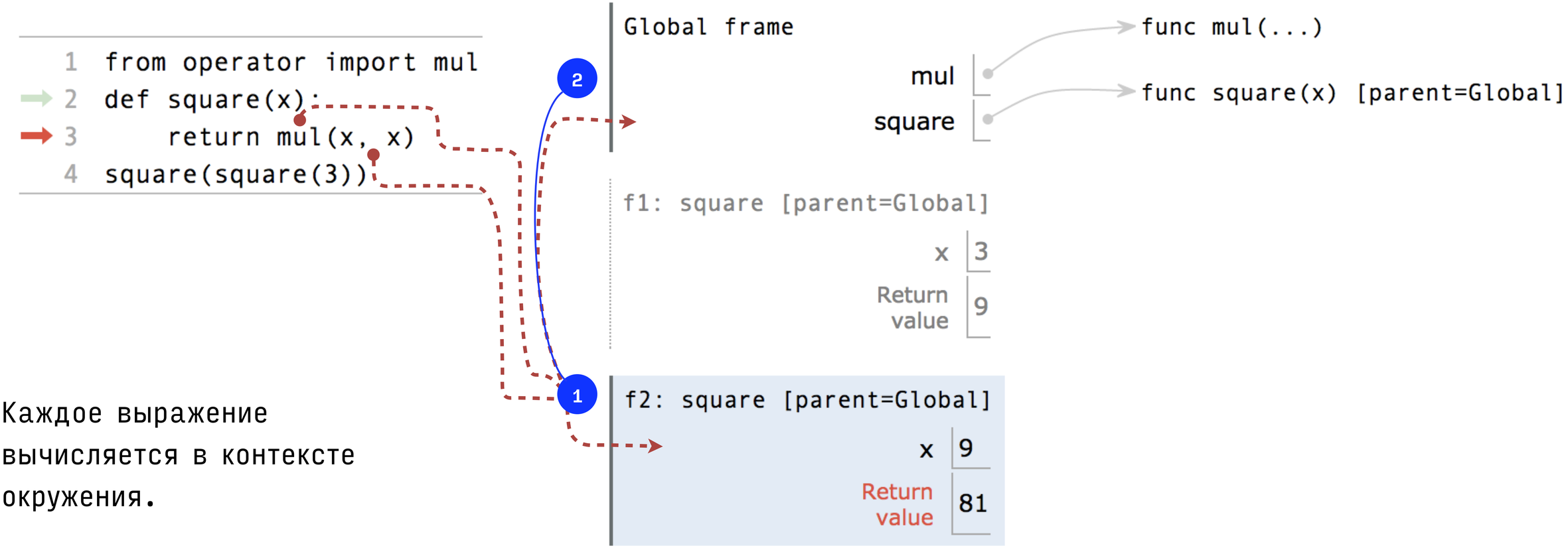
Окружение — это последовательность фреймов:

— единственный глобальный фрейм;

— локальный, затем глобальный фреймы.

Интерактивная диаграмма

Имена не имеют смысла без окружений



Каждое выражение вычисляется в контексте окружения.

Значение имени в текущем окружении берётся из первого фрейма, в котором это имя присутствует.

Окружение — это последовательность фреймов:

— единственный глобальный фрейм;

— локальный, затем глобальный фреймы.

Интерактивная диаграмма

Имена означают разное в разных окружениях

Каждое выражение вычисляется в контексте окружения.

Значение имени в текущем окружении берётся из первого фрейма, в котором это имя присутствует.

Интерактивная диаграмма

Имена означают разное в разных окружениях

Вызывающее выражение и тело функции обрабатываются в разных окружениях.

Каждое выражение вычисляется в контексте окружения.

Значение имени в текущем окружении берётся из первого фрейма, в котором это имя присутствует.

Интерактивная диаграмма

Имена означают разное в разных окружениях

Вызывающее выражение и тело функции обрабатываются в разных окружениях.

```
1 from operator import mul
2 def square(square):
3     return mul(square, square)
4 square(4)
```

Каждое выражение вычисляется в контексте окружения.

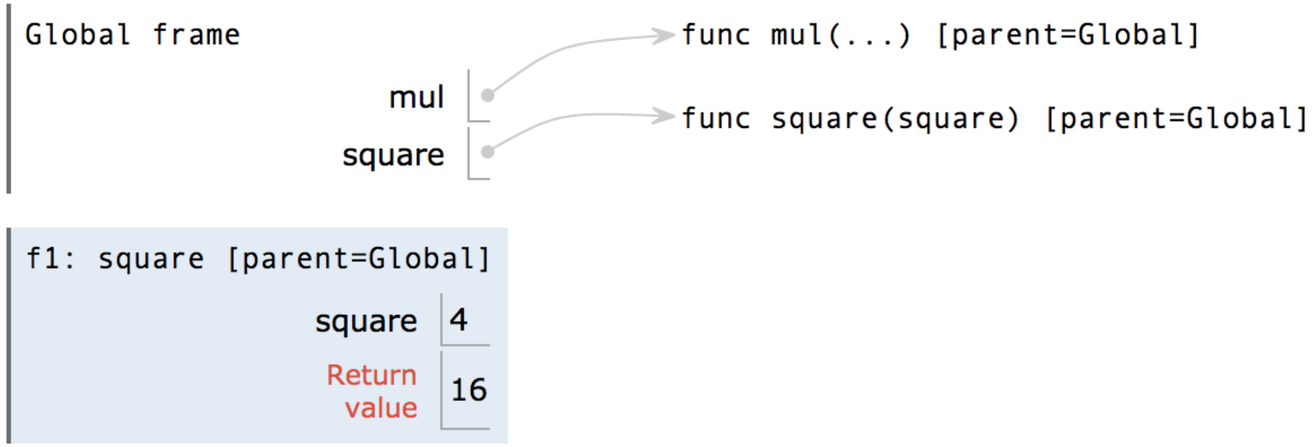
Значение имени в текущем окружении берётся из первого фрейма, в котором это имя присутствует.

Интерактивная диаграмма

Имена означают разное в разных окружениях

Вызывающее выражение и тело функции обрабатываются в разных окружениях.

```
1 from operator import mul
2 def square(square):
3     return mul(square, square)
4 square(4)
```



Каждое выражение вычисляется в контексте окружения.

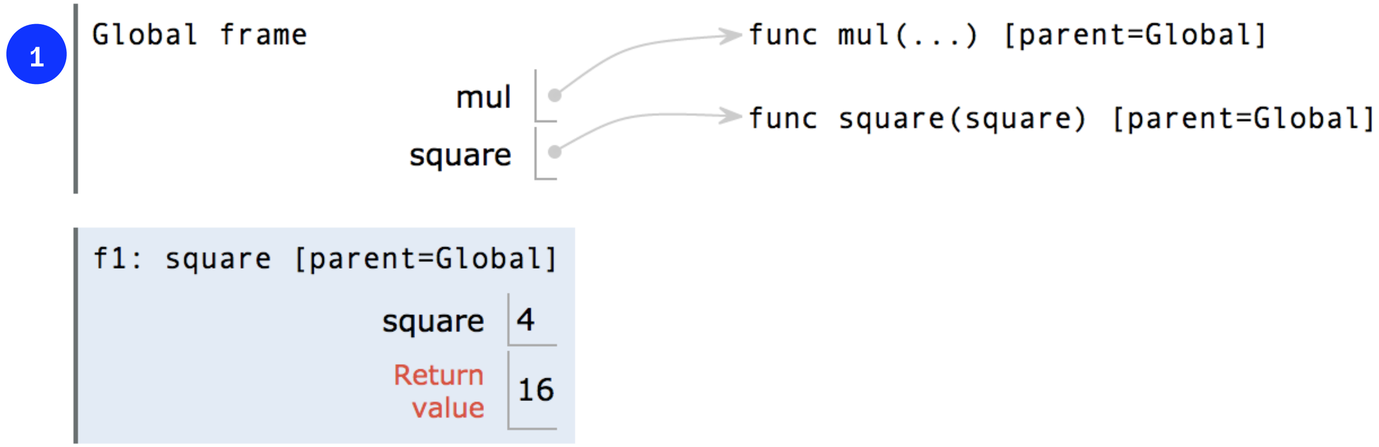
Значение имени в текущем окружении берётся из первого фрейма, в котором это имя присутствует.

Интерактивная диаграмма

Имена означают разное в разных окружениях

Вызывающее выражение и тело функции обрабатываются в разных окружениях.

```
1  from operator import mul
2  def square(square):
3      return mul(square, square)
4  square(4)
```



Каждое выражение вычисляется в контексте окружения.

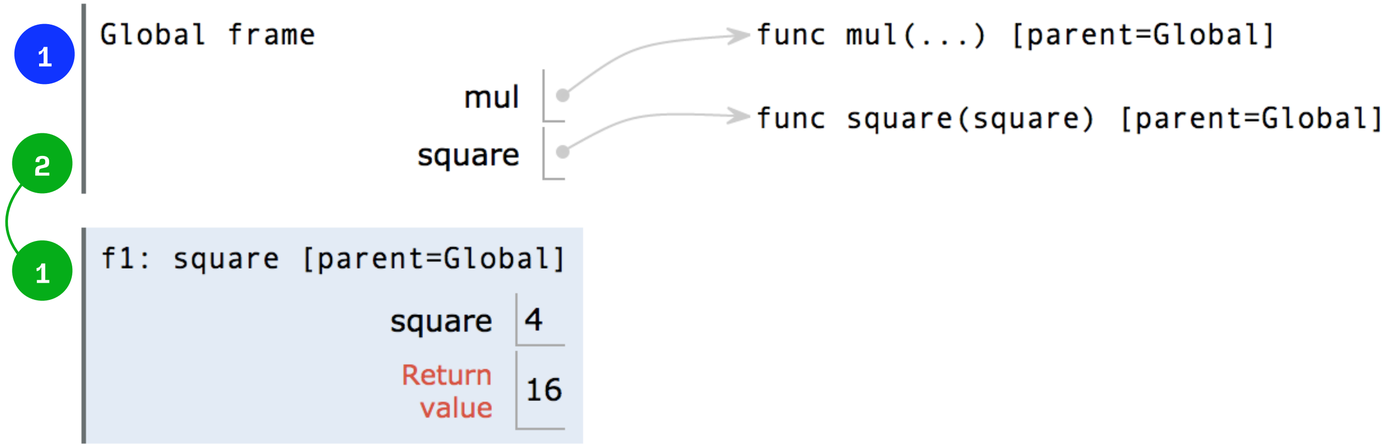
Значение имени в текущем окружении берётся из первого фрейма, в котором это имя присутствует.

Интерактивная диаграмма

Имена означают разное в разных окружениях

Вызывающее выражение и тело функции обрабатываются в разных окружениях.

```
1  from operator import mul
2  def square(square):
3      return mul(square, square)
4  square(4)
```



Каждое выражение вычисляется в контексте окружения.

Значение имени в текущем окружении берётся из первого фрейма, в котором это имя присутствует.

Интерактивная диаграмма

Разные возможности Python

Операторы

Возврат нескольких значений

Докстринги

Доктесты

Аргументы по-умолчанию

(Пример)

Условные инструкции

Инструкции

Инструкция выполняется интерпретатором для совершения действия.

Инструкции

Инструкция выполняется интерпретатором для совершения действия.

Сложные инструкции:

```
<заголовок>:
    <инструкция>
    <инструкция>
    ...
<разделяющий заголовок>:
    <инструкция>
    <инструкция>
    ...
...
```

Инструкции

Инструкция выполняется интерпретатором для совершения действия.

Сложные инструкции:

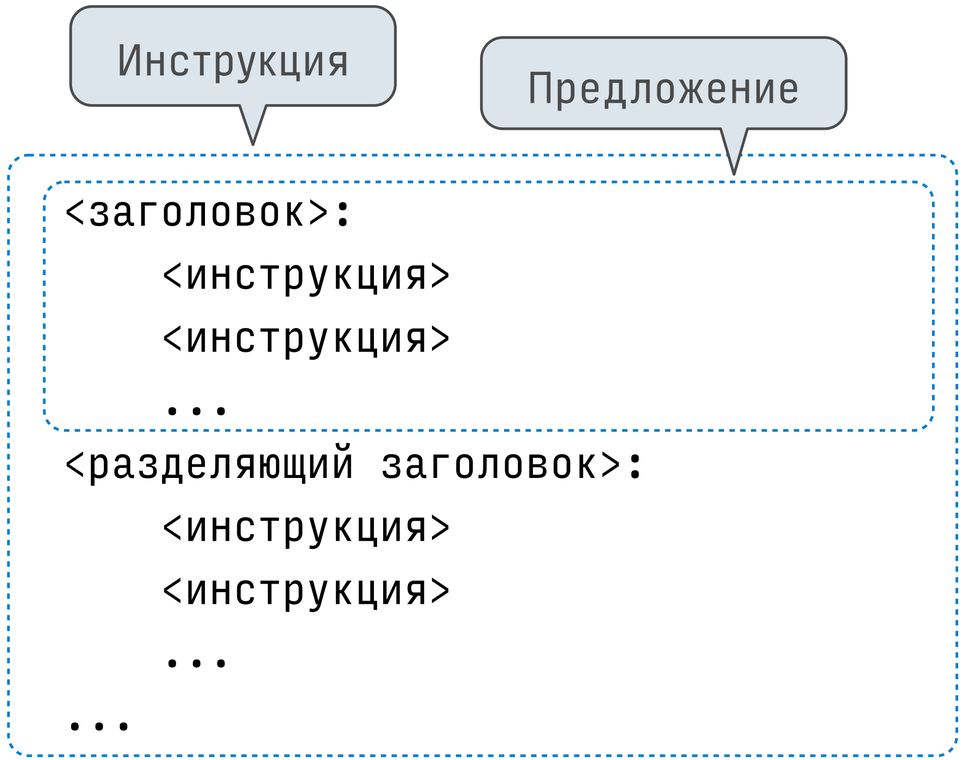
Инструкция

```
<заголовок>:
    <инструкция>
    <инструкция>
    ...
<разделяющий заголовок>:
    <инструкция>
    <инструкция>
    ...
...
```

Инструкции

Инструкция выполняется интерпретатором для совершения действия.

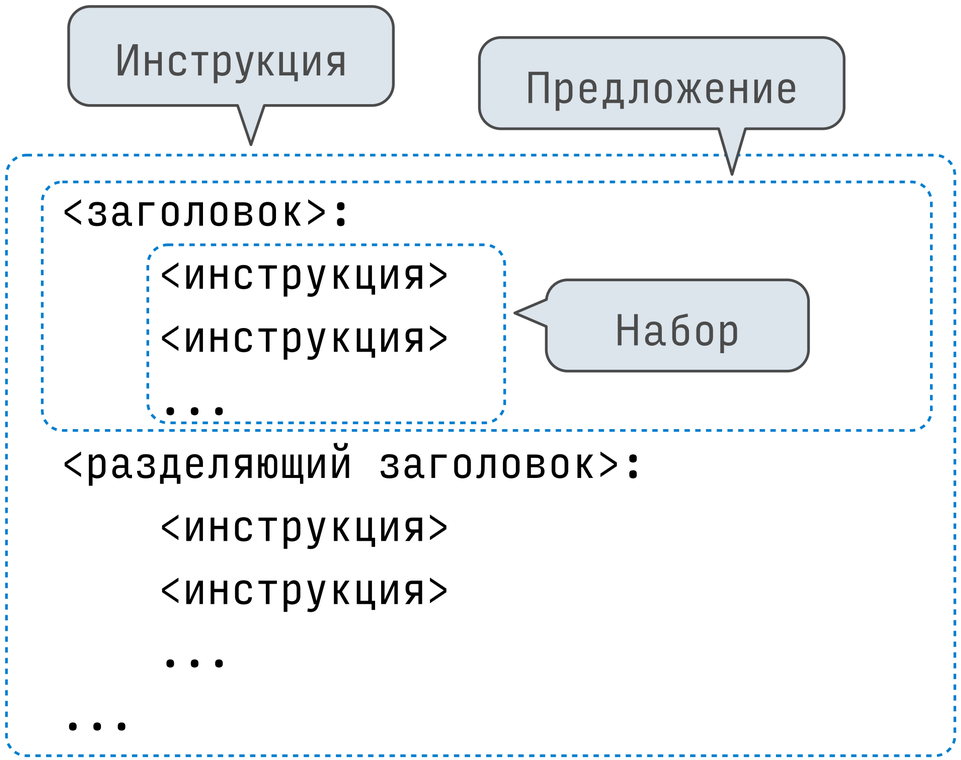
Сложные инструкции:



Инструкции

Инструкция выполняется интерпретатором для совершения действия.

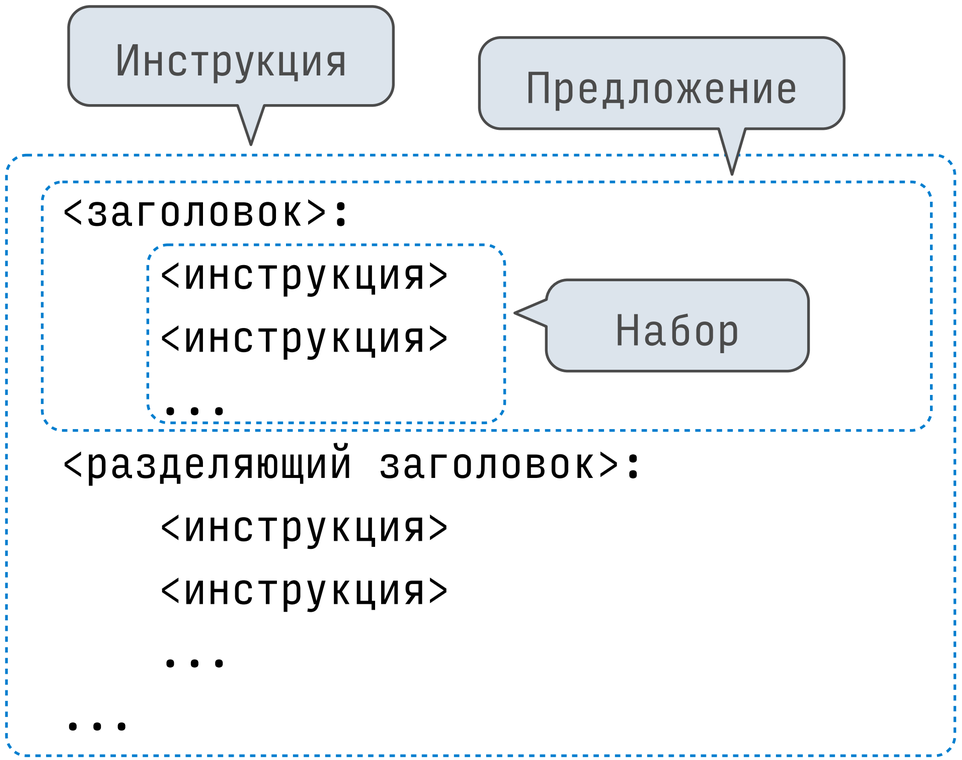
Сложные инструкции:



Инструкции

Инструкция выполняется интерпретатором для совершения действия.

Сложные инструкции:

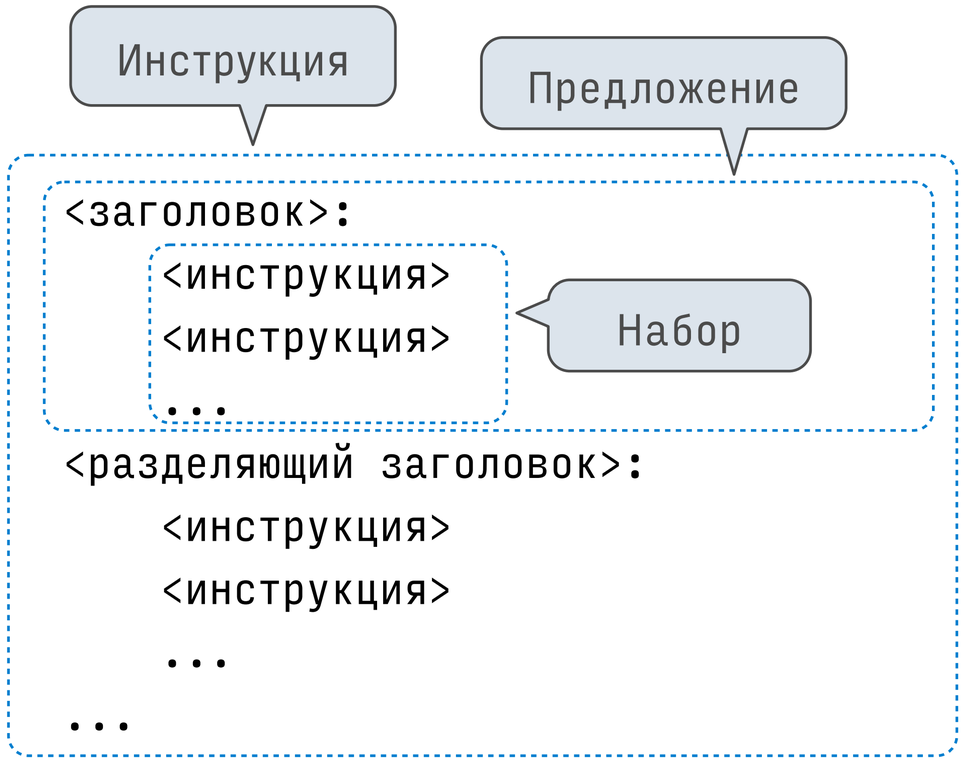


Первый заголовок задает тип инструкции.

Инструкции

Инструкция выполняется интерпретатором для совершения действия.

Сложные инструкции:



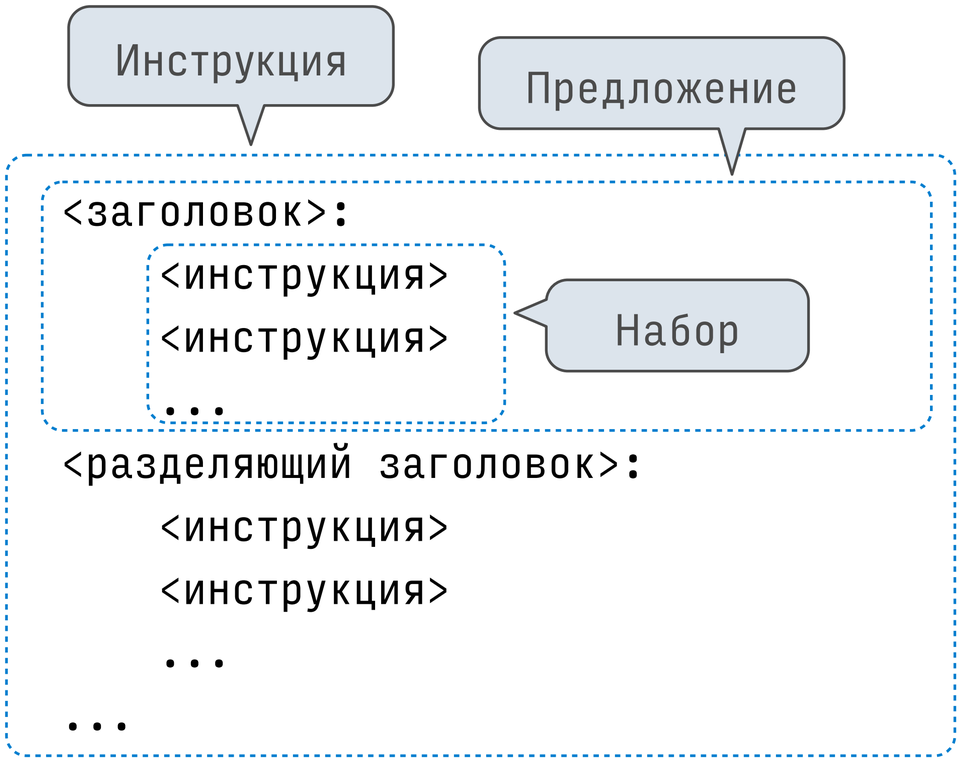
Первый заголовок задает тип инструкции.

Заголовок предложения «управляет» набором инструкций следующих за ним.

Инструкции

Инструкция выполняется интерпретатором для совершения действия.

Сложные инструкции:



Первый заголовок задает тип инструкции.

Заголовок предложения «управляет» набором инструкций следующих за ним.

Инструкция **def** является сложной.

Сложные инструкции

Сложные инструкции:

```
<заголовок>:
    <инструкция>
    <инструкция>
    ...
<разделяющий заголовок>:
    <инструкция>
    <инструкция>
    ...
...
```

Сложные инструкции

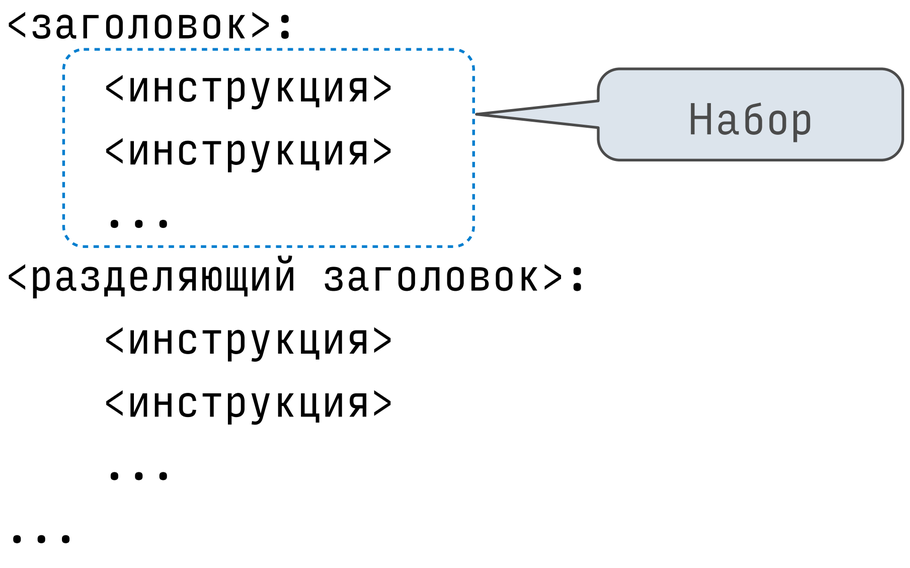
Сложные инструкции:

```
<заголовок>:
    <инструкция>
    <инструкция>
    ...
<разделяющий заголовок>:
    <инструкция>
    <инструкция>
    ...
...
```

Набор — это последовательность инструкций.

Сложные инструкции

Сложные инструкции:



Набор — это последовательность инструкций.

Исполнение набора означает выполнение всей последовательности инструкций по порядку.

Сложные инструкции

Сложные инструкции:

```
<заголовок>:
    <инструкция>
    <инструкция>
    ...
<разделяющий заголовок>:
    <инструкция>
    <инструкция>
    ...
...
```

Набор — это последовательность инструкций.

Исполнение набора означает выполнение всей последовательности инструкций по порядку.

Правило выполнения последовательности инструкций:

- выполнить первую инструкцию;
- пока не указано иное, выполнять остальное.

Условные инструкции

(Пример)

Условные инструкции

(Пример)

```
def absolute_value(x):
    """Возвращает абсолютное значение x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Условные инструкции

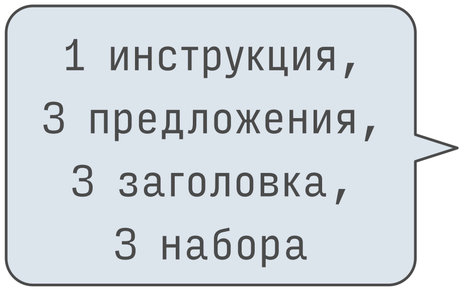
(Пример)

1 инструкция,
3 предложения,
3 заголовка,
3 набора

```
def absolute_value(x):
    """Возвращает абсолютное значение x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Условные инструкции

(Пример)




```
def absolute_value(x):
    """Возвращает абсолютное значение x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Правило выполнения для условных инструкций:

Каждое предложение рассматривается по порядку:

- вычислить выражение в заголовке;
- если оно истинно, выполнить первый набор и пропустить остальные предложения.

Условные инструкции

(Пример)

1 инструкция,
3 предложения,
3 заголовка,
3 набора

```
def absolute_value(x):
    """Возвращает абсолютное значение x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Правило выполнения для условных инструкций:

Каждое предложение рассматривается по порядку:

- вычислить выражение в заголовке;
- если оно истинно, выполнить первый набор и пропустить остальные предложения.

Синтаксические советы:

- всегда начинай с предложения «if»;
- ноль или несколько предложений «elif»;
- ноль или одно предложение «else» строго в конце.

Логические (булевы) контексты

Джордж Буль

```
def absolute_value(x):
    """Возвращает абсолютное значение x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Логические (булевы) контексты

Джордж Буль

```
def absolute_value(x):
    """Возвращает абсолютное значение x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Логические (булевы) контексты

Джордж Буль

```
def absolute_value(x):
    """Возвращает абсолютное значение x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Два логических контекста

Логические (булевы) контексты

Джордж Буль

```
def absolute_value(x):
    """Возвращает абсолютное значение x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Два логических контекста

Ложные значения в Python: False, 0, '', None

Логические (булевы) контексты

Джордж Буль

```
def absolute_value(x):
    """Возвращает абсолютное значение x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Два логических контекста

Ложные значения в Python: False, 0, '', None *(узнаем ещё попозже)*

Логические (булевы) контексты

Джордж Буль

```
def absolute_value(x):
    """Возвращает абсолютное значение x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Два логических контекста

Ложные значения в Python: False, 0, '', None *(узнаем ещё попозже)*

Истинные значения в Python: любые другие (True)

Итерации

Инструкция while

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Инструкция while

Джордж Буль

(Пример)

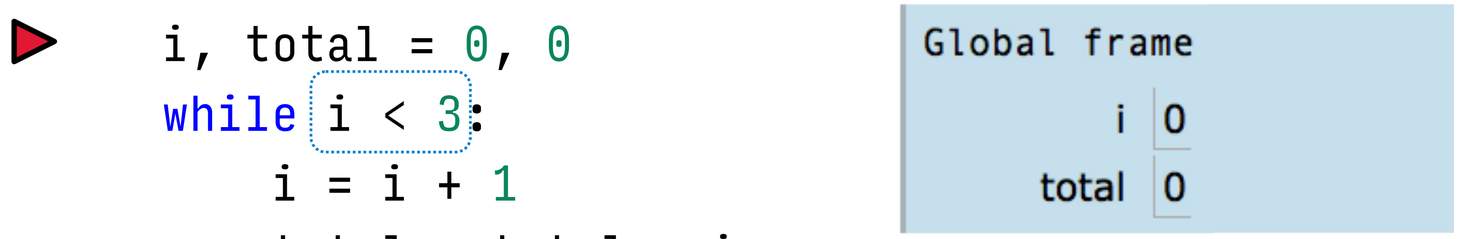
```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame	
i	0
total	0

Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

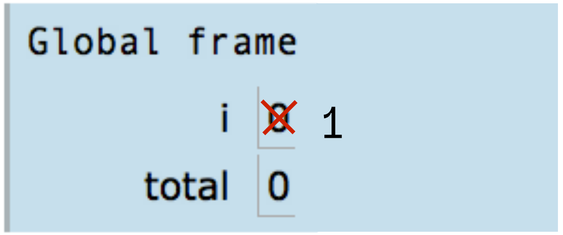
Global frame	
i	0
total	0

Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

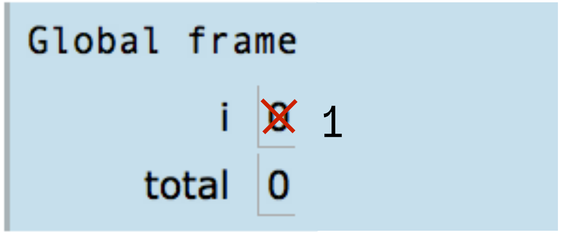


Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

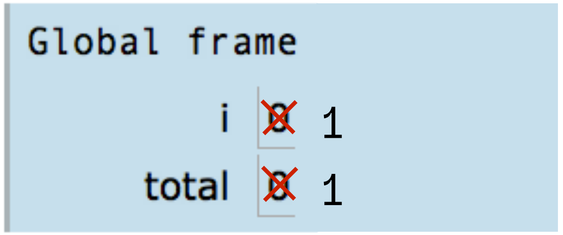


Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

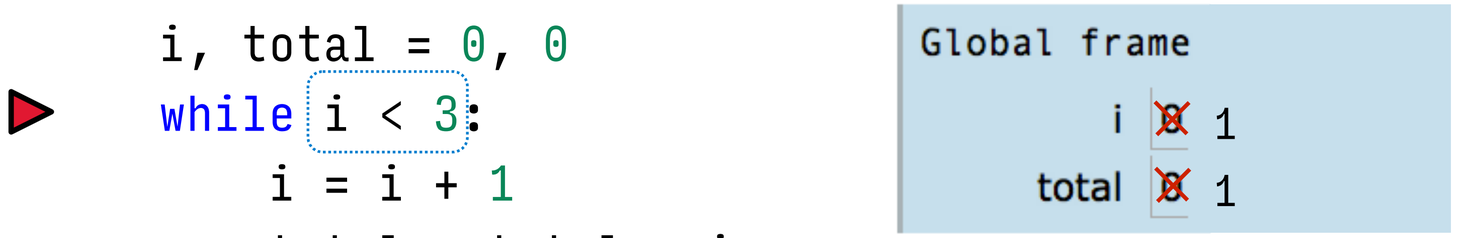


Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

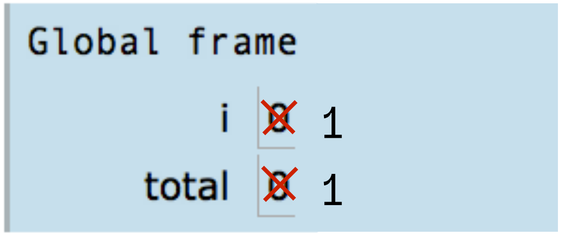


Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

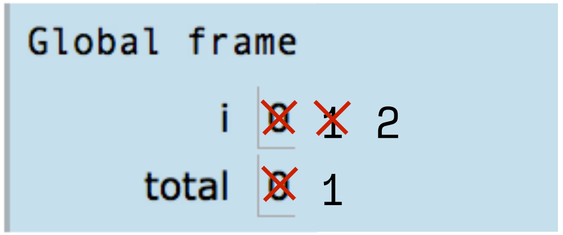


Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

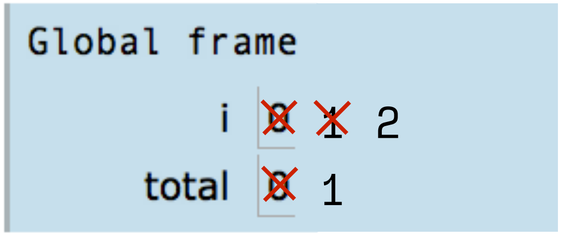


Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

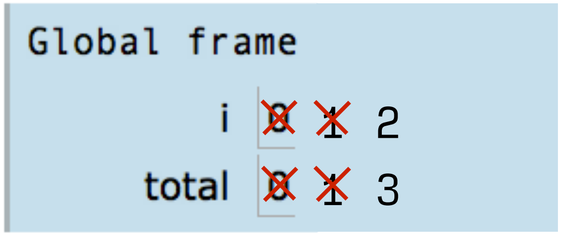


Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Инструкция while

Джордж Буль

(Пример)

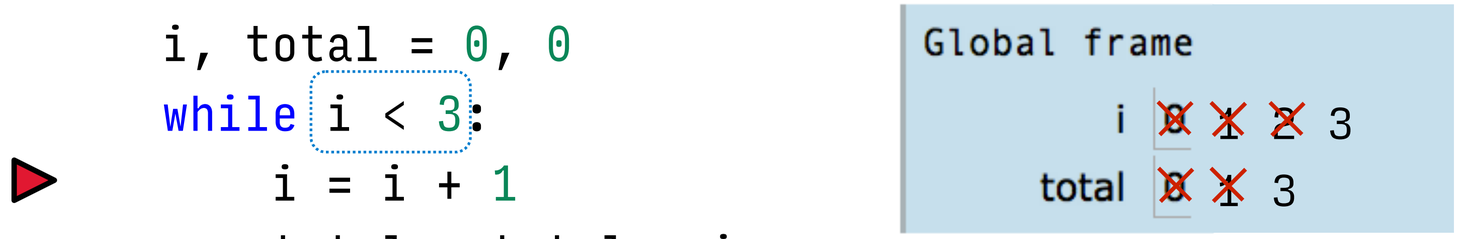
```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

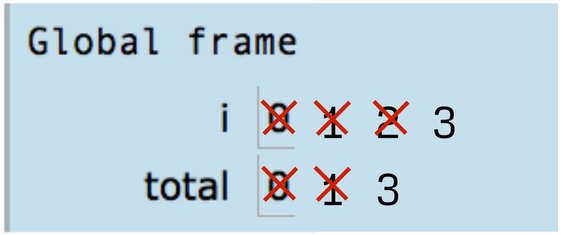


Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame				
i	~~0~~	~~1~~	~~2~~	3
total	~~0~~	~~1~~	~~3~~	6

Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

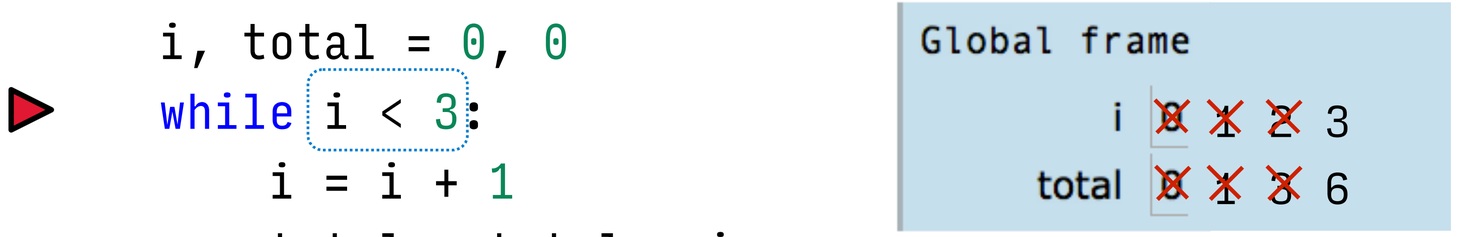
Global frame				
i	~~0~~	~~1~~	~~2~~	3
total	~~0~~	~~1~~	~~3~~	6

Инструкция while

Джордж Буль

(Пример)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



Правило выполнения инструкции while:

- вычислить выражение в заголовке;
- если оно истинно, выполнить набор (весь), затем вернуться к предыдущему шагу.