Mutable Functions

Announcements

Mutable Functions

A Function with Behavior That Varies Over Time

```
Let's model a bank account that has a balance of $100
```

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```
>>> withdraw(25)
```

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```
>>> withdraw(25)
75
```

A Function with Behavior That Varies Over Time

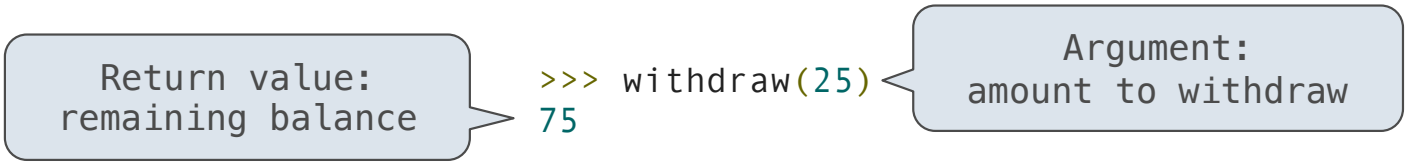
Let's model a bank account that has a balance of $100

```
>>> withdraw(25)
75
```

Argument:
amount to withdraw

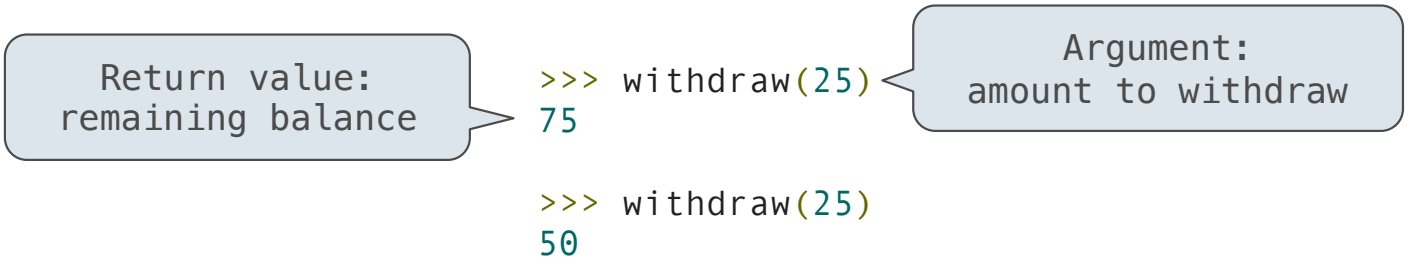
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100



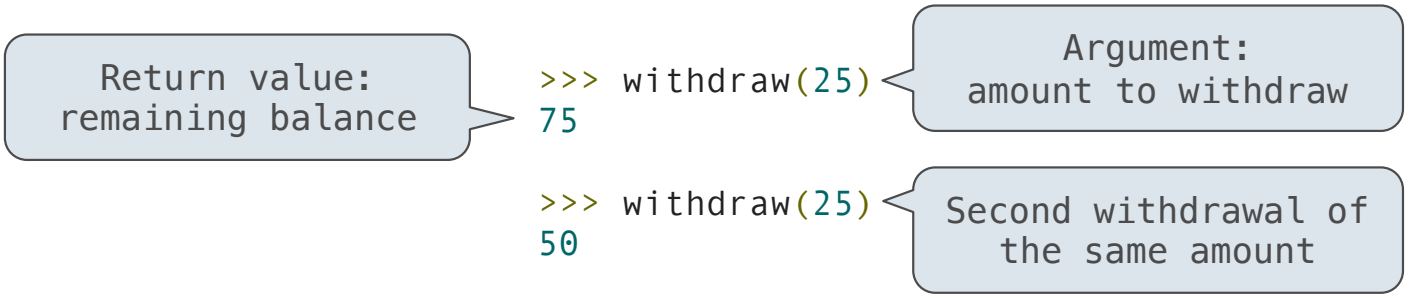
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100



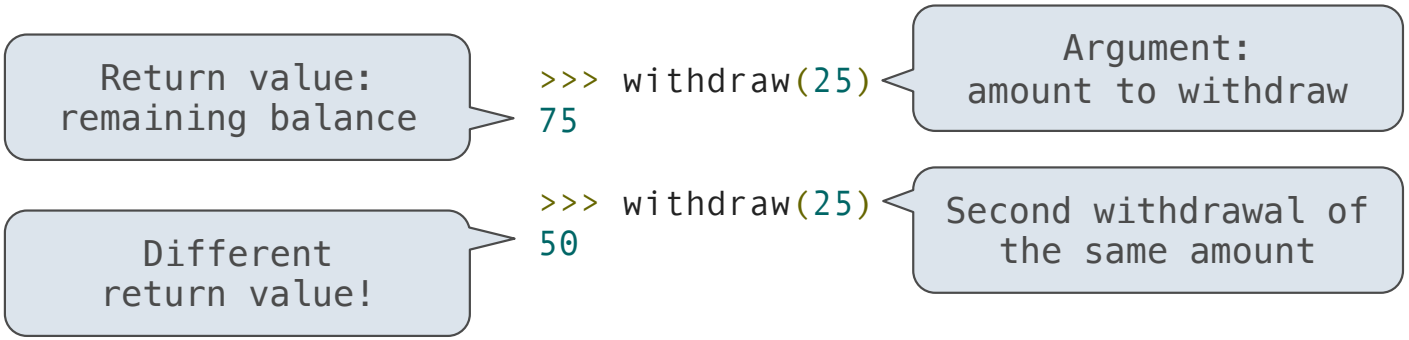
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100



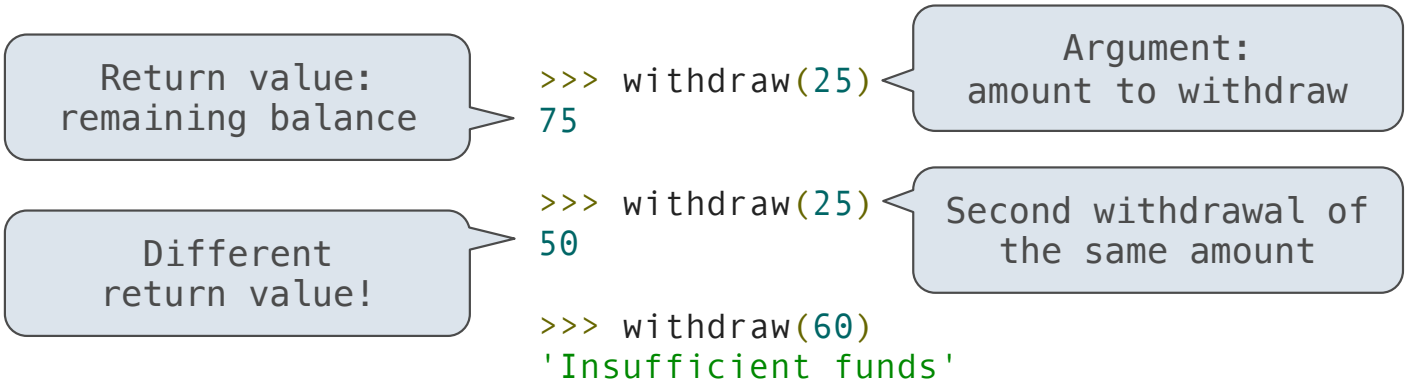
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100



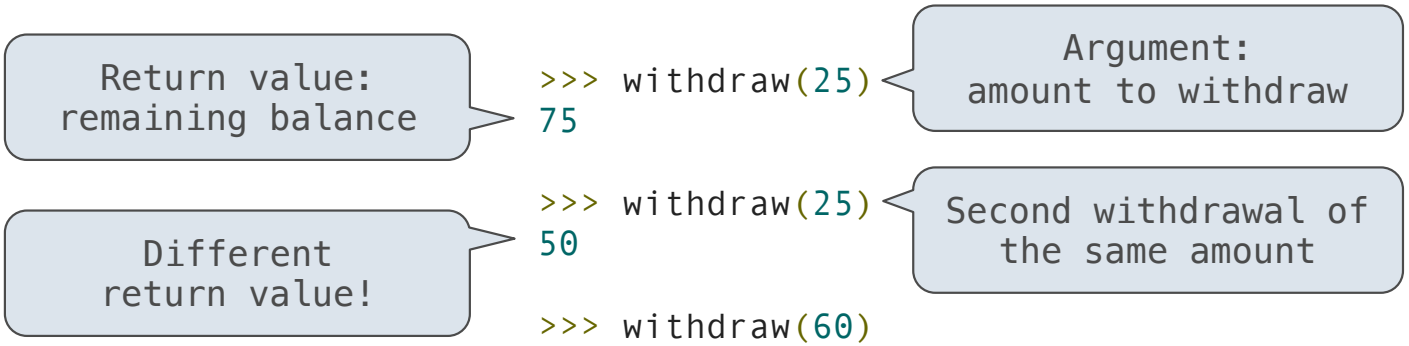
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100



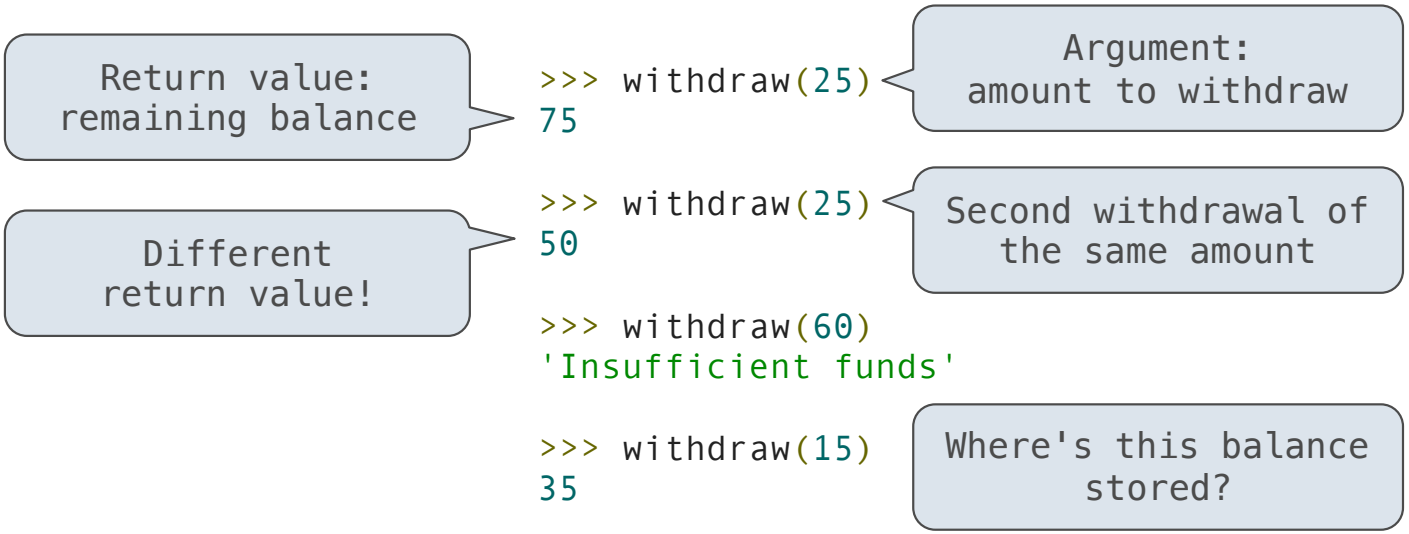
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100



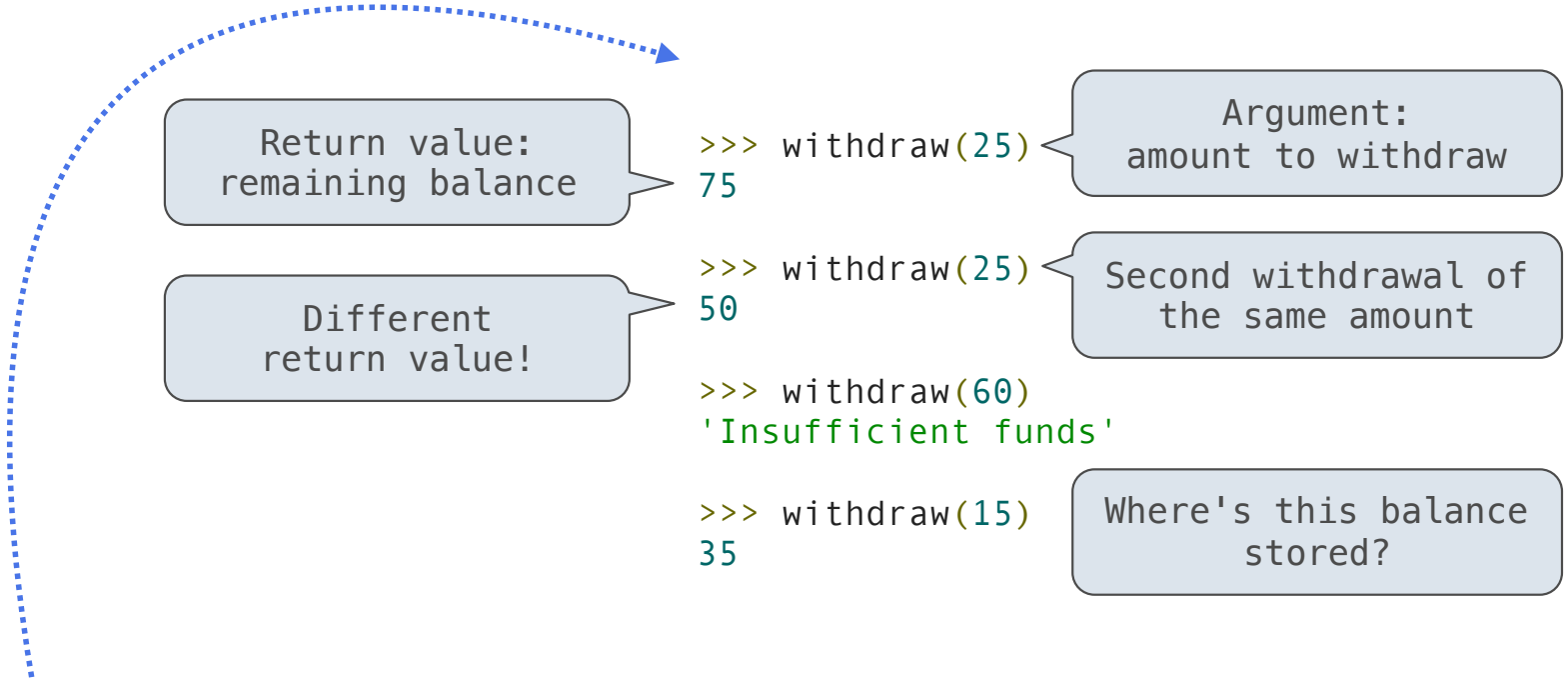
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100



A Function with Behavior That Varies Over Time

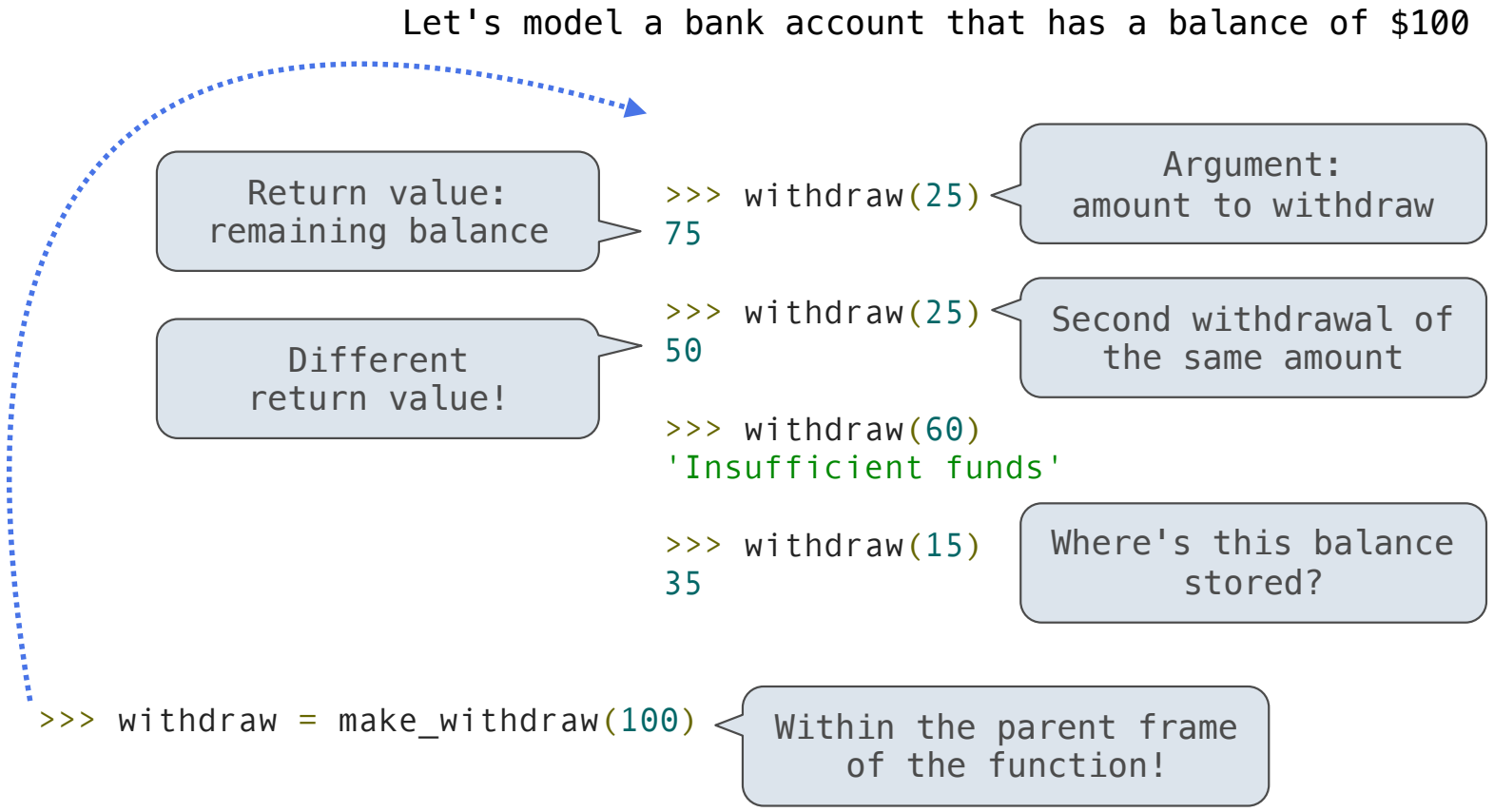
Let's model a bank account that has a balance of $100




```
>>> withdraw(25)
75
>>> withdraw(25)
50
>>> withdraw(60)
'Insufficient funds'
>>> withdraw(15)
35
```

```
>>> withdraw = make_withdraw(100)
```

A Function with Behavior That Varies Over Time



A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```
>>> withdraw(25)
75
>>> withdraw(25)
50
>>> withdraw(60)
'Insufficient funds'
>>> withdraw(15)
35
```

Return value: remaining balance

Argument: amount to withdraw

Different return value!

Second withdrawal of the same amount

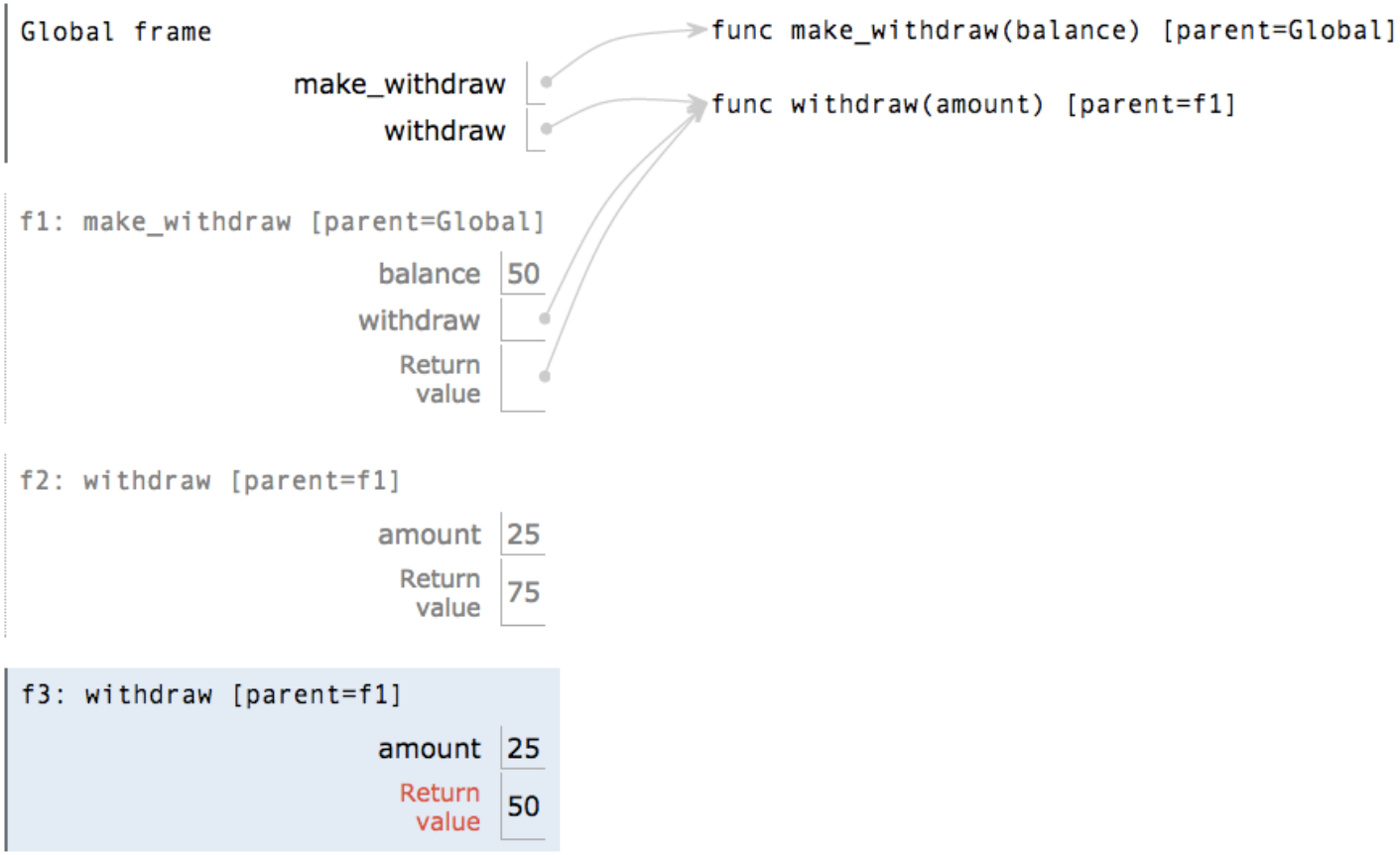
Where's this balance stored?

```
>>> withdraw = make_withdraw(100)
```

Within the parent frame of the function!

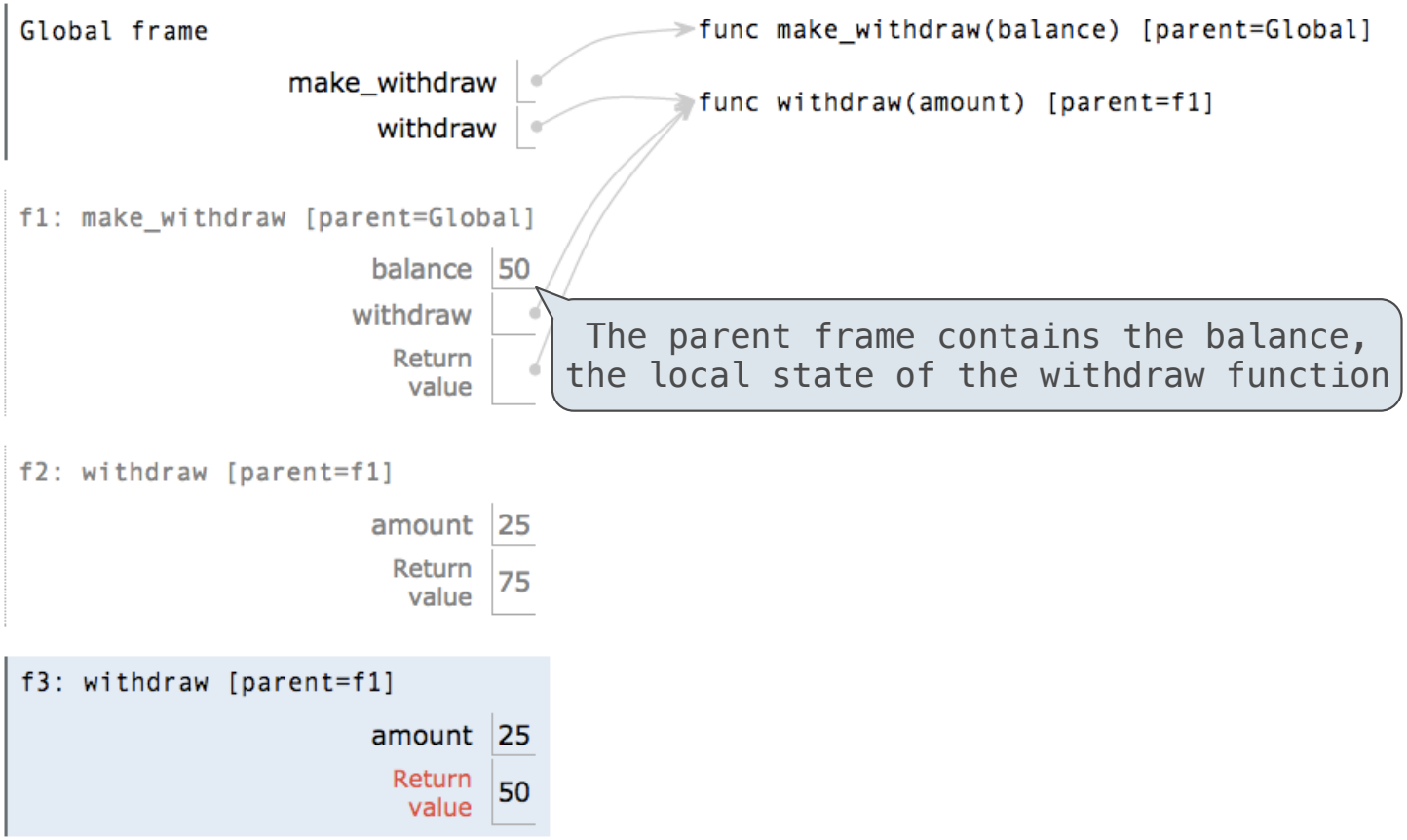
A function has a body and a parent environment

Persistent Local State Using Environments



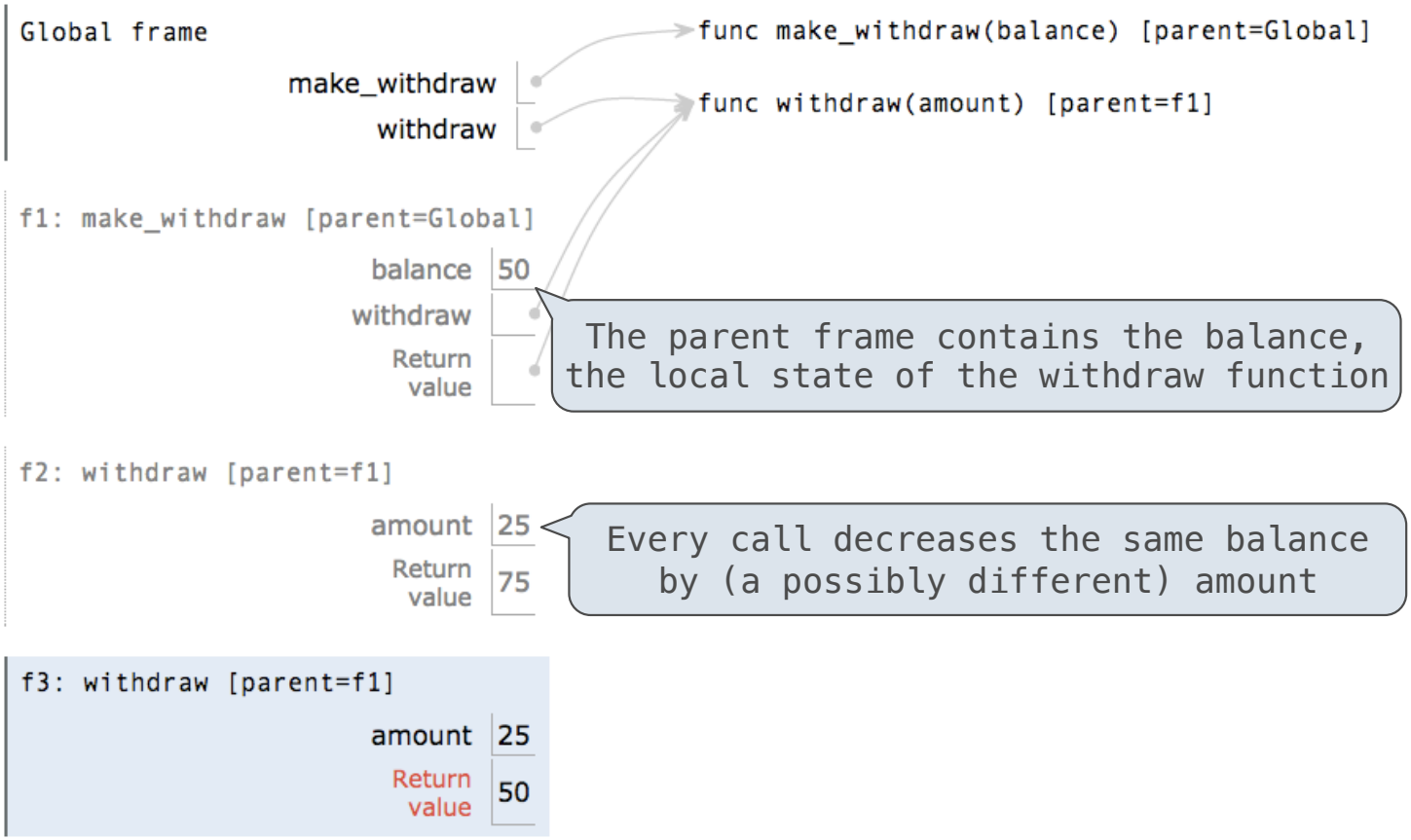
pythontutor.com/composingprograms.html#code=def%20make_withdraw%28balance%29%3A%0A%20%20%20%20def%20withdraw%28amount%29%3A%0A%20%20%20%20%20%20%20%20nonlocal%20balance%0A%20%20%20%20%20%20%20%20if%20amount%20%3E%20balance%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20'Insufficient%20funds'%0A%20%20%20%20%20%20%20%20balance%20%3D%20balance%20-%20amount%0A%20%20%20%20%20%20%20%20return%20balance%0A%20%20%20%20return%20withdraw%0A%0Awithdraw%20%3D%20make_withdraw%28100%29%0Awithdraw%2825%29%0Awithdraw%2825%29%0Awithdraw%2860%29%0Awithdraw%2815%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Persistent Local State Using Environments



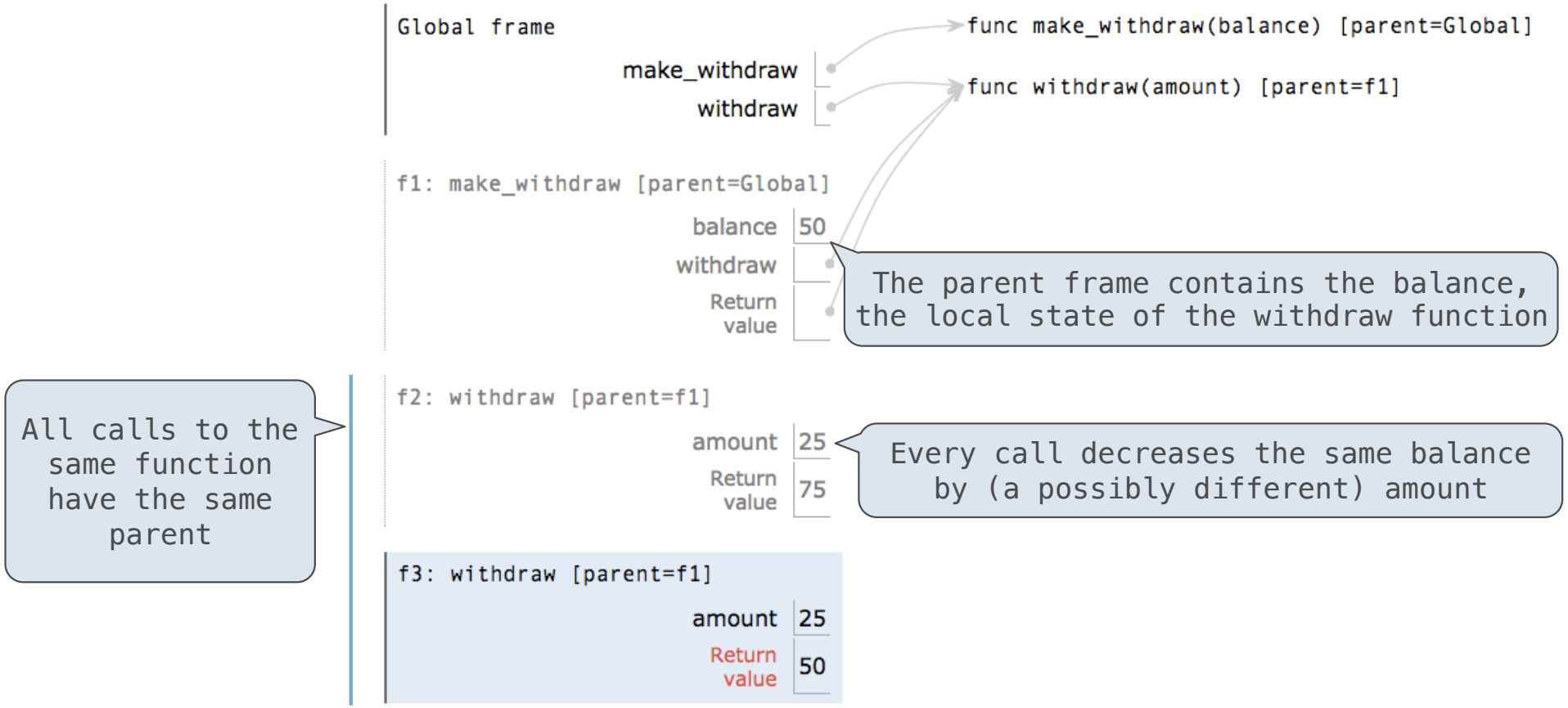
pythontutor.com/composingprograms.html#code=def%20make_withdraw%28balance%29%3A%0A%20%20%20%20def%20withdraw%28amount%29%3A%0A%20%20%20%20%20%20%20%20nonlocal%20balance%0A%20%20%20%20%20%20%20%20if%20amount%20%3E%20balance%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20'Insufficient%20funds'%0A%20%20%20%20%20%20%20%20balance%20%3D%20balance%20-%20amount%0A%20%20%20%20%20%20%20%20return%20balance%0A%20%20%20%20return%20withdraw%0A%0Awithdraw%20%3D%20make_withdraw%28100%29%0Awithdraw%2825%29%0Awithdraw%2825%29%0Awithdraw%2860%29%0Awithdraw%2815%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Persistent Local State Using Environments



pythontutor.com/composingprograms.html#code=def%20make_withdraw%28balance%29%3A%0A%20%20%20%20def%20withdraw%28amount%29%3A%0A%20%20%20%20%20%20%20%20nonlocal%20balance%0A%20%20%20%20%20%20%20%20if%20amount%20%3E%20balance%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20'Insufficient%20funds'%0A%20%20%20%20%20%20%20%20balance%20%3D%20balance%20-%20amount%0A%20%20%20%20%20%20%20%20return%20balance%0A%20%20%20%20return%20withdraw%0A%0Awithdraw%20%3D%20make_withdraw%28100%29%0Awithdraw%2825%29%0Awithdraw%2825%29%0Awithdraw%2860%29%0Awithdraw%2815%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

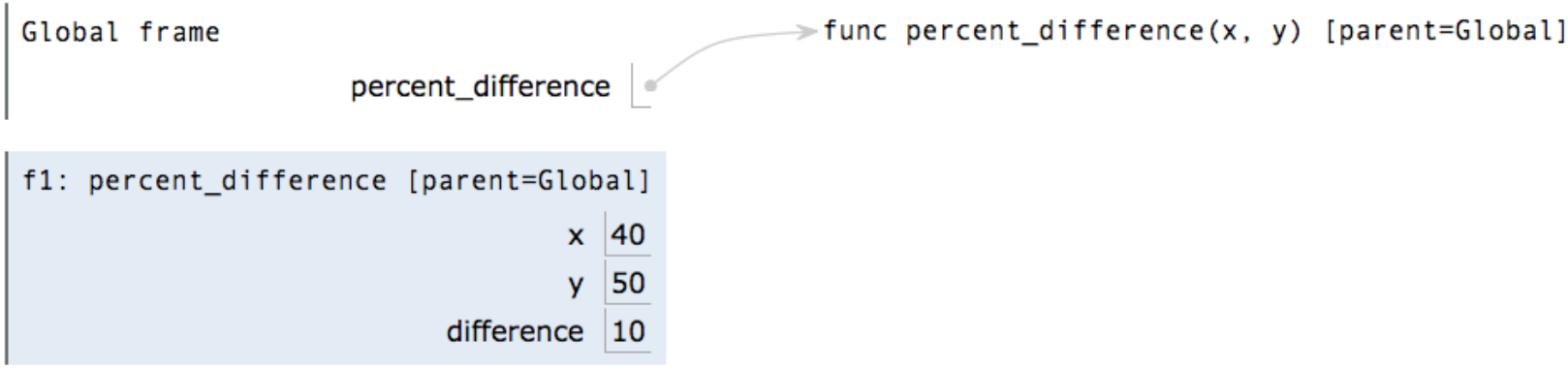
Persistent Local State Using Environments



pythontutor.com/composingprograms.html#code=def%20make_withdraw%28balance%29%3A%0A%20%20%20%20def%20withdraw%28amount%29%3A%0A%20%20%20%20%20%20%20%20nonlocal%20balance%0A%20%20%20%20%20%20%20%20if%20amount%20%3E%20balance%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20'Insufficient%20funds'%0A%20%20%20%20%20%20%20%20balance%20%3D%20balance%20-%20amount%0A%20%20%20%20%20%20%20%20return%20balance%0A%20%20%20%20return%20withdraw%0A%0Awithdraw%20%3D%20make_withdraw%28100%29%0Awithdraw%2825%29%0Awithdraw%2825%29%0Awithdraw%2860%29%0Awithdraw%2815%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Reminder: Local Assignment

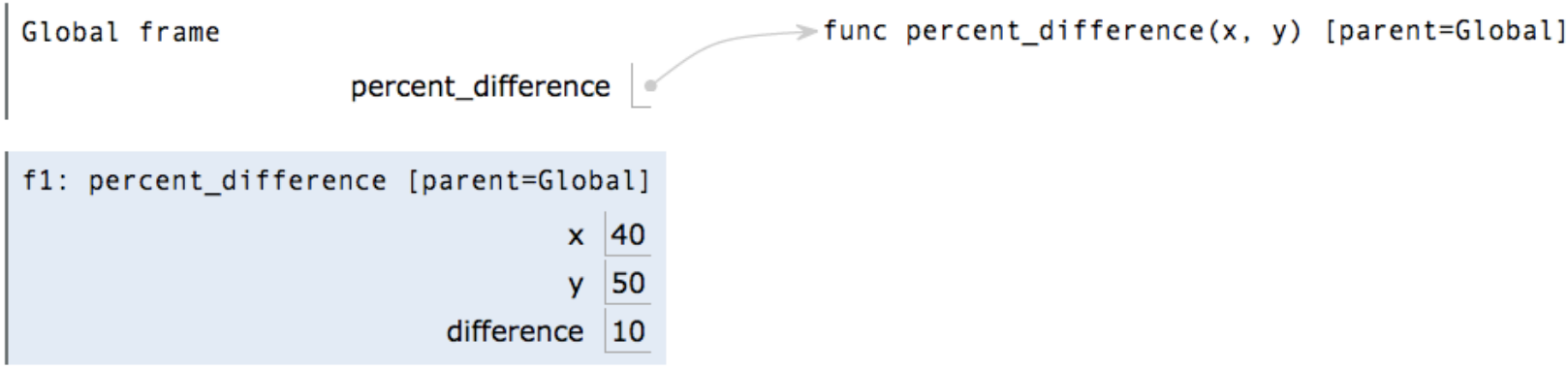
```
def percent_difference(x, y):
        difference = abs(x-y)
        return 100 * difference / x
diff = percent_difference(40, 50)
```



Reminder: Local Assignment

```
def percent_difference(x, y):
    difference = abs(x-y)
    return 100 * difference / x
diff = percent_difference(40, 50)
```

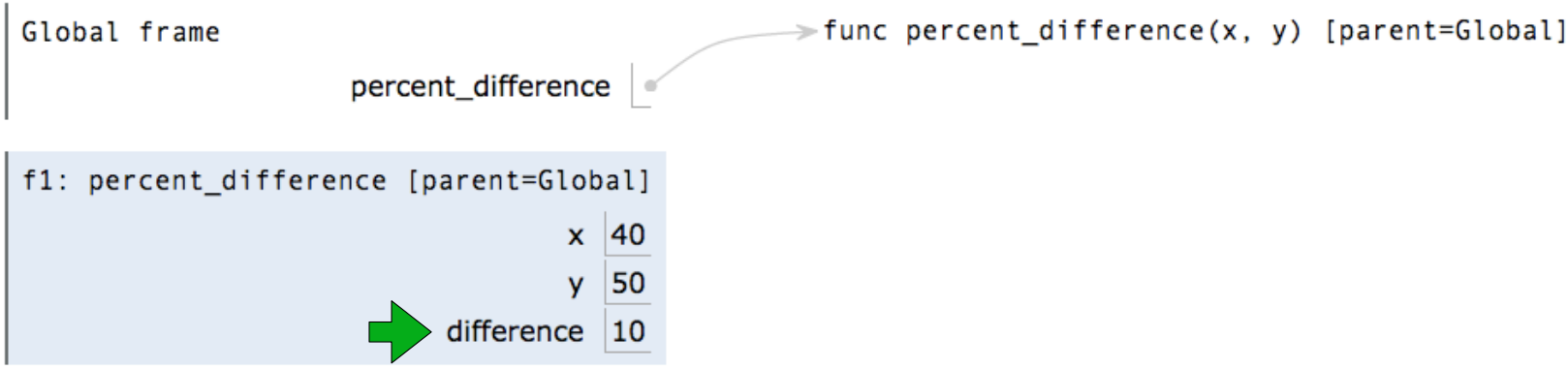
Assignment binds name(s) to value(s) in the first frame of the current environment



Reminder: Local Assignment

```
def percent_difference(x, y):
    difference = abs(x-y)
    return 100 * difference / x
diff = percent_difference(40, 50)
```

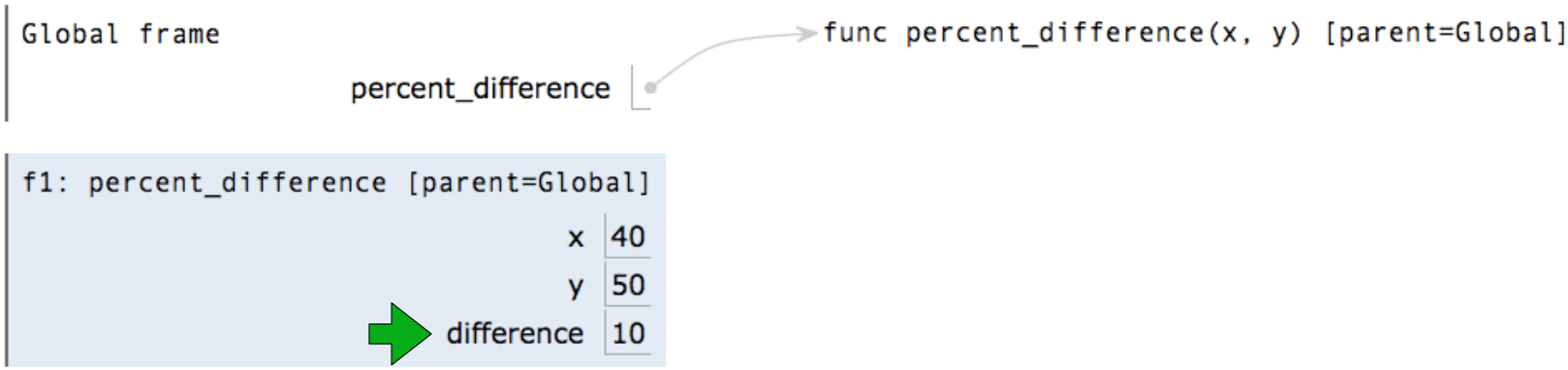
Assignment binds name(s) to value(s) in the first frame of the current environment



Reminder: Local Assignment

```
def percent_difference(x, y):
    difference = abs(x-y)
    return 100 * difference / x
diff = percent_difference(40, 50)
```

Assignment binds name(s) to value(s) in the first frame of the current environment

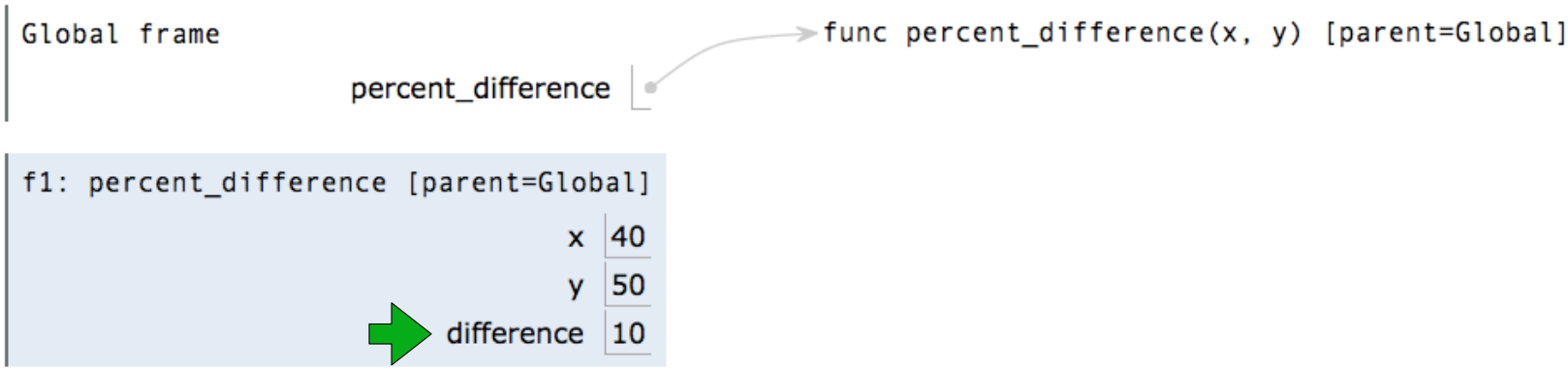


Execution rule for assignment statements:

Reminder: Local Assignment

```
def percent_difference(x, y):
    difference = abs(x-y)
    return 100 * difference / x
diff = percent_difference(40, 50)
```

Assignment binds name(s) to value(s) in the first frame of the current environment



Execution rule for assignment statements:

1. Evaluate all expressions right of =, from left to right

2. Bind the names on the left to the resulting values in the **current frame**

Non-Local Assignment & Persistent Local State

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
```

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
```

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
```

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
```

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
```

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
```

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
```

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
```

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw
```

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw
```

Declare the name "balance" nonlocal at the top of the body of the function in which it is re-assigned

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw
```

Declare the name "balance" nonlocal at the top of the body of the function in which it is re-assigned

Re-bind balance in the first non-local frame in which it was bound previously

Non-Local Assignment & Persistent Local State

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw
```

Declare the name "balance" nonlocal at the top of the body of the function in which it is re-assigned

Re-bind balance in the first non-local frame in which it was bound previously

(Demo)

Non-Local Assignment

The Effect of Nonlocal Statements

```
nonlocal <name>
```

The Effect of Nonlocal Statements

```
nonlocal <name>
```

Effect: Future assignments to that name change its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

The Effect of Nonlocal Statements

```
nonlocal <name>
```

Effect: Future assignments to that name change its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

The Effect of Nonlocal Statements

```
nonlocal <name>, <name>, ...
```

Effect: Future assignments to that name change its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

The Effect of Nonlocal Statements

```
nonlocal <name>, <name>, ...
```

Effect: Future assignments to that name change its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

From the Python 3 language reference:

The Effect of Nonlocal Statements

```
nonlocal <name>, <name>, ...
```

Effect: Future assignments to that name change its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

From the Python 3 language reference:

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope.

The Effect of Nonlocal Statements

```
nonlocal <name>, <name>, ...
```

Effect: Future assignments to that name change its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

From the Python 3 language reference:

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope.

Names listed in a nonlocal statement must not collide with pre-existing bindings in the local scope.

The Effect of Nonlocal Statements

```
nonlocal <name>, <name>, ...
```

Effect: Future assignments to that name change its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

From the Python 3 language reference:

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope.

Names listed in a nonlocal statement must not collide with pre-existing bindings in the local scope.

Current frame

The Effect of Nonlocal Statements

```
nonlocal <name>, <name>, ...
```

Effect: Future assignments to that name change its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

From the Python 3 language reference:

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope.

Names listed in a nonlocal statement must not collide with pre-existing bindings in the local scope.

Current frame

http://docs.python.org/release/3.1.3/reference/simple_stmts.html#the-nonlocal-statement

The Effect of Nonlocal Statements

```
nonlocal <name>, <name>, ...
```

Effect: Future assignments to that name change its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

From the Python 3 language reference:

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope.

Names listed in a nonlocal statement must not collide with pre-existing bindings in the local scope.

Current frame

http://docs.python.org/release/3.1.3/reference/simple_stmts.html#the-nonlocal-statement

http://www.python.org/dev/peps/pep-3104/

The Many Meanings of Assignment Statements

```
x = 2
```

The Many Meanings of Assignment Statements

```
x = 2
```

Status **Effect**

The Many Meanings of Assignment Statements

```
x = 2
```

Status **Effect**

- No nonlocal statement
- "x" **is not** bound locally

The Many Meanings of Assignment Statements

```
x = 2
```

Status	Effect
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment

The Many Meanings of Assignment Statements

```
x = 2
```

Status	Effect
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment
• No nonlocal statement • "x" **is** bound locally	

The Many Meanings of Assignment Statements

`x = 2`

Status	Effect
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current environment

The Many Meanings of Assignment Statements

```
x = 2
```

Status	Effect
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current environment
• nonlocal x • "x" **is** bound in a non-local frame	

The Many Meanings of Assignment Statements

`x = 2`

Status	Effect
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current environment
• nonlocal x • "x" **is** bound in a non-local frame	Re-bind "x" to 2 in the first non-local frame of the current environment in which "x" is bound

The Many Meanings of Assignment Statements

`x = 2`

Status	Effect
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current environment
• nonlocal x • "x" **is** bound in a non-local frame	Re-bind "x" to 2 in the first non-local frame of the current environment in which "x" is bound
• nonlocal x • "x" **is not** bound in a non-local frame	

The Many Meanings of Assignment Statements

`x = 2`

Status	Effect
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current environment
• nonlocal x • "x" **is** bound in a non-local frame	Re-bind "x" to 2 in the first non-local frame of the current environment in which "x" is bound
• nonlocal x • "x" **is not** bound in a non-local frame	SyntaxError: no binding for nonlocal 'x' found

The Many Meanings of Assignment Statements

`x = 2`

Status	Effect
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current environment
• nonlocal x • "x" **is** bound in a non-local frame	Re-bind "x" to 2 in the first non-local frame of the current environment in which "x" is bound
• nonlocal x • "x" **is not** bound in a non-local frame	SyntaxError: no binding for nonlocal 'x' found
• nonlocal x • "x" **is** bound in a non-local frame • "x" also bound locally	

The Many Meanings of Assignment Statements

`x = 2`

Status	Effect
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current environment
• nonlocal x • "x" **is** bound in a non-local frame	Re-bind "x" to 2 in the first non-local frame of the current environment in which "x" is bound
• nonlocal x • "x" **is not** bound in a non-local frame	SyntaxError: no binding for nonlocal 'x' found
• nonlocal x • "x" **is** bound in a non-local frame • "x" also bound locally	SyntaxError: name 'x' is parameter and nonlocal

Python Particulars

Python Particulars

Python pre-computes which frame contains each name before executing the body of a function.

Python Particulars

Python pre-computes which frame contains each name before executing the body of a function.

Within the body of a function, all instances of a name must refer to the same frame.

Python Particulars

Python pre-computes which frame contains each name before executing the body of a function.

Within the body of a function, all instances of a name must refer to the same frame.

```
def make_withdraw(balance):
    def withdraw(amount):
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw

wd = make_withdraw(20)
wd(5)
```

pythontutor.com/composingprograms.html#code=def%20make_withdraw%28balance%29%3A%0A%20%20%20%20def%20withdraw%28amount%29%3A%0A%20%20%20%20%20%20%20%20if%20amount%20%3E%20balance%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20%27Insufficient%20funds%27%0A%20%20%20%20%20%20%20%20balance%20%3D%20balance%20-%20amount%0A%20%20%20%20%20%20%20%20return%20balance%0A%20%20%20%20return%20withdraw%0A%0Awithdraw%20%3D%20make_withdraw%2820%29%0Awithdraw%285%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Python Particulars

Python pre-computes which frame contains each name before executing the body of a function.

Within the body of a function, all instances of a name must refer to the same frame.

```
def make_withdraw(balance):
    def withdraw(amount):
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw

wd = make_withdraw(20)
wd(5)
```

Local assignment

pythontutor.com/composingprograms.html#code=def%20make_withdraw%28balance%29%3A%0A%20%20%20%20def%20withdraw%28amount%29%3A%0A%20%20%20%20%20%20%20%20if%20amount%20%3E%20balance%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20'Insufficient%20funds'%0A%20%20%20%20%20%20%20%20balance%20%3D%20balance%20-%20amount%0A%20%20%20%20%20%20%20%20return%20balance%0A%20%20%20%20return%20withdraw%0A%0Awithdraw%20%3D%20make_withdraw%2820%29%0Awithdraw%285%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Python Particulars

Python pre-computes which frame contains each name before executing the body of a function.

Within the body of a function, all instances of a name must refer to the same frame.

```
def make_withdraw(balance):
    def withdraw(amount):
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw

wd = make_withdraw(20)
wd(5)
```

Local assignment

UnboundLocalError: local variable 'balance' referenced before assignment

pythontutor.com/composingprograms.html#code=def%20make_withdraw%28balance%29%3A%0A%20%20%20%20def%20withdraw%28amount%29%3A%0A%20%20%20%20%20%20%20%20if%20amount%20%3E%20balance%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20'Insufficient%20funds'%0A%20%20%20%20%20%20%20%20balance%20%3D%20balance%20-%20amount%0A%20%20%20%20%20%20%20%20return%20balance%0A%20%20%20%20return%20withdraw%0A%0Awithdraw%20%3D%20make_withdraw%2820%29%0Awithdraw%285%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Mutable Values & Persistent Local State

Mutable values can be changed *without* a nonlocal statement.

Mutable Values & Persistent Local State

Mutable values can be changed *without* a nonlocal statement.

```
def make_withdraw_list(balance):
    b = [balance]
    def withdraw(amount):
        if amount > b[0]:
            return 'Insufficient funds'
        b[0] = b[0] - amount
        return b[0]
    return withdraw

withdraw = make_withdraw_list(100)
withdraw(25)
```

Mutable Values & Persistent Local State

Mutable values can be changed *without* a nonlocal statement.

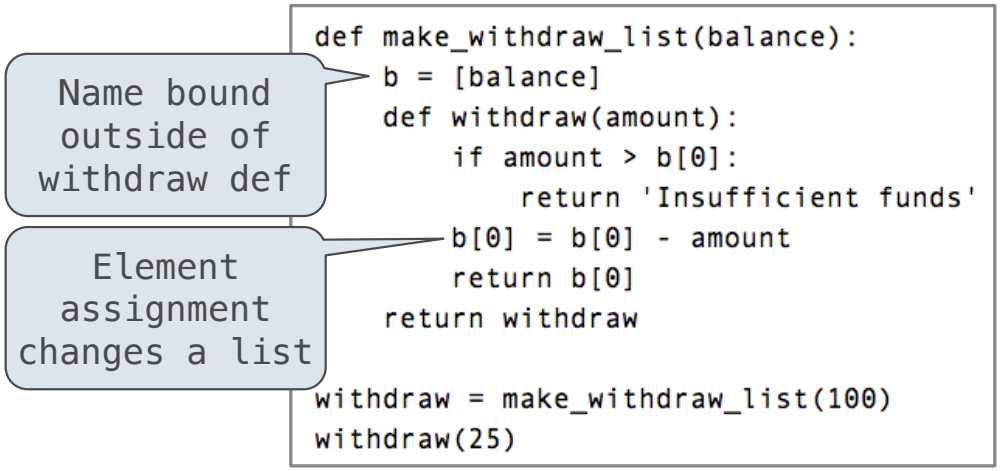
Name bound outside of withdraw def

```
def make_withdraw_list(balance):
    b = [balance]
    def withdraw(amount):
        if amount > b[0]:
            return 'Insufficient funds'
        b[0] = b[0] - amount
        return b[0]
    return withdraw

withdraw = make_withdraw_list(100)
withdraw(25)
```

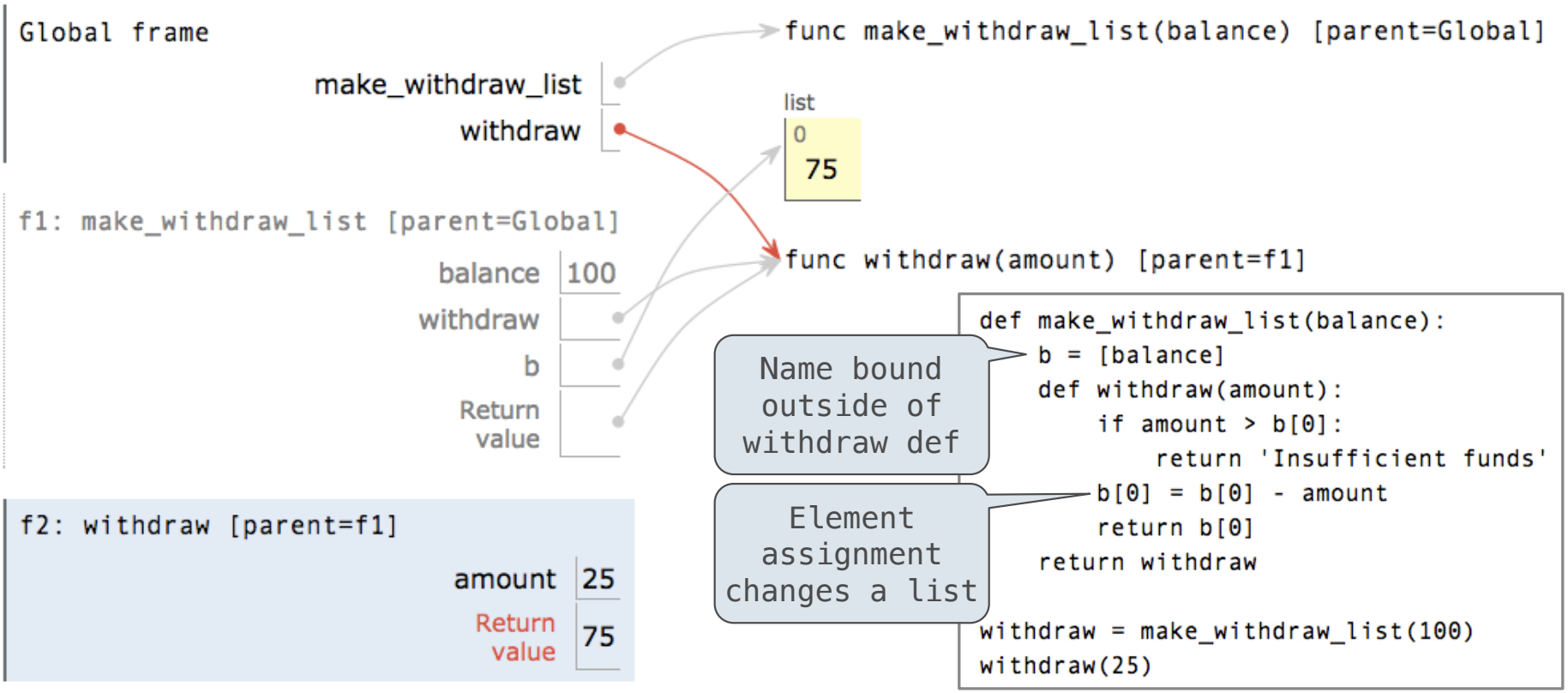
Mutable Values & Persistent Local State

Mutable values can be changed *without* a nonlocal statement.



Mutable Values & Persistent Local State

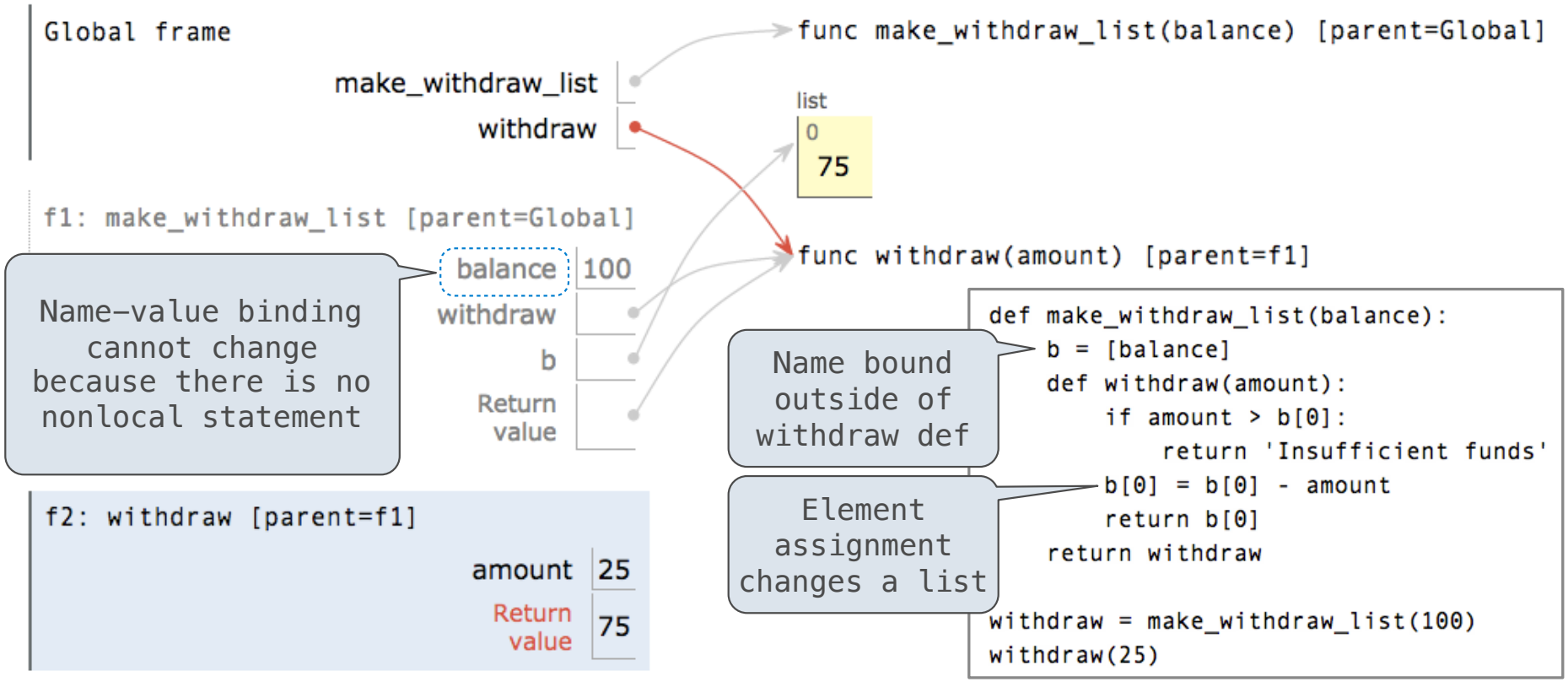
Mutable values can be changed *without* a nonlocal statement.



goo.gl/y4TyFZ

Mutable Values & Persistent Local State

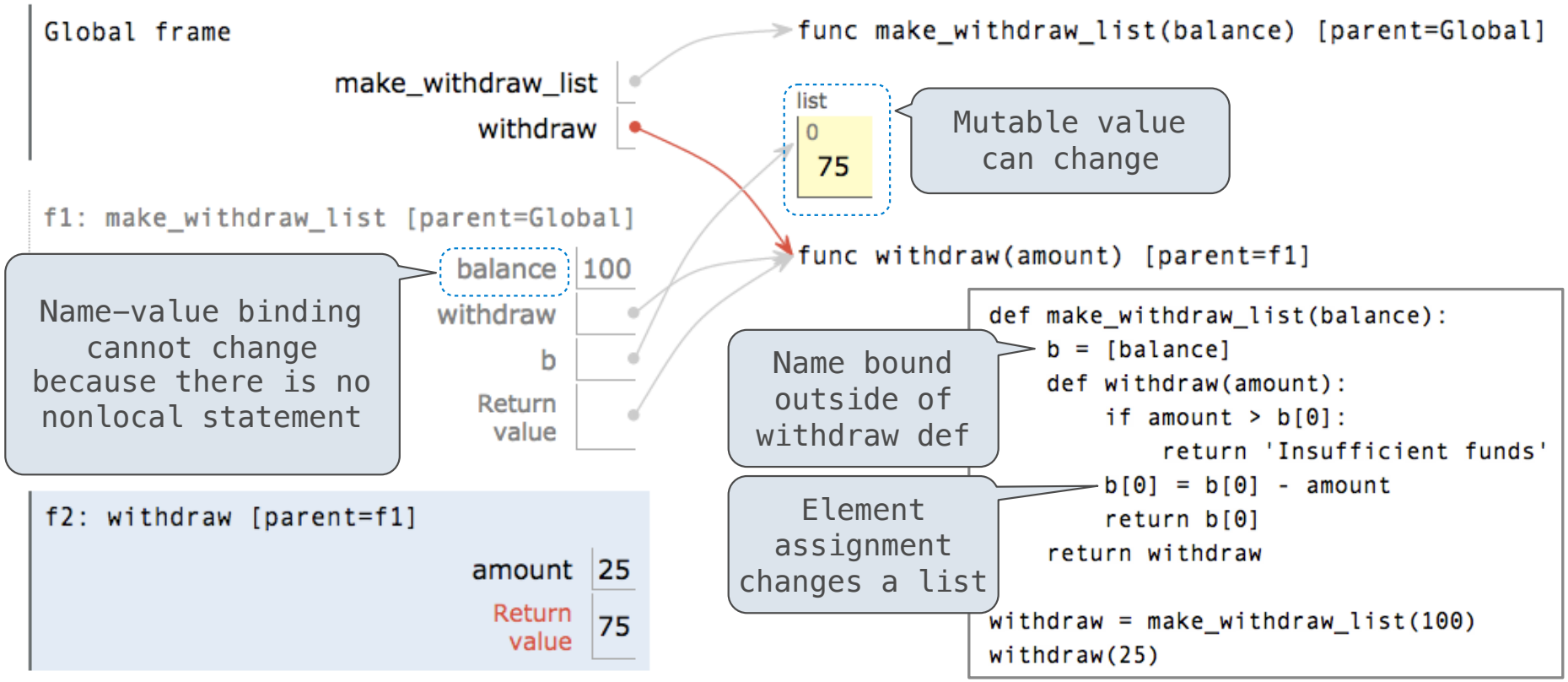
Mutable values can be changed *without* a nonlocal statement.



goo.gl/y4TyFZ

Mutable Values & Persistent Local State

Mutable values can be changed *without* a nonlocal statement.



goo.gl/y4TyFZ

Multiple Mutable Functions

(Demo)

Referential Transparency, Lost

pythontutor.com/composingprograms.html#code=def%20f%28x%29%3A%0A%20%20%20%20x%20%3D%204%0A%20%20%20%20def%20g%28y%29%3A%0A%20%20%20%20%20%20%20%20def%20h%28z%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20nonlocal%20x%0A%20%20%20%20%20%20%20%20%20%20%20%20x%20%3D%20x%20%2B%201%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20x%20%2B%20y%20%2B%20z%0A%20%20%20%20%20%20%20%20return%20h%0A%20%20%20%20return%20g%0Aa%20%3D%20f%281%29%0Ab%20%3D%20a%282%29%0Atotal%20%3D%20b%283%29%20%2B%20b%284%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Referential Transparency, Lost

- Expressions are **referentially transparent** if substituting an expression with its value does not change the meaning of a program.

pythontutor.com/composingprograms.html#code=def%20f%28x%29%3A%0A%20%20%20%20x%20%3D%204%0A%20%20%20%20def%20g%28y%29%3A%0A%20%20%20%20%20%20%20%20def%20h%28z%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20nonlocal%20x%0A%20%20%20%20%20%20%20%20%20%20%20%20x%20%3D%20x%20%2B%201%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20x%20%2B%20y%20%2B%20z%0A%20%20%20%20%20%20%20%20return%20h%0A%20%20%20%20return%20g%0Aa%20%3D%20f%281%29%0Ab%20%3D%20a%282%29%0Atotal%20%3D%20b%283%29%20%2B%20b%284%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Referential Transparency, Lost

- Expressions are **referentially transparent** if substituting an expression with its value does not change the meaning of a program.

```
mul(add(2, mul(4, 6)), add(3, 5))
```


pythontutor.com/composingprograms.html#code=def%20f%28x%29%3A%0A%20%20%20%20x%20%3D%204%0A%20%20%20%20def%20g%28y%29%3A%0A%20%20%20%20%20%20%20%20def%20h%28z%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20nonlocal%20x%0A%20%20%20%20%20%20%20%20%20%20%20%20x%20%3D%20x%20%2B%201%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20x%20%2B%20y%20%2B%20z%0A%20%20%20%20%20%20%20%20return%20h%0A%20%20%20%20return%20g%0Aa%20%3D%20f%281%29%0Ab%20%3D%20a%282%29%0Atotal%20%3D%20b%283%29%20%2B%20b%284%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Referential Transparency, Lost

- Expressions are **referentially transparent** if substituting an expression with its value does not change the meaning of a program.

```
mul(add(2, mul(4, 6)), add(3, 5))

mul(add(2,    24    ), add(3, 5))
```


pythontutor.com/composingprograms.html#code=def%20f%28x%29%3A%0A%20%20%20%20x%20%3D%204%0A%20%20%20%20def%20g%28y%29%3A%0A%20%20%20%20%20%20%20%20def%20h%28z%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20nonlocal%20x%0A%20%20%20%20%20%20%20%20%20%20%20%20x%20%3D%20x%20%2B%201%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20x%20%2B%20y%20%2B%20z%0A%20%20%20%20%20%20%20%20return%20h%0A%20%20%20%20return%20g%0Aa%20%3D%20f%281%29%0Ab%20%3D%20a%282%29%0Atotal%20%3D%20b%283%29%20%2B%20b%284%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Referential Transparency, Lost

- Expressions are **referentially transparent** if substituting an expression with its value does not change the meaning of a program.

```
mul(add(2, mul(4, 6)), add(3, 5))

mul(add(2,    24    ), add(3, 5))

mul(       26        , add(3, 5))
```


pythontutor.com/composingprograms.html#code=def%20f%28x%29%3A%0A%20%20%20%20x%20%3D%204%0A%20%20%20%20def%20g%28y%29%3A%0A%20%20%20%20%20%20%20%20def%20h%28z%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20nonlocal%20x%0A%20%20%20%20%20%20%20%20%20%20%20%20x%20%3D%20x%20%2B%201%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20x%20%2B%20y%20%2B%20z%0A%20%20%20%20%20%20%20%20return%20h%0A%20%20%20%20return%20g%0Aa%20%3D%20f%281%29%0Ab%20%3D%20a%282%29%0Atotal%20%3D%20b%283%29%20%2B%20b%284%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Referential Transparency, Lost

- Expressions are **referentially transparent** if substituting an expression with its value does not change the meaning of a program.

```
mul(add(2, mul(4, 6)), add(3, 5))

mul(add(2,    24    ), add(3, 5))

mul(       26        , add(3, 5))
```

- Mutation operations violate the condition of referential transparency because they do more than just return a value; **they change the environment.**

pythontutor.com/composingprograms.html#code=def%20f%28x%29%3A%0A%20%20%20%20x%20%3D%204%0A%20%20%20%20def%20g%28y%29%3A%0A%20%20%20%20%20%20%20%20def%20h%28z%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20nonlocal%20x%0A%20%20%20%20%20%20%20%20%20%20%20%20x%20%3D%20x%20%2B%201%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20x%20%2B%20y%20%2B%20z%0A%20%20%20%20%20%20%20%20return%20h%0A%20%20%20%20return%20g%0Aa%20%3D%20f%281%29%0Ab%20%3D%20a%282%29%0Atotal%20%3D%20b%283%29%20%2B%20b%284%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Referential Transparency, Lost

- Expressions are **referentially transparent** if substituting an expression with its value does not change the meaning of a program.

```
mul(add(2, mul(4, 6)), add(3, 5))

mul(add(2,    24    ), add(3, 5))

mul(       26        , add(3, 5))
```

- Mutation operations violate the condition of referential transparency because they do more than just return a value; **they change the environment.**

pythontutor.com/composingprograms.html#code=def%20f%28x%29%3A%0A%20%20%20%20x%20%3D%204%0A%20%20%20%20def%20g%28y%29%3A%0A%20%20%20%20%20%20%20%20def%20h%28z%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20nonlocal%20x%0A%20%20%20%20%20%20%20%20%20%20%20%20x%20%3D%20x%20%2B%201%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20x%20%2B%20y%20%2B%20z%0A%20%20%20%20%20%20%20%20return%20h%0A%20%20%20%20return%20g%0A%20%3D%20f%281%29%0Ab%20%3D%20a%282%29%0Atotal%20%3D%20b%283%29%20%2B%20b%284%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Referential Transparency, Lost

- Expressions are **referentially transparent** if substituting an expression with its value does not change the meaning of a program.

```
mul(add(2, mul(4, 6)), add(3, 5))

mul(add(2,    24    ), add(3, 5))

mul(       26        , add(3, 5))
```

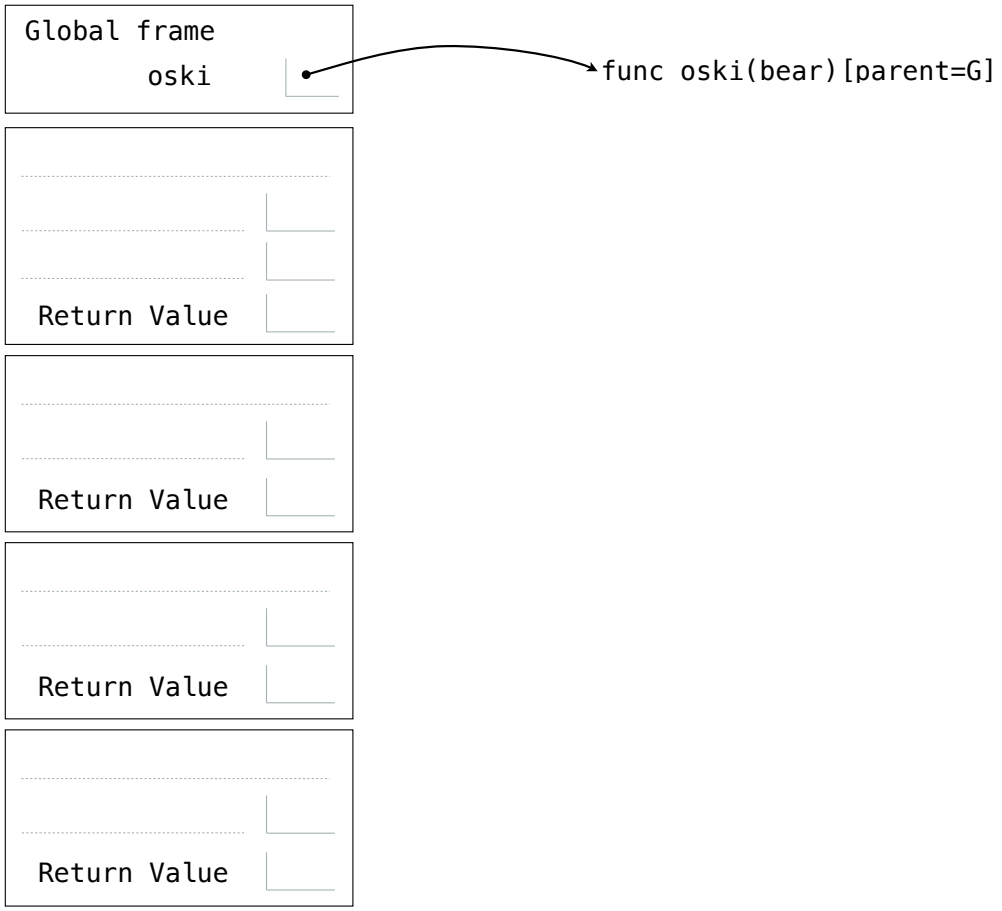
- Mutation operations violate the condition of referential transparency because they do more than just return a value; **they change the environment.**

pythontutor.com/composingprograms.html#code=def%20f%28x%29%3A%0A%20%20%20%20x%20%3D%204%0A%20%20%20%20def%20g%28y%29%3A%0A%20%20%20%20%20%20%20%20def%20h%28z%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20nonlocal%20x%0A%20%20%20%20%20%20%20%20%20%20%20%20x%20%3D%20x%20%2B%201%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20x%20%2B%20y%20%2B%20z%0A%20%20%20%20%20%20%20%20return%20h%0A%20%20%20%20return%20g%0A%20%3D%20f%281%29%0Ab%20%3D%20a%282%29%0Atotal%20%3D%20b%283%29%20%2B%20b%284%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Environment Diagrams

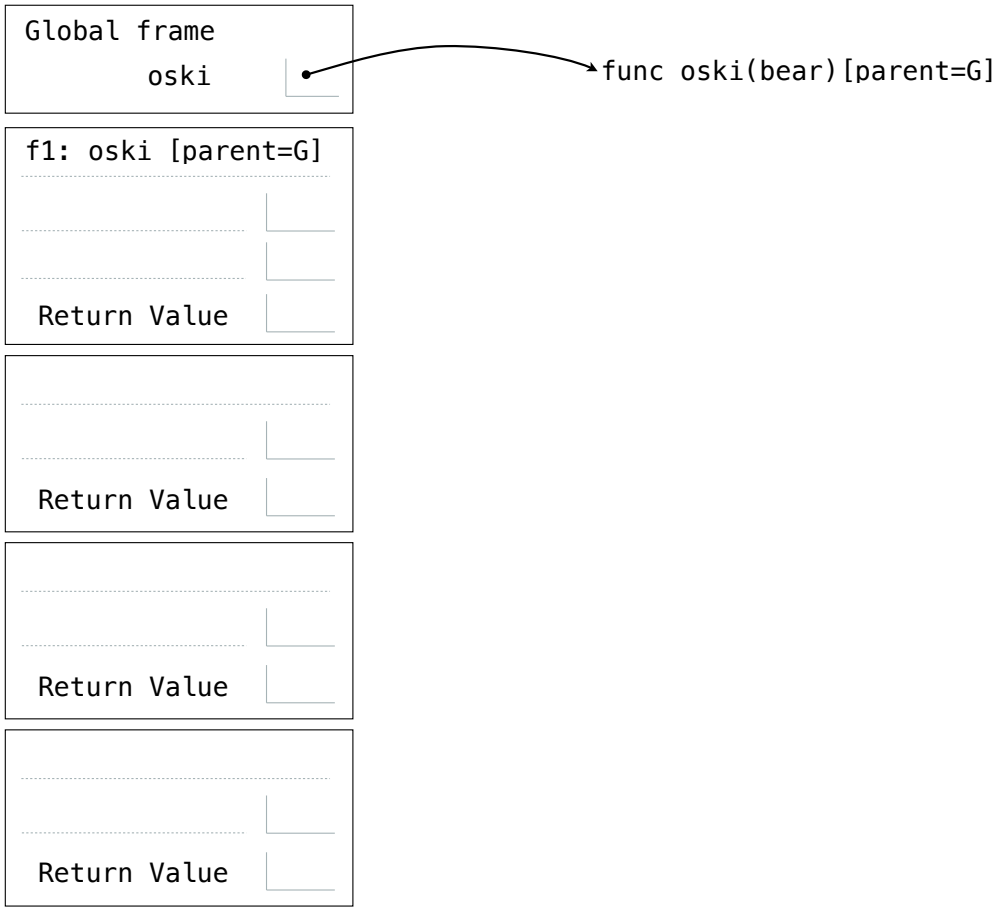
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



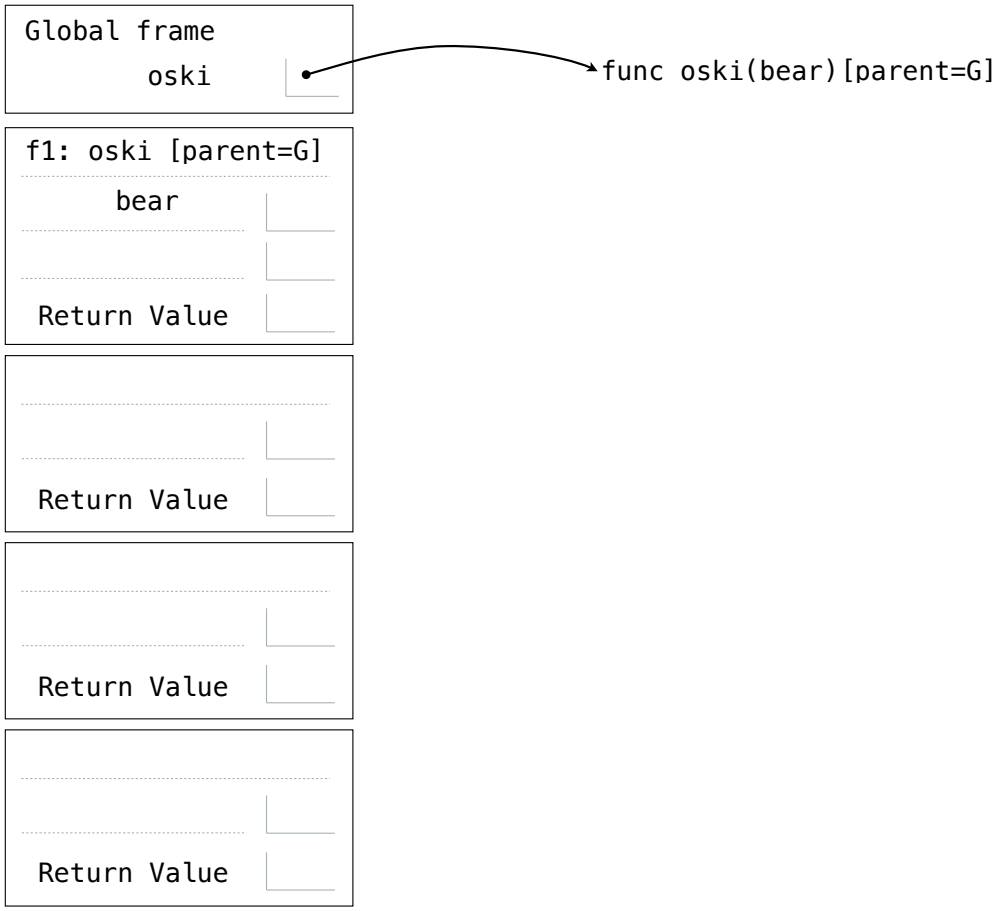
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



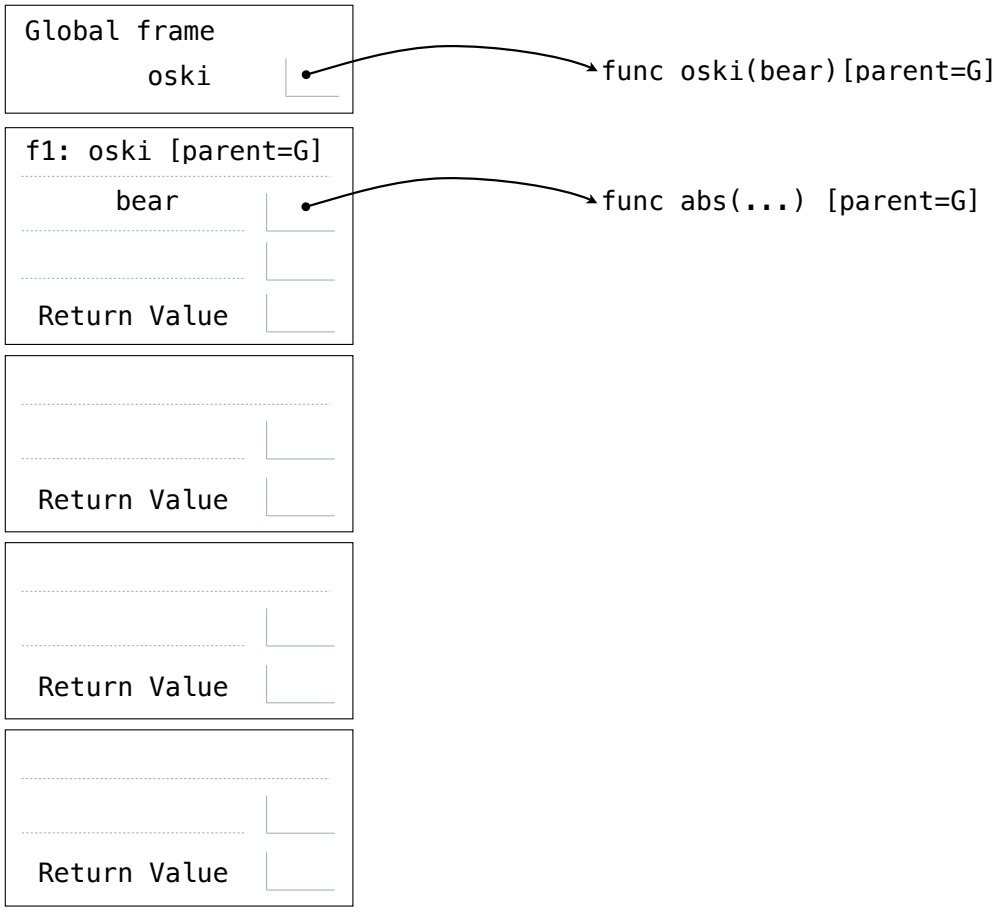
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



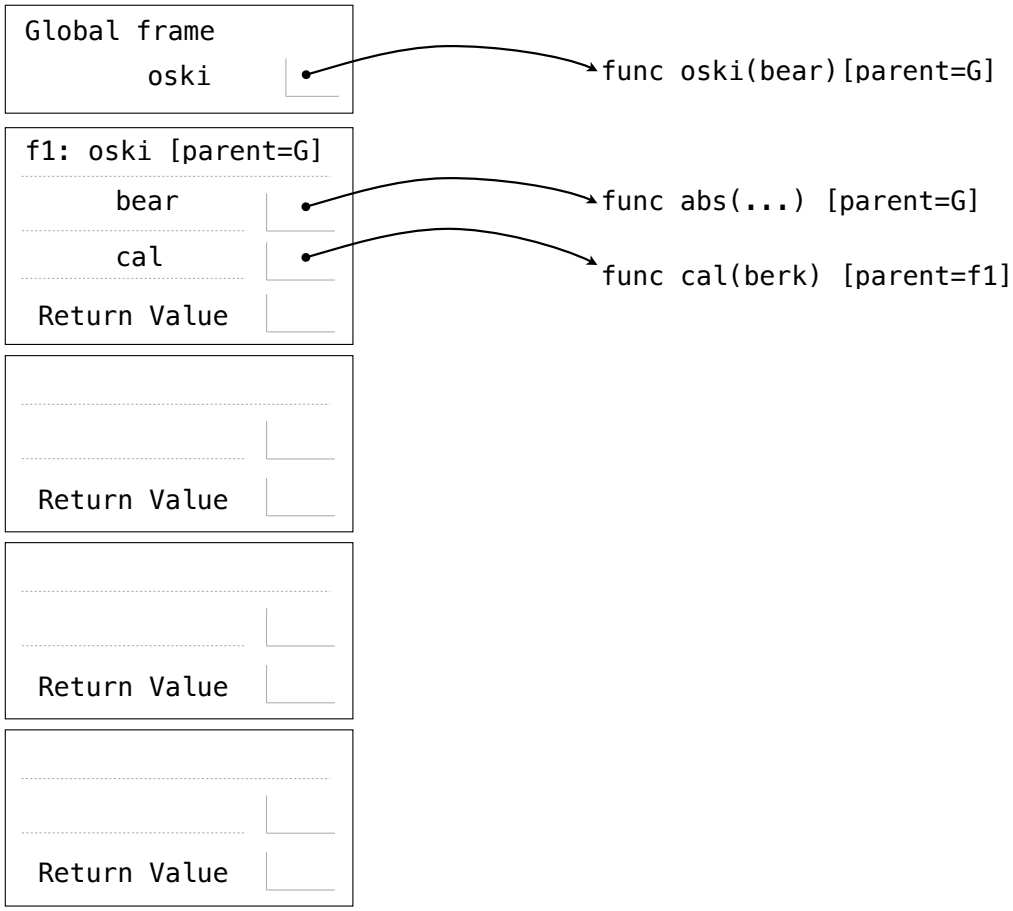
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



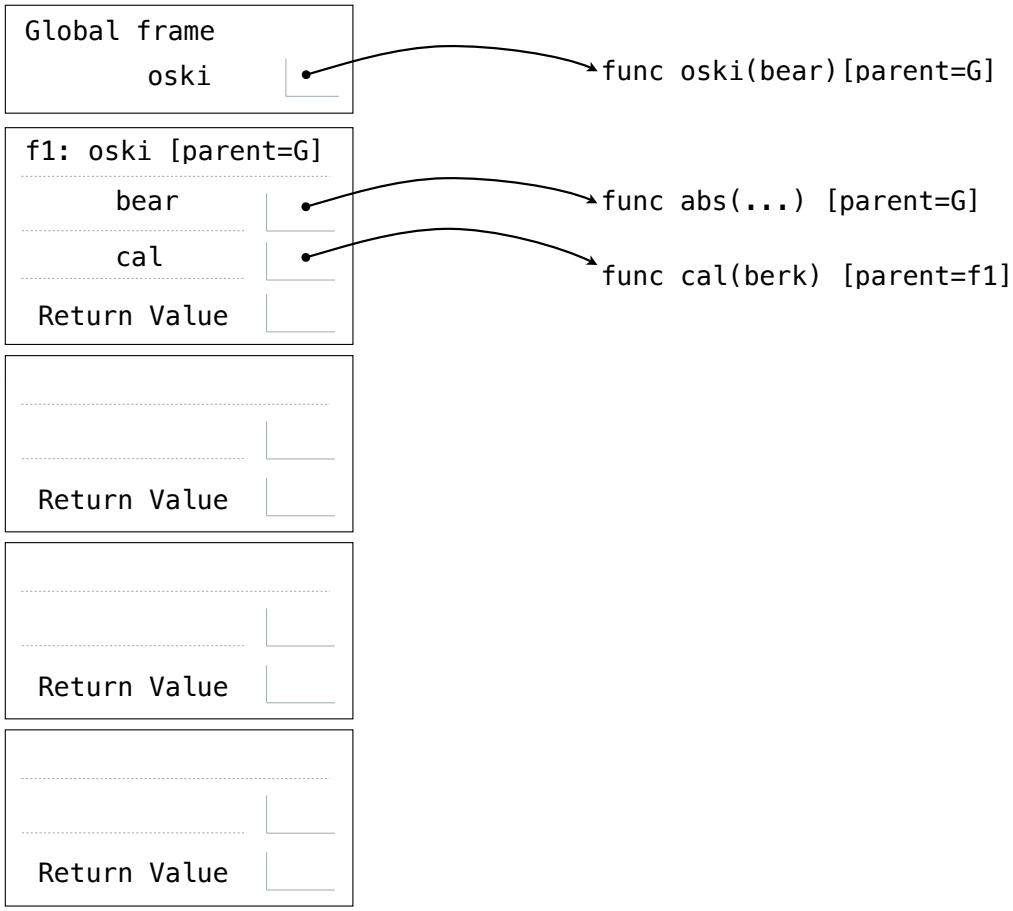
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



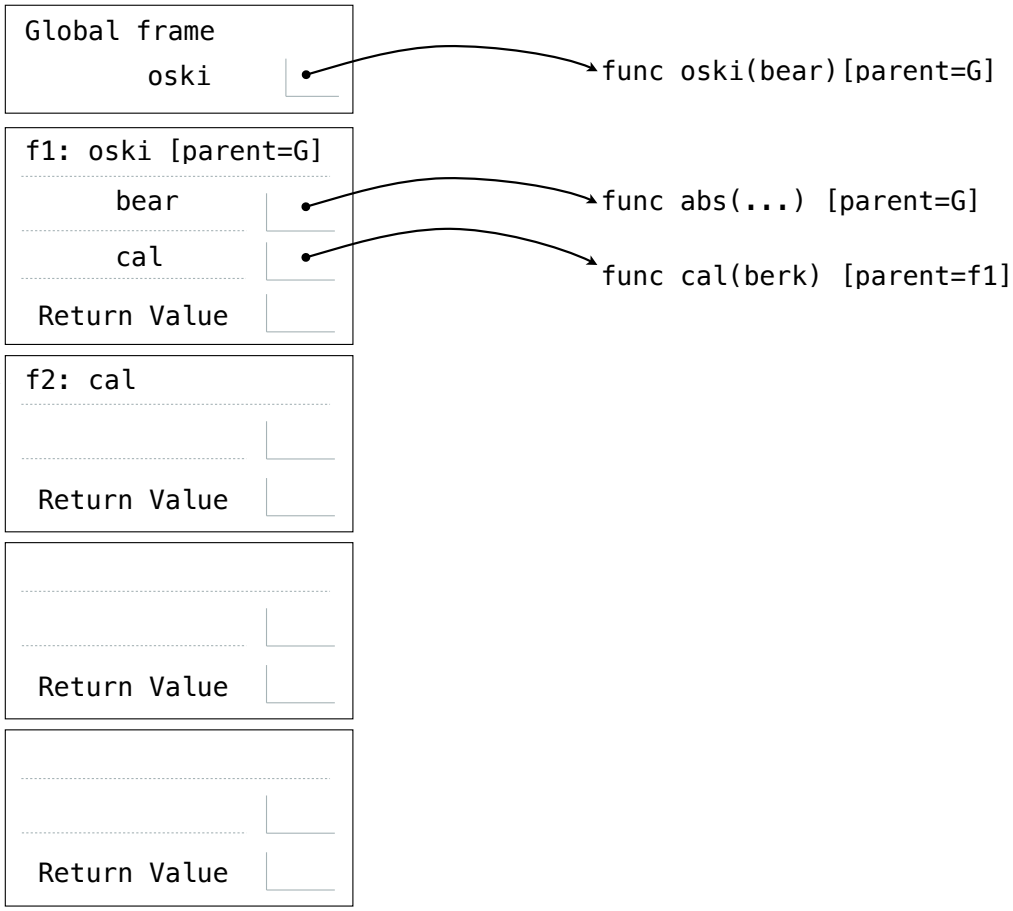
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



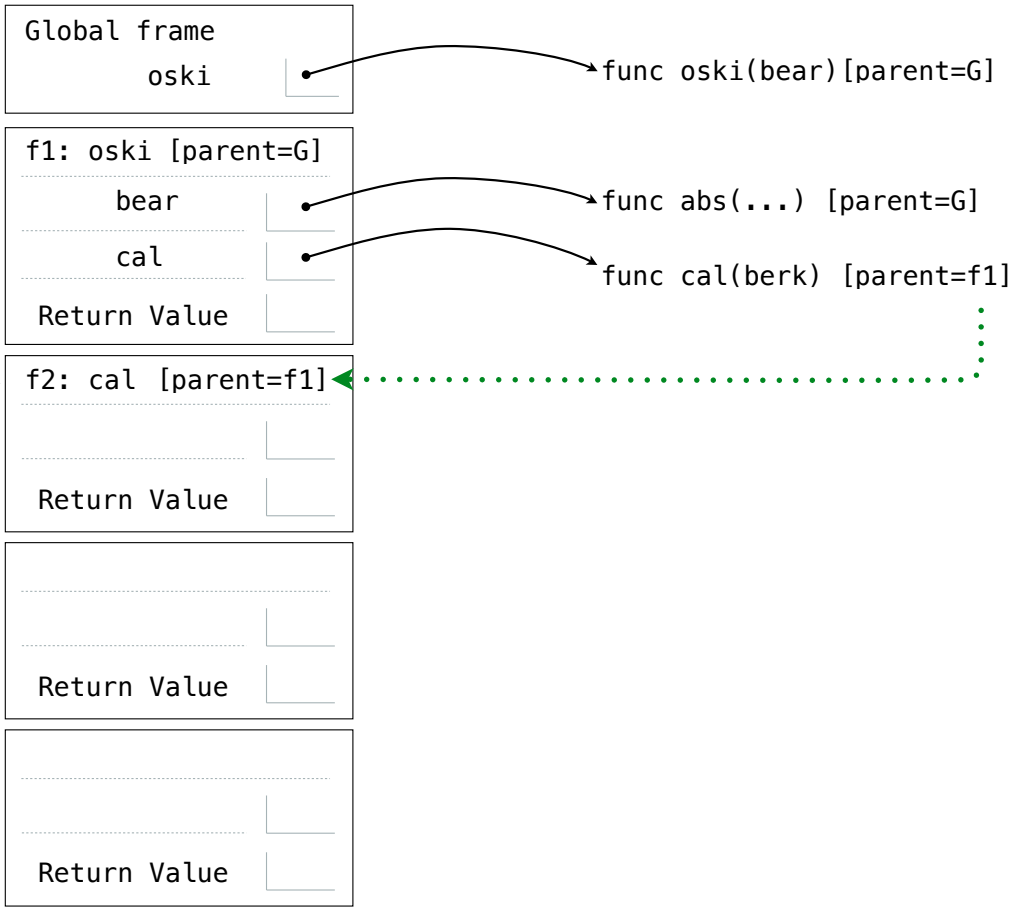
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



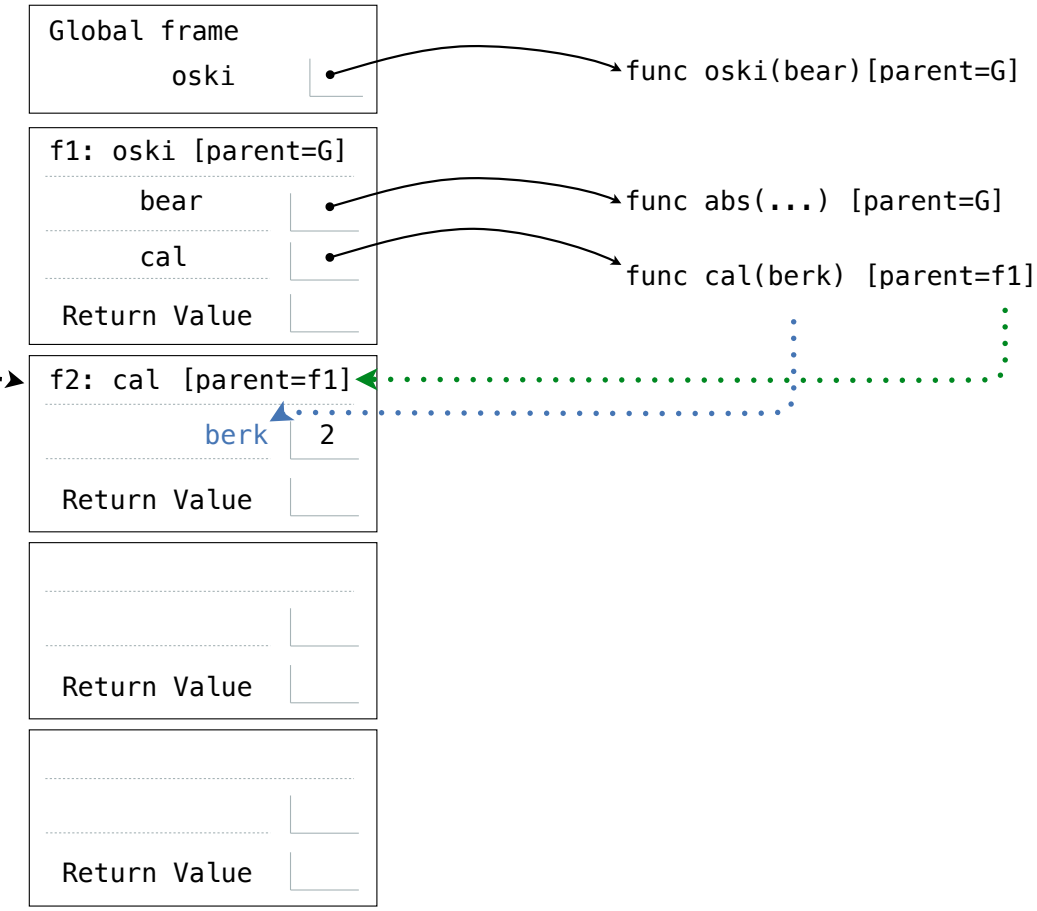
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



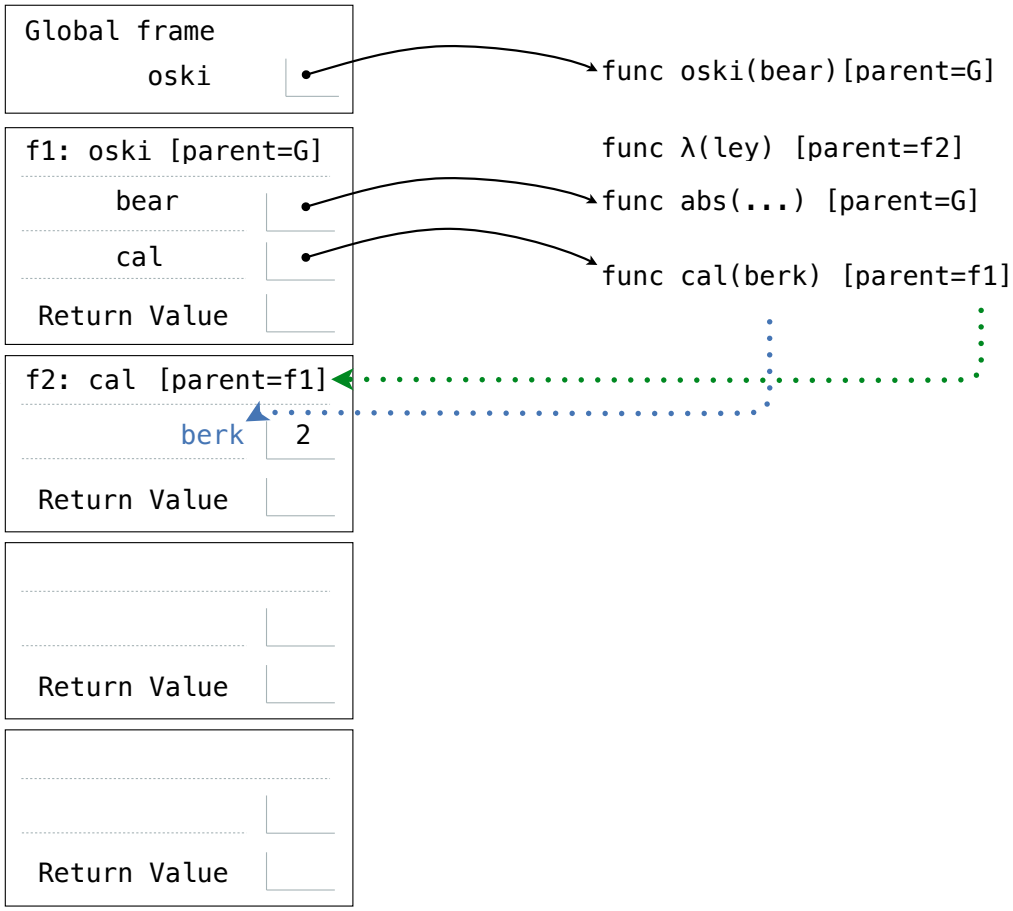
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



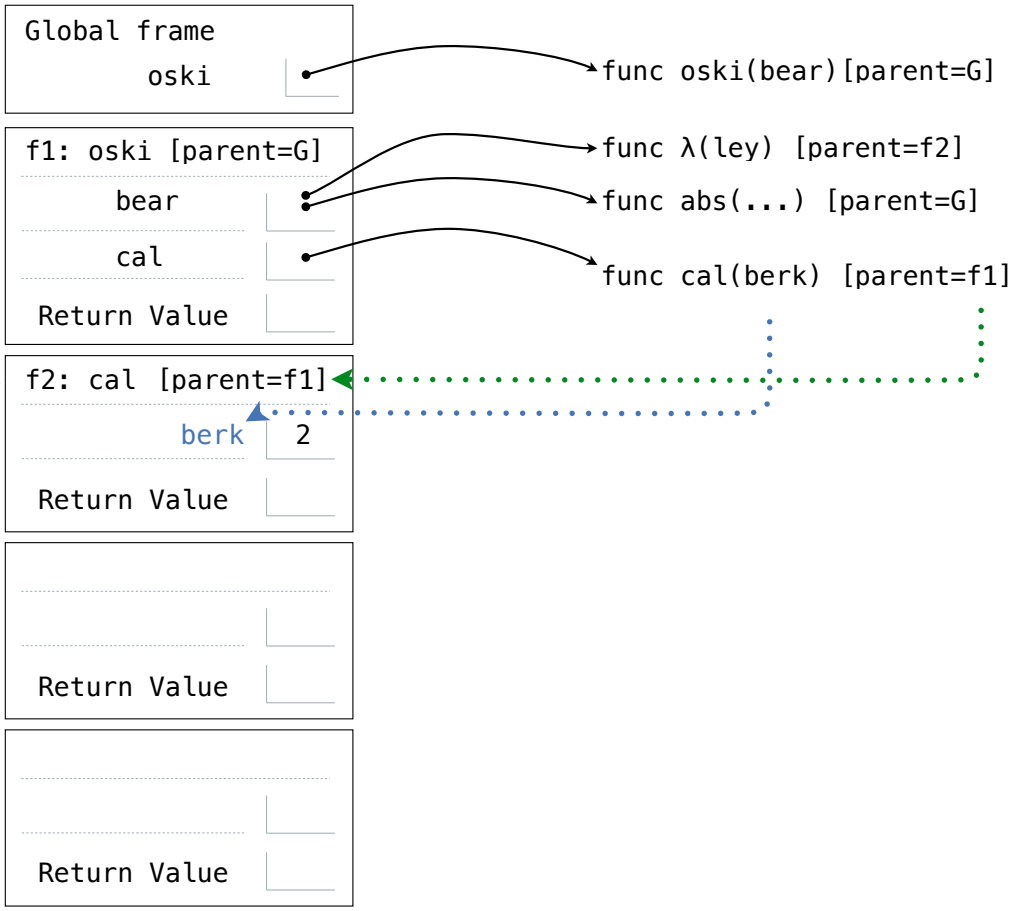
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



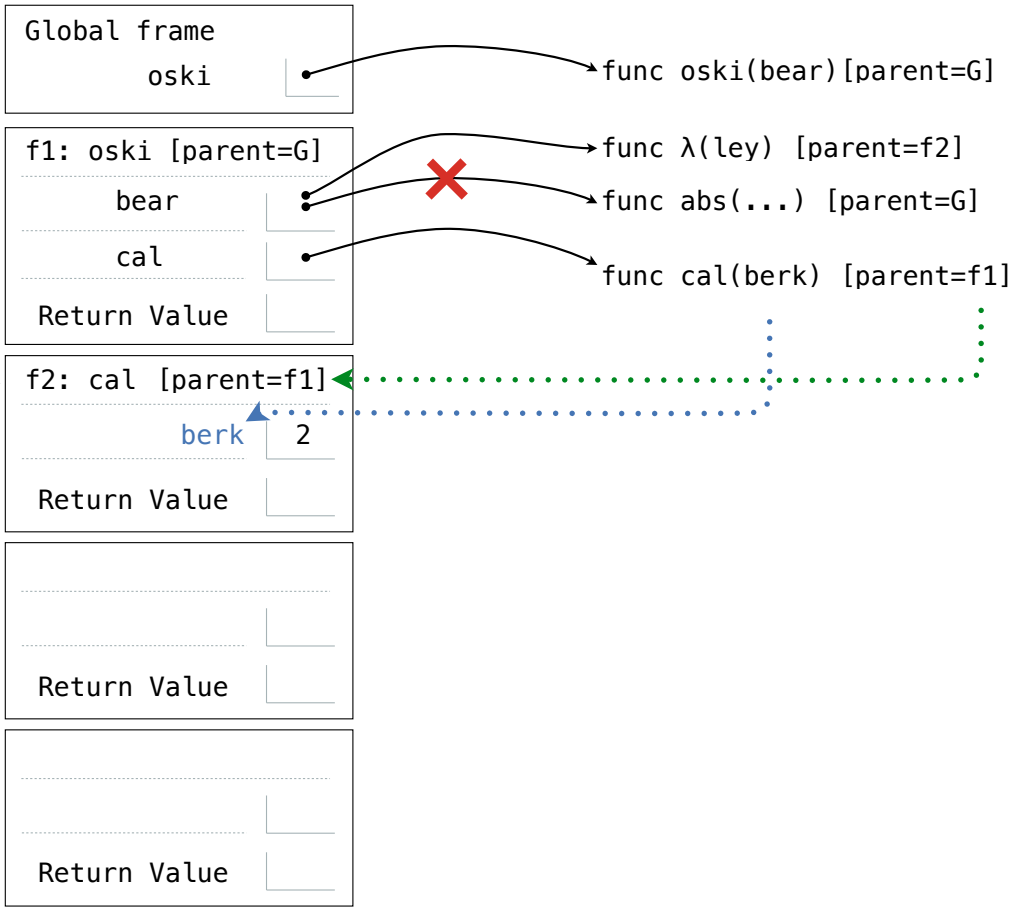
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



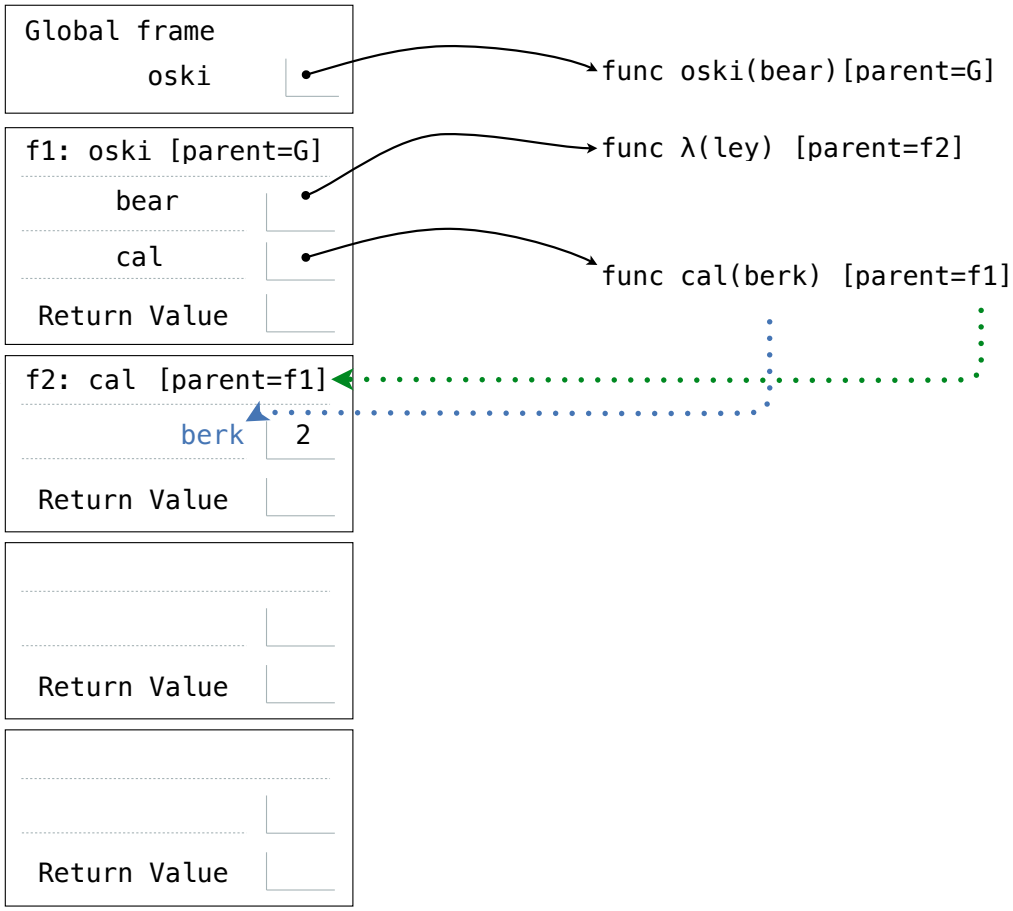
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



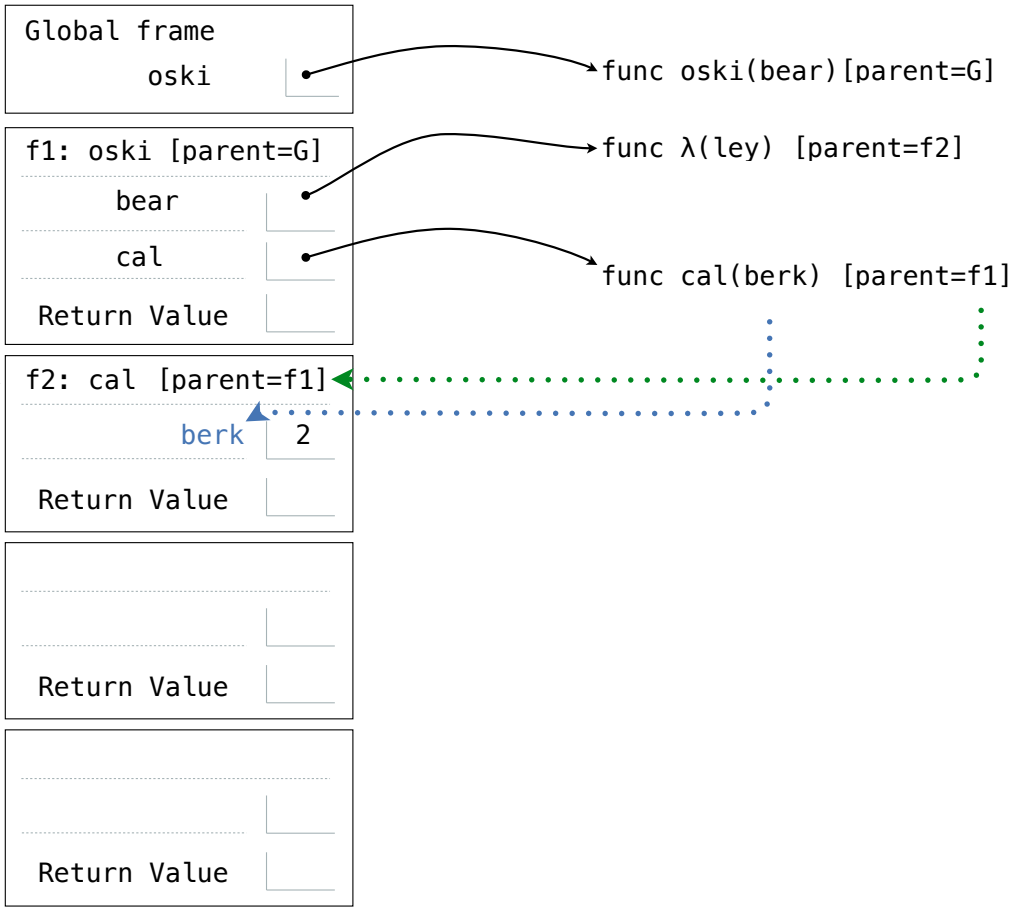
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



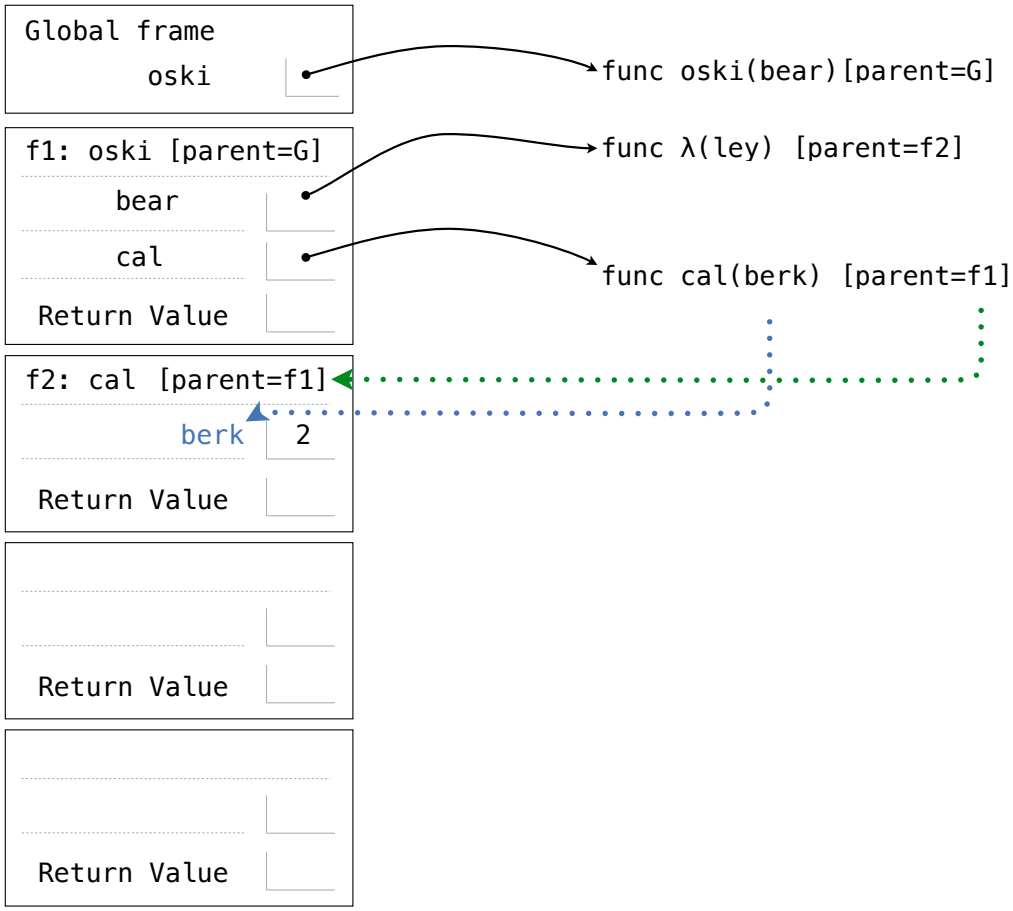
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



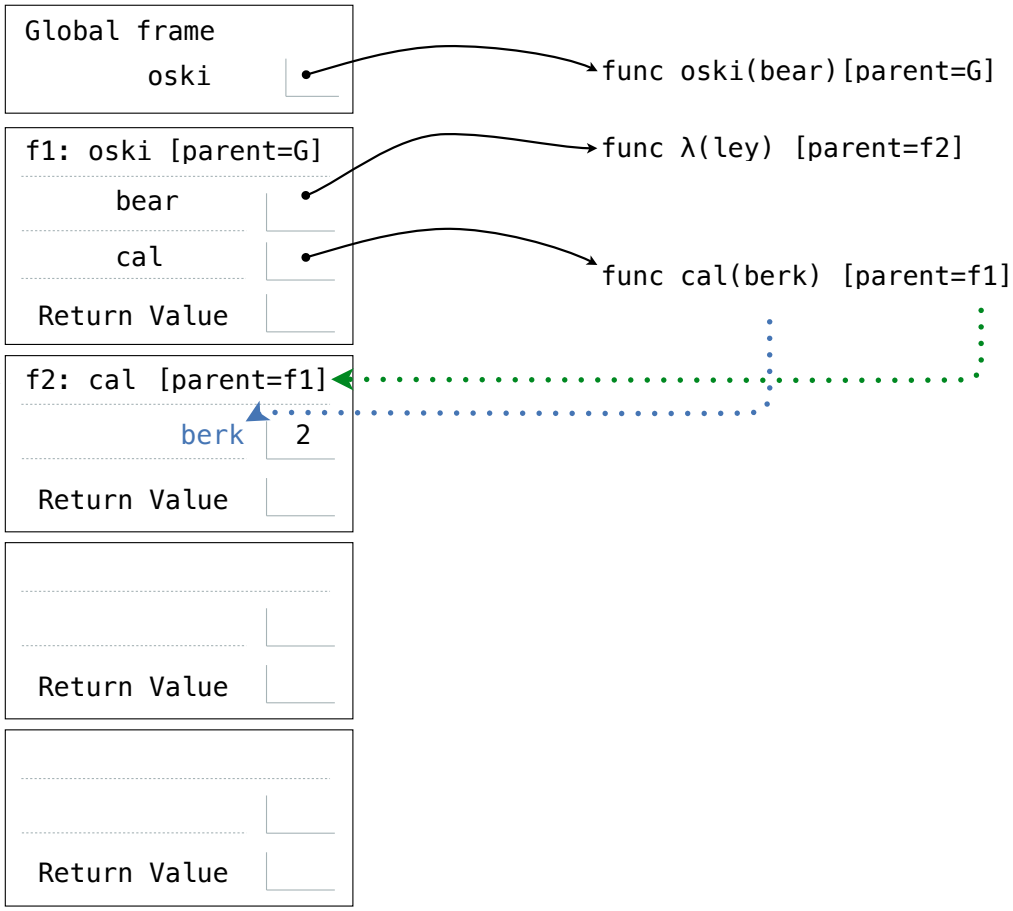
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



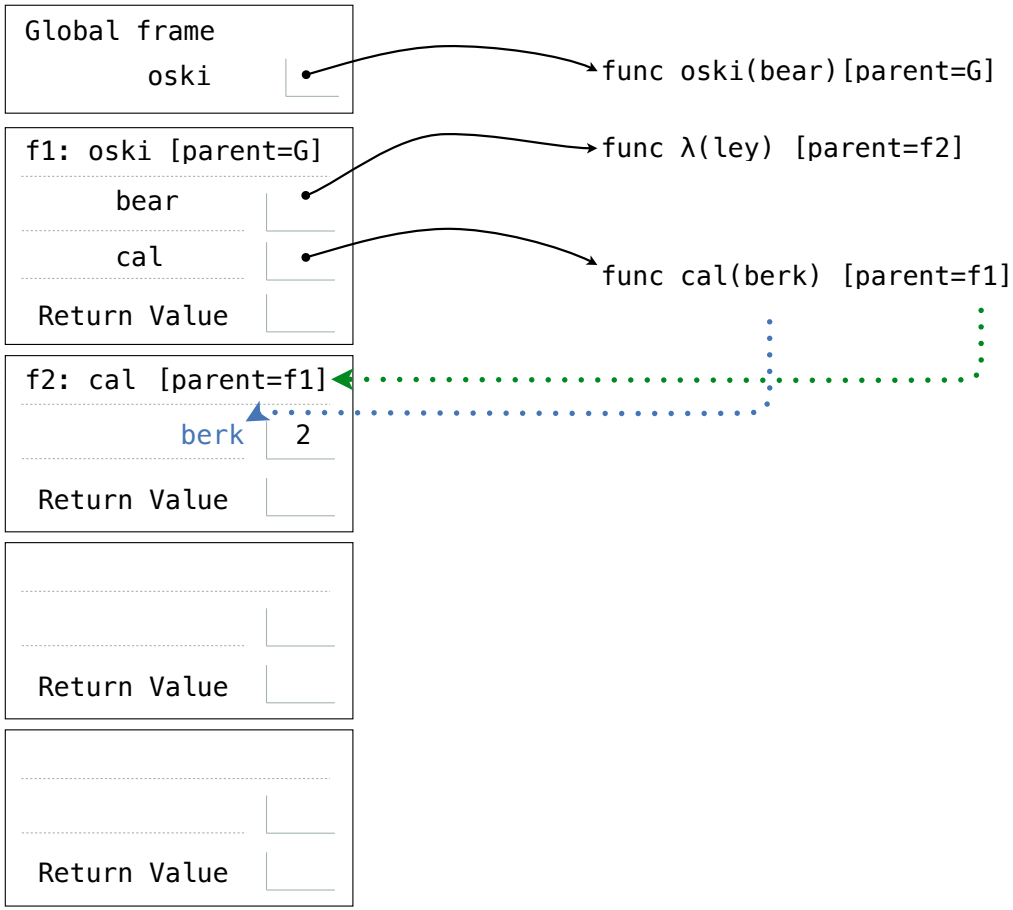
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



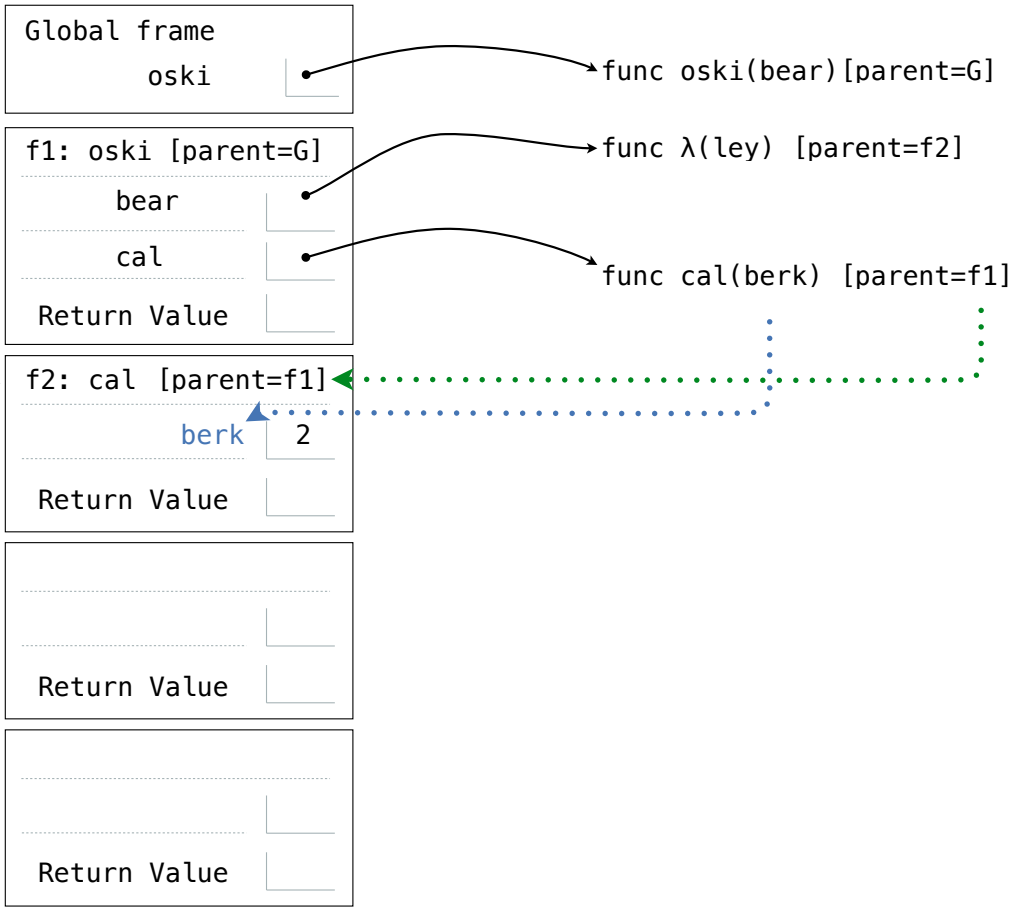
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



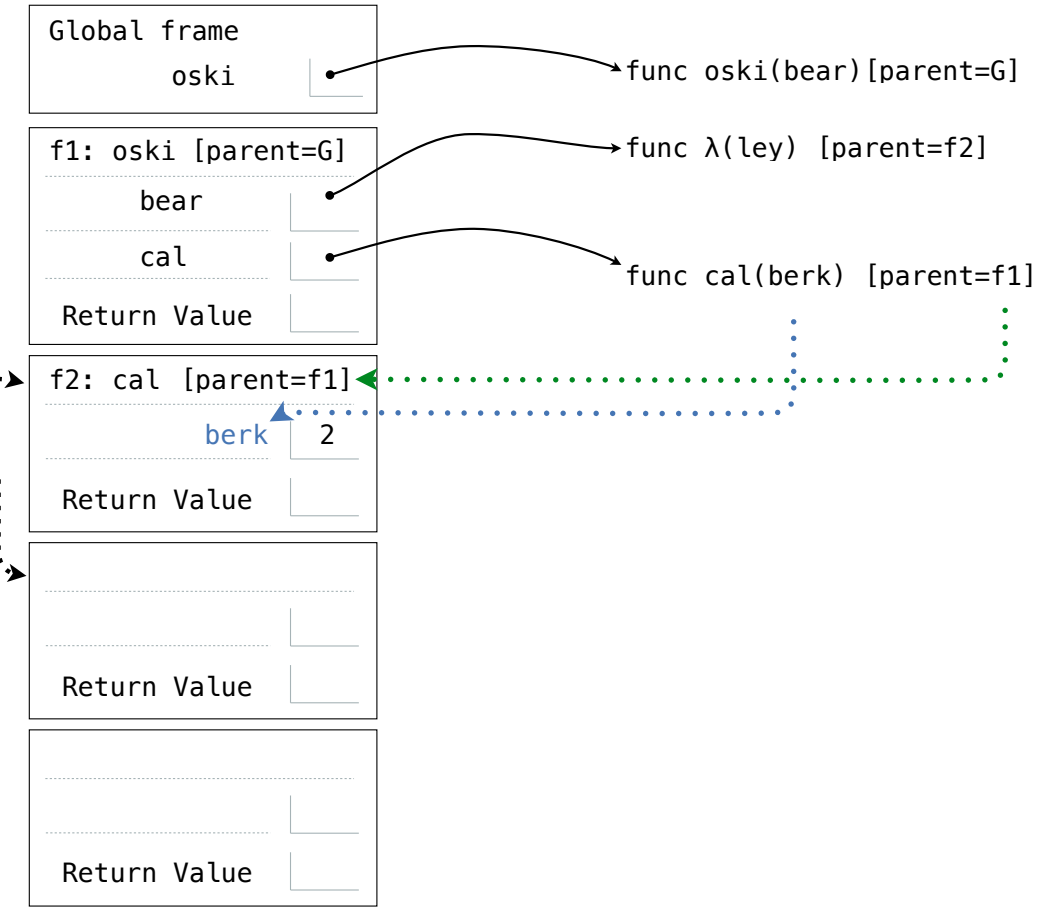
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



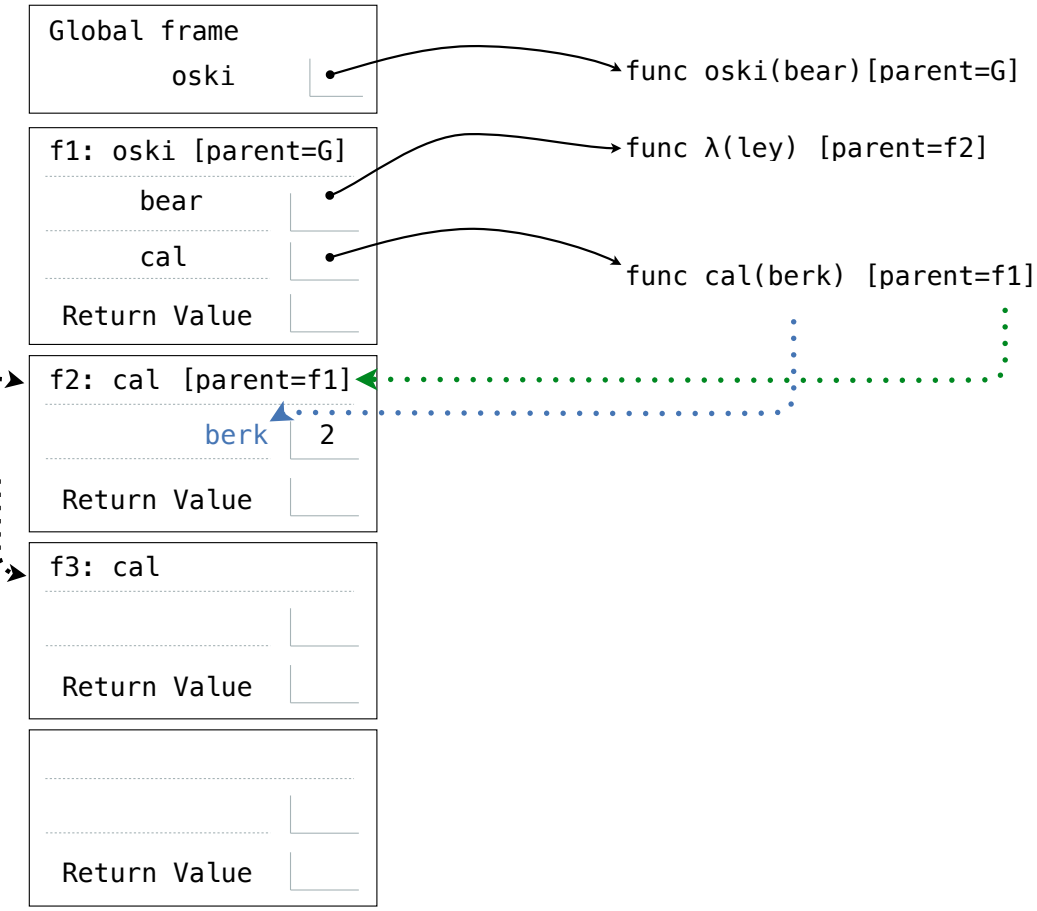
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



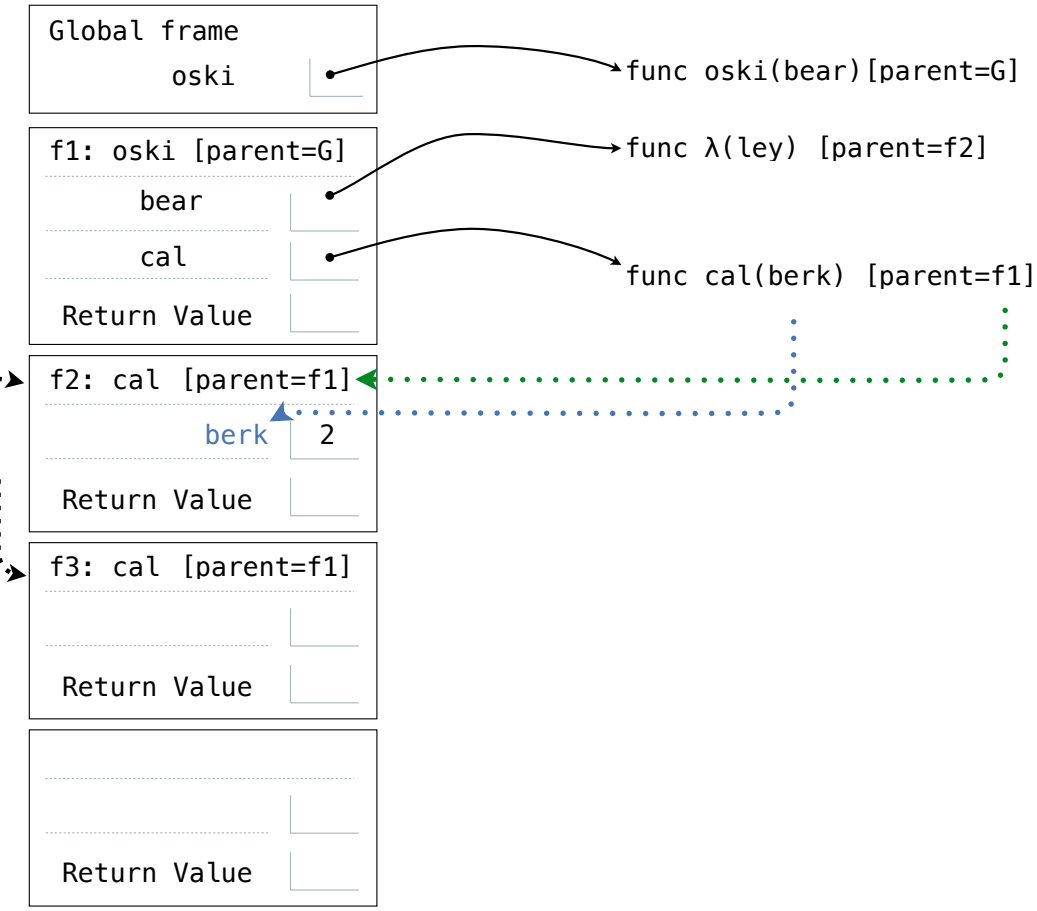
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



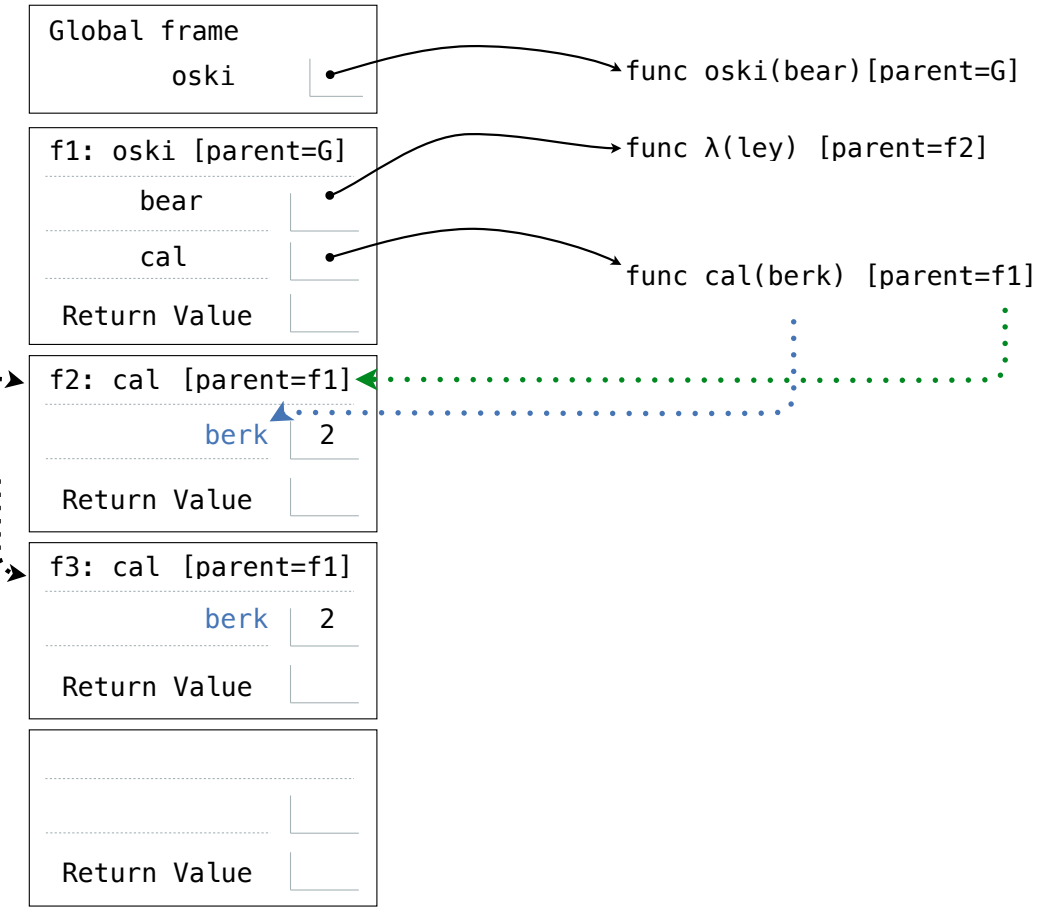
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



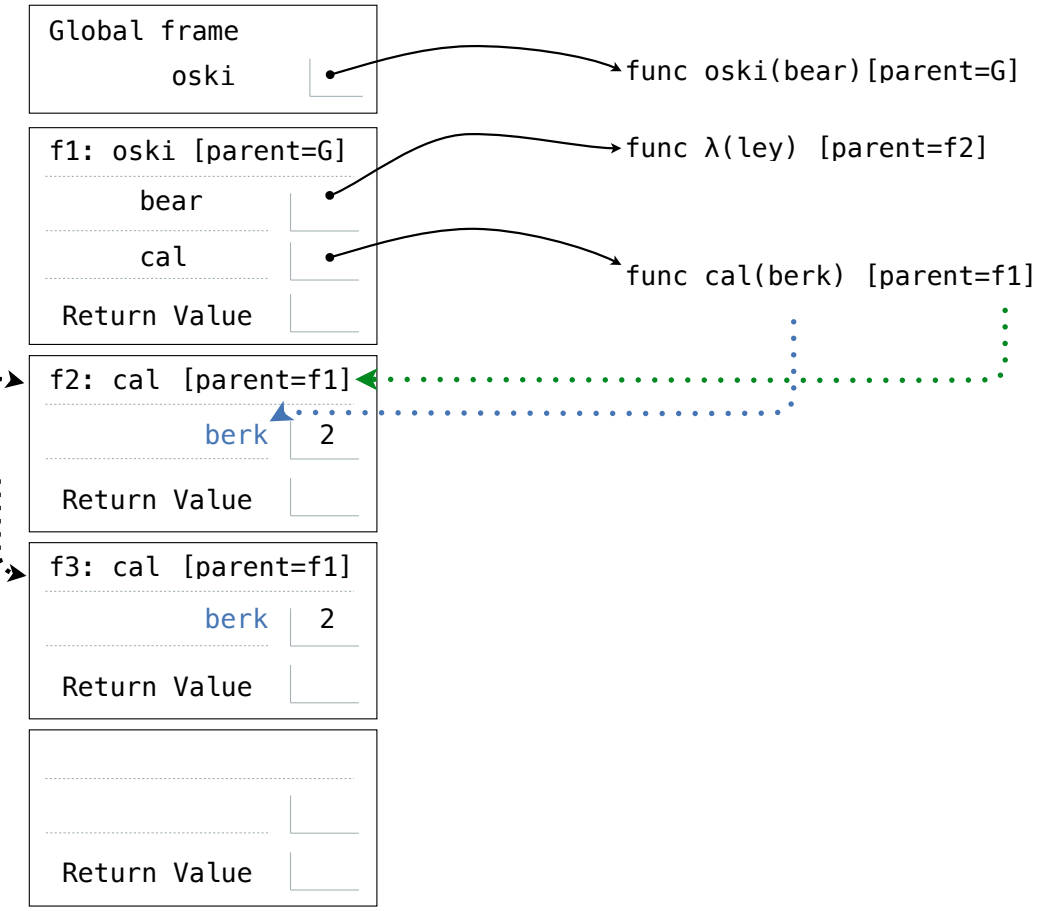
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



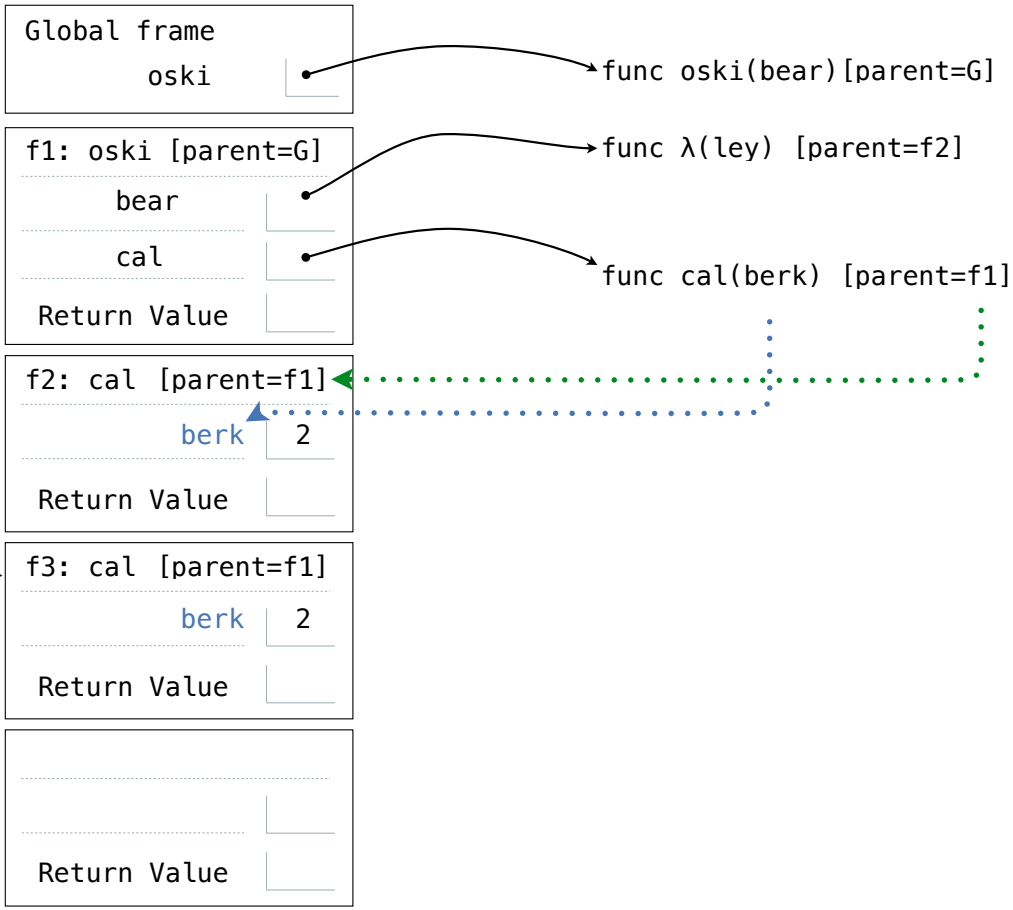
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



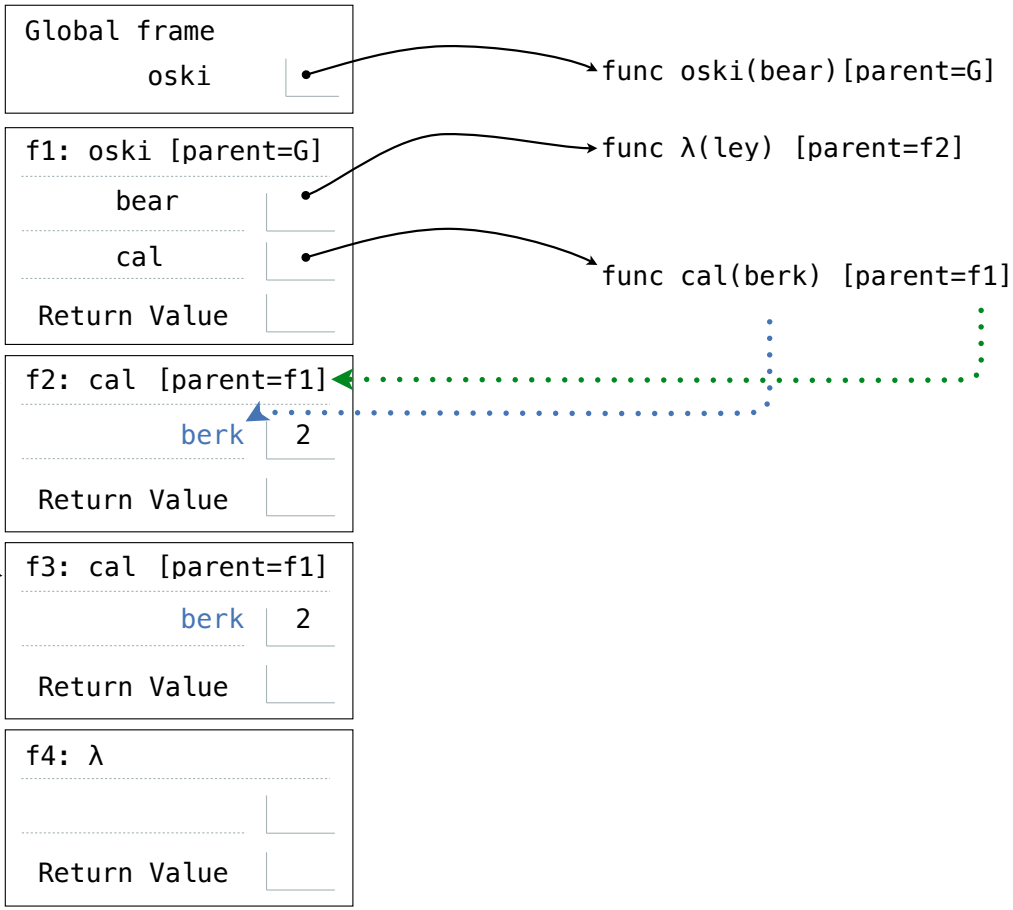
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



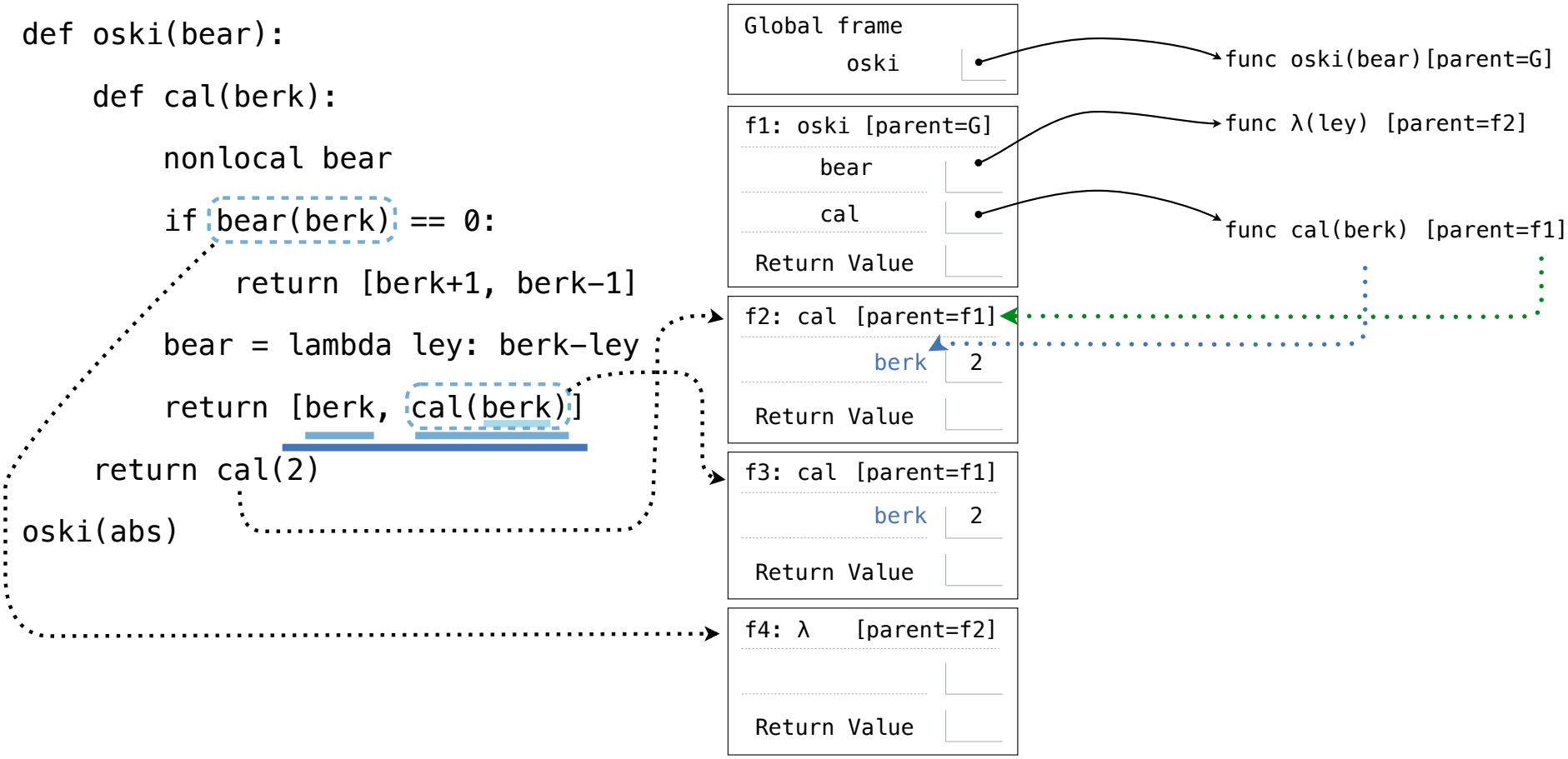
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



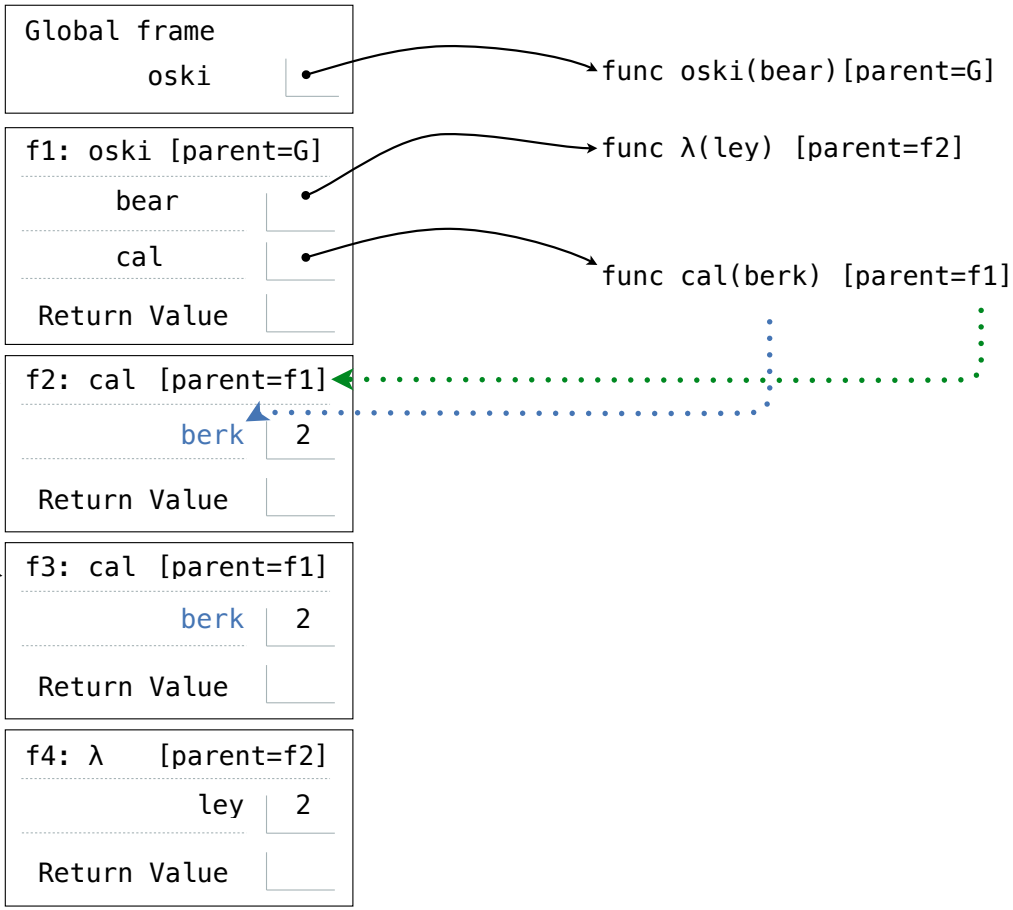
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



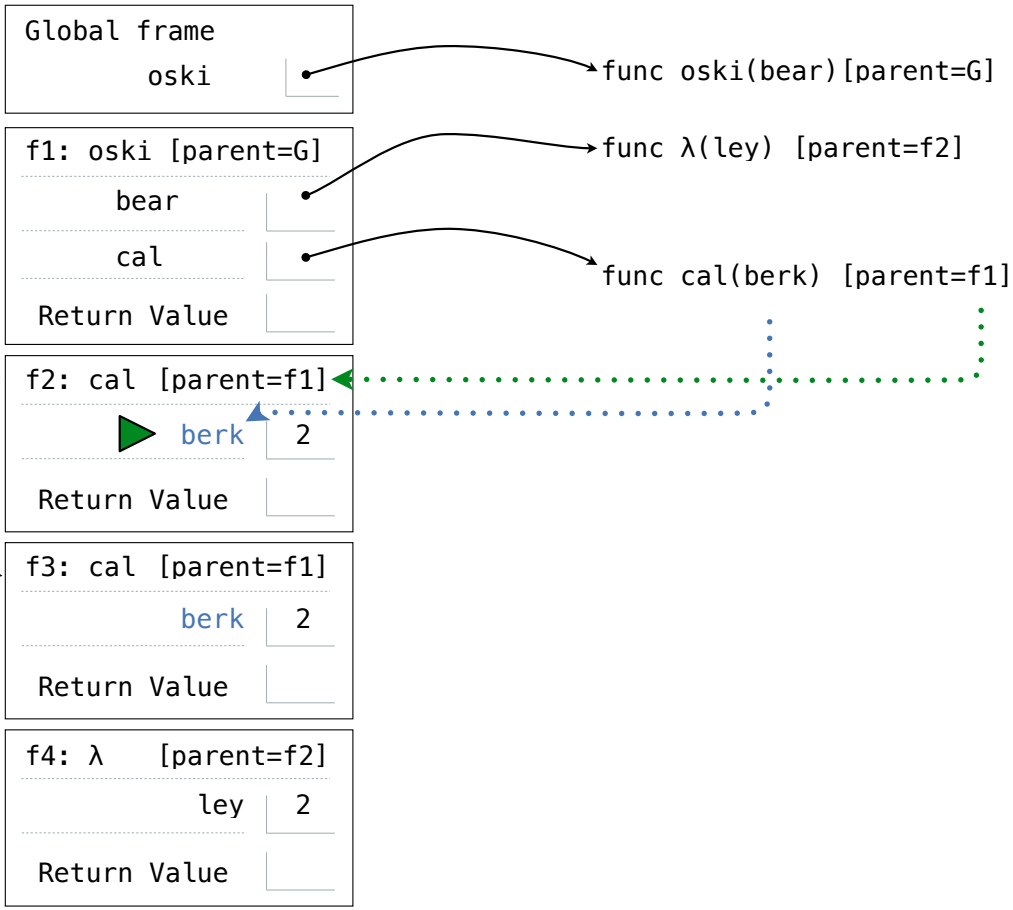
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



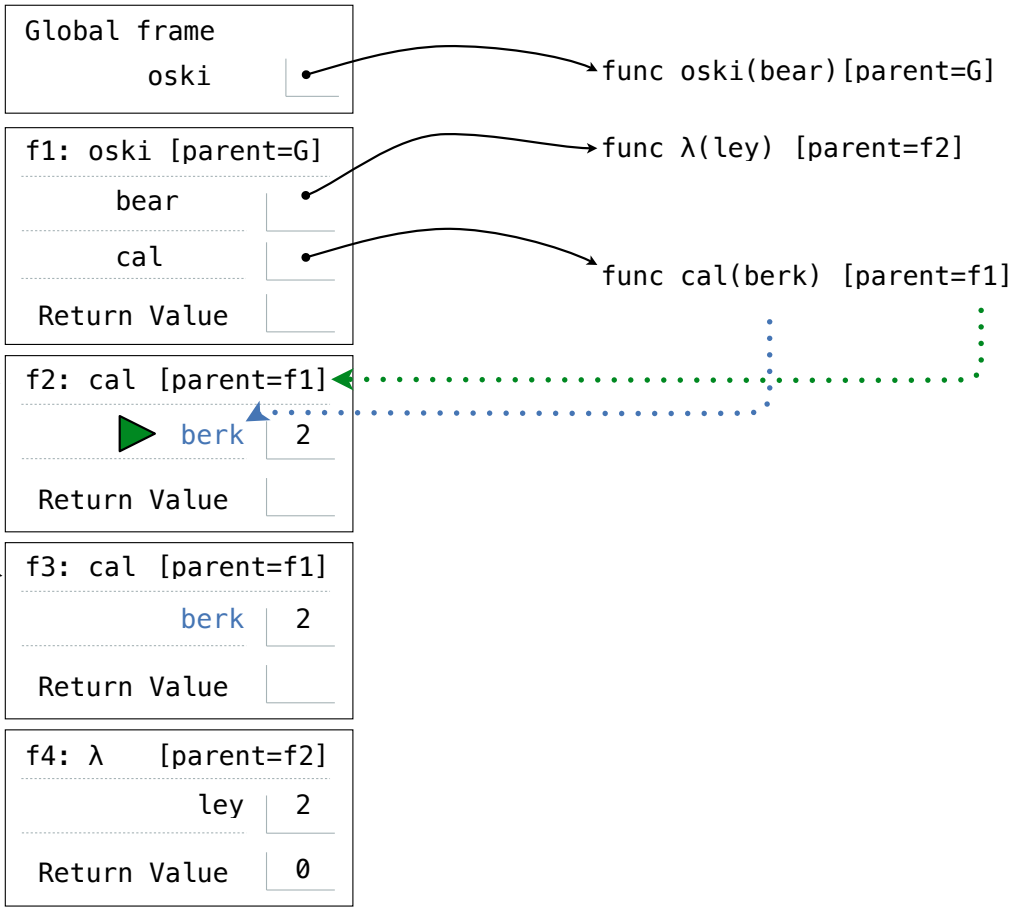
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



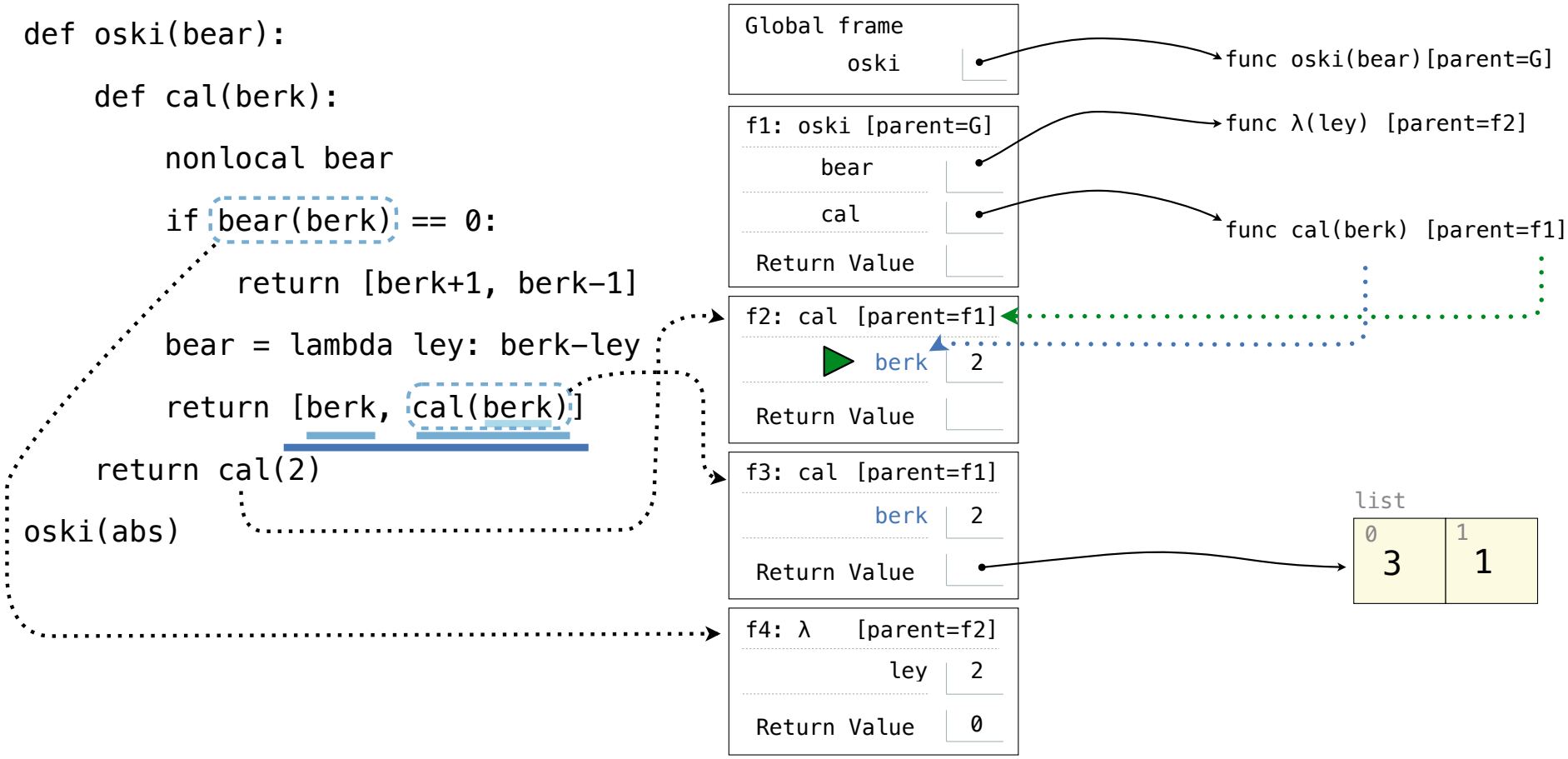
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



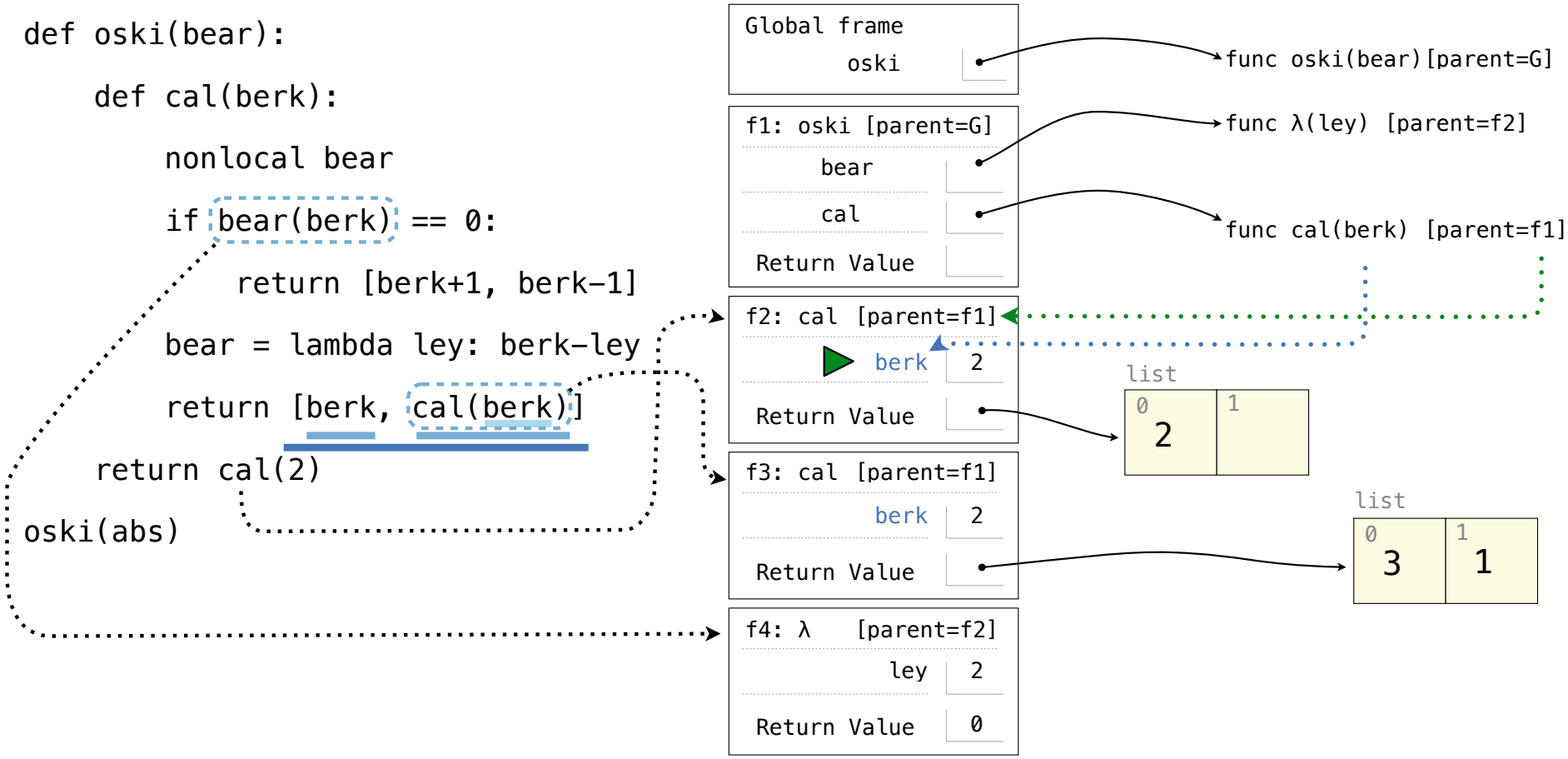
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



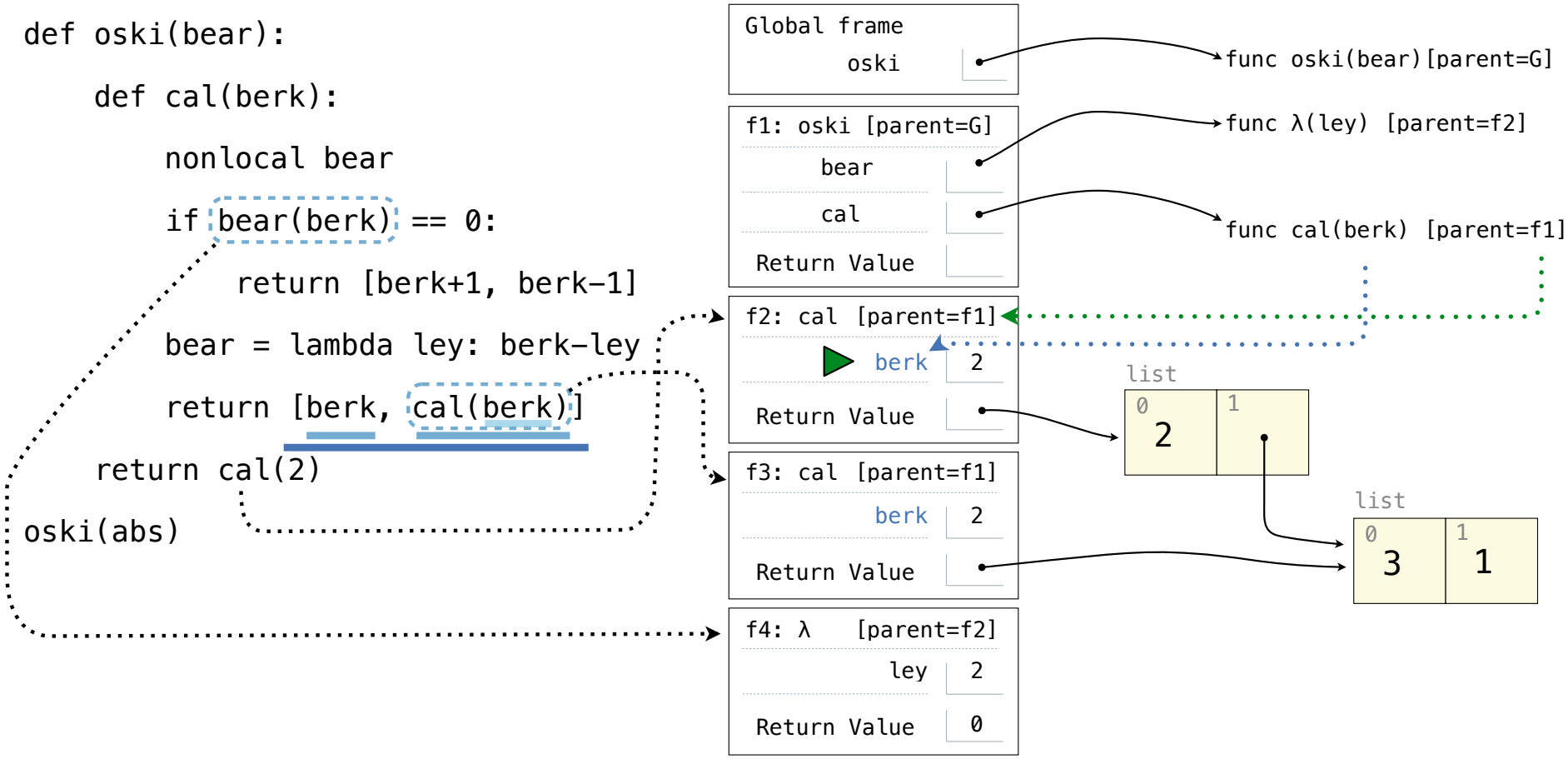
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



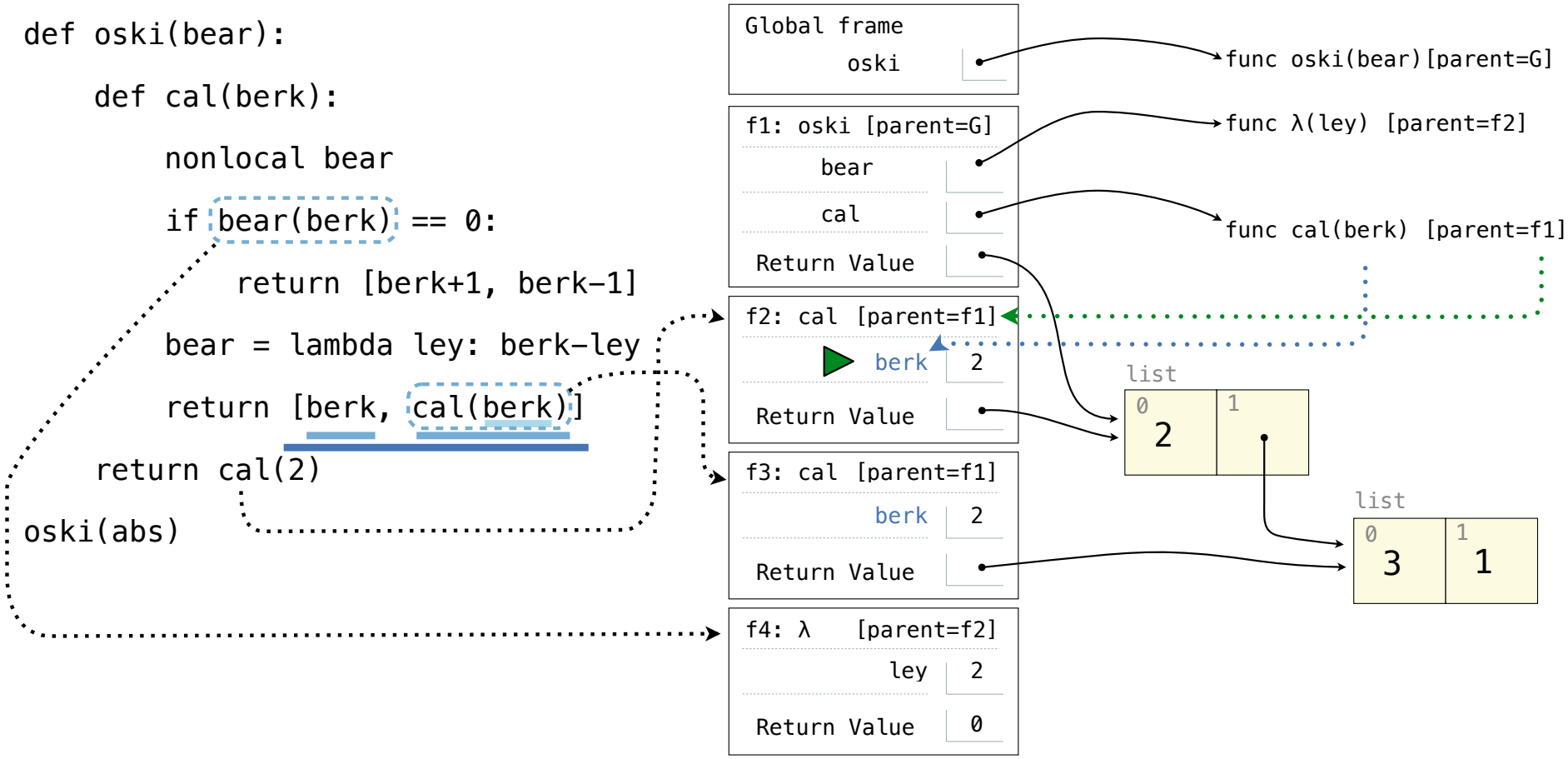
Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```