# Computing in the News

https://www.theverge.com/2019/2/25/18229714/cognizant-facebook-content-moderator-interviews-trauma-working-conditions-arizona (Content Warning: racism, discussion of mental health issues & serious violence)

https://www.fastcompany.com/90310970/the-tech-giant-fighting-anti-vaxxers-isnt-twitter-or-facebook-its-pinterest

"To spot harmful content trends on the platform, Pinterest constantly collects user feedback—via surveys or less formally— and does its own internal sweeps across content with content moderators. It has also built automated tools to block URLs that frequently share banned content, but the tools have their limits. For instance, they may fail to recognize bad content that's shared to Pinterest from Facebook, since Facebook appears as a trustworthy URL."

HOW I HEALED CANCER
without chemo or radiation
Watch My 10-part
Healing Cancer
Video Series
FREE & ONLINE
AUGUST 14-24
HERE'S HOW I DID IT
If diet is wrong,
medicine is of no
If diet is correc
medicine is of no
- Ancient Ay
"After I came out of med
school I vaccinated
thousands of children
and it's one of my
biggest regrets in life."
DR. JOSEPH MERCOLA, D.O.
The TRUTH About
VACCINES

# Mutable Functions

# Announcements

- HW5 due Friday
- Maps due Thursday
  - 1 point extra credit for submitting by tonight (2/27)
- Reminder: you have 3 slip days
  - Slip days are calculated independently for project partners
- Signups for CSM still open

# Mutable Functions

# Functions with behavior that changes over time

```
def square(x):
    return x * x

>>> square(5)
25
>>> square(5)
25
>>> square(5)
25
```

Returns the same value when called with the same input

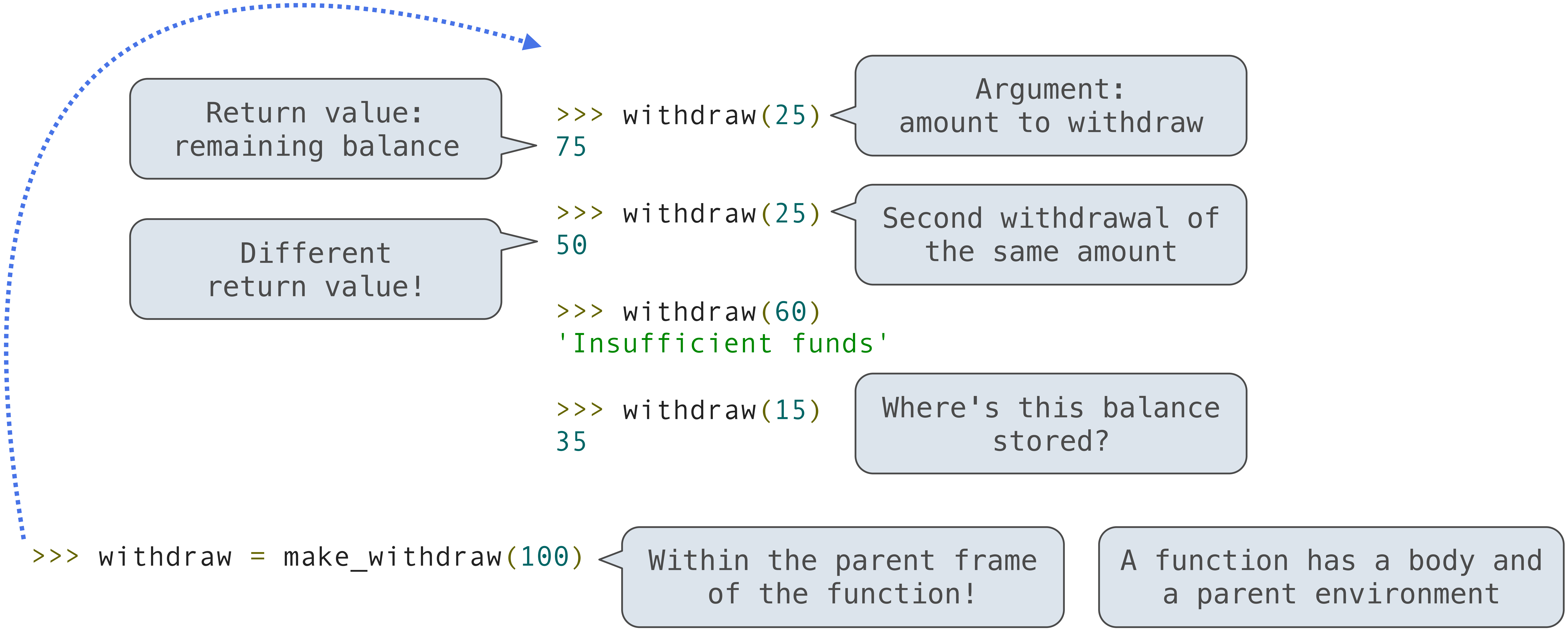
```
def f(x):
   ...

>>> f(5)
25
>>> f(5)
26
>>> f(5)
27
```

Return value is different when called with the same input

# Example - Withdraw

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)
```

Return value: remaining balance

Argument: amount to withdraw

Different return value!

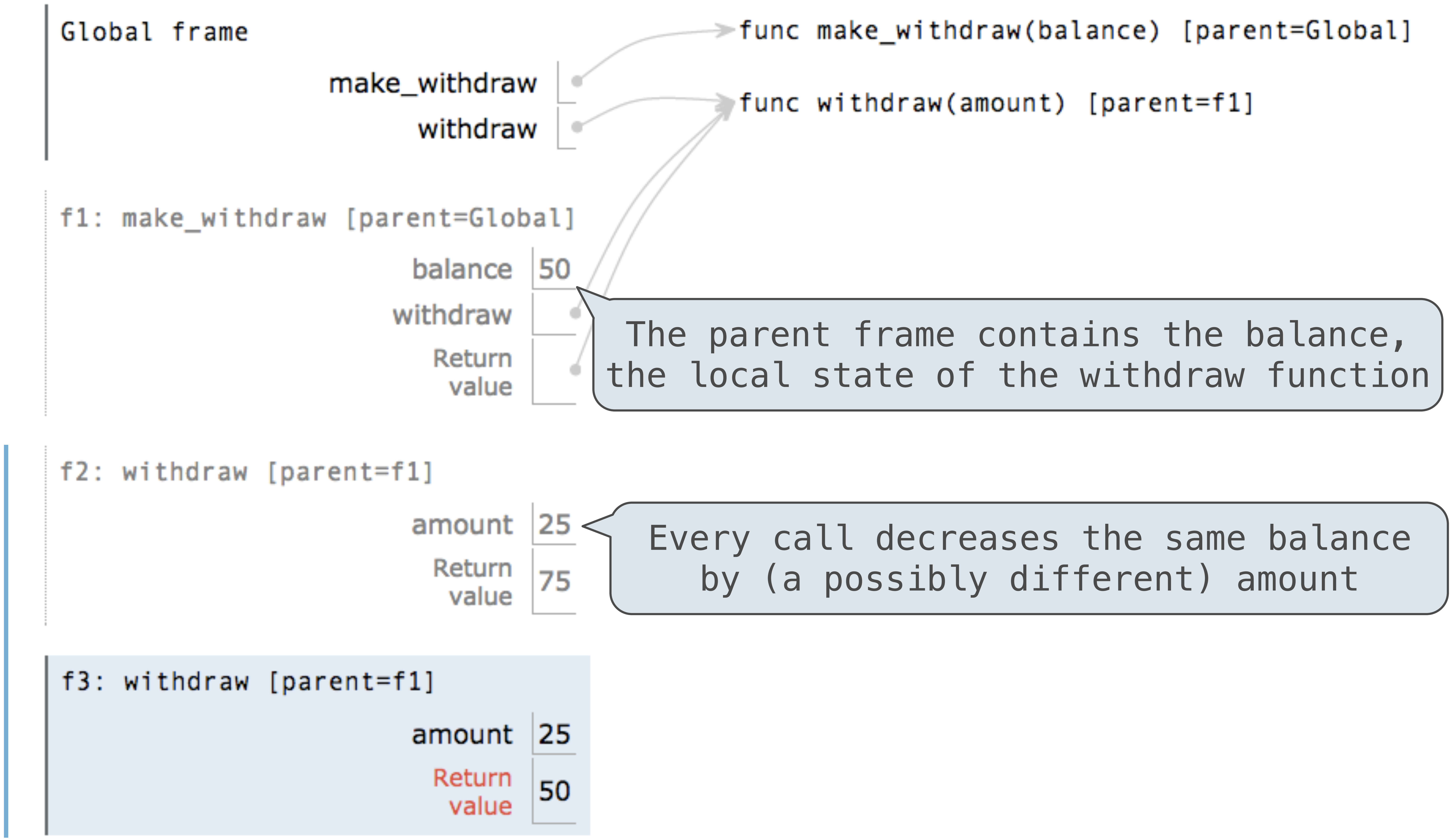
Second withdrawal of the same amount

Where's this balance stored?

Within the parent frame of the function!

A function has a body and a parent environment

# Persistent Local State Using Environments



The parent frame contains the balance, the local state of the withdraw function

Every call decreases the same balance by (a possibly different) amount

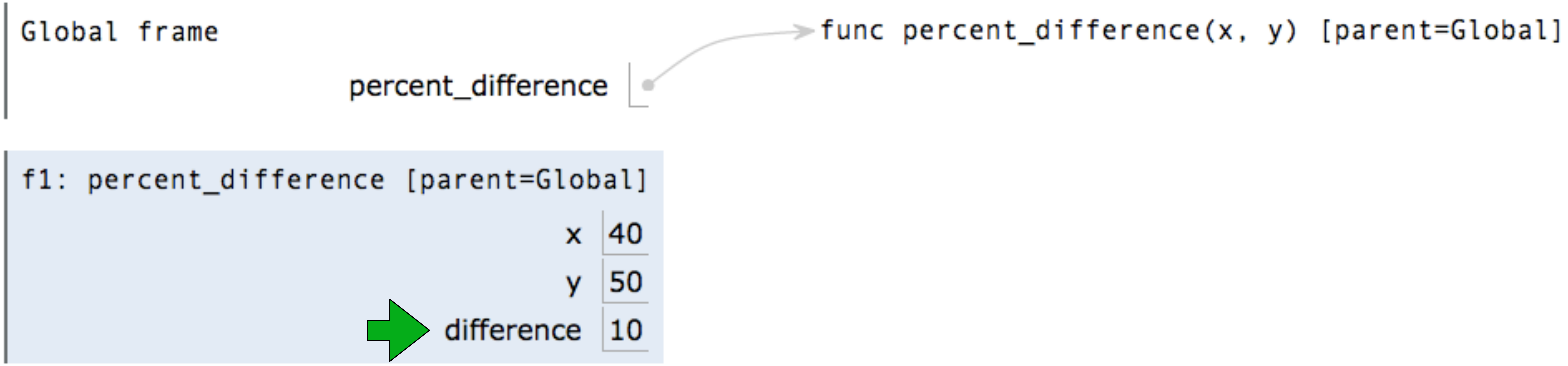
All calls to the same function have the same parent

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>%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)
```

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



Global frame

func percent_difference(x, y) [parent=Global]

| | |
|---|---|
| percent_difference | ● |

f1: percent_difference [parent=Global]

| | |
|---|---|
| x | 40 |
| y | 50 |
| difference | 10 |

**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**

pythontutor.com/composingprograms.html#code=def%20percent_difference%28x,%20y%29%3A%0A%20%20%20%20%20%20%20%20difference%20%3D%20abs%28x-y%29%0A%20%20%20%20%20%20%20%20return%20100%20*%20difference%20/%20x%0Adiff%20%3D%20percent_difference%2840,%2050%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=4

# 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>, <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

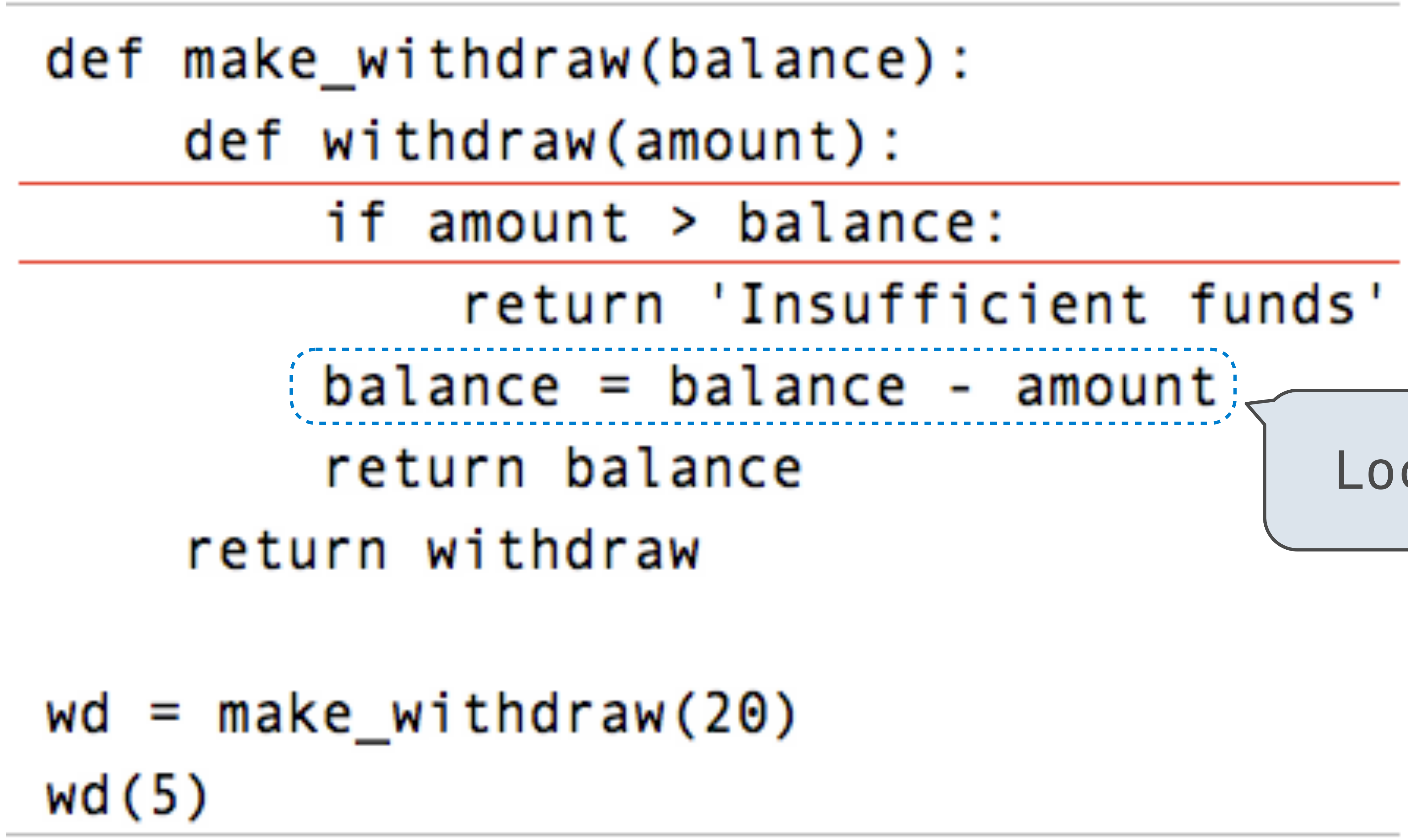
```
x = 2
```

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

# 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)
```

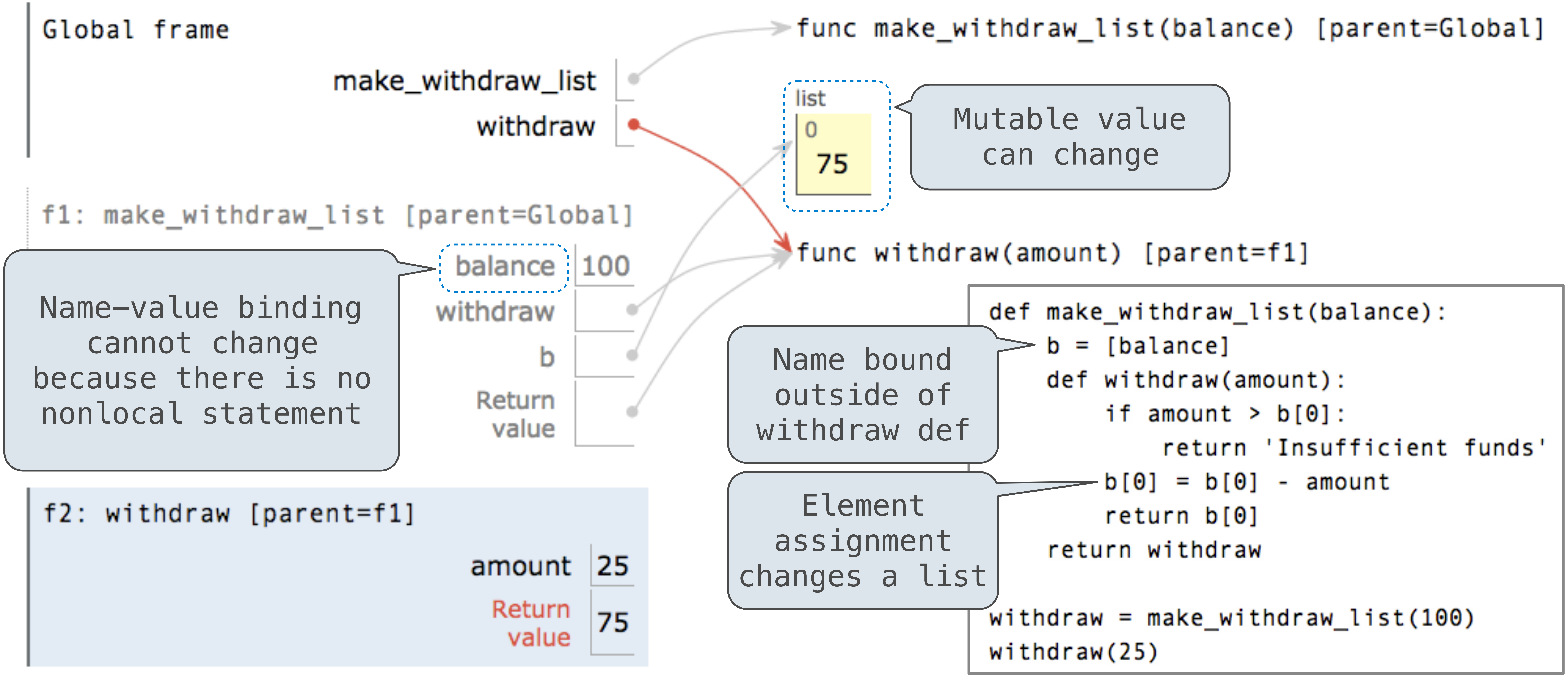
Local assignment

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>%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 value can change

Name-value binding cannot change because there is no nonlocal statement

Name bound outside of withdraw def

Element assignment changes a list

```
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)
```

goo.gl/y4TyFZ

# Multiple Mutable Functions

(Demo)

## 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

# Environment Diagrams

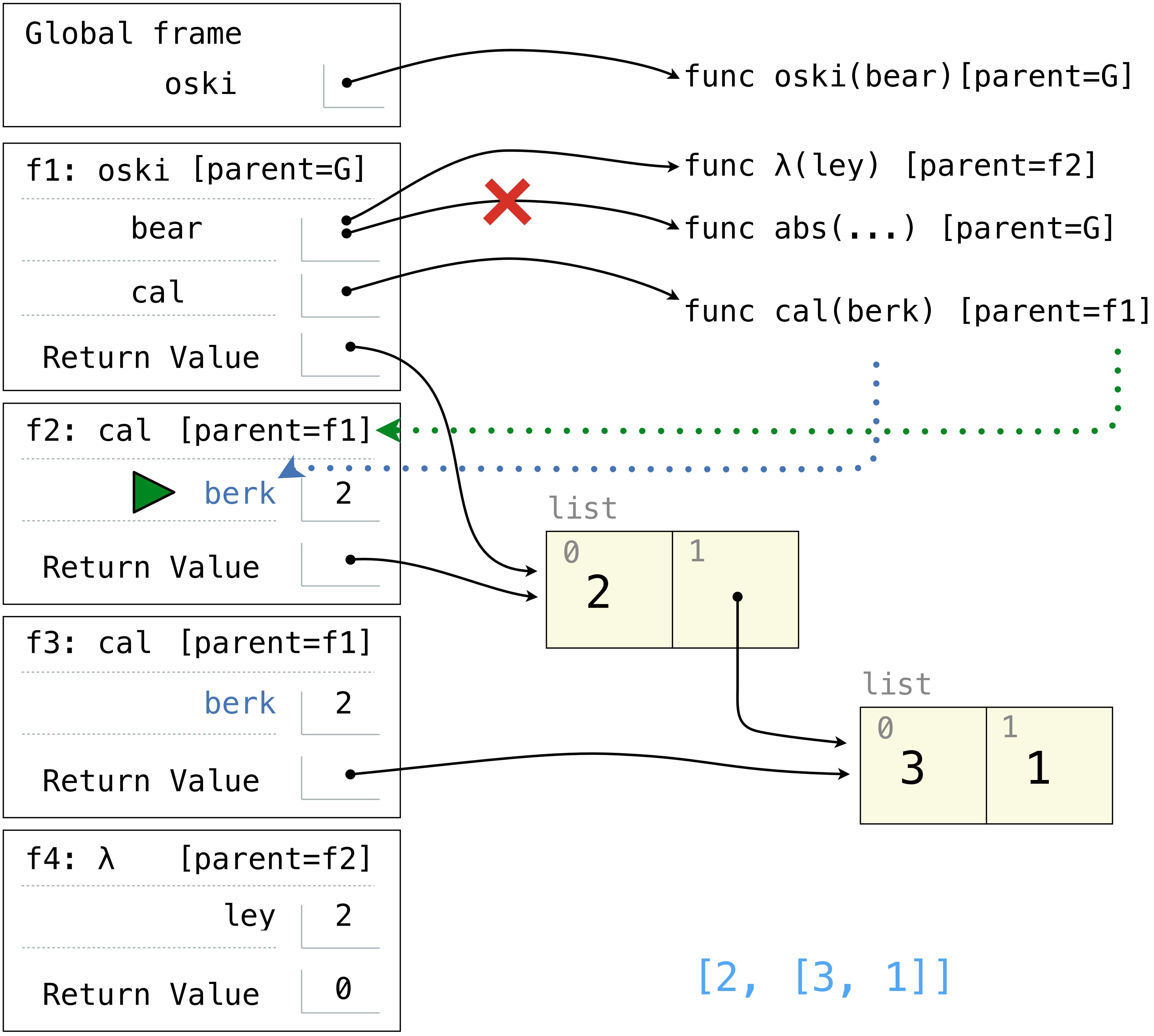
# Referential Transperency in Environment Diagrams

```
def f(x):
    x = 4
    def g(y):
        def h(z):
            nonlocal x
            x = x + 1
            return x + y + z
        return h
    return g
```

```
a = f(1)
b = a(2)
total = b(3) + b(4)
```

# 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)
```



Global frame
oski → func oski(bear)[parent=G]

f1: oski [parent=G]
bear → func λ(ley) [parent=f2]
bear ✕ → func abs(...) [parent=G]
cal → func cal(berk) [parent=f1]
Return Value → list

f2: cal [parent=f1]
berk 2
Return Value → list

f3: cal [parent=f1]
berk 2
Return Value → list

f4: λ [parent=f2]
ley 2
Return Value 0

list: 0: 2, 1: → list
list: 0: 3, 1: 1

[2, [3, 1]]

# Summary

- Nonlocal allows for functions whose behavior changes over time

- When declaring a variable nonlocal, we move part of the function's local state to its parent

- There are various rules for which variables may be declared nonlocal

- Nonlocal gives us a new type of assignment, where we change the binding in a parent instead

- Next time, we'll see more examples of functions which change state outside their local frame!