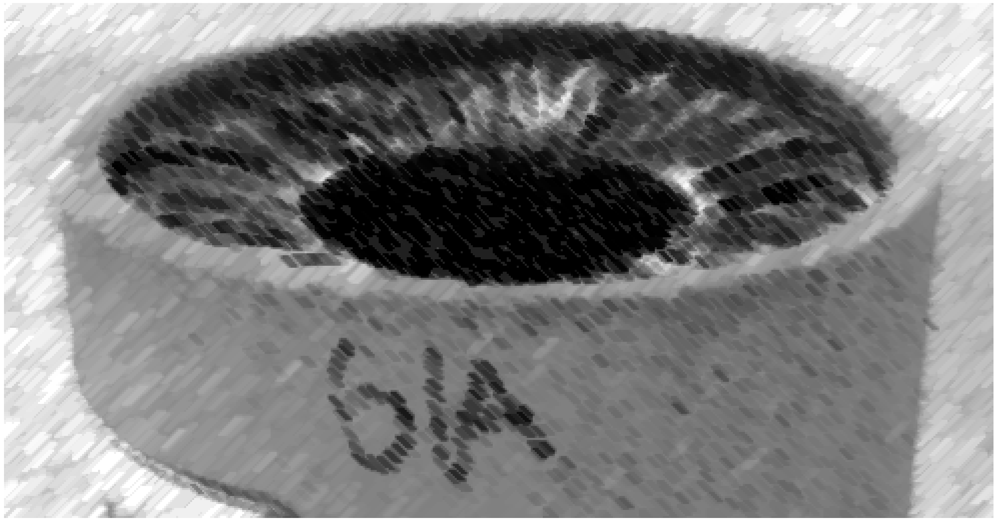


CS61A Lecture 17

Amir Kamil
UC Berkeley
March 1, 2013

Announcements

- HW6 due next Thursday
- Trends project due on Tuesday
  - Partners are required; find one in lab or on Piazza
  - Will not work in IDLE
  - New bug submission policy; see Piazza

Practical Guidance: Choosing Names

Names typically don't matter for correctness,

but they matter tremendously for legibility

`boolean` → `turn_is_over` `d` → `dice` `play_helper` → `take_turn`

Use names for repeated compound expressions

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

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

Use names for meaningful parts of compound expressions

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

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

Practical Guidance: DRY

Sometimes, removing repetition requires restructuring the code

```
def find_quadratic_root(a, b, c, plus=True):
    """Applies the quadratic formula to the polynomial
    ax^2 + bx + c."""
    if plus:
        return (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
    else:
        return (-b - sqrt(square(b) - 4 * a * c)) / (2 * a)
```

```
def find_quadratic_root(a, b, c, plus=True):
    """Applies the quadratic formula to the polynomial
    ax^2 + bx + c."""
    disc_term = sqrt(square(b) - 4 * a * c)
    if not plus:
        disc_term *= -1
    return (-b + disc_term) / (2 * a)
```

Test-Driven Development

Write the test of a function before you write a function

- A test will clarify the (one) job of the function
- Your tests can help identify tricky edge cases

Develop incrementally and test each piece before moving on

- You can't depend upon code that hasn't been tested
- Run your old tests again after you make new changes

Hog Contest

Contest rules:

- All entries run against every other entry
- An entry wins a match if its true win rate is > 0.5
- All strategies must be deterministic, pure functions and must not use pre-computed data
- Extra credit for entries with the most wins or the highest cumulative win rate
- Total of 54 valid submissions

We used **`itertools.combinations`** to determine the set of matches

Top Finishers

Congratulations to the team of Colin Lockard and Sherry Xu, who achieved a perfect 53-0 record <u>and</u> the highest win rate (28.77)!

Second-most wins (51-2): Eric Holt and Anna Carey

Second-highest win rate (28.70): Don Mai and Jeechee Chen

Third-highest in both (50-3, 28.67): Sean Scofield and Frank Lu

Complete rankings will be posted on the website

Computing Win Rates Exactly

A state in the game:

(who rolls next?, player score, opponent score)

A strategy is a table

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10

Each state has a chance to win

	When rolling 2 dice:
...	
(me,87,99)	1/36 * 1 + 35/36 * 0
...	
(you,88,99)	0
(you,90,99)	0
...	...
(you,98,99)	0
(you,100+,99)	1
(me,99,100+)	0

Requires access to both strategies, which must be deterministic

Achieving the Perfect Strategy

Optimal strategy given an opponent:

- At each state, compute probability of winning for each allowed number of dice
- Choose the number of dice that maximizes the probability

The perfect strategy: use iterative improvement!

- Initial guess: always roll 5
- Update to: optimal opponent of current strategy
- Done when: 0.5 win rate against optimal opponent

Takes only 16 steps to converge!

Can also compute perfect strategy directly using table

A Function with Evolving Behavior

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

Return value: remaining balance

Argument: amount to withdraw

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

Different return value!

Second withdrawal of the same amount

```
>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'
```

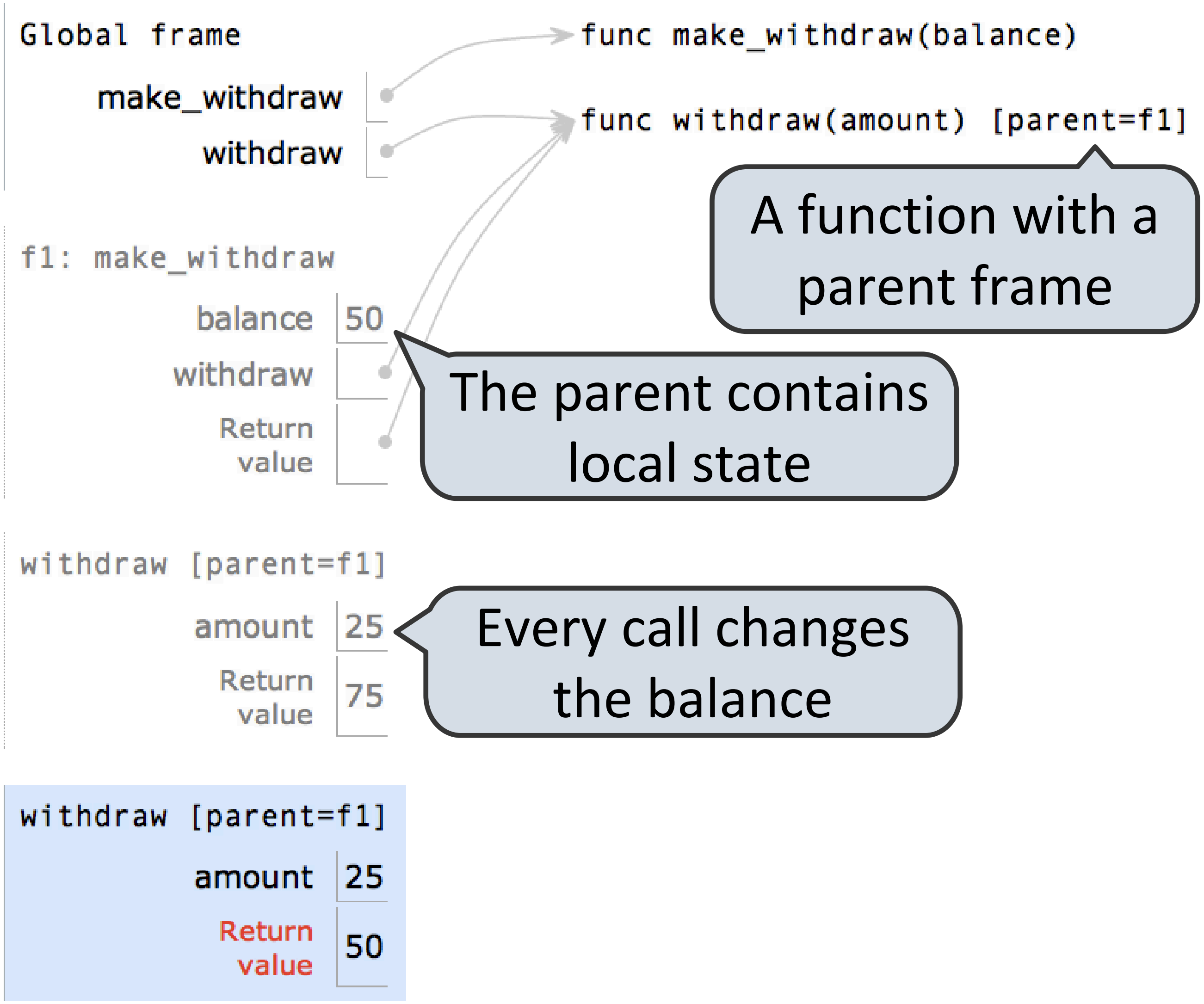
Where's this balance stored?

```
>>> withdraw(15)
35
```

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

Within the function!

Persistent Local State

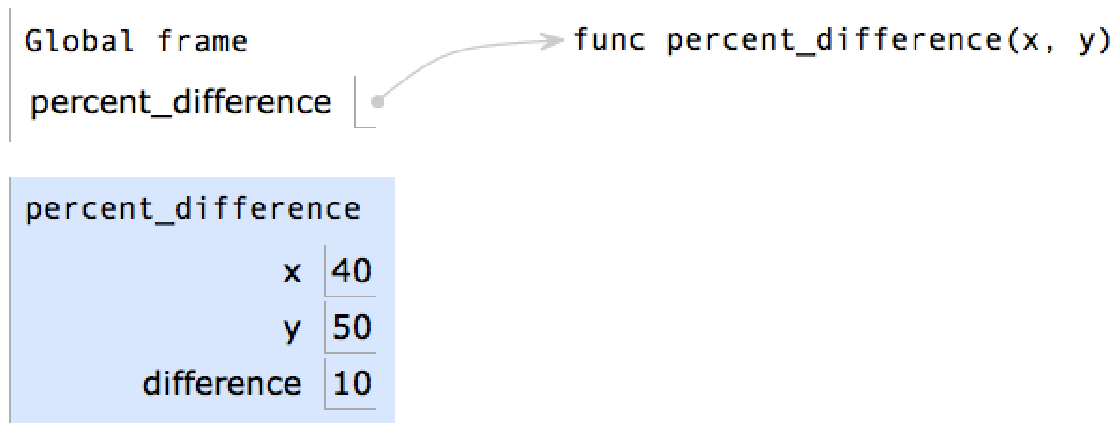


Example: http://goo.gl/5LZ6F

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 the resulting values in the first frame of the current environment.

Example: http://goo.gl/xkYgN

Non-Local Assignment

```
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

Re-bind balance where it was bound previously

The Effect of Nonlocal Statements

```
nonlocal <name>, <name 2>, ...
```

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.

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

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

Effects of Assignment Statements

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 env.
• 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 it 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

```
x = 2
```

Python Particulars

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

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

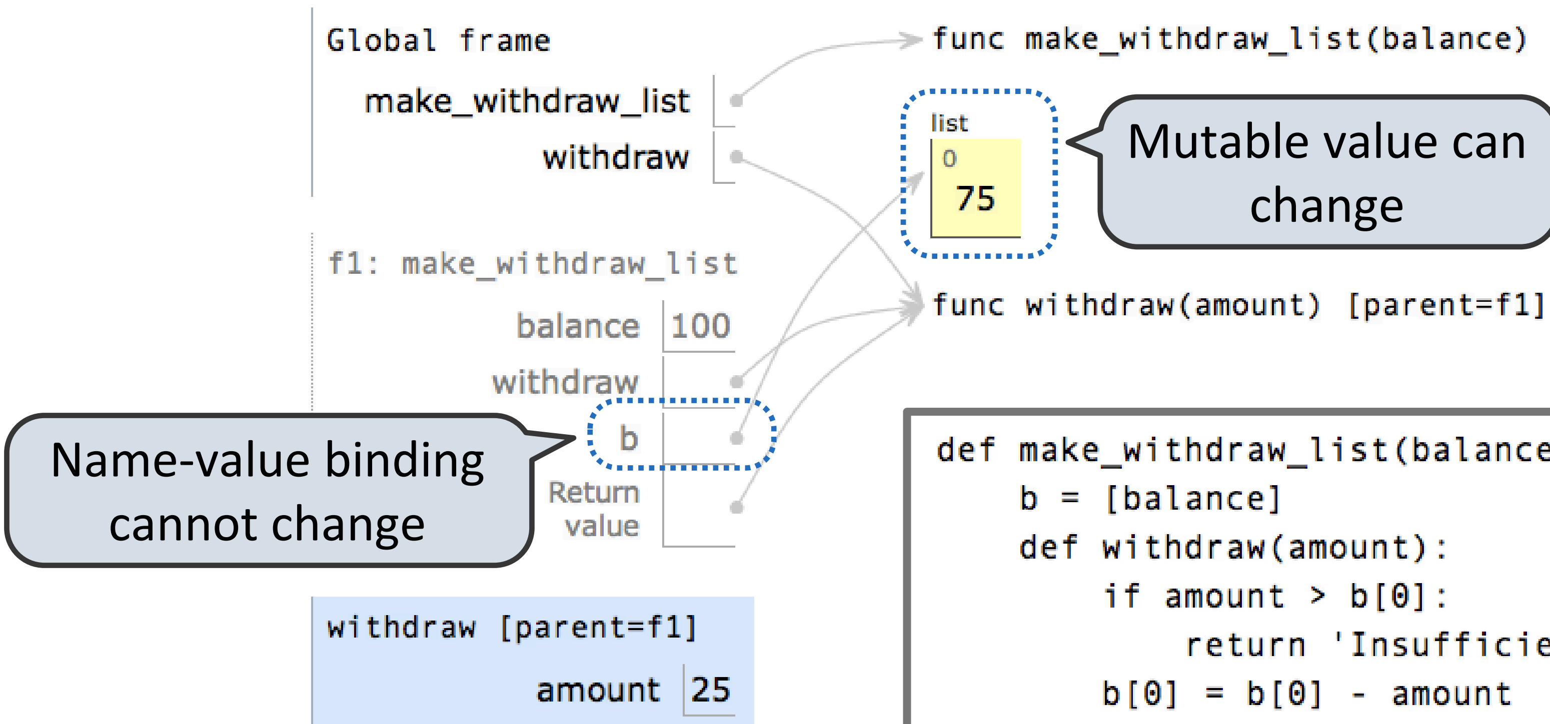
```
def make_withdraw(balance):
    def withdraw(amount):
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw

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

Local assignment

UnboundLocalError: local variable 'balance' referenced before assignment

Mutable Values and Persistent State

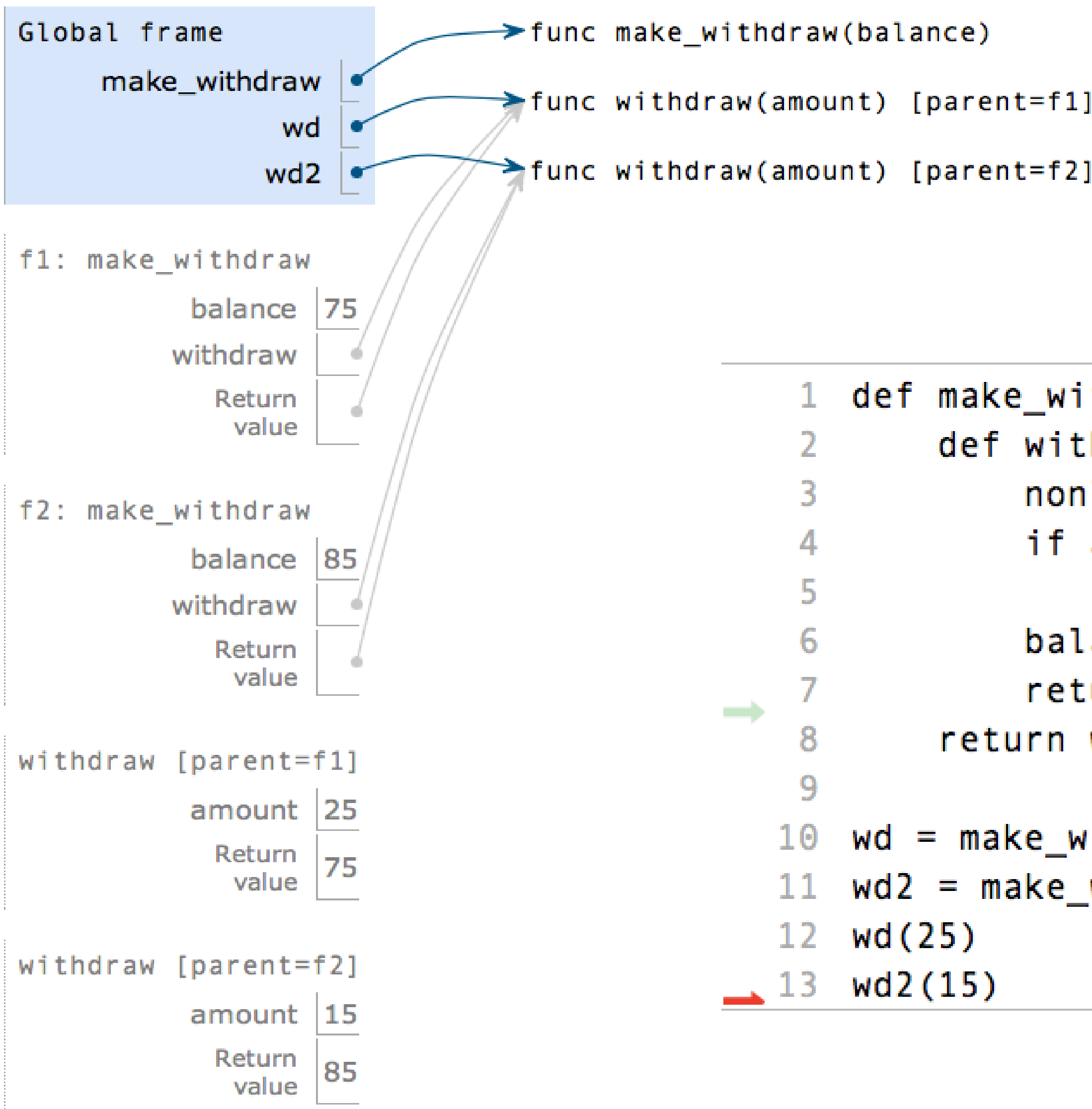
Mutable values can be changed without a nonlocal statement.




```
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)
```

Example: http://goo.gl/cEpmz

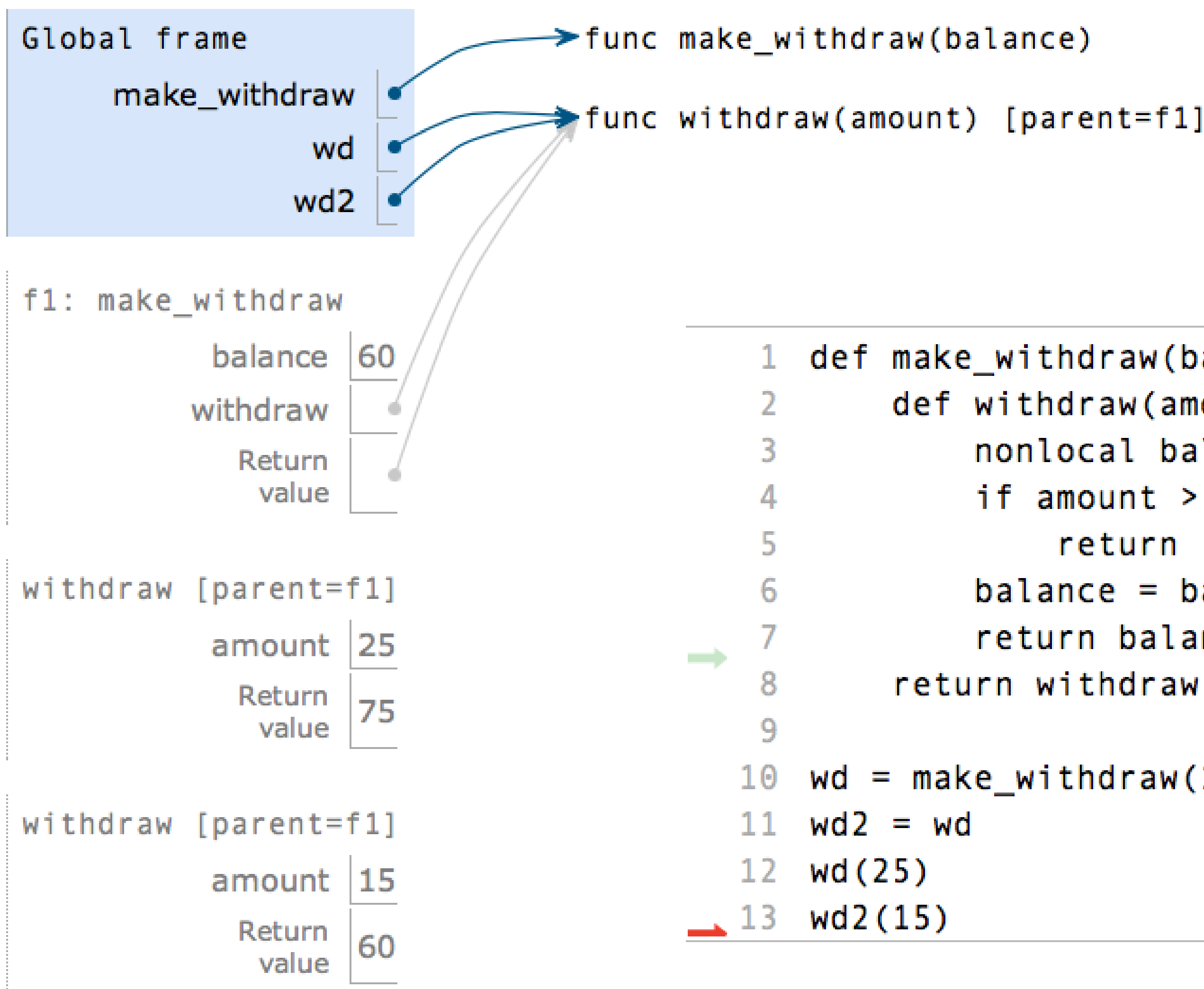
Creating Two Withdraw Functions




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

wd = make_withdraw(100)
wd2 = make_withdraw(100)
wd(25)
wd2(15)
```

Example: http://goo.gl/glTyB

Multiple References to a Withdraw Function




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

wd = make_withdraw(100)
wd2 = wd
wd(25)
wd2(15)
```

Example: http://goo.gl/X2qG9

The Benefits of Non-Local Assignment

- Ability to maintain some state that is local to a function, but evolves over successive calls to that function.

- The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is inaccessible to the rest of the program.

- An abstraction of a bank account that manages its own internal state.

Weasley Account	Potter Account
\$10	\$1,000,000

Referential Transparency

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)), 3)

mul(add(2,    24    ), 3)

mul(      26          , 3)
```

Mutation is a *side effect* (like printing)

Side effects violate the condition of referential transparency because they do more than just return a value; they change the state of the computer.