

## 61A Lecture 14

---

Friday, October 4

## Announcements

---

- Homework 4 due Tuesday 10/8 @ 11:59pm.
- Project 2 due Thursday 10/10 @ 11:59pm.
- Guerrilla Section 2 this Saturday 10/5 & Sunday 10/6 10am–1pm in Soda.
  - Topics: Data abstraction, sequences, and non-local assignment.
  - Please RSVP on Piazza!
- Guest lecture on Wednesday 10/9, Peter Norvig on Natural Language Processing in Python.
  - No video (except a screencast)! Come to Wheeler.

## Mutable Functions

## A Function with Behavior That Varies Over Time

---

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

Return value:  
remaining balance

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

Argument:  
amount to withdraw

Different  
return value!

```
>>> withdraw(25)  
50
```

Second withdrawal of  
the same amount

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

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

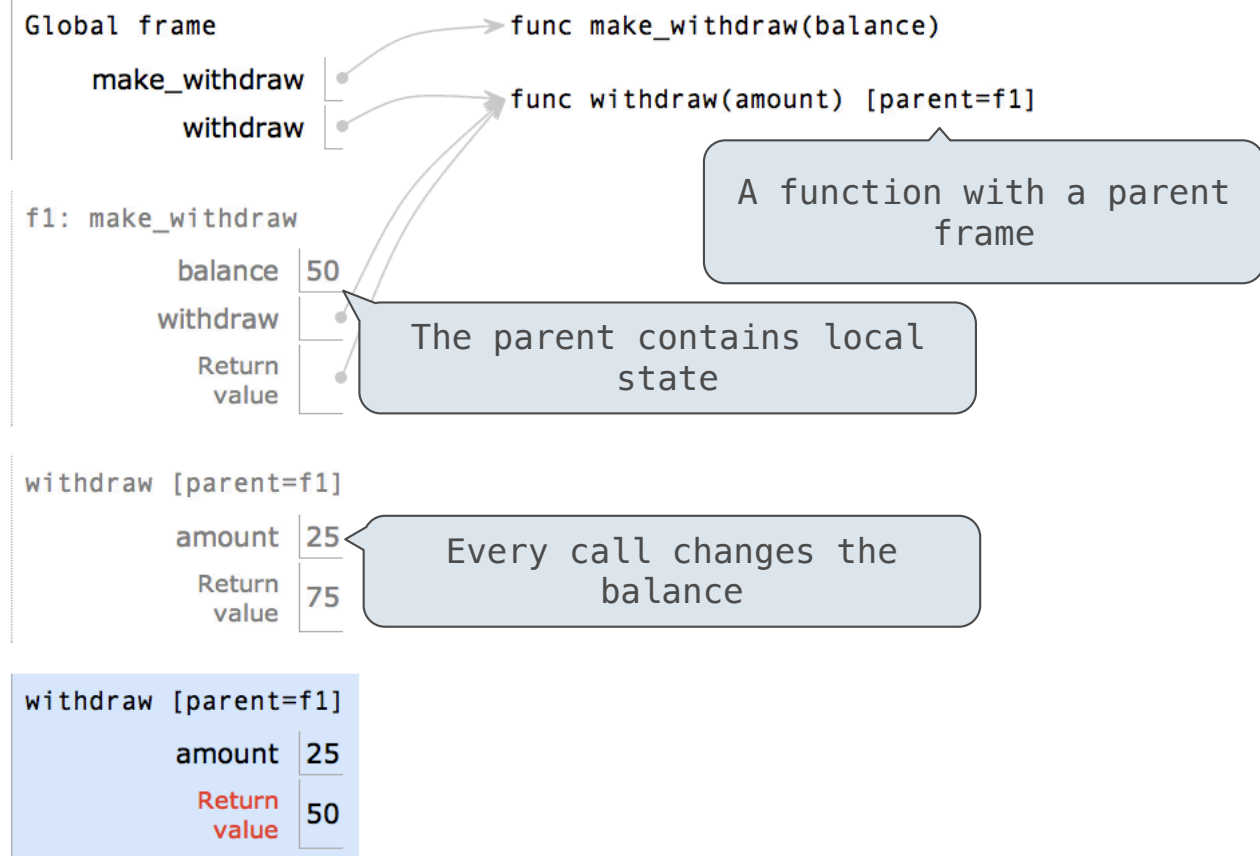
Where's this balance  
stored?

```
>>> 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



## 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

percent\_difference

func percent\_difference(x, y)

percent\_difference

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

## 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.

[http://docs.python.org/release/3.1.3/reference/simple\\_stmts.html#the-nonlocal-statement](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
- "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

## 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.

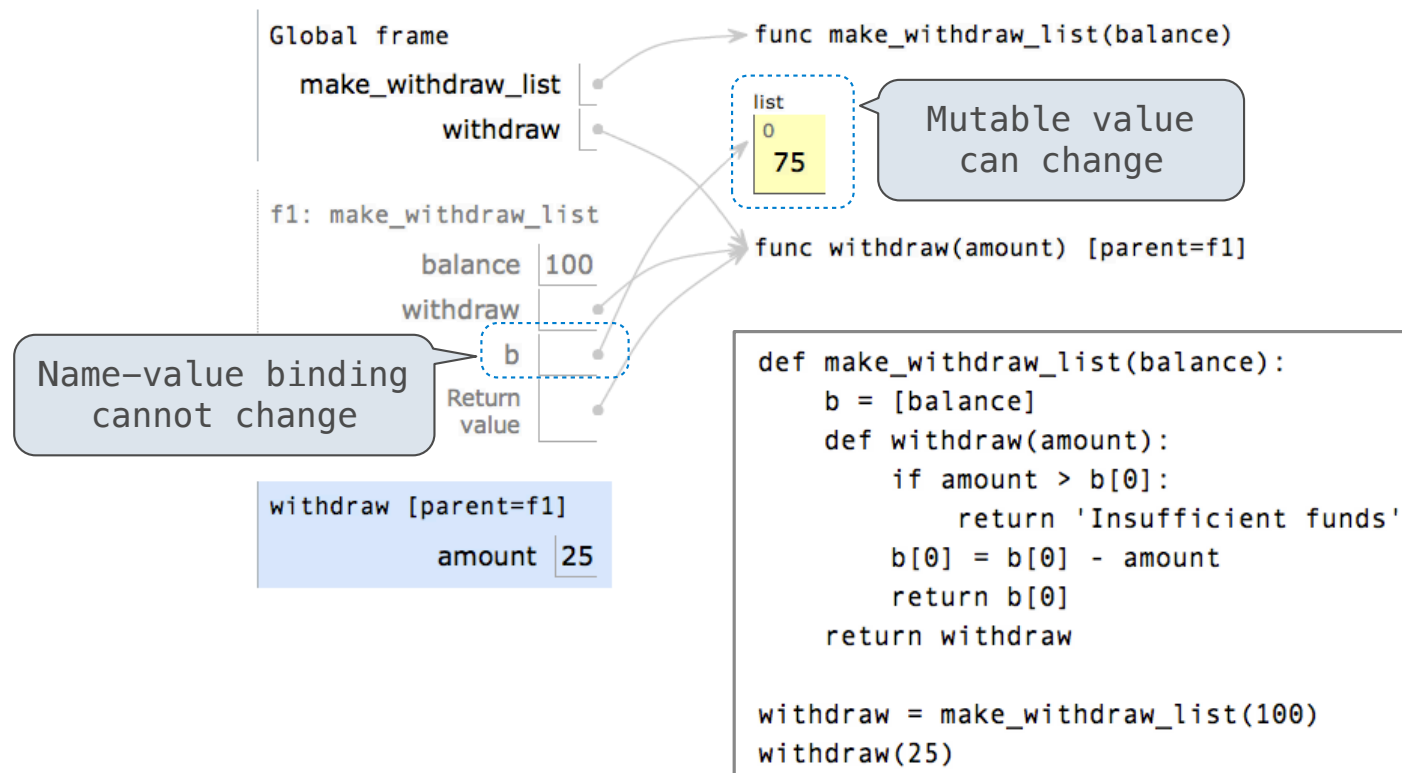
```
def make_withdraw(balance):  
    def withdraw(amount):  
        if amount > balance:  
            return 'Insufficient funds'  
            balance = balance - amount  
            return balance  
    return withdraw  
  
wd = make_withdraw(20)  
wd(5)
```

Local assignment

**UnboundLocalError: local variable 'balance' referenced before assignment**

## Mutable Values & Persistent Local State

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



Example: <http://goo.gl/y4TyFZ>

## Multiple Mutable Functions

(Demo)

## Sameness and Change

---

- As long as we **never modify** objects, we can regard a compound object to be precisely the **totality of its pieces**.
- A **rational number** is just its numerator and denominator.
- This view is no longer valid **in the presence of change**.
- Now, a compound data **object has an "identity"** that is something more than the pieces of which it is composed.
- A bank account is **still "the same" bank account even if we change the balance** by making a withdrawal.
- Conversely, we could have two bank accounts that happen to have the **same balance, but are different objects**.

John's Account
\$10

Steven's Account
\$10

## 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.**

(Demo)