CS61A Lecture 18

Amir Kamil
UC Berkeley
March 4, 2013

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

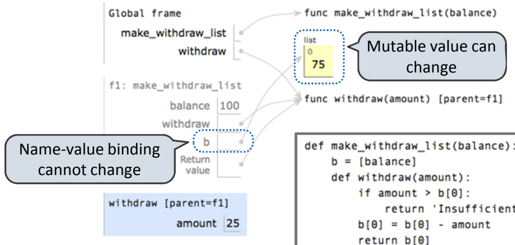
Re-bind balance where it was bound previously

Announcements

- HW6 due on Thursday
- Trends project due tomorrow
- Ants project out

Mutable Values and Persistent State

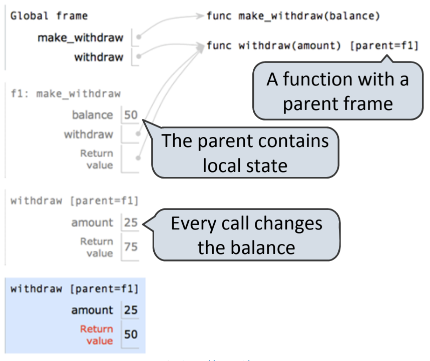
Mutable values can be changed without a nonlocal statement.



```
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)
```

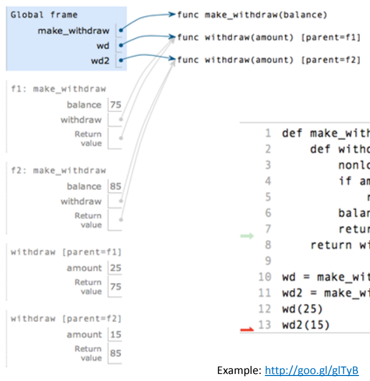
Example: http://goo.gl/cEpmz

Persistent Local State



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

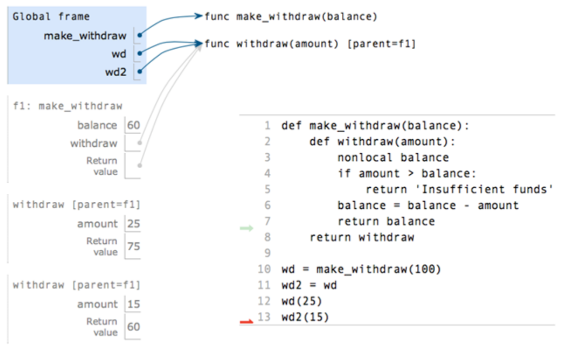
Creating Two Withdraw Functions



```
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



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

Example: http://goo.gl/X2qG9

A Mutable Container

```python
def container(contents):
    """Return a container that is manipulated by two
    functions.

    >>> get, put = container('hello')
    >>> get()
    'hello'
    >>> put('world')
    >>> get()
    'world'
    """
    def get():
        return contents
    def put(value):
        nonlocal contents
        contents = value
    return put, get
```

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

Dispatch Functions

A technique for packing multiple behaviors into one function

```python
def pair(x, y):
    """Return a function that behaves like a pair."""
    def dispatch(m):
        if m == 0:
            return x
        elif m == 1:
            return y
    return dispatch
```

Message argument can be anything, but strings are most common

The body of a dispatch function is always the same:

- One conditional statement with several clauses
- Headers perform equality tests on the message

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.

Message Passing

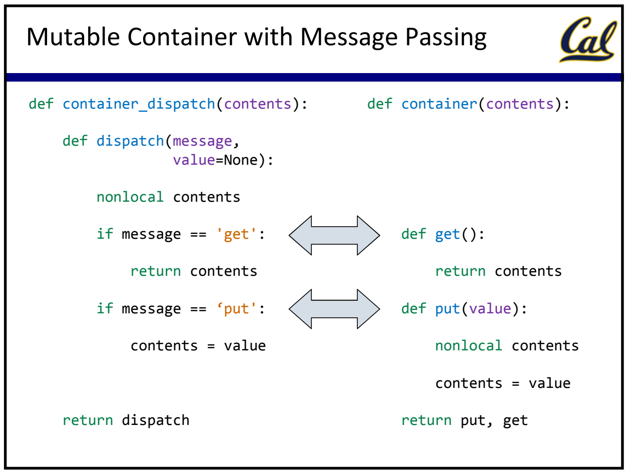
An approach to organizing the relationship among different pieces of a program

Different objects pass messages to each other

- What is your fourth element?
- Change your third element to this new value. (please?)

Encapsulates the behavior of all operations on a piece of data

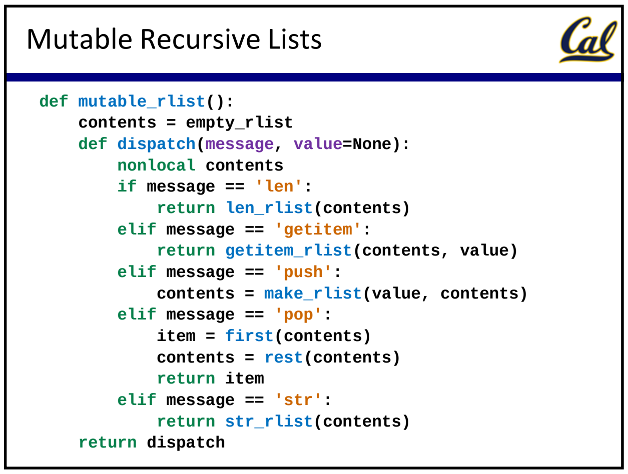
Important historical role:
The message passing approach strongly influenced object-oriented programming
(next lecture)



Mutable Container with Message Passing

```
def container_dispatch(contents):

    def dispatch(message,
                 value=None):

        nonlocal contents

        if message == 'get':

            return contents

        if message == ‘put':

            contents = value

    return dispatch
```

```
def container(contents):

    def get():

        return contents

    def put(value):

        nonlocal contents

        contents = value

    return put, get
```



Mutable Recursive Lists

```
def mutable_rlist():
    contents = empty_rlist
    def dispatch(message, value=None):
        nonlocal contents
        if message == 'len':
            return len_rlist(contents)
        elif message == 'getitem':
            return getitem_rlist(contents, value)
        elif message == 'push':
            contents = make_rlist(value, contents)
        elif message == 'pop':
            item = first(contents)
            contents = rest(contents)
            return item
        elif message == 'str':
            return str_rlist(contents)
    return dispatch
```