Mutable Functions
A Function with Behavior That Varies Over Time

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

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

Return value: remaining balance

Different return value!

Argument: amount to withdraw

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
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.
Reminder: Local Assignment

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

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

diff = percent_difference(40, 50)

# Global frame
percent_difference

# f1: percent_difference [parent=Global]
    x  40
    y  50
    difference 10
```
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

(Demo)
Non-Local Assignment
The Effect of Nonlocal Statements

```python
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.

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/
### The Many Meanings of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No nonlocal statement</td>
<td>Create a new binding from name &quot;x&quot; to object 2 in the first frame of the current environment</td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound locally</td>
<td></td>
</tr>
<tr>
<td>• No nonlocal statement</td>
<td>Re-bind name &quot;x&quot; to object 2 in the first frame of the current environment</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td></td>
</tr>
<tr>
<td>• nonlocal x</td>
<td>Re-bind &quot;x&quot; to 2 in the first non-local frame of the current environment in which &quot;x&quot; is bound</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a non-local frame</td>
<td></td>
</tr>
<tr>
<td>• nonlocal x</td>
<td>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound in a non-local frame</td>
<td></td>
</tr>
<tr>
<td>• nonlocal x</td>
<td>SyntaxError: name 'x' is parameter and nonlocal</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a non-local frame</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; is also bound locally</td>
<td></td>
</tr>
</tbody>
</table>
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
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)

UnboundLocalError: local variable 'balance' referenced before assignment
```
Mutable values can be changed *without* a nonlocal statement.
Multiple Mutable Functions

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

\[
\text{mul}\left(\text{add}(2, \text{mul}(4, 6)), \text{add}(3, 5)\right)
\]

\[
\text{mul}\left(\text{add}(2, 24), \text{add}(3, 5)\right)
\]

\[
\text{mul}\left(26, \text{add}(3, 5)\right)
\]

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