Mutable Functions

A Function with Behavior That Varies Over Time

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

Argument:
- amount to withdraw

Return value:
- remaining balance

Different return value!

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

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

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

Non-Local Assignment & Persistent Local State

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.

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.

Visit Python Docs: an "enclosing scope"

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

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

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.

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

Python Docs: an "enclosing scope"

**Interactive Diagram**

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:
            raise 'Insufficient funds'
        balance -= amount
        return balance
    return withdraw
wd = make_withdraw(200)
wd(5)
```

UnboundLocalError: local variable 'balance' referenced before assignment

**Interactive Diagram**

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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;x&quot; is bound locally</td>
<td>Re-bind &quot;x&quot; to object 2 in the first frame of the current environment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;x&quot; is not bound locally</td>
<td>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;x&quot; is bound in a non-local frame</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;x&quot; is bound locally</td>
<td>SyntaxError: name &quot;x&quot; is parameter and nonlocal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;x&quot; is bound in a non-local frame</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;x&quot; is bound locally</td>
<td>SyntaxError: name &quot;x&quot; is parameter and nonlocal</td>
</tr>
</tbody>
</table>

**Interactive Diagram**

Mutable Values & Persistent Local State

Mutable values can be changed without a nonlocal statement.

```
from collections import deque

def make_withdraw_list(balance):
    def withdraw(amount):
        if amount > balance:
            raise 'Insufficient funds'
        balance -= amount
        return balance
    return withdraw
wdl = make_withdraw_list([100])
wdl(25)
```

**Interactive Diagram**

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.