Mutable Functions

A Function with Behavior That Varies Over Time

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

Given a function `withdraw(amount)`:

```python
>>> withdraw(25)
75
>>> withdraw(25)
50
>>> withdraw(60)
'Insufficient funds'
>>> withdraw(15)
35
>>> make_withdraw(100)
```

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!

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!

Persistent Local State Using Environments

The parent frame contains the balance, the local state of the withdraw function

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

Non-Local Assignment & Persistent Local State

```python
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

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
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.

https://docs.python.org/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.

Python Docs: an "enclosing scope"

The Many Meanings of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;x&quot; is not bound locally</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 bound locally</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 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>
<tr>
<td>&quot;x&quot; is bound in a non-local frame</td>
<td>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
<tr>
<td>&quot;x&quot; is bound locally</td>
<td>SyntaxError: name 'x' is parameter and nonlocal</td>
</tr>
<tr>
<td>&quot;x&quot; also bound locally</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>

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.

Multiple Mutable Functions

(Demo)

Mutable Values & Persistent Local State

Mutable values can be changed without a nonlocal statement.

Referential Transparency, Lost

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

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

Review Problem