**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

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**

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 the resulting values in the first frame of the current environment.
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 -= amount
        return balance
    return withdraw

(Demo)
```

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.

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.

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

Mutable Values & Persistent Local State

Mutable values can be changed without a nonlocal statement.

```python
nonlocal x
x is bound in a non-local frame
x is bound in a non-local frame
x also bound locally
```

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://www.python.org/dev/peps/pep-3104/
HTTP://docs.python.org/release/3.1.3/reference/simple_stmts.html#the-nonlocal-statement
Multiple Mutable Functions

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.

<table>
<thead>
<tr>
<th>John's Account</th>
<th>Steven's Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10</td>
<td>$10</td>
</tr>
</tbody>
</table>

Referential Transparency, Lost

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

$\text{mul}(\text{add}(2, \text{mul}(4, 6)), \text{add}(3, 5))$

$\text{mul}(\text{add}(2, 24), \text{add}(3, 5))$

$\text{mul}(26, \text{add}(3, 5))$

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