A Function with Behavior That Varies Over Time

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

Return value: remaining balance

>>> withdraw(25) 75

Different return value!

Second withdrawal of the same amount

>>> withdraw(25) 50

'Insufficient funds'

>>> withdraw(60) 35

Where's this balance stored?

Within the function!

>>> withdraw = make_withdraw(100)

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

Argument: amount to withdraw

Second withdrawal of the same amount

Return value:

Different return value!

Where's this balance stored?

Within the function!

Local State via Non-Local Assignment

```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
```

The Effect of Nonlocal Statements

```python
nonlocal <name>, <name 2>, ...
```

**Effect:** Future references to that name refer to 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://docs.python.org/3/reference/grammar.html#nonlocal-statement
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>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>Re-bind name &quot;x&quot; to object 2 in the first frame of the current environment in which that name is already bound.</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>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
<tr>
<td>• nonlocal x</td>
<td>Re-binds &quot;x&quot; to 2 in the first non-local frame of the current environment in which that name is already bound.</td>
</tr>
</tbody>
</table>

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the turtle function
- Apply the mutant function to 5
- In the first frame, bind y to 6 and x to 7
- Look up ninja, which is bound to the turtle function
- Look up y, which is bound to 6
- Apply the turtle function to 6
- Look up x, which is bound to 6 in the local frame
- Look up y, which is bound to 5 in the global frame
- Return 32
- Return half the result: 16

Environment Diagram of Withdraw

Calling a Withdraw Function Twice

Creating Two Different Withdraw Functions
Creating Two Different Withdraw Functions

```python
withdraw(amount):
    balance:
    make_withdraw
wd:
wd(6)
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance

wd2:
wd2 = make_withdraw(20)
wd(5)
wd(3)
w2 = make_withdraw(7)
w2(6)
```

The Benefit 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.

Multiple References to a Single Withdraw Function

```python
withdraw(amount):
    balance:
    make_withdraw
wd:
wd(1)
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance

wd2:
wd2 = wd
wd2(1)
wd(1)
```

Sameness and Change

- So long as we never modify data objects, we can regard a compound object to be precisely the totality of its pieces.
- A rational number is determined by 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.

Referential Transparency, Lost

- An expression is referentially transparent if its value does not change when we substitute one of its subexpression with the value of that subexpression.

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

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

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

- Re-binding operations violate the condition of referential transparency because they do more than return a value; they change the environment.
- Two separately defined functions are not the same, because changes to one may not be reflected in the other.