Lecture 13: Mutable Functions

Brian Hou
July 12, 2016
Announcements
Announcements

• Project 2 is due today (submit early and often)
Announcements

- Project 2 is due today (submit early and often)
  - Look at your Hog submission for composition feedback
Announcements

• Project 2 is due today (submit early and often)
  • Look at your Hog submission for composition feedback
• Midterm is on 7/14 from 5–8 PM in 2050 VLSB
Announcements

• Project 2 is due today (submit early and often)
  • Look at your Hog submission for composition feedback
• Midterm is on 7/14 from 5–8 PM in 2050 VLSB
  • TA-led review session during lecture tomorrow
Announcements

- Project 2 is due today (submit early and often)
  - Look at your Hog submission for composition feedback
- Midterm is on 7/14 from 5–8 PM in 2050 VLSB
  - TA-led review session during lecture tomorrow
  - Office hours after 3 PM on Thursday and on Friday have been rescheduled
Announcements

- Project 2 is due today (submit early and often)
  - Look at your Hog submission for composition feedback
- Midterm is on 7/14 from 5–8 PM in 2050 VLSB
  - TA-led review session during lecture tomorrow
  - Office hours after 3 PM on Thursday and on Friday have been rescheduled
- More information on Piazza
Roadmap

Introduction
Functions
Data
Mutability
Objects
Interpretation
Paradigms
Applications
This short week (Mutability), the goals are:
This short week (Mutability), the goals are:

- To explore the power of values that can *mutate*, or change
Mutable Functions
Functions That Change
Functions That Change

How can we model a bank account that has a balance of $100?
Functions That Change

How can we model a bank account that has a balance of $100?

```python
>>> withdraw = make_withdraw(100)
```
Functions That Change

How can we model a bank account that has a balance of $100?

```python
>>> withdraw = make_withdraw(100)
>>> withdraw(25)
75
```
Functions That Change

How can we model a bank account that has a balance of $100?

```python
>>> withdraw = make_withdraw(100)
>>> withdraw(25)
75

Argument: amount to withdraw
```
Functions That Change

How can we model a bank account that has a balance of $100?

```python
>>> withdraw = make_withdraw(100)
>>> withdraw(25)
75
```

Return value: remaining balance

Argument: amount to withdraw
Functions That Change

How can we model a bank account that has a balance of $100?

```python
>>> withdraw = make_withdraw(100)
>>> withdraw(25)
75
>>> withdraw(25)
50
```

Return value: remaining balance

Argument: amount to withdraw
Functions That Change

How can we model a bank account that has a balance of $100?

```python
>>> withdraw = make_withdraw(100)
>>> withdraw(25)
75
>>> withdraw(25)
50
```

Return value: remaining balance

Argument: amount to withdraw

Second withdrawal of the same amount
Functions That Change

How can we model a bank account that has a balance of $100?

```python
>>> withdraw = make_withdraw(100)
>>> withdraw(25)
75
>>> withdraw(25)
50
```

Return value: remaining balance

Argument: amount to withdraw

Different return value!

Second withdrawal of the same amount
Functions That Change

How can we model a bank account that has a balance of $100?

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

Return value: remaining balance

Different return value!

Argument: amount to withdraw

Second withdrawal of the same amount
How can we model a bank account that has a balance of $100?

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

How can we model a bank account that has a balance of $100?

```python
>>> withdraw = make_withdraw(100)
```

75

```python
>>> withdraw(25)
```

50

```python
>>> withdraw(60)
'Insufficient funds'
```

```python
>>> withdraw(15)
```

35

Where is this balance stored?
Persistent Local State in Environments

All calls to the same function have the same parent.
Persistent Local State in Environments

All calls to the same function have the same parent.

Every call decreases the same balance by (a possibly different) amount.
Persistent Local State in 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.
- All calls to the same function have the same parent.
Nonlocal Assignment
def make_withdraw(balance):

Nonlocal Assignment
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):

def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""

def withdraw(amount):
    nonlocal balance
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
def make_withdraw(balance):
    """Return a withdraw function with a starting balance.""

def withdraw(amount):
    nonlocal balance
    if amount > balance:
        return 'Insufficient funds'
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
Nonlocal 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
```
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
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

Nonlocal Assignment

Declare the name balance nonlocal at the top of the function in which it is re-assigned
Nonlocal 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
```

Declare the name `balance` nonlocal at the top of the function in which it is re-assigned.

Re-bind `balance` in the first nonlocal frame in which it was bound previously.
```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
```
Nonlocal Assignment
Nonlocal Statements

nonlocal <name>
Nonlocal Statements

```
nonlocal <name>
```

**Effect:** Future assignments to that name change its pre-existing binding in the **first nonlocal frame** of the current environment in which that name is bound.
Nonlocal Statements

nonlocal <name>

**Effect:** Future assignments to that name change its pre-existing binding in the *first nonlocal frame* of the current environment in which that name is bound.
Nonlocal Statements

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

**Effect:** Future assignments to that name change its pre-existing binding in the **first nonlocal frame** of the current environment in which that name is bound.

*Python Docs: an "enclosing scope"*
Nonlocal Statements

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

**Effect:** Future assignments to that name change its pre-existing binding in the first nonlocal frame of the current environment in which that name is bound.

From the Python 3 language reference:
Nonlocal Statements

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

**Effect:** Future assignments to that name change its pre-existing binding in the **first nonlocal frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope.
Nonlocal Statements

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

**Effect:** Future assignments to that name change its pre-existing binding in the **first nonlocal 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.
Nonlocal Statements

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

**Effect:** Future assignments to that name change its pre-existing binding in the first nonlocal 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.
Nonlocal Statements

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

**Effect:** Future assignments to that name change its pre-existing binding in the first nonlocal 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/
Assignment Statements
Assignment Statements

\[ x = 2 \]
## Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>

\( x = 2 \)
Assignment Statements

\[ x = 2 \]

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No nonlocal statement</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound locally</td>
<td></td>
</tr>
</tbody>
</table>

$x = 2$
## Assignment Statements

### Status
- No nonlocal statement
- "x" is not bound locally

### Effect
Create a new binding from name "x" to value 2 in the first frame of the current environment
## Assignment Statements

\[ x = 2 \]

<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 value 2 in the first frame of the current environment</td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound locally</td>
<td></td>
</tr>
</tbody>
</table>
### 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 value 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></td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td></td>
</tr>
</tbody>
</table>

\[ x = 2 \]
## Assignment Statements

\[ x = 2 \]

<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 value 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 value 2 in the first frame of the current environment</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td></td>
</tr>
</tbody>
</table>
## Assignment Statements

\[ x = 2 \]

<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 value 2 in the first frame of the current environment</td>
</tr>
<tr>
<td>• &quot;x&quot; <strong>is not</strong> bound locally</td>
<td></td>
</tr>
<tr>
<td>• No nonlocal statement</td>
<td>Re-bind name &quot;x&quot; to value 2 in the first frame of the current environment</td>
</tr>
<tr>
<td>• &quot;x&quot; <strong>is</strong> bound locally</td>
<td></td>
</tr>
</tbody>
</table>
### Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>
| • No nonlocal statement  
• "x" is not bound locally | Create a new binding from name "x" to value 2 in the first frame of the current environment |
| • No nonlocal statement  
• "x" is bound locally | Re-bind name "x" to value 2 in the first frame of the current environment |
| • nonlocal x  
• "x" is bound in a nonlocal frame | |

\[ x = 2 \]
## Assignment Statements

### $x = 2$

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>
| • No nonlocal statement  
• "x" **is not** bound locally                                                | Create a new binding from name "x" to value 2 in the first frame of the current environment       |
| • No nonlocal statement  
• "x" **is bound locally**                                                    | Re-bind name "x" to value 2 in the first frame of the current environment                         |
| • nonlocal x  
• "x" **is bound in a nonlocal frame**                                   | Re-bind "x" to 2 in the first nonlocal frame of the current environment in which "x" is bound    |
## 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 value 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 value 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 nonlocal frame of the current environment in which &quot;x&quot; is bound</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a nonlocal frame</td>
<td></td>
</tr>
</tbody>
</table>
## Assignment Statements

\[ x = 2 \]

<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 value 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 value 2 in the first frame of the current environment</td>
</tr>
<tr>
<td>No nonlocal statement</td>
<td>Re-bind &quot;x&quot; to 2 in the first nonlocal frame of the current environment in which &quot;x&quot; is bound</td>
</tr>
<tr>
<td>nonlocal x</td>
<td>Re-bind &quot;x&quot; to 2 in the first nonlocal frame of the current environment in which &quot;x&quot; is bound</td>
</tr>
<tr>
<td>&quot;x&quot; is bound in a nonlocal frame</td>
<td>Re-bind &quot;x&quot; to 2 in the first nonlocal frame of the current environment in which &quot;x&quot; is bound</td>
</tr>
<tr>
<td>&quot;x&quot; is not bound in a nonlocal frame</td>
<td>Re-bind &quot;x&quot; to 2 in the first nonlocal frame of the current environment in which &quot;x&quot; is bound</td>
</tr>
</tbody>
</table>
## 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 value 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 value 2 in the first frame of the current environment</td>
</tr>
<tr>
<td>• Nonlocal x</td>
<td>Re-bind &quot;x&quot; to 2 in the first nonlocal frame of the current environment in which &quot;x&quot; is bound</td>
</tr>
<tr>
<td>• Nonlocal x</td>
<td>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td>Re-bind name &quot;x&quot; to value 2 in the first frame of the current environment</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a nonlocal frame</td>
<td>Re-bind &quot;x&quot; to 2 in the first nonlocal frame of the current environment in which &quot;x&quot; is bound</td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound in a nonlocal frame</td>
<td>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
</tbody>
</table>
## 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 value 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 value 2 in the first frame of the current environment</td>
</tr>
<tr>
<td>• No nonlocal statement</td>
<td>Re-bind &quot;x&quot; to 2 in the first nonlocal frame of the current environment in which &quot;x&quot; is bound</td>
</tr>
<tr>
<td>• nonlocal x</td>
<td>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a nonlocal frame</td>
<td></td>
</tr>
<tr>
<td>• nonlocal x</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound in a nonlocal frame</td>
<td></td>
</tr>
</tbody>
</table>
## Assignment Statements

**x = 2**

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>
| • No nonlocal statement  
• "x" **is not** bound locally                                           | Create a new binding from name "x" to value 2 in the first frame of the current environment |
| • No nonlocal statement  
• "x" **is** bound locally                                              | Re-bind name "x" to value 2 in the first frame of the current environment |
| • nonlocal x  
• "x" **is** bound in a nonlocal frame                                   | Re-bind "x" to 2 in the first nonlocal frame of the current environment in which "x" is bound |
| • nonlocal x  
• "x" **is not** bound in a nonlocal frame                              | SyntaxError: no binding for nonlocal 'x' found                         |
| • nonlocal x  
• "x" **is** bound in a nonlocal frame  
• "x" also bound locally                                                  |                                                                        |
## Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>
| **No nonlocal statement**  
**"x" is not bound locally** | Create a new binding from name "x" to value 2 in the first frame of the current environment |
| **No nonlocal statement**  
**"x" is bound locally** | Re-bind name "x" to value 2 in the first frame of the current environment |
| **nonlocal x**  
**"x" is bound in a nonlocal frame** | Re-bind "x" to 2 in the first nonlocal frame of the current environment in which "x" is bound |
| **nonlocal x**  
**"x" is not bound in a nonlocal frame** | SyntaxError: no binding for nonlocal 'x' found |
| **nonlocal x**  
**"x" is bound in a nonlocal frame**  
**"x" also bound locally** | SyntaxError: name 'x' is parameter and nonlocal |
Python Particulars
def make_withdraw(balance):
    def withdraw(amount):
        # nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw
```python
def make_withdraw(balance):
    def withdraw(amount):
        # nonlocal balance
        if amount > balance:
            return 'Insufficient funds'

        balance = balance - amount

        return balance

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

UnboundLocalError: local variable 'balance' referenced before assignment
def make_withdraw(balance):
    def withdraw(amount):
        # nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw

UnboundLocalError: local variable 'balance' referenced before assignment

Python pre-computes which frame contains each name before executing the body of a function.
```python
def make_withdraw(balance):
    def withdraw(amount):
        # nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw
```

Python Particulars (demo)

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):
        # nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw

UnboundLocalError: local variable 'balance' referenced before assignment

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.
Accounts
Mutable Sequences
Mutable Sequences

```
Global frame

make_withdraw
withdraw

func make_withdraw(balance) [p=G]

list

0
75

f1: make_withdraw [p=G]

balance 100
withdraw
b
Return value

func withdraw(amount) [p=f1]

f2: withdraw [p=f1]

amount 25

Return value 75
```
Mutable Sequences

def make_withdraw(balance):
    b = [balance]
    def withdraw(amount):
        if amount > b[0]:
            return 'Insufficient funds'
        b[0] = b[0] - amount
        return b[0]
    return withdraw
Mutable Sequences

```python
def make_withdraw(balance):
    b = [balance]

def withdraw(amount):
    if amount > b[0]:
        return 'Insufficient funds'
    b[0] = b[0] - amount
    return b[0]
return withdraw
```
Mutable Sequences

```python
def make_withdraw(balance):
b = [balance]
def withdraw(amount):
    if amount > b[0]:
        return 'Insufficient funds'
b[0] = b[0] - amount
return b[0]
return withdraw
```
def make_withdraw(balance):
    b = [balance]
    def withdraw(amount):
        if amount > b[0]:
            return 'Insufficient funds'
        b[0] = b[0] - amount
        return b[0]
    return withdraw

Mutable Sequences

Mutable value can change

Name-value binding cannot change because there is no nonlocal statement
Multiple Mutable Functions
Multiple Mutable Functions

(demo)
Multiple Mutable Functions

```python
>>> brian = make_withdraw(100)
>>> marvin = make_withdraw(100000)
>>> brian(10)
90

>>> marvin(10000)
90000

>>> brian(100)
'Insufficient funds'

>>> marvin(100)
89900
```
Break!
Referential Transparency
Referential Transparency

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

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

\[
\text{mul}(\text{add}(2, \text{mul}(4, 6)), \text{add}(3, 5))
\]
• Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

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

\[
\text{mul}(\text{add}(2, 24), \text{add}(3, 5))
\]
Referential Transparency

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

\[
\text{mul}(\text{add}(2, \text{mul}(4, 6)), \text{add}(3, 5)) \\
\text{mul}(\text{add}(2, 24), \text{add}(3, 5)) \\
\text{mul}(26, \text{add}(3, 5))
\]
Referential Transparency

• 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**
Referential Transparency

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

\[
\begin{align*}
\text{mul}(\text{add}(2, \text{mul}(4, 6)), \text{add}(3, 5)) \\
\text{mul}(\text{add}(2, \phantom{1}24\phantom{1}), \text{add}(3, 5)) \\
\text{mul}(\phantom{1}26\phantom{1}, \text{add}(3, 5))
\end{align*}
\]

• Mutation operations violate the condition of referential transparency because they do more than just return a value; **they change the environment**
Mutating Linked Lists
Summary
Summary

- The nonlocal statement allows us to mutate name-value bindings in a nonlocal frame
Summary

• The nonlocal statement allows us to mutate name-value bindings in a nonlocal frame

• Mutation is a powerful tool, but it also makes reasoning about programs more difficult
Summary

• The nonlocal statement allows us to mutate name-value bindings in a nonlocal frame

• Mutation is a powerful tool, but it also makes reasoning about programs more difficult

• The truth is: we don't usually use nonlocal to build our own objects with mutable state
Summary

• The nonlocal statement allows us to mutate name-value bindings in a nonlocal frame

• Mutation is a powerful tool, but it also makes reasoning about programs more difficult

• The truth is: we don't usually use nonlocal to build our own objects with mutable state
  • We'll see another way next week
Summary

- The nonlocal statement allows us to mutate name-value bindings in a nonlocal frame
- Mutation is a powerful tool, but it also makes reasoning about programs more difficult
- The truth is: we don't usually use nonlocal to build our own objects with mutable state
  - We'll see another way next week
- Good luck on the midterm!