61A Lecture 11

Friday, September 23
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100
A Function with Behavior That Varies Over Time

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

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

Argument: amount to withdraw
Let's model a bank account that has a balance of $100

Return value: remaining balance

>>> withdraw(25)
75

Argument: amount to withdraw
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

Argument: amount to withdraw

Second withdrawal of the same amount
Let's model a bank account that has a balance of $100.

Return value: remaining balance

>>> withdraw(25)
75

Different return value!

>>> withdraw(25)
50

Argument: amount to withdraw

Second withdrawal of the same amount
A Function with Behavior That Varies Over Time

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

Return value: remaining balance

Argument: amount to withdraw

Different return value!

>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)

First withdrawal

Second withdrawal of the same amount
Let's model a bank account that has a balance of $100

Return value: remaining balance

Argument: amount to withdraw

Different return value!

>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'
A Function with Behavior That Varies Over Time

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

Return value: remaining balance

```python
>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
```

Argument: amount to withdraw

Different return value!

Second withdrawal of the same amount
Let's model a bank account that has a balance of $100.

Return value: remaining balance

>>> withdraw(25)
75

>>> withdraw(25)
50

Second withdrawal of the same amount

>>> withdraw(60)
'Insufficient funds'

Different return value!

>>> withdraw(15)
35

Argument:
amount to withdraw
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

>>> withdraw(25)
75

>>> withdraw(25)
50

Second withdrawal of the same amount

>>> withdraw(60)
'Insufficient funds'

Different return value!

>>> withdraw(15)
35

Where's this balance stored?

Friday, September 23, 2011
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

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35

Argument: amount to withdraw

Second withdrawal of the same amount

Different return value!

>>> withdraw = make_withdraw(100)

Where's this balance stored?
A Function with Behavior That Varies Over Time

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

Return value: remaining balance

Argument: amount to withdraw

Different return value!

Second withdrawal of the same amount

Where's this balance stored?

Within the function!

>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35

>>> withdraw = make_withdraw(100)
Persistent Local State
Persistent Local State

\[\text{withdraw(amount)}: \]
\[\text{function body to be revealed momentarily}\]
Persistent Local State

balance: 100
withdraw:

make_withdraw

withdraw(amount):

function body
to be revealed momentarily

Friday, September 23, 2011
Persistent Local State

make_withdraw:
withdraw:

balance: 100
withdraw:

make_withdraw:
withdraw:

make_withdraw(balance):

function body
to be revealed
momentarily

withdraw(amount):

function body
to be revealed
momentarily
Local State via Non-Local Assignment
Local State via Non-Local Assignment

```python
def make_withdraw(balance):
```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
Local State via Non-Local Assignment

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

Friday, September 23, 2011
Local State via Non-Local Assignment

```python
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
        Declare the name "balance" nonlocal
def make_withdraw(balance):
    
    """Return a withdraw function with a starting balance."""

    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            Declare the name "balance" nonlocal

        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'
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
```

Declare the name "balance" nonlocal
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
    Re-bind balance where it was bound previously
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

Declare the name "balance" nonlocal
Re-bind balance where it was bound previously
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
Local State via Non-Local Assignment

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

Demo
Local, Non-Local, and Global Frames
Local, Non-Local, and Global Frames

An environment
Local, Non-Local, and Global Frames

The global frame

An environment
Local, Non-Local, and Global Frames

- The global frame
- An environment
- Local frame
Local, Non-Local, and Global Frames

The global frame

Non-local frame

Local frame

An environment
Local, Non-Local, and Global Frames

- **The global frame**
- **Non-local frame**
- **Non-local frame**
- **Local frame**
- **An environment**
Local, Non-Local, and Global Frames

The global frame

Non-local frame

Non-local frame

An environment

First frame

Local frame
Local, Non-Local, and Global Frames

- The global frame
- Non-local frame
- Non-local frame
- Local frame
- First frame
- Second frame

An environment
Local, Non-Local, and Global Frames

Non-local frame

First non-local frame

First frame

Second frame

Non-local frame

The global frame

An environment
Local, Non-Local, and Global Frames

- **First frame**
- **Second frame**
- **First non-local frame**
- **Second (and last) non-local frame**

- **Local frame**
- **Non-local frame**
- **Non-local frame**
- **The global frame**

An environment
The Effect of Nonlocal Statements

nonlocal <name>
The Effect of Nonlocal Statements

nonlocal <name>

**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.
The Effect of Nonlocal Statements

nonlocal <name>

**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.

Python Docs: an "enclosing scope"
The Effect of Nonlocal Statements

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.
The Effect of Nonlocal Statements

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:

Python Docs: an "enclosing scope"
The Effect of Nonlocal Statements

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.
The Effect of Nonlocal Statements

```
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.
The Effect of Nonlocal Statements

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
The Effect of Nonlocal Statements

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://www.python.org/dev/peps/pep-3104/
The Many Meanings of Assignment Statements

\[ x = 2 \]
The Many Meanings of Assignment Statements

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

$x = 2$
The Many Meanings of Assignment Statements

\[ x = 2 \]

**Status**

- No nonlocal statement
- "x" is not bound locally
The Many Meanings of Assignment Statements

Status

• No nonlocal statement
• "x" is not bound locally

Effect

Create a new binding from name "x" to object 2 in the first frame of the current environment.
The Many Meanings of Assignment Statements

\[ x = 2 \]

**Status**

- No nonlocal statement
- "x" **is not** bound locally

**Effect**

Create a new binding from name "x" to object 2 in the first frame of the current environment.

- No nonlocal statement
- "x" **is** bound locally
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</td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound locally</td>
<td>the current environment.</td>
</tr>
<tr>
<td>• No nonlocal statement</td>
<td>Re-bind name &quot;x&quot; to object 2 in the first frame of the current env.</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td></td>
</tr>
</tbody>
</table>
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 env.</td>
</tr>
<tr>
<td>• No nonlocal statement</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td></td>
</tr>
<tr>
<td>• nonlocal x</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a non-local frame</td>
<td></td>
</tr>
</tbody>
</table>
### The Many Meanings of 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 object 2 in the first frame of the current environment. |
| • No nonlocal statement  
  • "x" **is bound locally**                                              | Re-bind name "x" to object 2 in the first frame of the current env.    |
| • nonlocal x  
  • "x" **is bound in a non-local frame**                               | Re-bind "x" to 2 in the first non-local frame of the current environment in it is bound. |
The Many Meanings of 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 object 2 in the first frame of the current environment. |
| • No nonlocal statement  
• "x" is bound locally | Re-bind name "x" to object 2 in the first frame of the current env. |
| • nonlocal x  
• "x" is bound in a non-local frame | Re-bind "x" to 2 in the first non-local frame of the current environment in it is bound. |
| • nonlocal x  
• "x" is not bound in a non-local frame | |

x = 2
# The Many Meanings of 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 object 2 in the first frame of the current environment. |
| • No nonlocal statement  
• "x" is bound locally | Re-bind name "x" to object 2 in the first frame of the current env. |
| • nonlocal x  
• "x" is bound in a non-local frame | Re-bind "x" to 2 in the first non-local frame of the current environment in it is bound. |
| • nonlocal x  
• "x" is not bound in a non-local frame | SyntaxError: no binding for nonlocal 'x' found |
## The Many Meanings of 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 object 2 in the first frame of the current environment. |
| • No nonlocal statement  
• "x" **is** bound locally | Re-bind name "x" to object 2 in the first frame of the current env. |
| • nonlocal x  
• "x" **is** bound in a non-local frame | Re-bind "x" to 2 in the first non-local frame of the current environment in it is bound. |
| • nonlocal x  
• "x" **is not** bound in a non-local frame | SyntaxError: no binding for nonlocal 'x' found |
| • nonlocal x  
• "x" **is** bound in a non-local frame  
• "x" also bound locally | |
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; <strong>is not</strong> bound locally</td>
<td>Re-bind name &quot;x&quot; to object 2 in the first frame of the current env.</td>
</tr>
<tr>
<td>• nonlocal <code>x</code></td>
<td>Re-bind &quot;x&quot; to 2 in the first non-local frame of the current environment in it is bound.</td>
</tr>
<tr>
<td>• nonlocal <code>x</code></td>
<td>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
<tr>
<td>• nonlocal <code>x</code></td>
<td>SyntaxError: name 'x' is parameter and nonlocal</td>
</tr>
</tbody>
</table>

\[
x = 2
\]
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
def mutant(y):
    y, x = y + 1, y + 2
    return ninja(y) / 2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
Assignment Review: Teenage Mutant Ninja Turtles

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```

Friday, September 23, 2011
Assignment Review: Teenage Mutant Ninja Turtles

mutant(y):
   y, x = y+1, y+2
   return ninja(y)/2

ninja(x):
   return x + 2

turtle(x):
   return x * y + 2

mutant(y):
   y, x = y+1, y+2
   return ninja(y)/2

def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2
y, ninja = 5, turtle

mutant(y)
Assignment Review: Teenage Mutant Ninja Turtles

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```

Friday, September 23, 2011
Assignment Review: Teenage Mutant Ninja Turtles

mutant(y):
  y, x = y+1, y+2
  return ninja(y)/2

ninja(x):
  return x + 2

turtle(x):
  return x * y + 2

y, ninja = 5, turtle

mutant(y)
```
Assignment Review: Teenage Mutant Ninja Turtles

def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
Mutant(y):
  y, x = y+1, y+2
  return ninja(y)/2

Ninja(x):
  return x + 2

Turtle(x):
  return x * y + 2

Def mutant(y):
  y, x = y+1, y+2
  return ninja(y)/2

Def ninja(x):
  return x + 2

Def turtle(x):
  return x * y + 2

y, ninja = 5, turtle

Mutant(y)
Assignment Review: Teenage Mutant Ninja Turtles

```
mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

ninja(x):
    return x + 2

turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja, turtle = 5, turtle
mutant(y)
The diagram illustrates the recursive function structure for the Teenage Mutant Ninja Turtles assignment. The function `mutant(y)` is defined as follows:

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2
```

The function `ninja(x)` is defined as:

```python
def ninja(x):
    return x + 2
```

The function `turtle(x)` is defined as:

```python
def turtle(x):
    return x * y + 2
```

The starting values are:
- `y = 5`
- `x = 7`

After the first call to `mutant(y)`, `y` becomes 6 and `x` becomes 8. The values are calculated recursively until the base case is reached.

**Base Case:**
- `ninja(6) = 6 + 2 = 8`
- `turtle(8) = 8 * 6 + 2 = 48 + 2 = 50`

The final result is 50.
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
Assignment Review: Teenage Mutant Ninja Turtles

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```

Friday, September 23, 2011
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
Assignment Review: Teenage Mutant Ninja Turtles

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```

Friday, September 23, 2011
Assignment Review: Teenage Mutant Ninja Turtles

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

mutant(y)

---

Friday, September 23, 2011
Assignment Review: Teenage Mutant Ninja Turtles

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```

Friday, September 23, 2011
Assignment Review: Teenage Mutant Ninja Turtles

def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
Assignment Review: Teenage Mutant Ninja Turtles

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

ty, ninja = 5, turtle
mutant(y)
```

Friday, September 23, 2011
Assignment Review: Teenage Mutant Ninja Turtles

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
Assignment Review: Teenage Mutant Ninja Turtles

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```

Friday, September 23, 2011
Assignment Review: Teenage Mutant Ninja Turtles

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```

- `x`: 6
- `y`: 6
- `x`: 7
- `mutant(y)`: `y, x = y+1, y+2
  return ninja(y)/2`
- `ninja(x)`: `return x + 2`
- `turtle(x)`: `return x * y + 2`

---

Friday, September 23, 2011
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles
Assignment Review: Teenage Mutant Ninja Turtles

```
mutant(y):
  y, x = y+1, y+2
  return ninja(y)/2

ninja(x):
  return x + 2

turtle(x):
  return x * y + 2
```

```
def mutant(y):
  y, x = y+1, y+2
  return ninja(y)/2

def ninja(x):
  return x + 2

def turtle(x):
  return x * y + 2
```

---

Friday, September 23, 2011
Assignment Review: Teenage Mutant Ninja Turtles

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```

![Diagram showing the flow of the program with values assigned to variables and function calls.]
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
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

def mutant(y):
    y, x = y + 1, y + 2
    return ninja(y) / 2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
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

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
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

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
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

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
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

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
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

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
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

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
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

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
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

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
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 6 * 5 + 2 = 32

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
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 6 * 5 + 2 = 32
  - Return 32 / 2 = 16.0

```python
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle
mutant(y)
```
Environment Diagram of Withdraw

make_withdraw:

make_withdraw
Environment Diagram of Withdraw

```python
wd = make_withdraw(20)
wd(5)
```
Environment Diagram of Withdraw

```
wd = make_withdraw(20)
wdd(5)
```
Environment Diagram of Withdraw

make_withdraw:
wd:

balance: 20
withdraw:

withdraw(amount):
...

wd = make_withdraw(20)
wd(5)
Environment Diagram of Withdraw

```
make_withdraw:
  wd:
make_withdraw

withdraw:
  amount: 5
  balance: 20
withdraw:
  make_withdraw

withdraw(amount):
  ...
```

```
wd = make_withdraw(20)
w(5)
```
Environment Diagram of Withdraw

```python
wd = make_withdraw(20)
wd(5)

if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

Friday, September 23, 2011
make_withdraw:

withdraw:

... 

balance:

wd:

make_withdraw

wd(5)

20

Environment Diagram of Withdraw

wd = make_withdraw(20)

wd(5)

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

withdraw(amount):

... 

Friday, September 23, 2011
Environment Diagram of Withdraw

make_withdraw:

wd:

withdraw:

wd(5)

amount: 5

balance: 20

withdraw:

balance:

withdraw(amount):

...
Environment Diagram of Withdraw

```python
wd = make_withdraw(20)
wd(5)

if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance

wd = make_withdraw(20)
wd(5)
```

Monday, September 23, 2011
Environment Diagram of Withdraw

```
make_withdraw:
wd:

balance: 15
withdraw:

withdraw(amount):
...

amount: 5

wd = make_withdraw(20)
wd(5)

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

wd = make_withdraw(20)
wd(5)
```
Calling a Withdraw Function Twice

```
wd = make_withdraw(20)
wd(5)
wd(3)
```
Calling a Withdraw Function Twice

```python
wd = make_withdraw(20)
wd(5)
wd(3)
```

```
withdraw(amount):
...
```

`wd = make_withdraw(20)
wd(5)
wd(3)`
Calling a Withdraw Function Twice

```
wd = make_withdraw(20)
wd(5)
wd(3)
```

```
withdraw(amount):
...
```

```
wd = make_withdraw(20)
w(5)
w(3)
```
Calling a Withdraw Function Twice

```
make_withdraw:
wd:

amount: 5
withdraw

balance: 15
withdraw:

amount: 3
withdraw

nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wds
```
Calling a Withdraw Function Twice

```
make_withdraw:
  wd:

amount: 5
withdraw
balance: 15
withdraw:

amount: 3
withdraw

nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
wd(3)
```
Calling a Withdraw Function Twice

```python
wd = make_withdraw(20)
wd(5)
wd(3)

amount: 5
withdraw

balance: 12
withdraw:

amount: 3
withdraw:

wd:

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

wd = make_withdraw(20)
wd(5)
wd(3)
```

Friday, September 23, 2011
Calling a Withdraw Function Twice

```python
make_withdraw:
wd:

amount: 5
withdraw

balance: 12
withdraw:

amount: 3
withdraw

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

wd = make_withdraw(20)
wd(5)
wd(3)
```
Calling a Withdraw Function Twice

```python
make_withdraw:
wd = make_withdraw(20)
wd(5)
wd(3)
```

```python
amount: 5
withdraw
```

```python
amount: 3
withdraw
```

```python
make_withdraw
```

```python
withdraw(amount):
...
```

```python
balance: 12
```

```python
balance: 12
```

```python
balance:
```

```python
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```python
wd = make_withdraw(20)
wd(5)
wd(3)
```

Friday, September 23, 2011
Creating Two Different Withdraw Functions

```python
make_withdraw:
wd:

balance: 12
withdraw:

withdraw(amount):

wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```
Creating Two Different Withdraw Functions

```
make_withdraw:
  wd:
```

```
balance: 12
withdraw:
```

```
balance: 7
```

```
make_withdraw:
```

```
wd = make_withdraw(20)
wd(5)
wd(3)
```

```
wd2 = make_withdraw(7)
wd2(6)
```

Friday, September 23, 2011
Creating Two Different Withdraw Functions

```python
make_withdraw:
wd:

balance: 12
withdraw:

wd = make_withdraw(20)
wd(5)
wd(3)

wd2 = make_withdraw(7)
wd2(6)
```

Friday, September 23, 2011
Creating Two Different Withdraw Functions

```python
make_withdraw:
  wd:
  balance: 12
  withdraw:

balance: 7
make_withdraw

make_withdraw(7)
def withdraw(amount):
  ...
  return withdraw

wd = make_withdraw(20)
wd(5)
wd(3)

wd2 = make_withdraw(7)
wd2(6)
```
Creating Two Different Withdraw Functions

```python
def withdraw(amount):
    ...,
    return withdraw

wd = make_withdraw(20)
wd(5)
wd(3)

wd2 = make_withdraw(7)
wd2(6)
```
Creating Two Different Withdraw Functions

```python
def withdraw(amount):
    ... return withdraw

wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```
Creating Two Different Withdraw Functions

```python
make_withdraw:
wd:
wd2:

balance: 12
withdraw:

balance: 7
withdraw:

make_withdraw:

make_withdraw:

make_withdraw(7)
def withdraw(amount):
    ...
    return withdraw

wd = make_withdraw(20)
wd(5)
wd(3)

wd2 = make_withdraw(7)
wd2(6)
```
Creating Two Different Withdraw Functions

```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```
Creating Two Different Withdraw Functions

```python
wd = make_withdraw(20)
w(5)
w(3)
wd2 = make_withdraw(7)
wd2(6)
```
Creating Two Different Withdraw Functions

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

Friday, September 23, 2011
Creating Two Different Withdraw Functions

```python
make_withdraw:
wd:
wd2:

withdraw:
amount: 6

balance: 12
withdraw:

balance: 7
withdraw:

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

wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```
Creating Two Different Withdraw Functions

```python
make_withdraw:

wd:

wd2:

withdraw:

balance:

withdraw:

balance:

withdraw:

wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)

amount: 6
withdraw
balance: 7
withdraw:

balance: 12
withdraw:

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

wd2(6)
```

Friday, September 23, 2011
Creating Two Different Withdraw Functions

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

```python
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```
Creating Two Different Withdraw Functions

```
make_withdraw:
wd:  
wd2:

withdraw:

balance: 12
withdraw:

withdraw(amount):

wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)

nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```
The Benefit of Non-Local Assignment
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 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.
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.
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**.

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

<table>
<thead>
<tr>
<th>John's Account</th>
<th>Steven's Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10</td>
<td>$1,000,000</td>
</tr>
</tbody>
</table>
Multiple References to a Single Withdraw Function

```
wd = make_withdraw(12)
w2 = wd
w2(1)
w(1)
```
Multiple References to a Single Withdraw Function

```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```
Multiple References to a Single Withdraw Function

\[
\text{make\_withdraw:} \quad \text{balance: 12} \\
\text{wd:} \quad \text{withdraw:} \\
\text{wd2:} \\
\]

\[
\text{withdraw(\text{amount}):} \\
\]

\[
\text{wd} = \text{make\_withdraw}(12) \\
\text{wd2} = \text{wd} \\
\text{wd2}(1) \\
\text{wd}(1) \\
\]
Multiple References to a Single Withdraw Function

```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```
Multiple References to a Single Withdraw Function

withdraw(amount):

wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
Multiple References to a Single Withdraw Function

```python
make_withdraw:
    wd:
    wd2:

withdraw:
    wd2(1)
    wd(1)

amount: 1

balance: 12

withdraw:
    make_withdraw

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

wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
Multiple References to a Single Withdraw Function

```python
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)

amount: 1
withdraw
balance: 11
withdraw:
make_withdraw

make_withdraw:
wd:
wd2:

withdraw(amount):
make_withdraw

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

wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

Friday, September 23, 2011
Multiple References to a Single Withdraw Function

```python
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)

amount: 1
withdraw

balance: 11
make_withdraw:
wd:
wd2:
withdraw:

withdraw(amount):

nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

Friday, September 23, 2011
Multiple References to a Single Withdraw Function

```python
wd = make_withdraw(12)
w2 = wd
wd2(1)
wd(1)
```
Multiple References to a Single Withdraw Function

```python
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```
Multiple References to a Single Withdraw Function

```python
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

Friday, September 23, 2011
Multiple References to a Single Withdraw Function

 withdrawn(amount):

 - `wd`: 
 - `wd2`: 

 `amount`: 1

 `balance`: 11

 `wd(1)`

```python
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
Multiple References to a Single Withdraw Function

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

wd = make_withdraw(12)
wd2 = wd
wd2(amount)
wd(amount)
```

Friday, September 23, 2011
Multiple References to a Single Withdraw Function

```python
withdraw(amount):
    wd:  
    wd2:  
    wd = make_withdraw(12)
    wd2 = wd
    wd2(1)
    wd(1)
    nonlocal balance
    if amount > balance:
        return 'Insufficient funds'
    balance = balance - amount
    return balance
```

Friday, September 23, 2011
Sameness and Change
Sameness and Change

• As long as we never modify objects, we can regard a compound object to be precisely the totality of its pieces.
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.
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.
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.
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.
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.
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>
</tr>
</thead>
<tbody>
<tr>
<td>$10</td>
</tr>
</tbody>
</table>
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

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

\[ \text{mul}(\text{add}(2, \text{mul}(4, 6)), \text{add}(3, 5)) \]
• An expression is **referentially transparent** if its value does not change when we substitute one of its subexpression with the value of that subexpression.

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

\[
mul(add(2, 24), add(3, 5))
\]
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.

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

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

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

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

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

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

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

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

\[
mul(add(2, 24), add(3, 5))
\]

\[
mul(26, add(3, 5))
\]

• 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.