Announcements
Announcements

- Homework 4 due Tuesday 10/8 @ 11:59pm.
Announcements

• Homework 4 due Tuesday 10/8 @ 11:59pm.

• Project 2 due Thursday 10/10 @ 11:59pm.
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.
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.
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!
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.
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
A Function with Behavior That Varies Over Time

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

>>> withdraw(25)
A Function with Behavior That Varies Over Time

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

```python
>>> withdraw(25)
75
```
Let's model a bank account that has a balance of $100

>>> withdraw(25)
75
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
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)
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

>>> withdraw(25)

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

>>> withdraw(25)
75

>>> withdraw(25)
50

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
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

>>> withdraw(60)
Let's model a bank account that has a balance of $100.

Return value: remaining balance

Different return value!

>>> withdraw(25)
75

Argument: amount to withdraw

>>> withdraw(25)
50

Second withdrawal of the same amount

>>> 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

Argument: amount to withdraw

Different return value!

Second withdrawal of the same amount

>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
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

Different return value!

Second withdrawal of the same amount

>>> withdraw(60)
'Insufficient funds'

>>> 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

- Return value: remaining balance
- Different return value!

>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35

Argument: amount to withdraw
Second withdrawal of the same amount
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

>>> withdraw(25)
75

Argument: amount to withdraw

>>> withdraw(25)
50

Second withdrawal of the same amount

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35

Different return value!

>>> withdraw = make_withdraw(100)
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 parent frame of the function!
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

Different return value!

Argument: amount to withdraw

Second withdrawal of the same amount

Where's this balance stored?

Within the parent frame of the function!

A function has a body and a parent environment

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

Example: [http://goo.gl/cUC09s](http://goo.gl/cUC09s)
Persistent Local State Using Environments

Example: [http://goo.gl/cUC09s](http://goo.gl/cUC09s)
Persistent Local State Using Environments

Example: http://goo.gl/cUC09s
Persistent Local State Using Environments

Example:

http://goo.gl/cUC09s

A function with a parent frame

The parent contains local state

Every call changes the balance
Persistent Local State Using Environments

Example: http://goo.gl/cUC09s

A function with a parent frame
The parent contains local state
Every call changes the balance

All calls to the same function have the same parent
Reminder: Local Assignment

```python
def percent_difference(x, y):
    difference = abs(x-y)
    return 100 * difference / x

diff = percent_difference(40, 50)
```

Example: http://goo.gl/Wxpg5Z
Reminder: Local Assignment

Example:

```python
def percent_difference(x, y):
    difference = abs(x-y)
    return 100 * difference / x
diff = percent_difference(40, 50)
```

Assignment binds name(s) to value(s) in the first frame of the current environment

Example: [http://goo.gl/Wxpg5Z](http://goo.gl/Wxpg5Z)
Reminder: Local Assignment

Assignment binds name(s) to value(s) in the first frame of the current environment.

Example:

```python
def percent_difference(x, y):
    difference = abs(x - y)
    return 100 * difference / x

diff = percent_difference(40, 50)
```

Global frame:

<table>
<thead>
<tr>
<th>percent_difference</th>
</tr>
</thead>
</table>

| func percent_difference(x, y) |

<table>
<thead>
<tr>
<th>percent_difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>x 40</td>
</tr>
<tr>
<td>y 50</td>
</tr>
<tr>
<td>difference 10</td>
</tr>
</tbody>
</table>
Reminder: Local Assignment

**Execution rule for assignment statements:**

```
def percent_difference(x, y):
    difference = abs(x-y)
    return 100 * difference / x
diff = percent_difference(40, 50)
```

Assignment binds name(s) to value(s) in the first frame of the current environment
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.

Example: [http://goo.gl/Wxpg5Z](http://goo.gl/Wxpg5Z)
Non-Local Assignment & Persistent Local State
Non-Local Assignment & Persistent Local State

```python
def make_withdraw(balance):
```

```
Non-Local Assignment & Persistent Local State

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


Non-Local Assignment & Persistent Local State

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

    def withdraw(amount):
```

7
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:
Non-Local Assignment & Persistent Local State

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

def withdraw(amount):
    nonlocal balance
    if amount > balance:
        return 'Insufficient funds'
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
```
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
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.
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.
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)
Non-Local Assignment
The Effect of Nonlocal Statements

nonlocal <name>
The Effect of Nonlocal Statements

nonlocal <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.
The Effect of Nonlocal Statements

nonlocal <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.
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.

Python Docs: an "enclosing scope"
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:*
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.
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.
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.

[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://docs.python.org/release/3.1.3/reference/simple_stmts.html#the-nonlocal-statement]
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.

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>
<tbody>
<tr>
<td></td>
<td>$x = 2$</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></td>
</tr>
<tr>
<td>• &quot;x&quot; <em>is not</em> 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></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. |

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>
| • No nonlocal statement  
  • "x" is bound locally | |

x = 2
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></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 env.</td>
</tr>
<tr>
<td>• &quot;x&quot; is <strong>not</strong> bound locally</td>
<td></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 <strong>bound</strong> locally</td>
<td></td>
</tr>
<tr>
<td>• <strong>nonlocal</strong> x</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; is <strong>bound in a non-local frame</strong></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 which 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 which it is bound. |
| • nonlocal x  
• "x" **is not** bound in a non-local frame | |

```python
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. |

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>
| *No nonlocal statement*  
*x* is bound locally | Re-bind name "x" to object 2 in the first frame of the current env. |

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>
| **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 which it is bound. |

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>
| **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 which 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

![Assignment Statement Example]

<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 which 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 | SyntaxError: name 'x' is parameter and nonlocal |
Python Particulars

Example: http://goo.gl/b0Vzc6
Python Particulars

Python pre-computes which frame contains each name before executing the body of a function.

Example: [http://goo.gl/b0Vzc6](http://goo.gl/b0Vzc6)
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.

Example: [http://goo.gl/bOVzc6](http://goo.gl/bOVzc6)
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 = balance - amount
        return balance
    return withdraw

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

Example: [http://goo.gl/bOVzc6](http://goo.gl/bOVzc6)
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 = balance - amount
        return balance
    return withdraw

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

Example: [http://goo.gl/b0Vzc6](http://goo.gl/b0Vzc6)
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 = balance - amount
        return balance
    return withdraw

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

UnboundLocalError: local variable 'balance' referenced before assignment

Example: http://goo.gl/bOVzc6
Mutable values can be changed \textit{without} a nonlocal statement.

\begin{verbatim}
def make_withdraw_list(balance):
    b = [balance]
    def withdraw(amount):
        if amount > b[0]:
            return 'Insufficient funds'
        b[0] = b[0] - amount
        return b[0]
    return withdraw
withdraw = make_withdraw_list(100)
withdraw(25)
\end{verbatim}

Example: \url{http://goo.gl/y4TyFZ}
Mutable values can be changed \textit{without} a nonlocal statement.

Example: \url{http://goo.gl/y4TyFZ}
Mutable Values & Persistent Local State

Mutable values can be changed **without** a nonlocal statement.

Example: [http://goo.gl/y4TyFZ](http://goo.gl/y4TyFZ)
Multiple Mutable Functions

(Demo)
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>
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.
Referential Transparency, Lost
Referential Transparency, Lost

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

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

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

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

```
mul(add(2, mul(4, 6)), add(3, 5))
```

```
mul(add(2, 24), add(3, 5))
```
Referential Transparency, Lost

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

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

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

\[
\text{mul}(26, \text{add}(3, 5))
\]
• Expressions are **referentially transparent** if substituting an expression with its value does not change the meaning of a program.

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

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

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

• Mutation operations violate the condition of referential transparency because they do more than just return a value; they change the environment.
Referential Transparency, Lost

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

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

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

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

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

(Demo)