Announcements

• Homework 4 due Tuesday 10/7 @ 11:59pm
Announcements

• Homework 4 due Tuesday 10/7 @ 11:59pm

• Project 2 due Thursday 10/9 @ 11:59pm
Announcements

• Homework 4 due Tuesday 10/7 @ 11:59pm

• Project 2 due Thursday 10/9 @ 11:59pm

  • Project Party Monday 5pm–7pm in 271, 273, and 275 Soda (labs)
Announcements

• Homework 4 due Tuesday 10/7 @ 11:59pm

• Project 2 due Thursday 10/9 @ 11:59pm

  ▪ Project Party Monday 5pm–7pm in 271, 273, and 275 Soda (labs)

  ▪ Extra credit point for submitting your project at least 24 hours before the deadline
Announcements

• Homework 4 due Tuesday 10/7 @ 11:59pm

• Project 2 due Thursday 10/9 @ 11:59pm
  ▪ Project Party Monday 5pm–7pm in 271, 273, and 275 Soda (labs)
  ▪ Extra credit point for submitting your project at least 24 hours before the deadline

• Improving lab and discussion questions
Announcements

• Homework 4 due Tuesday 10/7 @ 11:59pm
• Project 2 due Thursday 10/9 @ 11:59pm
  • Project Party Monday 5pm–7pm in 271, 273, and 275 Soda (labs)
  • Extra credit point for submitting your project at least 24 hours before the deadline
• Improving lab and discussion questions
• Tips for approaching computer science problems
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
```
A Function with Behavior That Varies Over Time

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

```python
>>> 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

>>> withdraw(25)
75

Return value: remaining balance

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

Return value: remaining balance

>>> withdraw(25)
75

>>> withdraw(25)
50
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
A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100. The function `withdraw` takes an argument `amount to withdraw` and returns the `remaining balance`. The first withdrawal of $25 leaves a balance of $75.

\[ >>> \text{withdraw}(25) \]
\[ 75 \]

The second withdrawal of $25 leaves a balance of $50.

\[ >>> \text{withdraw}(25) \]
\[ 50 \]

Different return value!
A Function with Behavior That Varies Over Time

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

Return value: remaining balance

>>> withdraw(25)
75

Different return value!

>>> withdraw(25)
50

>>> withdraw(60)

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

Argument: amount to withdraw

Return value: remaining balance

>>> withdraw(25)
75

Second withdrawal of the same amount

>>> withdraw(25)
50

Different return value!

>>> 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

>>> withdraw(25)
75

>>> withdraw(25)
50

Different return value!

Second withdrawal of the same amount

Argument: amount to withdraw

'Insufficient funds'

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

>>> withdraw(25)
50

'Insufficient funds'

Different return value!

Argument: amount to withdraw

Second withdrawal of the same amount

>>> withdraw(60)
'Insufficient funds'

Where's this balance stored?

>>> 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

>>> withdraw(25) 75

>>> withdraw(25) 50

Second withdrawal of the same amount

>>> withdraw(60) 'Insufficient funds'

Different return value!

Where's this balance stored?

>>> withdraw = make_withdraw(100)

Within the parent frame of the function!
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'

Where's this balance stored?

>>> withdraw = make_withdraw(100)

Within the parent frame of the function!

A function has a body and a parent environment
Persistent Local State Using Environments

Interactive Diagram
Persistent Local State Using Environments

The parent frame contains the balance, the local state of the withdraw function.
Persistent Local State Using 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.
Persistent Local State Using Environments

The parent frame contains the balance, the local state of the withdraw function.

All calls to the same function have the same parent.

Every call decreases the same balance by (a possibly different) amount.

Interactive Diagram
Reminder: Local Assignment

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

diff = percent_difference(40, 50)
```

**Global frame**

```plaintext
func percent_difference(x, y) [parent=Global]

f1: percent_difference [parent=Global]

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

```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.
Reminder: Local Assignment

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

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

diff = percent_difference(40, 50)
```

Global frame

```
f1: percent_difference [parent=Global]

x 40
y 50
```

Interactive Diagram
Reminder: Local Assignment

Execution rule for assignment statements:
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 to the resulting values in the current frame
Non-Local Assignment & Persistent Local State
Non-Local Assignment & Persistent Local State

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


Non-Local Assignment & Persistent Local State

```python
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):
        pass
Non-Local Assignment & Persistent Local State

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

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:
```

7
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'
```

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
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
```
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
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.
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

\[
\text{nonlocal } <\text{name}>, <\text{name}>, \ldots
\]

**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.

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.

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>x = 2</td>
<td></td>
</tr>
</tbody>
</table>
## The Many Meanings of Assignment Statements

$$\textbf{x} = 2$$

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No nonlocal statement</td>
<td>• “\textit{x}” is not bound locally</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>
<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></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></td>
</tr>
<tr>
<td>---------------------------------------------</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 environment</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td></td>
</tr>
</tbody>
</table>
# The Many Meanings of 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 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 environment |
| • nonlocal x  
• "x" is bound in a non-local frame | |
### 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></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 environment</td>
</tr>
<tr>
<td>• &quot;x&quot; <strong>is bound locally</strong></td>
<td></td>
</tr>
<tr>
<td>• nonlocal x</td>
<td>Re-bind &quot;x&quot; to 2 in the first non-local frame of the current environment in which it is bound</td>
</tr>
<tr>
<td>• &quot;x&quot; <strong>is 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>
<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 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 non-local frame of the current environment in which it is bound</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a non-local frame</td>
<td></td>
</tr>
<tr>
<td>• nonlocal x</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; is not 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 environment |
| • 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 |
### 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>

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No nonlocal statement</td>
<td>Re-bind name &quot;x&quot; to object 2 in the first frame of the current environment</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• nonlocal x</td>
<td>Re-bind &quot;x&quot; to 2 in the first non-local frame of the current environment in which it is bound</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a non-local frame</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• nonlocal x</td>
<td>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound in a non-local frame</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• nonlocal x</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a non-local frame</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; also 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 |
| • No nonlocal statement  
  • "x" is bound locally | Re-bind name "x" to object 2 in the first frame of the current environment |
| • 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
Python Particulars

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

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.
Python Particulars

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.

```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)
```
Python Particulars

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.

```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)
```
Python Particulars

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.

```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
Mutable Values & Persistent Local State

Mutable values can be changed \textit{without} a nonlocal statement.
Mutable Values & Persistent Local State

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

```python
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)
```
Mutable Values & Persistent Local State

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

```python
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)
```
Mutable Values & Persistent Local State

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

```python
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)
```
Mutable Values & Persistent Local State

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

```
Global frame

make_withdraw_list
withdraw

f1: make_withdraw_list [parent=Global]

balance 100
withdraw
Return value

f2: withdraw [parent=f1]

amount 25
Return value 75

func make_withdraw_list(balance) [parent=Global]

list 0 75

func withdraw(amount) [parent=f1]

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)
```

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

**Interactive Diagram**
Mutable values can be changed \textit{without} a nonlocal statement.

\begin{itemize}
  \item \textbf{f1: \texttt{make}\_withdraw\_list} [parent=Global]
  \item \textbf{f2: \texttt{withdraw}} [parent=f1]
\end{itemize}
Multiple Mutable Functions

(Demo)
Referential Transparency, Lost

Interactive Diagram
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.

```
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.

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

\[
\text{mul}(\text{add}(2, 24), \text{add}(3, 5))
\]
Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

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

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

\[
\text{mul}(26, \text{add}(3, 5))
\]
Expressions are \textit{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, 24), \text{add}(3, 5)) \\
\text{mul}(26, \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.
Expressions are \textit{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; \textit{they change the environment}. 

\textit{Interactive Diagram}
Expressions are **referentially transparent** if substituting an expression with its value does not change the meaning of a program.

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

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