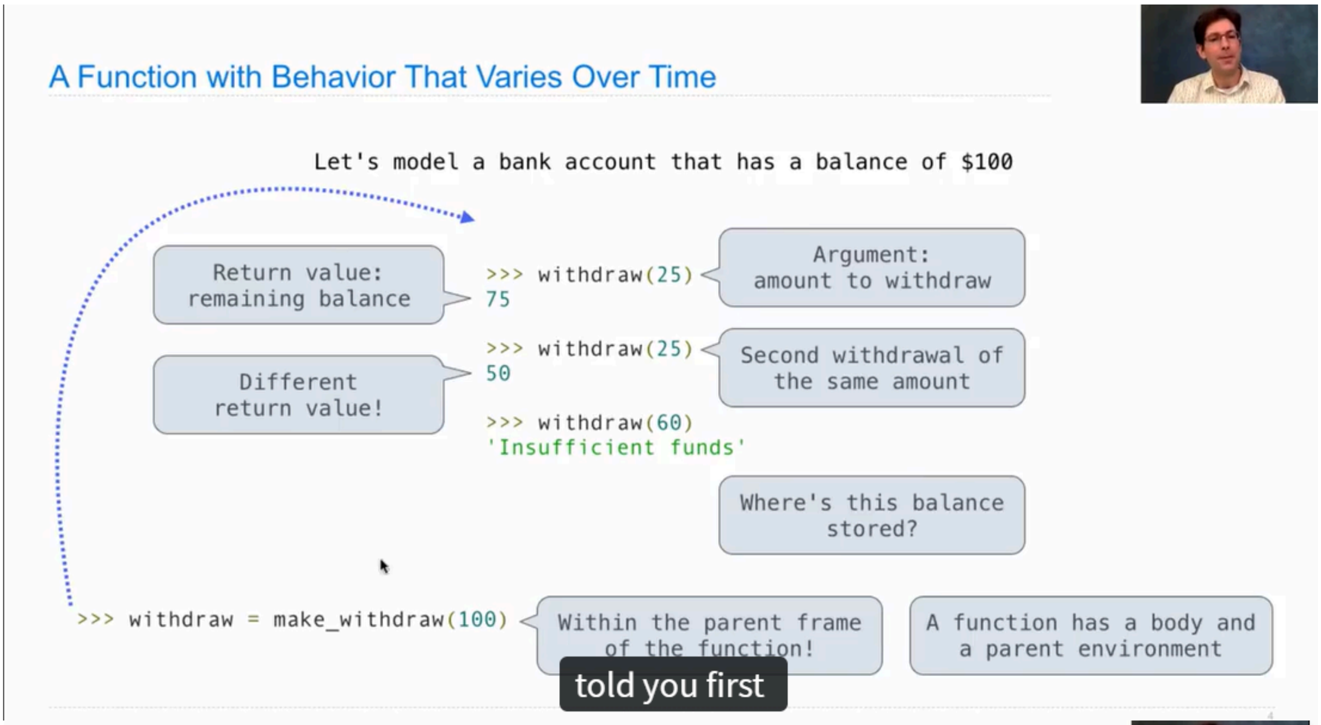
Lecture 16.Mutable functions

Mutable functions

functions with values associated to it that could change.



A Function with Behavior That Varies Over Time

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

```
>>> withdraw(25)
75
>>> withdraw(25)
50
>>> withdraw(60)
'Insufficient funds'
```

Return value: remaining balance

Argument: amount to withdraw

Different return value!

Second withdrawal of the same amount

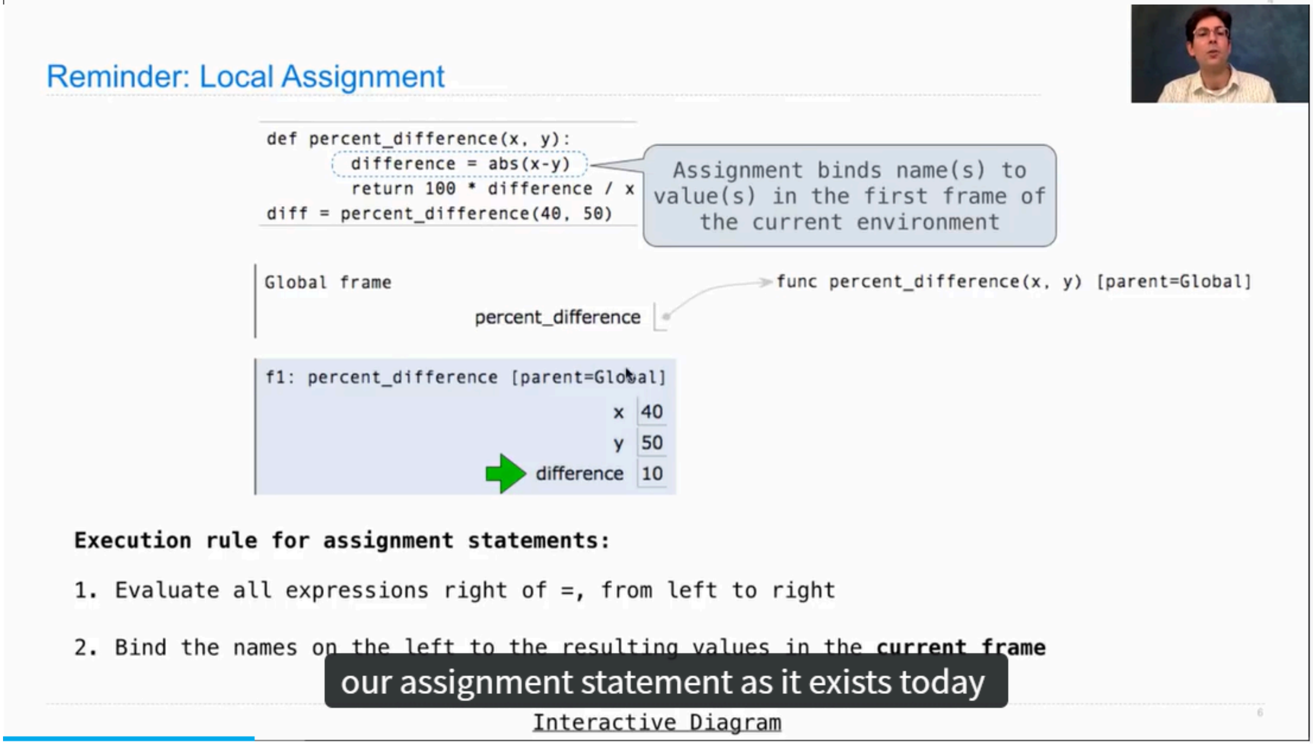
Where's this balance stored?

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

Within the parent frame of the function!

A function has a body and a parent environment

told you first



Reminder: Local Assignment

```
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

Global frame
percent_difference
func percent_difference(x, y) [parent=Global]
f1: percent_difference [parent=Global]
x 40
y 50
difference 10

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**

our assignment statement as it exists today

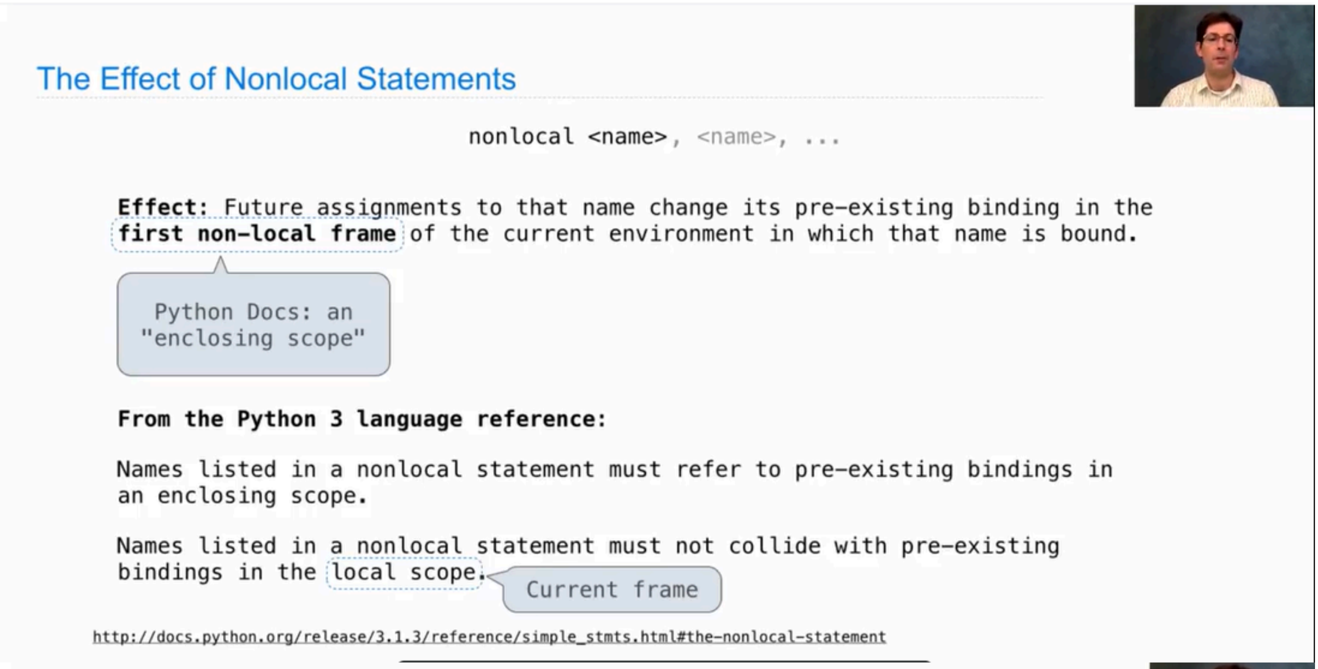
Interactive Diagram

```
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
```

如果涉及高阶函数创造的函数涉及外部的引用（比如此处的nonlocal声明），那么被引用的数据所在的frame就不会消亡（更进一步地说，是这些变量是被python存在了一个特殊的闭包环境中）

Non-local assignment



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"

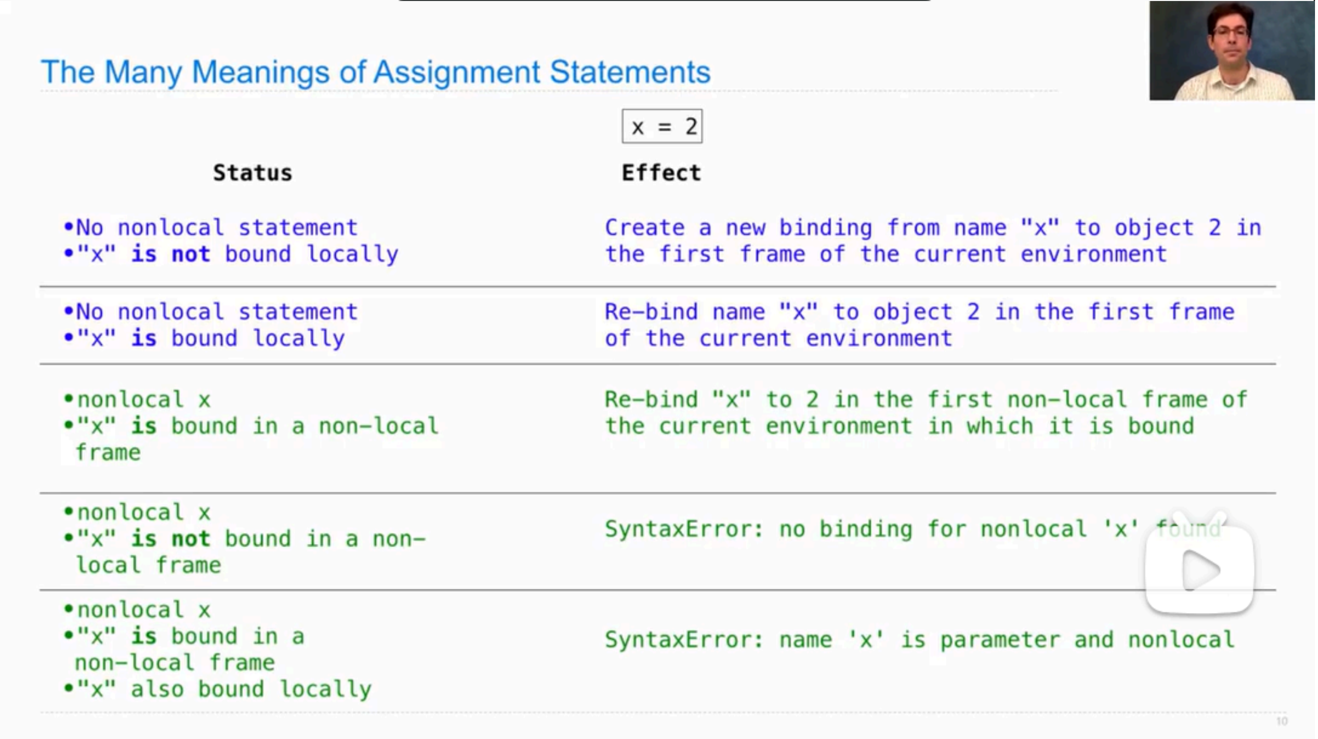
From the Python 3 language reference:

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.

Current frame

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

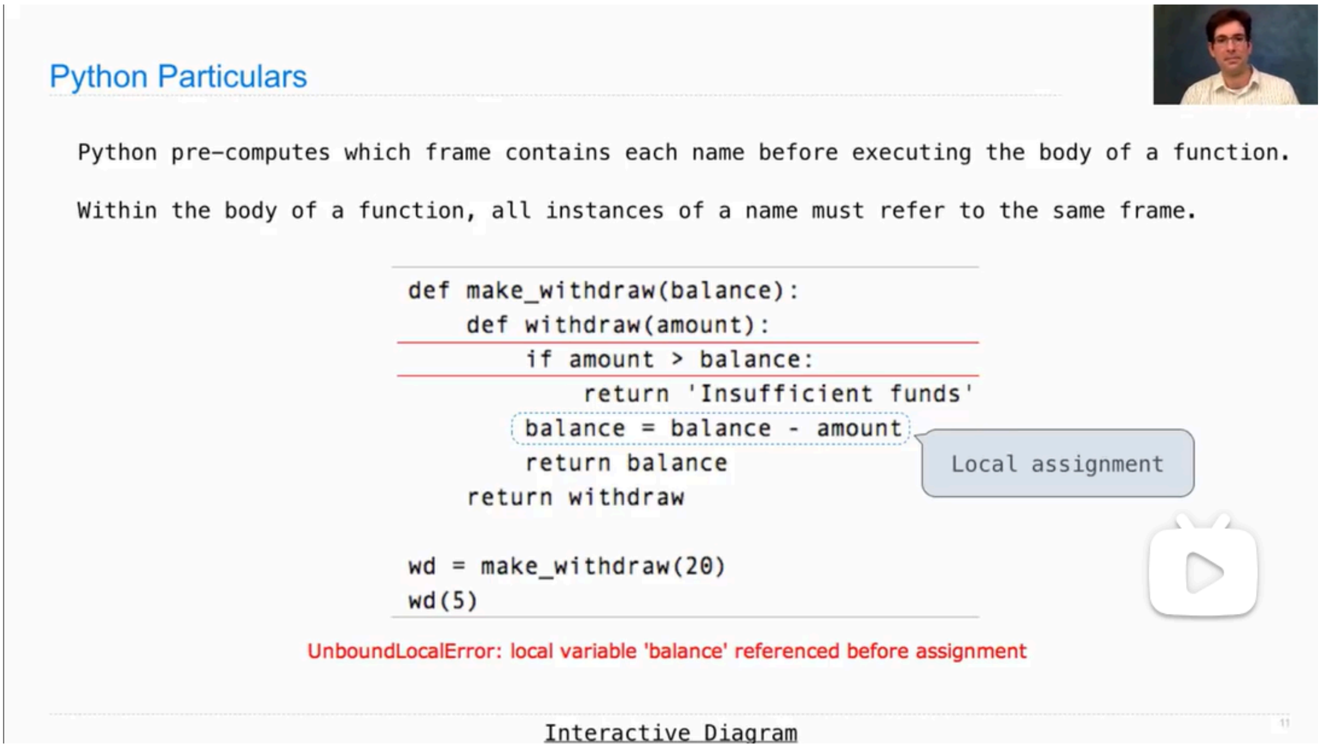


The Many Meanings of Assignment Statements

`x = 2`

Status	Effect
•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.

Within the body of a function, all instances of a name must refer to the same frame.

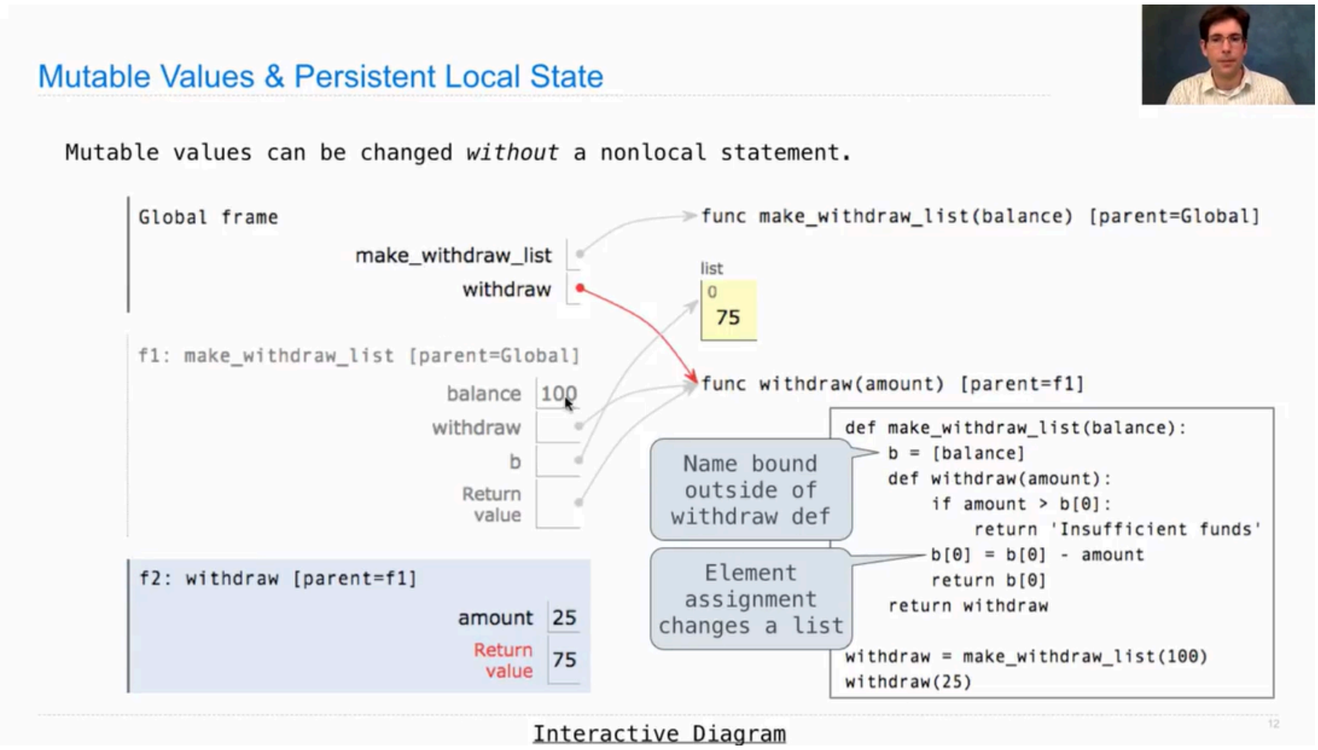
```
def make_withdraw(balance):
    def withdraw(amount):
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw

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

Local assignment

UnboundLocalError: local variable 'balance' referenced before assignment

Interactive Diagram

Mutable Values & Persistent Local State



Mutable Values & Persistent Local State

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

Global frame

make_withdraw_list

withdraw

func make_withdraw_list(balance) [parent=Global]

list

0

75

f1: make_withdraw_list [parent=Global]

balance 100

withdraw

b

Return value

func withdraw(amount) [parent=f1]

f2: withdraw [parent=f1]

amount 25

Return value 75

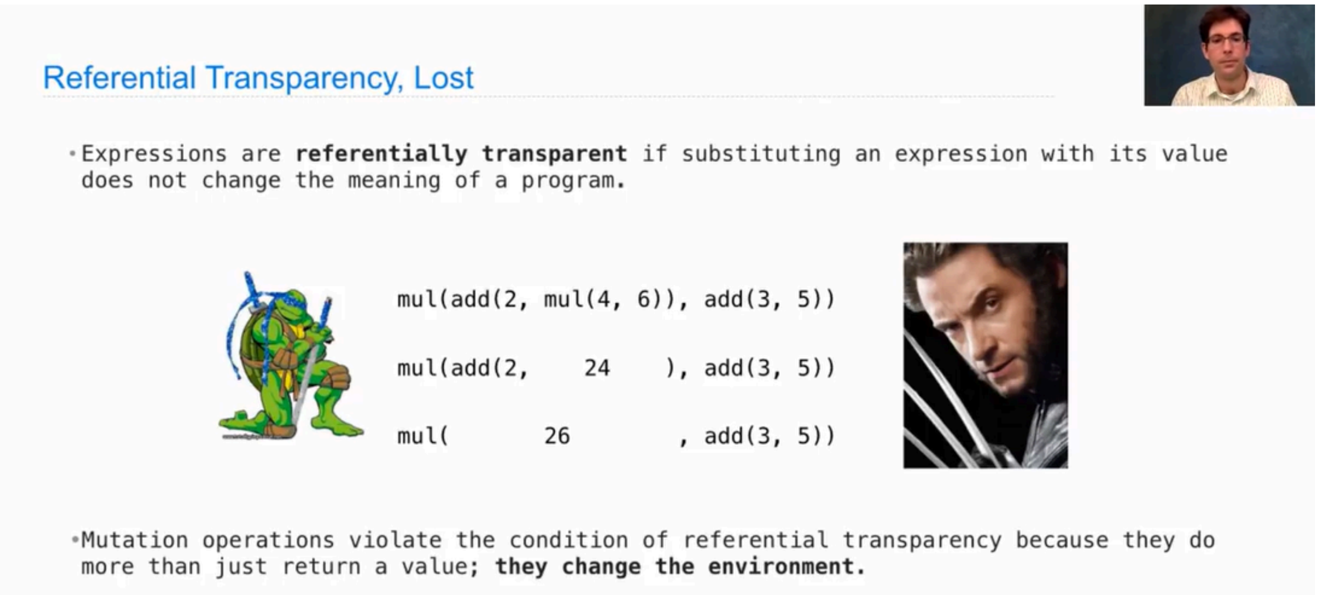
Name bound outside of withdraw def

Element assignment changes a list

```
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

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.

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

mul(add(2,    24    ), add(3, 5))

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.**