Mutable Functions

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'
>>> withdraw(15)
35
>>> withdraw = make_withdraw(100)
```

Return value: remaining balance

Argument: amount to withdraw

Second withdrawal of the same amount

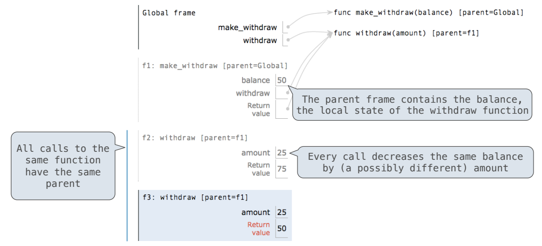
Different return value!

Where's this balance stored?

Within the parent frame of the function!

A function has a body and a parent environment

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

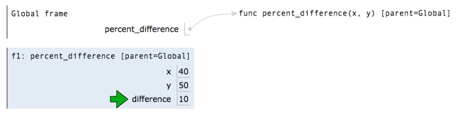
Every call decreases the same balance by (a possibly different) amount

pythontutor.com/composingprograms.html#code=def%20make_withdraw%28balance%29%3A%0A%20%20%20%20def%20withdraw%28amount%29%3A%0A%20%20%20%20%20%20%20%20nonlocal%20balance%0A%20%20%20%20%20%20%20%20if%20amount%20>%20balance%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20'Insufficient%20funds'%0A%20%20%20%20%20%20%20%20balance%20%3D%20balance%20-%20amount%0A%20%20%20%20%20%20%20%20return%20balance%0A%20%20%20%20return%20withdraw%0A%0Awithdraw%20%3D%20make_withdraw%28100%29%0Awithdraw%2825%29%0Awithdraw%2825%29%0Awithdraw%2860%29%0Awithdraw%2815%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0 5

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**

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'
        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

(Demo)

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

http://www.python.org/dev/peps/pep-3104/

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 "x" 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 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)
```

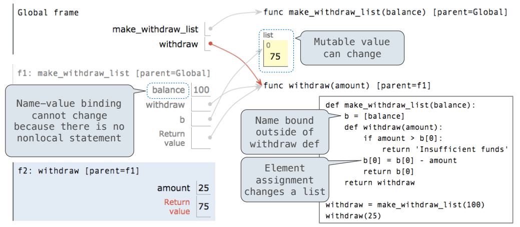
Local assignment

UnboundLocalError: local variable 'balance' referenced before assignment

pythontutor.com/
composingprograms.html#code=def%20make_withdraw%28balance%29%3A%0A%20%20%20%20def%20withdraw%28amount%29%3A%0A%20%20%20%20%20%20%20%20if%20amount%20>%20balance%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20'Insufficient%20funds'%0A%20%20%20%20%20%20%20%20balance%20%3D%20balance%20-%20amount%0A%20%20%20%20%20%20%20%20return%20balance%0A%20%20%20%20return%20withdraw%0A%0Awithdraw%20%3D%20make_withdraw%2820%29%0Awithdraw%285%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0 11

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
Mutable value can change
f1: make_withdraw_list [parent=Global]
balance 100
withdraw
b
Return value
func withdraw(amount) [parent=f1]
Name-value binding cannot change because there is no nonlocal statement
Name bound outside of withdraw def
Element assignment changes a list
f2: withdraw [parent=f1]
amount 25
Return value 75

```
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)
```

goo.gl/y4TyFZ

Multiple Mutable Functions

(Demo)

Referential Transparency, Lost

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

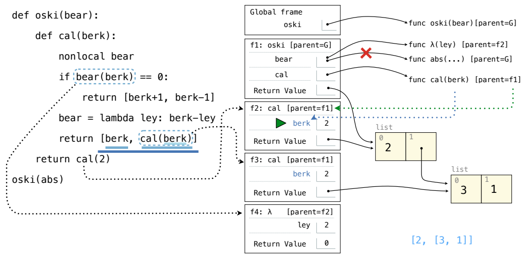
```
mul(add(2, mul(4, 6)), add(3, 5))

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.**

Environment Diagrams

Go Bears!

```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
    return cal(2)
oski(abs)
```



[2, [3, 1]]