

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)

```

Annotations:

- Return value: remaining balance (75)
- Different return value! (50)
- Argument: amount to withdraw (25)
- Second withdrawal of the same amount (25)
- Where's this balance stored? (35)
- Within the parent frame of the function! (make_withdraw)
- A function has a body and a parent environment (make_withdraw)

Persistent Local State Using Environments

```

Global frame
├── make_withdraw
│   ├── func make_withdraw(balance) [parent=Global]
│   └── withdraw
├── f1: make_withdraw [parent=Global]
│   ├── balance: 50
│   ├── withdraw
│   └── Return value
├── f2: withdraw [parent=f1]
│   ├── amount: 25
│   └── Return value: 75
└── f3: withdraw [parent=f1]
    ├── amount: 25
    └── Return value: 50

```

Annotations:

- The parent frame contains the balance, the local state of the withdraw function
- Every call decreases the same balance by (a possibly different) amount
- All calls to the same function have the same parent

Reminder: Local Assignment

```

def percent_difference(x, y):
    difference = abs(x - y)
    return 100 * difference / x
diff = percent_difference(48, 58)

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

```

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

Execution rule for assignment statements:

- Evaluate all expressions right of =, from left to right
- 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

```

Annotations:

- 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

https://docs.python.org/release/3.3.3/reference/simple_stmts.html#the-nonlocal-statement
<https://www.python.org/dev/peps/pep-3184/>

The Many Meanings of Assignment Statements

Status	Effect
<ul style="list-style-type: none"> No nonlocal statement "x" is not bound locally 	<div style="border: 1px solid black; padding: 2px; display: inline-block;">x = 2</div> Create a new binding from name "x" to object 2 in the first frame of the current environment
<ul style="list-style-type: none"> No nonlocal statement "x" is bound locally 	Re-bind name "x" to object 2 in the first frame of the current environment
<ul style="list-style-type: none"> 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
<ul style="list-style-type: none"> nonlocal x "x" is not bound in a non-local frame 	SyntaxError: no binding for nonlocal 'x' found
<ul style="list-style-type: none"> 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
```

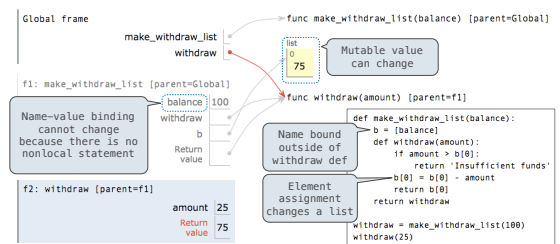
Local assignment

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

UnboundLocalError: local variable 'balance' referenced before assignment

Mutable Values & Persistent Local State

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



goo.gU/y4TyFZ

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**.

Multiple Mutable Functions

(Demo)

Environment Diagrams

Go Bears!

