Lecture 13: Mutable Functions

Brian Hou
July 12, 2016

Announcements

- Project 2 is due today (submit early and often)
- Look at your Hog submission for composition feedback
- Midterm is on 7/14 from 5-8 PM in 2050 VLSB
- TA-led review session during lecture tomorrow
- Office hours after 3 PM on Thursday and on Friday have been rescheduled
- More information on Piazza

Roadmap

Introduction
Functions
Data
Mutability
Objects
Interpretation
Paradigms
Applications

This short week (Mutability), the goals are:
- To explore the power of values that can mutate, or change

Mutable Functions

Functions That Change

How can we model a bank account that has a balance of $100?

```python
>>> withdraw = make_withdraw(100)
>>> withdraw(25)
75
>>> withdraw(25)
50
>>> withdraw(60)
'Insufficient funds'
>>> withdraw(15)
35
```

Where is this balance stored?

Persistent Local State in 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
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

Nonlocal Assignment

Nonlocal Statements

Assignment Statements

Python Particulars

Accounts
Mutable Sequences (demo)

```python
def make_withdraw(balance):
b = [balance]
def withdraw(amount):
    if amount > b[0]:
        return 'Insufficient funds'
    b[0] = b[0] - amount
    return b[0]
return withdraw
```

Multiple Mutable Functions (demo)

```python
>>> brian = make_withdraw(100)
>>> marvin = make_withdraw(100000)
>>> brian(10)
90
>>> marvin(10000)
90000
>>> brian(100)
'Insufficient funds'
>>> marvin(100)
89900
```

Referential Transparency

- 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*

Summary

- The nonlocal statement allows us to *mutate name-value bindings in a nonlocal frame*
- Mutation is a powerful tool, but it also makes reasoning about programs more difficult
- The truth is: we don’t usually use nonlocal to build our own objects with mutable state
- We’ll see another way next week
- Good luck on the midterm!