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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?

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

![Diagram]

Argument: amount to withdraw
Return value: remaining balance

Different return value!
Where's this balance stored?
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

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"

The Many Meanings of Assignment Statements

Status  Effect
*No nonlocal statement  *x* is not bound locally
*No nonlocal statement  *x* is bound locally

*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
SystemError: no binding for nonlocal 'x' found

*Nonlocal x  *x* is bound in a non-local frame
*x* also bound locally
SystemError: 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
    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.

```
label frame
make_withdraw_list
withdraw
make_withdraw_list()
withdraw

Name-value binding cannot change because there is no nonlocal statement
Name bound outside of withdraw def
Element assignment changes a list

Local assignment
Value can change

Withdraw def
```

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.