Announcements

- HW6 due on Thursday
- Trends project due tomorrow
- Ants project out
Persistent Local State

A function with a parent frame

The parent contains local state

Every call changes the balance

Example: http://goo.gl/5LZ6F
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

Re-bind balance where it was bound previously
Mutable Values and Persistent State
Mutable Values and Persistent State

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

Example: [http://goo.gl/cEpmz](http://goo.gl/cEpmz)
Mutable values can be changed without a nonlocal statement.

Example: [http://goo.gl/cEpmz](http://goo.gl/cEpmz)
Mutable values can be changed without a nonlocal statement.

Example: [http://goo.gl/cEpmz](http://goo.gl/cEpmz)
Creating Two Withdraw Functions

Example: http://goo.gl/glTyB
Multiple References to a Withdraw Function

Example: http://goo.gl/X2qG9
The Benefits of Non-Local Assignment
The Benefits of Non-Local Assignment

- Ability to maintain some state that is local to a function, but evolves over successive calls to that function.
The Benefits of Non-Local Assignment

- Ability to maintain some state that is local to a function, but evolves over successive calls to that function.

- The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is inaccessible to the rest of the program.
The Benefits of Non-Local Assignment

- Ability to maintain some state that is local to a function, but evolves over successive calls to that function.

- The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is inaccessible to the rest of the program.

- An abstraction of a bank account that manages its own internal state.
The Benefits of Non-Local Assignment

- Ability to maintain some state that is local to a function, but evolves over successive calls to that function.

- The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is inaccessible to the rest of the program.

- An abstraction of a bank account that manages its own internal state.

<table>
<thead>
<tr>
<th>Weasley Account</th>
<th>$10</th>
</tr>
</thead>
</table>
The Benefits of Non-Local Assignment

- Ability to maintain some state that is local to a function, but evolves over successive calls to that function.

- The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is inaccessible to the rest of the program.

- An abstraction of a bank account that manages its own internal state.

<table>
<thead>
<tr>
<th></th>
<th>Weasley Account</th>
<th>Potter Account</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$10</td>
<td>$1,000,000</td>
</tr>
</tbody>
</table>
Referential Transparency
Referential Transparency

Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.
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)), 3)
```
Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

\[
\text{mul}(\text{add}(2, \text{mul}(4, 6)), 3) \\
\text{mul}(\text{add}(2, 24), 3)
\]
Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

\[
\text{mul}(\text{add}(2, \text{mul}(4, 6)), 3)
\]

\[
\text{mul}(\text{add}(2, 24), 3)
\]

\[
\text{mul}(26, 3)
\]
Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

\[
\text{mul}(\text{add}(2, \text{mul}(4, 6)), 3) \\
\text{mul}(\text{add}(2, 24), 3) \\
\text{mul}(26, 3)
\]

Mutation is a \textit{side effect} (like printing)
Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

\[ \text{mul}(\text{add}(2, \text{mul}(4, 6)), 3) \]
\[ \text{mul}(\text{add}(2, 24), 3) \]
\[ \text{mul}(26, 3) \]

Mutation is a *side effect* (like printing)

Side effects violate the condition of referential transparency because they do more than just return a value; they change the state of the computer.
Referential Transparency

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

\[
\text{mul}(\text{add}(2, \text{mul}(4, 6)), 3)
\]

\[
\text{mul}(\text{add}(2, 24), 3)
\]

\[
\text{mul}(26, 3)
\]

Mutation is a *side effect* (like printing)

Side effects violate the condition of referential transparency because they do more than just return a value; they change the state of the computer.
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)), 3)
mul(add(2, 24), 3)
mul(26, 3)
```

Mutation is a *side effect* (like printing)

Side effects violate the condition of referential transparency because they do more than just return a value; they change the state of the computer.
A Mutable Container
A Mutable Container

def container(contents):

A Mutable Container

def container(contents):
    """Return a container that is manipulated by two functions."""
A Mutable Container

def container(contents):
    """Return a container that is manipulated by two functions.
    >>> get, put = container('hello')"""
def container(contents):
    """Return a container that is manipulated by two functions."
    >>> get, put = container('hello')
    >>> get()
A Mutable Container

def container(contents):
    """Return a container that is manipulated by two functions.
    >>> get, put = container('hello')
    >>> get()
    'hello'
A Mutable Container

def container(contents):
    """Return a container that is manipulated by two functions.
    >>> get, put = container('hello')
    >>> get()
    'hello'
    >>> put('world')
def container(contents):
    """Return a container that is manipulated by two functions.
    >>> get, put = container('hello')
    >>> get()
    'hello'
    >>> put('world')
    >>> get()
def container(contents):
    """Return a container that is manipulated by two functions."
    >>> get, put = container('hello')
    >>> get()
    'hello'
    >>> put('world')
    >>> get()
    'world'
def container(contents):
    """Return a container that is manipulated by two functions.
    >>> get, put = container('hello')
    >>> get()
    'hello'
    >>> put('world')
    >>> get()
    'world'
    """
def container(contents):
    """Return a container that is manipulated by two functions.
    >>> get, put = container('hello')
    >>> get()
    'hello'
    >>> put('world')
    >>> get()
    'world'
    """

def get():

A Mutable Container

def container(contents):
    """Return a container that is manipulated by two functions.
    >>> get, put = container('hello')
    >>> get()
    'hello'
    >>> put('world')
    >>> get()
    'world'
    """

def get():
    return contents
```python
def container(contents):
    """Return a container that is manipulated by two functions.

    >>> get, put = container('hello')
    >>> get()
    'hello'
    >>> put('world')
    >>> get()
    'world'
    """

def get():
    return contents

def put(value):
```
def container(contents):
    """Return a container that is manipulated by two functions.
    >>> get, put = container('hello')
    >>> get()
    'hello'
    >>> put('world')
    >>> get()
    'world'
    """
    def get():
        return contents
    def put(value):
        nonlocal contents
def container(contents):
    """Return a container that is manipulated by two functions.
    >>> get, put = container('hello')
    >>> get()
    'hello'
    >>> put('world')
    >>> get()
    'world'
    """
    def get():
        return contents
    def put(value):
        nonlocal contents
        contents = value
A Mutable Container

def container(contents):
    """Return a container that is manipulated by two functions.
    >>> get, put = container('hello')
    >>> get()
    'hello'
    >>> put('world')
    >>> get()
    'world'
    """
    def get():
        return contents
    def put(value):
        nonlocal contents
        contents = value
    return put, get
Dispatch Functions
Dispatch Functions

A technique for packing multiple behaviors into one function
A technique for packing multiple behaviors into one function

```python
def pair(x, y):
    """Return a function that behaves like a pair."""
    def dispatch(m):
        if m == 0:
            return x
        elif m == 1:
            return y
    return dispatch
```
A technique for packing multiple behaviors into one function

```python
def pair(x, y):
    """Return a function that behaves like a pair."""
    def dispatch(m):
        if m == 0:
            return x
        elif m == 1:
            return y
    return dispatch
```

Message argument can be anything, but strings are most common
Dispatch Functions

A technique for packing multiple behaviors into one function

```python
def pair(x, y):
    """Return a function that behaves like a pair."""
    def dispatch(m):
        if m == 0:
            return x
        elif m == 1:
            return y
    return dispatch
```

Message argument can be anything, but strings are most common

The body of a dispatch function is always the same:

```
st common
```
A technique for packing multiple behaviors into one function

```python
def pair(x, y):
    """Return a function that behaves like a pair.""
    def dispatch(m):
        if m == 0:
            return x
        elif m == 1:
            return y
    return dispatch
```

Message argument can be anything, but strings are most common

The body of a dispatch function is always the same:

• One conditional statement with several clauses
Dispatch Functions

A technique for packing multiple behaviors into one function

```python
def pair(x, y):
    """Return a function that behaves like a pair.""
    def dispatch(m):
        if m == 0:
            return x
        elif m == 1:
            return y
    return dispatch
```

Message argument can be anything, but strings are most common

The body of a dispatch function is always the same:

• One conditional statement with several clauses
• Headers perform equality tests on the message
Message Passing
Message Passing

An approach to organizing the relationship among different pieces of a program
Message Passing

An approach to organizing the relationship among different pieces of a program

Different objects pass messages to each other
Message Passing

An approach to organizing the relationship among different pieces of a program

Different objects pass messages to each other
  • What is your fourth element?
Message Passing

An approach to organizing the relationship among different pieces of a program

Different objects pass messages to each other
  • What is your fourth element?
  • Change your third element to this new value. (please?)
Message Passing

An approach to organizing the relationship among different pieces of a program

Different objects pass messages to each other
  • What is your fourth element?
  • Change your third element to this new value. (please?)

Encapsulates the behavior of all operations on a piece of data
Message Passing

An approach to organizing the relationship among different pieces of a program

Different objects pass messages to each other

• What is your fourth element?
• Change your third element to this new value. (please?)

Encapsulates the behavior of all operations on a piece of data

Important historical role:
The message passing approach strongly influenced object-oriented programming (next lecture)
def container(contents):

def get():
    return contents

def put(value):
    nonlocal contents
    contents = value
    return put, get
Mutable Container with Message Passing

```python
def container_dispatch(contents):
    def container(contents):
        def get():
            return contents
        def put(value):
            nonlocal contents
            contents = value
            return put, get
        return get
```
Mutable Container with Message Passing

```python
def container_dispatch(contents):
    def dispatch(message, value=None):
        def get():
            return contents
        def put(value):
            nonlocal contents
            contents = value
            return put, get
        return dispatch

def container(contents):
    return dispatch
```

Cal
Mutable Container with Message Passing

```python
def containerDispatch(contents):
    def dispatch(message, value=None):
        nonlocal contents
        ...

    def get():
        return contents

    def put(value):
        nonlocal contents
        contents = value
        return put, get
```

```python
def container(contents):
    def dispatch(message, value=None):
        nonlocal contents
        ...

    def get():
        return contents

    def put(value):
        nonlocal contents
        contents = value
        return put, get
```
Mutable Container with Message Passing

```python
def container(contents):
    def get():
        return contents
    def put(value):
        nonlocal contents
        contents = value
        return put, get
    return get, put

def container_dispatch(contents):
    def dispatch(message, value=None):
        nonlocal contents
        if message == 'get':
            return contents

    return dispatch
```

```
cal
```
Mutable Container with Message Passing

```python
def container_dispatch(contents):
    def dispatch(message, value=None):
        nonlocal contents

        if message == 'get':
            return contents

    def get():
        return contents

    def put(value):
        nonlocal contents
        contents = value

    return put, get

def container(contents):
    def get():
        return contents

    def put(value):
        nonlocal contents
        contents = value

    return put, get
```
def container_dispatch(contents):
    def dispatch(message, value=None):
        nonlocal contents
        if message == 'get':
            return contents
        if message == 'put':
            nonlocal contents
            contents = value
            return put, get
        return False
    return dispatch
```
def container_dispatch(contents):
    def dispatch(message, value=None):
        nonlocal contents
        if message == 'get':
            return contents
        if message == 'put':
            contents = value
        return put, get

    def get():
        return contents

    def put(value):
        nonlocal contents
        contents = value
        return put, get
```
Mutable Container with Message Passing

```python
def container_dispatch(contents):
    def dispatch(message, value=None):
        nonlocal contents
        if message == 'get':
            return contents
        if message == 'put':
            contents = value
        return dispatch

def container(contents):
    def get():
        return contents
    def put(value):
        nonlocal contents
        contents = value
        return put, get
```

```python
def container_dispatch(contents):
    def dispatch(message, value=None):
        nonlocal contents
        if message == 'get':
            return contents
        if message == 'put':
            contents = value
        return dispatch

def container(contents):
    def get():
        return contents
    def put(value):
        nonlocal contents
        contents = value
        return put, get
```
Mutable Container with Message Passing

```python
def container_dispatch(contents):
    def dispatch(message, value=None):
        nonlocal contents

        if message == 'get':
            return contents

        if message == 'put':
            contents = value

        return dispatch

    def get():
        return contents

    def put(value):
        nonlocal contents
        contents = value
        return put, get

    return dispatch

def container(contents):
    def get():
        return contents

    def put(value):
        nonlocal contents
        contents = value
        return put, get

    return get, put
```

Cal
def container_dispatch(contents):
    def dispatch(message, value=None):
        nonlocal contents
        if message == 'get':
            return contents
        if message == 'put':
            contents = value
        return dispatch
    return dispatch

def container(contents):
    def get():
        return contents
    def put(value):
        nonlocal contents
        contents = value
        return put, get
    return get, put
Mutable Recursive Lists
def mutable_rlist():
def mutable_rlist():
    contents = empty_rlist
def mutable_rlist():
    contents = empty_rlist

def dispatch(message, value=None):
Mutable Recursive Lists

```python
def mutable_rlist():
    contents = empty_rlist
def dispatch(message, value=None):
    nonlocal contents
```
def mutable_rlist():
    contents = empty_rlist

def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
Mutable Recursive Lists

```python
def mutable_rlist():
  contents = empty_rlist

def dispatch(message, value=None):
  nonlocal contents
  if message == 'len':
    return len_rlist(contents)
```
def mutable_rlist():
    contents = empty_rlist
def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
        return len_rlist(contents)
elif message == 'getitem':
def mutable_rlist():
    contents = empty_rlist

def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
        return len_rlist(contents)
    elif message == 'getitem':
        return getitem_rlist(contents, value)
Mutable Recursive Lists

def mutable_rlist():
    contents = empty_rlist
def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
        return len_rlist(contents)
    elif message == 'getitem':
        return getitem_rlist(contents, value)
    elif message == 'push':
Mutable Recursive Lists

def mutable_rlist():
    contents = empty_rlist

def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
        return len_rlist(contents)
    elif message == 'getitem':
        return getitem_rlist(contents, value)
    elif message == 'push':
        contents = make_rlist(value, contents)
def mutable_rlist():
    contents = empty_rlist

def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
        return len_rlist(contents)
    elif message == 'getitem':
        return getitem_rlist(contents, value)
    elif message == 'push':
        contents = make_rlist(value, contents)
    elif message == 'pop':
def mutable_rlist():
    contents = empty_rlist

def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
        return len_rlist(contents)
    elif message == 'getitem':
        return getitem_rlist(contents, value)
    elif message == 'push':
        contents = make_rlist(value, contents)
    elif message == 'pop':
        item = first(contents)
def mutable_rlist():
    contents = empty_rlist

def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
        return len_rlist(contents)
    elif message == 'getitem':
        return getitem_rlist(contents, value)
    elif message == 'push':
        contents = make_rlist(value, contents)
    elif message == 'pop':
        item = first(contents)
        contents = rest(contents)
def mutable_rlist():
    contents = empty_rlist

def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
        return len_rlist(contents)
    elif message == 'getitem':
        return getitem_rlist(contents, value)
    elif message == 'push':
        contents = make_rlist(value, contents)
    elif message == 'pop':
        item = first(contents)
        contents = rest(contents)
        return item
Mutable Recursive Lists

def mutable_rlist():
    contents = empty_rlist

def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
        return len_rlist(contents)
    elif message == 'getitem':
        return getitem_rlist(contents, value)
    elif message == 'push':
        contents = make_rlist(value, contents)
    elif message == 'pop':
        item = first(contents)
        contents = rest(contents)
        return item
    elif message == 'str':
def mutable_rlist():
    contents = empty_rlist

def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
        return len_rlist(contents)
    elif message == 'getitem':
        return getitem_rlist(contents, value)
    elif message == 'push':
        contents = make_rlist(value, contents)
    elif message == 'pop':
        item = first(contents)
        contents = rest(contents)
        return item
    elif message == 'str':
        return str_rlist(contents)
Mutable Recursive Lists

def mutable_rlist():
    contents = empty_rlist
def dispatch(message, value=None):
    nonlocal contents
    if message == 'len':
        return len_rlist(contents)
elif message == 'getitem':
        return getitem_rlist(contents, value)
elif message == 'push':
    contents = make_rlist(value, contents)
elif message == 'pop':
    item = first(contents)
    contents = rest(contents)
    return item
    return dispatch