

Containers

Announcements

Pairs

(Demo)

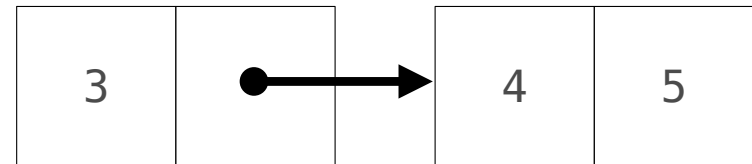
Pairs Implemented as Functions

```
def pair(x, y):  
    def dispatch(m):  
        if m == 'first':  
            return x  
        elif m == 'second':  
            return y  
    return dispatch
```

This
function
represents
the pair
(x, y)

Constructor is a higher-
order function

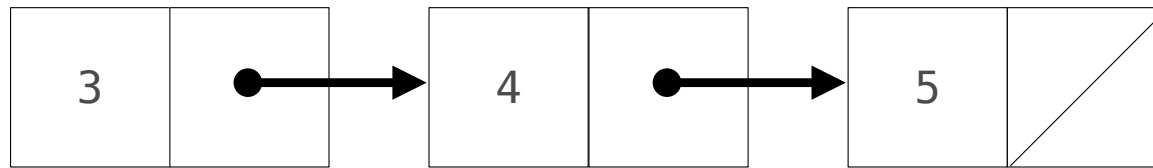
```
>>> p = pair(3, pair(4, 5))  
>>> p('first')  
3  
>>> p('second')('first')  
4  
>>> p('second')('second')  
5
```



(Demo)

Linked Lists (Sneak Preview)

- An empty list is called "nil" and represented as None
- A non-empty list is represented as a pair
 - The first element of the pair is the first element of the list
 - The second element of the pair is the rest of the list



```
nil = None
def list_len(s):
    if s == nil:
        return 0
    else:
        return 1 + list_len(s('second'))
```

(Demo)

Mutable Functions

A Function with Behavior That Varies Over Time

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

Return value:
remaining balance

Different
return value!

```
>>> withdraw = make_withdraw(100)
```

Within the parent frame
of the function!

```
>>> withdraw(25)  
75
```

Argument:
amount to withdraw

```
>>> withdraw(25)  
50
```

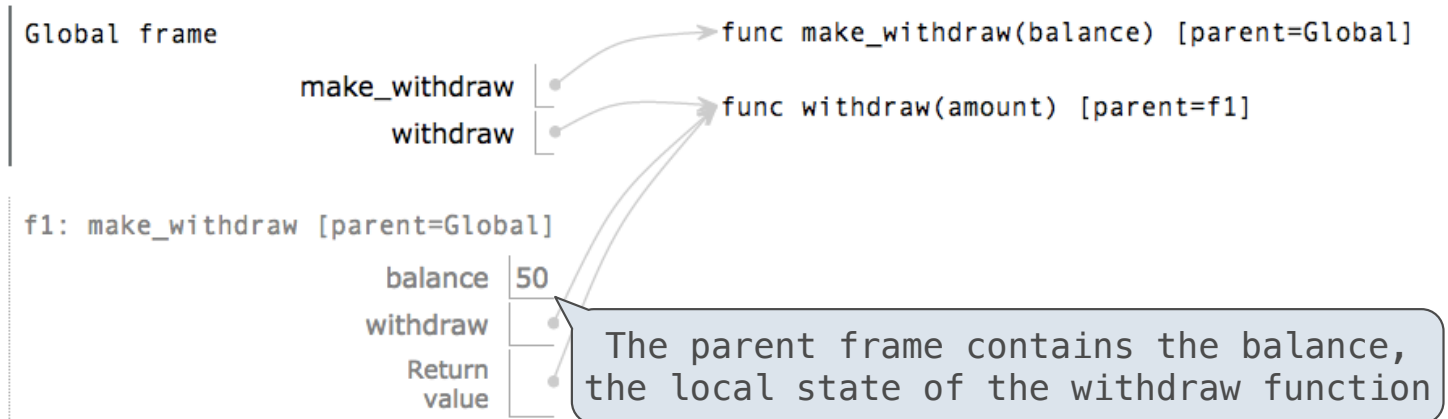
Second withdrawal of
the same amount

```
>>> withdraw(60)  
'Insufficient funds'
```

```
>>> withdraw(15)  
35
```

Where's this balance
stored?

Persistent Local State Using Environments



All calls to the
same function
have the same
parent

```
f2: withdraw [parent=f1]
```

amount	25
Return value	75

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

```
f3: withdraw [parent=f1]
```

amount	25
Return value	50

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
```

Declare the name "balance" nonlocal at the top of the body of the function in which it is re-assigned

```
        if amount > balance:
```

```
            return 'Insufficient funds'
```

```
        balance = balance - amount
```

Re-bind balance in the first non-local frame in which it was bound previously

```
        return balance
```

```
    return withdraw
```

(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

<code>x = 2</code>

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
```

Local assignment

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

UnboundLocalError: local variable 'balance' referenced before assignment

Mutable Data

Functions with Shared Local State

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

```
get, put = box('Hello')  
before = get()  
put('Goodbye')  
after = get()
```

