61A Lecture 3

Wednesday, August 29

Life Cycle of a User-Defined Function

**Def statement:**
- **Name:** square(x)
- **Body:** return mul(x, x)

**Call expression:**
- **Operator:** square(2+2)
- **Argument:** 4

**Calling/Applying:**
- square(x)

- **Op's evaluated**
- Function called with argument(s)

- **New frame!**
- Params bound

- **Function created**
- Name bound

- **What happens?**

Example:
- http://goo.gl/668fU

Multiple Environments in One Diagram!

(Demo)

Names Have No Meaning Without Environments

Every expression is evaluated in the context of an environment.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

```
mul(x, x)  
```

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

Formal Parameters

```
def square(x):
    return mul(x, x)  
```

```
def square(y):
    return mul(y, y)
```

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

An environment is a sequence of frames.

- The global frame alone
- A local, then the global frame

Examples:
- http://goj.gl/Ap8F
Python Feature Demonstration

Operators
Multiple Return Values
Docstrings
Doctests
Default Arguments
Statements

Compound Statements

Compound statements:

```
<header>:
  <statement>
  <statement>
  ...
<separating header>:
  <statement>
  <statement>
  ...
```

A suite is a sequence of statements
To “execute” a suite means to execute its sequence of statements, in order

Execution Rule for a sequence of statements:

- Execute the first
- Unless directed otherwise, execute the rest

Conditional Statements

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
```

Execution rule for conditional statements:
Each clause is considered in order.
1. Evaluate the header’s expression.
2. If it is a true value, execute the suite & skip the remaining clauses.

Local Assignment

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
```

Execution rule for assignment statements:
1. Evaluate all expressions right of =, from left to right.
2. Bind the names on the left the resulting values in the first frame of the current environment.

Example: [link](http://goo.gl/wcF71)

Boolean Contexts

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
```

George Boole
**Boolean Contexts**

```python
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
```

False values in Python:  
- `False`   
- `0`   
- `''`   
- `None`  
(more to come)

True values in Python:  
- Anything else (True)

Read Section 1.5.4!

**Iteration**

Execution rule for while statements:

1. Evaluate the header’s expression.
2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/O7y0D](http://goo.gl/O7y0D)