Announcements

- Reminder: hw1 due tonight

- In-class quiz on Friday
  - Covers through Wednesday’s lecture
  - Bring a writing implement

- Hog project out
  - Get started early!
  - Try it out online! See the announcement on the website
The Game of Hog

Expected Score vs. Number of Dice Rolled (6-Sided Dice)

Chance of 10+ vs. Number of Dice Rolled (6-Sided Dice)

- Expected Scores:
  - 1 dice: 3.7
  - 2 dice: 6.0
  - 3 dice: 7.5
  - 4 dice: 8.4
  - 5 dice: 8.9
  - 6 dice: 8.9
  - 7 dice: 8.7
  - 8 dice: 8.5
  - 9 dice: 8.0
  - 10 dice: 7.5

- Chance of 10+:
  - 1 dice: 17%
  - 2 dice: 49%
  - 3 dice: 48%
  - 4 dice: 40%
  - 5 dice: 33%
  - 6 dice: 28%
  - 7 dice: 23%
  - 8 dice: 19%
  - 9 dice: 16%
Every expression is evaluated in the context of an environment.

So far, the current environment is either:
- The global frame alone, or
- A local frame, followed by the global frame

Important properties of environments:
- An environment is a sequence of frames.
- The earliest frame that contains a binding for a name determines the value that the name evaluates to.

The scope of a name is the region of code that has access to it.
The environment in which a function is applied consists of:

- A new local frame each time the function is applied
- The environment in which the function was defined
  - We refer to this as *lexical scoping*
  - So far, this is just the global frame
  - The *current* state of the environment is used, not the state when the function definition was executed
Formal Parameters

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

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

Formal parameters have local scope

Example: http://goo.gl/boCk0
Multiple Environments in a Diagram

What happens when to the local frame when a function returns?

- It sticks around until Python realizes it is no longer needed
- We will soon see cases where it is needed after the call

Example:
```
from operator import mul

def square(x):
    return mul(x, x)
square(square(3))
```

Example: [http://goo.gl/hrfnV](http://goo.gl/hrfnV)
Life Cycle of a User-Defined Function

Def statement:

Formal parameter

Name

square( x ):

Return expression

return mul(x, x)

Body (return statement)

Def statement

Calling/Applying:

operator: square
function: func square(x)

Operand: 2+2
Argument: 4

Op's evaluated
Function called with argument(s)
Evaluates to return value below

New frame!
Params bound
Body executed

What happens?
Function created
Name bound

Argument
Signature

4

4

16

Return value
Python Feature Demonstration

Operators

Multiple Return Values

Docstrings

Doctests

Default Arguments

Statements
A *statement* is executed by the interpreter to perform an action

Types of statements we have seen so far

- An assignment
  ```python
  radius = 10
  ```

- A function definition
  ```python
  def square(x):
      return x * x
  ```

- Returns, imports, assertions
Local Assignment

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: http://goo.gl/1pyzL
A function definition is a *compound statement*

**Compound statements:**

- `<header>`: 
  - `<statement>`
  - `<statement>`
  - ...
  - `<suite>`

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

The first header determines a statement’s type.

The header of a clause “controls” the suite that follows.
Compound Statements

Compound statements:

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:

1. Execute the first

2. Unless directed otherwise, execute the rest
Conditional Statements

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

1 statement,
3 clauses,
3 headers,
3 suites

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

Boolean Contexts

Two boolean contexts

George Boole

False values in Python: False, 0, '', None

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/mk7Sc](http://goo.gl/mk7Sc)
Locally Defined Functions

Functions can be defined inside other functions

What happens when a def is executed?
1. Create a function value with the given signature and body
2. Bind the given name to that value in the current frame

The name can then be used to call the function.

def sum_of_squares(n):
    """Sum of the squares of the integers 1 to n""
    def square(x):
        return mul(x, x)
    total, k = 0, 1
    while k <= n:
        total, k = total + square(k), k + 1
    return total
Locally Defined Functions

The inner definition is executed each time the outer function is called

```python
from operator import mul

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

    square_inside()

square_inside()
```

Example: [http://goo.gl/pnU8f](http://goo.gl/pnU8f)
Locally defined functions can be returned
They have access to the frame in which they are defined

A function that returns a function

```python
def make_adder(n):
    """Return a function that adds n to its argument."
    def adder(k):
        return add(n, k)
    return adder
```

The name add_three is bound to a function

```python
>>> add_three = make_adder(3)
>>> add_three(4)
7
"""
```

A local def statement

Can refer to names in the enclosing function
Call Expressions as Operators

```
def make_adder(n):
    def adder(k):
        return add(n, k)
    return adder

make_adder(1)(2)
```