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

Environment Diagrams

- 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

Environment of Function Application

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

- `def square(x): return mul(x, x)` vs `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: http://goo.gl/hrfnV

Life Cycle of a User-Defined Function

Def statement:
- Formal parameter:
- Name
- Return expression

Def statement:
- Name
- Body (return statement)

Call expression:
- square(2+2)
- operand: 2+2 argument: 4

Calling/Applying:
- operator: square
- function: func

A statement is executed by the interpreter to perform an action

Types of statements we have seen so far

- An assignment
  \[ \text{radius} = 10 \]
- A function definition
  \[
  \text{def square}(x):
  \quad \text{return } x \times x
  \]
- 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/hrfnV

Python Feature Demonstration

Operators
- Multiple Return Values
Docstrings
Doctests
Default Arguments
Statements

Statements

A statement is executed by the interpreter to perform an action

Types of statements we have seen so far

- An assignment
  \[ \text{radius} = 10 \]
- A function definition
  \[
  \text{def square}(x):
  \quad \text{return } x \times x
  \]
- Returns, imports, assertions

Compound Statements

A function definition is a compound statement

Compound statements:
- The first header determines a statement’s type
- The header of a clause “controls” the suite that follows

Example: http://goo.gl/hrfnV
### 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:
1. Execute the first
2. Unless directed otherwise, execute the rest

### Conditional Statements

**def absolute_value(x):**

"Return the absolute value of x."

```python
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.

### Boolean Contexts

- False values in Python: False, 0, "", None
- True values in Python: Anything else (True)

Read Section 1.5.4!

### Iteration

**def absolute_value(x):**

"Return the absolute value of x."

```python
if x > 0:
    return x
elif x == 0:
    return 0
else:
    return -x
```

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://example.com](http://example.com)

### 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.

#### Locally Defined Functions

**def sum_of_squares(n):**

"""Sum of the squares of the integers 1 to n""

```python
def square(x):
    return mul(x, x)
def sum_of_squares(n):
    total, k = 0, 1
    while k <= n:
        total, k = total + square(k), k + 1
    return total
```

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

Example: [http://example.com](http://example.com)
Functions as Return Values

Locally defined functions can be returned
They have access to the frame in which they are defined

A function that returns a function

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

An expression that evaluates to a function value

Call Expressions as Operators

make_adder(1)(2)

make_adder(1) ( 2  )
Operator      Operand 0

An expression that evaluates to any value

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