Announcements
- hw1 is due tomorrow at 11:59PM
- Have to submit through your account
- Your TA will go over homework submission in lab
- Project1 is out! Find a partner if you haven’t already.
  - Will have all the tools you need to complete the project by the end of lecture today

Looking up names
Procedure for looking up a name from inside a function (v. 1):
1. Look it up in the local frame
2. If not in local frame, look it up in the global frame
3. If in neither frame, generate error

Let’s recap...

Life cycle of a user-defined function

Multiple environments in one diagram

Every expression is evaluated in the context of an environment.
**Python Feature Demonstration**

- Multiple Assignment
- Multiple Return Values
- Docstrings
- Doctests
- Default Arguments

---

**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!

---

**Keywords: “and” “or”**

- The keywords "and" and "or" are useful for combining values in a boolean context
- and returns a true value if all expressions are true in a boolean context
- (5 > 3) and (1 + 1 == 2) will return True
- or returns a true value if any expression is true in a boolean context
- (1 > 5) or (400 < 10) or (2 == 4 – 2) will return True
- But it's not quite that simple...

---

**“Short-circuiting”**

- The keyword “and” will return the first expression that is False in a boolean context
- If there are no expressions that are False, then the last value in the statement is returned
- The keyword “or” will return the first expression that is True in a boolean context
- If there are no expressions that are True, then the last value in the statement is returned

```python
>>> True and 5
5
>>> True or (5 / 0)
True
```

---

**Interpreter session**

---

**Break**
**Statements**

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

Types of statements we have seen so far:
- An assignment: `radius = 10`
- A function definition:
  ```python
def square(x):
    return x * x
  ```
- Returns, imports, assertions

**Compound Statements**

A function definition is a **compound statement**.

**Compound statements**:

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

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

**Conditional Statements**

Execution rule for conditional statements:

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
```

**Local Assignment**

```
def percent_difference(x, y):
    difference = abs(y - x)
    return 100 * difference / x
```

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 first frame of the current environment.

**Iteration**

```
1. total = 0, 0
   while 1 < 1:
     i = i + 1
     total = total + i
```

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: [link](http://goo.gl/mk7Sc)
Break

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
```

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

Locally Defined Functions
Functions are first-class: they can be manipulated as values in Python
Higher-order function: a function that takes a function as an argument value or returns a function as a return value

Higher-order functions:
- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions

The Art of the Function
- Give each function exactly one job
- Don’t repeat yourself (DRY).
- Define functions generally
- Proj1 has a composition score! Adhere to these guidelines

Generalizing Patterns with Parameters
Regular geometric shapes relate length and area.

Shape:  
```
\begin{align*}
\text{Circle:} & \quad r^2 \\
\text{Square:} & \quad r^2 \\
\text{Hexagon:} & \quad \frac{3\sqrt{3}}{2} r^2
\end{align*}
```

Area:  
Finding common structure allows for shared implementation
Interpreter session

Generalizing Over Computational Processes

The common structure among functions may itself be a computational process, rather than a number.

\[
\begin{align*}
\sum_{k=1}^{5} k &= 1 + 2 + 3 + 4 + 5 = 15 \\
\sum_{k=1}^{5} k^2 &= 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 55 \\
\sum_{k=-1}^{5} (k + 3)(k + 1) &= 8 + \frac{8}{3} + \frac{8}{9} + \frac{8}{15} + \frac{8}{33} = 3.04
\end{align*}
\]

Interpreter session

Functions as Arguments

Function values can be passed as arguments

```python
def cube(k):
    return pow(k, 3)
def summation(n, term):
    # Sum the first n terms of a sequence.
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

>>> summation(5, cube)
225
```

Function Values as Parameters

Parameters can be bound to function values

```python
... summation(5, cube)
225
```

That’s it for today

- This is all I wanted to get through for today, but if we have time left, we can go to the next slides
Functions as Return Values
Locally defined functions can be returned
They have access to the frame in which they are defined

```python
def make_adder(n):
    """Return a function that adds n to its argument."
    
    >>> add_three = make_adder(3)
    ... add_three(4)
    7
```

Call Expressions as Operators

```
make_adder(1) (2)
```

Interpreter Session
- This concept usually trips some students up
- Let's see it in the interpreter

Higher-Order Functions
Functions are first-class: they can be manipulated as values in Python

Higher-order function: a function that takes a function as an argument value or returns a function as a return value

Higher order functions:
- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions

Tomorrow...
- How do higher order functions look in Environment diagrams?
- Homework 1 is due
- Office hours today, see website