61A Lecture 5

Wednesday, September 7

Office Hours: You Should Go!

You are not alone!

http://inst.eecs.berkeley.edu/~cs61a/fall/www/staff.html

Reminder: Multiple Assignment & Return Values

```python
from operator import floordiv, mod

def divide_exact(n, d):
    '''Return the quotient and remainder of dividing n by d.
    >>> q, r = divide_exact(13, 5)
    >>> q
    2
    >>> r
    3
    
    Multiple assignment to two names
    
    Multiple return values, separated by commas
    
    Integer division, which rounds down
    Integer remainder after dividing
    
    >>> floordiv(13, 5)
    2
    >>> mod(13, 5)
    3
    >>> (13 // 5, 13 % 5)
    (2, 3)
```

The Structure of Project 1

Two functions implement the game simulation

```python
def play(...):
    while game is not over:
        get a plan (from the current player's strategy)
        call take_turn with a dice and plan
        return winner

def take_turn(...):
    while turn is not over:
        get an action (from plan) and outcome (from dice)
        call an action
        return points scored during the turn
```

The Structure of Project 1

Four types of functions are involved in simulating game

<table>
<thead>
<tr>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>(integer, integer)</td>
</tr>
<tr>
<td></td>
<td>Action</td>
</tr>
<tr>
<td></td>
<td>(integer, integer, boolean)</td>
</tr>
<tr>
<td></td>
<td>(integer, boolean)</td>
</tr>
<tr>
<td>Plan</td>
<td>integer</td>
</tr>
<tr>
<td>Strategy</td>
<td>(integer, integer)</td>
</tr>
<tr>
<td>Dice</td>
<td>No arguments</td>
</tr>
<tr>
<td></td>
<td>integer</td>
</tr>
</tbody>
</table>

The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

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
def cube(k):
    return pow(k, 3)

def summation(n, term):
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

def pi_term(k):
    return 8 / (k * 4 - 3) / (k * 4 - 1)

# Local function definitions; returning functions

def make_adder(n):
    def adder(k):
        return k + n
    return adder

def compose1(f, g):
    def h(x):
        return f(g(x))
    return h

@main
def run():
    interact()
Locally Defined Functions: Call Expressions

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

```
def make_adder(n):
def adder(k):
return k + n
return adder
make_adder(1)(2)
```

---

Locally Defined Functions: Environments

---

The Environment for Function Composition

```
def compose1(f, g):
def h(x):
    return f(g(x))
return h
```

```
a1 = make_adder(1)
a2 = make_adder(2)
compose1(a1, a2)(3)
```

---

Lambda Expressions

```
>>> ten = 10
>>> square = x * x
>>> square = lambda x: x * x
>>> square(4)
16
```

```
An expression: this one evaluates to a number
Also an expression: evaluates to a function
A function with formal parameter x and body “return x * x”
```

```
Lambda expressions are rare in Python, but important in general
```

---

More Higher-Order Function Examples

---

(Demo)