Announcements

- Homework 1 due Wednesday 9/10 at 2pm. Late homework is not accepted!

- Homework parties on Monday 9/8 (Today!)
  - 3pm–4pm in Wozniak Lounge in Soda Hall (100 person capacity)
  - 6pm–8pm in 2050 Valley Life Sciences Building (408 person capacity)

- More sections for students without prior programming experience! http://cs61a.org

- Take-home quiz 1 starts Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - Open-computer, but no external resources or friends
  - Content Covered: Lectures through last Friday 9/5 (same topics as Homework 1)

- Project 1 due next Wednesday 9/17 at 11:59pm
Iteration Example
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
    pred, curr = 0, 1  # First two Fibonacci numbers
    k = 1              # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr

The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987

The next Fibonacci number is the sum of the current one and its predecessor
Discussion Question 1

What does pyramid compute?

def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total

I'm still here
Designing Functions
Characteristics of Functions

A function's domain is the set of all inputs it might possibly take as arguments.

- **def square(x):**
  
  """Return X * X."""

  *x is a real number*

  *returns a non-negative real number*

A pure function's behavior is the relationship it creates between input and output.

- **def fib(n):**
  
  """Compute the nth Fibonacci number, for N >= 1."""

  *n is an integer greater than or equal to 1*

  *returns a Fibonacci number*

  *return value is the nth Fibonacci number*
A Guide to Designing Function

Give each function exactly one job.

Don’t repeat yourself (DRY). Implement a process just once, but execute it many times.

Define functions generally.
Generalization
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

<table>
<thead>
<tr>
<th>Shape</th>
<th>Area Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square</td>
<td>$A = r^2$</td>
</tr>
<tr>
<td>Circle</td>
<td>$A = \pi r^2$</td>
</tr>
<tr>
<td>Hexagon</td>
<td>$A = \frac{3\sqrt{3}}{2} r^2$</td>
</tr>
</tbody>
</table>

Finding common structure allows for shared implementation

(Demo)
Higher-Order Functions
Generalizing Over Computational Processes

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

\[ \sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15 \]

\[ \sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225 \]

\[ \sum_{k=1}^{5} \frac{8}{(4k-3) \cdot (4k-1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04 \] (Demo)
**Summation Example**

```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 of a single argument (not called "term")

A formal parameter that will be bound to a function

The cube function is passed as an argument value

The function bound to term gets called here
Functions as Return Values

(Demo)
Locally Defined Functions

Functions defined within other function bodies are bound to names in a local frame.

A function that returns a function

```python
def make_adder(n):
    """Return a function that takes one argument k and returns k + n."
    def adder(k):
        return k + n
    return adder
```

The name add_three is bound to a function

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

A def statement within another def statement

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

An expression that evaluates to a function

An expression that evaluates to its argument

Operator

Operand

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

func adder(k)

make_adder(1)

1

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

func adder(k)
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
The Game of Hog

(Demo)