

## 61A Lecture 4

---

Monday, September 9

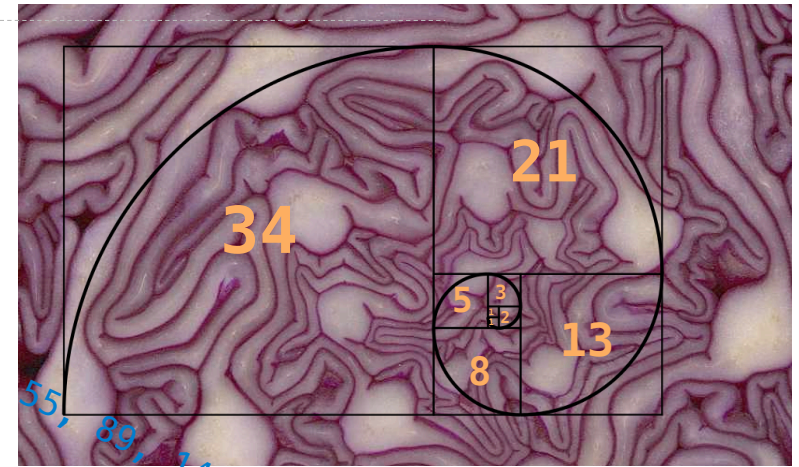
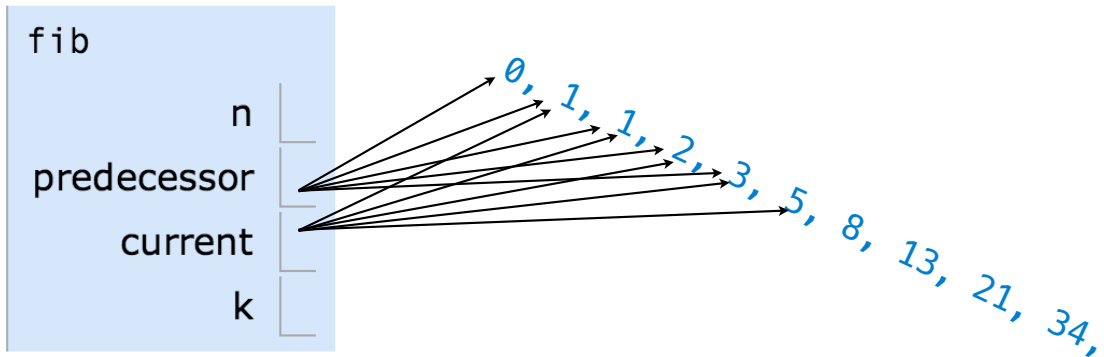
## Announcements

---

- Homework 1 due Tuesday 9/10 at 5pm; Late homework is not accepted!
- Quiz on Wednesday 9/11 released at 1pm, due Thursday 9/12 at 11:59pm
  - *Open-computer*: You can use the Python interpreter, watch course videos, and read the online text (<http://composingprograms.com>).
  - *No external resources*: Please don't search for answers, talk to your classmates, etc.
  - *Content Covered*: Lectures through last Friday 9/6; Same topics as Homework 1.
- Project 1 due next Thursday 9/19 at 11:59pm

## Iteration Example

## The Fibonacci Sequence



```
def fib(n):
```

```
    """Compute the nth Fibonacci number, for n >= 2."""
```

```
    predecessor, current = 0, 1 # First two Fibonacci numbers
```

```
    k = 2 # Tracks which Fibonacci number is called current
```

```
    while k < n:
```

```
        ▶ predecessor, current = current, predecessor + current
```

```
        k = k + 1
```

```
    return current
```

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



Example: <http://goo.gl/vfymhd>

## Discussion Question

Complete the following definition by placing an expression in \_\_\_\_\_ .

```
def choose(total, selection):  
    """Return the number of ways to choose SELECTION items from TOTAL.
```

*choose(n, k) is typically defined in math as:*  $\frac{n!}{(n-k)! \cdot k!}$

```
>>> choose(5, 2)
```

```
10
```

```
>>> choose(20, 6)
```

```
38760
```

```
"""
```

```
ways = 1
```

```
selected = 0
```

```
while selected < selection:
```

```
    selected = selected + 1
```

```
    ways, total = ways * total // selected, total - 1
```

```
return ways
```

The diagram shows a grey box containing the mathematical formula for combinations:  $\frac{n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \dots \cdot 2 \cdot 1}$ . A bracket above the denominator indicates it is  $k!$ . Arrows point from the code in the previous block to parts of the formula: a blue arrow from `total` to the numerator, a light blue arrow from `selected` to the denominator, an orange arrow from `selected` to the denominator, and a red arrow from `total - 1` to the numerator.

# Default Arguments

(Demo)

## Designing Functions

## Characteristics of Functions

---

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

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

*x is a number*

```
def choose(n, d):  
    """Return the number of ways to choose D of N items."""
```

*n and d are positive integers with  
n greater than or equal to d.*

A function's **range** is the set of output values it might possibly return.

*return value is a  
positive number*

*return value is a positive integer*

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

*return value is the  
square of the input*

*return value is the number of ways  
to choose d of n items.*



## A Guide to Designing Function

---

Give each function exactly one job.



not



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



Define functions generally.



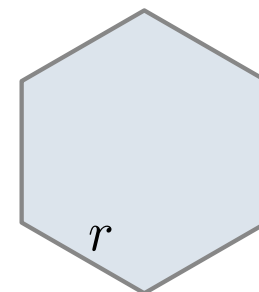
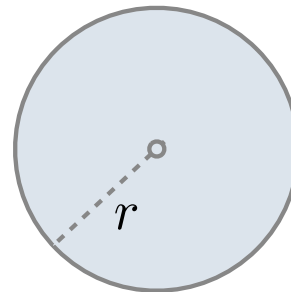
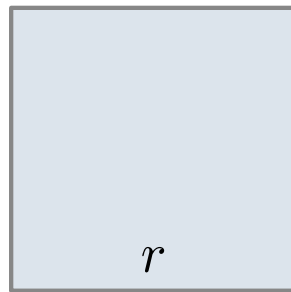
Generalization

## Generalizing Patterns with Arguments

---

Regular geometric shapes relate length and area.

Shape:



Area:

$$1 \cdot r^2$$

$$\pi \cdot r^2$$

$$\frac{3\sqrt{3}}{2} \cdot r^2$$

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

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

Function of a single argument  
(not called term)

```
def summation(n, term):  
    """Sum the first n terms of a sequence.
```

A formal parameter that will  
be bound to a function

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

```
225
```

The cube function is passed  
as an argument value

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

$0 + 1^3 + 2^3 + 3^3 + 4^3 + 5^3$

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

```
def make_adder(n):  
    """Return a function that takes one argument k and returns k + n.  
  
    >>> add_three = make_adder(3)  
    >>> add_three(4)  
    7  
    """  
    def adder(k):  
        return k + n  
    return adder
```

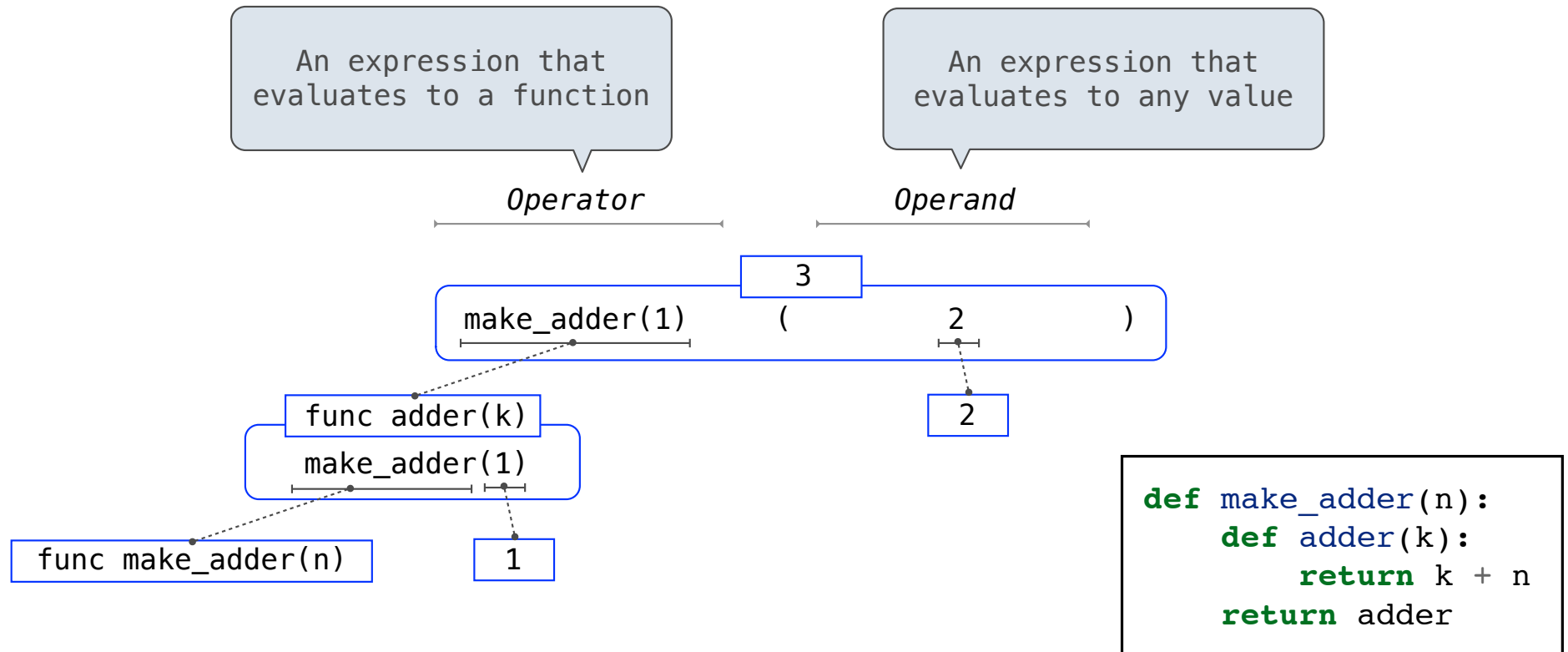
The name `add_three` is bound  
to a function

A local  
def statement

Can refer to names in the  
enclosing function



## Call Expressions as Operator Expressions



## 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)