61A Lecture 4

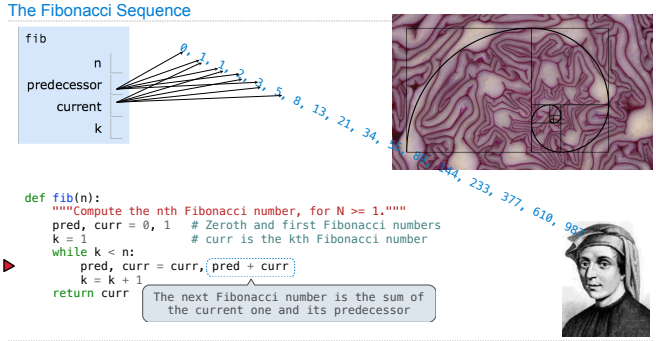
Wednesday, January 28

Announcements

- Homework 1 due Wednesday 1/28 at 11:59pm. Late homework is not accepted!
  - Check your submission on ok.cs61a.org and submit again if it's not right
- Take-home quiz 1 released Wednesday 1/28, due Thursday 1/29 at 11:59pm
  - Open-computer, open notes, closed friends
  - Content Covered: Lectures through Monday 1/26 (same topics as Homework 1)
  - If you receive 0/3, talk to your TA (or me) about how to approach the course
- Extra lectures: Earn 1 unit (pass/no pass) by learning about optional additional topics
  - First extra lecture: Thursday 1/29 5-6:30pm in 2050 VLSB (Come there to learn more)
- Project 1 due Thursday 2/5 at 11:59pm

Iteration Example

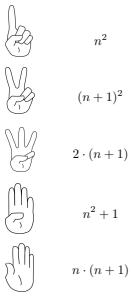
The Fibonacci Sequence




```
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
    pred, curr = 0, 1   # Zeroth and first Fibonacci numbers
    k = 1               # curr is the kth Fibonacci number
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

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

Discussion Question 1



n^2

$(n+1)^2$

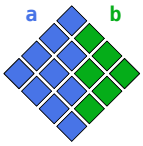
$2 \cdot (n+1)$

$n^2 + 1$

$n \cdot (n+1)$

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
```



Designing Functions

Characteristics of Functions

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

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

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

x is a real number

n is an integer greater than or equal to 1

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

returns a non-negative real number

returns a Fibonacci number

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 nth Fibonacci number

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.

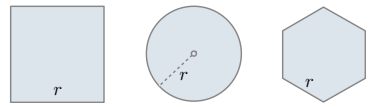
Define functions generally.

Generalization

Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:



Area: $1 \cdot r^2$ $\quad \pi \cdot r^2$ $\quad \frac{3\sqrt{3}}{2} \cdot r^2$

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.

$$\sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 \qquad = 15$$

$$\sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 \qquad = 225$$

$$\sum_{k=1}^{5} \frac{8}{(4k-3)\cdot(4k-1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} \qquad = 3.04$$

(Demo)

Summation Example

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

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

    >>> summation(5, cube)
    225
    """
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total
```

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

0 + 1 + 8 + 27 + 64 + 125

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

```
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
```

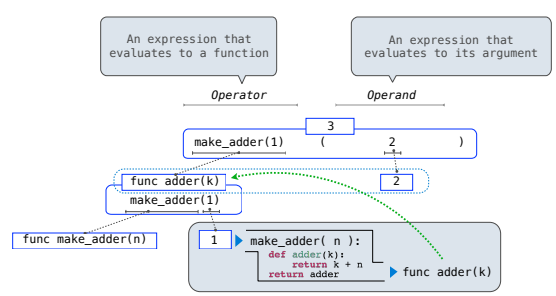
A function that returns a function

The name add_three is bound to a function

A def statement within another def statement

Can refer to names in the enclosing function

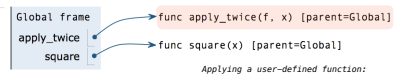
Call Expressions as Operator Expressions



Environments for Higher-Order Functions

Names can be Bound to Functional Arguments

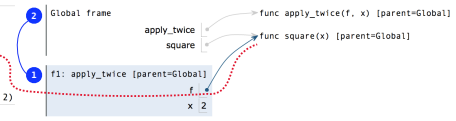
```
1 def apply_twice(f, x):
2     return f(f(x))
3
4 def square(x):
5     return x * x
6
7 result = apply_twice(square, 2)
```



Applying a user-defined function:

- Create a new frame
- Bind formal parameters (f & x) to arguments
- Execute the body: return f(f(x))

```
1 def apply_twice(f, x):
2     return f(f(x))
3
4 def square(x):
5     return x * x
6
7 result = apply_twice(square, 2)
```



Interactive Diagram

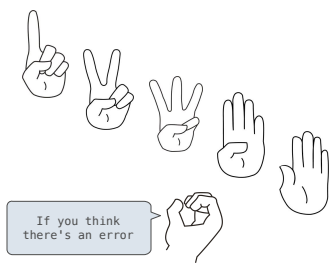
Discussion Question

What is the value of the final expression below? (Demo)

```
def repeat(f, x):
    while f(x) != x:
        x = f(x)
    return x

def g(y):
    return (y + 5) // 3

result = repeat(g, 5)
```



Interactive Diagram