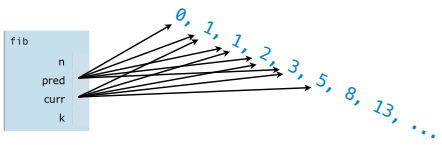
61A Lecture 4

Friday, August 31

The Fibonacci Sequence




```
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1   # First two Fibonacci numbers
    k = 2               # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

Example: http://goo.gl/dcaf0

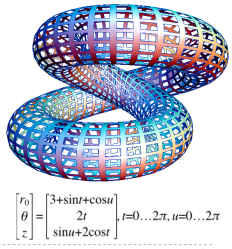
Practical Guidance: the Art of the Function

Give each function exactly one job.

vs

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

Define functions generally.

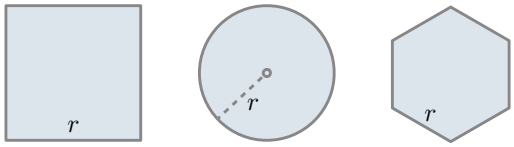


$$\begin{bmatrix} r_0 \\ \theta \\ z \end{bmatrix} = \begin{bmatrix} 3+\sin t+\cos u \\ 2t \\ \sin u+2\cos t \end{bmatrix}, t=0\ldots 2\pi, u=0\ldots 2\pi$$

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

Generalizing Over Computational Processes

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

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

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

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

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

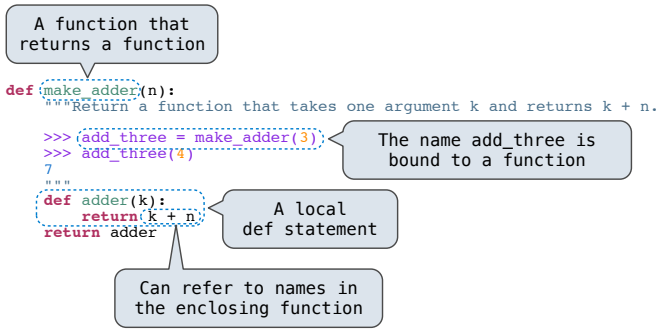
The cube function is passed as an argument value

The function bound to term gets called here

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

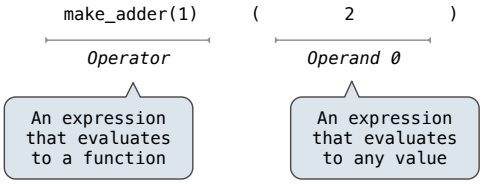
Locally Defined Functions

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



Call Expressions as Operator Expressions

make_adder(1)(2)




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


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

Pig Introduction

(Demo)