COMPUTER SCIENCE IN THE NEWS
The (Literal) Evolution of Music
• Scientists from Imperial College London created a computer program powered by Darwinian natural selection.
  Theory that cultural changes in language, art, and music evolve like living things do, since consumers choose what is “popular”.
• Program would produce loops of random sounds and analyze opinions of musical consumers. The top loops were then “mated”.
• “DarwinTunes” (darwintunes.org) has evolved through at least 2513 generations.
Source: http://www3.imperial.ac.uk/newsandeventspggrp/imperialcollege/newssummary/news_19-6-2012-9-59-1

REVIEW

def foo(n):
    k = 2
    while k < n:
        if n % k == 0:
            return False
        k = k + 1
    return True

foo(7)? foo(9)? foo(1)?

TODAY

Domain and range of functions
... and feeding functions to functions

DOMAIN AND RANGE

Domain is the set of values that the function is defined for.

Range (or image) is the set of values that the function returns.

Domain of square: All real numbers.
Range of square: All non-negative real numbers.

DOMAIN AND RANGE

Remember the domain and range of the functions that you are writing and using!

Many bugs arise because programmers forget what a function can and cannot work with.

The domain and range are especially important when working with higher order functions.
PROBLEM: FINDING SQUARES

>>> square_of_0 = 0*0
>>> square_of_1 = 1*1
>>> square_of_2 = 2*2
>>> square_of_3 = 3*3
...
>>> square_of_65536 = 65536*65536
>>> square_of_65537 = 65537*65537
...

PROBLEM: FINDING SQUARES

>>> square_of_0 = 0*0
>>> square_of_1 = 1*1
>>> square_of_2 = 2*2
>>> square_of_3 = 3*3
...
>>> square_of_65536 = 65536*65536
>>> square_of_65537 = 65537*65537
... Can we generalize?

FINDING SQUARES: GENERALIZATION

THE SQUARE OF A NUMBER

return n * n

IS THE NUMBER MULTIPLIED BY ITSELF

PROBLEM: SUMS OF SERIES

def sum_of_n_squares(n):
    '''Returns 1^2 + 2^2 + 3^2 + ... + n^2.'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + square(k), k + 1
    return sum

def sum_of_n_cubes(n):
    '''Returns 1^3 + 2^3 + 3^3 + ... + n^3.'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + cube(k), k + 1
    return sum

def sum_of_n_sines(n):
    '''Returns sin(1) + sin(2) + sin(3) + ... + sin(n).'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + sin(k), k + 1
    return sum

def sum_of_n_sqrts(n):
    '''Returns sqrt(1) + sqrt(2) + ... + sqrt(n).'''  
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + sqrt(k), k + 1
    return sum

and so on...
**Problem: Sums of Series**

```python
from math import sin, sqrt

def sum_of_n_sines(n):
    '''Returns sin(1) + sin(2) + sin(3) + ... + sin(n).'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + sin(k), k + 1
    return sum

def sum_of_n_squares(n):
    '''Returns sin(1)^2 + sin(2)^2 + ... + sin(n)^2.'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + square(k), k + 1
    return sum

def sum_of_n_cubes(n):
    '''Returns sin(1)^3 + sin(2)^3 + ... + sin(n)^3.'''
    sum, k = 0, 1
    while k <= n:
        sum, k = sum + cube(k), k + 1
    return sum

Can we generalize?
```

---

**Announcements**

- HW2 is released, due **Tuesday, June 26**.
- Discussions earlier held in 320 Soda will now be held in **310 Soda** for the next six weeks. We will move back to 320 Soda for the last two weeks.
- Groups for studying and midterms will be assembled on **Thursday, June 21** in discussion section.

---

**Evaluation of Primitive Expressions**

```python
>>> 3
3
```  
```
>>> x = 5
```

**Function Names**

```python
>>> square
evaluates to
<function square at 0x000000002279648>

Python's representation of '

square is the name of the
function, or "machine", that
squares its input
```

```python
>>> cube
evaluates to
<function cube at 0x0000000022796C8>
```
USING PRIMITVE EXPRESSIONS

We have seen primitive expressions used as arguments to functions:

```python
>>> square(3)
9
>>> x = 4
>>> square(x)
16
```

Can we then use function names as arguments?

PROBLEM: SUMS OF SERIES

```python
def sum_of_n_squares(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + square(k)
        k = k + 1
    return sum
def sum_of_n_cubes(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + cube(k)
        k = k + 1
    return sum
def sum_of_n_sines(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + sin(k)
        k = k + 1
    return sum
def sum_of_n_sqrts(n):
    sum, k = 0, 1
    while k <= n:
        sum = sum + sqrt(k)
        k = k + 1
    return sum
```

FUNCTIONS AS ARGUMENTS

```python
def summation(n, term):
    sum, k = 0, 1
    while k <= n:
        sum = sum + term(k)
        k = k + 1
    return sum
def sum_of_n_squares(n):
    return summation(n, square)
```

FUNCTIONS AS ARGUMENTS

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

FUNCTIONS AS ARGUMENTS

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

FUNCTIONS AS ARGUMENTS

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

FUNCTIONS AS ARGUMENTS

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

def sum_of_n_cubes(n):
    return summation(n, lambda x: x ** 3)

def sum_of_n_sines(n):
    return summation(n, sin)

def sum_of_n_positive_ints(n):
    return summation(n, lambda x: x)

Functions that can take other functions as arguments are considered higher order functions.

The domains of these functions now include other functions.

Functions can be treated as data!
**HOFs ELSEWHERE**

Recipe (Function) + Ingredients ➔ Food

YOU (Knead or) are a function (Knead or)

---

**FUNCTIONS AS ARGUMENTS**

```python
def summation2(n, term, next):
    sum, k = 0, 1
    while k <= n:
        sum = sum + term(k)
        k = next(k)
    return sum
```

---

**FUNCTIONS AS ARGUMENTS: PRACTICE**

```python
def summation2(n, term, next):
    sum, k = 0, 1
    while k <= n:
        sum = sum + term(k)
        k = next(k)
    return sum

def summation(n, term):
    return summation2(n, term, lambda x: x + 1)
```

---

**HIGHER ORDER FUNCTIONS: PRACTICE**

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

def sum_of_n_even(n):
    '''Returns the sum of the first n even numbers.'''
    return summation(n, lambda x: x + 1)

def sum_of_n_odd(n):
    '''Returns the sum of the first n odd numbers.'''
    return summation(n, lambda x: x + 1)

def sum_of_n_starting_from_m(m, n):
    '''Returns the sum of the first n numbers starting from m.'''
    return summation(n, lambda x: x + (m - 1))
```
Higher Order Functions: Practice

```python
def summation(n, term):
    sum, k = 0, 1
    while k <= n:
        sum = sum + term(k)
        k = k + 1
    return sum
def sum_of_n_even(n):
    '''Returns the sum of the first n even numbers.'''
    return summation(n, lambda x: 2 * x)
def sum_of_n_odd(n):
    '''Returns the sum of the first n odd numbers.'''
    return summation(n, lambda x: 2 * x + 1)
def sum_of_n_starting_from_m(m, n):
    '''Returns the sum of the first n numbers starting from m.'''
    return summation(n, lambda x: x + m)
```

Higher Order Functions: Practice

The value of the derivative of a function $f$ at a point $x$ can be approximated as

$$
\frac{f(x + \Delta x) - f(x)}{\Delta x}
$$

if $\Delta x$ is really small.

Write the function `derivative` that takes as arguments a function $f$, a point $x$, and a really small value `delta_x`, and returns the approximate value of the derivative at $x$.

```python
def deriv(f, x, delta_x):
    return (f(x + delta_x) - f(x)) / delta_x
```

Conclusion

- It is important to remember the domain and range of functions!
- Functions can take functions as arguments.
- Preview: Functions can also return functions.