Welcome *Back* to CS 5 *Black*!

PENGUIN GETS $1B IN FUNDING

San Jose (AFP): A penguin who was chased out of a Harvey Mudd College computer science lab by an angry mob has turned the experience into a startup with a billion dollars in venture funding. The new company will market an app that helps penguins track and dodge predators. "The market is huge," said one investor. "Antarctica is full of penguins and they don't have any way to know where the sharks are. We expect massive returns."

The founding penguin will celebrate in a local sushi restaurant.

Read sections 2.1–2.9

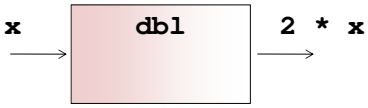
Python and the Command Line

```
bow:2:1169> python3
Python 3.4.5 (default, Jul 03 2016, 13:32:18) [GCC] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> "Hello, world"
'Hello, world'
>>> 7*6
42
>>> import math
>>> math.pi
3.141592653589793
>>> equator = 40000 / 1.609
>>> equator / pi / 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
>>> equator / math.pi / 2
3956.6176032789394
>>> from math import pi
>>> equator / pi
7913.235206557879
>>> quit()
bow:2:1170>
```

Python makes it easy to experiment!

Defining Your Own Functions!

```
def dbl(x):
  return 2 * x
```




```
def dbl(myArgument):
  myResult = 2 * myArgument
  return myResult
```

Notice the indentation. This is done using "tab" and it's absolutely necessary!

Be sure to set your editor to indent using spaces!

Docstrings!

```
def dbl(x):
  """This function takes a number x
     and returns 2 * x"""
  return 2 * x
```

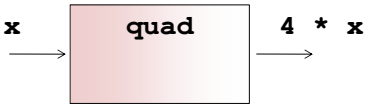
This is sort of like teaching your programs to talk to you!

Docstrings…and Comments

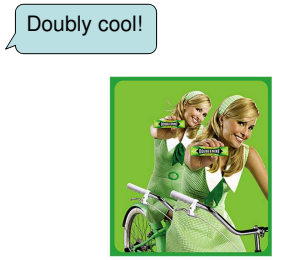
```
# Doubling program
# Author:  Ran Libeskind-Hadas
# Date:  August 27, 2011

def dbl(x):
  """This function takes a number x
     and returns 2 * x"""
  return 2 * x
```

Composition of Functions

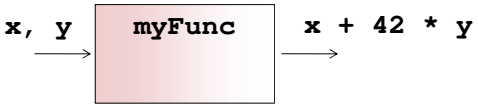
```
def quad(x):
  return 4 * x
```

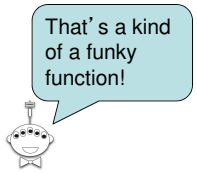



```
def quad(x):
  return dbl(dbl(x))
```



Multiple Arguments...






```
# myFunc
# Author:  Ran Libeskind-Hadas
# Date:  August 27, 2011

def myFunc(x, y):
  """Returns x + 42 * y"""
  return x + 42 * y
```

Mapping with Python...

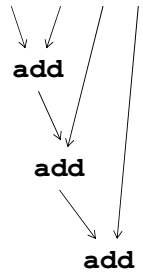
```
def dbl(x):
  """Returns 2 * x"""
  return 2 * x

>>> list(map(dbl, [0, 1, 2, 3, 4]))
[0, 2, 4, 6, 8]

def evens(n):
  myList = range(n)
  doubled = list(map(dbl, myList))
  return doubled
```

Alternatively….

```
def evens(n):
  return list(map(dbl, range(n)))
```

`reduce`-ing with Python...

```
from functools import reduce

def add(x, y):
  """Returns x + y"""
  return x + y

>>> reduce(add, [1, 2, 3, 4])
10
```



Google's "Secret"

This is what put Google on the map!

Google Research Publications

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

Abstract

MapReduce is a programming model and an associated implementation for processing intermediate key/value pairs, and a reduce function that merges all intermediate values paper.

Programs written in this functional style are automatically parallelized and executed on scheduling the program's execution across a set of machines, handling machine failure

Try This…

Write a function called `span` that returns the difference between the maximum and minimum numbers in a list…

```
>>> span([3, 1, 42, 7])
41
>>> span([42, 42, 42, 42])
0
```

```
min(x, y)
max(x, y)
```

These are built into Python!

Try This...

1. Write a python function called `gauss` that accepts a positive integer argument N and returns the sum $1 + 2 + \ldots + N$
2. Write a python function called `sumOfSquares` that accepts a positive integer N and returns the sum $1^2 + 2^2 + 3^2 + \ldots + N^2$

You can write extra "helper" functions too!

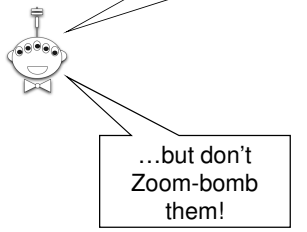
return vs print..


```
def dbl(x):
    return 2 * x

def trbl(x):
    print(2 * x)

def happy(yay):
    y = dbl(yay)
    return y + 42

def sad(boo):
    y = trbl(boo)
    return y + 42
```

```
def friendly(pal):
    y = dbl(pal)
    print(y, "is very nice!")
    return y + 42
```

Strings are in single or double quotes

The Alien's Life Advice



What Happens Inside a Function?

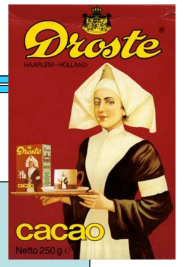
```
def f(x):
    x = x-1
    return g(x)+1

def g(x):
    return x*2

def h(x):
    if x%2 == 1:        # x odd
        return f(x) + x//2
    else:               # x even
        return f(f(x))
```

```
h(3):  ↑6
  return f(3) + 3//2
         ↓  ↑5
f(3):
    return g(2) + 1
           ↓  ↑4
g(2):
  return 4
```

Two key points…

- Functions return to where they were called from
- Each function keeps its own values of its variables

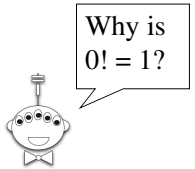
Recursion…



$n! = n \times (n-1) \times (n-2) \times \ldots \times 1$

$n! = n \times [(n-1)!]$ "inductive definition"

$0! = 1$ "base case"



Math Induction = CS Recursion

Math

inductive definition

```
0! = 1

n! = n×[(n-1)!]
```

Python (Functional)

recursive function

```
# recursive factorial
def factorial(n):
```

Math Induction = CS Recursion

Math

inductive definition

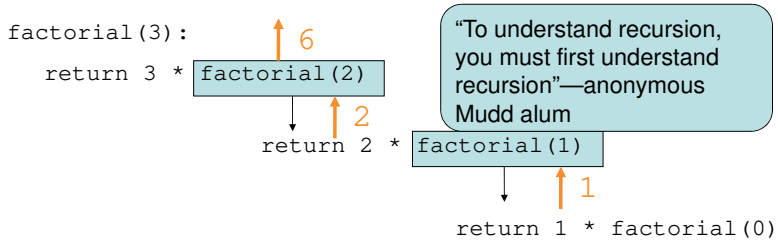
```
0! = 1

n! = n×[(n-1)!]
```

Python (Functional)

recursive function

```
# recursive factorial
def factorial(n):
  if n == 0:
     return 1
  else:
     return n*factorial(n - 1)
```

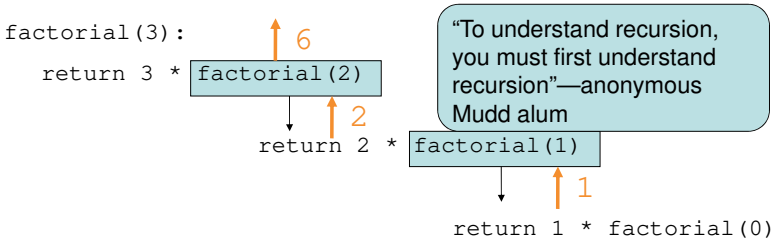
Is Recursion Magic?



"To understand recursion, you must first understand recursion"—anonymous Mudd alum

```
# recursive factorial
def factorial(n):
  '''This computes n!'''
  if n == 0:
    return 1
  else:
    return n*factorial(n-1)
```

Is Recursion Magic?



"To understand recursion, you must first understand recursion"—anonymous Mudd alum

Computing the Length of a List

```
>>> len([1, 42, "spam"])
3
>>> len([1, [2, [3, 4]]])
```


```
def len(List):
  '''Returns the length of List'''
```

A Tower of Fun!

Math

$tower(3) = 2^{2^2}$

$tower(4) = 2^{2^{2^2}}$

$tower(5) =$

Python (Functional)

recursive function

```
# recursive tower
def tower(n):
```

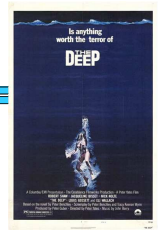
Reversing a List

```
>>> reverse([1, 2, 3, 4])
[4, 3, 2, 1]

def reverse(L):
  '''Returns a new list that is the
     reverse of the input list'''
```

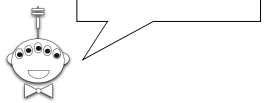
Reversing a List

```
>>> reverse([1, [2, [4, 5], 6], 7])
```

Deep-Reversing a List




```
>>> reverse([1, [2, [4, 5], 6], 7])
[7, [2, [4, 5], 6], 1]

>>> deepReverse([1, [2, [4, 5], 6], 7])
[7, [6, [5, 4], 2], 1])
```



Recursion = :^)

Recursion, conditional statements, and lists suffice to give us a Turing-complete programming language!

Variables, assignment (=), if, while, etc. are all unnecessary!