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

```python
>>> python3
Python 3.4.5 (default) , Jul 4 2016, 13:32:18) [GCC] on Linux
Type "help", "copyright", "credits" or "license" for more information.
>>> "Hello, world"
'Hello, world'
>>> 7+0
7
>>> 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
8966.616032785984
>>> from math import pi
>>> equator / pi
7913.235206357879
>>> quit()
```

Python makes it easy to experiment!

---

Defining Your Own Functions!

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

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

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

Docstrings!

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

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

Be sure to set your editor to indent using spaces!
# 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

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

- Doubly cool!

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

---

## Multiple Arguments...

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

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

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

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

- Alternatively...

```python
def evens(n):
    return list(map(dbl, range(n)))
```
reduce-ing with Python...

```python
from functools import reduce
def add(x, y):
    """Returns x + y""
    return x + y

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

Try This...

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

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

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

These are built into Python!

Google’s “Secret”

This is what put Google on the map!

Try This...

1. Write a python function called `gauss` that accepts a positive integer argument `N` and returns the sum `1 + 2 + … + N`

2. Write a python function called `sumOfSquares` that accepts a positive integer `N` and returns the sum `1^2 + 2^2 + 3^2 + … + N^2`

You can write extra "helper" functions too!
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

Reach out to a stranger in class

…but don’t Zoom-bomb them!

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))

Two key points...
• Functions return to where they were called from
• Each function keeps its own values of its variables

Recursion...

n! = n×(n-1)×(n-2)×…×1

n! = n×[(n-1)!]  “inductive definition”
0! = 1          “base case”

Why is 0! = 1?
Math Induction = CS Recursion

Math
inductive definition

Math
recursive function

0! = 1
n! = n \times (n-1)!

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

Is Recursion Magic?

factorial(3):
  return 3 * factorial(2)
    return 2 * factorial(1)
      return 1 * factorial(0)

"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)
### Computing the Length of a List

```python
>>> len([1, 42, “spam”])
3
>>> len([1, [2, [3, 4]]])
```

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

Python has this built in!

---

### A Tower of Fun!

Math

- `tower(3) = 2^2`
- `tower(4) = 2^{2^2}`
- `tower(5) =`

The tower function is taking recursion to new heights!

Python (Functional)

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

---

### Reversing a List

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

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

---

### Reversing a List

```python
>>> reverse([1, [2, [4, 5], 6], 7])
```
Deep-Reversing a List

```python
>>> 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]
```

This definitely requires recursion! Fun problem on this week’s HW!

Recursion = :^)

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

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