Welcome to CS 5! Be sure to watch your head...

```python
def svtree(t, levels):
    if levels == 0: return
    forward(t)
    left(30)
    svtree(t/2, levels-1)
    right(60)
    svtree(t/2, levels-1)
    left(30)
    backward(t)
```

The colors show the entire stack of calls up to that branch!

The numbers show the value of N for each call...

STACK

- levels = 4
  - t = 200, levels = 4

- levels = 3
  - t = 100, levels = 3

- levels = 2
  - t = 50, levels = 2

- levels = 1
  - t = 25, levels = 1

CS 5: now recursing...

We're computationally complete!

What's next?

putting Python to work...

& adding **building-blocks**

Bourton-on-the-water

Hw 2 – due Monday evening ~ usual time

- pr0 reading – Watson!
- pr1 lab – Turtle!
- pr2, pr3 - Python probs...
- pr4 - extra-credit turtle...!

Or re-cursing, depending on your feelings about recursion!
**Functional programming**

```python
>>> 'fun' in 'functional'
True
```

- representation via list structures (**data**)
- leverage self-similarity (**recursion**)
- create small building blocks (**functions**)

*Composed together* -- to solve/investigate problems.

---

**sum**

```python
def sum(L):
    """ input: L, a list of #s
    output: L's sum
    """
    if len(L) == 0:  # Base Case
        return 0.0
    else:  # Recursive Case
        return L[0] + sum(L[1:])
```

**range**

```python
def range(low, hi):
    """ input: ints low and hi
    output: int list from low to hi, excluding hi
    """
    if low >= hi:
        return []
    else:
        return list(range(low, hi))
```
Data
Functions

\[ [8,9,10] \]

\[ \text{sq( )} \]

\[ [64,81,100] \]

\[ [ \text{sq(x) for x in [8,9,10]} ] \]

...together

Various approaches...

many options for \textit{mapping} a function onto a list:

List Comprehensions

\[ \text{List Comprehension result} \]

\[ [ 2\times \text{ for x in [0,1,2,3,4,5] } ] \]

\[ [0, 2, 4, 6, 8, 10] \]

List Comprehensions?

\[ \text{List Comprehension result} \]

\[ [ 2\times \text{ for x in [0,1,2,3,4,5] } ] \]

\[ [0, 2, 4, 6, 8, 10] \]
Syntax ?!

```python
>>> [ 2*x for x in [0,1,2,3,4,5] ]
[0, 2, 4, 6, 8, 10]
```

at first...

a jumble of characters and random other stuff

a (frustrated!) rendering of an unfamiliar math problem

LCs for Monte Carlo Analysis...

```python
# this line runs guess(42) 1000 times
LC = [ guess_np(42) for x in range(1000) ]

# Let's look at the first 10 of them:
print(LC[0:10])

# Let's find the average:
print("av.#guesses:", sum(LC)/len(LC))
```

Zillion-times testing!

```python
# this runs the doubles-counter 600 times...
cd_np( 600 )  # np: no printing

# Run _that_ 1000 times (600,000 rolls total!)
LC = [ cd_np(600) for x in range(1000) ]

# Look at the first 10 of these
print(LC[0:10])

# Then, find the average:
print("av.dbls /600":, sum(LC)/len(LC))
```

Designing with LCs

```python
# this runs the doubles-counter 600 times...
input >>> [ _______ for x in range(4) ]

output [0, 14, 28, 42]

# Then, find the average:
print("av.dbls /600":, sum(LC)/len(LC))
```

```python
# this runs the doubles-counter 600 times...
input >>> [ ___________ for c in 'igetthis' ]

output [True, False, False, False, False, False, True, False]
```

And if we wanted the ints (in red)...

```python
output [1, 0, 0, 0, 0, 0, 1, 0]
```
Using LCs

**Short and sweet!**

```python
def fun1(L):
    LC = [1 for x in L]
    return sum(LC)
```

**twelve**

```python
def fun2(S):
    LC = [letScore(c) for c in S]
    return sum(LC)
```

'dcs5'

Possible, but not recommendable!

Write each of these functions using list comprehensions...

**One-line** LCs

```python
def len(L):
    LC = [1 for x in L]
    return sum(LC)
```

'cs5'

Maybe too terse!

```python
def letScore(c):
    score = 'twelve'
    return score
```

What fun are these?

```python
def vwl(s):
    LC = [1 for c in s]
    return sum(LC)
```

‘sequoia’

```python
def count(e,L):
    LC = [1 for x in L]
    return sum(LC)
```

'nodds'

```python
def nodds(L):
    LC = [1 for x in L]
    return sum(LC)
```

'cs5'

```python
def letScore(c):
    return score
```

Write each of these functions using list comprehensions...

```python
def nodds(L):
    LC = [1 for x in L]
    return sum(LC)
```

'cs5'

```python
def letScore(c):
    score = 'twelve'
    return score
```

What fun are these?

```python
def vwl(s):
    LC = [1 for c in s]
    return sum(LC)
```

‘sequoia’

```python
def count(e,L):
    LC = [1 for x in L]
    return sum(LC)
```

'nodds'

```python
def nodds(L):
    LC = [1 for x in L]
    return sum(LC)
```

'cs5'

```python
def letScore(c):
    return score
```

Write each of these functions using list comprehensions...

```python
def nodds(L):
    LC = [1 for x in L]
    return sum(LC)
```

'cs5'

```python
def letScore(c):
    score = 'twelve'
    return score
```

What fun are these?

```python
def vwl(s):
    LC = [1 for c in s]
    return sum(LC)
```

‘sequoia’

```python
def count(e,L):
    LC = [1 for x in L]
    return sum(LC)
```

'nodds'

```python
def nodds(L):
    LC = [1 for x in L]
    return sum(LC)
```

'cs5'

```python
def letScore(c):
    return score
```

Write each of these functions using list comprehensions...

```python
def nodds(L):
    LC = [1 for x in L]
    return sum(LC)
```

'cs5'

```python
def letScore(c):
    score = 'twelve'
    return score
```

What fun are these?

```python
def vwl(s):
    LC = [1 for c in s]
    return sum(LC)
```

‘sequoia’

```python
def count(e,L):
    LC = [1 for x in L]
    return sum(LC)
```

'nodds'

```python
def nodds(L):
    LC = [1 for x in L]
    return sum(LC)
```

'cs5'

```python
def letScore(c):
    return score
```

Write each of these functions using list comprehensions...

```python
def nodds(L):
    LC = [1 for x in L]
    return sum(LC)
```
\textbf{hw2pr3: areas from rectangles}

\textbf{Building blocks == CS!}

\begin{align*}
\text{scaledfracs}(\text{low, hi, } N) \\
\text{f_of_fracs}(f, \text{low, hi, } N) \\
\text{integrate}(f, \text{low, hi, } N)
\end{align*}

only a few lines...

Where are the LCs?
Quiz!

A range of list comprehensions...
Write Python's result for each L.C.:

Name(s): ____________________

[ n**2 for n in range(0,5) ]

[ 42 for z in [0,1,2] ]
[ z for z in [0,1,2] ]

[ s[1::2] for s in ['aces','451!'] ]

[ -7*b for b in range(-6,6) if abs(b)>4 ]

[ a*(a-1) for a in range(8) if a%2==1 ]

Got it! But what about that name?