Welcome to CS 5! Be sure to watch your head...
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)
Today only!

Office hours in Olin 1269 (not the LAC)
```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

<table>
<thead>
<tr>
<th>levels = 4</th>
<th>t = 200, levels = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>levels = 3</td>
<td>t = 100, levels = 3</td>
</tr>
<tr>
<td>levels = 2</td>
<td>t = 50, levels = 2</td>
</tr>
<tr>
<td>levels = 1</td>
<td>t = 25, levels = 1</td>
</tr>
</tbody>
</table>
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 numbers show the value of N for each call …

The colors show the entire stack of calls up to that branch!
Bourton-on-the-water
Bourton-on-the-water

town of 2000 people
Bourton-on-the-water's 1/9 model
has a level-2 model...
has a level-2 model...
and a level-3 model...
and a level-3 model...
and even a (very small!) level-4 model
We're computationally complete!

What's next?

putting Python to work...

& adding building-blocks
this week's hw2pr0

Category: U.S. Cities. Clue: Its largest airport is named for a World War II hero, its second largest for a World War II battle.

Watson
**functional programming**

'fun' in 'functional'

True

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

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

*Functional programming*

conceptually concise vs. efficient *for the computer*...
Data

[13, 14, 15]

[3, 4, 5, 6, 7, 8, 9]

Functions

\text{sum}( )

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

**Base Case**
```
if len(L) == 0:
    return 0.0
```

**Recursive Case**
```
else:
    return L[0] + sum(L[1:])
```
def range(low, hi, step):
    """ input:  ints low and hi
    output: int list from low to hi
    """
    if low >= hi:
        return []
    else:
        return list([low]) + range(low+step, hi, step)

list(range(low, hi))
```python
>>> sum(list(range(1, 101)))
```

I'd bet you have a 50/50 chance on this one...

- Ben, L'14
```python
>>> sum(list(range(1,101)))
```

I'd bet you have a 50/50 chance on this one…

- Ben. L’14


Gauss's storied story…

and 100 more…
sum $\text{and }$ list range

>>> `sum(list(range(1,101)))`

5050

$1 + 2 + 3 + \ldots + 98 + 99 + 100$

50+51

50 101's == 5050

Looks sort of scruffy for a 7-year old...!
Data

Functions

\[ \{ 8, 9, 10 \} \]

\[ \text{sq}(\ ) \]

\[ \{ 64, 81, 100 \} \]

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

...together
Various approaches...

many options for mapping a function onto a list:

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

# three syntaxes for applying a function to a list of data:
# usually, people choose L3!
L1 = list(map(dbl, range(6)))
L2 = [dbl(x) for x in range(6)]
L3 = [2*x for x in range(6)]
```
List Comprehensions

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

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

What's the syntax saying here?
List Comprehensions

**Expression** to evaluate for each list element

**Name** for each list element

The **list** - or **string** to run

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

What's the syntax saying here?
List Comprehensions

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

This "runner" variable can have any name...

\[ x \text{ takes on each value} \]

\[ \text{and } 2x \text{ is output for each one} \]
List Comprehensions

```python
>>> [ 10*x for x in [0,1,2,3,4,5] if x%2==0 ]
[0, 20, 40]
```

```python
>>> [ y*21 for y in list(range(0,3)) ]
[0, 21, 42]
```

```python
>>> [ s[1] for s in ["hi", "5Cs!"] ]
["i", "C"]
```
Quiz!

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

- `[n**2 for n in range(0,5)]` → `[0, 1, 4, 9, 16]`
- `[42 for z in [0,1,2]]` → `[42, 42, 42]`
- `[z for z in [0,1,2]]` → `[0, 1, 2]`
- `[s[1::2] for s in ['aces','451!']]` → `[ac, es, 45]`
- `[-7*b for b in range(-6,6) if abs(b)>4]` → `[28, 21, 14, 7, 0]`
- `[a*(a-1) for a in range(8) if a%2==1]` → `[0, 6, 20, 42, 90]`

Name(s): _______________________

Got it! But what about that name?
**Quiz!**  
A range of list comprehensions...  
Write Python's result for each L.C.:

<table>
<thead>
<tr>
<th>L.C.</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ n**2 \text{ for } n \text{ in } \text{list range}(0,5) ]</td>
<td>[0, 1, 4, 9, 16]</td>
</tr>
<tr>
<td>[ 42 \text{ for } z \text{ in } {0,1,2} ]</td>
<td>[42, 42, 42]</td>
</tr>
<tr>
<td>[ z \text{ for } z \text{ in } {0,1,2} ]</td>
<td>[0, 1, 2]</td>
</tr>
<tr>
<td>[ s[1::2] \text{ for } s \text{ in } ['aces', '451!'] ]</td>
<td>[cs, 51]</td>
</tr>
<tr>
<td>[ -7*b \text{ for } b \text{ in } \text{list range}(-6,6) \text{ if } \text{abs}(b)&gt;4 ]</td>
<td>[42, 35, -35]</td>
</tr>
<tr>
<td>[ a*(a-1) \text{ for } a \text{ in } \text{list range}(8) \text{ if } a%2==1 ]</td>
<td>[0, 6, 20, 42]</td>
</tr>
</tbody>
</table>
Quiz! A range of list comprehensions...
Write Python's result for each L.C.:

\[
\begin{align*}
\text{[ } & n^2 \text{ for } n \text{ in list range(0,5) } \\
& [0,1,4,9,16]
\end{align*}
\]

\[
\begin{align*}
\text{[ } & 42 \text{ for } z \text{ in [0,1,2] } \\
& [42,42,42]
\end{align*}
\]

\[
\begin{align*}
\text{[ } & -7b \text{ for } b \text{ in range(-6,6) if } \text{abs}(b)>4 \\
& [-6, -5, 5]
\end{align*}
\]

\[
\begin{align*}
\text{[ } & a^2(a-1) \text{ for } a \text{ in list range(8) if } a\%2==1 \\
& [0, 6, 20, 42]
\end{align*}
\]
List Comprehensions?

Is this really the best name Guido Van Rossum could think of?

Guido van Rossum
From Wikipedia, the free encyclopedia

Guido van Rossum (born 31 January[2] 1956[citation needed]) is a Dutch computer programmer who is best known as the author of the Python programming language. In the Python community, Van Rossum is known as a "Benevolent Dictator For Life" (BDFL), meaning that he continues to oversee the Python development process, making decisions where necessary.[3] He is currently employed by Google, where he spends half his time developing the Python language.
List Comprehensions?

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

Google maps?

FunLists!

Datafuncs?

A list comprehension by *any* other name would be even sweeter...

Guido van Rossum (born 31 January[2] 1956[citation needed]) is a Dutch computer programmer who is best known as the author of the Python programming language. In the Python community, Van Rossum is known as a "Benevolent Dictator For Life" (BDFL), meaning that he continues to oversee the Python development process, making decisions where necessary.[3] He is currently employed by Google, where he spends half his time developing the Python language.
List Comprehensions?

```python
[2*x for x in [0,1,2,3,4,5]]
```

FunLists!

Guido van Rossum (born 31 January, 1956) is a Dutch computer programmer who is best known as the author of the Python programming language. In the Python community, Van Rossum is known as a "Benevolent Dictator For Life" (BDFL), meaning that he continues to oversee the Python development process, making decisions where necessary. He is currently employed by Google, where he spends half his time developing the Python language.
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
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
Syntax ~ *is CS's key resource!*

Where'd the change happen?

a (frustrated!) rendering of an unfamiliar math problem which was likely similar to these...
LCs for Monte Carlo Analysis...

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

Hah! Now I see why they told me I'd be making a zillion euros as spokesalien for this class!

a.k.a. Run it a "zillion" times!
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))
```

the average #doubles per 600 rolls

average == expected
Zillion-times testing!

```python
# this runs the MCMH-counter 300 times...
MCMHnp( 1, 'switch', 300 )  # np: no printing

# Run _that_ 1000 times (300,000 games total!)
LC = [ MCMHnp(300) for x in range(1000) ]

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

# Let's find the average:
print("av. spams (/300):" , sum(LC)/len(LC))
```

Monagasques might use foie gras instead...
On balance?
or maybe lighter is better?
Designing with LCs

input >>> [ 14 * x for x in list(range(4)) ]

output [0, 14, 28, 42]

input >>> [c == 'c' for c in 'igetthis'

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

[ 1, 0, 0, 0, 0, 0, 0, 1, 0 ]

And if we wanted the ints (in red)...?
Using LCs

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

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

What fun are these?
Using LCs

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

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

But one-liners are my specialty...

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

What fun are these!
"One-line" LCs

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

Possible, but *not* recommendable!

I only ever get one line – who are the writers around here...?
def vwl(s):
    LC = [1 for c in s if c in 'aeiou']
    return sum(LC)

def count(e,L):
    LC = [1 for x in L if x == e]
    return sum(LC)
Write each of these functions using list comprehensions...

```python
def nodds(L):
    LC = [1 if x % 2 == 1 for x in L]
    return sum(LC)
example: nodds([3,4,5,7,42]) == 3

def lotto(Y, W):
    LC = [1 if x in Y and x in W for x in L]
    return sum(LC)
example: lotto([5,7,42,47], [3,5,7,44,47]) == 3

def ndivs(N):
    LC = [1 if N % x == 0 for x in range(1, N+1) if x >= 2]
    return sum(LC)
example: ndivs(12) == 6 (1,2,3,4,6,12)

def primesUpTo(P):
    LC = [x for x in range(2, P+1) if ndivs(x) == 2]
    return LC
example: primesUpTo(12) == [2,3,5,7,11]
```

Write each of these functions using list comprehensions...

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

def lotto(Y,W):
    LC = [y for y in Y & W]
    return sum(LC)

def ndivs(N):
    LC = [y for y in range(1,N+1) if N % y == 0]
    return sum(LC)

def primesUpTo(P):
    return LC
```

input:  L, any list of #s
output: the # of odd #s in L
example: nodds([3,4,5,7,42]) == 3

input: Y and W, two lists of "lottery" numbers (ints)
output: the # of matches between Y & W
example: lotto([5,7,42,47], [3,5,7,44,47]) == 3

input: N, an int >= 2
output: the # of positive divisors of N
example: numdivs(12) == 6 (1,2,3,4,6,12)

input: P, an int >= 2
output: the list of prime #s up to + incl. P
example: primesUpTo(12) == [2,3,5,7,11]

Extra!
Write each of these functions using list comprehensions...

```python
def nodds(L):
    LC = [1 for x in L if x%2 == 1]
    return sum(LC)

def lotto(Y,W):
    LC = [x%2 for x in Y]
    return sum(LC)

def ndivs(N):
    LC = [1 for x in range(1,N+1) if N%x==0]
    return sum(LC)

def primesUpTo(P):
    LC = [x for x in range(2,P+1) if ndivs(x)==2]
    return LC
```

Extra!
Write each of these functions using list comprehensions...

```python
def nodds(L):
    LC = [1 for x in L if x%2 == 1]
    return sum(LC)

def lotto(Y,W):
    LC = [1 for y in Y if y in W]
    return sum(LC)

def ndivs(N):
    LC = [1 for x in range(1,N+1) if N%x==0]
    return sum(LC)

def primesUpTo(P):
    LC = [x for x in range(2,P+1) if ndivs(x)==2]
    return LC
```

Extra!
hw2pr3: areas from rectangles

Areas of 4 rectangles

Areas of 8 rectangles
hw2pr3: areas from rectangles

Area of N rectangles in the limit
hw2pr3: Maya Lin, Architect...
Maya Lin, *Artist and Computer Scientist...*

"two-by-four landscape"
Maya Lin, Artist and Computer Scientist...

One building block, deliberately applied, over 50,000 times...
Building blocks == CS!

\[ y = 2x \]

\[ (0,0) \] (2.5,5) (5,10) (7.5,15)

Areas of 8 rectangles

\texttt{scaledfracs}(low, hi, N)

\texttt{f_of_fracs}(f, low, hi, N)

\texttt{integrate}(f, low, hi, N)

only a few lines...

Where are the LCs?
Building blocks == CS!

Good luck with hw#2...

... may this and all your weekends be syntactically smooth!