Welcome to CS 5! Be sure to watch your head...
Welcome to CS 5! Be sure to watch your head...
The colors show the entire stack of calls up to that branch!

So many head-aching signs...!

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...
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
---
levels = 3
---
levels = 2
---
levels = 1
---

t = 200, levels = 4

t = 100, levels = 3

t = 50, levels = 2

t = 25, levels = 1
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></td>
<td><strong>svtree(t/2, levels-1)</strong></td>
</tr>
<tr>
<td>levels = 3</td>
<td>t = 100, levels = 3</td>
</tr>
<tr>
<td></td>
<td><strong>svtree(t/2, levels-1)</strong></td>
</tr>
<tr>
<td>levels = 2</td>
<td>t = 50, levels = 2</td>
</tr>
<tr>
<td></td>
<td><strong>svtree(t/2, levels-1)</strong></td>
</tr>
<tr>
<td>levels = 1</td>
<td>t = 25, levels = 1</td>
</tr>
<tr>
<td></td>
<td>forward(t)</td>
</tr>
</tbody>
</table>
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
CS 5: *now recursing*...

We're computationally complete!

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

What's next?

putting Python to work...

& adding **building-blocks**

Hw 2 – due Monday evening ~ usual time

pr0 reading – *Watson!*

pr1 lab – *Turtle!*

pr2, pr3 - Python probs...

pr4 – *extra-credit turtle... !*
Recursive Art ~ hw2pr4

Submit things that work ...

septagonal confetti

dramatic spiral!
Recursive Art Ex. Cr. ~ hw2pr4
Recursive Art ~ hw2pr4

"Cyriak’s pet snake..."

Submit things that work ...

... and even things that "don't"!
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*...

*functional* vs. *procedural or sequential*
Data

Functions

\[ [13, 14, 15] \]

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

\text{sum( )}

... and their compositions
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

Recursive Case
```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
```

what's cookin' here? 
Recursion's `range` Name(s): ______________

```
def range(low, hi, stride):
    """ input:  low and hi, integers
    output: a list from low upto hi but excluding hi
    ""
    if low >= hi:
        return
    else:
        return
```

```
Extra Extra! What if stride were negative?

Extra! Take a (positive) third input in stride
```
Recursion's `range`

```python
def range(low, hi, stride):
    """ input:  low and hi, integers
    output: a list from low upto hi but excluding hi
    ""
    if low >= hi:
        return []
    else:
        return [low + range(low+1, hi, stride)]
```

Base case: What if low is greater than or equal to hi?

Recursive case: How could we use another call to `range` to help us?!

Extra! Take a (positive) third input in `stride`
Recursion's `range`

```
def range(low, hi , stride):
    """ input: low and hi, integers
    output: a list from low upto hi but excluding hi
    ""
    if low >= hi:
        return []
    else:
        return [low] + range(low+1, hi, stride)
```

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>range(3,7)</code></td>
<td><code>[3,4,5,6]</code></td>
</tr>
<tr>
<td><code>range(3,7,2)</code></td>
<td><code>[3,5]</code></td>
</tr>
</tbody>
</table>

**Name(s):** ______________

**Extra!** Take a (positive) third input in `stride`

**Solution! Try on the back page first!**

We're on target!
Recursion's `range`

```python
def range(low, hi, stride):
    ''' input:  low and hi, integers
    output: a list from low up to hi
    '''
    if low >= hi:
        return
    else:
        return
```

Name(s): ______________

**Pass these backward(42)**

Extra Extra! Take a (positive) third input in `stride`.

Extra! Take a (positive) third input in `stride`.

*We're on target!*
```python
sum and list range

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

I'd bet you have a 50/50 chance on this one…
- Ben, L '14
sum and range

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

I'd bet you have a 50/50 chance on this one…
- Ben, L '14

Gauss's storied story...

http://www.americanscientist.org/template/AssetDetail/assetid/50686
sum and list range

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

5050

1 + 2 + 3 + ... + 98 + 99 + 100

50+51

101

50 101's == 5050
Data Functions

\[
\text{sq}(\ ) \\
[8, 9, 10] \\
[64, 81, 100] \\
[\text{sq}(x) \text{ for } x \text{ in } [8, 9, 10]\ ]
\]

...together
Various approaches...

many options for *mapping* a function onto a list:
List Comprehensions

What's the syntax saying here?

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

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

Result
List Comprehensions

Expression to evaluate for each list element: \(2 \times x\)

Name for each list element: \(x\)

The list - or string to run: \([0, 1, 2, 3, 4, 5]\)

Result: \([0, 2, 4, 6, 8, 10]\)

What's the syntax saying here?
List Comprehensions

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

- `x` takes on each value
- and `2*x` is output for each one
- this "runner" variable can have *any* name...
List Comprehensions

>>> \[ 10x \text{ for } x \text{ in } [0,1,2,3,4,5] \text{ if } x \% 2 == 0 \]\n
result

>>> \[ y*21 \text{ for } y \text{ in } \text{list(range(0,3))} \]\n
result

>>> \[ s[1] \text{ for } s \text{ in } ["hi", "5Cs!"] \]\n
result
Try it! A range of list comprehensions... Write Python’s result for each L.C.:

<table>
<thead>
<tr>
<th>List Comprehension</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]</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>['c','4']</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>[-28,-21,-14,-7,7,14,21,28]</td>
</tr>
<tr>
<td>[ a*(a-1) \text{ for } a \text{ in } \text{list range}(8) \text{ if } a%2==1 ]</td>
<td>[3,15,35,63]</td>
</tr>
</tbody>
</table>
## Quiz!

A range of list comprehensions...

Write Python's result for each L.C.:

\[
\begin{align*}
\text{[ } n^{2} \text{ for } n \text{ in list range(0,5)} \text{ ]} & \quad \rightarrow [0,1,4,9,16] \\
\text{[ } 42 \text{ for } z \text{ in [0,1,2]} \text{ ]} & \quad \rightarrow [42,42,42] \\
\text{[ } s[1::2] \text{ for } s \text{ in ['aces','451!']} \text{ ]} & \quad \rightarrow \text{[ ]} \\
\text{[ } -7*b \text{ for } b \text{ in list range(-6,6) if abs(b)>4 } \text{ ]} & \quad \rightarrow [-6, -5, 5] \\
\text{[ } a*(a-1) \text{ for } a \text{ in list range(8) if a%2==1 } \text{ ]} & \quad \rightarrow [0, 6, 20, 42] \\
\end{align*}
\]

Join with a neighbor and try this on the back page first.

Got it! But what about that name?
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 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.
List Comprehensions?

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

Google maps?

Datafuncts?

FunLists!

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?

>>> 

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

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

A list comprehension by any other name would be even sweeter…

---

Guido van Rossum (born 31 January[2] 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.[3] 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!*

A (frustrated!) rendering of an unfamiliar math problem which was likely similar to these...

\[
\frac{2a \cdot \frac{7^3}{r^4}}{3^2 \cdot \sqrt{3}}
\]

\[
\frac{12xy^4}{18x^3y^2}
\]

\[
\frac{4\sqrt{x} \cdot 3\sqrt{a}}{3\sqrt{x} \cdot 4\sqrt{x^3}}
\]
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))
```

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

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

Montasque might use foie gras instead...
On balance?

or maybe lighter is better?
Designing with LCs

input >>> [ _______ for x in \[0, 1, 2, 3\] range(4) ]

output [0, 14, 28, 42]

input >>> [ _____________ for c in 'igetthis' ]

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

... and what is the sum of this output?!
Using LCs

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

fun1([7, 8, 9])
```

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

fun2(['twelve'])
```

`Short and sweet!`

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)
```

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

But one-liners are my specialty...

What fun are these!
"One-line" LCs

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

possible, but **not** recommendable!

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

*I only ever get one line – who are the writers around here...?*

Maybe too terse!
def vwl(s):
    LC = [ 1 for c in s ]
    return sum( LC )

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

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

def lotto(Y,W):
    LC = [1 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):
    return LC
```

Extra!
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 = [x for x in Y & W]
    return sum(LC)

def ndivs(N):
    LC = [x for x in N]
    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

Y are your #s  
W are the winning #s

inputs:  
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]
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 = [
    ]
    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
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!

\[ y = 2x \]

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

**scaledfracs** \((low, hi, N)\)

**f_of_fracs** \((f, low, hi, N)\)

**integrate** \((f, low, hi, N)\)

only a few lines...

**Where are the LCs?**

Areas of 8 rectangles
Building blocks == CS!

Good luck with hw#2...

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

Areas of 8 rectangles