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)
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
```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)
```

<table>
<thead>
<tr>
<th>levels</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>t = 200, levels = 4</td>
</tr>
<tr>
<td>3</td>
<td>t = 100, levels = 3</td>
</tr>
<tr>
<td>2</td>
<td>t = 50, levels = 2</td>
</tr>
<tr>
<td>1</td>
<td>t = 25, levels = 1</td>
</tr>
</tbody>
</table>

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

The numbers show the value of N for each call...
A photo of a photo of sweet victory!
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...*

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

We're computationally complete!

What's next?

putting Python to work...

& adding **building-blocks**

---

**Hw 3** – *due Monday evening ~ usual time*

- pr0 reading – *Watson!*
- pr1 lab – Turtle!
- pr2, pr3 - Python probs...
- pr4 – extra-credit turtle...!
Recursive Art ~ hw3pr4

Submit things that work ... septagonal confetti

dramatic spiral!
Recursive Art Ex. Cr. ~ hw3pr4

Raj!

Frederick!

Madeline!
Recursive Art ~ hw3pr4

Submit things that work ... 

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

"Cyriak's pet snake..."
this week's hw3pr0

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 procedural or sequential
Data

[13,14,15]
[3,4,5,6,7,8,9]

Functions

```python
def sum(L):
    if L == []: return 0
    else:
        return L[0] + sum(L[1:])
```

... 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
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))
sum and range

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

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

- Ben. L.'14

50 x 101 = 5050
\[ \text{sum} \ \text{and} \ \text{list} \ \text{range} \]

>>> \text{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...
Looks sort of scruffy for a 7-year old...!

\[
\text{sum and list range}
\]

\[
>>> \text{sum(list(range(1,101)))}
\]

5050

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

50+51

101

101

50 101's == 5050
Data

Functions

\[ \text{sq}() \]

\[ [64, 81, 100] \]

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

...together
Various approaches...

many options for mapping a function onto a list:
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

>>> [ 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 takes on each value

and $2 \times x$ is output for each one
List Comprehensions

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

```
[0, 20, 40]
```

result

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

```
[0, 21, 42]
```

result

>>> [ s[1] for s in ["hi", "5Cs!" ] ]

```
["i", "5"]
```

result
### Quiz!

A range of list comprehensions...

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>n**2 for n in range(0,5)</code></td>
<td><code>[0, 1, 4, 9, 16]</code></td>
</tr>
<tr>
<td><code>42 for z in [0,1,2]</code></td>
<td><code>[42, 42, 42]</code></td>
</tr>
<tr>
<td><code>s[1::2] for s in ['aces', '451!']</code></td>
<td><code>['cs', 's!']</code></td>
</tr>
<tr>
<td><code>-7*b for b in range(-6,6) if abs(b)&gt;4</code></td>
<td><code>[42, 35, -35]</code></td>
</tr>
<tr>
<td><code>a*(a-1) for a in range(8) if a%2==1</code></td>
<td><code>[0, 6, 20, 42]</code></td>
</tr>
</tbody>
</table>

Name(s): _____________________________
Quiz! A range of list comprehensions...
Write Python's result for each L.C.:

```python
[n**2 for n in list(range(0,5))]
[0,1,4,9,16]
```

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

```python
[z for z in [0,1,2]]
[0,1,2]
```

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

```python
[-7*b for b in list(range(-6,6)) if abs(b)>4 ]
[-6, -5, 5 ]
```

```python
[a*(a-1) for a in list(range(8)) if a%2==1 ]
[ 0, 6, 20, 42 ]
```

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

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

\[
[ n**2 \text{ for } n \text{ in } \text{list} \text{range}(0,5) ]
\]

\[ [0,1,4,9,16] \]

\[
[ 42 \text{ for } z \text{ in } [0,1,2] ]
\]

\[ [42,42,42] \]

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

\[ [-6, -5, 5] \]

\[
[ a*(a-1) \text{ for } a \text{ in } \text{list} \text{range}(8) \text{ if } a\%2==1 ]
\]

\[ [0, 6, 20, 42] \]
List Comprehensions?

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

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

Google maps?

Datafuncs?

FunLists!

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?

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

FunLists!

[python-committers] Transfer of power

Guido van Rossum | Thu, 12 Jul 2018 07:59:19 -0700

Now that PEP 572 is done, I don't ever want to have to fight so hard for a PEP and find that so many people despise my decisions.

I would like to remove myself entirely from the decision process. I'll still be there for a while as an ordinary core dev, and I'll still be available to mentor people -- possibly more available. But I'm basically giving myself a permanent vacation from being BDFL, and you all will be on your own.

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.
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 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 is 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

```python
input >>> [ x for x in list(range(4)) ]

output [ 0, 14, 28, 42]
```

```python
input >>> [ c == 'i' 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)
```

---

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)
```

But one-liners are my specialty...

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

What fun are these!
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...?'

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...

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

def lotto(Y,W):
    LC = [Y, W]
    return sum(LC)

def ndivs(N):
    LC = [
        return sum(LC)
    ]

def primesUpTo(P):
    return LC
```

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

**inputs:** $Y$ and $W$, two lists of "lottery" numbers (ints)
**output:** the # of matches between $Y$ & $W$
**example:** $\text{lotto([5,7,42,47], [3,5,7,44,47])} = 3$

**input:** $N$, an int $\geq 2$
**output:** the # of positive divisors of $N$
**example:** $\text{numdivs(12)} = 6$ (1,2,3,4,6,12)

**input:** $P$, an int $\geq 2$
**output:** the list of prime #s up to + incl. $P$
**example:** $\text{primesUpTo(12)} = [2,3,5,7,11]$
Write each of these functions using list comprehensions...

```python
def nodds(L):
    LC = [x for x in L if x % 2 != 0]
    return sum(LC)

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

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

def primesUpTo(P):
    return LC
```

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

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

input: \( N \), an int \( \geq 2 \)
output: the \# of positive divisors of \( N \)
example: \( \text{numdivs}(12) = 6 \) (1,2,3,4,6,12)

input: \( P \), an int \( \geq 2 \)
output: the list of prime \#s up to + incl. \( P \)
example: \( \text{primesUpTo}(12) = [2,3,5,7,11] \)
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 list(range(1,N+1)) if N%x==0 ]
    return sum(LC)

def primesUpTo(P):
    LC = [ x for x in list(range(2,P+1)) if ndivs(x)==2 ]
    return LC
```
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!
hw3pr3: areas from rectangles

Areas of 4 rectangles

Areas of 8 rectangles
hw3pr3: areas from rectangles

Area of N rectangles in the limit
hw3pr3: 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!

\[ y = 2x \]

\[
\text{scaledfracs}(\text{low}, \text{hi}, N)
\]

\[
\text{f_of_fracs}(f, \text{low}, \text{hi}, N)
\]

\[
\text{integrate}(f, \text{low}, \text{hi}, N)
\]

Where are the LCs?

only a few lines...

Areas of 8 rectangles
Building blocks == CS!

Good luck with hw#3...

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

Areas of 8 rectangles

(0,0)