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)
```

 levels = 4
 levels = 3
 levels = 2
 levels = 1

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

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

STACK

- t = 200, levels = 4
- t = 100, levels = 3
- t = 50, levels = 2
- t = 25, levels = 1

Bourton-on-the-water

CS 5: now recursing...

- Hw 3 – due Monday evening ~ usual time
  - pr0 reading – Watson!
  - pr1 lab – Turtle!
  - pr2, pr3 - Python probs...
  - pr4 – extra-credit turtle...!

We're computationally complete!

What's next?

- putting Python to work...
- & adding building-blocks

Or recursing, depending on your feelings about recursion!
Recursive Art ~ hw3pr4

Submit things that work ...

septagonal confetti

dramatic spiral

Recursive Art Ex. Cr. ~ hw3pr4

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 vs. procedural or sequential

---

**Data**

**Functions**

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

... and their compositions

---

**sum**

**range**

```
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:])
```

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

I'd bet you have a 50/50 chance on this one... Ben L '14
Data Functions

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

[8, 9, 10]

[64, 81, 100]

[ sq(x) for x in [8,9,10] ]

...together

Various approaches...

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

List Comprehensions

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

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

List Comprehensions

expression iteration condition

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

result

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

result

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

result
Quiz!  A range of list comprehensions...
Write Python’s result for each L.C.

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

\[ 42 \text{ for } z \text{ in } \text{range}(0,2) \]

\[ s[1::2] \text{ for } s \text{ in } ['aces','451!'] \]

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

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

Syntax ?!

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

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

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, known as “Guido” among his peers) is a Dutch computer programmer who is best known as the author of the Python programming language. He is a member of the Python community. Van Rossum has been the de facto leader of the Python community since 1990, a position he still holds as of 2008. His contributions have been numerous and varied, including improving the technical aspects of the language and making decisions where necessary. He is currently employed by Google, where he spends half his time developing the Python language.

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 a spokesalien for this class!

a (frustrated!) rendering of an unfamiliar math problem

a.k.a.  Run it a "zillion" times!
Zillion-times testing!

# 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))

On balance? or maybe lighter is better?

Designing with LCs

```python
input >>> [ _____ for x in range(4) ]
output [0, 14, 28, 42]
```

```python
input >>> [ __________ for c in 'igetthis' ]
output [True, False, False, False, False, False, True, False]
```

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)
```

Let's see how the various funs are working...

def letScore(c):
    score = 1 if c == 't' else 0
    return score
"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])
```

Write each of these functions using list comprehensions...

Go!

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

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

```python
def lotto(Y,W):
    LC = [1 for x in L]
    return sum(LC)
```

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`

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

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

```python
def primesUpTo(P):
    return LC
```

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

Extra!

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

# of vowels 'sequoia'

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

# of times e is in L [3,4,2,5,7,42]

```python
if y == 2x
(0,0)
(2.5,5)
(5,10)
(7.5,15)
```

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

# of vowels 'sequoia'

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

# of times e is in L [3,4,2,5,7,42]

```python
if y == 2x
(0,0)
(2.5,5)
(5,10)
(7.5,15)
```
Building blocks == CS!

\[ y = 2x \]

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

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

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

only a few lines...

\textit{Where are the LCs?}
**Quiz!** A `range` of list comprehensions...

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

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

2. `[42 for z in [0,1,2]]`
   - `[42, 42, 42]`

3. `[s[1::2] for s in ['aces','451!']]`
   - `['c', 'e']`

4. `[-7*b for b in list(range(-6,6)) if abs(b)>4]`
   - `[-42, -14, 21]`

5. `[a*(a-1) for a in list(range(8)) if a%2==1]`
   - `[-6, -5, -6, -8, -9, -10]`