Thinking *loopily* and *cumulatively* for a while sounds natural to me!

**Today Loops** have arrived...

**This week + next:** putting loops to good use:
Cyriak: *conceptually disruptive* recursion...
Cyriak: *conceptually disruptive recursion*...

Other differences in Europe... !?
**A random aside...**

```python
import random
from random import *
```

allows use of `dir(random)` and `help(random)`

all random functions are now available!

```python
choice(L)  # chooses 1 element from the sequence L
choice('mudd')  # ... or 1 character from a string
choice(['cmc','scripps','pitzer','pomona'])
```

```python
list(range(1,5)) → [1,2,3,4]
```

How would you get a random integer from 0 to 99 inclusive?

```python
uniform(low, hi)  # chooses a random float from low to hi
```

```python
>>> uniform(41.9, 42.1)
42.08010107642389
```

`floats` have **16** places of precision  

*Aargh – so close!*
from random import *

def guess( hidden ):
    """ tries to guess our "hidden" # """
    compguess = choice( list(range(100)) )

    if compguess == hidden:  # at last!
        print('I got it!')

    else:
        guess( hidden )
from random import *
import time

def guess( hidden ):
    """ guessing game ""
    compguess = choice( list(range(100)) )

    # print('I choose', compguess)
    # time.sleep(0.05)

    if compguess == hidden:  # at last!
        # print('I got it!')
        return 1
    else:
        return 1 + guess( hidden )
Extra!

Quiz

A few random thoughts...

```
from random import *
choice( [1,2,3,2] )  # What are the chances this returns a 2?
choice( list(range(1,5)) + [4,2,4,2] )  # What are the chances of this returning a 4?
choice( '1,2,3,4' )  # What's the most likely return value here?
choice( [ '1,2,3,4' ] )  # What's the most likely return value here?
choice( [ '1,2,3,4' ] )  # What's the most likely return value here?
choice( list(range(5)) )  # Is this more likely to be even or odd (or same)?
uniform( -20.5, 0.5 )  # What're the chances of this being > 0? Extra!
choice(0,1,2,3,4)
choice([list(range(5))])
choice[list(range(5))]
```

Which two of these 3 are syntax errors?

Also, what does the third one – the one syntactically correct – actually do?
from random import *
choice([1,2,3,2])  # What are the chances this returns a 2?  2/4 or 50%
choice(list(range(1,5))+[4,2,4,2])  # What are the chances of this returning a 4?  3/8
choice('1,2,3,4')  # What's the most likely return value here?  ', '
choice(['1,2,3,4'])  # What's the most likely return value here?  '1,2,3,4'  1/1
choice('[1,2,3,4]')  # What's the most likely return value here?  ', '  3/9
choice(list(range(5)))  # Is this more likely to be even or odd (or same)? even  3/5
uniform(-20.5, 0.5)  # What're the chances of this being > 0?  1/42
choice(0,1,2,3,4)  # syntax error: needs [...] or '...'
choice([list(range(5))])  # correct: always returns [0,1,2,3,4]
choice[list(range(5))]  # syntax error: needs choice(...)
The two *Monte Carlos* and their denizens...

Monte Carlo casino, *Monaco*

Insights via *random trials*

Monte Carlo methods, *Math/CS*
The two *Monte Carlo* and their denizens...

Monte Carlo casino, *Monaco*

Monte Carlo methods, *Math/CS*
Monte Carlo in action

How many doubles will you get in \( N \) rolls of 2 dice?

\[ \text{def} \ \text{countDoubles( N )}: \]

\[ """ \ \text{input: the \# of dice rolls to make} \]

\[ \text{output: the \# of doubles seen } """ \]

\[ \text{if } N == 0: \]

\[ \text{return } 0 \quad \# \ \text{zero rolls, zero doubles...} \]

\[ \text{else:} \]

\[ \text{d1 = choice( [1,2,3,4,5,6] )} \]

\[ \text{d2 = choice( list(range(1,7)) )} \]

\[ \text{two dice from} \]

\[ \text{1-6 inclusive} \]

\[ \text{if } \text{d1} \neq \text{d2}: \]

\[ \text{return } 0+\text{countDoubles( N-1 )} \quad \# \ \text{don't count it} \]

\[ \text{else:} \]

\[ \text{return } 1+\text{countDoubles( N-1 )} \quad \# \ \text{COUNT IT!} \]

\[ \text{where and how is the check for doubles?} \]
Another Monty...?
Let's make a deal...

inspiring the Monty Hall paradox
Monte Carlo Monty Hall

Suppose you always switch to the other door... What are the chances that you will win the prize?

Let's play (randomly) 300 times and see!
Monte Carlo Monty Hall

```python
def MCMH( init, sors, N ):
    """ plays the "Let's make a deal" game N times
    returns the number of times you win the *Spam!* ""
    if N == 0: return 0  # don't play, can't win
    przDoor = choice([1,2,3])  # where the spam (prize) is...
    if init == przDoor and sors == 'stay': result = 'Spam!'
    elif init == przDoor and sors == 'switch': result = 'pmfp.'
    elif init != przDoor and sors == 'switch': result = 'Spam!'
    else: result = 'pmfp.'
    print 'You get the', result
    if result == 'Spam!': return 1 + MCMH( init, sors, N-1 )
    else: return 0 + MCMH( init, sors, N-1 )
```

Your initial choice!

'switch' or 'stay'

number of times to play
Let's make a deal: XKCD's take...

... but what if you considered the goat the grand prize!?
An overworked 5C student (S) leaves for class after a "late-night" breakfast – or lunch. Each moment, they randomly stumble toward class (W) or the dorm (E).

Once the student arrives at the dorm or classroom, the trip is complete. The program should then print the total number of steps taken.

Write a program to model and analyze! this scenario...

```
rwpos(s, nsteps)
   take nsteps random steps starting at s
rwsteps(s, low, hi)
   take random steps starting at s until you reach either low or hi
```
An example closer to home

An overworked 5C student (s) leaves for class after a "late-night" breakfast – or lunch. Each moment, they randomly stumble toward class (W) or the dorm (E).

Once the student arrives at the dorm or classroom, the trip is complete. The program should then print the total number of steps taken.

Write a program to model and analyze! this scenario...

\textbf{rwpos}(s,n\text{steps}) \hspace{1cm} \textbf{rwsteps}(s,low,hi)

\begin{itemize}
\item take \text{\textit{nsteps}} random steps starting at \text{s}
\item take random steps starting at \text{s} until you reach either low or hi
\end{itemize}
for loops: examples...

```python
for x in [2,4,6,8]:
    print(x)

for y in [7]*6:
    print(y)

for c in 'down with loops!':
    print(c)

for i in range(42):
    print(i)
```

How could we get this loop to run 42 times?

There are a range of answers to this one...
Two design strategies:

- **Recursion** ~ self-similarity
  
  \[ \text{fac}(5) = 5 \times \text{fac}(4) \]

- **Iteration** ~ sequencing through loops
  
  \[ \text{fac}(5) = 5 \times 4 \times 3 \times 2 \times 1 \]
Python's for loop

```python
for x in [2,4,6,8]:
    print(x)
```

Looks like a list comprehension – but with no need for a list
for loops: 4 examples...

```python
for x in [2,4,6,8]:
    print(x)

for y in [7]*6:
    print(y)

for c in 'down with loops!':
    print(c)

for i in [6,9,12,15,18] * 5:
    print(i)
```

How could we get this loop to run 42 times?

There are a range of answers to this one...
for!

1. x is assigned each value from this sequence

```
for x in [2,4,6,8]:
    print('x is', x)
```

2. the BODY or BLOCK of the for loop runs with that x

```
print('Done!')
```

3. LOOP back to the top for EACH value in the list

4. Code AFTER the loop will not run until the loop is finished.

It's what the fox says: Duck!
That's why they're called *variables*

\[
\text{age} = 41 \\
\text{age} = \text{age} + 1
\]

The "old" value (41)

The "new" value (42) BEFORE

The "new" value (42) AFTER

Only in code can one's newer age be older than one's older age...!
That's why they're called **variables**

```
age = 41
age = age + 1
```

The "old" value (41)

The "new" value (42)

---

**Python shortcuts**

```
hwToGo = 3
hwToGo = hwToGo - 1

amoebas = 100000
amoebas = amoebas * 2

u235 = 1000000000000000;
u235 = u235 / 2
```

```
hwToGo -= 1
amoebas *= 2
u235 /= 2
```
four questions for for

for x in [1,2,3,4,5,6,7):

    print('x is', x)

avoid writing the whole list?
find the sum of the list?
showing partial sums?
factorial function?
def fac(N):
    result = 1
    for x in list(range(1,N+1)):
        result = result * x
    return result
def fac(N):
    result = 1
    for x in list(range(1, N+1)):
        result = result * x
    return result
Laddering for loops

```python
def fac(N):
    result = 1
    for x in list(range(1, N+1)):
        result = result * x
    return result
```

Warning: no one else uses this term…
These seem unexpected, but only at first... !?

**Quiz**

**What does the loop say?**

```python
result = 1
for x in [2, 5, 1, 4]:
    result *= x

print(result)
```

```python
x = 0
for i in list(range(4)):
    x += 10

print(x)
```

```python
L = ['golf', 'fore!', 'club', 'tee']
for i in list(range(len(L))):
    if i%2 == 1:
        print(L[i])
```

```python
S = 'time to think this over! '
result = ''
for i in list(range(len(S))):
    if S[i-1] == ' ':
        result += S[i]

print(result)
```

**Extra!** How could you change one character above to yield eoks! or another to yield etns! or another to yield eoks!
Quiz

What does the loop say?

result = 1
for x in [2,5,1,4]:
    result *= x
print result
Quiz

What does the loop say?

```python
x = 0
for i in list(range(4)):
    x += 10
print(x)
```

`0, 1, 2, 3`
Quiz

What does the loop say?

L = ['golf','fore!','club','tee']

for i in list(range(len(L))):
    if i%2 == 1:
        print L[i]
Quiz  What does the loop say?

S = 'time to think this over! '  

result = ''

for i in list(range(len(S))):
    if S[i-1] == ' ':
        result += S[i]

print result
Quiz

What does the loop say?

S = 'time to think this over!'

result = ''

for i in list(range(len(S))):
    if S[i-1] == ' ':
        result += S[i]

print result  # 'tttto'

Looks like a four-'t' 'to' to me!

Extra! How could you change one character above to yield mns or another to yield etns! or another to yield eoks!

<table>
<thead>
<tr>
<th>res.</th>
<th>S[i-1]</th>
<th>S[i]</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>'t'</td>
<td>' '</td>
<td>'t'</td>
<td>0</td>
</tr>
<tr>
<td>'t'</td>
<td>'t'</td>
<td>'i'</td>
<td>1</td>
</tr>
<tr>
<td>'i'</td>
<td>'m'</td>
<td>'m'</td>
<td>2</td>
</tr>
<tr>
<td>'m'</td>
<td>'e'</td>
<td>'e'</td>
<td>3</td>
</tr>
<tr>
<td>'e'</td>
<td>' '</td>
<td>'t'</td>
<td>4</td>
</tr>
<tr>
<td>'t'</td>
<td>' '</td>
<td>'t'</td>
<td>5</td>
</tr>
<tr>
<td>'t'</td>
<td>'t'</td>
<td>'o'</td>
<td>6</td>
</tr>
<tr>
<td>'t'</td>
<td>'o'</td>
<td>'o'</td>
<td>7</td>
</tr>
<tr>
<td>'t'</td>
<td>'o'</td>
<td>'t'</td>
<td>8</td>
</tr>
</tbody>
</table>
for: two types

L = [3, 15, 17, 7]

for x in L:
    print(x)

element-based loops
for: two types

\[ L = [3, 15, 17, 7] \]

- **element-based loops**
  ```python
  for x in L:
      print(x)
  ```

- **index-based loops**
  ```python
  for i in range(len(L)):
      print(L[i])
  ```
for: two types

L = \[3, 15, 17, 7\]

for i in range(len(L)):
    print(L[i])

index-based loops

for x in L:
    print(x)

element-based loops
simpler vs. flexibler

$L = [3, 15, 17, 7]\)

```
def sum(L):
    total = 0
    for x in L:
        total += x
    return total
```

*element*-based loops
simpler vs. flexibler

\[
L = [3, 15, 17, 7]
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
i & & & \\
\end{array}
\]

def sum(L):
    total = 0
    for x in L:
        total += x
    return total

def sum(L):
    total = 0
    for i in range(len(L))
        total += ____
    return total

*element*-based loops

*index*-based loops
At the top of a CS5 project file ...

```cpp
// Author: Matt Beaumont-Gay
// Purpose: To get me out of CS5...
//          ...no, really...
// Purpose: To create and maintain a list
//          of films and directors

/* Notes:
   * I haven't liked for-loops since the day I met them.
   * They bother me for some reason. Hence, no for-loops...
  */
```

... and it is possible to avoid them entirely
At the top of a CS5 project file...

// Author: Matt Beaumont
// Purpose: To get me out of CS5...
//          ...no, really...
// Purpose: To create and maintain a list
//          of films and directors

/* Notes:
  * I haven't liked for-loops
  * since the day I met them.
  * They bother me for some reason. Hence, no for-loops...
*/

Perspective on **for loops**

**Barbara Usher** (Google)

CGU
Perspective on for loops

Barbara Usher (Google)

MBG, now willing to use for-loops...

Perspective on for loops

At the top of a CS5 project file...

// Author:  Matt Beaumont
// Purpose: To get me out of CS5...
//          ...no, really...
// Purpose: To create and maintain a list
//          of films and directors

/* Notes:
* I haven't liked
  for-loops
  since the day I met them.
* They bother me for some reason. Hence, no for-loops...
*/

... and it is (temporarily) possible to avoid them entirely
What does this code do?

```python
print('It keeps on')
while 41+1 == 42:
    print('going and')
print('Phew! I\'m done!')
```

Extreme Looping

I'm whiling away my time with this one!
**Extreme Looping**

**Anatomy of a while**

```python
while 41+1 == 42:
    print('going and')
```

```python
print('It keeps on')
print('going and')
```

```python
print('Phew! I\'m done!')
```

This won't print until the while loop finishes - In this case, it *never* prints!

I'm whiling away my time with this one!
lots of different tests...

print('It keeps on')

while 42 == 42:
    print('going and')

print('Phew! I\'m done!')

I\'m whiling away my time with this one!
Extreme Looping

lots of different tests...

```python
print('It keeps on')
while True:
    print('going and')
print('Phew! I\'m done!')
```

I'm whiling away my time with this one!
import random
escape = 0

while escape != 42:
    print('Help! Let me out!')
    escape = random.choice([41, 42, 43])
    print('At last!')

how could we count the number of loops we run?
how could we make it easier/harder to escape?
counter = 0

while escape != 42:
    escape = random.choice(range(0,10000))
    counter += 1
    print counter, ":",
    print 'Help! Let me out!'
How long til a repeat?

import random

def random_date():
    """ returns a random date (a string) in month/day form (no leap year...) """
    DiM = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    month = random.choice( range(1,12+1) )
    day = random.choice( range(1,DiM[month]+1) )
    randate = str(month) + "/" + str(day)
    return randate

What's DiM? why the +1's? Programs ~ executable thought...
def bday_list():
    """ birthday-paradox example!
    returns the list of bdays up to and including the first repeat...
    """

    LoBs = []  # List of BDays

    while all_unique(LoBs) == True:
        bday = random_date()
        LoBs += [bday]

    return LoBs

How long til a repeat?

birthday paradox!
How long til a **repeat**?

```python
random_date()    # is it today?

bday_list()     # find the repeat (w/end)

len( bday_list() )  # how many til a repeat?

LoLS = [ len( bday_list() ) for i in range(10**5) ]

min(LoLS)        # what will this be?
max(LoLS)        # ballpark?
sum(LoLS)/len(LoLS)  # average?
```

*birthday paradox!*
How long til a \textit{repeat}?

\textit{Sooner than you might think...}

\begin{quote}
\textit{Understanding the Birthday Paradox}

by Kalid Azad · 186 comments Tweet 219

23 people. In a room of just 23 people there’s a 50-50 chance of two people having the same birthday. In a room of 75 there’s a 99.9% chance of two people matching.

Put down the calculator and pitchfork, I don’t speak heresy. The birthday paradox is strange, counter-intuitive, and \textit{completely true}. It’s only a “paradox” because our brains can’t handle the compounding power of exponents. We expect probabilities to be linear and only consider the scenarios we’re involved in (both faulty assumptions, by the way).
\end{quote}

http://betterexplained.com/articles/understanding-the-birthday-paradox/
Try these...

What do these two loops return?

```
Let WORD = 'forty-two'

def count( WORD ):
    n = 0
    for c in WORD:
        if c not in 'aeiou':
            n += 1
    return n

def mystery( n ):
    while n != 1:
        if n%2 == 0:
            n = n/2
        else:
            return False
    return True

Let n = 12  Let n = 8

Challenge: for what inputs n does mystery return True?
```
Let $\text{WORD} = \text{\textit{forty-two}}$

```python
def count( WORD ):
    n = 0
    for c in WORD:
        if c not in 'aeiou':
            n += 1
    return n
```

Let $n = 12$ (then 8)

```python
def mystery( n ):
    while n != 1:
        if n%2 == 0:
            n = n/2
        else:
            return False
    return True
```

**Challenge:** what are inputs for which `mystery` returns True?
XKCD's loop

Lab today: problems #1 and #2 of homework #3!

and this was before watches – or glasses...