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

**Today** *Loops* have arrived...

**This week + next:** putting loops to good use:
Coding in circles!

It's July!

Thinking *loopily* and *cumulatively* for a while

Chat Q'n of the day...

What's your birthday?
... and birthsign!

Extra: How popular is it to be born on your birthday?

Do we have any *same-birthdays*? *Same-signs*?
Coding in *circles*!

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

Today *Loops* have arrived...

*This week* ... where they've arrived *from*

*Hmmm*... Assembly Language?
Why Assembly Language?

Is assembly really a thing?

It’s only the foolish who never climb Mt. Fuji -- or who climb it again.
The instruction is `addn`

- **Argument 1**: register `r3` (1011)
- **Argument 2**: constant `4` (0100)

**Details**

- **Old value of Reg3**: 7
- **New value of Reg3**: 11

A Von Neumann machine

---

**Instruction Decoding Guide**
The instruction is addn.

Argument 1: r3

Argument 2: 4

The instruction addn is instr #2 (#10 in binary).

The instruction decoding guide is not visible in the image.
Hmmm-thinking *in Python*

**Loops in Python**

```python
def fac(x):
    result = 1
    while x != 0:
        result *= x
        x -= 1
    return result
```

**Jumps in Hmmm**

```
00  read  r1
01  setn  r13  1
02  jeqzn  r1  6
03  mul  r13  r13  r1
04  addn  r1  -1
05  jumpn  02
06  write  r13
07  halt
```

It figures a Python would prefer looping to jumping!
Loops in Python

```python
def fac(x):
    result = 1
    while x != 0:
        result *= x
        x -= 1
    return result
```

Jumps in Hmmm

```
00 read r1
01 setn r13 1
02 jeqzn r1 6
03 mul r13 r13 r1
04 addn r1 -1
05 jumpn 02
06 write r13
07 halt
```

It figures a Python would prefer looping to jumping!
Hmmm-thinking in Python

Loops in Python

```python
def fac(x):
    result = 1
    while x != 0:
        result *= x
        x -= 1
    return result
```

Jumps in Hmmm

```plaintext
00 read r1
01 setn r13 1
02 jeqzn r1 6
03 mul r13 r13 r1
04 addn r1 -1
05 jumpn 02
06 write r13
07 halt
```

It figures a Python would prefer looping to jumping!
Hmmm-thinking in Python

Loops in Python

```python
def fac(x):
    result = 1
    while x != 0:
        result *= x
        x -= 1
    return result
```

Jumps in Hmmm

```assembly
00 read r1
01 setn r13 1
02 jeqzn r1 6
03 mul r13 r13 r1
04 addn r1 -1
05 jumpn 02
06 write r13
07 halt
```

It figures a Python would prefer looping to jumping!
Hmmm-thinking in Python

Loops in Python

```python
def fac(x):
    result = 1
    while x != 0:
        result *= x
        x -= 1
    return result
```

Jumps in Hmmm

```
00  read r1
01  setn r13 1
02  jeqzn r1 6
03  mul r13 r13 r1
04  addn r1 -1
05  jumpn 02
06  write r13
07  halt
```

It figures a Python would prefer looping to jumping!
Hmmm-thinking in Python

Loops in Python

```python
def fac(x):
    result = 1
    while x != 0:
        result *= x
        x -= 1
    return result
```

Jumps in Hmmm

```
00  read  r1
01  setn  r13  1
02  jeqzn  r1  6
03  mul  r13  r13  r1
04  addn  r1  -1
05  jumpn  02
06  write  r13
07  halt
```

It figures a Python would prefer looping to jumping!
## Hmmm-thinking in Python

Loops in Python

```python
def fac(x):
    result = 1
    while x != 0:
        result *= x
        x -= 1
    return result
```

Jumps in Hmmm

```plaintext
00  read  r1
01  setn  r13  1
02  jeqzn  r1  6
03  mul  r13  r13  r1
04  addn  r1  -1
05  jumpn  02
06  write  r13
07  halt
```

It figures a Python would prefer looping to jumping!
Loops in Python

```python
def fac(x):
    result = 1
    while x != 0:
        result *= x
        x -= 1
    return result
```

We get the advantages of loop-expressions AND self-contained functions.

All the advantages of Hmm? I'm sold!
Recursive Hmmm
factorial, hw6pr4

00 read r1
01 setn r15 42
02 call r14 5
03 jump 21
04 nop
05 jnezn r1 8
06 setn r13 1
07 jumpr r14
08 storer r1
09 addn r15 1
10 storer r14
11 addn r15 1
12 addn r1 -1
13 call r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jumpr r14
20 nop
21 write r13
22 halt

Loopy Hmmm factorial, similar to hw6pr2 and pr3

00 read r1
01 setn r13 1
02 jeqzn r1 6
03 mul r13 r13 r1
04 addn r1 -1
05 jumpn 02
06 write r13
07 halt

Loops!

Functional
programming

Iterative
programming

Hmmm... I think I'll take Python!
Iterative design in Python

for x in [40, 41, 42]:
    print(x)

while x > 0:
    print(x)
    x -= 1

variables vary a lot!

But we change it as we go...

the initial value is often not the one we want in the end
for loops: four examples...

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

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

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

def i in print(i)
```

How could we get this loop to run 42 times?

There are a range of answers to this one...
for loops: four examples...

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

1. **x** is assigned each value from this sequence.
2. The BODY or BLOCK of the for loop runs with that **x**.
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**

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

The "old" value (41)

The "new" value (42)

Echoes from Hmmm: \[\text{05 addn r1 1}\]

Only in code can one's newer age be older than one's older age...!

Truth-in-powerpoint disclosure: *all of this will soon be true* – well, in base 13
That's why they're called **variables**

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

The "old" value (41)

The "new" value (42)

Python shortcuts

```python

hwToGo = 7
hwToGo = hwToGo - 1

amoebas = 21000000
amoebas = amoebas * 2

u235 = 84000000000000000;
u235 = u235 / 2

hwToGo -= 1
amoebas *= 2
u235 /= 2
```

Only in code can one's newer age be older than one's older age...!
four questions for for

```python
for x in range(1,8):
    print('x is', x)
```
four questions for for

for x in range(1,8):

[1,2,3,4,5,6,7]

print('x is', x)

Shortcuts!

- tab & shift-tab
- control-/

what list is this!?
find the sum of the list?
printing partial sums?
factorial function?
def fac(N):
    result = 1
    for x in range(1, N+1):
        result = result * x
    return result

Hey!? This is *not* the right answer... YET

thought experiments w/ return
for-loop "laddering"

result = 1

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

print(result)
These answers seem unexpected, but only at first... !?
Quiz

What does the loop say?

result = 1

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

print(result)

result

1

2

2

5

10

10

1

4

40

40
Quiz  What does the loop say?

\[ x = 0 \]
\[ \text{for } i \text{ in } \text{range}(4): \]
\[ x += 10 \]
\[ \text{print}(x) \]

It's ok not to use the loop variable!

\[ [0,1,2,3] \]
Quiz

What does the loop say?

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

$\text{for } i \text{ in range(len(L))}:
\text{if } i \% 2 == 1:
\text{print}(L[i])$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{L[i]}$</td>
<td>$i % 2$</td>
<td>$i$</td>
</tr>
<tr>
<td>'golf'</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>'fore!'</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>'club'</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>'tee'</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Quiz

What does the loop say?

S = 'time to think this over!'

result = ''

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

print(result)
Quiz

What does the loop say?

\[ S = \text{'time to think this over! ' } \]

\[ \text{result } = \text{''} \]

\[ \text{for i in list(range(len(S))):} \]
\[ \quad \text{if S[i-1] == ' ':} \]
\[ \quad \text{result } += \text{S[i]} \]

\[ \text{print(result)} \]

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

Extra! How could you change one character above to yield mns or another to yield etnsr or another to yield eoks!
for: two types

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

for \( x \) in \( L \):
    print(\( x \))

element-based loops

Elements vs Indexes

Indices
for: two types

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

for \( i \) in \text{range}(\text{len}(L)):
    print(L[i])

\textit{index}-based loops

for \( x \) in L:
    print(\( x \))

\textit{element}-based loops
for: two types

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

\[
\begin{align*}
\text{for } i \text{ in range(len}(L)):\quad & \text{print}(L[i]) \\
\text{for } x \text{ in } L:\quad & \text{print}(x)
\end{align*}
\]

index-based loops

printing is NOT unusually common in loops – but it is good for debugging!

element-based loops
simpler vs. flexibler

L = [3, 15, 17, 7]

```
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 += L[i]
    return total
```

*element*-based loops

*index*-based loops
**simpler vs. flexibler**

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

\[ L = [x_0, x_1, x_2, x_3] \]

**Elements vs Indexes**

**element**-based loops

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

**index**-based loops

```python
def sum(L):
    total = 0
    for i in range(len(L)):
        total += L[i]
    return total
```
for perspective

At the top of a CS5 project file ...

```
// 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, guest-presenting
Perspective on for loops

Barbara Usher (Google)

CGU, guest-presenting

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

... and it is (temporarily) possible to avoid them entirely
Extreme Looping

What does this loop do?

```
print('It keeps on')

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

print('Phew! I\'m done!')
```
**Extreme Looping**

**Anatomy of a while loop**

```python
print('It keeps on')
while 41 + 1 == 42:
    print('going and')
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!
Extreme Looping

lots of different tests...

```python
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...

```
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 many til \textit{escape}?

\texttt{letmeout()} \hspace{1cm} \# \textit{how many tries to escape?}

\texttt{LC = [ letmeout() for i in range(100) ]}

\textit{List Comprehensions} \hspace{1cm} "maps"
How many til *escape*?

`letmeout()`  # how many tries to escape?

```
LC = [ letmeout() for i in range(100) ]
```

\[ \text{min}(LC) \]  # ____ what might this be?
\[ \frac{\text{sum}(LC)}{\text{len}(LC)} \]  # ____ the average?
\[ \text{max}(LC) \]  # ____ "ballpark" this...

*List Comprehensions*  "*maps*"
"Birthday Room Experiment"
"Birthday Room Experiment"

rand_date()

birthday_list()

design ideas?
what's easy?
what's tricky?
why?
How long til a **repeat**?

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

birthday paradox!
"Birthday Room Experiment"

```
rand_date()
bday_list()
```

Experiment!

How?
How many til a **repeat**?

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

# gather til a repeat
bday_list()

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

```python
LC = [ len(bday_list()) for i in range(100) ]
```

```python
# ____ what might this be?
min(LC)

# ____ the average?
sum(LC)/len(LC)

# ____ "ballpark" this...
max(LC)
```

*List Comprehensions*  
"maps"

---

*birthday "paradox"*

Let's try a more representative experiment...  
100,000 10,000

十万
How long til a repeat?

Sooner than you might think...

Understanding the Birthday Paradox

by Kalid Azad - 186 comments

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 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).

http://betterexplained.com/articles/understanding-the-birthday-paradox/
Data Functions

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

...together

\[
[64, 81, 100]
\]

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

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

\[ [64, 81, 100] \]

List Comprehensions...together
Various approaches...

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

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

In [14]: run map_test

In [15]: L1
Out[15]: [0, 2, 4, 6, 8, 10]

In [16]: L2
Out[16]: [0, 2, 4, 6, 8, 10]

In [17]: L3
Out[17]: [0, 2, 4, 6, 8, 10]

In [18]:
List Comprehensions

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

Result:

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

>>> [ 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 \[ 2*x \] is output for each one
List Comprehensions

Expression: $10 \times x$
Iteration: $x$ in [0, 1, 2, 3, 4, 5]
Condition: $x \% 2 == 0$

Result:

Expression: $y \times 21$
Iteration: $y$ in list(range(0, 3))

Result:

Expression: $s[1]$
Iteration: $s$ in ['hi', '5Cs!']

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><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>['c', 'e', 's', '4', '5', '1']</code></td>
</tr>
<tr>
<td><code>[-7*b for b in range(-6,6) if abs(b)&gt;4]</code></td>
<td><code>[-28, -21, -14, -7, 0, 7, 14]</code></td>
</tr>
<tr>
<td><code>[a*(a-1) for a in range(8) if a%2==1]</code></td>
<td><code>[1, 3, 5, 7]</code></td>
</tr>
</tbody>
</table>
Quiz! A range of list comprehensions...
Write Python's result for each L.C.:

1. \[ n \times n \text{ for } n \text{ in } \text{list range}(0, 5) \]  
   \[ [0, 1, 4, 9, 16] \]

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

3. \[ 0, 1, 2, 3, 4 \]  
   \[ [0, 1, 2] \]

4. \[ 42 \text{ for } z \text{ in } [0, 1, 2] \]  
   \[ [42, 42, 42] \]

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

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

7. \[ -7 \times b \text{ for } b \text{ in } \text{list range}(-6, 6) \text{ if } \text{abs}(b) > 4 \]  
   \[ [-6, -5, 5] \]

8. \[ a \times (a-1) \text{ for } a \text{ in } \text{list range}(8) \text{ if } a \% 2 == 1 \]  
   \[ [0, 6, 20, 42] \]

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

Got it! But what about that name?
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?

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!

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^1 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 ~ *is CS's key resource*

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

\[
\frac{\frac{2a^7}{\sqrt[3]{x}}}{\frac{\sqrt[3]{y^2}}{3^2}}
\]

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

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

Where'd the change happen?
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!
# 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))
# 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))
On balance?

or maybe lighter is better?
import random

def rand_date():
    """ returns a random date (a string) in month/day form (no leap year...) """

    month = random.choice(range(1,12+1))
    day = random.choice(range(1,31+1))

    randomdate = str(month) + "/" + str(day)

    return randomdate
import random

def rand_date():
    """ returns a random date (a string) in month/day form (no leap year...) ""
    month = random.choice(range(1,12+1))
    day = random.choice(range(1,DiM[month]+1))
    randomdate = str(month) + "/" + str(day)
    return randomdate

DiM = [0,31,28,31,30,31,30,31,31,30,31,30,31,30,31]
import random

def rand_date():
    """ returns a random date (a string) in month/day form (no leap year...)
    """
    DiM = [0,31,28,31,30,31,30,31,31,30,31,30,31,30,31]
    month = random.choice( range(1,12+1) )
    day = random.choice( range(1,DiM[month]+1) )
    randomdate = str(month) + "/" + str(day)
    return randomdate
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*?
Pi from Pie (via Py-thon!)
Pi from Pie?

Pizza is the universal constant, after all...
Pi from Pie?

This couldn't be just a coincidence!
Estimating $\pi$ from pie?

What if we just throw darts at this picture?
Estimating $\pi$ from pie?

(1) Suppose you throw 100 darts at the square. (All of them hit the square)

(2) Suppose 80 of the 100 hit inside the circle.

(3) How could you estimate $\pi$ from these throws?

Hints

How big is a side of the square? its area?

How big is the radius of the circle? its area?

How do these help!??
Estimating $\pi$ from pie?

$$\frac{\pi}{4} = \frac{\text{circle area}}{\text{square area}}$$

$$\pi \sim \frac{4 \times \text{circle hits}}{\text{square hits}}$$

*Box*

*Pie*
Loops: **for** or **while**?

\[
\text{pi\_one}(e) \quad e == \text{how close to } \pi \text{ we need to get}
\]

\[
\text{pi\_two}(n) \quad n == \text{number of darts to throw}
\]

*Which function will use which kind of loop?*
Loops: `for` or `while`?

- `pi_one(e)`
  - `while` `e == how close to π we need to get`

- `pi_two(n)`
  - `for` `n == number of darts to throw`
Integrated Python
hw8pr4: \textit{areas from rectangles}

Areas of 4 rectangles

Areas of 8 rectangles
hw8pr4: areas from rectangles

Area of N rectangles in the limit
hw8pr4: 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...
Loop of life, XKCD's take:

See you in Lab 78!

and this, before watches – or glasses...