Coding in circles!
No lab today!

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

Today Loops have arrived...

This week + after break: putting loops to good use:

hw #8 due... Mon., Apr. 1
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 Argument 1 Argument 2 the same Reg3 old value of Reg3 = 7 new value of Reg3 = 11

E85 details the journey from circuits through assembly...
Loops in Python

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

It figures a Python would prefer looping to jumping!

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
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!
Hmmm-thinking in Python

Loops in Python

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

It figures a Python would prefer looping to jumping!

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
```
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

```
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 explicit looping AND self-contained functions.

All the advantages of Hmm? I'm sold!
Recursive Hmmmm factorial, hw7pr4

Loops!

Looping Hmmmm factorial, similar to hw7pr2 and pr3

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 load r14 r15
16 addn r15 -1
17 load r1 r15
18 mul r13 r13 r1
19 jumpr r14
20 nop
21 write r13
22 halt

Functional programming

Iterative programming

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

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

```python
x = 42
while x > 0:
    print(x)
    x -= 1
```

The initial value is often not the one we want in the end.

`variables vary a lot!`

But we change it as we go...

```
x = 41
x += 1
```

**Loops!**
for loops: four 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...

This slide is four for for!
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.
That's why they're called **variables**

\[ \text{age} = 41 \]

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

The "old" value (41)

The "new" value (42)

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

Echoes from Hmmm: 05 addn r1 1

Truth-in-powerpoint disclosure: *all of this was true last semester*—though not in base 10
That's why they're called **variables**

\[
\begin{align*}
\text{age} & = 41 \\
\text{age} & = \text{age} + 1 \\
\text{age} & += 1
\end{align*}
\]

The "old" value (41)
The "new" value (42)

---

**Python shortcuts**

\[
\begin{align*}
\text{hwToGo} & = 7 \\
\text{hwToGo} & -= 1 \\
\text{amoebas} & = 21000000 \\
\text{amoebas} & *= 2 \\
\text{u235} & = 84000000000000000000 \\
\text{u235} & /= 2
\end{align*}
\]

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

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

print('x is', x)
four questions for \texttt{for}

\begin{verbatim}
for x in [1,2,3,4,5,6,7]:
    print('x is', x)
\end{verbatim}

Shortcuts!

\begin{verbatim}
    tab & shift-tab
\end{verbatim}

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

Hey!? This is not the right answer... YET
for loop "laddering"

result = 1

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

print(result)

Warning: no one else uses this term...

meets up with Jacob's ladder
These seem unexpected, but only at first... !?
result = 1

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

print(result)
Quiz

What does the loop say?

\[ x = 0 \]

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

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

It's ok not to use the loop variable!
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])

<table>
<thead>
<tr>
<th>( L[i] )</th>
<th>( i % 2 )</th>
<th>( i )</th>
</tr>
</thead>
<tbody>
<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 list(range(len(S))):
    if $S[i-1] == ' '\n        result += $S[i]$

print(result)
Quiz

What does the loop say?

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

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

\[
\text{for } i \text{ in list(range(len(S))):}
\]

\[
\text{if } S[i-1] == ': \\
\text{result } += S[i]
\]

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

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

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

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

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

\textbf{Elements vs Indexes}

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

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

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

for \( x \) in \( L \):
    print(\( x \)) \( \text{element-based loops} \)
**for:** two types

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

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

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

printing is **NOT**
unusually common
in loops – but it is
good for debugging!
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 += ___
    return total
```

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

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

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

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

**Elements vs Indexes**

- **element-based loops**
  - `for x in L:
    total += x
  return total`

- **index-based loops**
  - `for i in range(len(L)):
    total += ____
  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
Perspective on for loops

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 (temporarily) possible to avoid them entirely
What does this code do?

```
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')
```

- The loop keeps on running as long as the test is `True`.

Other tests we could use here?

```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!
**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!
Extremee Looping

lots of different tests...

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

while True:
    print('going and')

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

during a "while True" loop

I'm whiling away my time with this one!
import random
escape = 0
guesses = []
while escape != 42:
    escape = random.choice([41, 42, 43])
    guesses += [escape]
    print('Help! Let me out!')
print('At last!')
print(guesses)

starting value, **not** the final or desired value!

test to see **if we keep looping**

watch out for infinite loops!

**after the loop ends**

how could we count the number of loops we run?
how could we accumulate a LIST of all the guesses?
counter = 0

while escape != 42:
    escape = random.choice(range(0, 10000))
    counter += 1
    print counter, "::",
    print 'Help! Let me out!'
"Birthday Room Experiment"

rand_date()

bday_list()
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

What's DiM?

why the +1? Programs ~ executable thought...
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]
    month = random.choice(range(1, 12 + 1))
    day = random.choice(range(1, DiM[month]))
    randomdate = str(month) + "/" + str(day)
    return randomdate
How long til a **repeat**?

```python
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]
    month = random.choice( range(1,12+1) )
    day = random.choice( range(1,DiM[month]+1) )
    randomdate = str(month) + "/" + str(day)
    return randomdate

Days in Month!
```

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 many til a *repeat*? birthday "paradox"

Name(s): ______________
Bday(s): ______________

```python
rand_date()  # is it today?
bday_list()  # gather til a repeat
len(bday_list())  # how many til a repeat?

LoLs = [ len(bday_list()) for i in range(100) ]
min(LoLs)  # _____ what might this be?
sum(LoLs)/len(LoLs)  # ____ the average?
max(LoLs)  # _____ "ballpark" this...

LoLs = [ len(bday_list()) for i in range(10000) ]
min(LoLs)  # _____ what might this be?
sum(LoLs)/len(LoLs)  # _____ and the average?
max(LoLs)  # _____ "ballpark" this?

L = [ rand_date() for i in range(330) ]
# _____ How many repeats would you "ballpark" to be in L?
```
How many til a repeat?

birthday "paradox"

Name(s): ______________
Bday(s): ______________

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

LoLs = [
    len( bday_list() ) for i in range(100)
]
min(LoLs)
sum(LoLs)/len(LoLs)
max(LoLs)

LoLs = [
    len( bday_list() ) for i in range(10000)
]
min(LoLs)
sum(LoLs)/len(LoLs)
max(LoLs)
```

Pass those in and up!

... I can't hear you!
How long til a *repeat*?

_Sooner than you might think..._

*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 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/
**Loop on!**

What do these two loops return?

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

**Finish this loop to find and return the min of a list, L**

L will be a non-empty list of numbers.

```python
def min(L):
    result = L[0]
    for x in L:
        if 
        return result
```

Extra: Write a loop so that this function returns True if the input n is prime and False otherwise

```python
def isPrime(n):
    n will be a positive integer >= 2
    n will be 12, 8, 6, 4, 3, 2, 1
    return True
```

**Challenge:** for what inputs n does mystery return True?

**Hint:** check all possible divisors to see if they "work"...

---

Let WORD = 'forty-two'

```
def count( WORD ):
    n = 0
    for c in WORD:
        if c not in 'aeiou':
            n += 1  # (for every consonant (non-vowel))
    return n
```

Let n = 12, 8

```
def mystery( n ):
    while n != 1:
        if n%2 == 0:
            n = n/2
        else:
            return False
    return True
```
Let WORD = 'forty-two'

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

```python
def min( L ):
    result = L[0]
    for x in L:
        if
        return result
```

Let n = 12 Let n = 8

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

```python
def isPrime( n ):
    for divisor in range(2, int(n**0.5) + 1):
        if n % divisor == 0:
            return False
    return True
```

**Challenge:** for what inputs n does mystery return True?

**Hint:** check all possible divisors to see if they "work"...

**We'll keep revisiting for and while loops over the upcoming weeks!**
and this was before watches – or glasses...