Thinking *loopily* and *cumulatively* sounds natural to me!

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

**This week + next:** putting loops to good use:

---

Why Assembly Language?

Is assembly really a thing?

It's only the foolish who never climb Mt. Fuji -- or who climb it again.

---

Hmmm-thinking *in Python*

**Loops in 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
```

---

E85 details the journey from circuits through assembly...
**Hmmm-thinking in Python**

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

---

**Iterative design in Python**

```python
for x in range(42):
    print(x)
```

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

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

---

**for loops: four examples...**

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

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

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

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

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

1. x is assigned each value from this sequence

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

That's why they're called **variables**

1. age = 41
2. age = age + 1
3. age += 1

Echoes from Hmmm: 05 addn rl 1

four questions for **for**

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

- avoid writing the whole list?
- find the sum of the list?
- showing partial sums?
- factorial function?

**fac** with **for**

def fac( N ):
    result = 1
    for x in list(range(1,N+1)):
        result = result * x
    return result
for loop "laddering"

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

result = 1
for x in [0, 1, 2, 3]:
x = 0
for i in list(range(4)):
x += 10
print(x)

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

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

for: two types

L = [3, 15, 17, 7]
for x in L:
    print(x)

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

Elements vs Indexes

index-based loops

simpler vs. flexibler
for perspective

At the top of a CS5 project file ...

```flexible
// 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

---

**Extreme** Looping

What does this code do?

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

I'm whiling away my time with this one!

---

Perspective on **for** loops

At the top of a CS5 project file ...

```flexible
// 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

---

**Extreme** Looping

Anatomy of a **while**

```flexible
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!
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 accumulate a LIST of all the guesses?

How long til a repeat?
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

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?
Loop on!

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

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

Challenge: for what inputs n does mystery return True?

Finish this loop to find and return the min of a list, L. L will be a non-empty list of numbers.

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

Extra: Write a loop so that this function returns the min of a list, L. L will be a non-empty list of numbers.

Let n = 12

def isPrime( n ):
    n will be a positive integer >= 2

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

Loop of life, XKCD's take:

SPEAK BLANKLY
AT SCREEN

ABSENTLY CHECK
SMALLER DEVICE
OPEN NEWS SITE

GET BORED ← READING

and this was before
watches – or glasses...
These seem unexpected, but only at first... ?!