CS5 Coding in *circles*!

Read Section 5.3

Thinking *loopily*

`for a while`

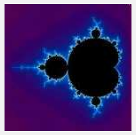
sounds natural to me!

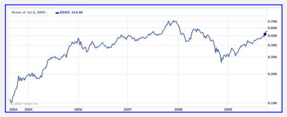
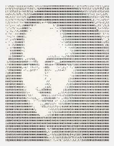
and *cumulatively*

`+=`

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

This week + next: putting loops to good use:



Hmmm-thinking *in Python*

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
```

Iterative design in Python

`for`

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

`jumpn, jeqzn, …`

`while`

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

variables **vary**

a lot!

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

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

`addn r1 1`

But we change it as we go…

`for` loops: examples…

This slide is four for `for`!

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


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


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


for i in
   print(i)
```

How could we get this loop to run 42 times?

There is a ***range*** of answers to this one…

`for`!

It's what the fox says: *Duck!*

1 x is assigned each value from this sequence

```
for x in [2, 4, 6, 8]:

    print('x is', x)

print('Done!')
```

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

This is the #1 for-loop error! (**what**? **why**?)

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

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

Anatomy?

Empty?

x unused?

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

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

```
age = 41

age = age + 1
```

The "old" value (41)

The "new" value (42)

```
age += 1
```

Python shortcuts

```
hwToGo = 7
hwToGo = hwToGo - 1
```

```
hwToGo -= 1
```

```
amoebas = 100000
amoebas = amoebas * 2
```

```
amoebas *= 2
```

```
u235 = 10000000000000
u235 = u235 / 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?

`fac` with `for`

```
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`: *two types*

```
      L[0] L[1] L[2] L[3]
L = [3,   15,  17,   7]
     0     1    2    3
              i
```

```
for i in range(len(L)):
  print(      )
```

Index-based loops

list

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

Element-based loops

Simpler vs. *More Flexible*

```
L = [3, 15, 17, 7]
     0   1   2  3
           i
```

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

Element-based loops

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

list

Index-based loops

Extreme Looping

Anatomy of a **`while`**

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


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

while loop body

The loop keeps on running as long as the test is **True**

Other tests we could use here?

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

I'm whiling away my time with this one!

Escape ?!

```
import random
escape = 0

while escape != 42:

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

print('At last!')
```

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 make it easier/harder to escape?
random.uniform!

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
```

Let n = 12   Let n = 8

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

    return True
```

12	8

Challenge: for what values of 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
```

What to check about x?

What to do?

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

n will be a positive integer >= 2

```
def isPrime(n):
```

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

Name(s): ____________________

Quiz

What does the loop say?

mild

res. | x

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

print(result)
```

mild-ish

x | i

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

print(x)
```

medium

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

i | i%2 | L[i]

spicy

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

print(result)
```

25

only ladder as needed...

These seem unexpected, but only *at first*… !?

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