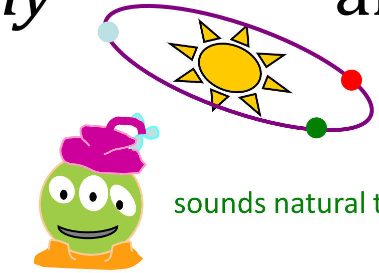
Coding in *circles*!

Thinking *loopily*

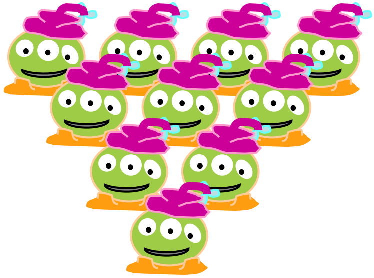
`for a while`



sounds natural to me!

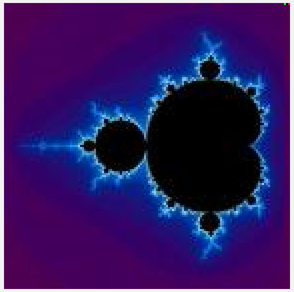
and *cumulatively*

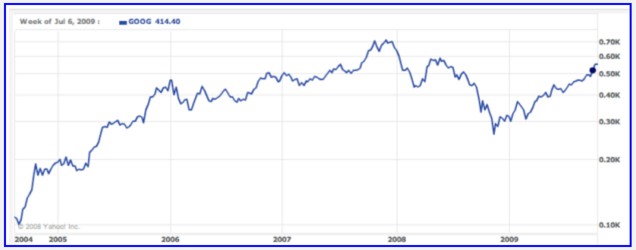
`+=`



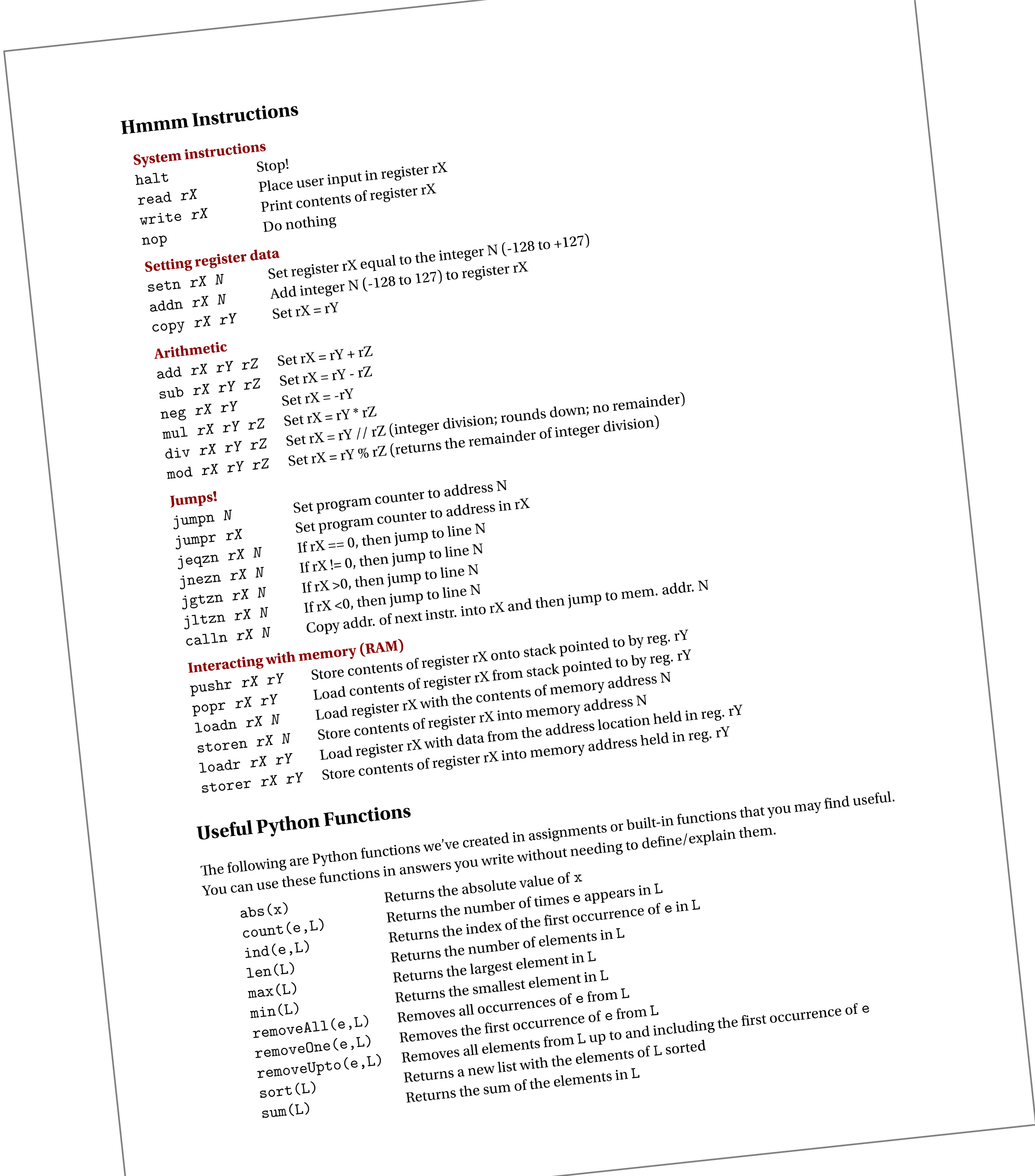
Today ***Loops*** have arrived…

Next week: putting loops to good use:





What we give you on the midterm…



Hmmm Instructions

System instructions

`halt`	Stop!
`read rX`	Place user input in register rX
`write rX`	Print contents of register rX
`nop`	Do nothing

Setting register data

`setn rX N`	Set register rX equal to the integer N (-128 to +127)
`addn rX N`	Add integer N (-128 to 127) to register rX
`copy rX rY`	Set rX = rY

Arithmetic

`add rX rY rZ`	Set rX = rY + rZ
`sub rX rY rZ`	Set rX = rY - rZ
`neg rX rY`	Set rX = -rY
`mul rX rY rZ`	Set rX = rY * rZ
`div rX rY rZ`	Set rX = rY // rZ (integer division; rounds down; no remainder)
`mod rX rY rZ`	Set rX = rY % rZ (returns the remainder of integer division)

Jumps!

`jumpn N`	Set program counter to address N
`jumpr rX`	Set program counter to address in rX
`jeqzn rX N`	If rX == 0, then jump to line N
`jnezn rX N`	If rX != 0, then jump to line N
`jgtzn rX N`	If rX >0, then jump to line N
`jltzn rX N`	If rX <0, then jump to line N
`calln rX N`	Copy addr. of next instr. into rX and then jump to mem. addr. N

Interacting with memory (RAM)

`pushr rX rY`	Store contents of register rX onto stack pointed to by reg. rY
`popr rX rY`	Load contents of register rX from stack pointed to by reg. rY
`loadn rX N`	Load register rX with the contents of memory address N
`storen rX N`	Store contents of register rX into memory address N
`loadr rX rY`	Load register rX with data from the address location held in reg. rY
`storer rX rY`	Store contents of register rX into memory address held in reg. rY

Useful Python Functions

The following are Python functions we've created in assignments or built-in functions that you may find useful. You can use these functions in answers you write without needing to define/explain them.

`abs(x)`	Returns the absolute value of x
`count(e,L)`	Returns the number of times e appears in L
`ind(e,L)`	Returns the index of the first occurrence of e in L
`len(L)`	Returns the number of elements in L
`max(L)`	Returns the largest element in L
`min(L)`	Returns the smallest element in L
`removeAll(e,L)`	Removes all occurrences of e from L
`removeOne(e,L)`	Removes the first occurrence of e from L
`removeUpto(e,L)`	Removes all elements from L up to and including the first occurrence of e
`sort(L)`	Returns a new list with the elements of L sorted
`sum(L)`	Returns the sum of the elements in L

Jumping for Conditionals

```
00   read r1
01   read r2
02   sub r3 r1 r2
03   jltzn r3 07
04   write r2
05   write r1
06   jumpn 09
07   write r1
08   write r2
09   halt
```

Hmmm — Assembly

```
100 INPUT X
110 INPUT Y

130 IF X < Y THEN GOTO 170
140 PRINT Y
150 PRINT X
160 GOTO 190
170 PRINT X
180 PRINT Y
190 STOP
```

BASIC — Dartmouth College, 1963

Jumping for Conditionals

```python
x = int(input())
y = int(input())

if not x < y:
    print(y)
    print(x)
else:
    print(x)
    print(y)
```

Python

```basic
100 INPUT X
110 INPUT Y

130 IF X < Y THEN GOTO 170
140 PRINT Y
150 PRINT X
160 GOTO 190
170 PRINT X
180 PRINT Y
190 STOP
```

BASIC — Dartmouth College, 1963

Factorial Revisited

```
00  read r1
01  setn r2 1
02  jeqzn r1 06
03  mul r2 r2 r1
04  addn r1 -1
05  jumpn 02
06  write r2
07  halt
```

Hmmm — Assembly

```
100 INPUT N
110 LET R = 1
120 IF N == 0 THEN GOTO 160
130 LET R = R * N
140 LET N = N - 1
150 GOTO 120
160 PRINT R
170 STOP
```

BASIC — Dartmouth College, 1963

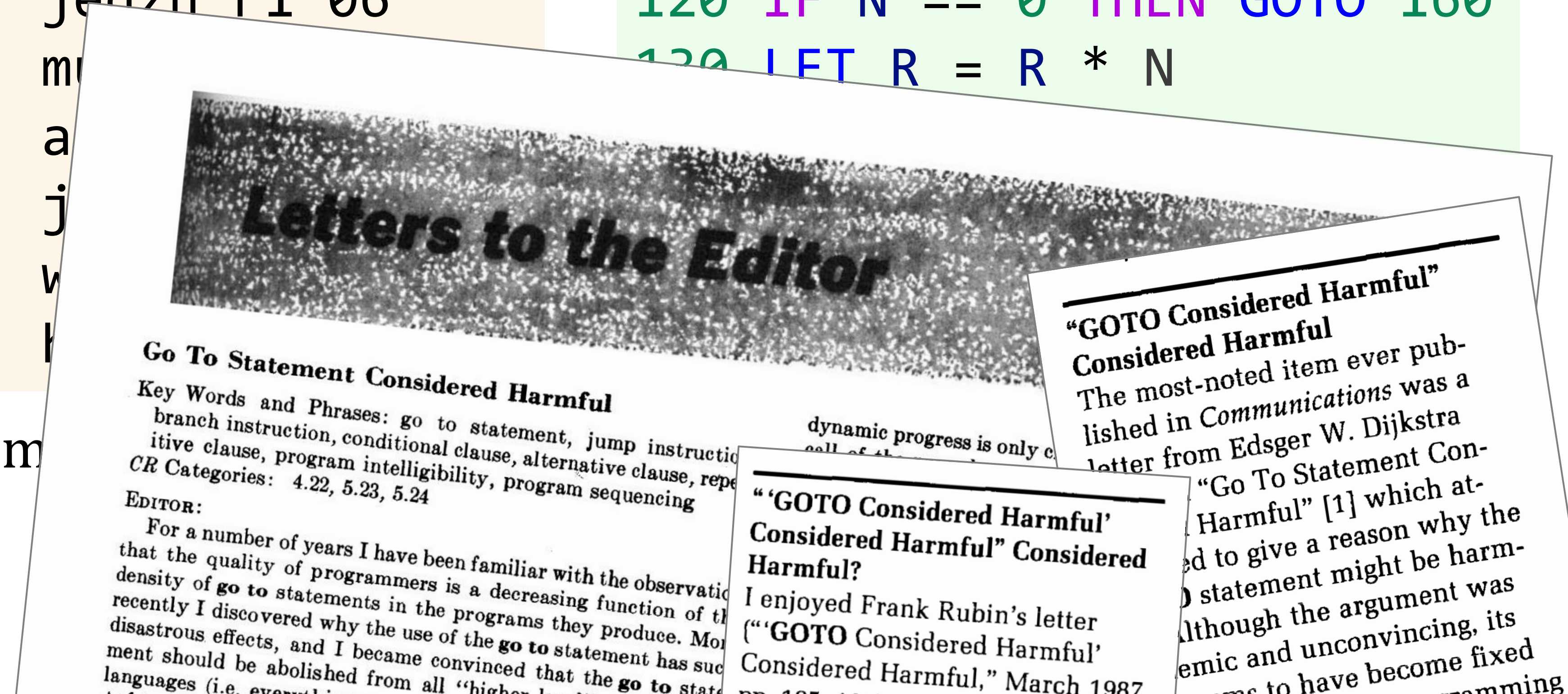
Factorial Revisited

The epic battle for
"Structured Programming"

```
00  read r1
01  setn r2 1
02  jeqzn r1 06
03  mu
04  a
05  j
06  w
07  h
```

```
100 INPUT N
110 LET R = 1
120 IF N == 0 THEN GOTO 160
130 LET R = R * N
```

Hm



Letters to the Editor

Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instructio
branch instruction, conditional clause, alternative clause, repe
itive clause, program intelligibility, program sequencing

CR Categories: 4.22, 5.23, 5.24

Editor:

For a number of years I have been familiar with the observatio
that the quality of programmers is a decreasing function of th
density of **go to** statements in the programs they produce. Mor
recently I discovered why the use of the **go to** statement has suc
disastrous effects, and I became convinced that the **go to** state
ment should be abolished from all "higher level
languages (i.e. everythi

dynamic progress is only c

"GOTO Considered Harmful" Considered Harmful

The most-noted item ever published in *Communications* was a letter from Edsger W. Dijkstra
"Go To Statement Con-
Harmful" [1] which at-
ed to give a reason why the
statement might be harm-
lthough the argument was
emic and unconvincing, its
to have become fixed

" 'GOTO Considered Harmful' Considered Harmful" Considered Harmful?

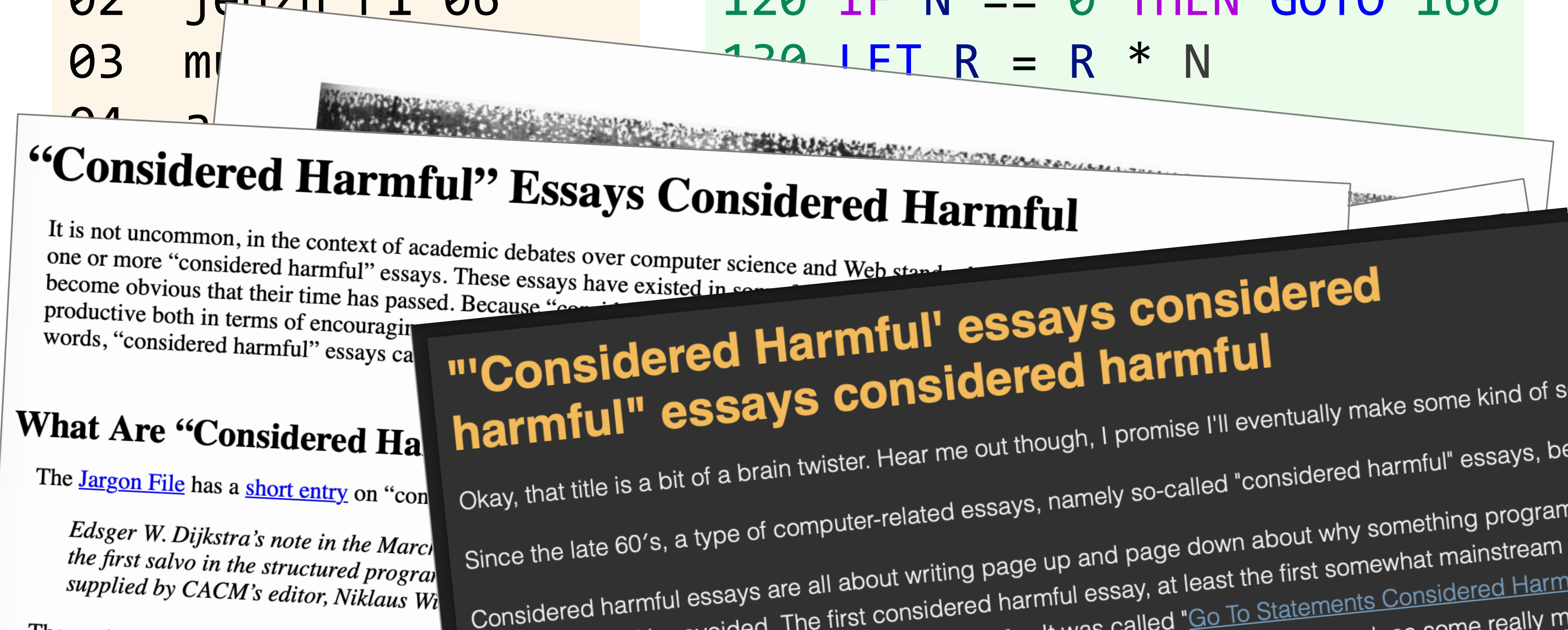
I enjoyed Frank Rubin's letter ("'GOTO Considered Harmful' Considered Harmful," March 1987

Factorial Revisited

The epic battle for
...whatever...

```
00  read r1
01  setn r2 1
02  jeqzn r1 06
03  mu
04  a
```

```
100 INPUT N
110 LET R = 1
120 IF N == 0 THEN GOTO 160
130 LET R = R * N
```



“Considered Harmful” Essays Considered Harmful

It is not uncommon, in the context of academic debates over computer science and Web stand one or more “considered harmful” essays. These essays have existed in so become obvious that their time has passed. Because “consid productive both in terms of encouragin words, “considered harmful” essays ca

What Are “Considered Ha

The Jargon File has a short entry on “con

Edsger W. Dijkstra’s note in the Marc the first salvo in the structured progra supplied by CACM’s editor, Niklaus Wi

"Considered Harmful' essays considered harmful" essays considered harmful

Okay, that title is a bit of a brain twister. Hear me out though, I promise I'll eventually make some kind of s

Since the late 60’s, a type of computer-related essays, namely so-called "considered harmful" essays, be

Considered harmful essays are all about writing page up and page down about why something program avoided. The first considered harmful essay, at least the first somewhat mainstream It was called "Go To Statements Considered Harm some really m

Factorial Revisited

Invent the **`while`** loop…
Lots in common with **`if`**

```
100 INPUT N
110 LET R = 1
120 IF N == 0 THEN GOTO 160
130 LET R = R * N
140 LET N = N - 1
150 GOTO 120
160 PRINT R
170 STOP
```

BASIC — Dartmouth College, 1963

```python
n = int(input())
r = 1
while n != 0:
   r = r * n
   n = n - 1

print(r)
```

Python

Two ways to program...

Imperative code!

- Inspired by machine
- Modify old variables
- Repeat using loops

What we're doing now...

Functional code!

- Inspired by math
- Make new variables
- Repeat using recursion

What did in week one...

A common pattern…

```
foods = ["apple", "banana", "cherry"]
```

```
i = 0
while i < len(foods):
    food = foods[i]
    print(food)
    i = i + 1
```

A common pattern…

```
foods = ["apple", "banana", "cherry"]
```

```
i = 0
while i < len(foods):
    food = foods[i]
    print(food)
    i = i + 1
```

```
for food in foods:
    print(food)
```

Invent the **for** loop…
A better way?

`for` loops: four examples...

This slide is four for `for`!

For loops define and assign a variable!!!

```
for i in [0,1,2]:
    print("i is", i)
```

The variable has each value in turn from some sequence

There's an indented block of code it'll execute each time

Imperative design in Python

for

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

while

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

Loops!

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

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

But we change it as we go...

```
addn r1 1
```

variables **vary**

a lot!

for loops: four examples...

This slide is four for **for**!

```
for i in [0,1,2]:
    print("i is", i)
```

For loops define and assign a variable!

for loops: four examples...

This slide is four for for!

```
for i in [0,1,2]:
    print("i is", i)
```

```
i is 0
i is 1
i is 2
```

For loops define and assign a variable!!

for loops: four examples...

This slide is
four for **for**!

```
for i in [0,1,2]:
    print("i is", i)
```

```
[0,1,2]
```

```
for i in range(0,3):
    print("i is", i)
```

```
i is 0
i is 1
i is 2
```

For loops <u>define</u> and <u>assign</u> a variable!!!

for loops: four examples…

This slide is
four for for!

```
for i in [0,1,2]:
    print("i is", i)
```

```
for i in range(0,3):
    print("i is", i)
```

[0,1,2]

```
i is 0
i is 1
i is 2
```

```
for x in [2,5,2024]:
    print("x is", x)
```

```
x is 2
x is 5
x is 2024
```

```
for i in 
    print('Happy birthday!')
```

How could we get
this to run 42 times?

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

for loops: four examples...

This slide is
four for for!

```
for i in [0,1,2]:
    print("i is", i)
```

[0,1,2]

```
for i in range(0,3):
    print("i is", i)
```

```
i is 0
i is 1
i is 2
```

```
for x in [2,5,2024]:
    print("x is", x)
```

```
x is 2
x is 5
x is 2024
```

```
for i in range(42)
    print('Happy birthday!')
```

How could we get
this to run 42 times?

`range(1,43)`
`range(0,42)`

There are a ***range*** of answers to this one...

for fun(ctions)

[0,1,2]

```
def funA():
    for i in range(0,3):
        print("i is", i)
    return
```

[0,1,2]

```
def funB():
    for i in range(0,3):
        print("i is", i)
        return
```

for fun(ctions)

[0,1,2]

```python
def funA():
    for i in range(0,3):
        print("i is", i)
    return
```

[0,1,2]

```python
def funB():
    for i in range(0,3):
        print("i is", i)
        return
```

for vs. return ?

Who wins???

Epic keyword battle...

for fun(ctions)

[0,1,2]

```
def funA():
    for i in range(0,3):
        print("i is", i)
    return
```

[0,1,2]

```
def funB():
    for i in range(0,3):
        print("i is", i)
        return
```

return wins!

return ?

wins???

e keyword battle...

for fun(ctions)

[0,1,2]

```
def funA():
    for i in range(0,3):
        print("i is", i)
    return
```

outside the loop

```
i is 0
i is 1
i is 2
```

return!

[0,1,2]

```
def funB():
    for i in range(0,3):
        print("i is", i)
        return
```

inside the loop!

```
i is 0
```

return!

Name: ________________ BDay! ________

```
def fun1():
  for i in range(1,6):
    if i%2 == 0:
      print("i is", i)
  return
```

# of times the for loop runs?

# of times the if-test is True?

```
def fun3():
  for i in range(1,6):
    if i%2 == 0:
      print("i is", i)
    return
```

```
def fun2():
  for i in range(1,6):
    if i%2 == 0:
      print("i is", i)
      return
```

```
def fun4():
  for i in range(1,6):
    if i%2 == 0:
      print("i is", i)
return
```

four fors

A

what prints:

no printing...

The loop runs **1** time,
then the function returns
i=1, i=2, i=3, i=4, i=5

The if-test is never True

B

what prints:

```
syntax
error
```

The loop never runs...
The function never runs...

The if-test never runs

C

what prints:

```
i is 2
```

The loop runs **2** times,
then the function returns
i=1, i=2, i=3, i=4, i=5

The if-test is True **1** time

D

what prints:

```
i is 2
i is 4
```

The loop runs **5** times,
then the function returns
i=1, i=2, i=3, i=4, i=5

The if-test is True **2** times

Name: ________________ BDay! ________

```
def fun1():
   for i in range(1,6):
      if i%2 == 0:
         print("i is", i)
   return
```

# of times the for loop runs?

# of times the if-test is True?

D

```
def fun3():
   for i in range(1,6):
      if i%2 == 0:
         print("i is", i)
      return
```

A

```
def fun2():
   for i in range(1,6):
      if i%2 == 0:
         print("i is", i)
         return
```

C

```
def fun4():
   for i in range(1,6):
      if i%2 == 0:
         print("i is", i)
return
```

B

four fors

A

what prints:

no printing...

The loop runs **1** time,
then the function returns
i=1, i=2, i=3, i=4, i=5

The if-test is never True

B

what prints:

syntax error

The loop never runs...
The function never runs...

The if-test never runs

C

what prints:

```
i is 2
```

The loop runs **2** times,
then the function returns
i=1, i=2, i=3, i=4, i=5

The if-test is True **1** time

D

what prints:

```
i is 2
i is 4
```

The loop runs **5** times,
then the function returns
i=1, i=2, i=3, i=4, i=5

The if-test is True **2** times

Iterative design in Python

for

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

while

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

Loops!

variables **vary**

a lot!

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

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

But we change it as we go…

```
addn r1 1
```

?

!

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

```
age = 41
```

The "old" value (41)

```
age = age + 1
```

The "new" value (42)

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

```
age += 1
```

```
age *= 2
age -= 74
age /= 7
```

Echoes from Hmmm:

```
05    addn    r1    1
```

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

```
age = 41

age = age + 1
```

The "old" value (41)

The "new" value (42)

```
age += 1
```

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

Python shortcuts

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

```
hwToGo -= 1
```

```
amoebas = 21000000
amoebas = amoebas * 2
```

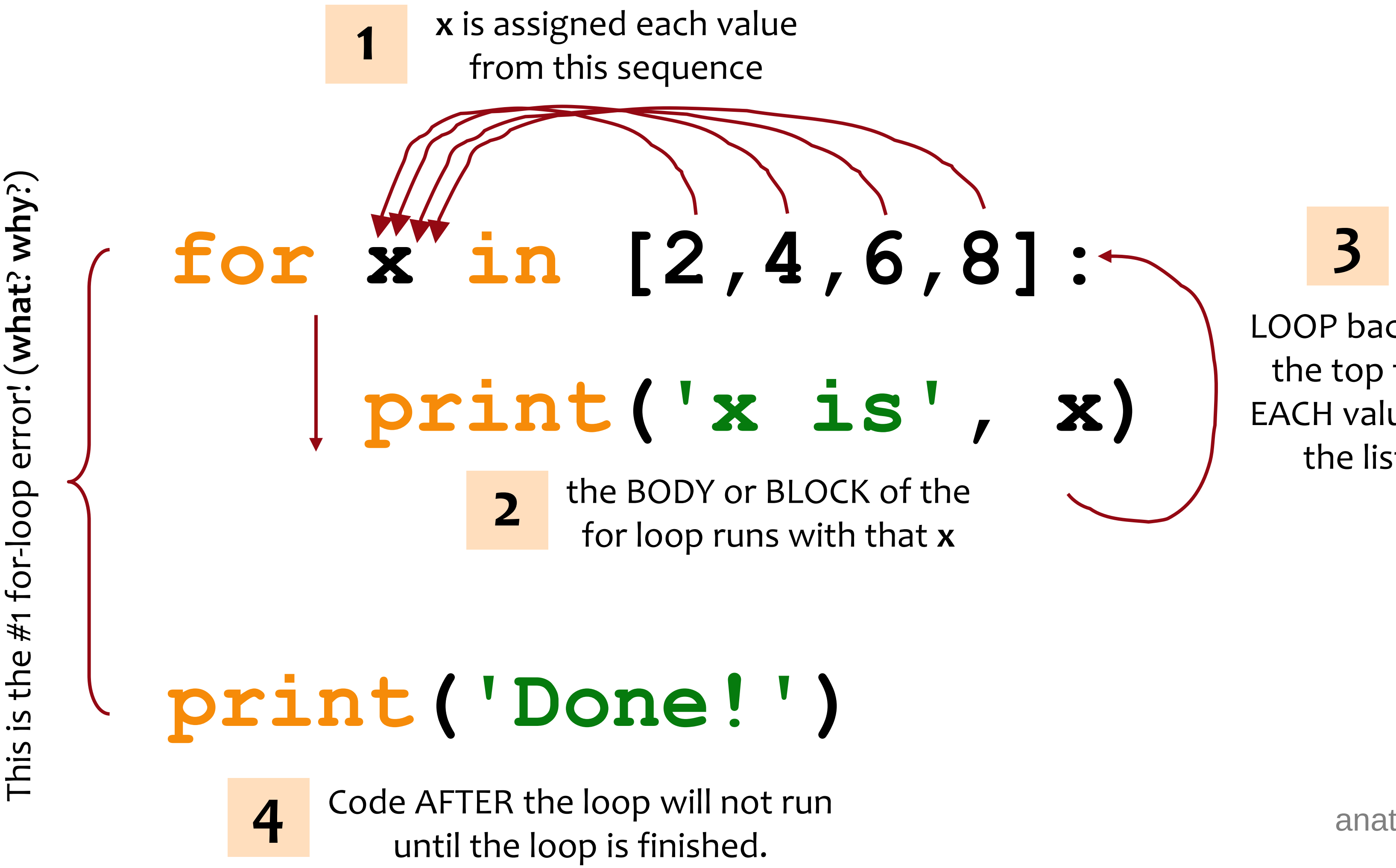
```
amoebas *= 2
```

```
u235 = 84000000000000000;
u235 = u235 / 2
```

```
u235 /= 2
```

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!')
```

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.

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

anatomy?

empty?

x unused?

Hmmm

Recursive Hmmm factorial, hw6pr4

```
00   setn r15 42
01   read r1
02   calln r14 5
03   write r13
04   halt
05   jnezn r1 8
06   setn r13 1
07   jumpr r14
08   pushr r14 r15
09   pushr r1 r15
10   addn r1 -1
11   calln r14 5
12   popr r1 r15
13   popr r14 r15
14   mul r13 r1 r13
15   jumpr r14
```

Loops!

Looping Hmmm factorial, similar to hw6pr2 and pr3

```
00 read r1
01 setn r2 1
02 jeqzn r1 06
03 mul r2 r2 r1
04 addn r1 -1
05 jumpn 02
06 write r2
07 halt
```

Functional programming → ***Iterative*** programming

Hmmm... I think I'll take Python!

four questions for `for`

what list is this!?

find the sum of the list?

printing partial sums?

factorial function?

```
for x in range(1,8):

    print('x is', x)
```

four questions for **for**

what list is this!?

find the sum of the list?

printing partial sums?

factorial function?

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

```
for x in range(1,8):


    print('x is', x)
```

tsum with for

how to use N?

find the sum of the list?

printing partial sums?

create factorial?!

Four questions...

```
def tsum( N ):

    for x in range(1,5):
        print("x is", x)
```

tsum with for

```
def tsum( N ):
    result = 0
    for x in range(0,N+1):
        result = result + x
    return result
```

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

thought experiments w/ return

`fac` with `for`

```
fac( 5 ):
```

We want to return `1 * 2 * 3 * 4 * 5`

fac with for

```
fac( 5 ):
```

We want to return `1 * 2 * 3 * 4 * 5`

```
fac( N ):
```

We want to return `1 * 2 * 3 * ... * N`

fac with for

how to use N?

find the sum of the list?

printing partial sums?

create factorial?!

Four questions...

```
def fac( N ):

    for x in range(       ):

    return result
```

fac with for

```
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

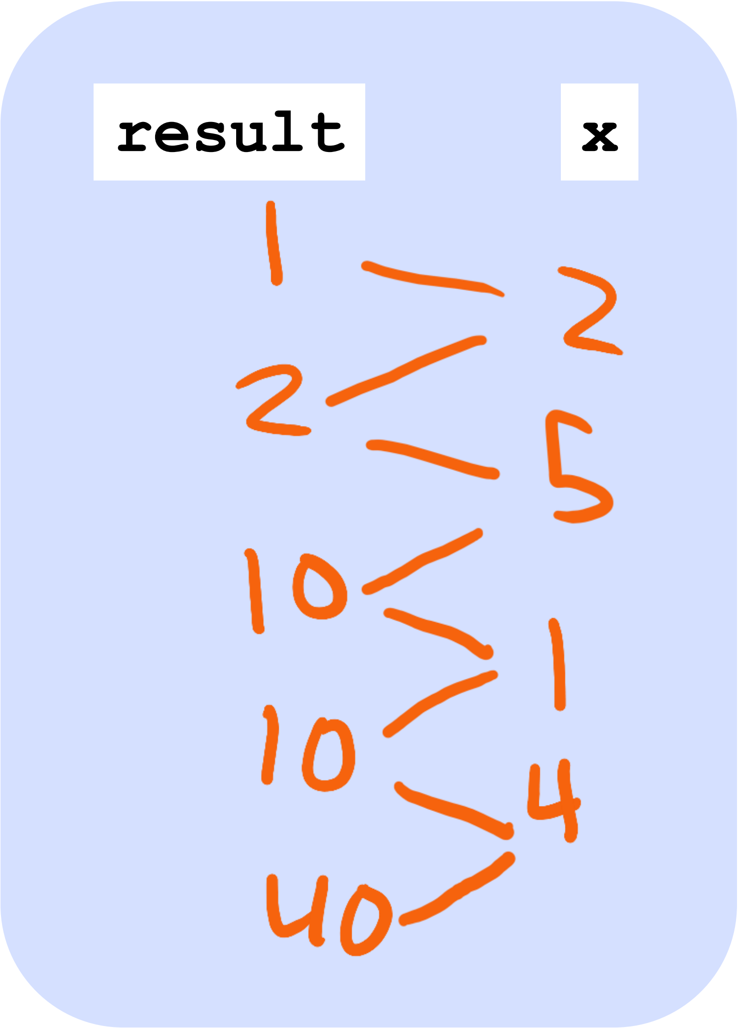
`for`-loop *"laddering"*

Warning: **no one** else uses this term...

```
result = 1

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

print(result)
```



result

x

meets up with Jacob's ladder

Fun!

What does the loop say?

```
     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
S = 'time to think this over! '


result = ''

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

print(result)
```

[0,1,2,...,24]

25

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

res.	S[i-1]	S[i]	i
			0
			1
			2
			3
			4
			5
			6
			7
			8

Fun!

What does the loop say?

```
       0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
S = 'time to think this over! '
```

```
result = ''
```

[0,1,2,...,24]

25

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

print(result)
```

tttto

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

res.	S[i-1]	S[i]	i
	' '	't'	0
	't'	'i'	1
	'i'	'm'	2
	'm'	'e'	3
	'e'	' '	4
	' '	't'	5
	't'	'o'	6
	'o'	' '	7
	' '	't'	8

Fun!

What does the loop say?

```
      0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
S = 'time to think this over! '
```

```
result = ''
```

[0,1,2,...,24]

25

```
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!

res.	S[i-1]	S[i]	i
't'	' '	't'	0
	't'	'i'	1
	'i'	'm'	2
	'm'	'e'	3
	'e'	' '	4
'tt'	' '	't'	5
	't'	'o'	6
	'o'	' '	7
'ttt'	' '	't'	8

Extra! How could you change one character above to yield

change ' ' to 'i'

mns

or another to yield

change 1 to 4

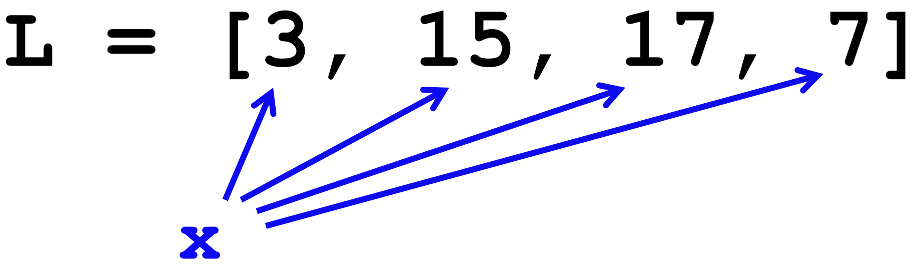
etnsr

or another to yield

change - to +

eoks!

for: *two types*



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

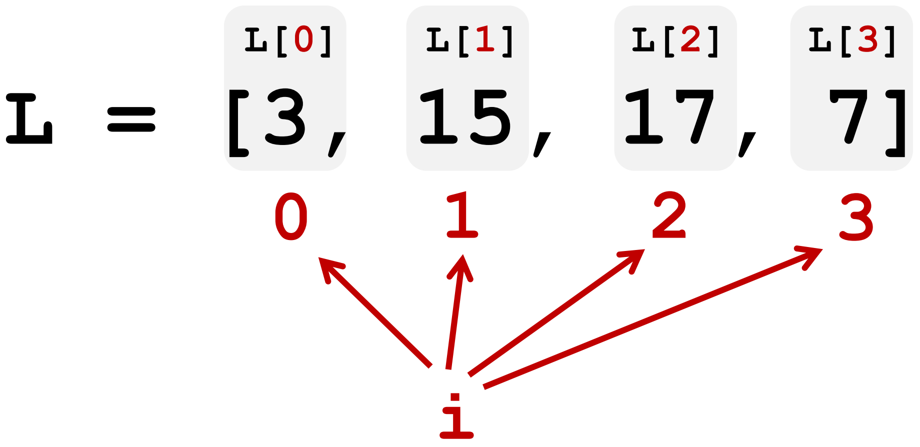
Elements vs Indexes

Indices

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

***element*-based loops**

`for`: *two types*



L[0] L[1] L[2] L[3]

L = [3, 15, 17, 7]

0 1 2 3

i

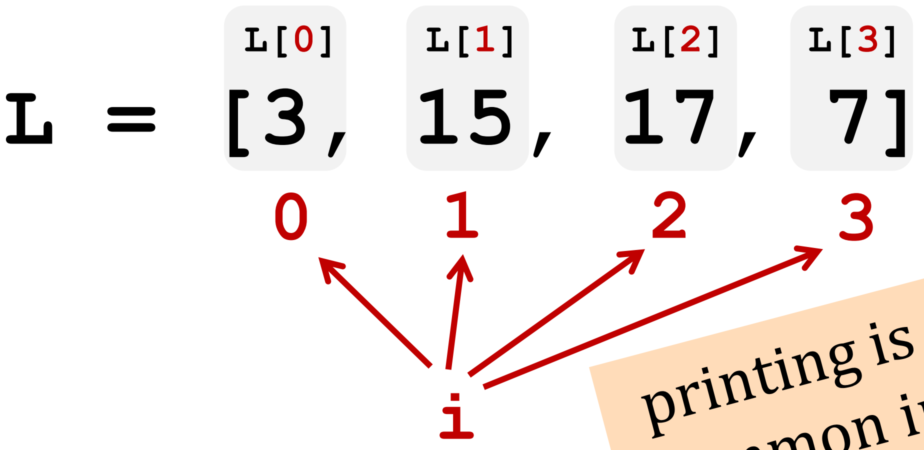
```
         list
for i in range(len(L)):
   print(L[i])
          x
```

***index*-based loops**

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

***element*-based loops**

`for`: *two types*



L[0] L[1] L[2] L[3]

L = [3, 15, 17, 7]

0 1 2 3

i

printing is ***NOT*** especially common in loops – *but it's good for debugging!*

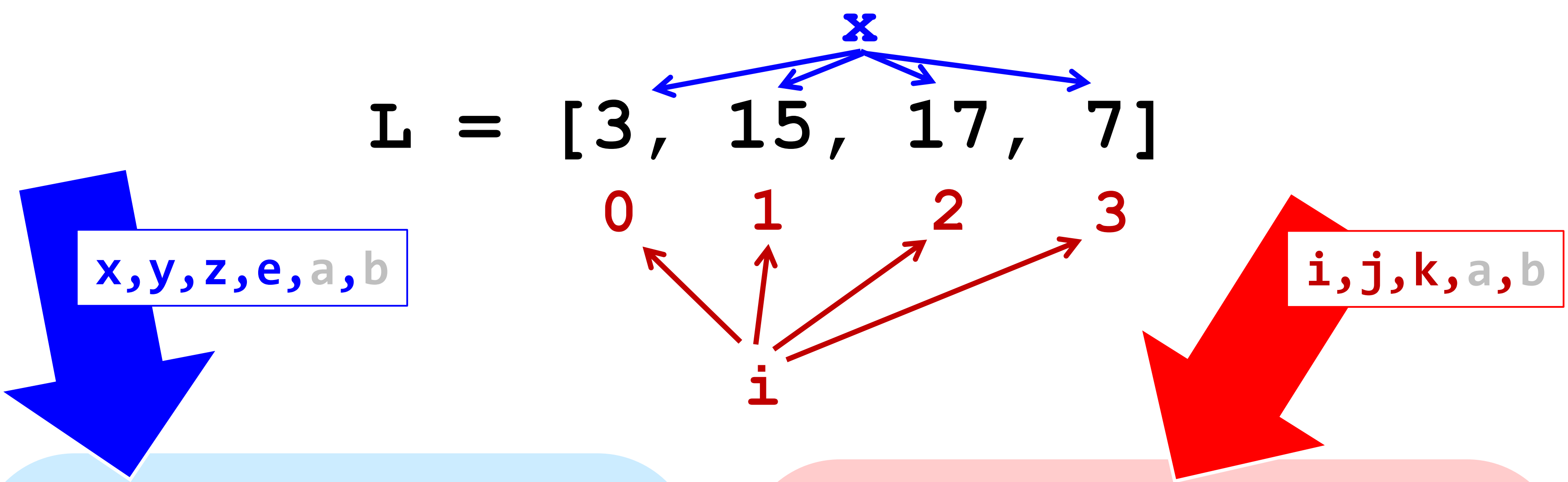
```
                list
for i in range(len(L)):
   print(L[i])
           x
```

index-based loops

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

element-based loops

simpler vs. *flexibler*



x

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

0 1 2 3

i

x,y,z,e,a,b

i,j,k,a,b

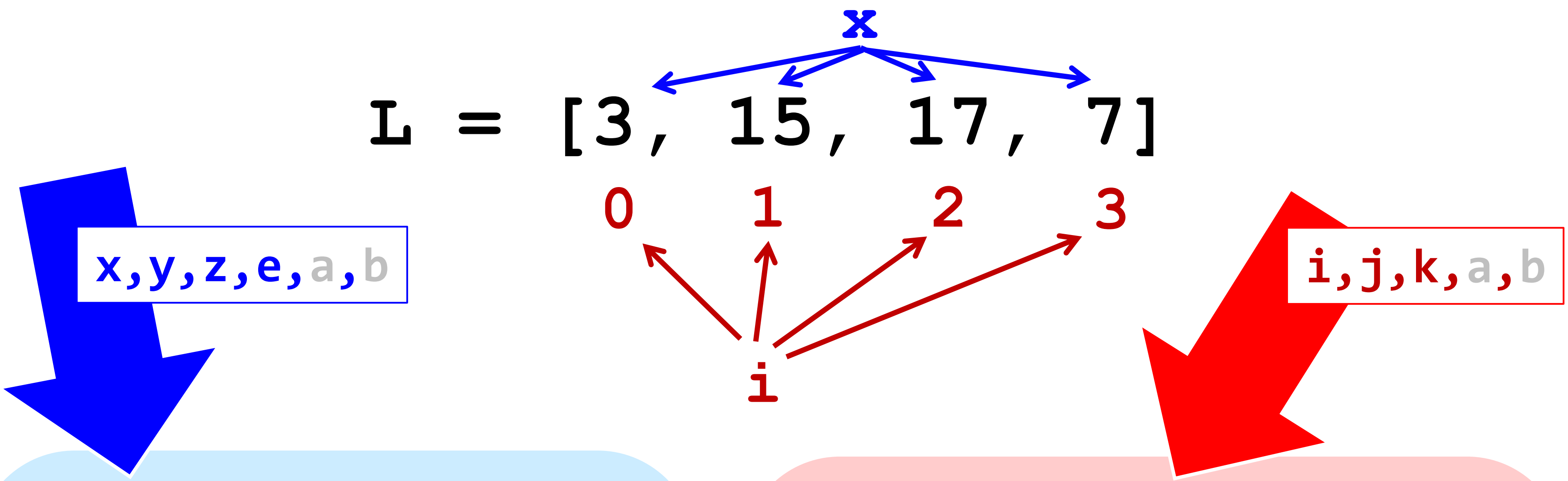
element-based loops

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

index-based loops

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

simpler vs. *flexibler*



***element*-based loops**

```
def sum(L):
   total = 0

   return total
```

***index*-based loops**

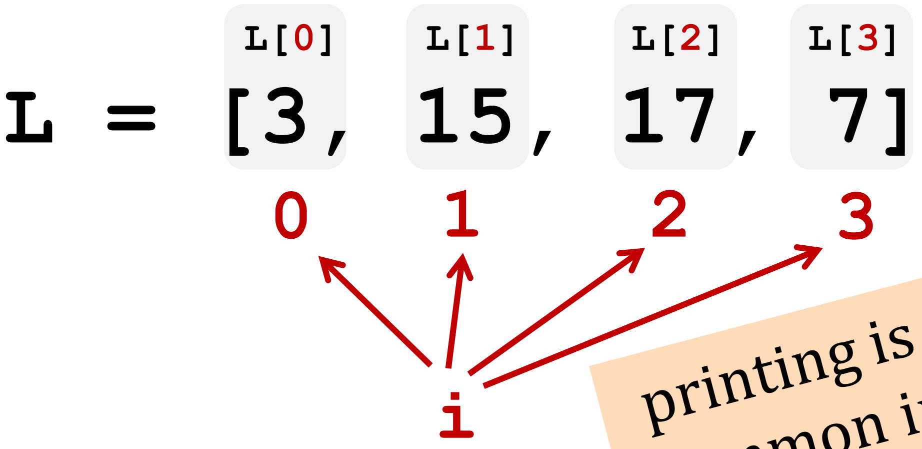
```
def sum(L):

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

Elements vs Indexes

(indices)

for: *two types*



printing is ***NOT*** especially common in loops – *but it's good for debugging!*

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

index-based loops

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

element-based loops

Extreme Looping

What does this loop do?

```
guess = 42

print('It keeps on')

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


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

continuing if

I'm whiling away my time with this one!

Extreme Looping

What does this loop do?

```
guess = 42

print('It keeps on')

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

while loop body

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

other tests we could use here?

```
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

many different tests…

```
print('It keeps on')

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

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

others?

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!')
```

a *"while True"* loop

I'm whiling away my
time with this one!

while we escape?!

```
import random

def escape( N ):
    """ keeps guessing! """
    guess = 0
    while guess != 42:

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

    print('At last!')
    return count
```

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

test to see ***if we keep looping***

watch out for infinite loops!

after the loop ends

Yikes! How should we count here?!

how could we count the number of loops we run?

how could we accumulate a LIST of all the guesses?

random.uniform!