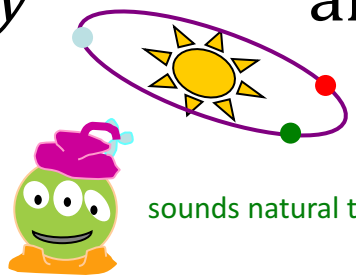
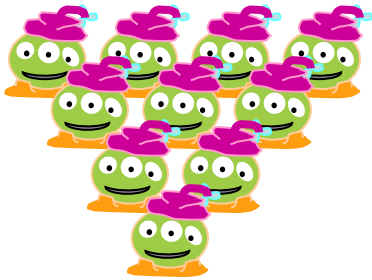
Today Coding in *circles*!

Thinking *loopily* and *cumulatively*

`for` a `while` `+=`



sounds natural to me!



Today ***Loops*** have arrived…

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
```

But we change it as we go…

```
addn r1 1
```

`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` loops: examples...

This slide is four for `for`!



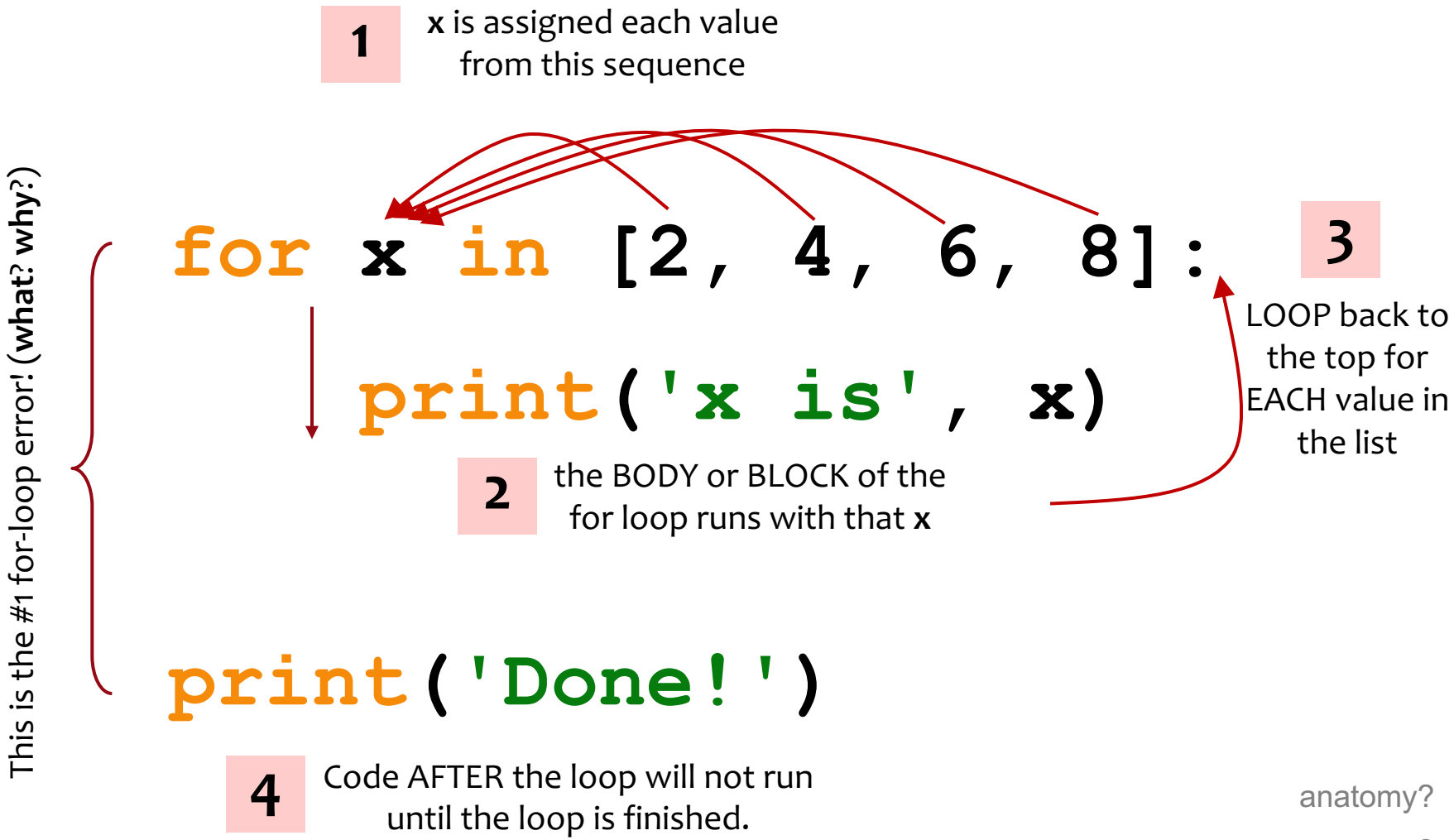
```
for x in [2, 4, 6, 8]:
    print('x is', x)


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


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


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

for!



anatomy?

empty?

x unused?

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

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

The "old" value (41)

The "new" value (42)

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



```
age += 1
```

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

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

The "old" value (41)

The "new" value (42)

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



```
age += 1
```

Python shortcuts

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

```
hwToGo -= 1
```

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

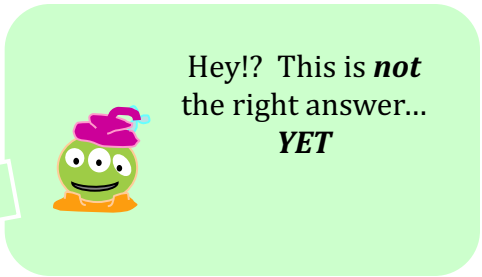
```
amoebas *= 2
```

```
u235 = 10000000000000;
u235 = u235 / 2
```

```
u235 /= 2
```

fac with for

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



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



Laddering `for` loops

4

```
def fac(N):
    result = 1
    for x in range(1, N + 1):
        result = result * x

    return result
```

result	x

meets up with Jacob's ladder

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



Laddering `for` loops

4

```
def fac(N):
    result = 1
    for x in range(1, N + 1):
        result = result * x

    return result
```

result	x
1	

meets up with Jacob's ladder

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



Laddering `for` loops

4

```
def fac(N):
    result = 1
    for x in range(1, N + 1):
        result = result * x

    return result
```

result	x
1	1
1	

meets up with Jacob's ladder

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



Laddering `for` loops

4

```
def fac(N):
    result = 1
    for x in range(1, N + 1):
        result = result * x

    return result
```

result	x
1	1
1	2
2	

meets up with Jacob's ladder

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



Laddering `for` loops

4

```
def fac(N):
    result = 1
    for x in range(1, N + 1):
        result = result * x

    return result
```

result	x
1	1
1	2
2	3
6	

meets up with Jacob's ladder

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



Laddering `for` loops

4

```
def fac(N):
    result = 1
    for x in range(1, N + 1):
        result = result * x

    return result
```

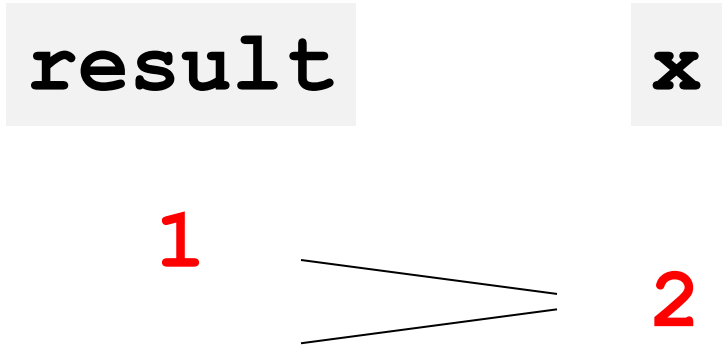
result	x
1	1
1	2
2	3
6	4
24	

meets up with Jacob's ladder

Quiz

What does the loop say?

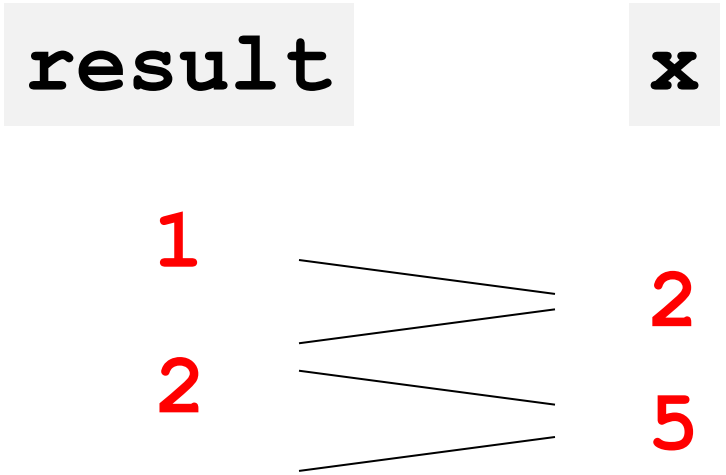
```
result = 1
for x in [2, 5, 1, 4]:
  result *= x
print(result)
```



Quiz

What does the loop say?

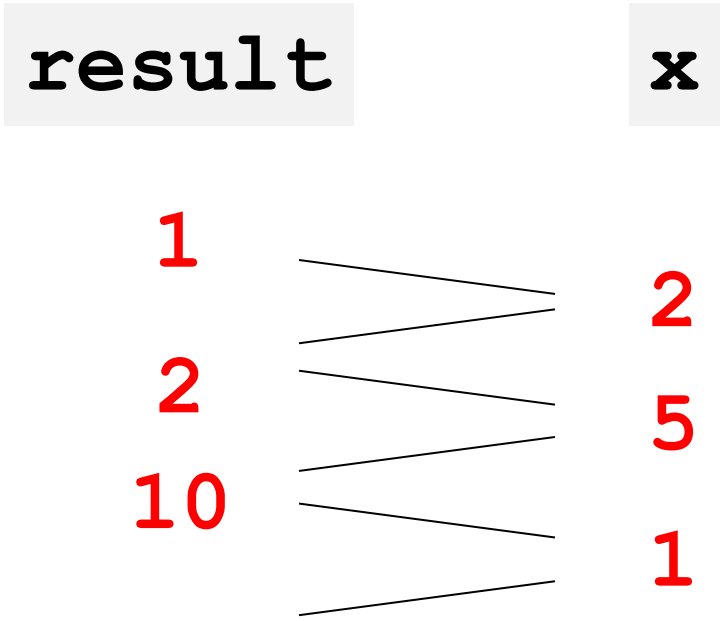
```
result = 1
for x in [2, 5, 1, 4]:
  result *= x
print(result)
```



Quiz

What does the loop say?

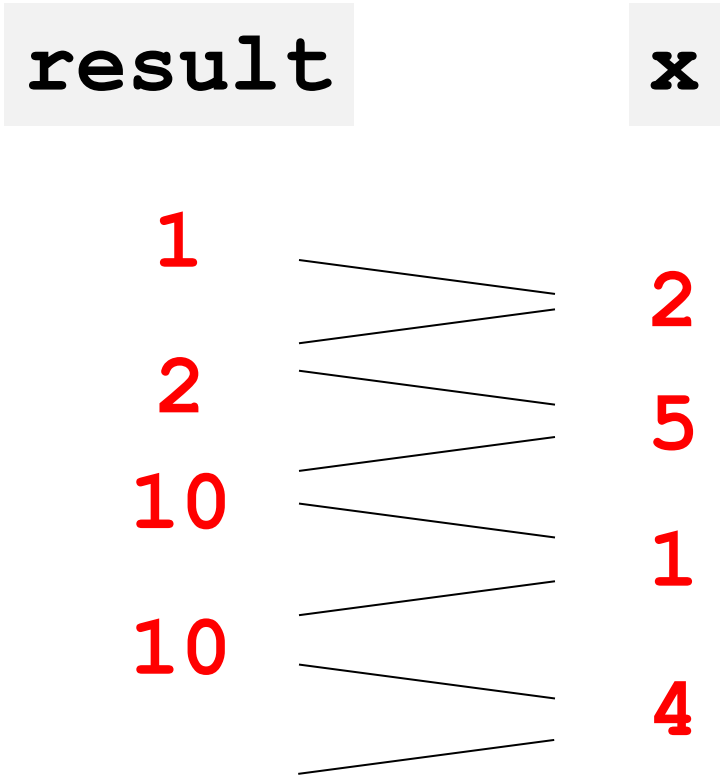
```
result = 1
for x in [2, 5, 1, 4]:
  result *= x
print(result)
```



Quiz

What does the loop say?

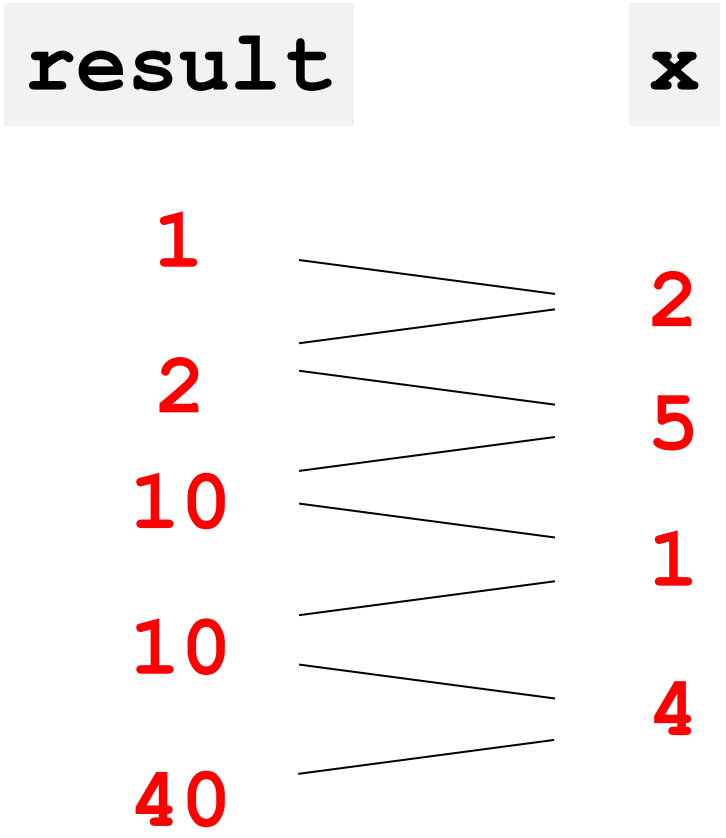
```
result = 1
for x in [2, 5, 1, 4]:
  result *= x
print(result)
```



Quiz

What does the loop say?

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



Quiz

What does the loop say?

```
x = 0

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

print(x)
```

[0, 1, 2, 3]



x	i
0	0

Quiz

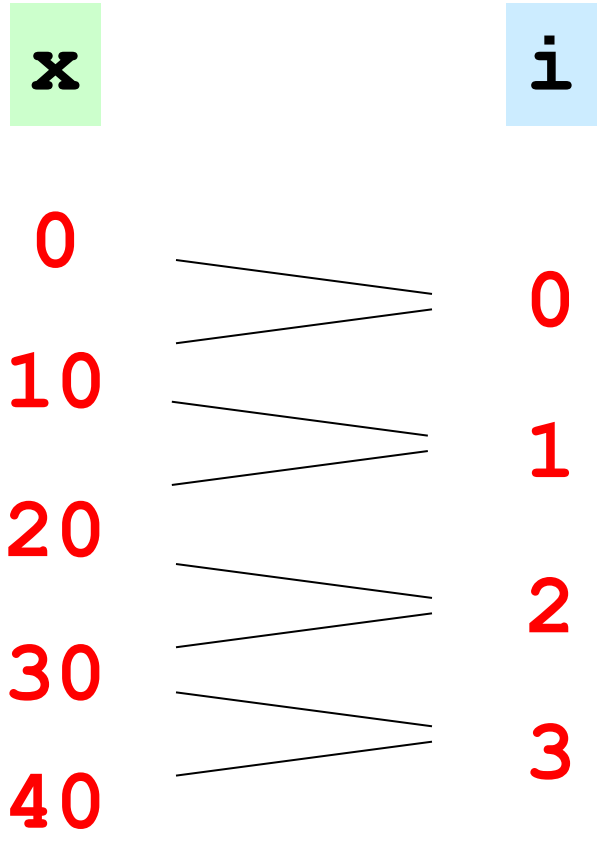
What does the loop say?

```
x = 0

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

print(x)
```

[0, 1, 2, 3]



Quiz

What does the loop say?

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

[0, 1, 2, 3]

4

```
for i in range(len(L)):
    if i%2 == 1:
        print(L[i])
```

L[i]	i%2	i

Quiz

What does the loop say?

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

for i in range(len(L)):
    if i%2 == 1:
        print(L[i])
```

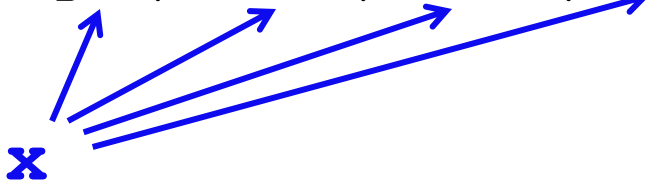
[0, 1, 2, 3]

4

L[i]	i%2	i
'golf'	0	0
'fore!'	1	1
'club'	0	2
'tee'	1	3

`for`: *two types*

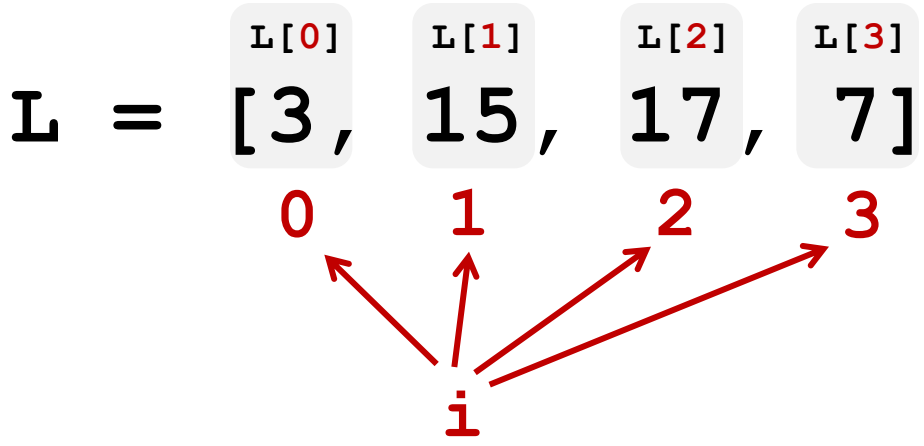
```
L = [3, 15, 17, 7]
```




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

element-based loops

`for`: *two types*



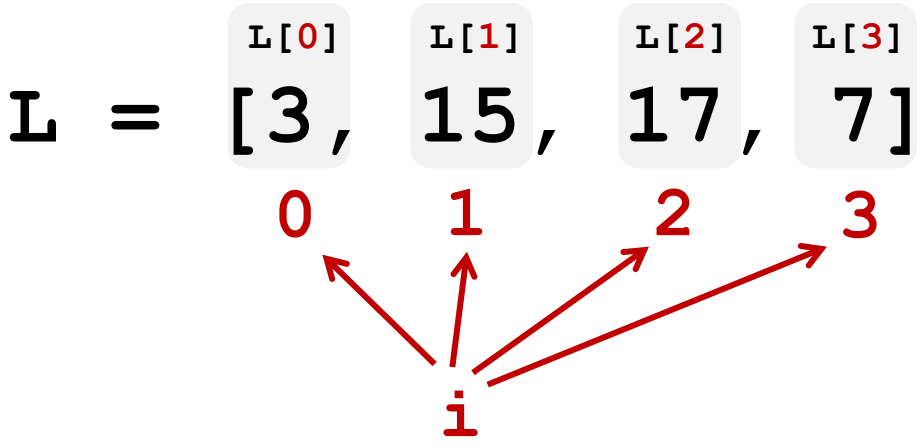

```
for i in range(len(L)):
  print(    )
```

index-based loops

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

element-based loops

`for`: *two types*



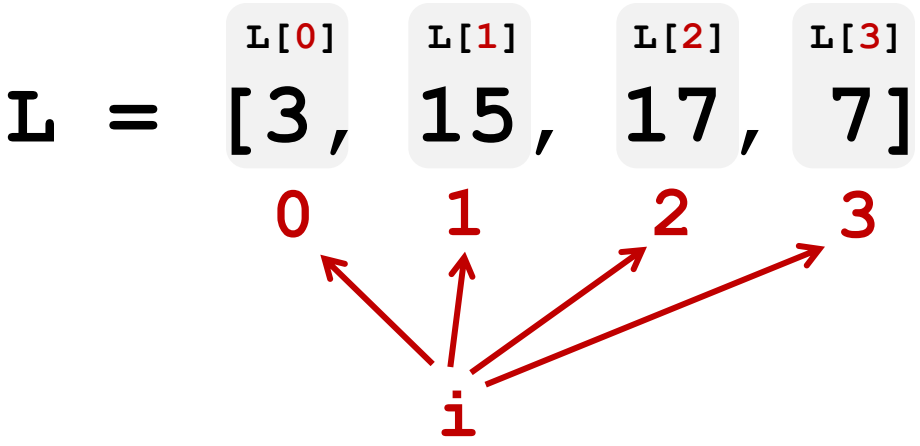

```
for i in range(len(L)):
    print(L[i])
```

index-based loops

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

element-based loops

`for`: *two types*




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

index-based l

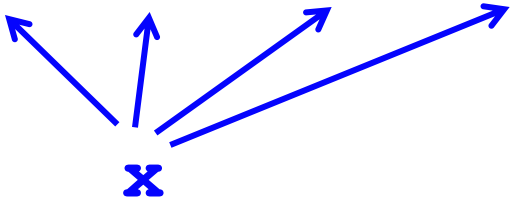
```
for x in L:
    print(x)
```

elemen

printing is **NOT** common in loops—these are just by way of example...

Simpler vs. *More Flexible*

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

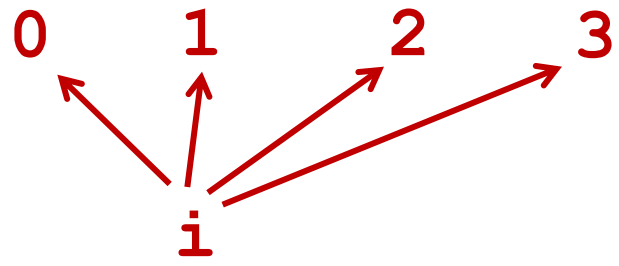



```
def sum(L):
    total = 0
    for x in L:
        total += x
    return total
```

element-based loops

Simpler vs. *More Flexible*

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




```
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
```

index-based loops

Extreme Looping

What does this code do?

```
print('It keeps on')

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


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

continuing `if`

I'm whiling away my time with this one!



Extreme Looping

Anatomy of a `while`

```
print('It keeps on')

while 41+1 == 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

lots of different tests...

```
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...

a "while True" loop

```
print('It keeps on')

while True:
    print('going and')


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

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?

How long till 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/

How long til a *repeat*?

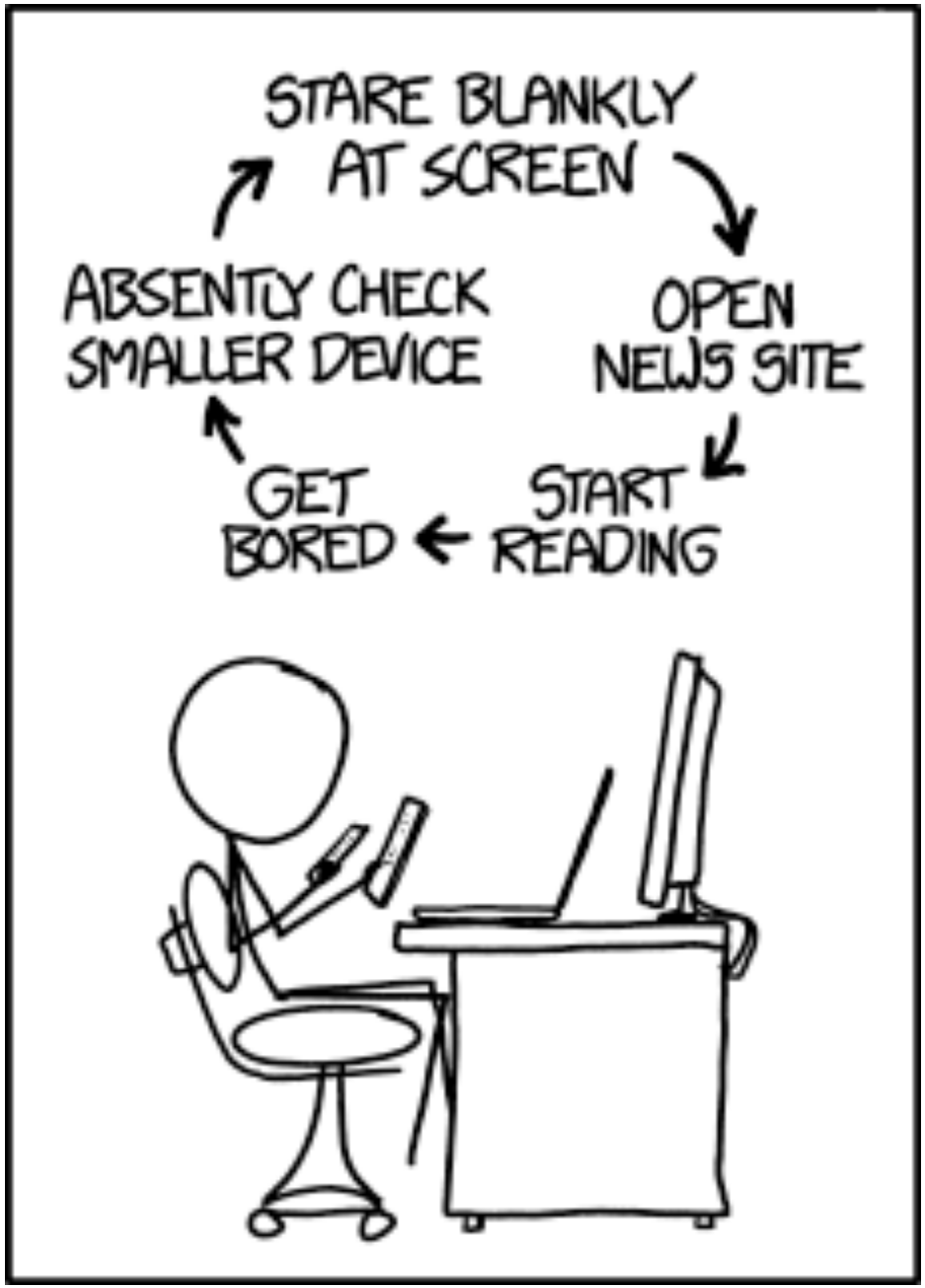
```
import random
def bday():
    LoBs = []    # List of BDays
    b = random.choice(range(365))
    while b not in LoBs:
        LoBs += [b]
        b = random.choice(range(365))

print("Repeat! # tries:", len(LoBs)+1)
```

start w/ an empty list

add a new birthday – need to add list + list!

get a new birthday



XKCD's loop

and this was before watches – or glasses...

