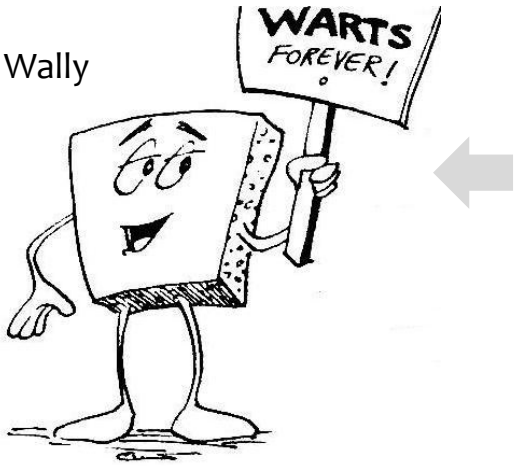
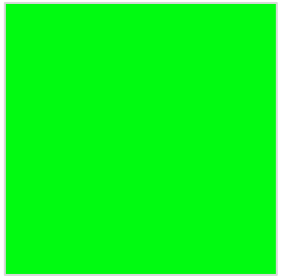
Welcome back to CS 5 !

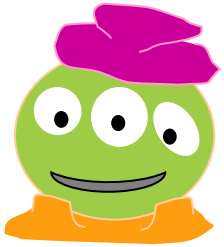
Wally



Our Week 0 CS5 gallery



Picobot



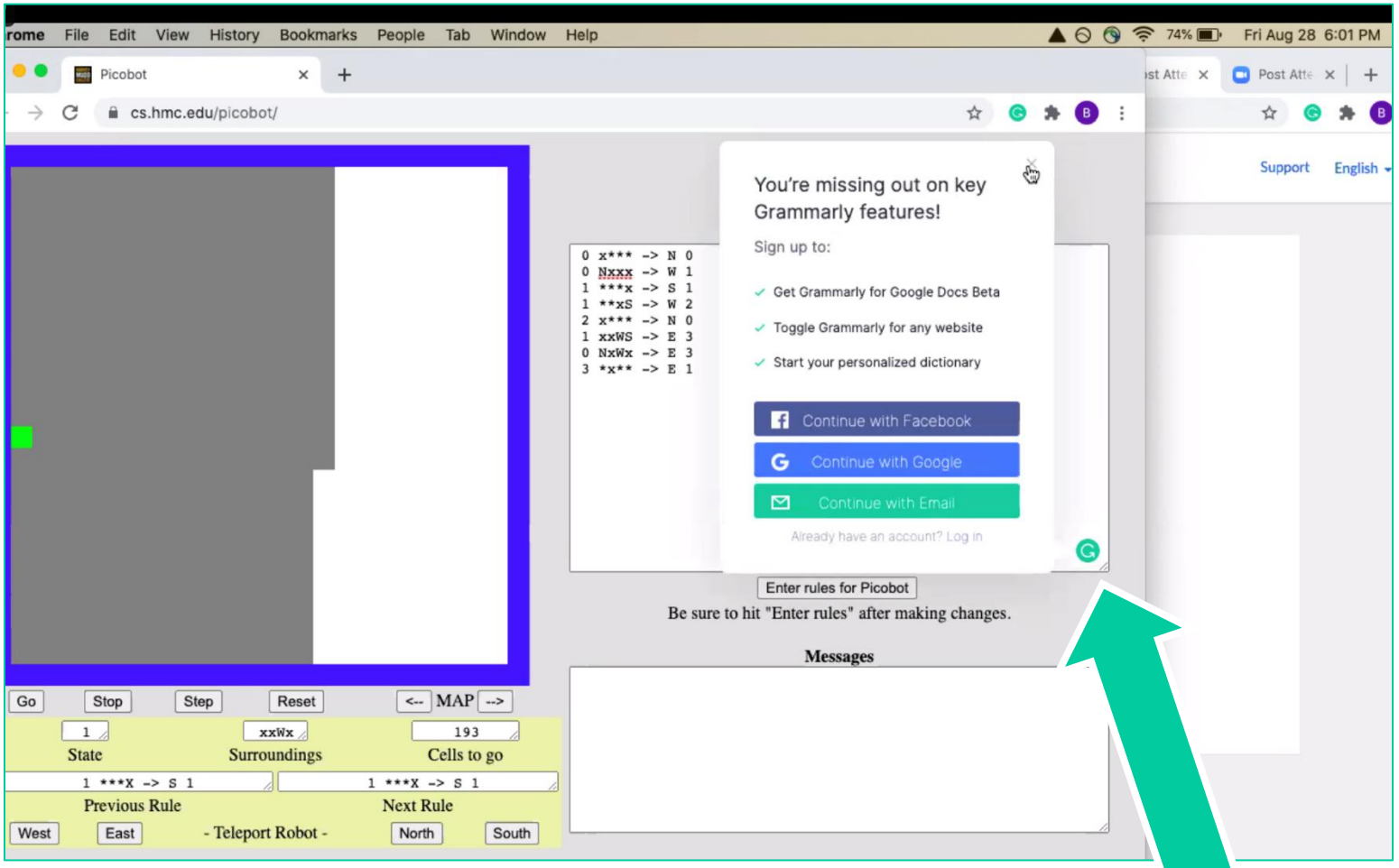
Alien

Homework 0

- **Problem 0:** Reading + response…
- **Problem 1:** Four-fours program: Can be done for lab…
- **Problem 2:** Rock-paper-scissors + Adventure
- **Problems 3-4:** Picobot! empty room (3) maze (4)

Picobot ~ problems... ?

My Grammarly is in *valid-Picobot-only* mode

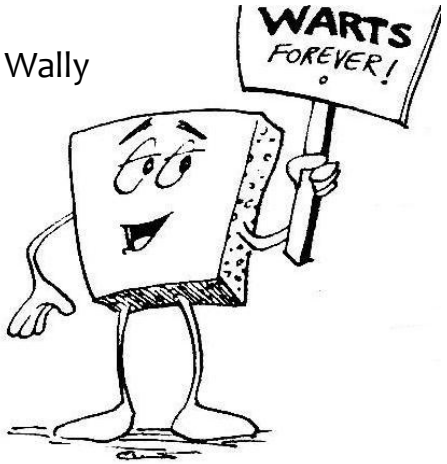


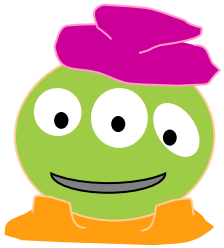
Grammarly *agrees* !

Looking forward to Week 1...

[A] What other work *might* **`adventure()`** have encouraged you to procrastinate... ?

[B] What if CS 5 were now ***finished*** with Picobot?



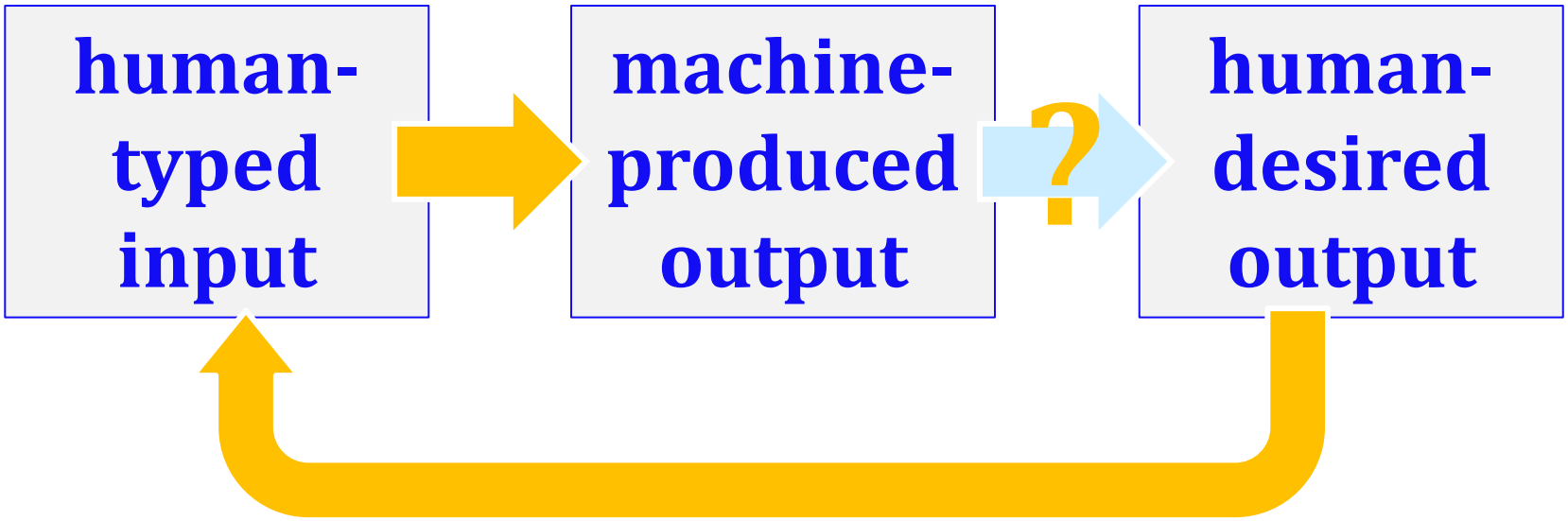


Homework 1

- **Problem 0:** Reading + response… — *due next Tuesday*
- **Problems 1+2:** Slicing and indexing: These are lab … — *due next Tuesday*
- **Problem 3:** Functions! In lab or beyond… — *due next Tuesday*
- **hw2pr4** PythonBat functions (not due this week - but they can be addictive!) — *due next next Tuesday!*

The ***challenge*** of programming…

syntax	semantics	intent
How it looks	What it does	What it should do

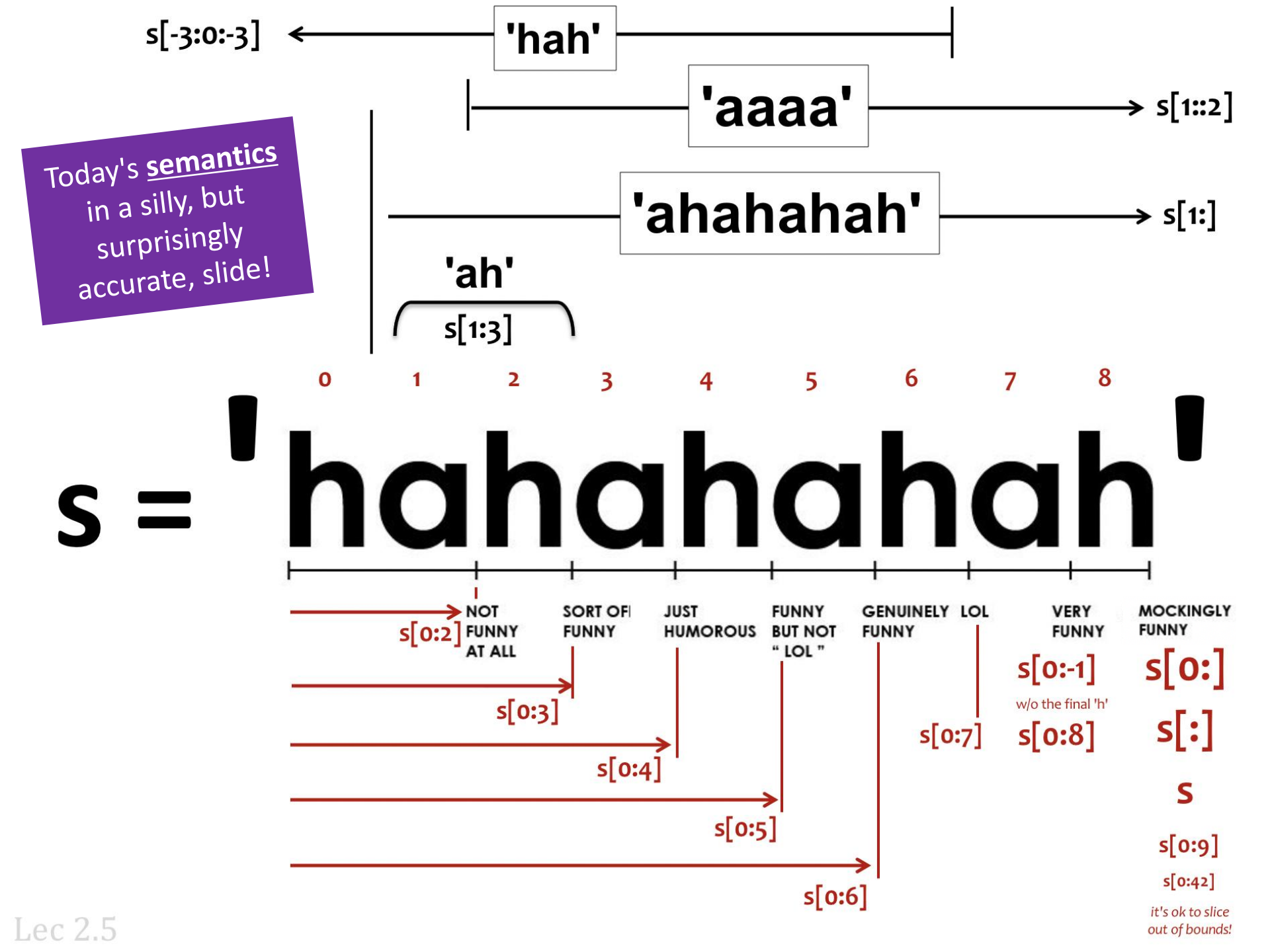


learning a language ~ ***syntax***

unavoidable, but not the point

… but learning CS ~ ***semantics*** 

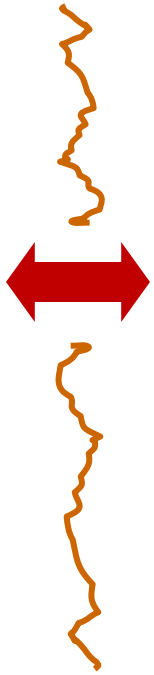
guiding how machines ***think***!


s[-3:0:-3]
'hah'
'aaaa'
s[1::2]
'ahahahah'
s[1:]
'ah'
s[1:3]
Today's semantics in a silly, but surprisingly accurate, slide!
0 1 2 3 4 5 6 7 8
s = 'hahahah'
s[0:2]
NOT FUNNY AT ALL
SORT OF FUNNY
JUST HUMOROUS
FUNNY BUT NOT " LOL "
GENUINELY FUNNY
LOL
VERY FUNNY
MOCKINGLY FUNNY
s[0:3]
s[0:4]
s[0:5]
s[0:6]
s[0:7]
s[0:-1]
w/o the final 'h'
s[0:8]
s[0:]
s[:]
s
s[0:9]
s[0:42]
it's ok to slice out of bounds!

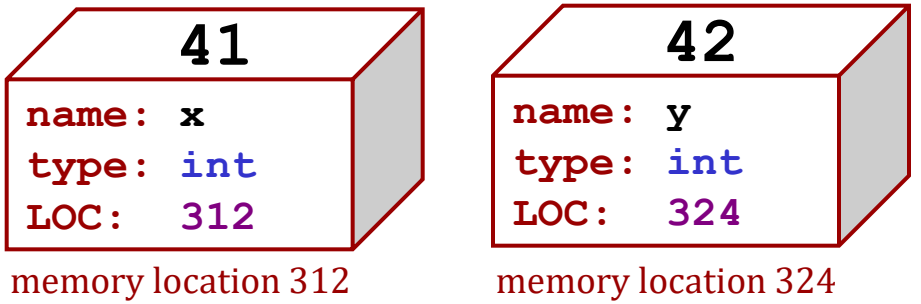
Inside the machine…

What's behind the scenes: Processing + Memory:

Computation



Data Storage



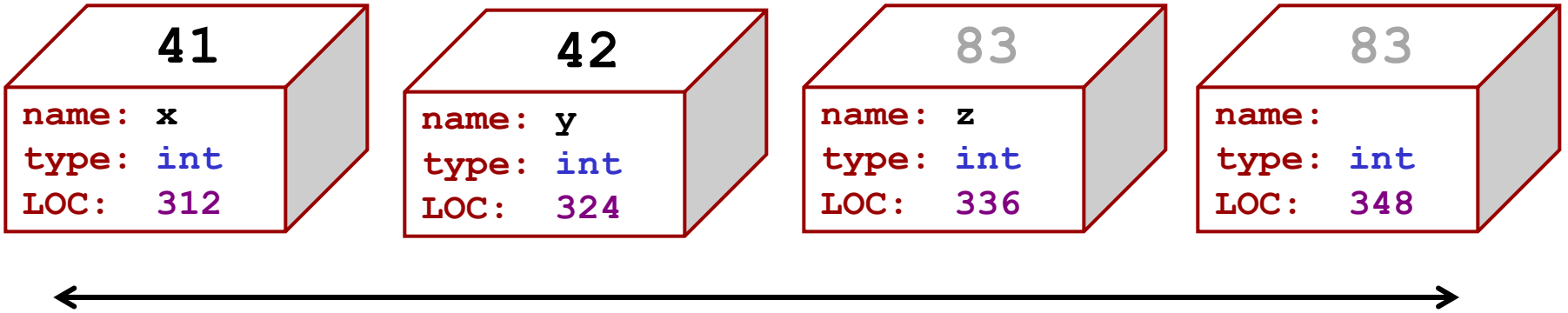
memory location 312

memory location 324

variables ~ ***boxes***

Memory!

Random Access Memory

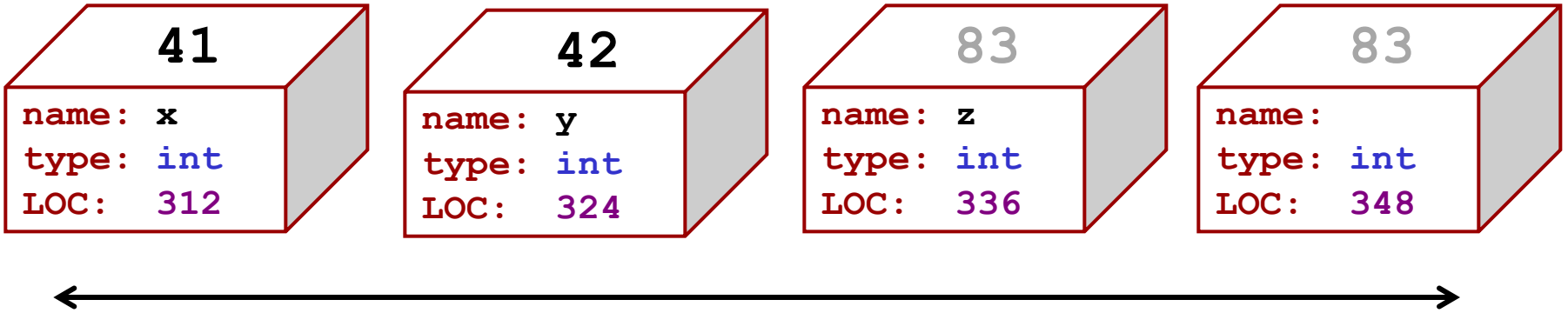


a big line of boxes, each with a name, type, location, and value

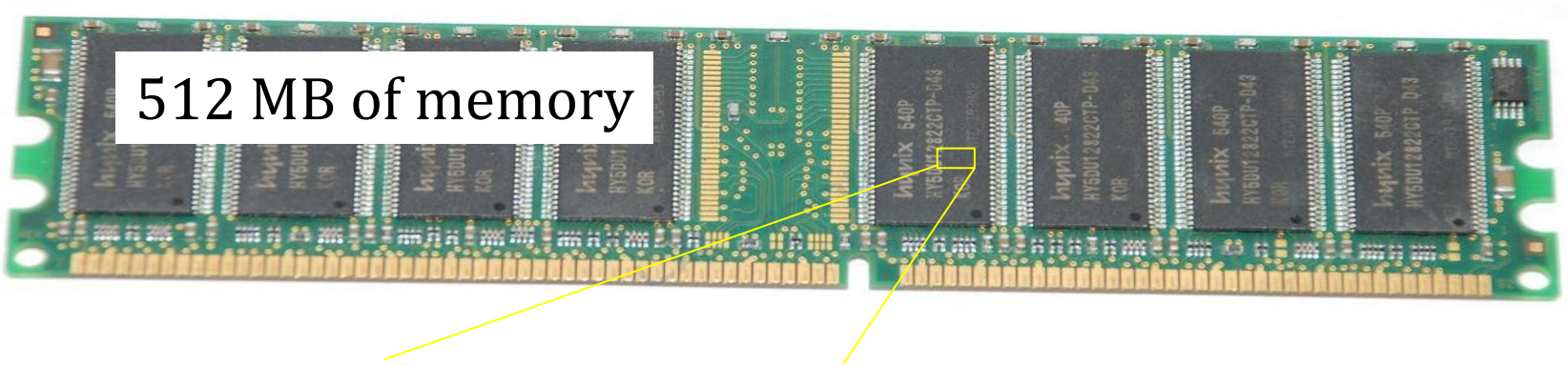
Join me,
in the
machine!

Memory!

Random Access Memory



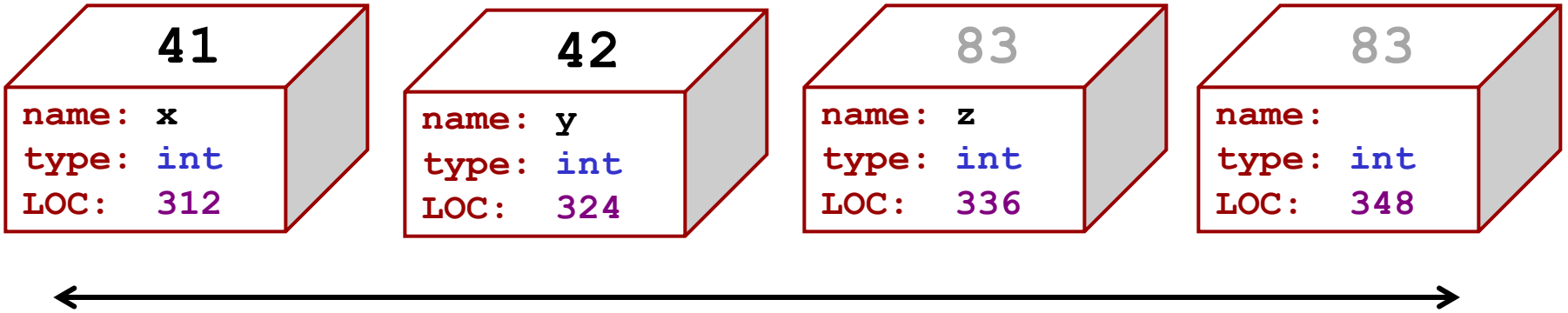
a big line of boxes, each with a name, type, location, and value



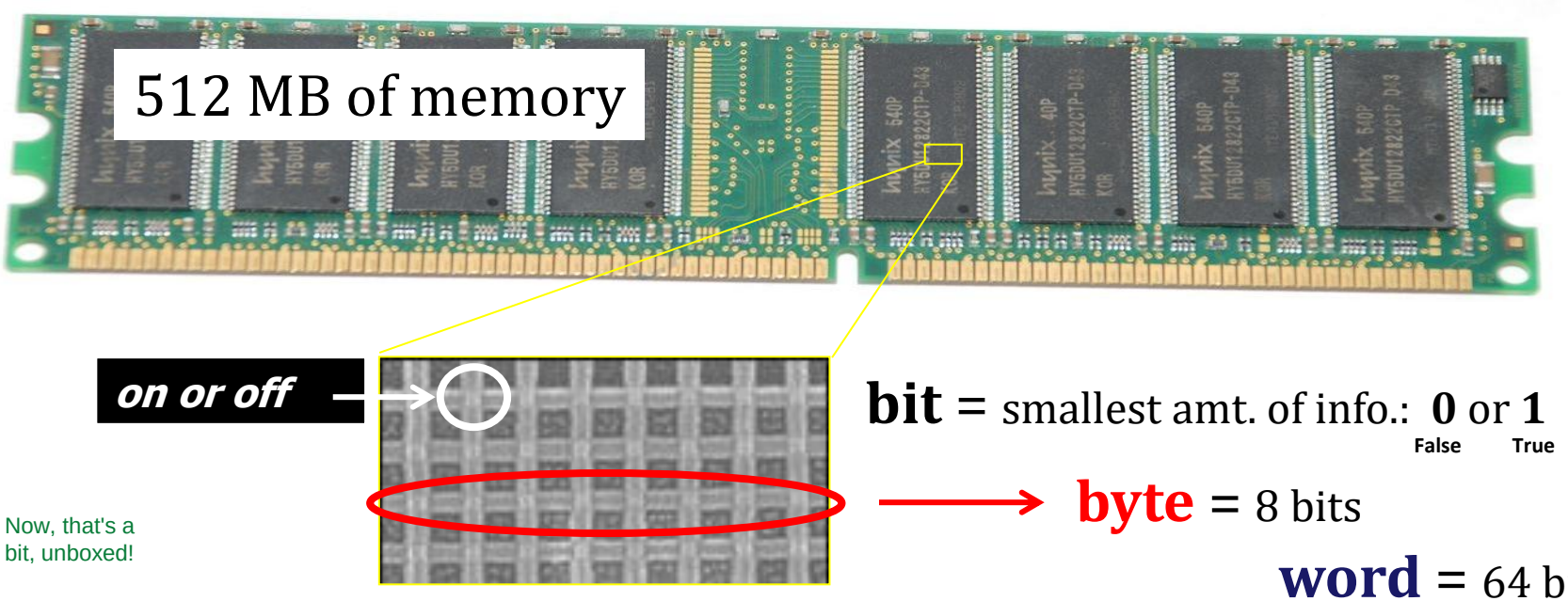
Deeper?

Memory!

Random Access Memory



a big line of boxes, each with a name, type, location, and value



bit = smallest amt. of info.: **0** or **1**
False True

byte = 8 bits

word = 64 bits

Now, that's a bit, unboxed!

All languages use ***types***

Type	Example	What *is* it?
`float`	`3.14` or `3.0`	numeric values with a fractional part, *even if the fractional part is .0*
`int`	`42` or `10**100`	integers – Python has *"infinite" precision ints!*
`bool`	`True` or `False`	the T/F results from a test or comparison: **==, !=, <, >, <=, >=**

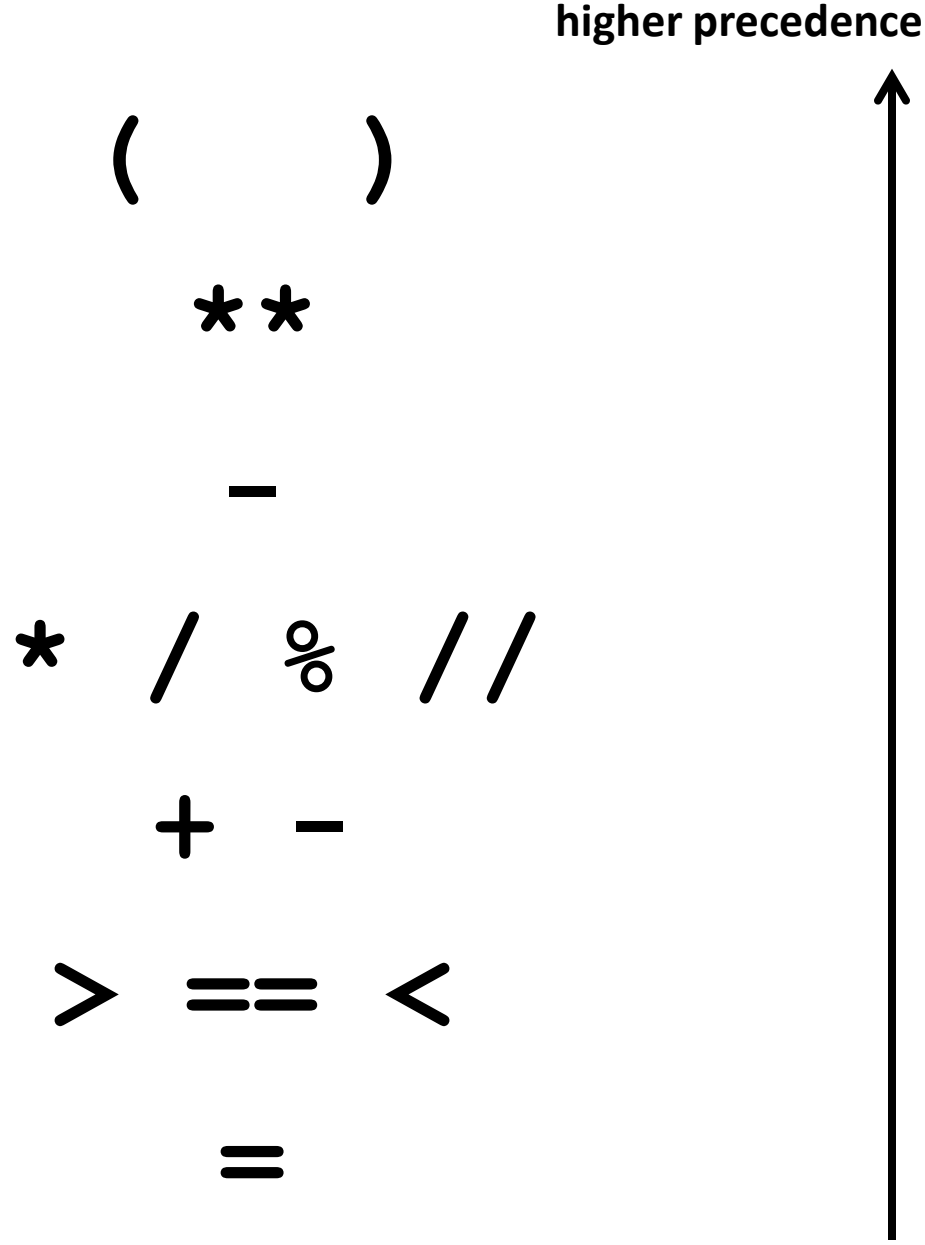
"Boolean values"

"Boolean operators"

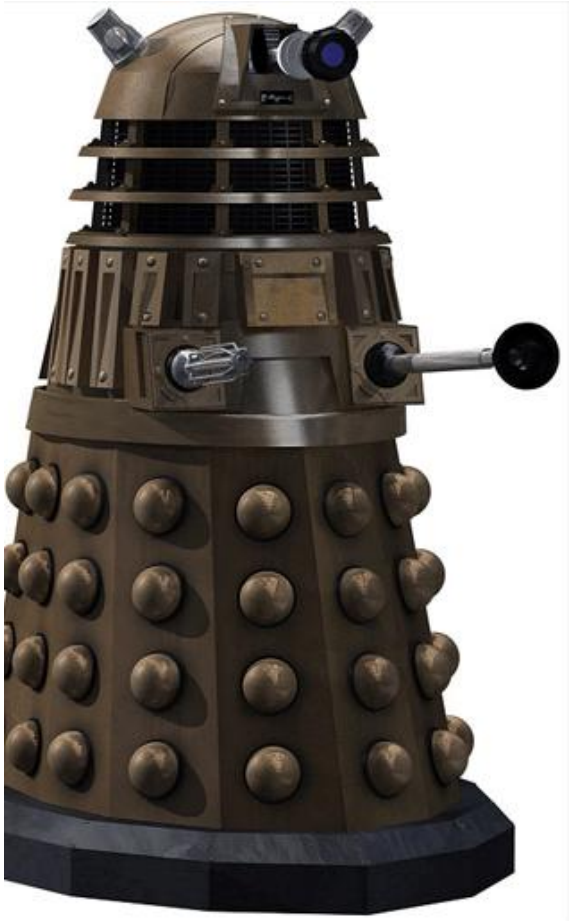
George Boole

type(x)

Operate!



Operate!



higher precedence

( )

**

-

* / % //

+ -

> == <

=

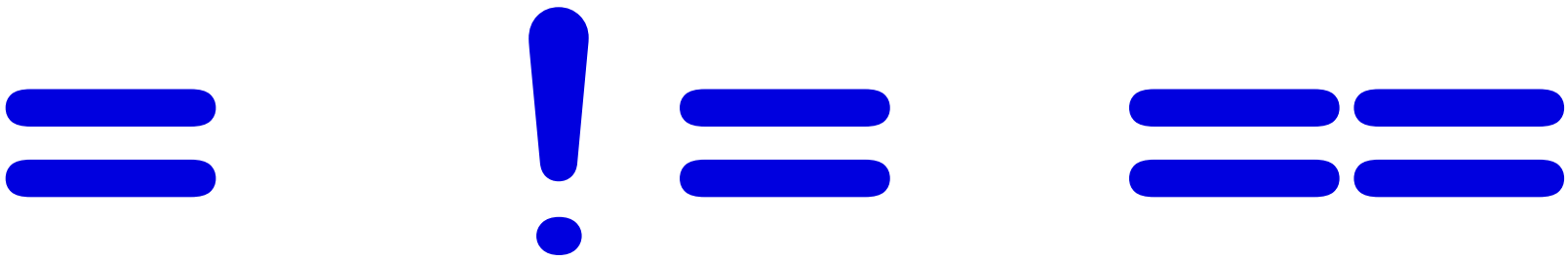
Python operators

higher precedence

parens	( )
power	**
negate	-
times, mod, divide	* / % //
add, subtract	+ -
compare	> == <
assign	=

It's not worth remembering all these `%+/*` things!
I'd recommend parentheses over precedence.

the "equals" operators



This is true – *but what is it saying!?*

the "equals" operators

=

SET
(make equal to)

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

!=

isn't equal to

==

TEST equals

```
x == 42
y == 42
```

the "equals" operators

I want === !

=

SET
(make equal to)

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

!=

isn't equal to

==

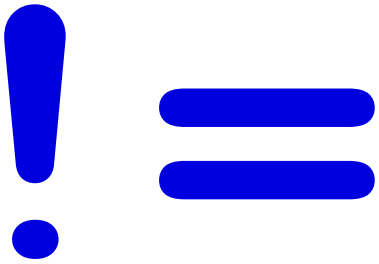
TEST equals

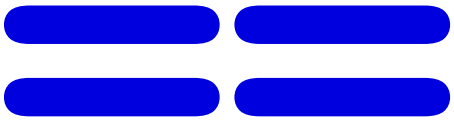
```
x == 42
```
False

```
y == 42
```
True

the "equals" operators

I want === !





SET
(make equal to)

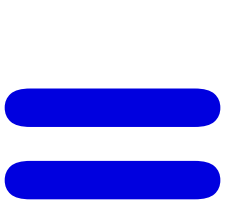
isn't equal to

TEST equals

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

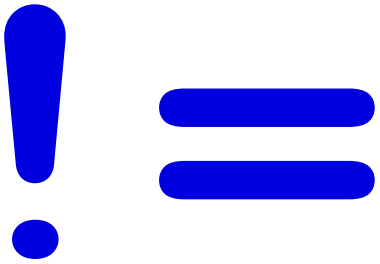
```
x != 42
y != 42
```

the "equals" operators



SET
(make equal to)

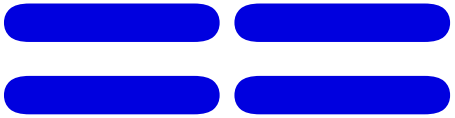
```
x = 41
y = x+1
```



isn't equal to

```
x != 42
```
True

```
y != 42
```
False



TEST equals

`%` the ***mod*** operator

```
7 % 3
```

```
9 % 3
```

```
8 % 3
```

```
30 % 7
```

`x%y` is the ***remainder*** when **`x`** is divided by **`y`**

For what values of **`x`** are these **`True`**?

- `x%2 == 0` ← What values x make this test True?
- `x%2 == 1` ← What values x make this test True?
- `x%4 == 0` ← If x is a year, what happens on these years!?
- `x%4 == 3` ← What happens on these years, *football-wise*!?

% the ***mod*** operator

```
7 % 3
```

```
9 % 3
```

```
8 % 3
```

```
30 % 7
```

`x%y` is the ***remainder*** when **`x`** is divided by **`y`**


```
x%2 == 0
```

What values x make this test True?

```
x%2 == 1
```

What values x make this test True?

```
x%4 == 0
```

If x is a year, what happens on these years!?

```
x%4 == 3
```

What happens on these years, *football-wise*!?

`//` *integer* division

```
7 // 3
```

```
8 // 3
```

```
9 // 3
```

```
30 // 7
```

`x//y` is **`x/y`**, *rounded-down* to an integer

`//` *integer* division

```
7 // 3
```

```
8 // 3
```

```
9 // 3
```

```
30 // 7
```

`x//y` is **`x/y`**, *rounded-down* to an integer

Why?

Decomposition of 30 into 7's:

```
30 ==  (4)*7  +  (2)
```

Decomposition of x into y's:

```
x ==  (x//y)*y  +  (x%y)
```

# of full y's in x

remainder after "taking" all of the full y's in x

how = works "Quiz"

Name(s) _____

Run these lines

```
x = 41
y = x + 1
z = x + y
```

What are **x**, **y**, and **z** at this time?

x	y	z

Then run this line

```
x = x + y
```

What are **x**, **y**, and **z** at this time?

x	y	z

Try it!

```
a = 11//2
b = a%3
c = b** a+b *a
```

What are the values of **a**, **b**, and **c** after the 3 lines, at left, run?

a	b	c

Inside the machine…

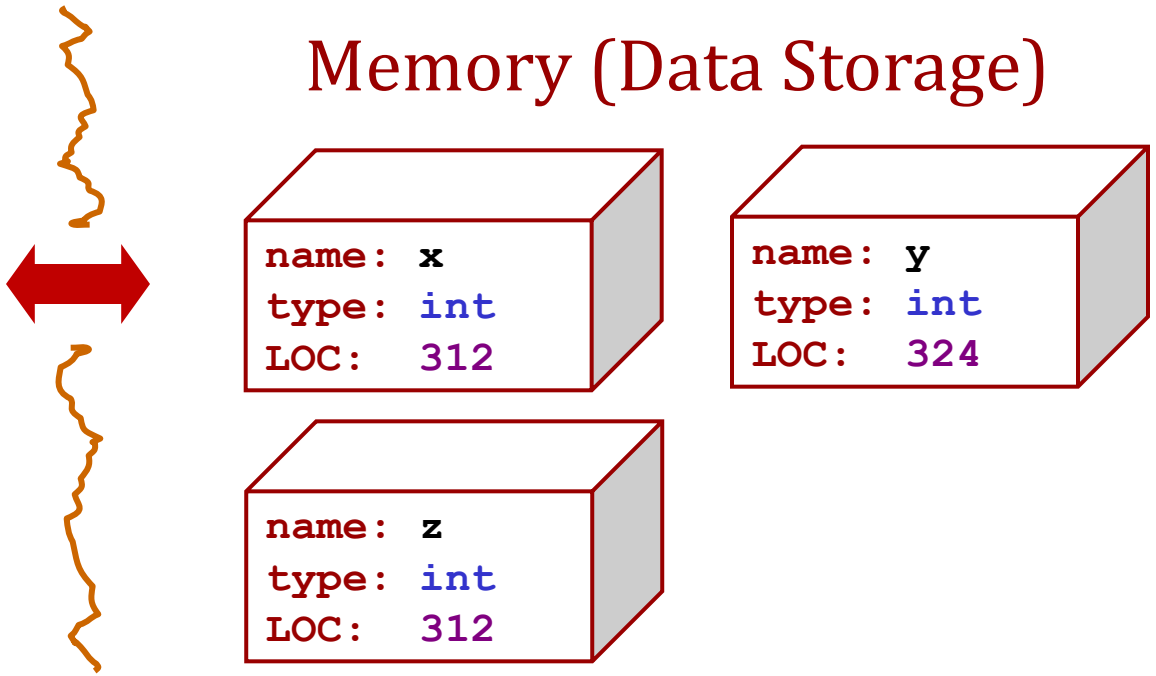
```
x = 41
y = x + 1
z = x + y
x = x + y
```

What's happening in python:

What's happening behind the scenes (in memory):

Computation

Memory (Data Storage)



id, del

Inside the machine…

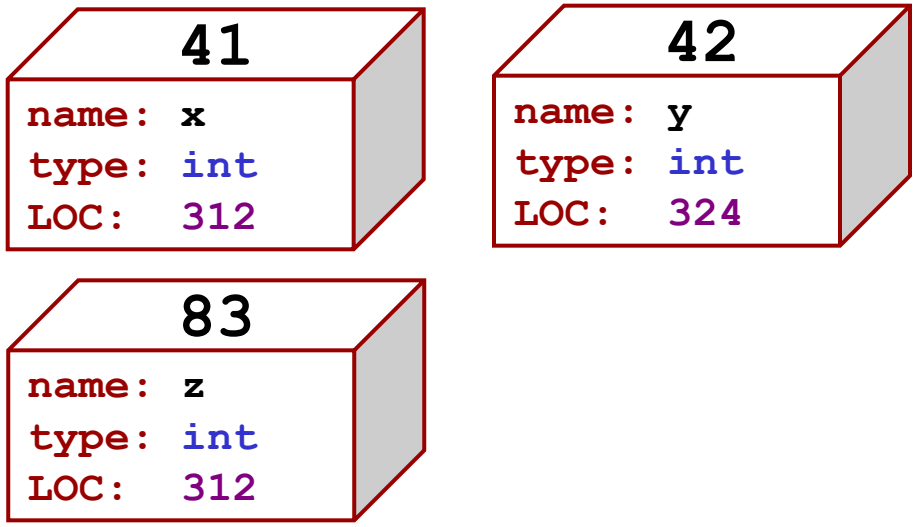
```
x = 41
y = x + 1
z = x + y
x = x + y
```

What's happening in python:

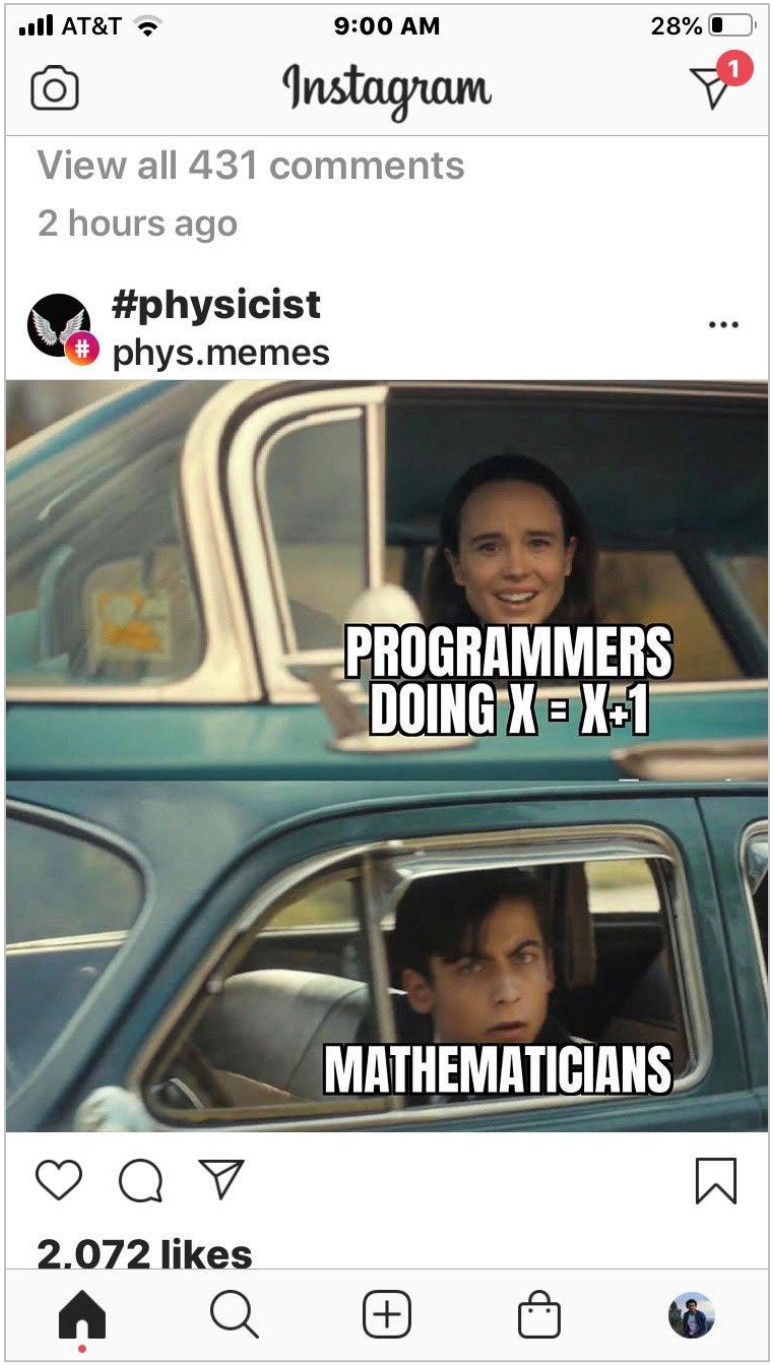
What's happening behind the scenes (in memory):

Computation

Memory (Data Storage)



id, del



CS ~ names are "current data" (really, current *state*)

they're changing all the time – *intentionally* – and their behavior is their purpose

Math ~ names are concepts

they're consistent – *intentionally* – and their inherent relationships are their purpose

[Thank you, Lucas!]

how = works

"Quiz"

Most of the solutions...

Run these lines

```
x = 41
y = x + 1
z = x + y
```

What are **x**, **y**, and **z** at this time?

x	y	z
41	42	83

Then run this line

```
x = x + y
```

What are **x**, **y**, and **z** at this time?

x	y	z
83	42	83

Try it!

```
a = 11//2
b = a%3
c = b** a+b *a
```

What are the values of **a**, **b**, and **c** after the 3 lines, at left, run?

a	b	c
5	2	??

Popular culture [edit]

The Hitchhiker's Guide to the Galaxy [edit]

The number 42 is, in *The Hitchhiker's Guide to the Galaxy* by Douglas Adams, the "Answer to the Ultimate Question of Life, the Universe, and Everything", calculated by an enormous supercomputer named Deep Thought over a period of 7.5 million years. Unfortunately, no one knows what the question is. Thus, to calculate the Ultimate Question, a special computer the size of a small planet was built from organic components and named "Earth". The Ultimate Question "What do you get when you multiply six by nine"[17] was found by Arthur Dent and Ford Prefect in the second book of the series, *The Restaurant at the End of the Universe*. This appeared first in the radio play and later in the novelization of *The Hitchhiker's Guide to the Galaxy*. The fact that Adams named the episodes of the radio play "fits", the same archaic title for a chapter or section used by Lewis Carroll in "The Hunting of the Snark", suggests that Adams was influenced by Carroll's fascination with and frequent use of the number. The fourth book in the series, the novel *So Long, and Thanks for All the Fish*, contains 42 chapters. According to the novel *Mostly Harmless*, 42 is the street address of Stavromula Beta. In 1994 Adams created the *42 Puzzle*, a game based on the number 42.

The Answer to the Ultimate Question of Life, The Universe, and Everything.

among *many* 42 references...

mostly in cs5...!

Are numbers enough for ***everything***?

Yes and no…

You need ***lists*** of numbers, as well!

and ***strings*** - lists of characters - too.

Both of these are Python ***sequences…*** 

strings: *textual* data

strings

```
s = 'scripps'
c = 'college'
```

type...

```
type(s)
```

len

```
len(s)
```

add!

```
s + c
```

multiply!!

```
2*s + 3*c
```

strings: *textual* data

Given
```
s1 = 'ha'
s2 = 't'
```

What are `s1 + s2`

`2*s1 + s2 + 2*(s1+s2)`

What did you say!?!

strings: *textual* data

Given
```
s1 = 'ha'
s2 = 't'
```

What are `s1 + s2`

hat

`2*s1 + s2 + 2*(s1+s2)`

hahathathat

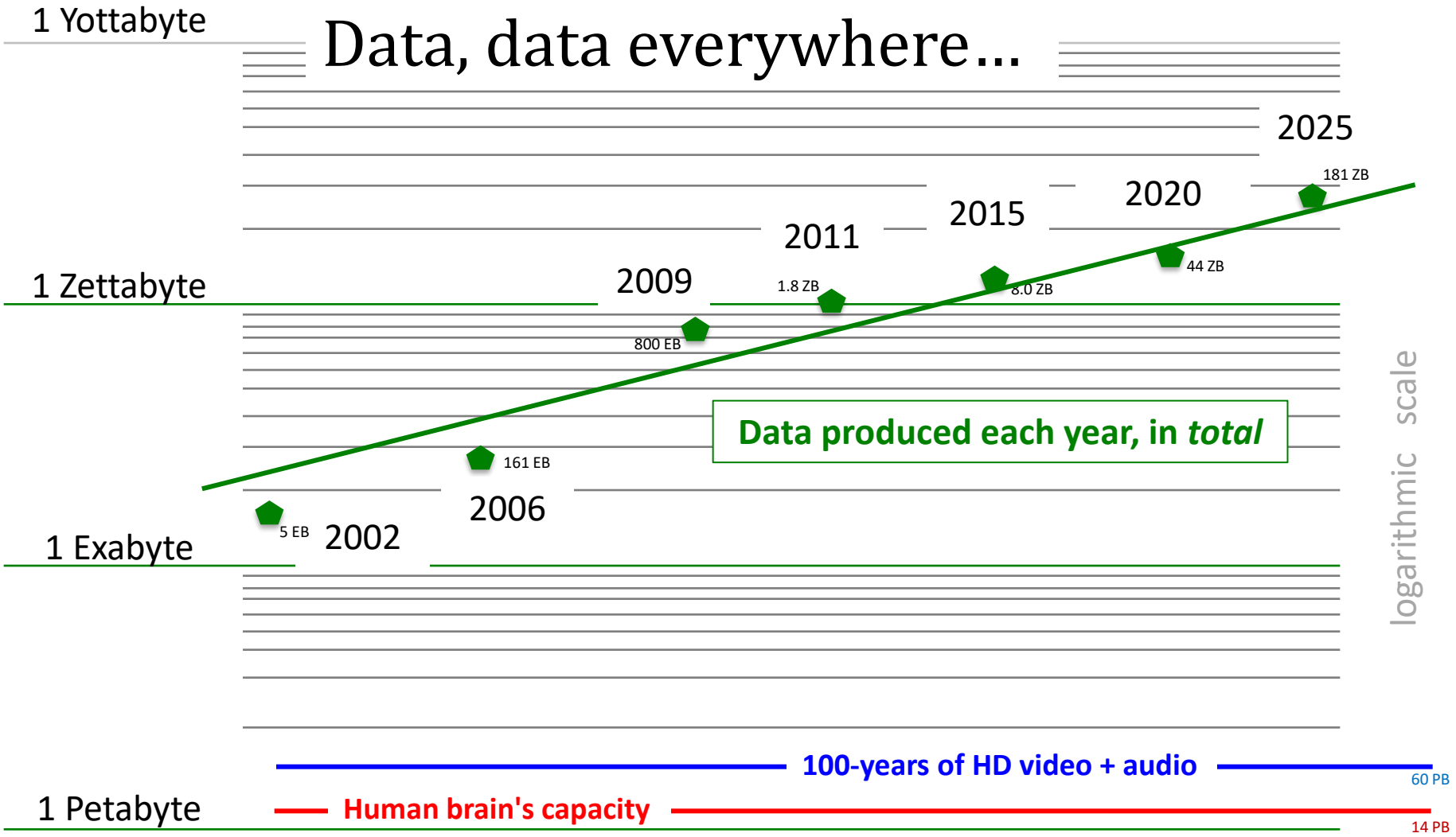
What did you say!?!

Data

Big Data?

Data, data everywhere…

alotta-bytes?



1 Petabyte, PB == 1000 Terabytes, TB

1 Terabyte, TB == 1000 Gigabytes, GB

References

(2025) 16-175ZB: https://seedscientific.com/how-much-data-is-created-every-day/
(2020) 44ZB: http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm
(2015) 8 ZB: http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf
(2011) 1.8 ZB: http://www.emc.com/leadership/programs/digital-universe.htm
(2009) 800 EB: http://www.emc.com/collateral/analyst-reports/idc-digital-universe-are-you-ready.pdf
(2006) 161 EB: http://www.emc.com/collateral/analyst-reports/expanding-digital-idc-white-paper.pdf
(2002) 5 EB: http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/execsum.htm
(2023) https://explodingtopics.com/blog/data-generated-per-day (estimate of 181zb in 2025)
(life in video) 60 PB: in 4320p resolution, extrapolated from 16MB for 1:21 of 640x480 video (w/sound) – almost certainly a gross overestimate, as sleep can be compressed significantly!
(brain) 14 PB: http://www.quora.com/Neuroscience-1/How-much-data-can-the-human-brain-store

Big Data?

McKinsey&Company

Careers

Global Locations

Big data: The next frontier for innovation, competition, and productivity

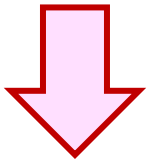
The New York Times

SundayReview | The Opinion Pages

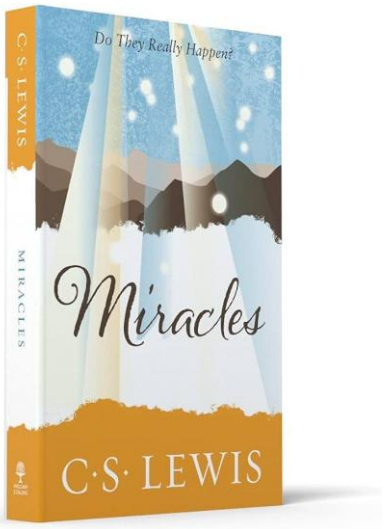
WORLD U.S. N.Y. / REGION BUSINESS TECHNOLOGY

NEWS ANALYSIS

Is Big Data an Economic Big Dud?



C.S. Lewis



wisdom

Lewis, et al. ⇧

knowledge

Google's users ⇧

information

Google ⇧

data

Data's *elevation*?

Lists ~ collections of *any* data

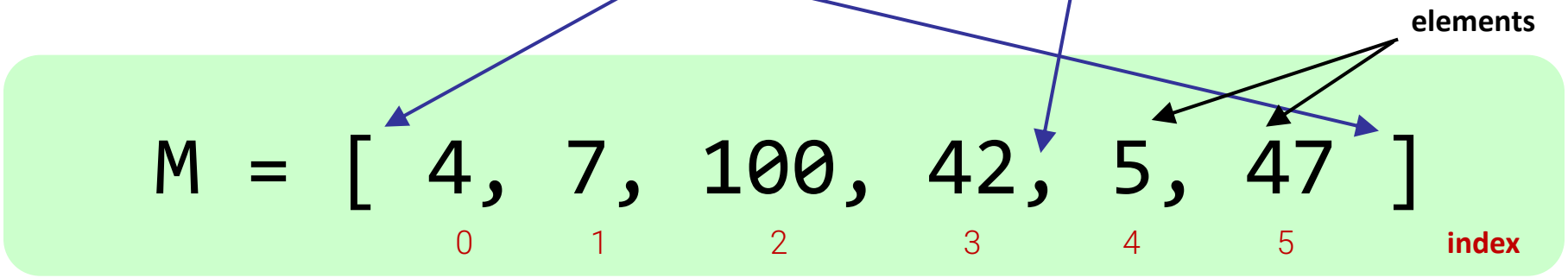
```
M = [ 4, 7, 100, 42, 5, 47 ]
```

Lists ~ collections of *any* data

syntax

Square brackets tell python you want a list.

Commas separate elements.



`len(M)`	`M[0]`	`M[0:3]`
top-level length	indexing	slicing

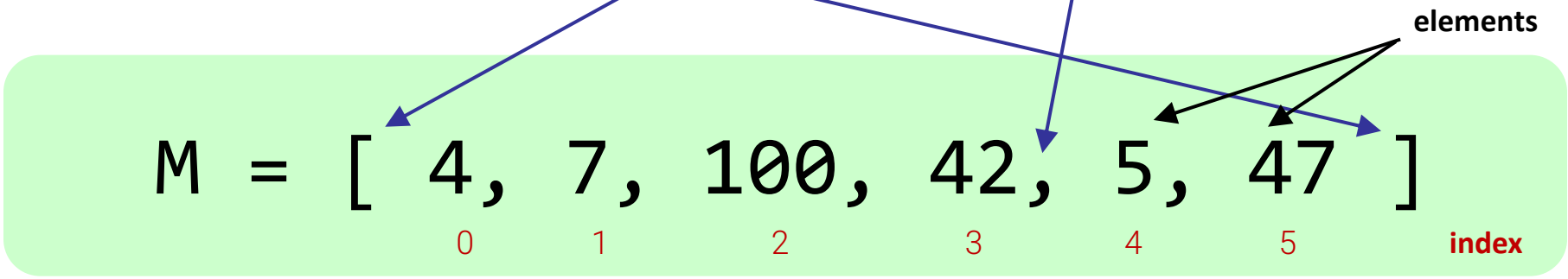
semantics

Lists ~ collections of *any* data

syntax

Square brackets tell python you want a list.

Commas separate elements.



`len(M)`	`M[0]`	`M[0:3]`
6	4	[4,7,100]
top-level length	indexing	slicing

semantics

Lists ~ collections of *any* data

string

```
L = [ 3.14, [2,40], 'third', 42 ]
      0       1        2      3    index
```

`len(L)`	`L[0]`	`L[3]`	`L[1]`	`L[0:1]`
top-level length	**indexing**	**indexing**	**indexing**	**slicing!**
*only counts **top-level** elements*	***always** returns an **element***	***always** returns an **element***	***always** returns an **element***	***always** returns a substructure!*

Lists ~ collections of *any* data

string

```
L = [ 3.14, [2,40], 'third', 42 ]
      0        1        2      3    index
```

`len(L)`

4

top-level length

*only counts **top-level** elements*

`L[2]`

indexing

always** returns an **element

`L[ ][ : ]`

`'hi'`

indexing + slicing

***C**omposition **S**cience!*

Indexing and Slicing!

Feeling ***positive***? Python!
Feeling ***negative***? Python!

```
          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
s = 'harvey mudd college'
   -19 | -17 | -15 | -13 | -11 | -9 | -7 | -5 | -3 | -1
     -18   -16   -14   -12   -10   -8   -6   -4   -2
```

```
s[0] == 'h'
s[17] == 'g'
s[8] == 'u'
s[1] == 'a'
s[19]   error!
s[6] == ____
```

```
s[-1] == 'e'
s[-2] == 'g'
s[-11] == 'u'
s[-6] == 'o'
s[-20]   error!
s[-0] == ____
```

Indexing

single-location *in a sequence*

Can *go out of bounds!*
Let's see that...

Indexing and Slicing!

Feeling ***positive***? Python!
Feeling ***negative***? Python!

```
          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
s = 'harvey mudd college'
      -19 -17 -15 -13 -11 -9  -7  -5  -3  -1
        -18 -16 -14 -12 -10 -8  -6  -4  -2
```

```
s[0] == 'h'
s[17] == 'g'
s[8] == 'u'
s[1] == 'a'
s[19]   error!
s[6] == ' '
```

```
s[-1] == 'e'
s[-2] == 'g'
s[-11] == 'u'
s[-6] == 'o'
s[-20]   error!
s[-0] == 'h'
```

Indexing

single-location *in a sequence*

Can *go out of bounds!*
Let's see that...

Indexing and Slicing!

Feeling ***positive***? Python!
Feeling ***negative***? Python!

```
 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18
 s  =  'harvey mudd college'
-19 -17 -15 -13 -11  -9  -7  -5  -3  -1
   -18 -16 -14 -12 -10  -8  -6  -4  -2
```

Slicing

two-index-subsequence

*Optional third value is the "**stride**"*

*Omit an index to say "**the end**"*

```
s[0:2] == 'ha'
s[15:18] == 'leg'
s[-2:] == 'ge'
s[:3] == 'har'
s[5:3] == ''
s[5:3:-1] == 'ye'
s[10:17:3] == _____
s[1::6] == _____
```

I love that last Latin one.

- C.S. Lewis

Indexing and Slicing!

Feeling ***positive***? Python!
Feeling ***negative***? Python!

```
 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18
 s  =  'harvey mudd college'
-19 -17 -15 -13 -11  -9  -7  -5  -3  -1
   -18 -16 -14 -12 -10  -8  -6  -4  -2
```

Slicing

two-index-subsequence

*Optional third value is the "**stride**"*

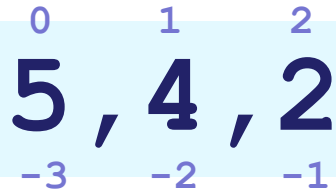
*Omit an index to say "**the end**"*

```
s[0:2] == 'ha'
s[15:18] == 'leg'
s[-2:] == 'ge'
s[:3] == 'har'
s[5:3] == ''
s[5:3:-1] == 'ye'
s[10:17:3] == 'doe'
s[1::6] == 'amo'
```

I love that last Latin one.

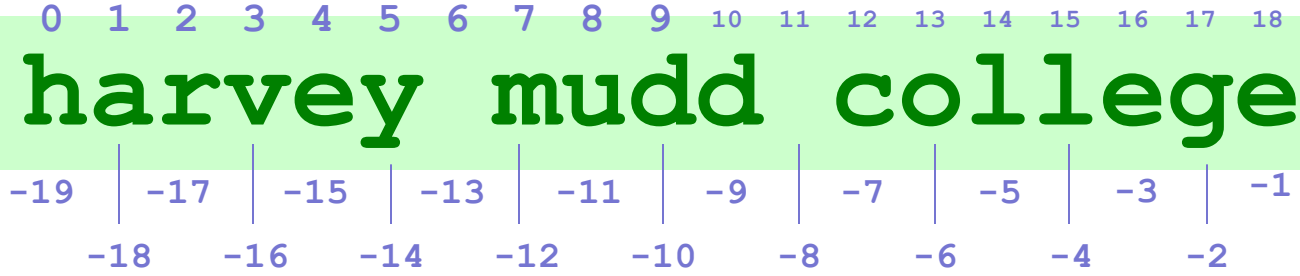
- C.S. Lewis

```
L = [5,4,2]
```



First + Rest

```
s = 'harvey mudd college'
```




```
L[0] ==
s[0] ==
L[1:] ==
s[1:] ==
```

```
L = [5,4,2]
```

0 1 2

-3 -2 -1

First + Rest

```
s = 'harvey mudd college'
```

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

-19 -17 -15 -13 -11 -9 -7 -5 -3 -1

-18 -16 -14 -12 -10 -8 -6 -4 -2

```
L[0] == 5
s[0] == 'h'
```

Index

"first"

```
L[1:] == [4,2]
s[1:] == 'arvey mudd college'
```

Slice

"rest"

```
pi = [3,1,4,1,5,9]
     0 1 2 3 4 5
    -6 -5 -4 -3 -2 -1
```

```
L = [ 'pi', "isn't", [4,2] ]
```

```
M = 'You need parentheses for chemistry !'
    0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34
```

Try it...

We <3 ()

Part 1

"first of pi"

What is `pi[0]` _

What is `pi[1:]` _____ "rest of pi"

_

What are `len(pi)`, `len(L)`, `len(L[1])`?

What is `pi[2:4]`

What slice of `pi` is `[3,1,4]`

What slice of `pi` is `[3,4,5]`

Part 2

What is `L[0]`

What is `L[0][1]`

What is `L[0:1]`

These three are all different

What slice of `M` is `'try'`

What slice of `M` is `'shoe'`

What is `M[9:15]`

What is `M[::5]`

Extra! Mind Muddlers

What are `pi[0]*(pi[1]+pi[2])` and `pi[0]*(pi[1:2]+pi[2:3])` ?

These two are different! ⇨

```
pi = [3,1,4,1,5,9]
     0 1 2 3 4 5
    -6 -5 -4 -3 -2 -1
```

```
L = [ 'pi', "isn't", [4,2] ]
```

```
M = 'You need parentheses for chemistry !'
    0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34
```

Try it...

We <3 ()

Part 1

What is `pi[0]` **3**

What is `pi[1:]` **[1,4,1,5,9]**

6

What are `len(pi)`, `len(L)`, `len(L[1])`?

What is `pi[2:4]`

What slice of `pi` is `[3,1,4]`

What slice of `pi` is `[3,4,5]`

Part 2

What is `L[0]`

What is `L[0][1]`

What is `L[0:1]`

These three are all different

What slice of `M` is `'try'`

What slice of `M` is `'shoe'`

What is `M[9:15]`

What is `M[::5]`

Extra! Mind Muddlers

What are `pi[0]*(pi[1]+pi[2])` and `pi[0]*(pi[1:2]+pi[2:3])` ?

These two are different!

```
pi = [3,1,4,1,5,9]
     0 1 2 3 4 5
    -6 -5 -4 -3 -2 -1
```

```
L = [ 'pi', "isn't", [4,2] ]
```

```
M = 'You need parentheses for chemistry !'
    0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34
```

Tried!

We <3 ()




Part 1

What is `pi[0]` **3**

What is `pi[1:]` **[1,4,1,5,9]**

6 **3** **5**

What are `len(pi)`, `len(L)`, `len(L[1])`?

What is `pi[2:4]` **[4, 1]**

What slice of `pi` is `[3,1,4]` **pi[0:3]**

What slice of `pi` is `[3,4,5]` **pi[0:5:2]** or **pi[0::2]**

Part 2

What is `L[0]` **'pi'**

What is `L[0][1]` **'i'**

What is `L[0:1]` **['pi']**

These three are all different

What slice of `M` is `'try'` **M[31:34]** or **M[-5:-2]** or ...

What slice of `M` is `'shoe'`

What is `M[9:15]` **'parent'**

What is `M[::5]`

Extra! Mind Muddlers

What are `pi[0]*(pi[1]+pi[2])` and `pi[0]*(pi[1:2]+pi[2:3])` ?

These two are different!

3*(1+4) == **15**

3*([1]+[4]) == 3*[1,4] == **[1,4,1,4,1,4]**

```
pi = [3,1,4,1,5,9]
     0 1 2 3 4 5
    -6 -5 -4 -3 -2 -1
```

Tried!

We <3 ()

```
L = [ 'pi', "isn't", [4,2] ]
```

```
M = 'You need parentheses for chemistry !'
    0  2  4  6  8  10 12 14 16 18 20 22 24 26 28 30 32 34
```

Part 1

What is `pi[0]` **3**

What is `pi[1:]` **[1,4,1,5,9]**

6 **3** **5**

What are `len(pi)`, `len(L)`, `len(L[1])`?

What is `pi[2:4]` **[4, 1]**

What slice of `pi` is `[3,1,4]` **pi[0:3]**

What slice of `pi` is `[3,4,5]` **pi[0:5:2]** or **pi[0::2]**

Part 2

What is `L[0]` **'pi'**

What is `L[0][1]` **'i'**

What is `L[0:1]` **['pi']**

These three are all different

What slice of `M` is `'try'` **M[31:34]** or **M[-5:-2]** or ...

What slice of `M` is `'shoe'` M[30:17:-4] M[-6:-19:-4]

What is `M[9:15]` **'parent'**

What is `M[::5]` **'Yeah cs!'**

Extra! Mind Muddlers

What are `pi[0]*(pi[1]+pi[2])` and `pi[0]*(pi[1:2]+pi[2:3])` ?

These two are different! ⇨

3*(1+4) == **15**

3*([1]+[4]) == 3*[1,4] == **[1,4,1,4,1,4]**

Python slices - it dices...

(**data**, at least)

... *but wait*, there's more!

Python slices - it dices...

(**data**, at least)

Python
functions

... *but wait,* there's more!

Functioning in Python

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    return 2x
```

This doesn't ***look*** quite right...

Functioning in Python

```
1  #
2  # Putting the "fun" into Python functions!
3  #
4
5  def dbl( x ):
6      """ returns double its input, x """
7      return 2x
8
```

More *visibly* broken… !

Functioning in Python

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    return 2*x
```

comment for other ***coders***

documentation string for all ***users***

Python's keywords

Some of Python's *baggage*...

Function *Fun* !

```
def adjectify(s):
    """ makes its input an adjective """
    return s + '-tastic'
```

```
In[1] adjectify('cs5')
'cs5-tastic
```

strings, lists, numbers ...
*all **data** are fair game*

Have a cs5-tastic Week!

and Semester, too

This week's lab ~

*first **two** hw problems*