Python!



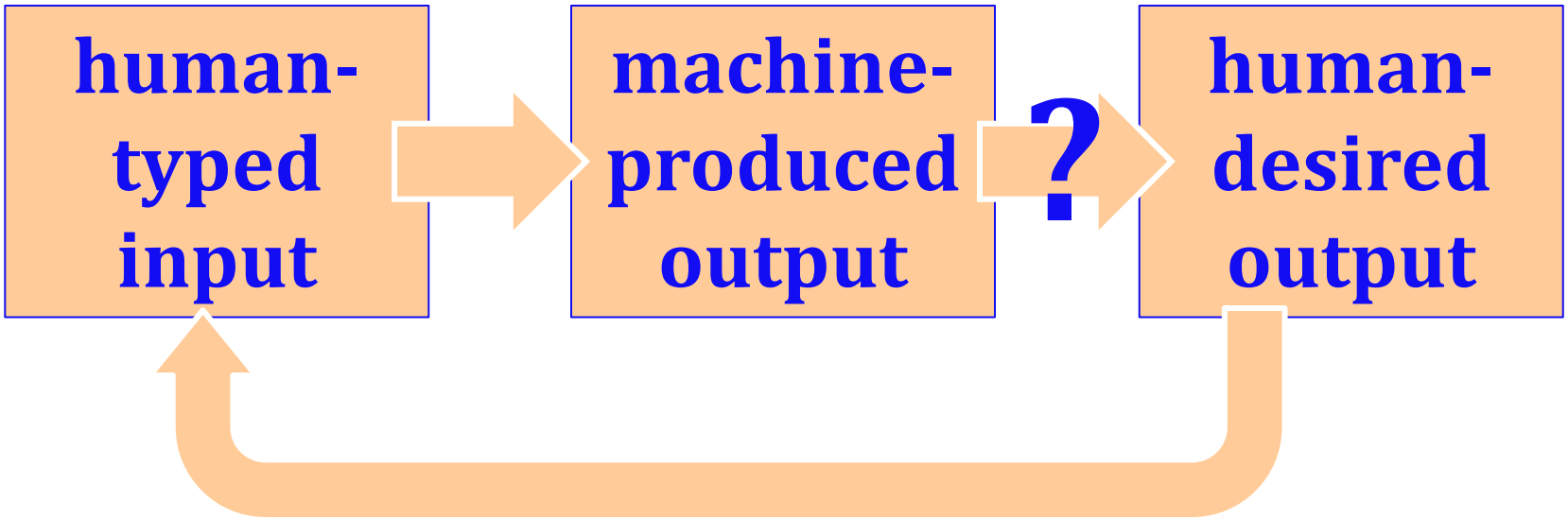
One possible relationship...



Happy co-existence…
It can even be comfy!

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*** 

learning how machines ***think***!

Starting Python

- Use python3 by default
  - `alias python=python3`
- You should see the following:

```
jawaterman@clove:~$ python
Python 3.6.5 (default, Apr  1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Python operators

higher precedence

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



It's hard to remember all these `%+/*` things! At first, prefer parentheses over precedence.

% 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`**
- **`x % 2 == 1`**
- **`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

Decomposition of 31 into 7's:

31 == (4) * 7 + (3)

But why?

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

Naming Things

- Python lets you give names to the results of calculations (values)

```
>>> pi = 3.145926
>>> pi * 10
31.45926
```

  - Left hand side (of the = sign) is always a name
  - Right hand side (of the = sign)is always a value
  - We call these named values **variables**

What's in a name?

- Variables must start with a letter
- Can only use letters and numbers
  - Why?
- Variables are case sensitive
  - Pi and PI are two **different** variables
- Python has special words that can't be used as variable names
  - You'll get a Python error if you try to assign something to them

All languages use ***data 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"

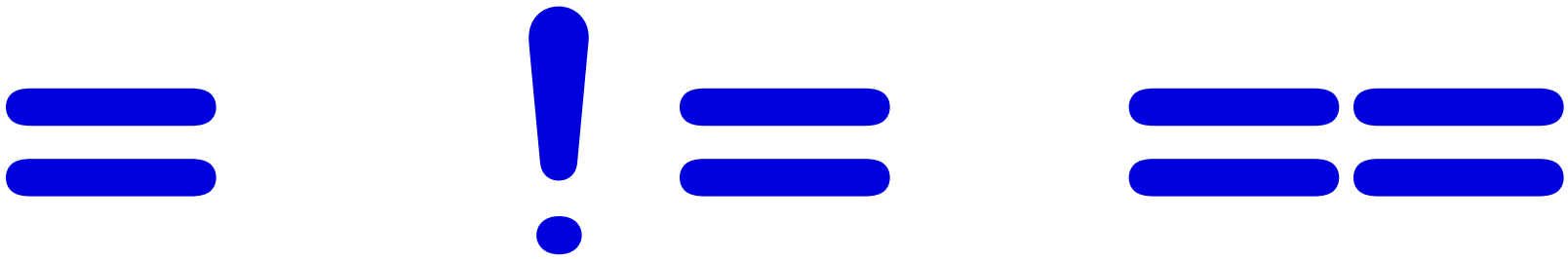
"Boolean operators"



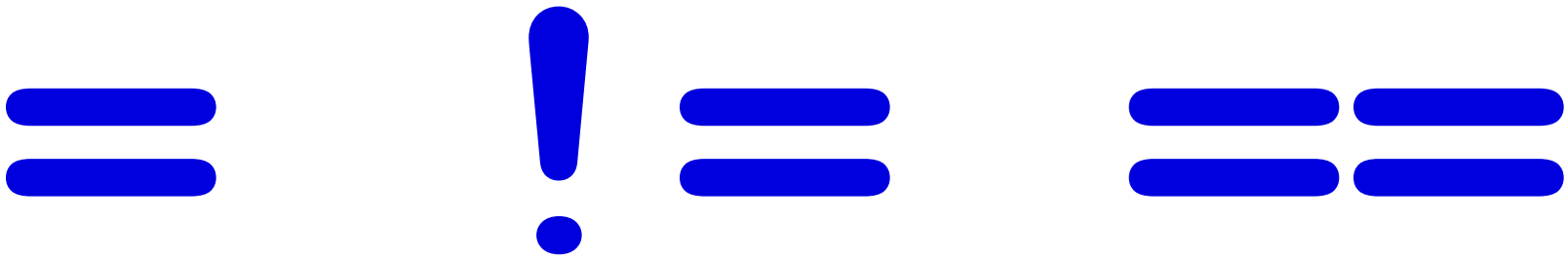
George Boole

type(x)

the "equals" operators



the "equals" operators



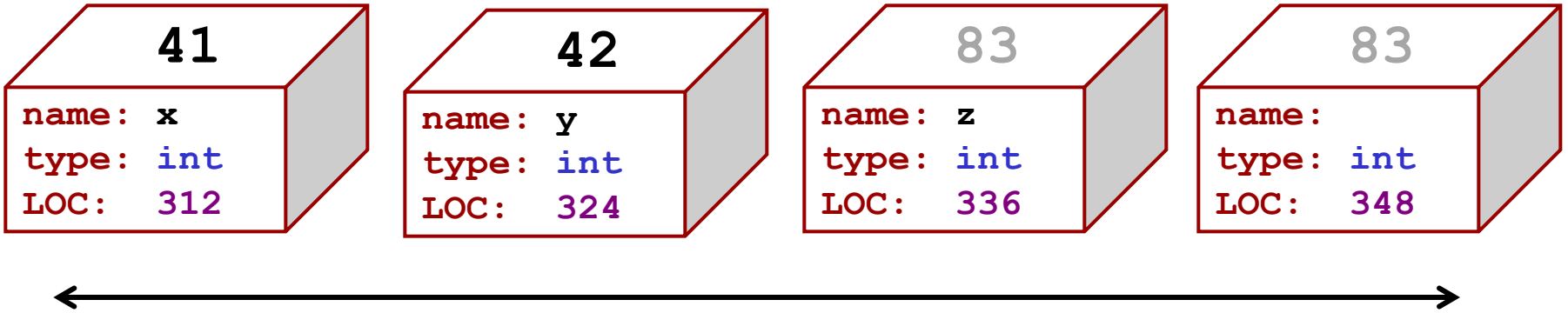
assignment **isn't** equal to **TEST** equals

I want === !

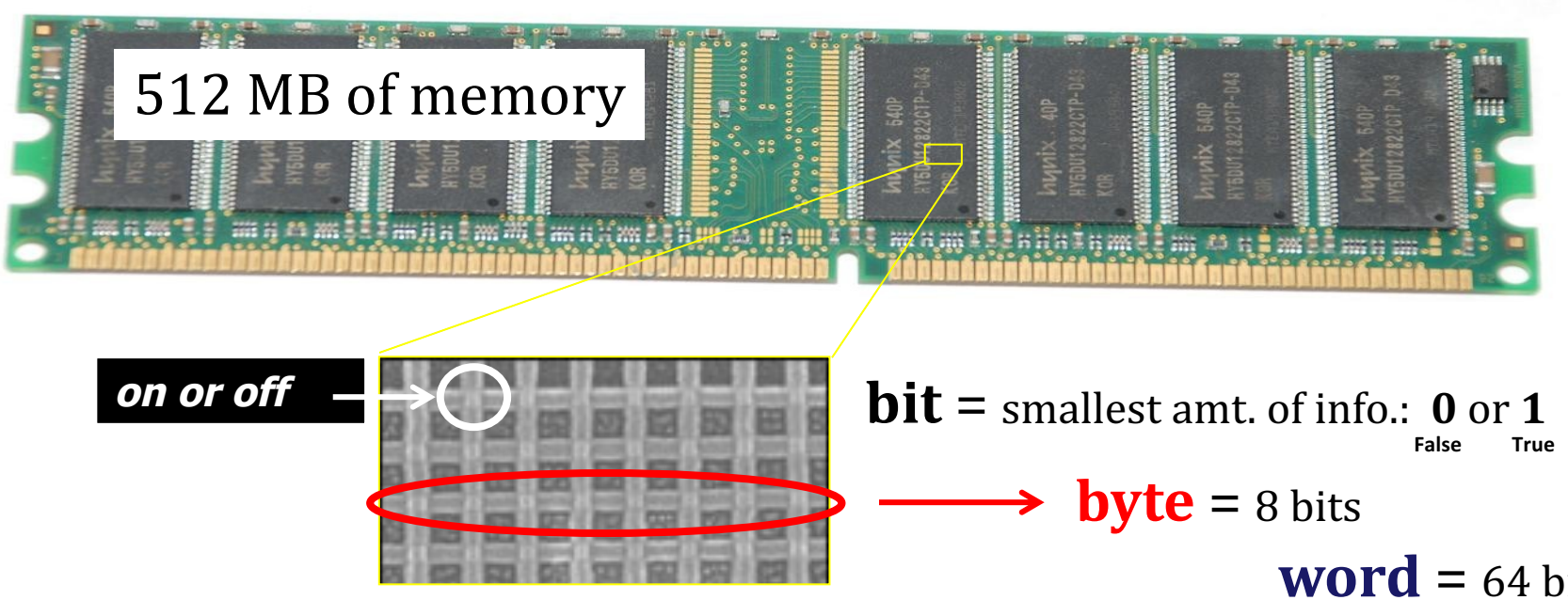


Memory!

Random Access Memory



a big list 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

how = works

"Quiz Yourself"

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

Extra!

```
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
```

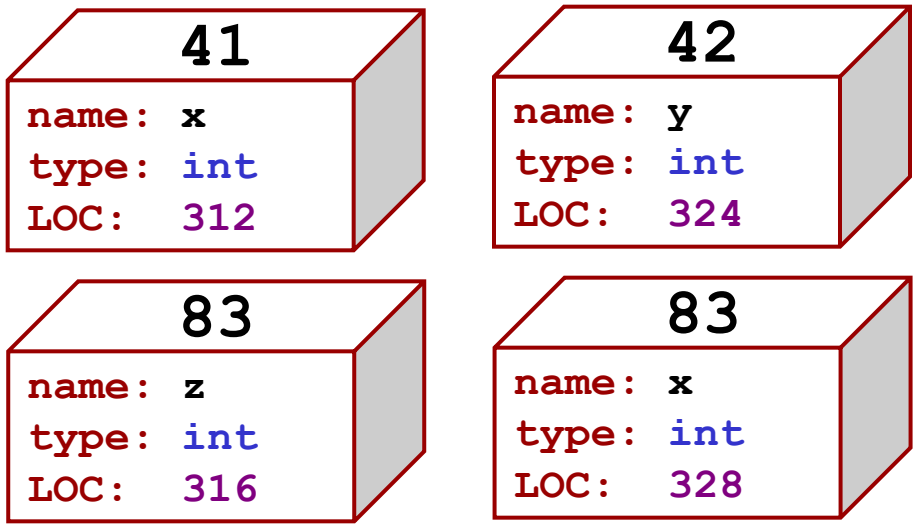
What's happening in python:

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

Computation



Memory (Data Storage)



id, del

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 = 'vassar'
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)
```

Logical operators

- and, or, not
- Similar to their meaning in English
  - x > 0 and x < 10
    - True only x is greater than 0 *and* at the same time, x is less than 10
  - x > 0 or x < 10
    - True if x is greater than 0 *or* if x is less than 10
    - Entire statement is true if either condition holds
  - not x > y
    - is true if x > y is false, that is if x is less than or equal to y

Conditional Executation

- The syntax for an `if` statement

```
if Boolean_Expression:
    Statements_True
else:
     Statements_False
```

- The else clause is optional

Chained conditionals

```
if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x and y must be equal")
```

Chained conditionals

- elif is short for “else if”
- Exactly one branch will be executed
- No limit to the number elif statements
- Only a single else statement
  - Must be the last branch

Statement Groups

- How does Python know which statements belong to which clause?

Nesting

for *decision-making*, we now have it ***all***…



Nesting

- Statements that have the same level of indentation (nesting) are grouped together
- Vscode makes this easy with the tab key