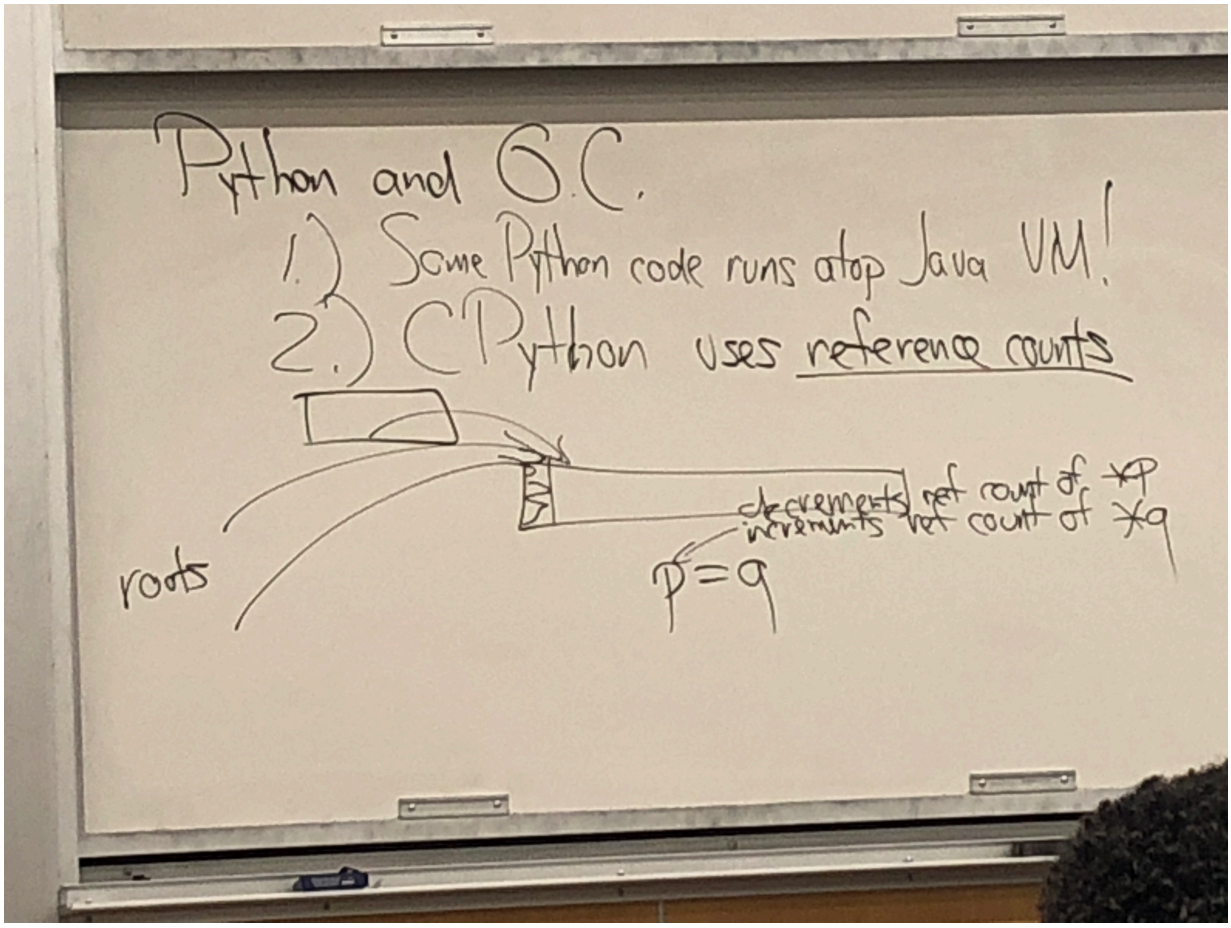
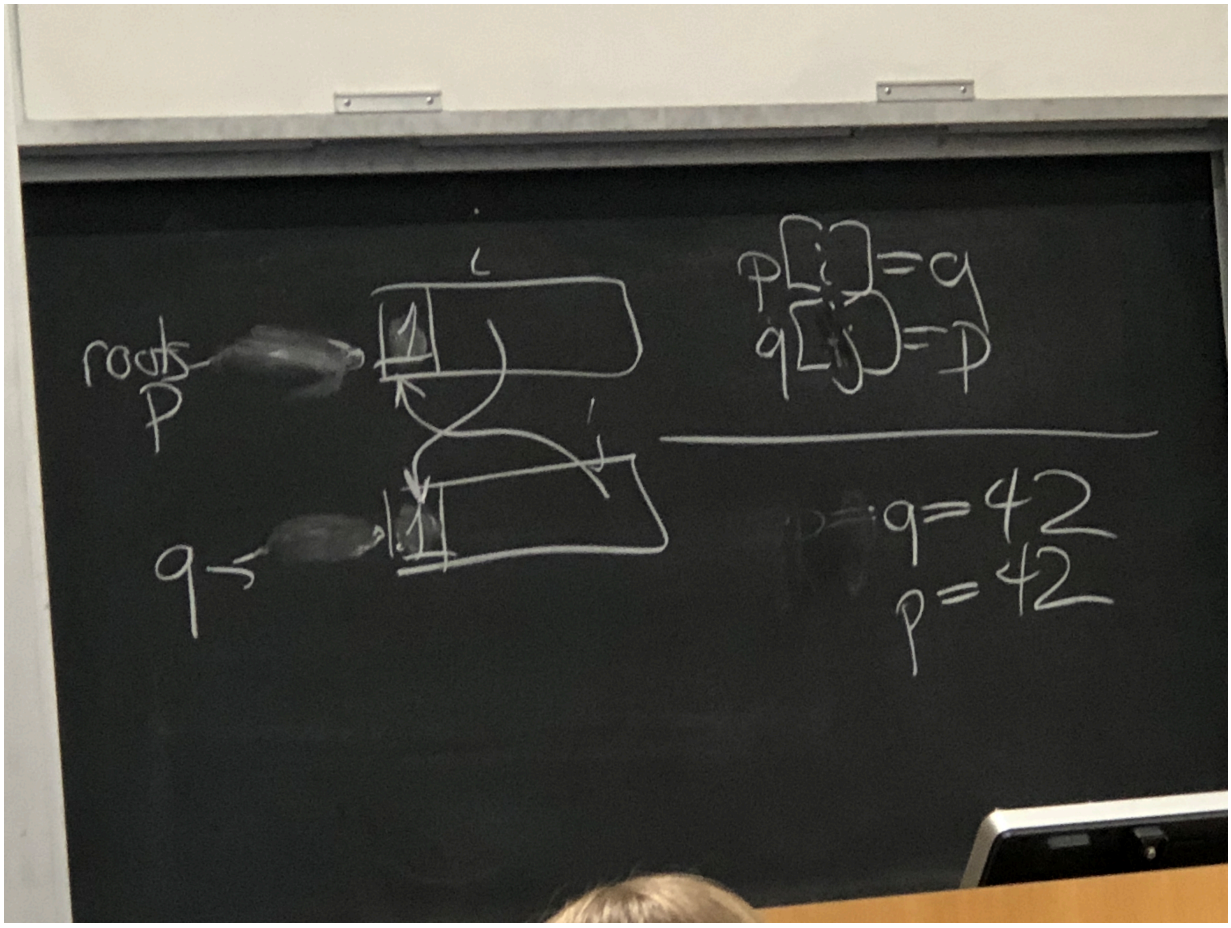


-slows down assignment
+zero refcount -> free immediately

PROBLEM: Refcounts can have cycles

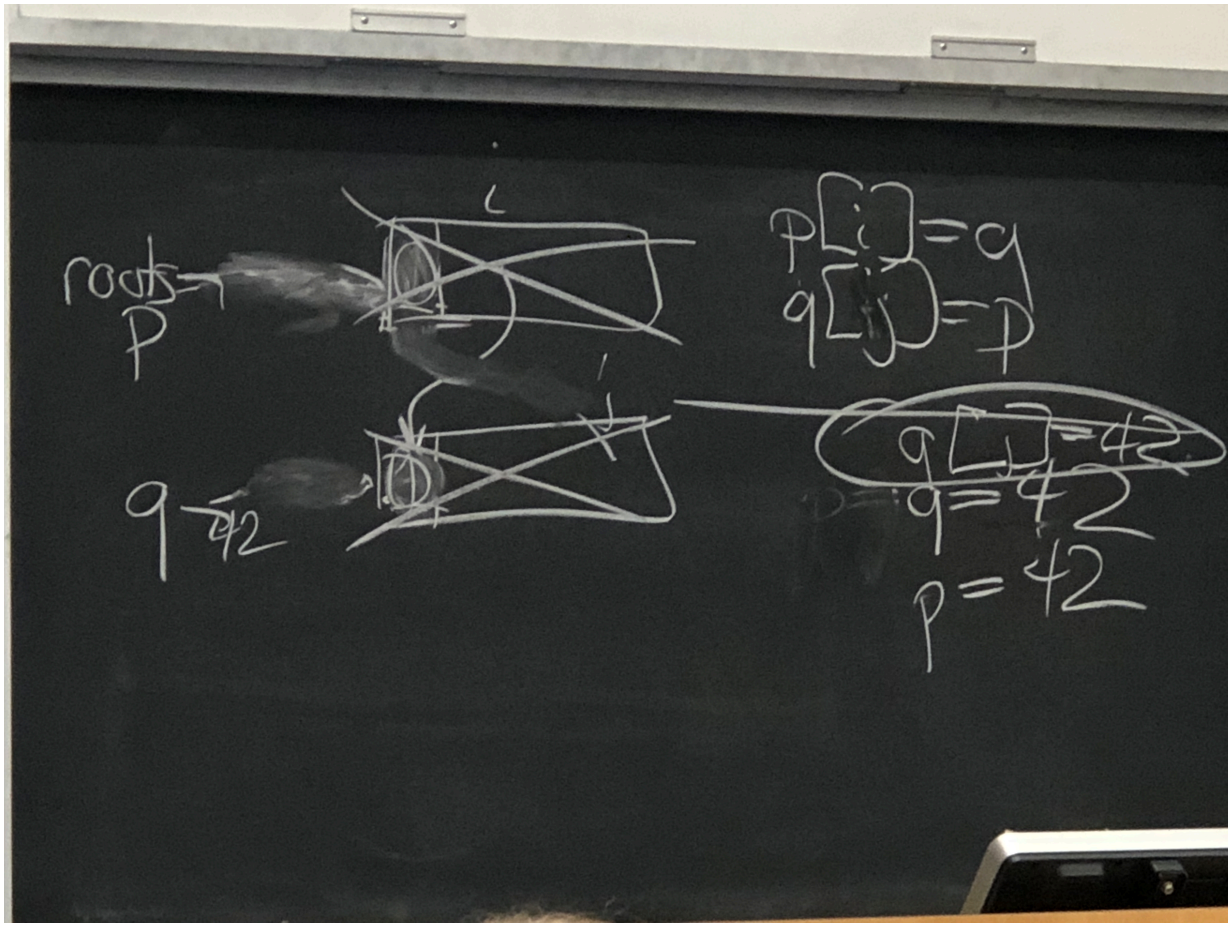
Refcount cycles

- p and q will first have refcounts of 2
- when reassigning p and q, their memory addresses change, decrement refcount
- but now its 1!

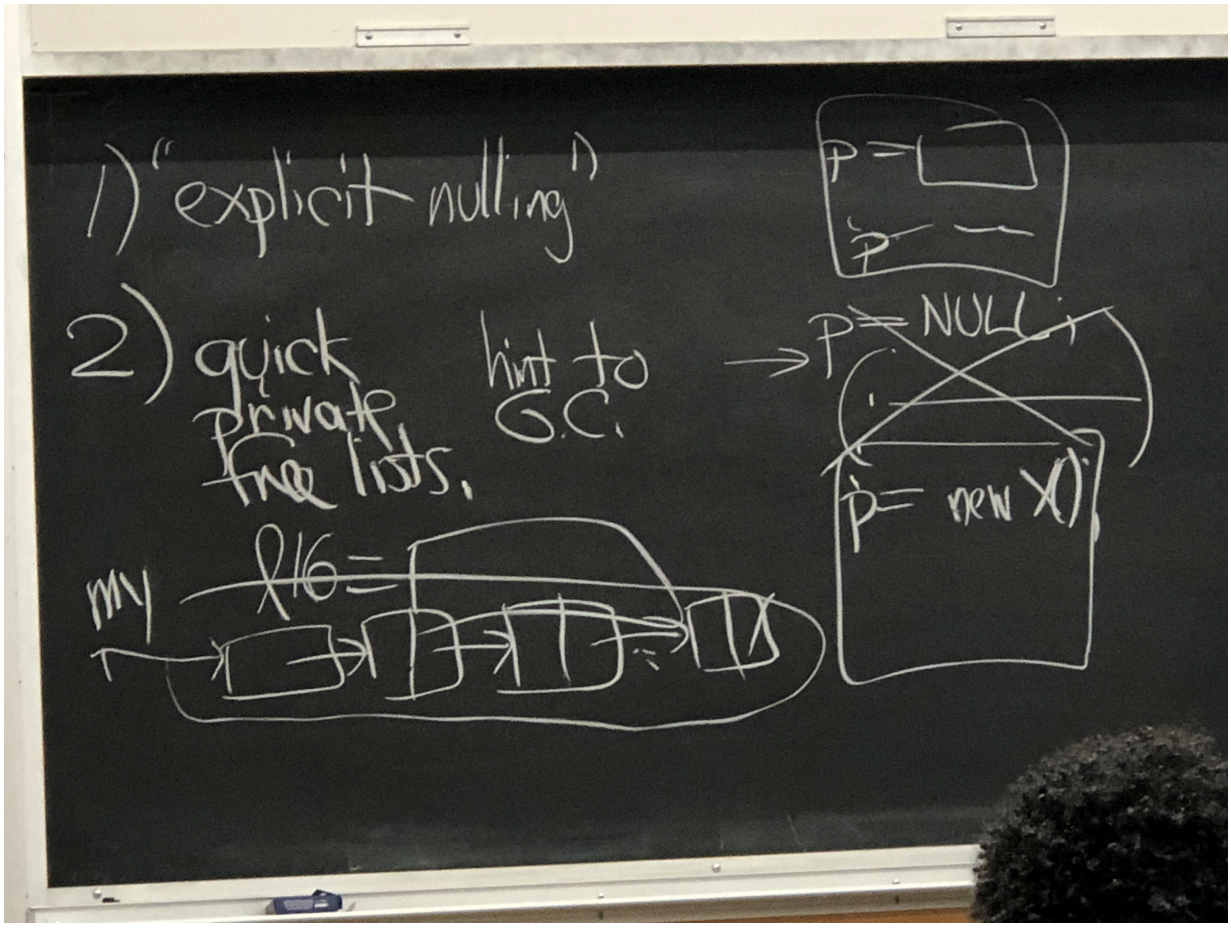


Ways to break cycles:

- add the extra command to remove q's reference to p

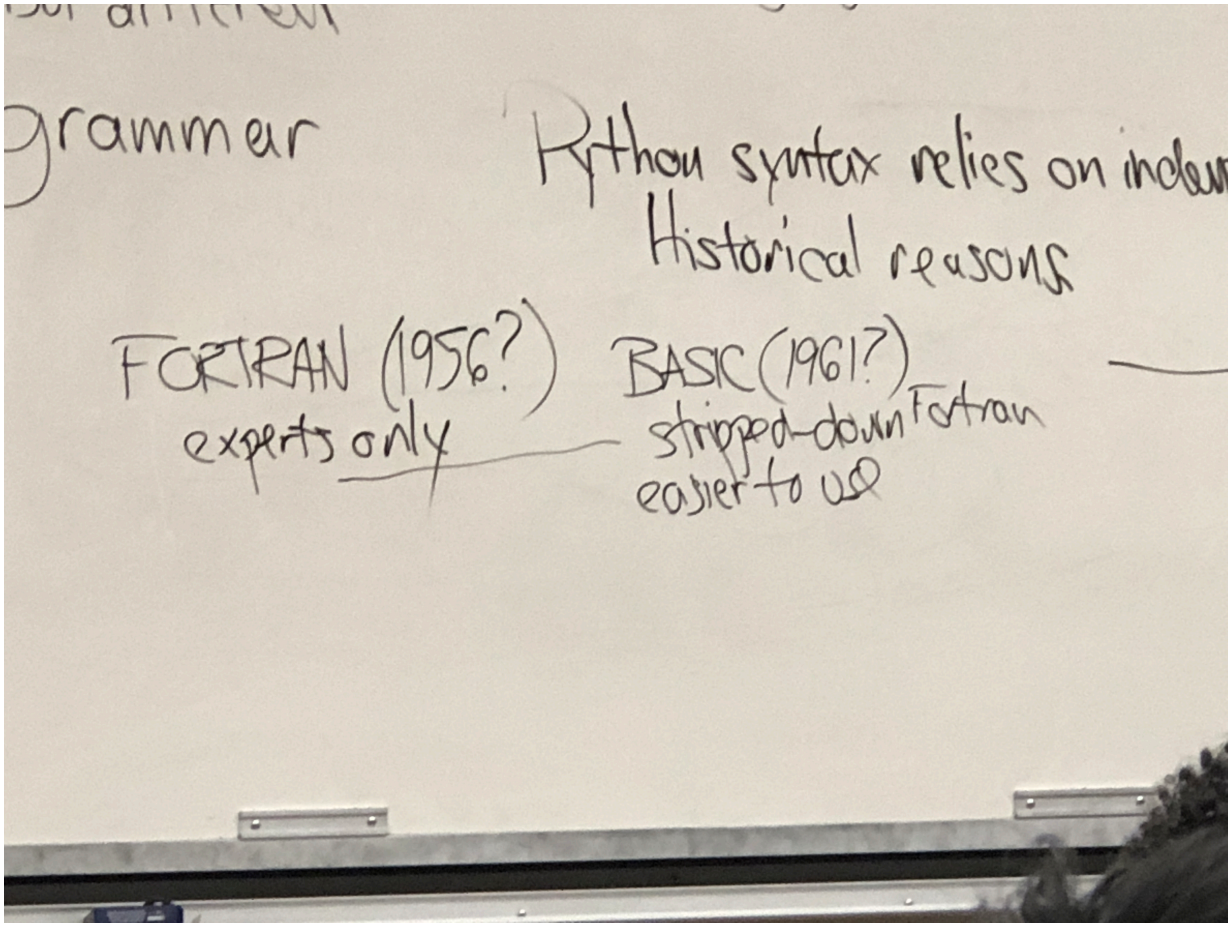


1. **Explicit nulling**
   after done using, set p = NULL (gives hint to GC to free memory)
2. **Quick private free lists**
   works well with traditional mark-and-sweep
   SLOW IN JAVA since it doesn't work well with generation-based garbage collection (has to pull all generations into RAM, cache, etc)

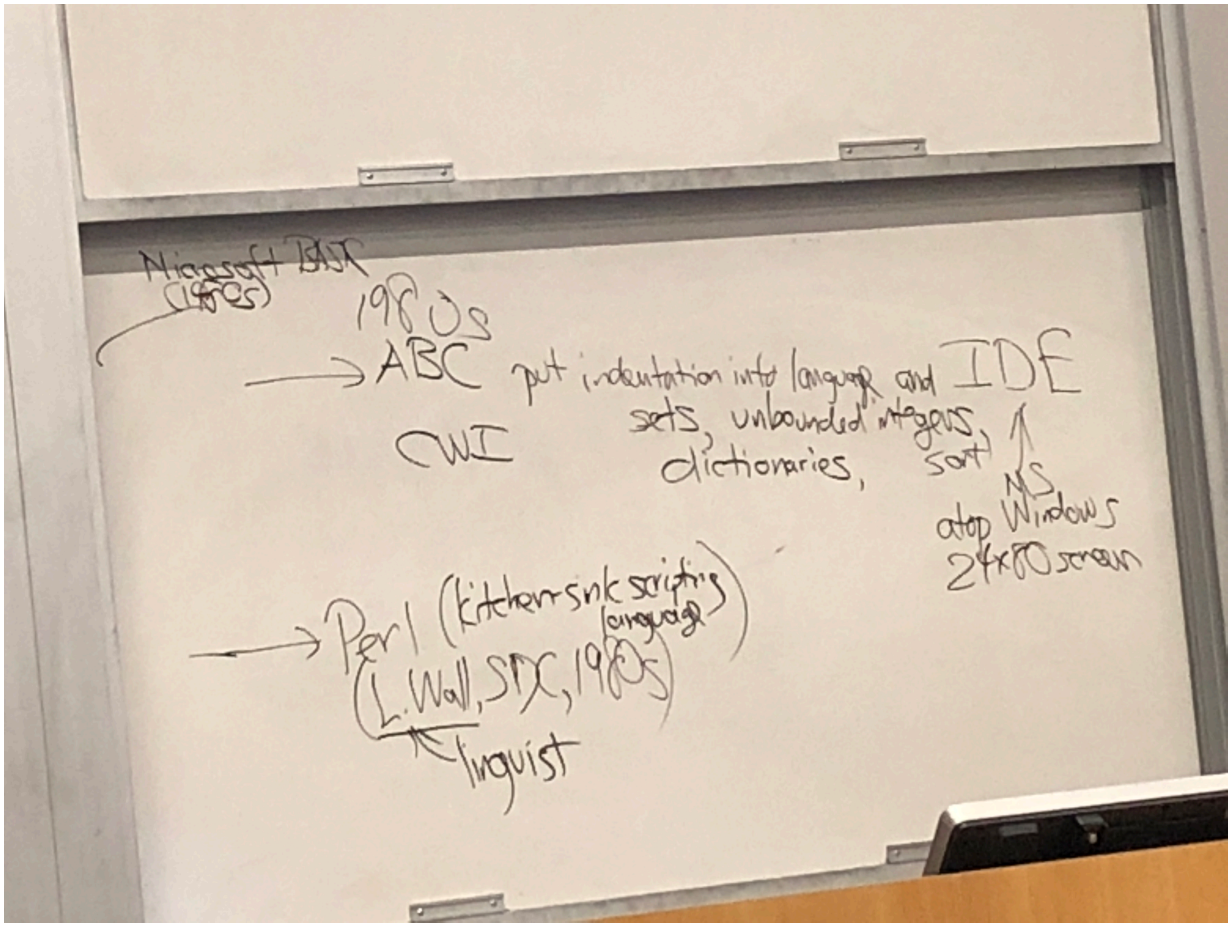


Python
Python similarities and differences from other programming languages

Grammar
- Python syntax heavy on indentation
- History of Python



BASIC was made in Dartmouth to make it easier to teach programming to people



L Wall made Perl in his spare time, and he was a linguist, so he wanted to make a language that would be very accepting of multiple ways to do things

- the motto of Perl is “there is more than one way to do it”
- i.e. `if (X) Y()` is the same thing as `Y() if (X)`

Python Objects

Every value is an object (like Scheme/OCaml) and has

- identity <- cannot be changed
- type <- cannot be changed
- value <- can be changed if mutable
- attributes o.a
- methods o.m(args)

Builtin ops on object

- a is b (eq? a b)
- a == b ~= (eqv? a b) *Scheme
- type(a) ~= a.getClass() *Java <- returns a class that represents a type
  ^ return a’s type
- id(a) ~= a.hashCode()
- isinstance(a, c) ~= a instanceOf C

Namespaces

- A class is an object
- It has a member __dict__ containing its names, as a dictionar
  - c.__dict__['m'] = <some object>

Modules (typically source files)
import heapsort

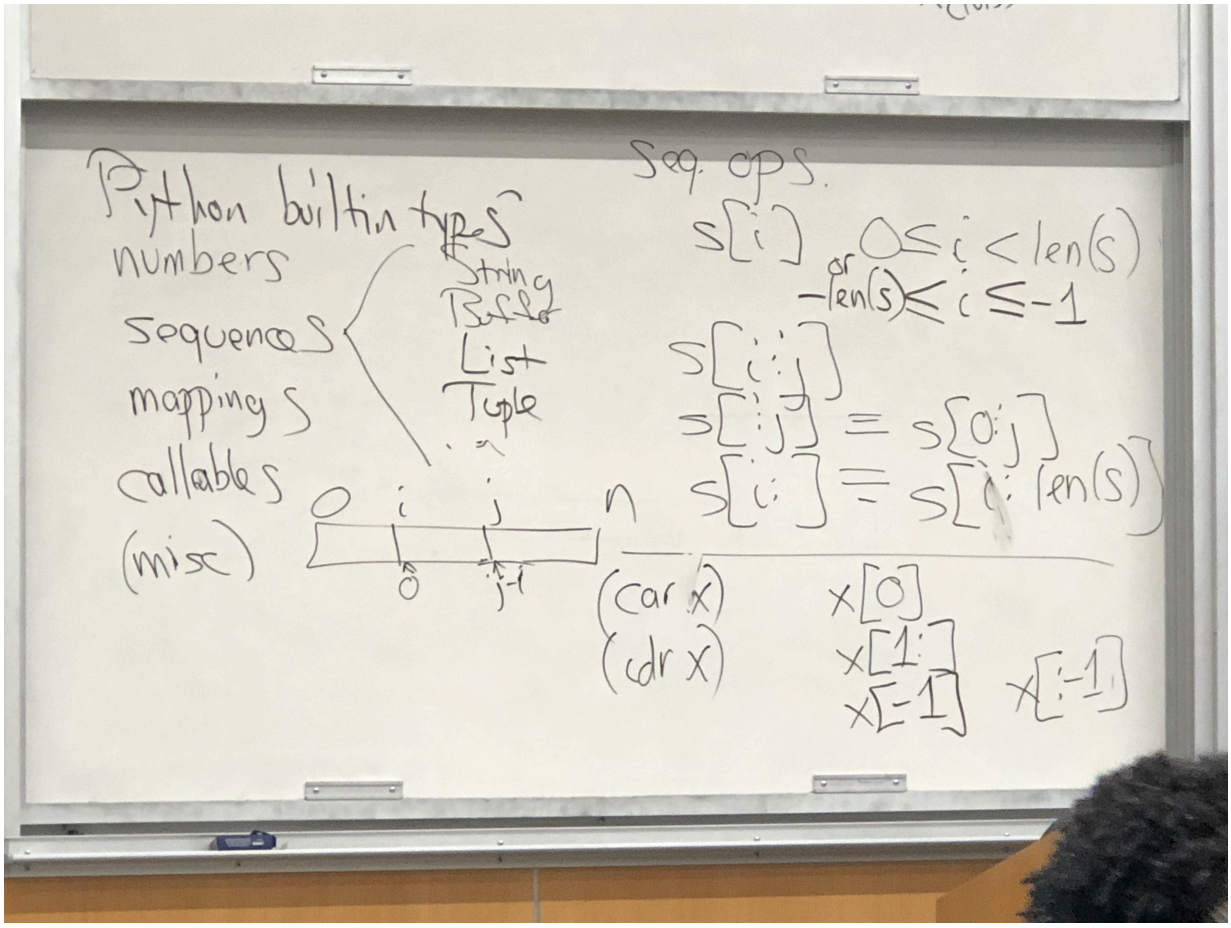
1. creates a new namespace
2. reads & executes source file in this namespace
3. creates a new name "heapsort" in the caller's namespace, heapsort points at namespace created in 1)
   1. caller; heapsort.f

Python's Success
One of reasons of Python's success and success of any programming language

- has good set of built-in primitives and libraries that everybody likes and uses because it's a well-designed set

Python's Built-in types

- numbers
- sequences
  - String
  - Buffer
  - List
  - Tuple
  - ...
- mappings
- callables
- (misc)


Python builtin types
numbers
sequences
String
Buffer
List
Tuple
mappings
callables
(misc)
0
i
j
n
i
j-1
Seq. ops.
s[i]
0 ≤ i < len(s)
or
-len(s) ≤ i ≤ -1
s[i:j]
s[:j] = s[0:j]
s[i:] = s[i:len(s)]
(car x)
(cdr x)
x[0]
x[1:]
x[-1]
x[:-1]