Final exam

- In class, Tuesday 12/18 9am–noon
  In this room
  If you have another final at that time, then Monday 12/19 2–5pm
  Exam target: 75 minutes
  You can bring 2, double-sided pages of notes

- Cumulative, but will focus a bit more on recent topics
  OOP • dynamic programming • summations • graphs
  nope: ~~Turing machines~~ • ~~proofs about languages~~ • ~~sorting~~

Poster session!

1	**Circuits**
2	**Finite-state machines**
3	**How a Hmmm program runs** *registers, RAM, stack, etc.*
4	**Linear data structures** *lists, arrays, stacks, queues*
5	**The building blocks of functional programming** *map, filter, foldl, foldr, list comprehensions*
6	**Recursion and use-it-or-lose-it**
7	**Analysis techniques** *summations, recurrence relations, asymptotic complexity*
8	**Python namespaces**
9	**Dynamic programming** *with tabulation*
10	**Object-oriented programming** *Python vs Java*
11	**Trees and graphs** *Properties and algorithms*

Brainstorm

Idea generators: call out things big and small related to this topic
Scribes: write down the ideas
You can switch roles!

Goal: **quantity, not quality**
we'll deal with quality next…

- Nothing is too big or too small
- **No judgements:** Nothing is better than anything else

Organize

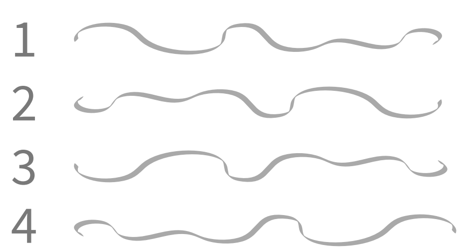
- How might you start to arrange / organize / explain this topic?
  Can you make sub-categories?

- What’s most important?
- What’s “good to know”, but not that important?
- What’s unimportant or unrelated?

- What do you feel you can explain clearly?
- What is not yet clear?

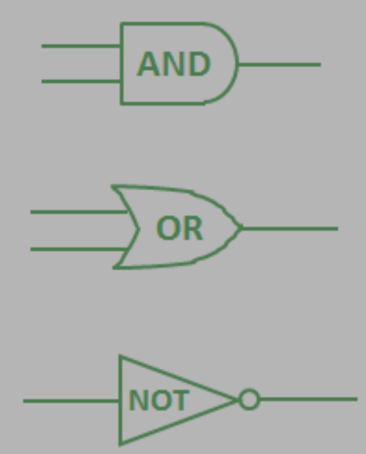
Posters!

13 myths about functional programming and one important truth



...

All My Circuits
an exciting new Netflix show



How to care for and feed a Python

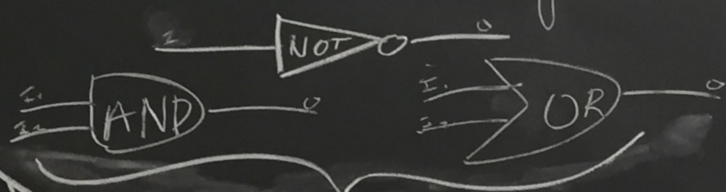
rains
0's?

FA

on

ns

① Latch on to the Logic ◡



Good Circuits

- Correct
- Simple, easy to read
- Efficient

- FUN!

EVERYONE THINKS YOU NEED ALL 3 TO MAKE A COMPUTER, BUT WE ONLY USED 2!

COMPUTER SCIENTISTS HATE HIM!

IN	OUT
1	0
0	1

NOT!

IN1	IN2	OUT
0	0	0
0	1	1
1	0	1
1	1	1

OR!

IN1	IN2	OUT
0	0	0
0	1	0
1	0	0
1	1	1

AND

1 = ON / HIGH VOLTAGE

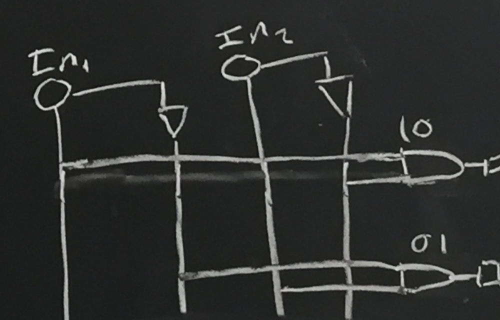
0 = OFF / LOW VOLTAGE

Truth Table
↓

in1	in2	out
0	0	0
1	0	1
0	1	1
1	1	0

XOR

Minterm Expansion

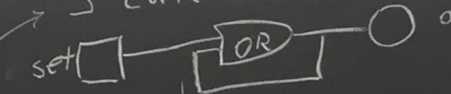


Each output "1" gets an and gate connected to each input directly or through a not gate, and then goes to the output.

PUT IT TOGETHER TO MAKE PARTS OF COMPUTERS!

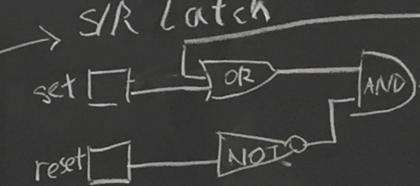
S Latch

Once set = 1, remembers this input



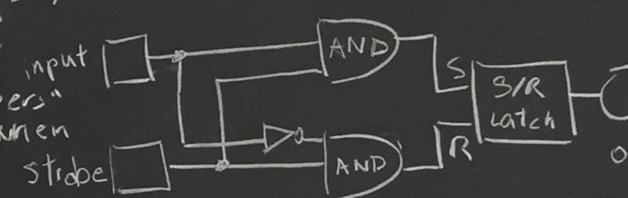
S/R Latch

S Latch with reset button

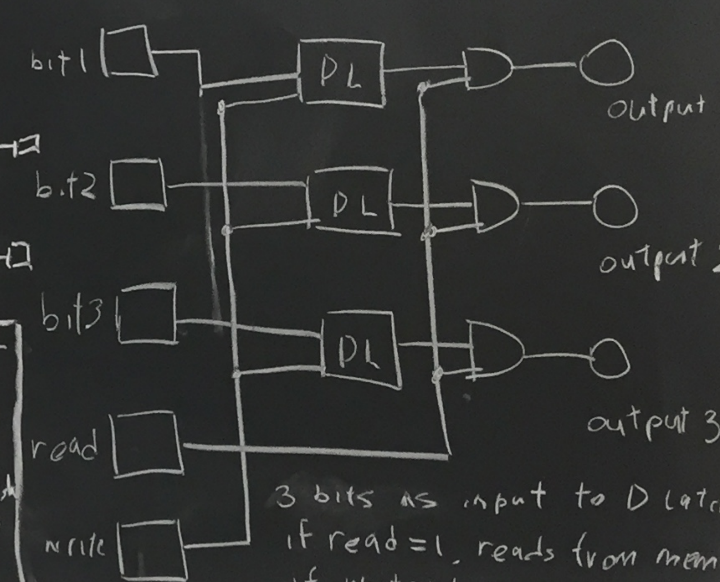


D Latch

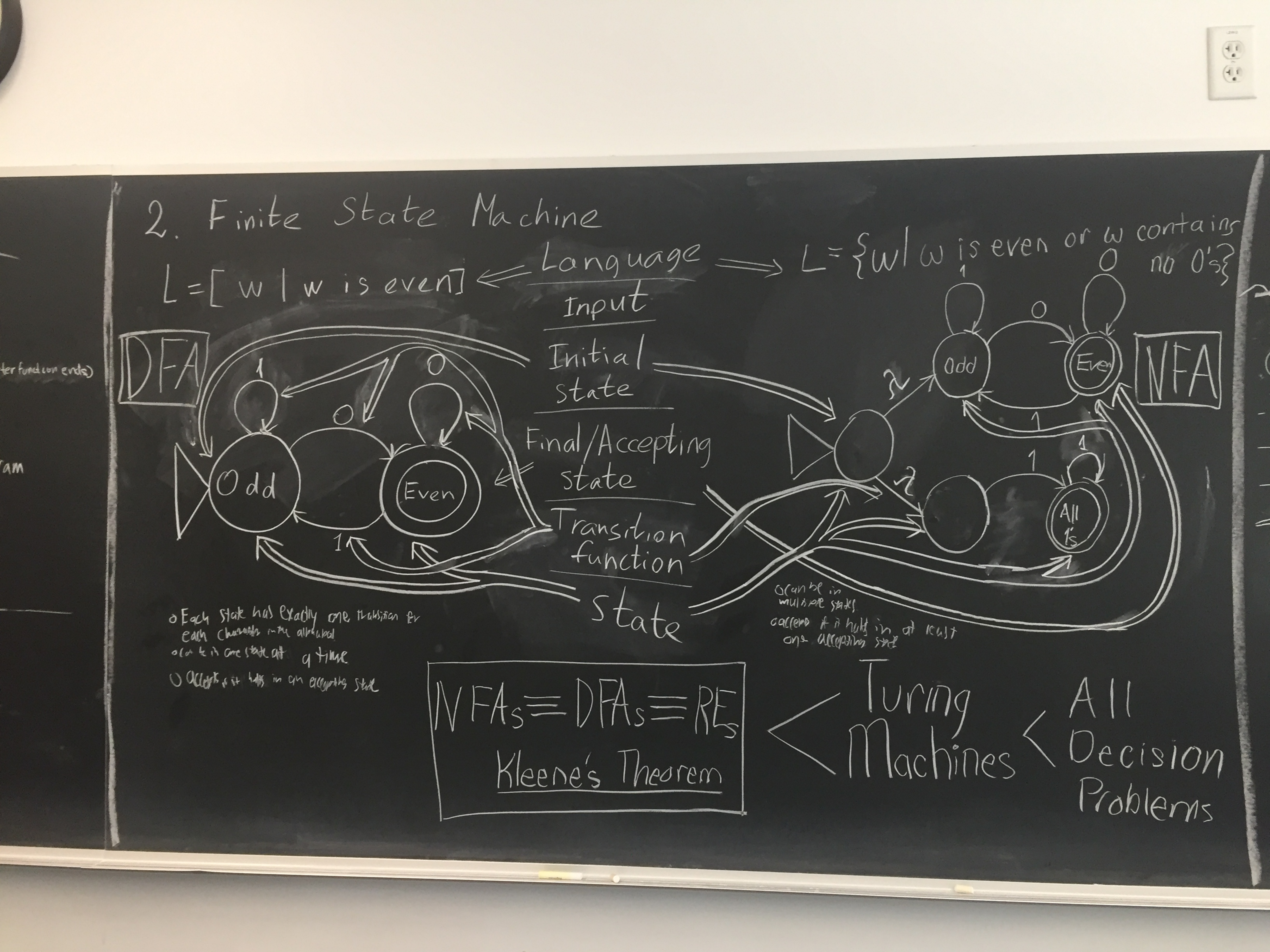
Pass through when strobe is on. "remembers" last value when off



3-Bit Memory



3 bits as input to D Latch
if read = 1, reads from mem
if write = 1, writes input to mem


2. Finite State Machine
L=[w | w is even]
Language
L={w | w is even or w contains no 0's}
Input
Initial state
Final/Accepting state
Transition function
State
DFA
NFA
Odd
Even
All 1's
Each state has exactly one transition for
a time
can be in multiple states
at least
NFAs ≡ DFAs ≡ REs
Kleene's Theorem
Turing Machines
All Decision Problems

3. OMG - 4 tips for HMMM that'll Clear your Skin

CPU

Fetch-Execute Cycle:

1. Retrieve the instruction at the program counter
2. Execute the instruction
3. Increment the program counter

This is how HMMM executes a program

Instruction types

- Computations (add, subtract, etc registers)
- GOTOs (jump & various jumpifs)
- Memory management (move values between registers & RAM)

```
00   Set n r15 Q # set Stack pointer
01   read r1 # read a val
02   Calln r14N # go to Func
03   write r13
0N   push r1
0N+1 push r14 # save
0N+2.. DO Stuff
0N?  JumpON Someth
0N?? pop r14
0X   pop r1
0Y   copy r13 r1
0Z   jumpn r14
```

Memory

r_{13} : return value of function

r_{14} : return line (where the program counter goes after function ends)

r_{15} : stack pointer (points to top of stack)

STACK: LIFO; stores variables throughout program outside of registers.

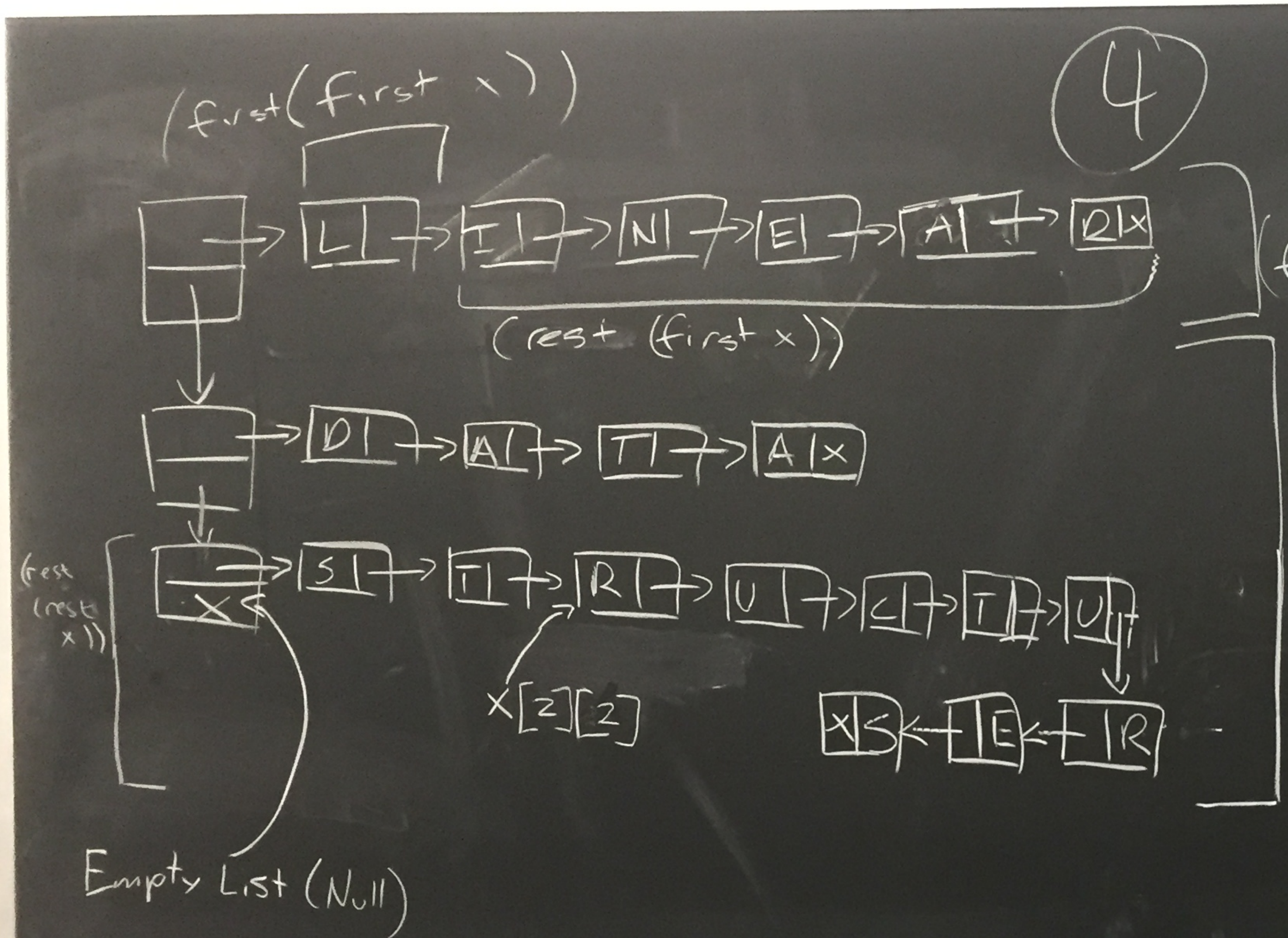
Push: Add a value to top of stack

Pop: Remove most recent value from stack

Stack builds to higher RAM addresses

Other important Skills

- Tracing Code (given code, say what it does)



(first x) = x[0]

(rest x) = x[1:]

Python

(mutable)

lists: [_, _, ..., _]

tuples: (_, _, ..., _) (immutable)

[x**2 for x in range(10) if x>5]

map / element / list / filter

Java

ArrayList<Object>

int[]

Racket list (linked) pros: insertion, deletion, first element

cons: search/access

haha

great for stacks

→ Stacks

pop - remove top element

push - add element to top

Arrays: pros: search/access

cons: resizing

Better for queues (you have to grab the last element)

Fun-ctional Programming!

Docstring

;;; PROS!

;;Referential Transparency

;;; - Same input -> Same output

;;; - (replace variables w/ vals & vice versa)

;;; - No side effects (changes in state ie printing to screen)

→ Easier debugging

;;; - Order of execution is irrelevant

;;; - not as much Memory

;;; CONS :(

;;; no side-effects

;;; more abstract

;;; harder to understand (at first)

Functional — Racket — Other Langs — Imperative

Interface	Implementation
Data Structures — Function	(define f(x) (+ x 1))
Data Structures — List	[1 |] → [2 | /] (cons 1 '(2))
Data Structures — tree	3 → 1, 5
Anonymous functions	(define adder (n) (lambda (x) (+ n x)))
HOFs — map	(map (adder 1) L)
HOFs — filter	(filter (lambda (n) (= 0 (% n 2))))
HOFs — fold	(foldl + 0 L) (1+2)+3
"Variables" (constant values)	(let ([CS 42]) body expr)

1. Base Case
2. "I": one piece
3. What is the solution with it?
4. What is the solution without it?
5. How do we combine them?

..., redundant

...ations

S

...mance of a program
...have finite resources
depends on situation
time, memory
calls, steps

RECURRENCE RELATIONS

...N) = cost given input size N

```
...efine (sum n)
 (if (= n 0)
     0
     (+ n
        (sum
          (- n 1)))))
```

cost metric = recursive calls

$T(0) = 0$
$T(1) = 1 + T(0)$
$T(2) = 1 + T(1)$
...
$T(N) = N$
which is $O(n)$.

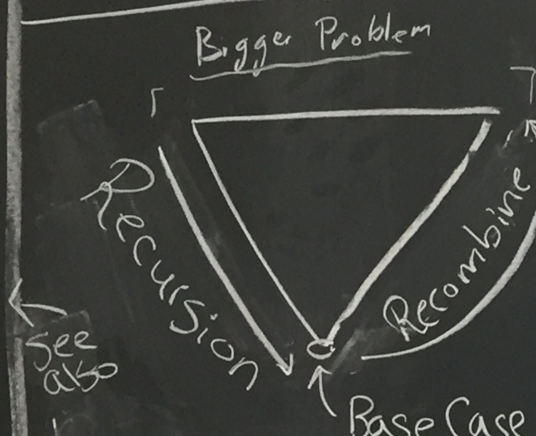
*Used to find
...to find time complexity
...of recursive functions

...tractiby
(N^m) $O(2^N)$
$(n!)$

↳ $T(N)$ can be recursive

Recursion

- Solving a problem by breaking it into smaller problems (ex: sum of a list)



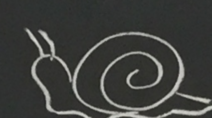
REPRESSED MEMORIES

(1) Align
(2) Make-change
(3) Car pne
(4) Tree traversals
(5) Word difference (EditDistance.py)

Use it or Lose it

- A specific type of recursion
- Used by making a template:
  1. Base Case
  2. "It" = one piece
  3. What is the solution with it?
  4. What is the solution without it?
  5. How do we combine them?

Benefits	Drawbacks
- Intuitive	- Slow
- Gets Right answer	- Inefficient, redundant Computations
- Useful for inductive structures	



Docstring

Fun...

;;; PROS!

;; Referential Transparency
;;; - Same input -> Same output
;; - (replace variables w vals (free) used)
;;; - No side effects (changes in state ie printing to screen)
;;; ↳ Easier debugging
;;; - Order of execution is irrelevant
;;; - Not as much Memory

;;;;;;;;;;;;;;;;

;;; CONS :(

;;; no side-effects
;;; more abstract
;;; harder to understand (at first)

;;;;;;;;;;;;;;

Analysis Techniques

What: Measuring the performance of a program
why: Real world programs have finite resources

CONCEPTS cost metric: some metric of cost, depends on situation
empirical: measured Ex. time, memory
theoretical: ↓ Ex. calls, steps

SUMMATION

T(N) = cost given input size N

for j in range(N)
. . . . . } cost K

$$\sum_{j=0}^{N-1} k = N \cdot K$$ ← if k is constant

nested loop: $\sum \sum k \in O(N^2)$ (outer loop, inner loop)

*use for loops

RECURRENCE RELATIONS

T(N) = cost given input size N

```
(define (sum n)
  (if (= n 0)
      0
      (+ n
         (sum
           (- n 1)))))
```

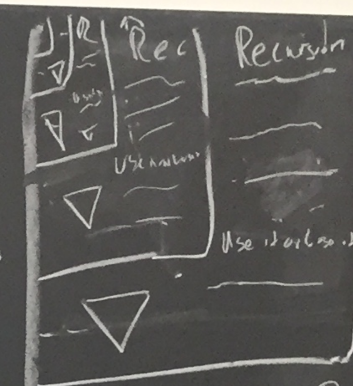
cost metric = recursive calls

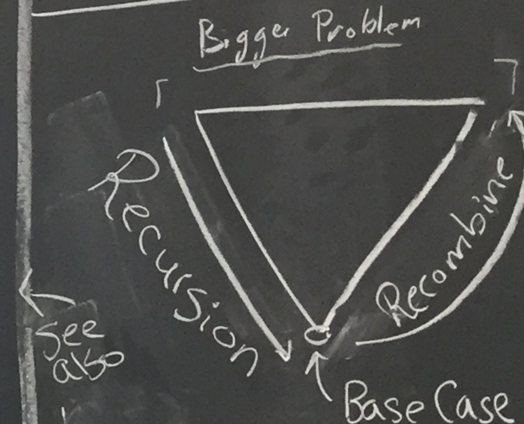
T(0) = 0
T(1) = 1 + T(0)
T(2) = 1 + T(1)
...
T(N) = N
which is O(n).

*Used to find to find time complexity of recursive functions

↳ T(N) can be recursive

no problem	tractable	Intractable
$O(\log_2 N)$, $O(1)$, $O(\sqrt{n})$	$O(N)$, $O(n \log N)$	$O(N^N)$, $O(2^N)$, $O(n!)$





REPRESSED MEMORIES

(1) Align
(2) Make-change
(3) Car pne
(4) Tree traversals
(5) Word difference (EditDistance.py)

-Sol...
br...
pr...

Us...
- Asp...
- Used...

Benefi...
- Intuitive
- Gets Right...
- Useful fo... structu...

s to

[n-2]

(8)

Namespaces are one honking great idea

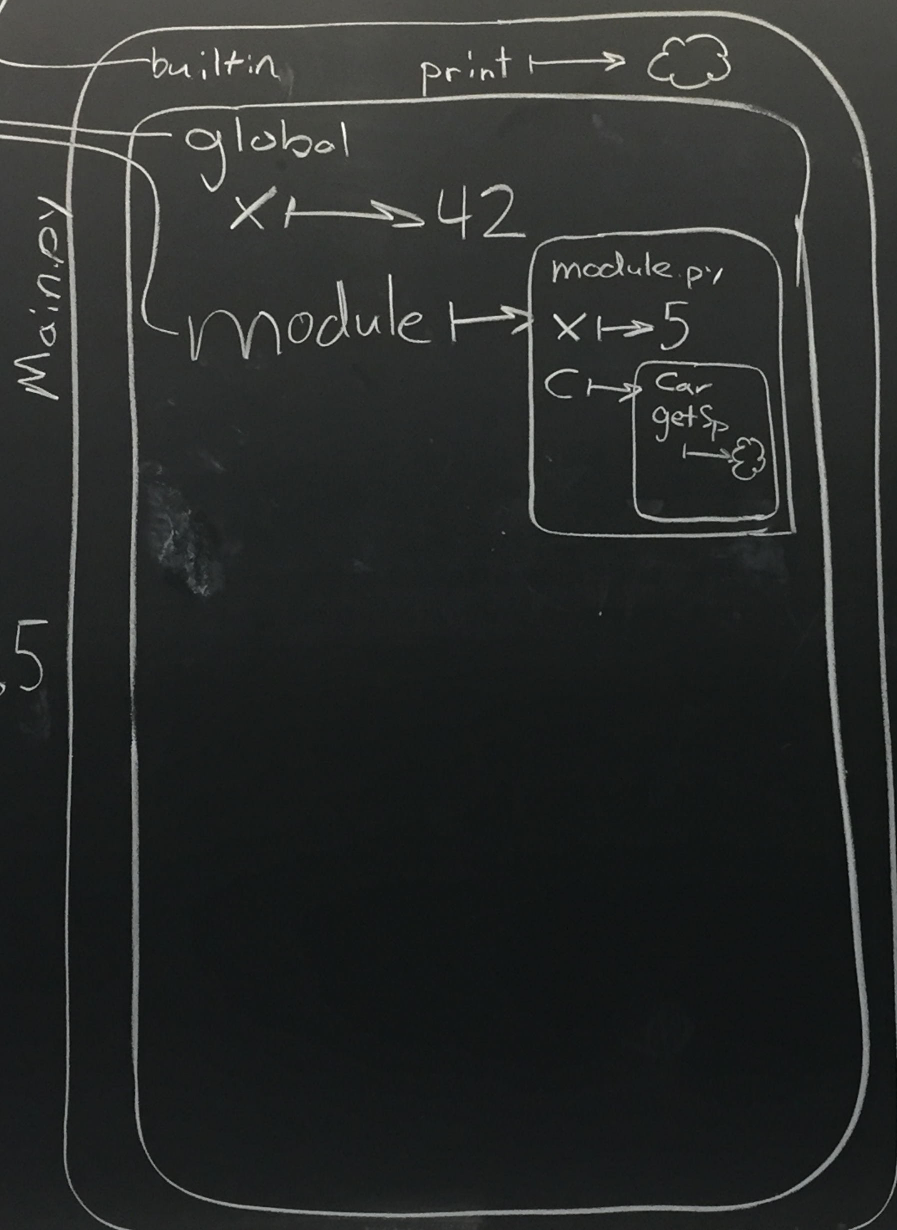
the builtin namespace contains builtin functions

global is for the file

when we import a module, a new namespace gets created

We refer to names in an imported namespace as

module.x #evaluates to 5



How are names looked up?

1) Local

↳ 2) Global

↳ 3) Builtin

↳ ERROR

\- main.py -

```
import module
x = 42
class Car:
    def __init(sp):
        self.carSp = sp
    def getSp()
        return self.carSp

class Honda(Car)
    def __init__(sp):
        super().__init(sp)
        self.type = "Honda
    def getType
        return self.type
```

\- module.py -

```
x = 5
c = Car(10)
h = Honda
```

Dynamic Programming

with tabulation

Why: Recursive functions include a lot of repeat calculations.

What: By keeping track of old calculations you can avoid doing repeat work, making a program run faster.

9

Memoization:

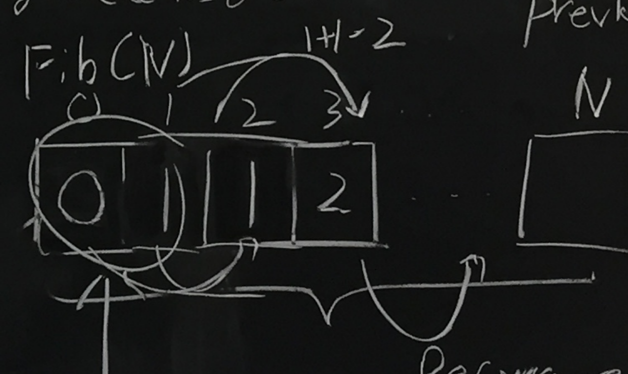
* When solving a problem, check to see if it has already been computed to avoid redundancy.

```
f(a)   if a in memo
           return memo[a]
       else:
           recursion
```

tabulation

* use previous values in a generated table starting at the N=0 case.

Example Fib(N)



Base Case

Recurrence Relation
table[n] = table[n-1] + table[n-2]

① Consider the base case
② Calculate how many cells to use
③ Use recurssion to fill cells using previous cells
④ Return result in the result cell

(10) OOP in a table

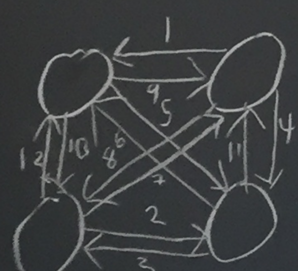
	Interface	Implementation	
	• What a thing does • Determined beforehand	• How it is done (What's in the blackbox)	
		Java	Python
Public vs Private (global vs local)	The user uses the public variables and methods (functions).	• Private variables • Public variables	• "_" notation (private) • ALL CAPS (global) *Social construct!
Objects -data structures -classes	Each kind of object has a defined set of operations. The user interacts with instances of objects via their interface.	• new Object (initialization) • This ★ Knows about itself • Constructor • toString, equals, hash	• Simply declare the variable !! • self • __init__ • __str__
Inheritance	Subclasses can have more variables and methods available to the user than their superclasses Each instance of a subclass can work as an instance of its superclass. "Is-a"	• Extends, Implements ★ Implement the super class	• Class Name (superClass)

11

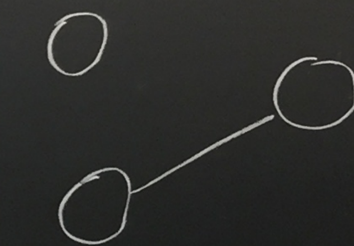
Graphs

Def: a data structure of nodes and edges
- edges connect nodes and can have direction and weight

Uses: modeling things (ie internet)

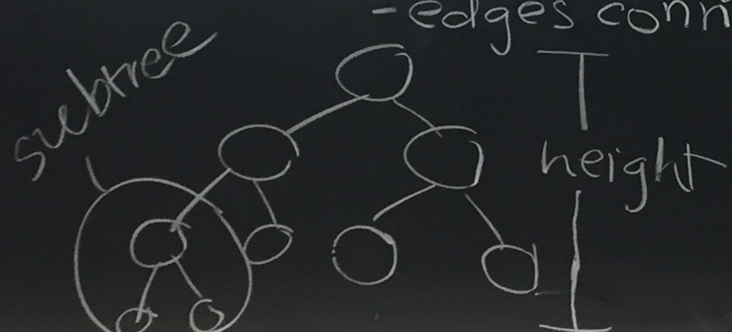


Types: connected v. nonconnected
complete v. noncomplete
directional v. nondirectional
weighted v. nonweighted
dense v. sparse
cyclic v. acyclic



Trees

↳ specific type of connected, directional, nonweighted, acyclic graph
- edges connect roots and leaves



Binary Search Tree
↳ specific type of tree that makes it easier to store and find things