Sudoku as CSP.

See Slides for sample board.

Initial state: partially filled board.
Action: Fill in a blank square.
Goal: Full board s.t. each digit from 1 to 9

appears exactly once in every row, column, and
3x3 box:

just refer with a number [0, 9]

Variables: A1, A2, ..., A9 0, 1, ..., 9
B1, B2, ..., B9 = 9, 10, ..., 17 81 variables
I1, I2, ..., I9 72, ..., 80

Domains: A1 = 0, 1, ..., 9
B1 = 9, 9, ..., 93

Constraints: AllDiff (0, 1, ..., 9) large set of
c - all other rows -
AllDiff (0, 9, ..., 72)
c - all other columns -
AllDiff (0, 1, 2, 
9, 10, 11
18, 19, 20)
c - all other 3x3 boxes -

Clever ways to code this?
You will implement backtracking search with AC3 (forward checking not required).

**initial board**

```
\[ \begin{array}{c|c|c}
    A1 & A2 & 1 \\
\end{array} \]
```

Run AC3() to see if all arcs are consistent.
- If yes, try B1 = 1
- If no, backtrack

**Data Structures**

1. `vals[9][9]` - 9x9 array with initial board values (0 if empty)
   - To Do: Store final cell values.
   - Example: `vals[0][0] (A1) = 0`
   - `vals[1][0] (B1) = 9`

2. `globalDomains[9]` - array of size 81 of ArrayLists of Integers
   - To Do: Store domains of each cell
   - Example: `D_0 = [1, 2, ..., 9]`
   - `D_2 = [3, 3, ..., 3] (cell A3)`
3. neighbors - array of size 81 of ArrayLists of Integers

To Do: Stores the neighbors of each variable (i.e., other vars that share a constraint)

Neighbors of 0-2 variables in the same row, column, or unit

4. Arc - object with variables $X_i, X_j$
   - represents arc/constraint $\langle X_i, X_j \rangle$

5. Global Queue - To Do: stores all arcs.

ex $(0,1), (0,2), \ldots (0,8)$ \text{ all } $(0,1), (0,2), (0,9)$

$(1,0), (1,1), \ldots (1,8)$ \text{ rows}

$(8,0), (8,1), \ldots (8,8)$ \text{ cols}$ (0,20)$

$(0,9), (0,19), \ldots (0,72)$ \text{ all boxes.}

$(0,9), (1,10), \ldots (1,72)$ \text{ all cols}$
Methods:

1. **Init**:
   - Set up `globalDomains`
   - call `allDiff()` to set up neighbors, `globalQueue`
   - call `backtrack(0, globalDomains)` - already in code
   - set final board values in `vals`

2. **allDiff**
   (called by `init`)
   - fills neighbors, `globalQueue` (using binary constraints)

3. **backtrack** (performs backtracking search with `AC3`)
   - (int `cell`, `GlobalDomains`)
   - tries a value for cell (using `globalDomains`)
   - calls `AC3()` to check for arc consistencies
   - if not: backtrack to try another value
   - if yes: call `backtrack` on next cell

4. **AC3**
   - calls `Revise()` to update domains
   - return true if consistent; o/w false

Most of the code in `backtrack()`...
backtrack(cellnum, gD)

(5) When do we know a solution is found?
if (cellnum < 80), return true.

First, check if cellnum already assigned an initial value.
if (vals[cellnum] for cellnum != 0)
   // call backtrack on next cell
   backtrack(cellnum + 1, gD).

Suppose: (2) // call AC3 on current assignment (one set by previous call to backtrack, ie on cellnum-1)
if (!AC3(gD))
   return false.

(4) for some value v in cellnum's domain:
   try assigning cellnum = v
   init
   if solution found ((backtrack) returns true), return true
   (else? try another value v)

How to know if this assignment works? Try it!
Call backtrack on next cell.