Section with Barbara

Week 3

AI Stories

Crossword

Optimization

ANY QUESTIONS BEFORE WE BEGIN?

LOCAL SEARCH

Finds the optimal value based on local constraints

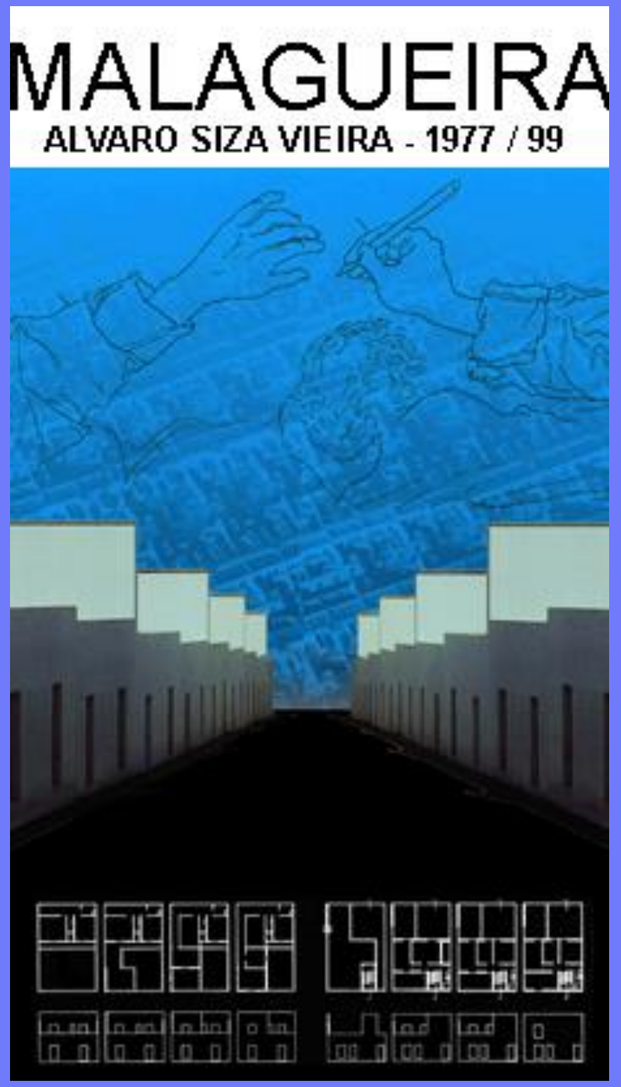
LINEAR PROGRAMMING

Finds the optimal value based on linear constraints

CONSTRAINT SATISFACTION PROBLEM

Finds the optimal value based on variable constraints

AI Stories: CAD



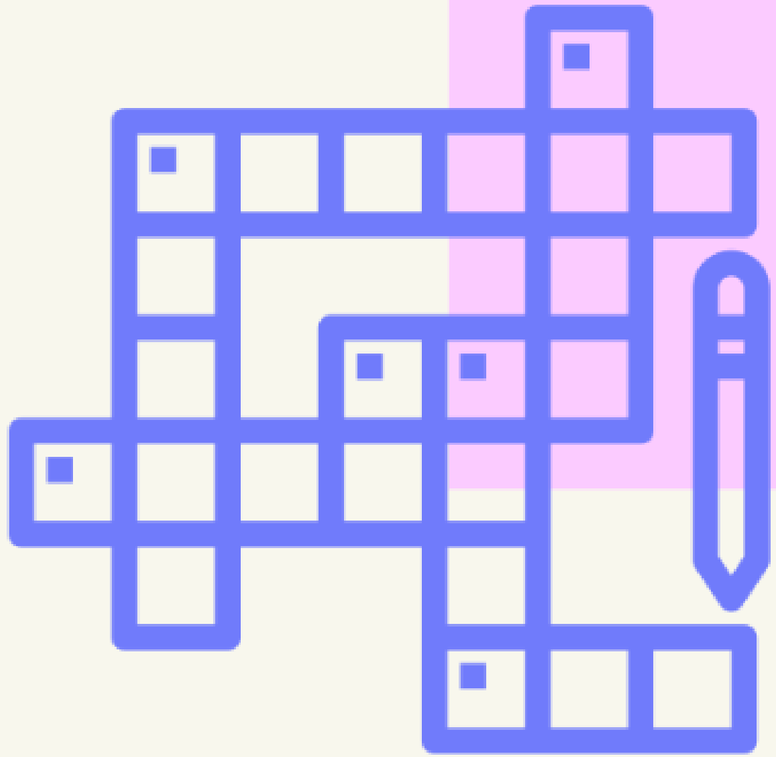
local search

derivatives.ipynb

groceries.ipynb

linear programming

crossword (aka csp)

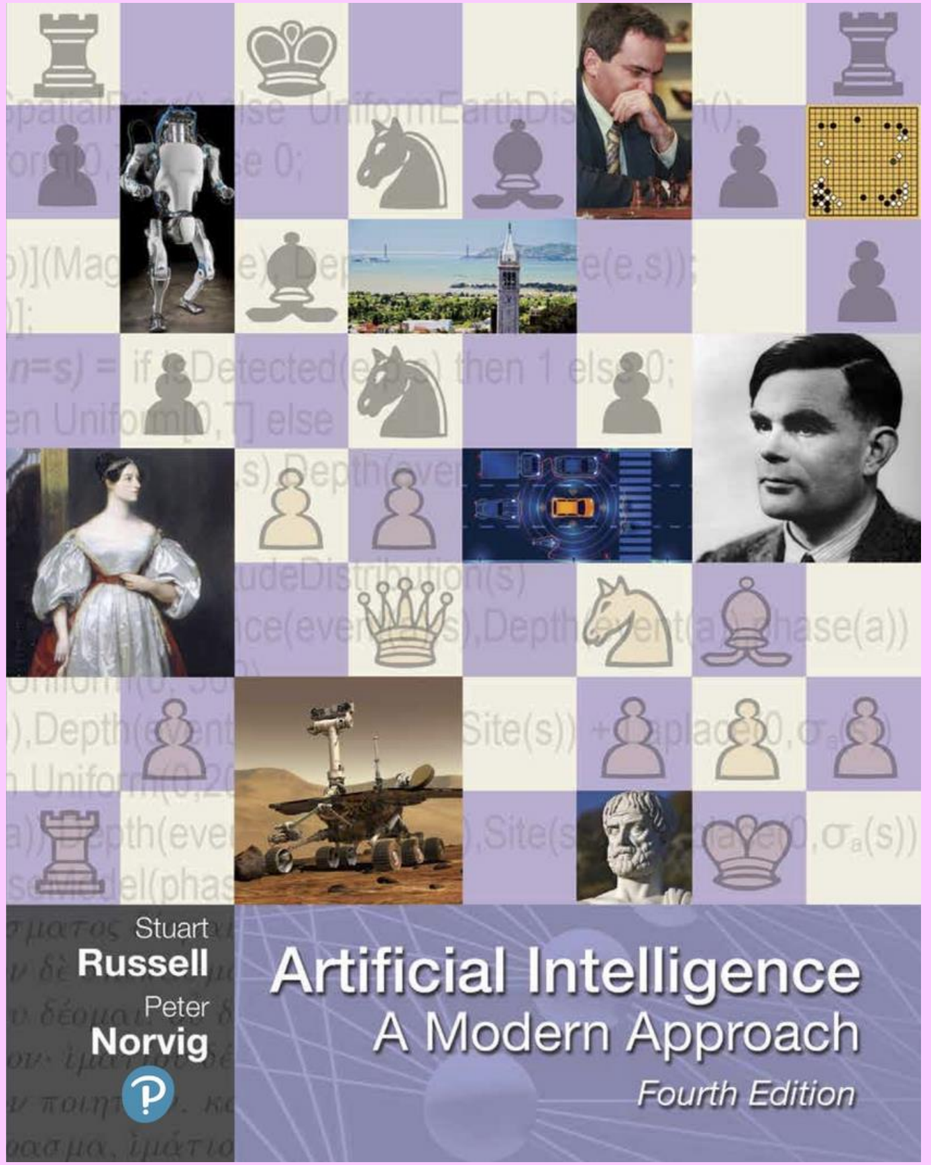


CHAPTER 6

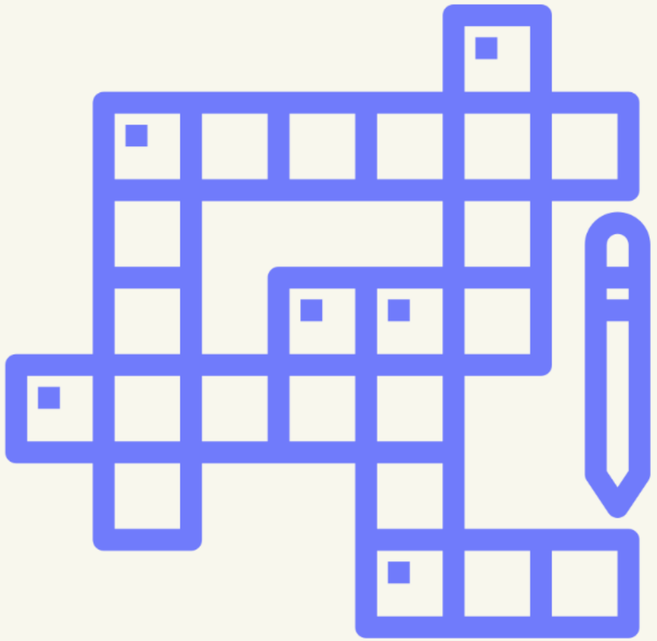
RUSSELL & NORVIG

ARTIFICIAL INTELLIGENCE: A MODERN APPROACH

Is all about CSPs!

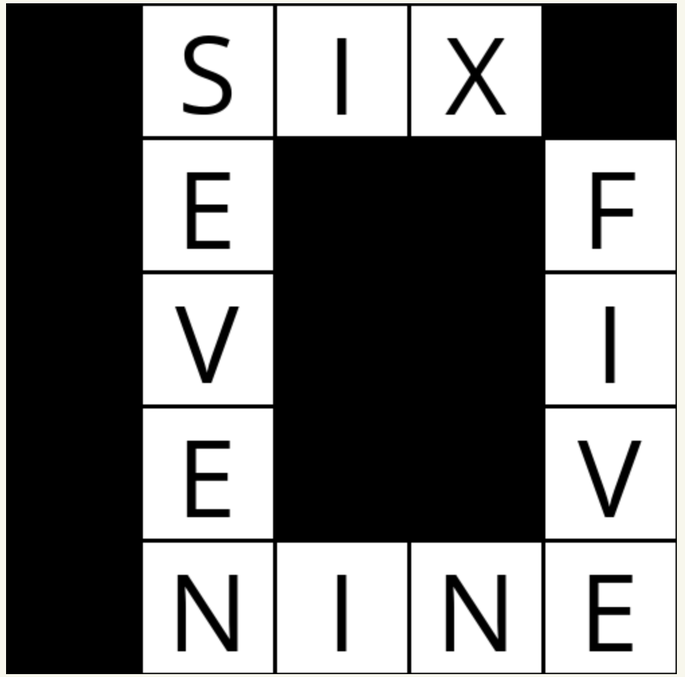


crossword

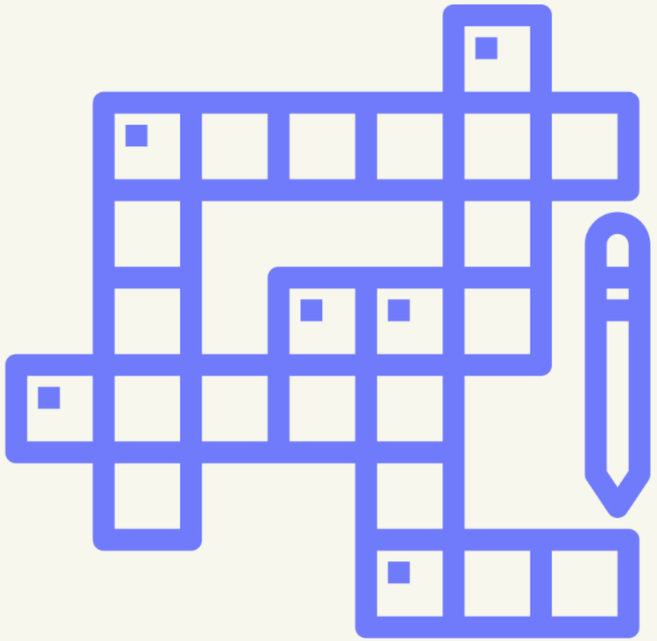


Class VARIABLE

Variable(i=4, j=1, direction='across', length=4)



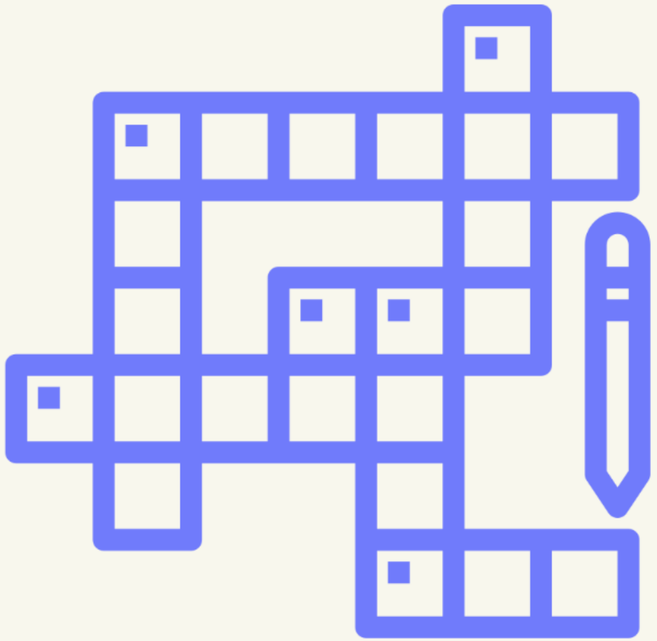
crossword



CrosswordCreator.DOMAINS

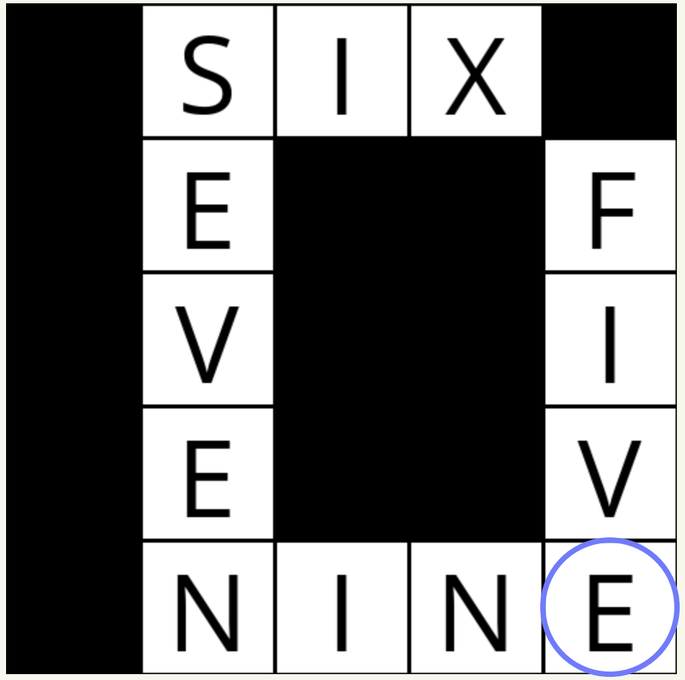
This records all the potential words a Variable can be assigned, given what constraints have been imposed.

crossword

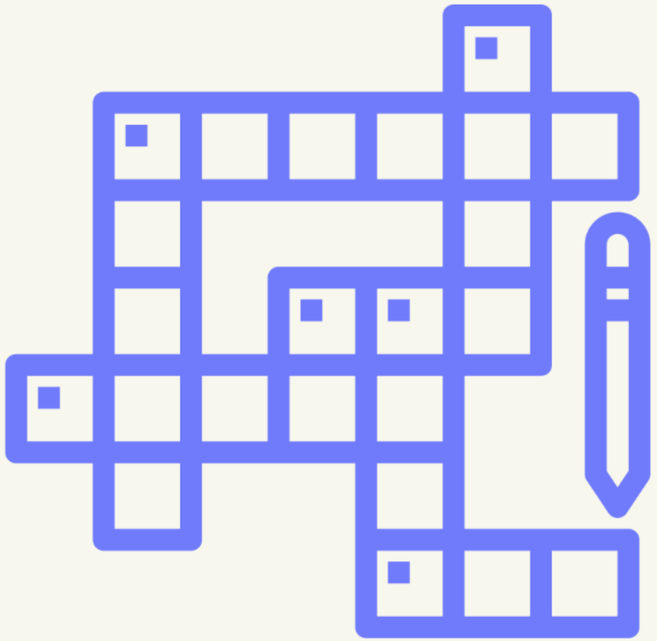


Crossword.OVERLAPS

This is a tuple where two variables overlap.



crossword

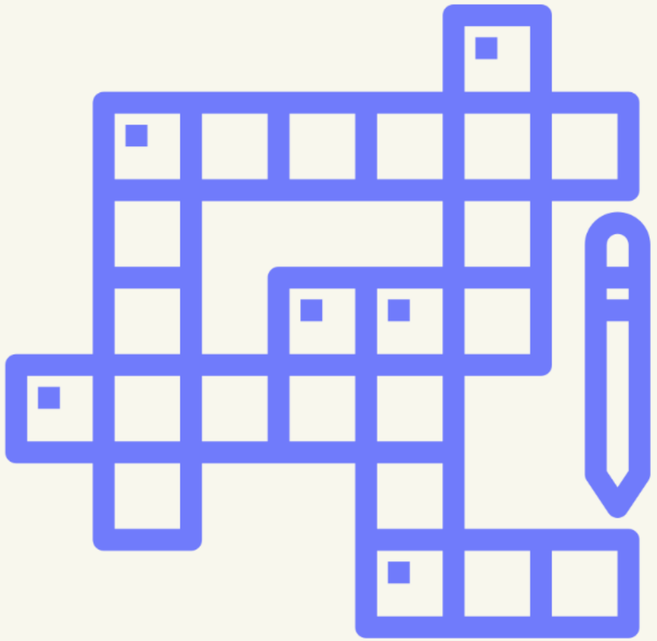


The ASSIGNMENT variable

This is a dictionary of the hash values of each Variable, and their assigned words from self.domain[Variable].

Be mindful of the data type that self.domain[Variable] should be in `assignment`.

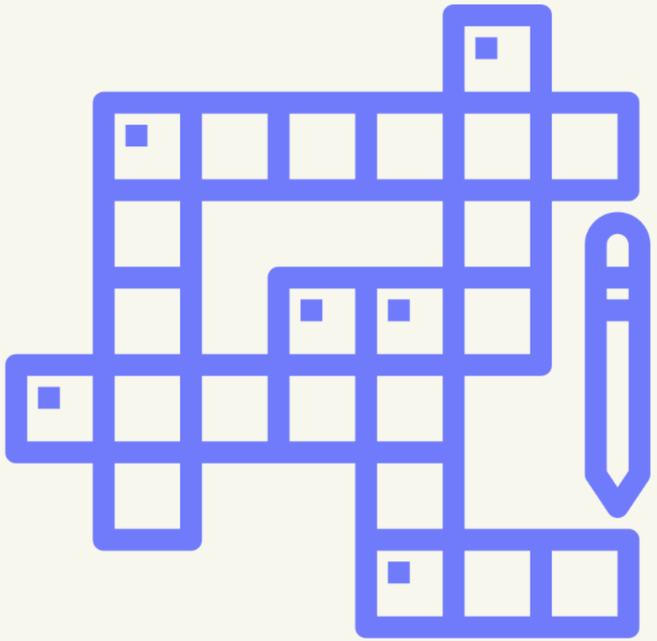
crossword



Crossword.NEIGHBORS

This is a set of overlapping variables to the given Variable.

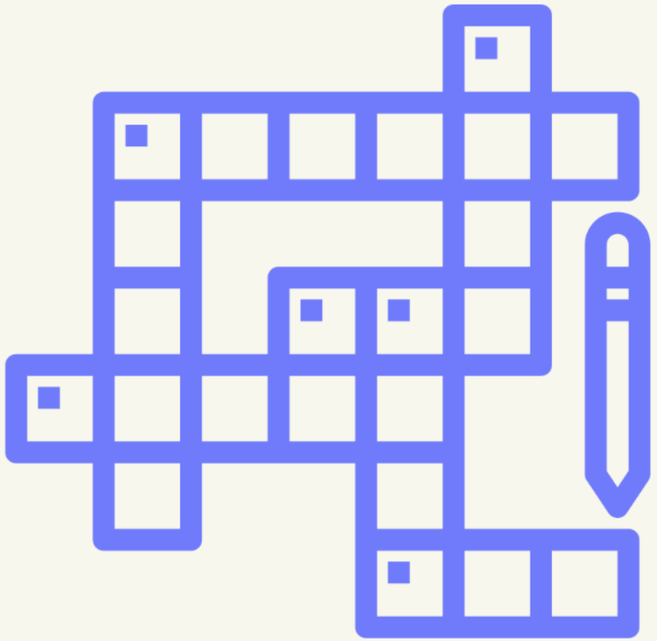
crossword



```
def enforce_node_consistency(self):
        """
        Update `self.domains` such that each variable is
        node-consistent. (Remove any values that are
        inconsistent with a variable's unary
        constraints; in this case, the length of the
        word.)
        """
```

Understand the data structure of self.domains - what does updating look like?

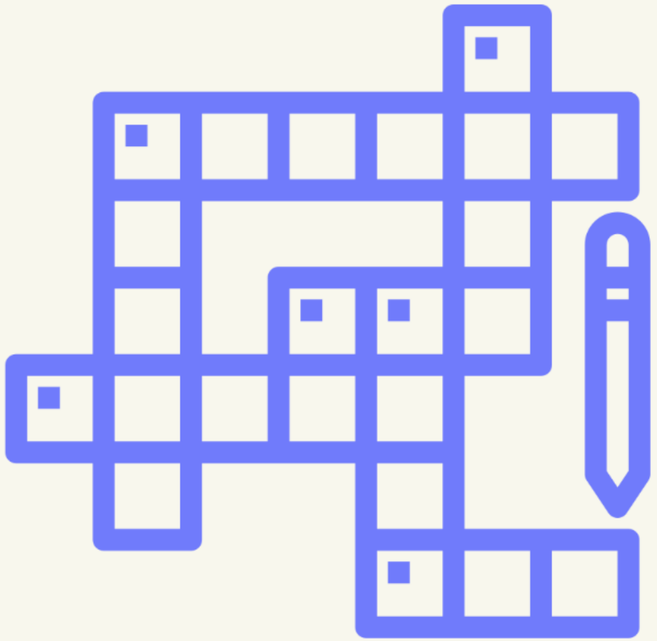
crossword



```
def revise(self, x, y):
    """
    Make variable `x` arc consistent with  variable
    `y`. To do so, remove values from
    `self.domains[x]` for which there is no possible
    corresponding value for `y` in
    `self.domains[y]`. Return True if a revision was
    made to the domain of `x`; return False if no
    revision was made.
    """
```

Definitely reference the lecture pseudocode!

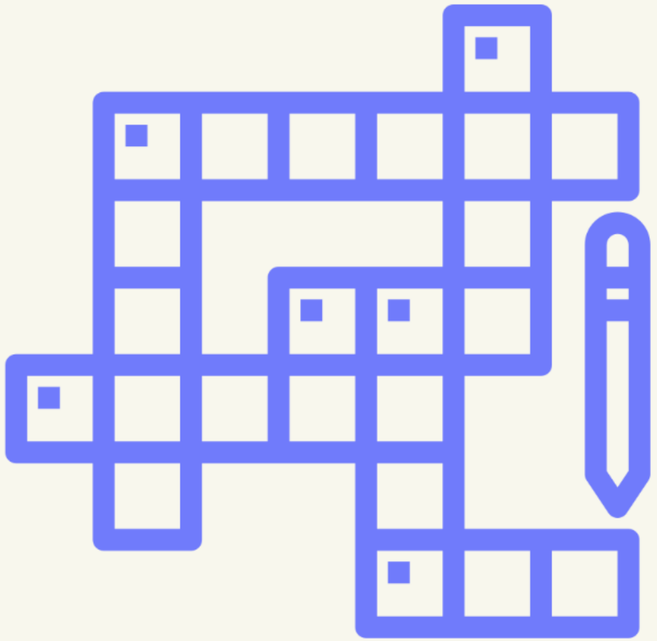
crossword



```
def ac3(self, arcs=None):
    """
    Update `self.domains` such that each variable
    is arc consistent.If `arcs` is None, begin with
    initial list of all arcs in the problem.
    Otherwise, use `arcs` as the initial list of arcs
    to make consistent.Return True if arc consistency
    is enforced and no domains are empty; return False
    if one or more domains end up empty.
    """
```

Definitely reference the lecture pseudocode!

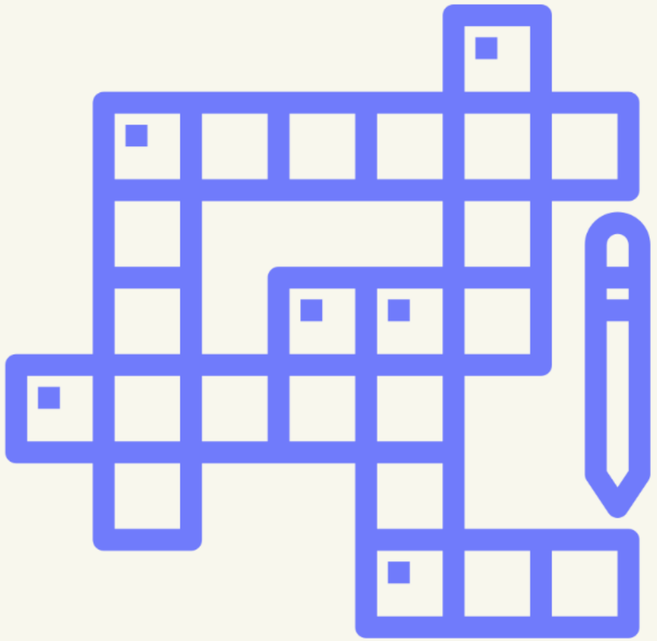
crossword



```
def assignment_complete(self, assignment):
    """
    Return True if `assignment` is complete
    (i.e., assigns a value to each crossword
    variable); return False otherwise.
    """
```

This is the shortest one you will be coding - it should only be around one line of code!

crossword

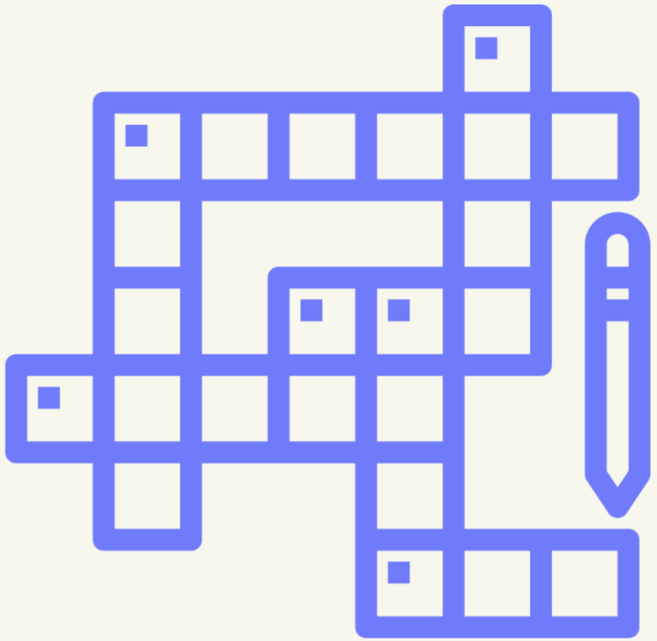


```
def consistent(self, assignment):
    """
    Return True if `assignment` is consistent (i.e.,
    words fit in crossword puzzle without
    conflicting characters); return False otherwise.
    """
```

What are the conditions for assignment to be consistent? (Hint: there are 3!)

Good design tip: return False right away once you know `assignment` is not consistent!

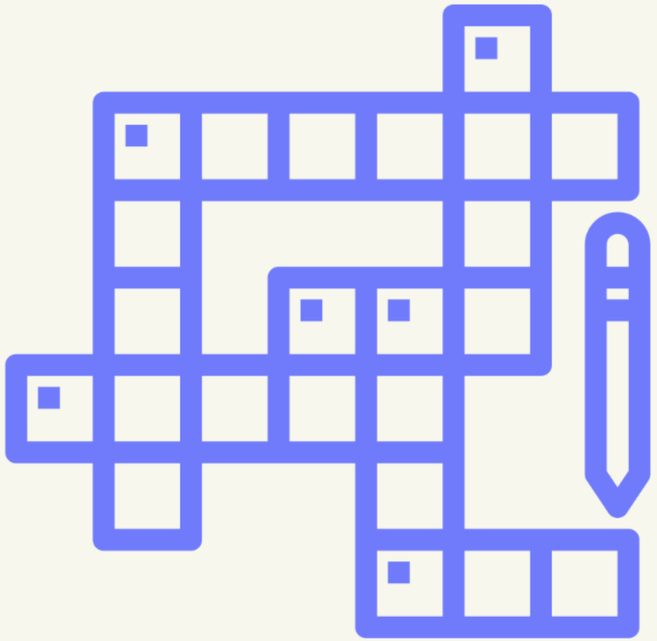
crossword



```
def select_unassigned_variable(self, assignment):
    """
    Return an unassigned variable not already part
    of `assignment`. Choose the variable with the
    minimum number of remaining values in its
    domain. If there is a tie, choose the variable
    with the highest degree. If there is a tie, any
    of the tied variables are acceptable return
    values.
    """
```

Returns the Variable to be assigned next - a sort function will be helpful here! You will be implementing two heuristics: minimum remaining value heuristic and the degrees heuristic.

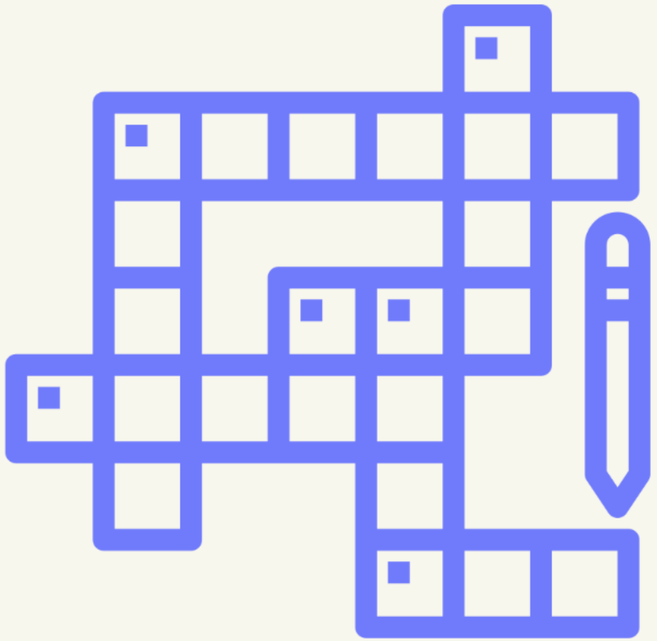
crossword



```
def order_domain_values(self, var, assignment):
    """
    Return a list of values in the domain of `var`,
    in order by the number of values they rule out
    for neighboring variables. The first value in
    the list, for example, should be the one that
    rules out the fewest values among the neighbors
    of `var`.
    """
```

Once an unassigned Variable is selected, determine the order to examine the Variable's domain. What functions/methods already in the code can help you implement this?

crossword



```
def backtrack(self, assignment):
    """
    Using Backtracking Search, take as input a
    partial assignment for the crossword and return
    a complete assignment if possible to do so.
    `assignment` is a mapping from variables (keys)
    to words (values).If no assignment is possible,
    return None.
    """
```

Do not reinvent the wheel! Definitely reference the lecture pseudocode and schedule0.py