1. In class, we discussed the problem of placing *k* queens in an *n x n* chessboard. In this problem, you will be considering the problem of placing *k* knights in a *n x n* chessboard such that no two knights can *attack* (it **moves** to a square that is two squares away horizontally and one square vertically, or two squares vertically and one square horizontally) each other. *k* is given and *k* <= *n*2. Formulate this problem as a Constraint Satisfaction Problem. [**20 points**]

- What are the variables? [5 points]

The variable is represented by "knights" that will be represented in the format of a list. Containing a value to measure their row number (distance from the top), and the index representing which column they reside.

- What is the set of possible values for each variable? [5 points]

This was kind of answered in the question above. The possible value for each individual variable is mainly just the position information. It can however hold information directly from functions that detect conflicts for the specific queen if the programmer wanted to. I wouldn't do this personally, I would just text total conflicts of each board instance.

- How is the set of variables constrained? [10 points]

The constraints:

Each knight can not be placed on top of another knight
Each knight in N must remain in the bound of the NxN board. Meaning (#of knights == col) (#of knights == row)

Each knight must not be able to attack another knight.
This means that for each (x,y) value of one knight to another knights (x, y) cannot have values equal to,

(x2-x1) = 1 && (y2-y1) = 2 | (y1-y2) = 1 && (x2-x1) = 2

X values being the column number, which in my algorithm is essentially the index of the list.
Y values are the actual value in the list for that queen. Each new index of a list is considered its own knight.
I have come to the conclusion that the way I proposed is not necessarily the best for knights compared to queens. Knights can have multiple instances on one col or row meaning the

idea of using a list as indice = x && value = y. This is naive in this problem, therefore creating a list of tuple values for x,y would be the better choice.

2. Explain why it is a good heuristic to choose the variable that is most constrained but the value that is least constraining in a CSP search. [**10 points**]

Choosing the most constrained variable is a good choice most of the time, this is partly due to the fact that a constricting variable means variables affected by it are restricted from the search part of the algorithm (I.E pruning, splitting the domain, reduction in backtracking). Choosing the least constraining value is impactful because it allows for more options for the variables affected by its choice. This means there is a higher likelihood of finding the solution in a faster and easier manner.

How to run.

1. Download zip and extract pythonProject from zip

2. Have a python IDE to run the main.py file
3. Enter generation/pop size and it will produce the best found n-queens solution in CLI format.