A description of the search algorithm
Greedy best first search algorithm is used to solve the problem. The following describes the formulation used to solve the problem

- **State** : Valid parse trees of the expression by applying valid rules(actions)
- **Actions** :
  - Evaluate - Solve simple arithmetic sub terms. For example, (+ 2 3) can be replaced by 5
  - Commutative - Rearrange all numbers together so that they can be evaluated. Example: on applying this action on (+ (+ x 1) 2), we would get (+(+ 2 1) x)
  - Inverse - Take terms on other side of the expression to solve for variable. Example: (= (+ x 5) 6) will become (= x(- 6 5)) on applying the rule
- **Initial State** : Parse tree of given expression
- **Goal test** : Check if variable to be solved for is on one side of the actions can be applied again on the current node

A complete list of the actions you have developed, preferably grouped by type

- Evaluate: It simplifies sub-terms of type (binary-operator, n1, n2) where n1 and n2 are integers or float.
- Commutative: It rearranges terms applying commutative rules for ‘+’ and ‘*’ to simplify and apply other actions on the terms.
- Inverse: It tries to keep the variable to be solved on one side and take all other terms on other side.

Limitations: Inverse has been applied only for addition. Similar approach can be used to apply for other binary operators

A specification of the heuristic you employ, and the rationale behind it.
The heuristic function is the depth of the parse tree. The goal is to simplify the expression as much as possible, which will reduce the number of operators, numbers and variables in the expression. Hence, depth could be a good heuristic to reach the goal.

Output from your program:

Evaluation:

1.
   ```
   eq > x=1+2
   var > x
   This is parsed at: (= x (+ 1 2))
   Solution: x = 3
   ```

2.
   ```
   eq > x=(2+10)*(2^2)
   var > x
   ```

This is parsed at: (= x (* (+ 2 10) (^ 2 2)))
Solution: x = 48

3. eq > 4*6/3 =y
   var > y
   This is parsed at: (= (/ (* 4 6) 3) y)
   Solution: 8 = y

Commutavity

1. eq > x= 1+y+4
   var > x
   This is parsed at: (= x (+ (+ 1 y) 4))
   Solution: x = 5 + y

2. eq > x = 1 * y * 6* z
   var > x
   This is parsed at: (= x (* (* (* 1 y) 6) z))
   Solution: x = 6 * y * z

Inverse

1. eq > x+5=6
   var > x
   This is parsed at: (= (+ x 5) 6)
   Solution: x = 1

2. eq > 1+2+x=7
   var > x
   This is parsed at: (= (+ (+ 1 2) x) 7)
   Solution: x = 4

Description of how to run the program
Run the following command:
$python eqsimplifier.py

List of Resources used
1. https://code.google.com/p/aima-python/ - Used the classes and functions from this.
2. https://github.com/clojure-numerics/expresso/blob/master/src/main/clojure/numeric/expresso/simplify.clj -