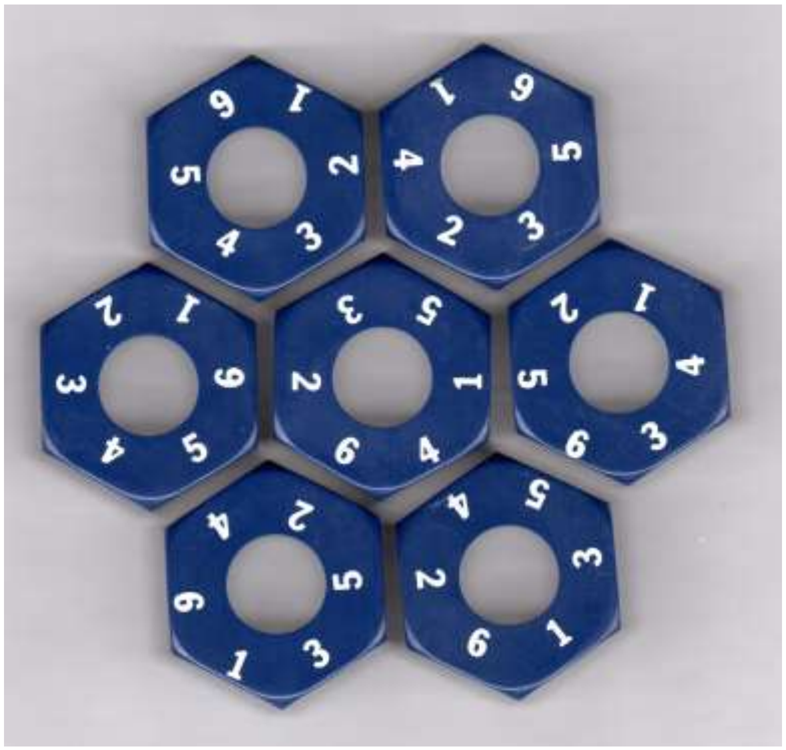
COSI 101a
Problem 1
“Drive Ya’ Nuts!”

Due Thurs Mar 11th 11:55PM via Latte
100 points worth 10% of final grade
Don’t wait till the last minute!
5% late penalty per day.

You’ve seen this: there are n+1 regular polygons with n sides, each a permutation of the n numbers (or colors or shapes) on each side. The problem is to arrange the polygons in a ring with one in the center, such that the numbers on each adjacent edge will match. In the commercial version, there are 7 pieces, each with 6 possible rotations, and hexagons tile the plane. But we can generalize. 4 triangles would fold up to a pyramid, 5 squares give you 5/6ths of a cube, 6 pentagons give you a bowl, and so on, even though 11 decagons or 13 dodecagons will not tile or fold up. As N increases, the difficulty of solving rises like the factorial (i.e. exponential), so you need more intelligence (or lots more CPU time) to create and solve these puzzles. This is known as "Combinatorial Explosion" and affects many approaches to AI.



The assignment is to work with Lisp (or python) on a suite of fun issues surrounding this family of puzzles, learning about how search over representations can solve puzzles thought to take “intelligence”.

1) (10%) GENERATE: Generate candidate puzzles with N+1 N-agons with no repeats. One simple way is to use a permute function which uses "equal" as its test and a remove-duplicates which uses "equal" as its test, generate all nuts of a certain size n which are different when rotated to a canonical position, e.g. 1 in the first position. In other words, although there are 6! Nuts, there are only 5! nuts unique under rotation. Then just pick N+1 of them. For larger puzzles, you need to generate 1 permutation at a time, rotating to a canonical position, and collecting N+1 unique nuts (This might fail for smaller sizes).

2) (20%) SOLVE: You want a program which can “solve” a given puzzle and return the FIRST found solution using some form of search. The simplest way is "brute force", searching thru (N+1)! Permutations, placing the first nut in the center and the other N nuts around it matching the center edge, then finding the first where all the edges are in compliance.
3) (30%) PRUNING: A smarter way is to do a depth first search placing one nut at a time while pruning when there are no possible solutions. For example, you can place the center, and place sides 1,3,5 matching the center edge, then filter the remaining nuts to see if they match 3 edges. Or you can place one nut at a time, checking that there exist nuts to continue the puzzle. In your discussion, report which approach (brute vs prune) finds a solution faster for different N.
4) (20%) COUNT: You want a program which can COUNT how many solutions exist, returning that number. (Hint: Your brute force or pruning solution can be modified to APPEND or COUNT answers.)
5) (15%) GRAPH: Next, you want a program to (1000 or 10000 times) randomly generate legal puzzles, and count how many have zero, one, or more than one solution. Graph these 3 percentages for 5 -12 sized nut puzzles (if you can:), and discuss. If you find a "promiscuous" puzzle with a lot of solutions, include it in your discussion!
6) (5%) Make sure to DEFUN a top-level program called MAIN which shows one solution to the standard hexnut puzzle given below.
7) 10% Extra Credit[1]: write a Print-Puzzle to print out your solutions for 4-9 sized puzzles. (Hint: one way could be using a character array and trigonometry).

Please submit (A) a few paragraphs discussion of your approach and results, (B) a documented source program listing, and (C) an edited log of the programs demonstrating enough detail that the TA’s might not have to run your code to grade it.

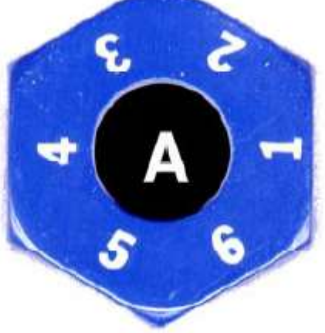
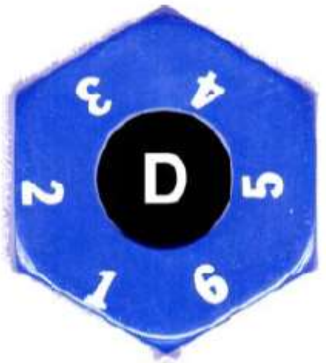
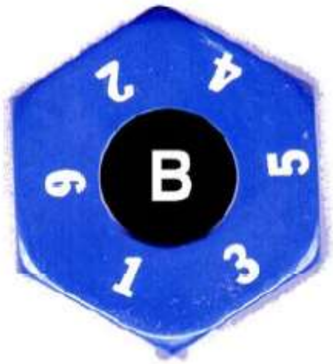
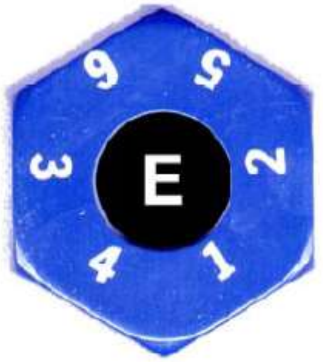
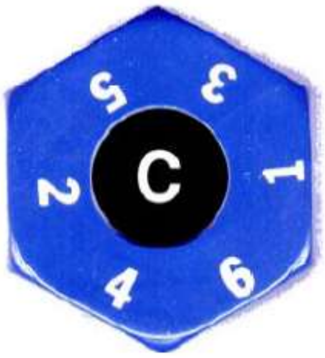
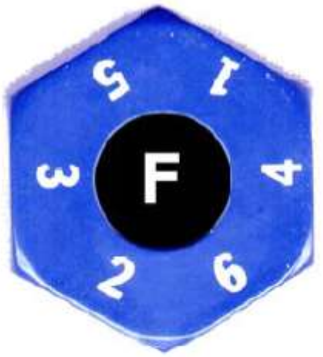
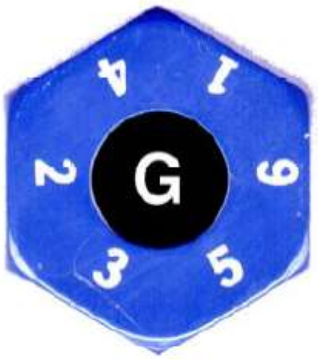
It is fine to discuss the problem and your approach and abstract algorithms/data structures with others but it is not ethical to show or share your program text or use code from the internet without proper attribution. This PERMUTE will not permute lists, just symbols or numbers because REMOVE needs a different :test parameter.

```
(defun permute (list)
  (if list
    (mapcan #'(lambda (x)
          (mapcar #'(lambda
(y) (cons x y))
               (permute
(remove x list))))
         list)
    (list nil)))
```

```
(defvar *nuts* '(
          (1 6 5 4 3 2)
          (1 6 4 2 5 3)
          (1 2 3 4 5 6)
          (1 6 2 4 5 3)
          (1 4 3 6 5 2)
          (1 4 6 2 3 5)
          (1 6 5 3 2 4)))
```

```
If you want to solve it yourself by hand, cut out the nuts on
the next page.
```

[1] Generally, my extra credits are small, fun projects, but never worth the work!


A
3
2
1
6
5
4

D
3
4
5
6
1
2

B
2
4
5
3
1
6

E
6
5
2
1
4
3

C
5
3
1
6
4
2

F
5
1
4
6
2
3

G
4
1
6
5
3
2