Last name _____

First name _____

LARSON—MATH 356–SAGE WORKSHEET 18
Eulerian & Hamiltonian Cycles

1. Log in to your Sage/CoCalc account.

   (a) Start the Chrome browser.

   (b) Go to `http://cocalc.com` and sign in.

   (c) You should see an existing Project for our class. Click on that.

   (d) Click "New", call it **s18**, then click "Sage Worksheet".

   (e) For each problem number, label it in the Sage cell where the work is. So for Problem 1, the first line of the cell should be `#Problem 1`.

   (f) When you are finished with the worksheet, click "make pdf", email me the pdf (at `clarson@vcu.edu`, with a header that says **Math 356 s18 worksheet attached**).

Saving and Re-using Code

We've coded several graphs now, and have added code for functions of graph invariants and auxiliary functions and stored them in "graphs.sage". I pushed my updated version to your Handouts folder. Either copy that file to your Home directory—or add the new stuff to your own "graphs.sage" file. We'll need those functions.

2. I've updated the copy of "graphs.sage" in your Handouts folder to include what we've added in class. *Copy* the current version from Handouts to your Home directory.

3. *Load* your copy of "graphs.sage". Run: `load('graphs.sage')`.

4. Run: `my_graphs` to see what graphs we have so far. (This is partly a test to make sure your file is loaded.)

Concepts from Our Text

These include: *size, order, complete graphs, bipartite-ness, isomorphic graphs, subgraph, complement, incidence matrix, adjacency matrix, degrees, minimum degree, maximum degree, identical graphs, connectedness, number of components, tree-ness, distances, cut vertices, cut edges, finding spanning trees, Eulerian-ness, Hamiltonian-ness, finding Eulerian and Hamiltonian cycles.*

These are all built-in to Sage/Cocalc!

5. Use the built-in functions to find cut vertices, find cut edges, find spanning trees, test for Eulerian-ness and Hamiltonian-ness, and find Eulerian and Hamiltonian cycles.

6. Write a function that tests if a graph has the "Dirac property" (that is, that $\delta \geq \frac{\nu}{2}$) and, thus, has a Hamiltonian cycle.

7. Write a function that *finds* a minimum total weight Hamiltonian cycle if one exists (that is, that *solves* the Traveling Salesman Problem).

Independent Sets

The *independence number* of a graph is the largest number of vertices in the graph that have no edges between them. So if `C5=graphs.CompleteGraph(5)`, its independence number is 1.

Now we will write an algorithm to find the independence number. We will write it as an ordinary function (rather than as a Graph method). The first thing we need to be able to do is to test whether the vertices S from a graph G are *independent*. This means there are no edges between the vertices in S. So we need to *search* through the edges of G.

8. Evaluate `pete.edges(labels=False)` to get a list of the edges of the Petersen graph. Evaluate `E=pete.edges(labels=False)` to give this list the name E. Evaluate `(0,1) in E` to test if $(0, 1)$ is the collection of edges E. Now check if $(0, 2)$ is an edge.

9. Now we'll test every pair i and j from a set of vertices S to check if S is independent. If S is independent then the test for (i, j) will be false for each possible pair.

```
def is_independent(G, S):
    E=G.edges(labels=False)
    for i in S:
        for j in S:
            if (i,j) in E:
                return False
    return True
```

10. Use `is_independent()` to test if the sets $[1, 2, 3]$ and $[1, 2]$ are independent in $p3$. Find the largest independent set S you can find in the Petersen graph. Test if it is independent.

    One way to find a largest independent set in a graph is to test every subset of vertices, check if it is independent, and then keep track of the largest one you've seen up to that point.

11. We'll use the subsets generator of a list to run through the subsets. Let `V=p3.vertices()`. Evaluate V to see what you have. lets see how it works. Let `L=Subsets(V)`. Now try:

```
for S in L:
    print S
```

12. Now we'll write our function to find a maximum independent set of a graph.

```
def maximum_independent_set(G):
    independent = []
    L=Subsets(G.vertices())
    for S in L:
        if is_independent(G,S)==True:
            if len(S) > len(independent):
                independent = S
    return independent
```

13. Use this function to find a maximum independent set of the Petersen graph.