Last name _____

First name _____

LARSON—MATH 756–SAGE WORKSHEET 01
Getting Started with Sage/Colcalc.

1. Create a Sage/Cocalc account.

   (a) Start the Chrome browser.

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

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

   (d) If you don't see a Project, ask me and we'll get the right email address associated with your account.

   (e) Click "New", then "Worksheets", then call it **s01**.

   Graphs in Sage

   Sage includes the `graphs` class which contains a number of *methods*. Some of these include constructors for making well-known graphs. "pete" is an arbitrarily chosen name in what follows. We could use G or anything else instead!

2. Evaluate:

```
pete=graphs.PetersenGraph()
pete.show()
```

   Here is a list of common graphs that have pre-built constructors: `http://doc.sagemath.org/html/en/reference/graphs/sage/graphs/graph_generators.html`

   Find another one of these that interests you and `show` it.

   There are also graph *classes*: there require one or more *parameters*. So if you want a cycle on five vertices named "c5" use: `c5 = graphs.CycleGraph(5)`. Now `show` it.

   Complete partite graphs are also built it. If you want the complete bipartite graph $K_{2,3}$ on partite sets of cardinalities 3 and 4 named "`k_2_3`" use:
   `k_2_3 = graphs.CompleteBipartiteGraph(2,3))`. Now `show` it.

3. We can create our own graph using the `Graph()` constructor, and the `add_vertex()` and `add_edge()` methods. Lets make a `cycle` on 5 vertices. First initialize the graph and make the vertices:

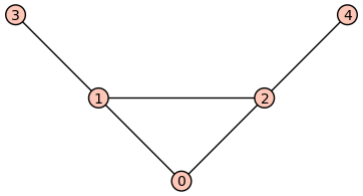
```
g=Graph()
for i in [1..5]:
    g.add_vertex()
g.show()
```

Notice that the vertex labels start at 0. Now make the edges:

```
for i in [0..3]:
    g.add_edge(i,i+1)
g.show()
```

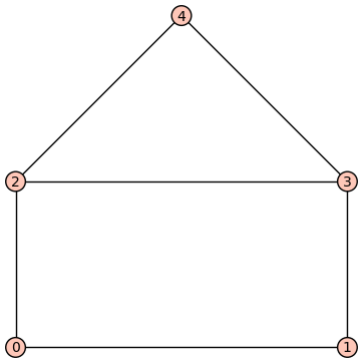
You're still missing an edge. What will you add to the code to get the missing edge?

4. Now use `Graph()`, `add_vertex()` and `add_edge()` to make the *bull*:



Start by letting `bull=Graph(5)`. (Instead of using `add_vertex()`, you can start with `Graph(5)` to get a graph with 5 vertices and no edges.) Now add the edges that you see in the diagram of the bull using `bull.add_edge()`. Remember that the layout of the graph doesn't matter—only that it has the same edges. Write the code that worked. When you are done you can view it with `bull.show()`.

5. Make the following graph, called "the house". Start by letting `house=Graph(5)`. Write the code that worked. When you are done you can view it with `house.show()`.



Graph Representations in Sage

One way to represent a graph with order n is with an $n \times n$ *adjacency matrix* A. If the vertices of the graph are $\{v_0, v_1, \ldots, v_{n-1}\}$ (or {1,2,...,n-1} for short) then the $A_{i,j}$ is 1 if there is an edge from vertex i to vertex j, and 0 if there is not.

Graphs can be represented by—and created from—(adjacency, incidence) matrices. Here's how to enter the matrix $A = \begin{bmatrix} 2 & 1 & 5 \\ 1 & 3 & 7 \end{bmatrix}$

Evaluate: `A=matrix(2,3,[2, 1, 5, 1, 3, 7])`, then evaluate A to see what Sage thinks A is.

6. Here's how to *get* the adjacency matrix of an existing graph in Sage. Evaluate:

```
pete=graphs.PetersenGraph()
pete.adjacency_matrix()
pete.show()
```

Write the matrix. Check the pattern of 0's and 1's.

Here's how to make a graph from an adjacency matrix M: $M = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$

7. To enter M into Sage evaluate: `M=matrix(4,4,[0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0])`. Now evaluate M to check what Sage thinks M is.

8. To get a graph from M we can use the super-powerful graph *constructor* `Graph`. Evaluate: `g=Graph(M)`. Then `show` g to see the graph you constructed.

9. You can also construct a graph from an incidence matrix. This is the incidence matrix of a graph. Draw the graph. $\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$

10. Now evaluate `g2=Graph(matrix(3,2,[1, 0, 1, 1, 0, 1]))` and then `show` $g2$ to see what you have.

11. You can also make a graph using just a list of edges as input. Evaluate: `g3=Graph([(0, 1), (0, 3), (1, 2), (2, 3)])` and then `show` $g3$ to see what you have.

12. To see more examples of the kinds of inputs the `Graph` constructor can use to make graphs from various inputs, look at the *help* for this function. Evaluate: `Graph?`.

Graph Independence in Sage

13. To find a maximum independent set in the Petersen graph, evaluate `pete.independent_set()`. What do you get?

    To find out the options for this function, evaluate `pete.independent_set?`. One of the options shows you how to return the independence number α.

14. According to the help you read, we can use `pete.independent_set(value_only=True)`. Evaluate.

    If we wanted an independence number function we could use this. Evaluate the following code:

```
def independence_number(g):
    return g.independent_set(value_only=True)
```

15. Now try `independence_number(pete)`. What did you get?

16. What can you type to find the independence number of $K_{3,4}$?

Graph Invariants in Sage

An *invariant* is a number associated with a graph (or a function which returns a number); it may be an integer or a real number, etc. The independence number of a graph is a graph invariant. The *order* of a graph is the number of vertices it has. The *size* of a graph is the number of edges it has. How many vertices and edges does the Petersen graph have? Evaluate `pete.order()` and `pete.size()`.

17. Find the order and size of the icosahedron graph. Use `icos=graphs.IcosahedralGraph()`.

18. Find the order and size of the dodecahedron graph. Use `dode=graphs.DodecahedralGraph()`.

19. Find the order and size of the tetrahedron graph. Use `tetra=graphs.TetrahedralGraph()`.

20. Find the order and size of the octahedral graph. Use `octa=graphs.OctahedralGraph()`.

    Now let's compute some invariants. While we can write our own algorithms, Sage often has the state-of-the-art (fastest) algorithms built-in.

    We'll start by defining some built-in graphs in Sage and then use Sage to calculate the four invariants: α, α', β, and β'.

    The vertex covering number β is not a built-in Sage function. But using the Gallai identities, and our independence number function, we can write our own function to calculate β:

21. Now try `vertex_covering_number(pete)`. What did you get?

22. What can you type to find the vertex covering number of $K_{3,4}$?

23. To find a maximum matching in the Petersen graph, evaluate `pete.matching()`. What do you get?

    To find out the options for this function, evaluate `pete.matching?`. One of the options shows you how to return the matching number α'.

    If we wanted a matching number function we could define the following function. Evaluate.

```
def matching_number(g):
    return g.matching(value_only=True)
```

24. Now try `matching_number(pete)`. What did you get?

25. What can you type to find the matching number of $K_{3,4}$?

    The edge covering number β' is not a built-in Sage function. But using the Gallai identities, and our matching number function we can write our own function to calculate ρ:

```
def edge_covering_number(g):
    return g.order() - matching_number(g)
```

26. Now try `edge_covering_number(pete)`. What did you get?

27. What can you type to find the edge covering number of $K_{3,4}$?

28. Lovasz's ϑ is built-in. Evaluate: `pete.lovasz_theta()` to find that value.

29. Eigenvalues are built-in to Sage. To get the eigenvalues of the adjacency matrix of the Petersen graph, evaluate: `pete.adjacency_matrix().eigenvalues()`.

30. Eigenvalues aren't by definition graph invariants. But things like the largest eigenvalue are indeed graph invariants. So is the number of non-negative eigenvalues. (This is in fact an upper bound for the independence number of a graph). Write a function whose input is a graph and whose output is the number of non-negative eigenvalues.

31. The *Cvetkovic bound* is the minimum of the number of non-negative or non-positive eigenvalues. This too an upper bound for the independence number of a graph (which we'll prove soon enough). Write a function whose input is a graph and whose output is the Cvetkovic bound.

32. See how many invariants you can identify:

    `http://doc.sagemath.org/html/en/reference/graphs/sage/graphs/generic_graph.html`.

Graph Properties in Sage

A *property* is a boolean (or a function which returns a boolean): True or False.

33. Is the Petersen graph bipartite? Sage has a built-in test to check. Evaluate `pete.is_bipartite()`. What do you get?

34. Is $K_{3,4}$ bipartite? Use Sage to check.

35. Other built-in properties (so these are *methods*) include `is_hamiltonian` and `is_clique`. Use Sage to test if the Petersen graph is hamiltonian.

    Here's a list of *some* graph properties that are built-in in Sage:

    `http://doc.sagemath.org/html/en/reference/graphs/sage/graphs/generic_graph.html`.