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) Click “New”, then “Worksheets”, then call it s04.

   **We will attempt to generate the poset of all connected subgraphs of the cycle on 5 vertices.**

2. Create our graph. Evaluate: \( c5 = \text{graphs.CycleGraph}(5) \). Run: \( c5\text{.show()} \) to see the labelings, etc.

3. Get the set of edges of \( c5 \) and call them \( E \). Run: \( E=\text{c5\text{.edges()}} \). Then evaluate \( E \).

4. Note the mysterious \texttt{none} in each edge tuple. These are spots for labels (weights or names, etc). Let’s get rid of these. Run: \( E=\text{c5\text{.edges(labels=False)}} \). Then evaluate \( E \).

5. Each connected subgraph will consist of a subset of these edges. Unfortunately the built-in subgraph constructor will take a set of edges and generate the subgraph with all the vertices together with the edges we want. Let’s grab the vertices out of any set of edges:

   ```python
   def vertices(edge_set):
       S = Set({})
       for e in edge_set:
           if e[0] not in S:
               S = S.union(Set([e[0]]))
           if e[1] not in S:
               S = S.union(Set([e[1]]))
       return S
   ```

6. Let’s test it. We’ll generate all the subsets of edges, print those and the corresponding vertex list.

   ```python
   for S in Subsets(E):
       print S
       print "vertices are {}"\text{.format(\text{vertices(S)})}
   ```

7. Now we can generate all subgraphs with a given edge set.
for S in Subsets(E):
    print S
    c5.subgraph(vertices(S),S).show()

8. Now let’s generate all edge lists that define connected subgraphs of \( c5 \) and put them in a list.

```python
def edges_connected_subgraphs(g):
    connected = []
    E = g.edges(labels=False)
    for S in Subsets(E):
        V = vertices(E)
        H = g.subgraph(V,edges=S)
        if H.is_connected():
            connected.append(S)
    return connected
```

9. To see the output for \( c5 \) you can give the output list a name and then show all of the graphs that are in that list:

```python
connected = edges_connected_subgraphs(c5)
for S in connected:
    h = g.subgraph(vertices(S), S)
    h.show()
```

10. Unfortunately the Poset constructor requires immutable objects as inputs. We could make our graphs immutable, but since we will always be dealing with subgraphs of a particular graph—defined by edge lists—it may be better to define our graph posets using the edge lists directly: we can simply define the poset relation in terms of edge set inclusion.

Run:
```
CyclePo = Poset((edges_connected_subgraphs(c5), lambda x,y:x.issubset(y))
```
Then view it. Run: `view(CyclePo)`.

11. Now let’s return to what we figured out last week: can you find a maximum antichain, maximum chain, the height and width of this poset, etc.

12. What could you do to generate the posets of connected subgraphs of \( c5 \) with no more than 3 edges? (When we talk about larger graphs we may only be able to usefully generate “small” subgraphs).

13. To create the (built-in) Petersen graph is Sage and call it “pete”, evaluate (run):
```
pete=graphs.PetersenGraph()
```
, and then to see what the graph looks like evaluate `pete.show()`.

14. Can you construct the poset of connected subgraphs of the Petersen graph?