Last name _____

First name _____

LARSON—MATH 556—SAGE WORKSHEET 07
An Independence Number Algorithm

1. Log in to your Sage/Cocalc account.

   (a) Start Chrome browser.

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

   (c) Click "Sign In".

   (d) Click project **Classroom Worksheets**.

   (e) Click "New", call it **s07**, then click "Sage Worksheet".

   The *independence number* of a graph is the largest number of points in the graph that have no edges between them. The next big idea for finding a maximum independent set was due to Tarjan and Trojanowski in the 1970s: they noted that each vertex v of a graph is either in a maximum independent set *or* it is not. And, if v is in a maximum independent set then none of the points it is touching (that it is *adjacent* to is), called the *neighbors* of v, can be in that set.

   So let's find the neighbors of a vertex v in a graph g.

```
def neighbors(g,v):
    N = []
    E = g.edges(labels=False)
    for w in g.vertices():
        if (v,w) in E or (w,v) in E:
            N.append(w)
    return N
```

2. Let `g=graphs.PetersenGraph()`. Find the neighbors of vertex 0 in g. Use `g.show()` to check. Find the neighbors of vertex 9 in the Petersen graph.

   So our problem of finding a maximum independent set in a graph can be *reduced* to the problem of finding a maximum independent set in two smaller subgraphs: (1) the graph formed by removing vertex v and (2) the graph formed by removing v and its neighbors. In this case, we assume that v is in the maximum independent set.

3. Now we need to form these graphs. Let g be a graph with vertex set V. Let S be any subset of V. Then you can find the graph formed by S together with all the edges that are between points of S in the original graph g with the command `g.subgraph(S)`. This is a new graph. We can give it a name, say h by `h=g.subgraph(S)`. Let `S=[2,3,5,7,8]`. Now try `h=g.subgraph(S)` and then `h.show()`.

4. Lets see the graphs that need to be formed when we apply the Tarjan-Trojanowski idea to vertex 0 of the Petersen graph. We'll need to form two sets $S1$ and $S2$ and the corresponding graphs. $S1$ is all the points of g except 0 and $S2$ is all the points of g except 0 and its neighbors.

Try `S1=g.vertices()`, `S1.remove(0)`. Now evaluate $S1$ to see this set. Then try `h=g.subgraph(S1)` and then `h.show()`.

5. Now removing v and its neighbors will require more work:

```
S2=g.vertices()
S2.remove(0)
for w in neighbors(g,0):
    S2.remove(w)
```

Evaluate $S2$ to see this set. Now try `h=g.subgraph(S2)` and `h.show()`.

6. To simplify things in the future, we should write a function to remove a vertex and its neighbors from a graph and produce a new graph with v and its neighbors removed.

```
def remove_vertex_and_neighbors(g,v):
    S2=g.vertices()
    S2.remove(v)
    for w in neighbors(g,v):
        S2.remove(w)
    return g.subgraph(S2)
```

Try `remove_vertex_and_neighbors(g,0)`. How come it didn't do anything???

7. Remember what happens to V and the graph's points when we `pop()` a vertex off of the end of V. Evaluate: `g.vertices()`, then `V = g.vertices()`, then `v = V.pop()`, then `V`, and finally `g.vertices()`.

8. Now we are ready to write our new maximum independent set function. We will need two new vertex sets $S1$ and $S2$. Note that this function is *recursive*.

```
def tt_maximum_independent_set(g, IndependentSet):
    V = g.vertices()
    if V == []:
        return IndependentSet
    v = V.pop()
    S1 = V
    S2 = remove_vertex_and_neighbors(g,v)
    g1 = g.subgraph(S1)
    g2 = g.subgraph(S2)
    Max1 = tt_maximum_independent_set(g1, IndependentSet)
    Max2 = tt_maximum_independent_set(g2, IndependentSet+[v])
    if len(Max1) > len(Max2):
        return Max1
    else:
        return Max2
```

Try `tt_maximum_independent_set(g, [])`. Now **test** this function with a variety of other graphs where you `know` the answer. Does it work?