Last name _____

First name _____

LARSON—MATH 756–SAGE WORKSHEET 03
Getting Started with Sage/Colcalc.

1. Login to your Sage/Cocalc account.

   (a) Start the Chrome browser.

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

   (c) Login. You created a new Project for our class. Click on that.

   (d) Click "New", then "Worksheets", then call it **s03**.

   Review

   (**Cut and paste any needed relevant definitions from worksheets s01 and s02 into this Sage Worksheet.**)

2. What can you run to find the independence number of $K_{3,4}$? (What code will you enter?)

3. What can you run to check if $K_{3,4}$ is bipartite? (What code will you enter?)

   More Graph Invariants in Sage

   We'll 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:

```
def vertex_covering_number(g):
    return g.order() - independence_number(g)
```

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

5. Find the vertex covering number of $K_{3,4}$?

6. 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)
```

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

8. 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)
```

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

10. Find the edge covering number of $K_{3,4}$

Searching Graphs In Sage

What is the average independence number of a connected graph on 4 vertices? We could generate them all, count them, keep a running total, and then produce the average:

```
count = 0.0
total = 0.0
for g in graphs.nauty_geng("4 -c"):
    count = count + 1
    total = total + independence_number(g)
print "answer = {}".format(total/count)
```

11. What is the average independence number of a connected graph on 4 vertices?

12. What is the average independence number of a connected graph on 7 vertices?

13. For what proportion of graphs on 7 vertices does Lovasz's ϑ equal the independence number?

A way to estimate averages is to generate random graphs. Evaluate:

```
total = 0.0
for n in [1..1000]:
    g=graphs.RandomGNP(7,.5)
    total = total + independence_number(g)
print "answer = {}".format(total/1000)
```

14. What is your result from this experiment?

15. What might it be a little higher than the "whole truth" you previously computed?

16. What can you do to modify the previous code to more closely approximate the whole truth?

Integer and Linear Programming in Sage

Here's the linear program we'd like to solve:

maximize: $x_1 + x_2 + x_3$

$$\begin{array}{lccccccc} & x_1 & + & x_2 & & & \leq & 1 \\ \text{subject to:} & x_1 & & & + & x_3 & \leq & 1 \\ & & & x_2 & + & x_3 & \leq & 1 \end{array}$$

We will let `LP` be the name of our linear program. Of course, the name can be *anything*; it can be `Dantzig`, `D` or `Mathzilla`, anything that you are not already using for something else.

We will also tell Sage that our objective is to find the **maximum** value of the objective function.

17. Evaluate `LP = MixedIntegerLinearProgram(maximization=True)`

Now we will let x be the name of the variable vector, and also require that the vector entries be non-negative.

18. Evaluate `x = LP.new_variable(nonnegative=True)`.

Now we will tell Sage what the objective function is. Notice that we are also implicitly giving the components of vector x the names $x[1]$, $x[2]$ and $x[3]$.

19. Evaluate `LP.set_objective(x[1] + x[2] + x[3]))`.

Now lets add our constraints.

20. Evaluate:

```
LP.add_constraint(x[1] + x[2], max = 1)
LP.add_constraint(x[1] + x[3], max = 1)
LP.add_constraint(x[2] + x[3], max = 1)
```

21. Now evaluate `LP.solve()` to solve the linear program. What do you get?

    This should print the maximum possible value of the objective function. (And there is probably a teeny error - all LP solvers are subject to some numerical instability.) All the work was done in the last step. If the LP is big this could take some time. Now let's see a feasible solution that attains the optimal value.

22. Evaluate `LP.get_values(x)`. What do you get?

    If you want only integer solutions for x—turning our problem into an Integer Programming problem. We can set x to be integer.

23. Evaluate `LP.set_integer(x)`. Then we can resolve. Evaluate `LP.solve()` and `LP.get_values(x)` again. What do you get? What does it mean?

    For small graphs we can calculate the independence number by solving an integer program (IP). Let x_1, x_2 and

    maximize: $x_1 + x_2 + x_3$

$$\begin{array}{rrrrrrr} & x_1 & + & x_2 & & \leq & 1 \\ \text{subject to:} & x_1 & & & + \; x_3 & \leq & 1 \\ & & & x_2 & + \; x_3 & \leq & 1 \end{array}$$

    $x_1, x_2, x_3 \in \{0, 1\}$.

    Because the variables are restricted to integers this is an *Integer Programming* (IP) problem. If we allow the variables to be real numbers instead of integers (that is, to *relax* the IP) we get a linear program (LP). The relaxation of the independent set IP gives an optimum value that is necessarily an upper bound for the vertex packing number. Here's how we can write this in Sage in a general way.

```
def fractional_independence_LP(g):

    p = MixedIntegerLinearProgram(maximization=True)
    x = p.new_variable(nonnegative=True)
    p.set_objective(sum(x[v] for v in g.vertices()))

    for v in g.vertices():
        p.add_constraint(x[v], max=1)

    for (u,v) in g.edge_iterator(labels=False):
        p.add_constraint(x[u] + x[v], max=1)

    return p.solve()
```

24. Type in the above definition and evaluate.

25. The graph represented by the IP above is the complete graph $k3$ on three vertices. Evaluate the following to get an upper bound for the independence number of $k3$:

```
k3=graphs.CompleteGraph(3)
fractional_independence_LP(k3)
```

26. Now find the FILP upper bound for the Petersen graph Evaluate:
`fractional_independence_LP(pete)`

27. It is possible to find LP bounds for graphs where calculating the true independence number may be practically impossible. Let g be a random graph on 500 vertices. Evaluate: `g=graphs.RandomGNP(500, 0.5)`? What do you expect the upper bound to be. Now calculate.