Last name _____

First name _____

LARSON—MATH 750–SAGE WORKSHEET 05
Linear Extensions, Zeta and Mobius Functions

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 **s05**.

   The goal of today's lab is to investigate the linear extension, zeta and mobius function poset methods in Sage.

   We have 3 posets now to experiment with: $P1$, the poset $(\mathcal{P}([3]), \subseteq)$ of the subsets of [3] with inclusion; $P2$, the poset $([10], |)$ of the integers [10] with the divisibility relation; and $P3$, the poset of all connected subgraphs of the cycle on 5 vertices.

2. Instead of regenerating these I've put the initialization code in the file `posets.sage` in your Handouts folder. Copy that to your main folder (the Handouts version will change when I change it. The version in your main directory will only change when you change it).

3. In your $s05$ worksheet, run: `load('posets.sage')`.

4. Run: `P1` to check that you have a poset object with that name. Repeat for $P2$ and $P3$.

5. Recall that can you find a maximum anti-chain, maximum chain, the height and width of this poset, view it, show it, etc. These may be useful commands that you may want to review from previous worksheets.

6. Off today's main topic, lets see that Sage can find things like a chain partition guaranteed by Dilworth's Theorem. Find the width of $P1$ and then run: `P1.dilworth_decomposition()`. Check that the output is correct.

7. Repeat this for $P2$ and $P3$.

8. Now let's find linear extensions of our posets. For $P1$ run: `P1.linear_extension()`. The output is a *list*, that is, an object that has a built-in order. Check that, indeed, the earlier items in the list are never more than the later items in the list (it won't necessarily be true that earlier items are comparable).

9. Now that we've seen it, let's give this linear extension a name—so we can *use* it. Run: `L1=P1.linear_extension()`.

10. We can automate run our visual check that this is indeed a linear extension:

```
card1 = len(L1)
for i in range(card1):
    for j in range(card1):
        if i < j:
            bool1 = P1.is_lequal(L1[i],L1[j])
            bool2 =  P1.compare_elements(L1[i],L1[j]) == None
            print L1[i], L1[j], bool1 or bool2
```

11. The `is_lequal` is a built-in zeta function that will compare any pair of elements of a poset. It returns `False` for incomparable elements. Let's make it integer-valued, and then test it.

```
def zeta(P, x, y):
    if P.is_lequal(x,y):
        return 1
    else:
        return 0

card1 = len(L1)
for i in range(card1):
    for j in range(card1):
        print L1[i], L1[j], zeta(P1, L1[i], L1[j])
```

12. There is also a built-in *mobius* function. Let's test it:

```
for i in range(card1):
    for j in range(card1):
        print L1[i], L1[j], P1.moebius_function(L1[i], L1[j])
```

13. Are the zeta and mobius functions inverses with respect to the convolution product? We'd need to define the convolution product to check.

    Challenge. Define a function `convolution_product(P, f, g, x, y)` that takes a poset P and 2-valued functions f and g, elements x and y from the ground set and returns $(f * g)(x, y)$.

14. Repeat steps (8) through (12) for the poset $P2$.

15. Repeat steps (8) through (12) for the poset $P3$.