Last name _____

First name _____

LARSON—MATH 310–HOMEWORK WORKSHEET 08
Vector and Matrix Applications.

1. Start the Chrome browser.
2. Go to `https://cocalc.com`
3. Log in.
4. You should see an existing Project for our class. Click on that.
5. Copy the folder `CodingTheMatrix-fixed` to your Home directory. This has the new version of `solver`.
6. Make sure you are in your own `CodingTheMatrix-fixed` directory (if you work in the Handouts directory version, your work could get overwritten).
7. Click "New", then "Jupyter Notebook", then call it **310-h08**. This file should be in your `CodingTheMatrix-fixed` directory.
8. Make sure you have PYTHON as the *kernel*.

Annotate your Jupyter notebook cells so that it is clear to anyone what problem your work corresponds to.

1. We need to import various modules/libraries.

```
from solver import *
from GF2 import *
from vec import *
from matutil import *

```

2. Make the vectors, domain D, and linear combination in the following example.

Example 3.1.6: *Products and resources:* The JunkCo factory makes things using five resources: metal, concrete, plastic, water, and electricity. Let D be this set of resources. The factory has the ability to make five different products.





Here is a fabricated table that shows how much of each resource is used in making each product, on a per-item basis:

	metal	concrete	plastic	water	electricity
garden gnome	0	1.3	.2	.8	.4
hula hoop	0	0	1.5	.4	.3
slinky	.25	0	0	.2	.7
silly putty	0	0	.3	.7	.5
salad shooter	.15	0	.5	.4	.8

The i^{th} product's resource utilization is stored in a D-vector $\boldsymbol{v}_i$ over $\mathbb{R}$. For example, a gnome is represented by

$$\boldsymbol{v}_{\text{gnome}} = \texttt{Vec(D,\{'concrete':1.3,'plastic':.2,'water':.8,'electricity':.4\})}$$

Suppose the factory plans to make α_{gnome} garden gnomes, α_{hoop} hula hoops, α_{slinky} slinkies, α_{putty} silly putties, and α_{shooter} salad shooters. The total resource utilization is expressed as a linear combination

$$\alpha_{\text{gnome}}\,\boldsymbol{v}_{\text{gnome}} + \alpha_{\text{hoop}}\,\boldsymbol{v}_{\text{hoop}} + \alpha_{\text{slinky}}\,\boldsymbol{v}_{\text{slinky}} + \alpha_{\text{putty}}\,\boldsymbol{v}_{\text{putty}} + \alpha_{\text{shooter}}\,\boldsymbol{v}_{\text{shooter}}$$

For example, suppose JunkCo decides to make 240 gnomes, 55 hoops, 150 slinkies, 133 putties, and 90 shooters. Here's how the linear combination can be written in Python using our `Vec` class:

```
>>> D = {'metal','concrete','plastic','water','electricity'}
>>> v_gnome = Vec(D,{'concrete':1.3,'plastic':.2,'water':.8,'electricity':.4})
>>> v_hoop = Vec(D, {'plastic':1.5, 'water':.4, 'electricity':.3})
>>> v_slinky = Vec(D, {'metal':.25, 'water':.2, 'electricity':.7})
>>> v_putty = Vec(D, {'plastic':.3, 'water':.7, 'electricity':.5})
>>> v_shooter = Vec(D, {'metal':.15, 'plastic':.5, 'water':.4,'electricity':.8})

>>> print(240*v_gnome + 55*v_hoop + 150*v_slinky + 133*v_putty + 90*v_shooter)
```

3. Make the row dictionary, corresponding matrix M and print it, in the next problem.
4. Then, from the same problem, make the vector $\hat{u}$ and vector-matrix product $\hat{u} * M$.

Example 4.5.10: In Section 3.1.2, we gave examples of applications of linear combinations. Recall the JunkCo factory data table from Example 3.1.6 (Page 145):

	metal	concrete	plastic	water	electricity
garden gnome	0	1.3	.2	.8	.4
hula hoop	0	0	1.5	.4	.3
slinky	.25	0	0	.2	.7
silly putty	0	0	.3	.7	.5
salad shooter	.15	0	.5	.4	.8

Corresponding to each product is a vector. In Example 3.1.6 (Page 145), we defined the vectors

v_gnome, v_hoop, v_slinky, v_putty, and v_shooter,

each with domain

{'metal','concrete','plastic','water','electricity'}

We can construct a matrix M whose rows are these vectors:

```
>>> rowdict = {'gnome':v_gnome, 'hoop':v_hoop, 'slinky':v_slinky,
               'putty':v_putty, 'shooter':v_shooter}
>>> M = rowdict2mat(rowdict)
>>> print(M)

            plastic metal concrete water electricity
           ----------------------------------------
  putty  |      0.3     0        0   0.7         0.5
  gnome  |      0.2     0      1.3   0.8         0.4
  slinky |        0  0.25        0   0.2         0.7
    hoop |      1.5     0        0   0.4         0.3
 shooter |      0.5  0.15        0   0.4         0.8
```

In that example, JunkCo decided on quantities $\alpha_{\text{gnome}}, \alpha_{\text{hoop}}, \alpha_{\text{slinky}}, \alpha_{\text{putty}}, \alpha_{\text{shooter}}$ for the products. We saw that the the vector giving the total utilization of each resource, a vector whose domain is $\{$*metal, concrete, plastic, water, electricity*$\}$, is a linear combination of the rows of the table where the coefficient for product p is α_p.

We can obtain the total-utilization vector as a vector-matrix product

$$[\alpha_{\text{gnome}}, \alpha_{\text{hoop}}, \alpha_{\text{slinky}}, \alpha_{\text{putty}}, \alpha_{\text{shooter}}] * M \tag{4.1}$$

Here's how we can compute the total utilization in Python using vector-matrix multiplication. Note the use of the asterisk $*$ as the multiplication operator.

```
>>> R = {'gnome', 'hoop', 'slinky', 'putty', 'shooter'}
>>> u = Vec(R, {'putty':133, 'gnome':240, 'slinky':150, 'hoop':55,
                'shooter':90})
>>> print(u*M)

 plastic metal concrete water electricity
----------------------------------------
     215    51      312   373         356
```

5. Now we can use/test the `solve` procedure!

Example 4.5.15: We use `solve(A,b)` to solve the industrial espionage problem. Suppose we observe that JunkCo uses 51 units of metal, 312 units of concrete, 215 units of plastic, 373.1 units of water, and 356 units of electricity. We represent these observations by a vector $\boldsymbol{b}$:

```
>>> C = {'metal','concrete','plastic','water','electricity'}
>>> b = Vec(C, {'water':373.1,'concrete':312.0,'plastic':215.4,
                'metal':51.0,'electricity':356.0})
```

We want to solve the vector-matrix equation $\boldsymbol{x} * M = \boldsymbol{b}$ where M is the matrix defined in Example 4.5.10 (Page 196). Since `solve(A,b)` solves a matrix-vector equation, we supply the transpose of M as the first argument A:

```
>>> solution = solve(M.transpose(), b)
```

6. Now test your solution vector.

```
>>> print(solution)

 putty gnome slinky hoop shooter
--------------------------------
   133   240    150   55      90
```

Does this vector solve the equation? We can test it by computing the *residual vector* (often called the *residual*):

```
>>> residual = b - solution*M
```

If the solution were exact, the residual would be the zero vector. An easy way to see if the residual is *almost* the zero vector is to calculate the sum of squares of its entries, which is just its dot-product with itself:

```
>>> residual * residual
1.819555009546577e-25
```

About 10^{-25}, so zero for our purposes!

However, we cannot yet truly be confident we have penetrated the secrets of JunkCo. Perhaps the solution we have computed is not the only solution to the equation! More on this topic later.

Getting your homework recorded

When you are done,

1. Click the "Print" menu choice (under "File") and make a pdf of this worksheet (html is OK too).
2. Send me an email (`clarson@vcu.edu`) with an informative header like "Math 310 - h08 worksheet attached" (so that it will be properly recorded).
3. Remember to attach your homework worksheet!