Quick Start

Check Installation

The installation command:

```
setup.py install
```

will probably put Matalg.py and testMat.py in the following directory on your ubuntu PC:

```
/usr/local/lib/python3.2/dist-packages/matalg/
```

To check the installation, start python3.2 or higher as follows:

```
python3
```

Now execute the following commands from the Python shell:

```
>>> from matalg import Matalg as _m
```

```
>>> from matalg import testMat
This works, if no failures reported...
```

You are ready for the journey

Try Matrix multiplication

Let us create two smallest matrices, **amat** and **bmat**:

```
>>> Matrix = _m.Matrix
>>> amat = Matrix(2, 2)
>>> bmat = Matrix(2, 2)
```

The two (2 x 3) matrices have zero terms. We can put any numerical values in the terms. For instance

```
>>> amat[0, 0] = 7
```

will make the first term equal to 7. Verify it:

```
>>> print(amat)

[7, 0.0]
[0.0, 0.0]
```

That does not look nice, so let us print it in a neater format:

```
>>> amat.neatprint()
A matrix of dimensions (m x n), where m, n, LineLen =  2 2 5
   7.00000E+00   0.00000E+00
   0.00000E+00   0.00000E+00
```

We can similarly fill all the terms with non zero values, either the same way as we just did for amat[0, 0] or using a function **enterdata** as follows:

```
>>> amat = _m.enterdata(2, 2, [[7, 6], [5, 3]], False)
>>> bmat = _m.enterdata(2, 2, [[1, 2], [3, 4]], False)
```

We have recreated amat and bmat and put some values in it. **False** simply signals to enterdata *not* to echo the data.

To multiply the two matrices and store the result in cmat, write:

```
>>> cmat = amat * bmat
```

Let us check the data and the results:

```
A matrix of dimensions (m x n), where m, n, LineLen =  2 2 5
  7.00000E+00   6.00000E+00
  5.00000E+00   3.00000E+00
>>> bmat.neatprint()
A matrix of dimensions (m x n), where m, n, LineLen =  2 2 5
  1.00000E+00   2.00000E+00
  3.00000E+00   4.00000E+00
>>> cmat.neatprint()
A matrix of dimensions (m x n), where m, n, LineLen =  2 2 5
  2.50000E+01   3.80000E+01
  1.40000E+01   2.20000E+01
```

I trust the Matalg, but please verify the matrix multiplication on a piece of paper...

Let us now multiply a matrix by a scalar:

```
>>> xscl = 2.1
>>> amat = xscl * amat
>>> amat.neatprint()
A matrix of dimensions (m x n), where m, n, LineLen =  2 2 5
  1.47000E+01   1.26000E+01
  1.05000E+01   6.30000E+00
```

Mental arithmetic will show that the matrix amat has been correctly scaled. Post multiplication has exactly the same effect.

Usually matrices are scaled in situ, so to multiply bmat by the same scalar we can write in shorthand:

```
>>> xscl * bmat
[[2.1, 4.2], [6.300000000000001, 8.4]]
>>> bmat.neatprint()
A matrix of dimensions (m x n), where m, n, LineLen =  2 2 5
  2.10000E+00   4.20000E+00
  6.30000E+00   8.40000E+00
```

Other Matrix Operations

I have used the terminal to enter these commands, but they can be tested in the IDLE Python Shell just as easily and actually a little more conveniently.

Let us recreate the amat and bmat before they were scaled by scalar multiplication:

```
>>> amat = _m.enterdata(2, 2, [[7, 6], [5, 3]])
Echo check of enterdata
[7, 6]
[5, 3]
>>> bmat = _m.enterdata(2, 2, [[1, 2], [3, 4]])
Echo check of enterdata
[1, 2]
[3, 4]
```

By default, the last parameter is **True** and signals to the enterdata method to echo print the data.

Matrix Addition

To add the two amat and the bmat matrices, write:

```
>>> cmat = amat + bmat
>>> cmat.neatprint()
A matrix of dimensions (m x n), where m, n, LineLen =  2 2 5
  8.00000E+00   8.00000E+00
  8.00000E+00   7.00000E+00
```

OK, now see what happens if we try an operation that is not defined in matrix algebra - try to add a scalar and a matrix:

```
>>> cmat = amat + xscl
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'Matrix' and 'float'
```

So Python has saved us again from doing something illegal!

Matrix Subtraction

Subtraction is very similar to addition:

```
>>> cmat = amat - bmat
>>> cmat.neatprint()
A matrix of dimensions (m x n), where m, n, LineLen =  2 2 5
  6.00000E+00   4.00000E+00
  2.00000E+00  -1.00000E+00
```

Solution of Simultaneous Equations

This is probably the most important matrix operation of all. To signal the solution of simultaneous equations, we borrowed the two starts, which in scalar arithmetic would signal raising to a power. So if we define **rhs** as the right had matrix and specify it as

```
>>> rhs = _m.enterdata(2, 1, [[10], [1]])
Echo check of enterdata
[10]
[1]
```

we can solve the equation:

```
cmat * sol = rhs
```

as follows:

```
>> sol = cmat ** rhs
>>> sol.neatprint()
A matrix of dimensions (m x n), where m, n, LineLen =  2 1 5
  1.00000E+00
  1.00000E+00
```

Happy computing!