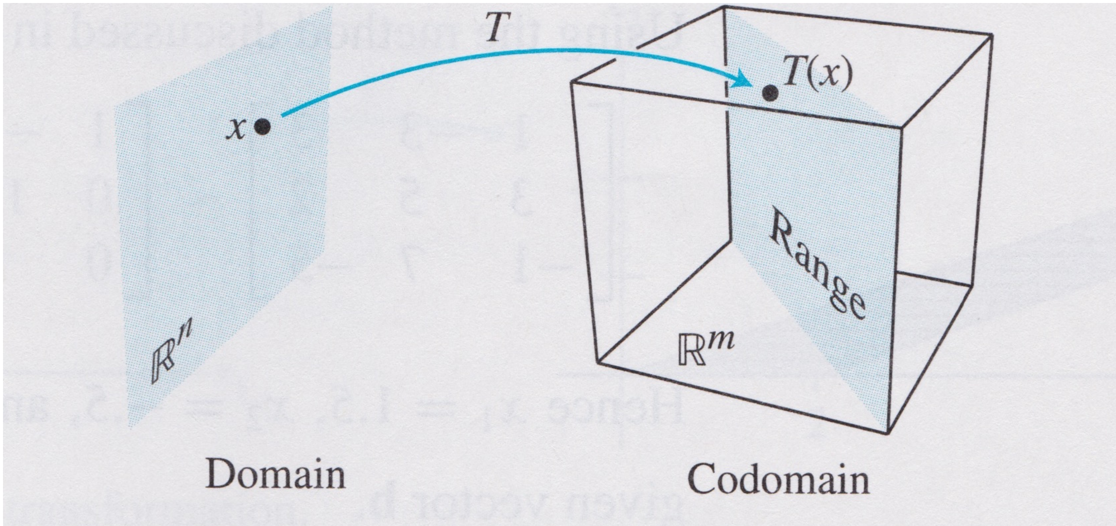
The Matrix of a Linear Transformation

In the last lecture we introduced the idea of a **linear transformation**:



We have seen that every matrix multiplication is a linear transformation from vectors to vectors.
But, are there any other possible linear transformations from vectors to vectors?
No.
In other words, the reverse statement is also true:
every linear transformation from vectors to vectors is a matrix multiplication.

We'll now prove this fact. We'll do it **constructively**, meaning we'll actually show how to find the matrix corresponding to any given linear transformation T.

Theorem. Let $T : \mathbb{R}^n \to \mathbb{R}^m$ be a linear transformation. There there is a unique matrix A such that:

$$T(\mathbf{x}) = A\mathbf{x} \quad \text{for all } \mathbf{x} \in \mathbb{R}^n.$$

In fact, A is the $m \times n$ matrix whose jth column is the vector $T(\mathbf{e_j})$, where $\mathbf{e_j}$ is the $j$the column of the identity matrix in $\mathbb{R}^n$:

$$A = [T(\mathbf{e_1}) \dots T(\mathbf{e_n})] \,.$$

A is called the *standard matrix* of T.
Proof. Write

$$\mathbf{x} = I\mathbf{x} = [\mathbf{e_1} \dots \mathbf{e_n}] \, \mathbf{x}$$

$$= x_1\mathbf{e_1} + \cdots + x_n\mathbf{e_n}.$$

Because T is linear, we have:

$$T(\mathbf{x}) = T(x_1\mathbf{e_1} + \cdots + x_n\mathbf{e_n})$$

$$= x_1T(\mathbf{e_1}) + \cdots + x_nT(\mathbf{e_n})$$

$$= [T(\mathbf{e_1}) \dots T(\mathbf{e_n})] \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = A\mathbf{x}.$$

The term *linear transformation* focuses on a **property** of the mapping, while the term *matrix multiplication* focuses on how such a mapping is **implemented.**

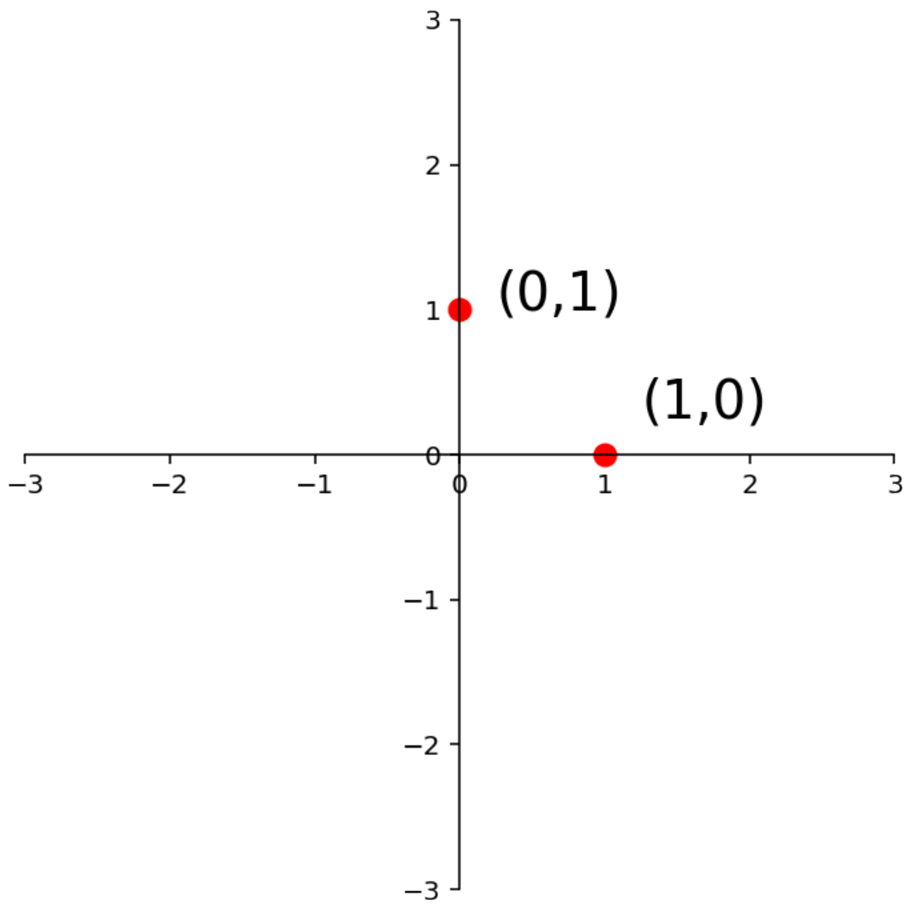
For example, we find the standard matrix of a linear tranformation of $\mathbb{R}^2 \to \mathbb{R}^2$ by asking what the transformation does to the columns of I.

Now, in $\mathbb{R}^2$, $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. So:

$$\mathbf{e_1} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } \mathbf{e_2} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

.

So to find the matrix of any given linear transformation, we only have to know what that transformation does to these two points:



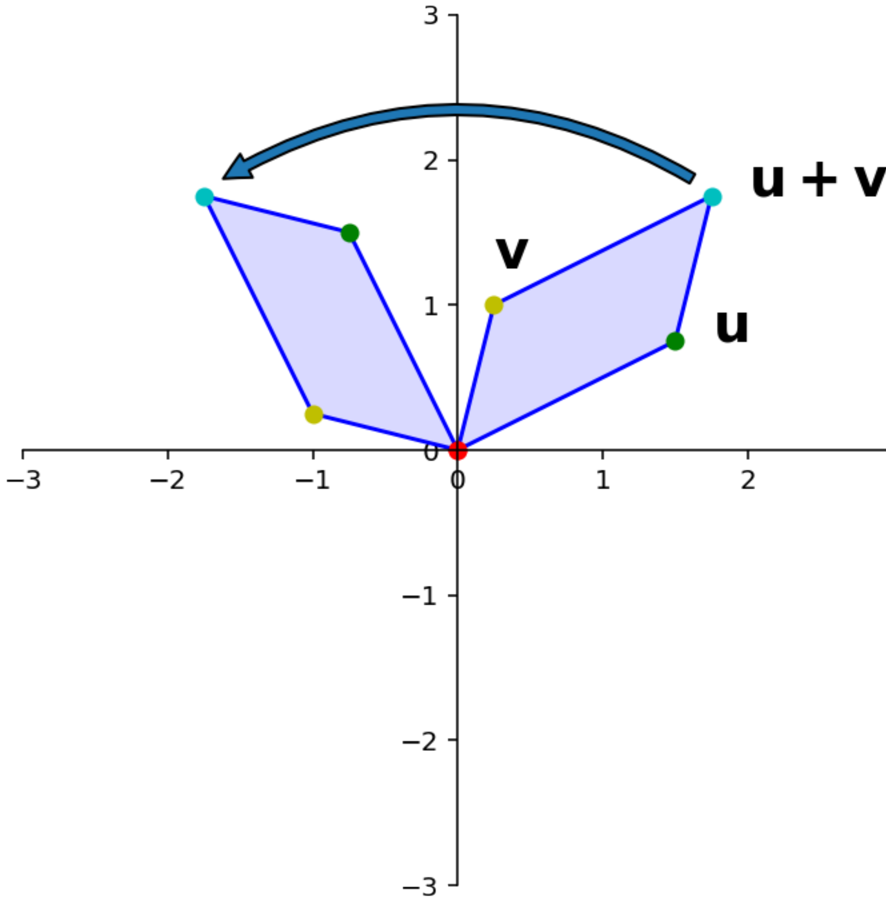
This is a **hugely** powerful tool.

Let's say we start from some given linear transformation; we can use this idea to find the matrix that implements that linear transformation.

For example, let's consider rotation about the origin as a kind of transformation.

Is it a **linear** transformation?

Recall that a for a transformation to be linear, it must be true that $T(\mathbf{u}+\mathbf{v}) = T(\mathbf{u}) + T(\mathbf{v})$.

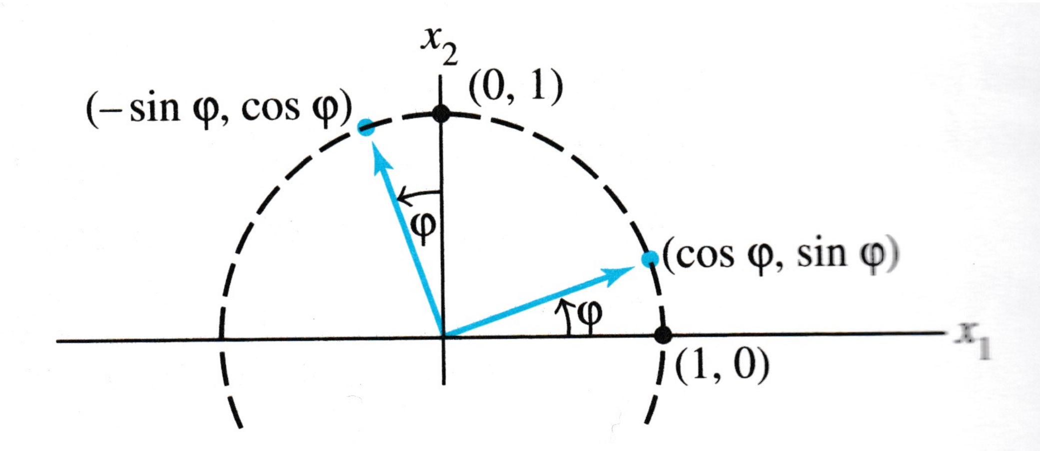


OK, so it is linear. let's see how to compute the linear transformation that is a rotation.

Example. Let $T : \mathbb{R}^2 \to \mathbb{R}^2$ be the transformation that rotates each point in $\mathbb{R}^2$ about the origin through an angle φ, with counterclockwise rotation for a positive angle. Find the standard matrix A of this transformation.

Solution. The columns of I are $\mathbf{e_1} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $\mathbf{e_2} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

Referring to the diagram below, we can see that $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ rotates into $\begin{bmatrix} \cos\varphi \\ \sin\varphi \end{bmatrix}$, and $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ rotates into $\begin{bmatrix} -\sin\varphi \\ \cos\varphi \end{bmatrix}$.

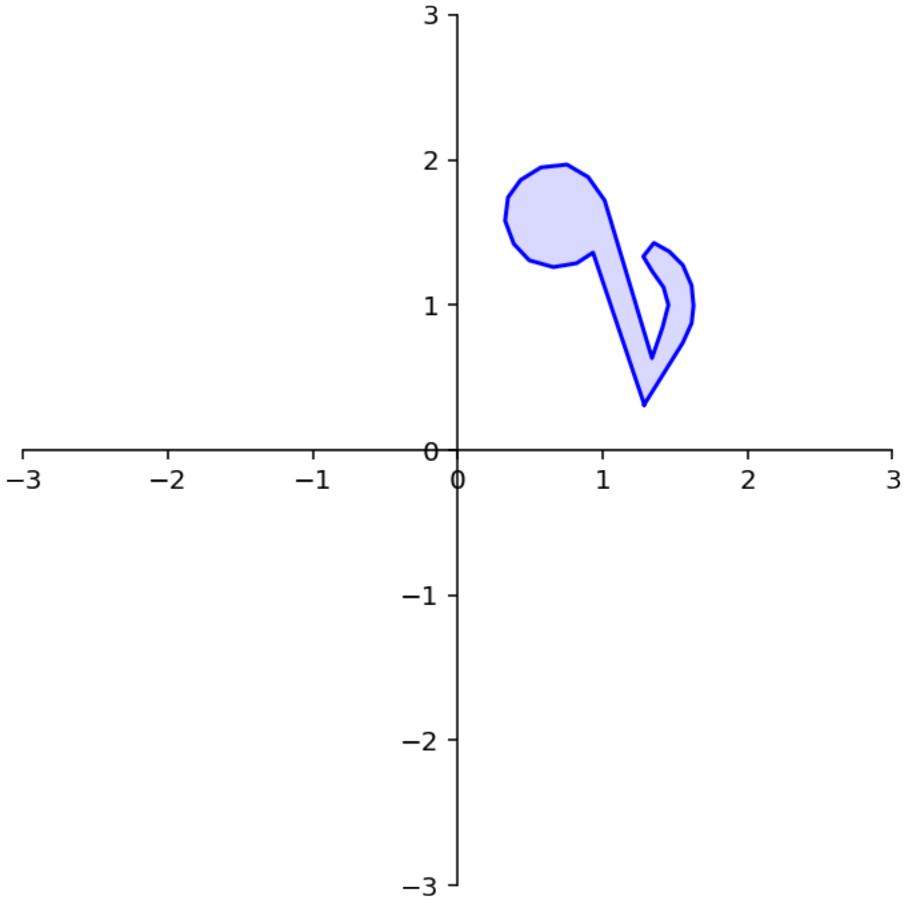


So by the Theorem above,

$$A = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}.$$

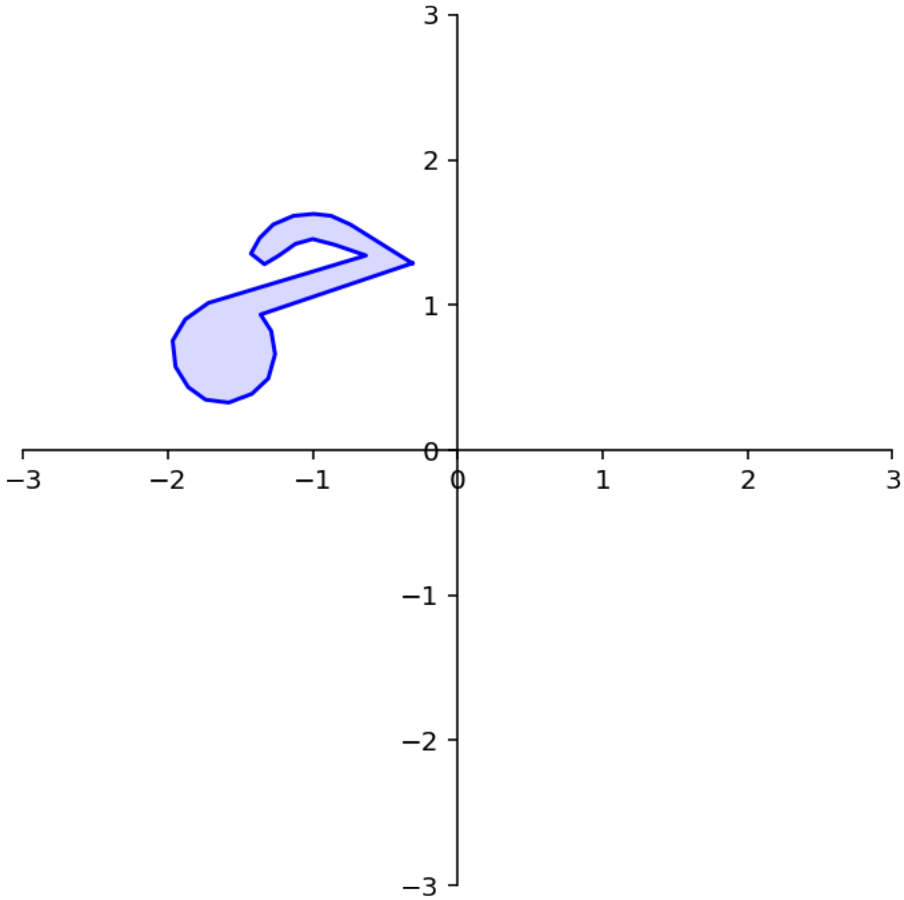
To demonstrate the use of a rotation matrix, let's rotate the following shape:

[7]:
```
dm.plotSetup()
note = dm.mnote()
dm.plotShape(note)
```



The variable `note` is a array of 26 vectors in $\mathbb{R}^2$ that define its shape.
In other words, it is a 2×26 matrix.
To rotate `note` we need to multiply each column of `note` by the rotation matrix A.
In Python matrix multiplication is performed using the `@` operator.
That is, if `A` and `B` are matrices,
`A @ B`
will multiply `A` by every column of `B`, and the resulting vectors will be formed into a matrix.

[8]:
```
dm.plotSetup()
angle = 90
phi = (angle/180) * np.pi
A = np.array(
    [[np.cos(phi), -np.sin(phi)],
     [np.sin(phi), np.cos(phi)]])
rnote = A @ note
dm.plotShape(rnote)
```



Geometric Linear Transformations of $\mathbb{R}^2$

Let's use our understanding of how to constuct linear transformations to look at some specific linear transformations of $\mathbb{R}^2$ to $\mathbb{R}^2$.

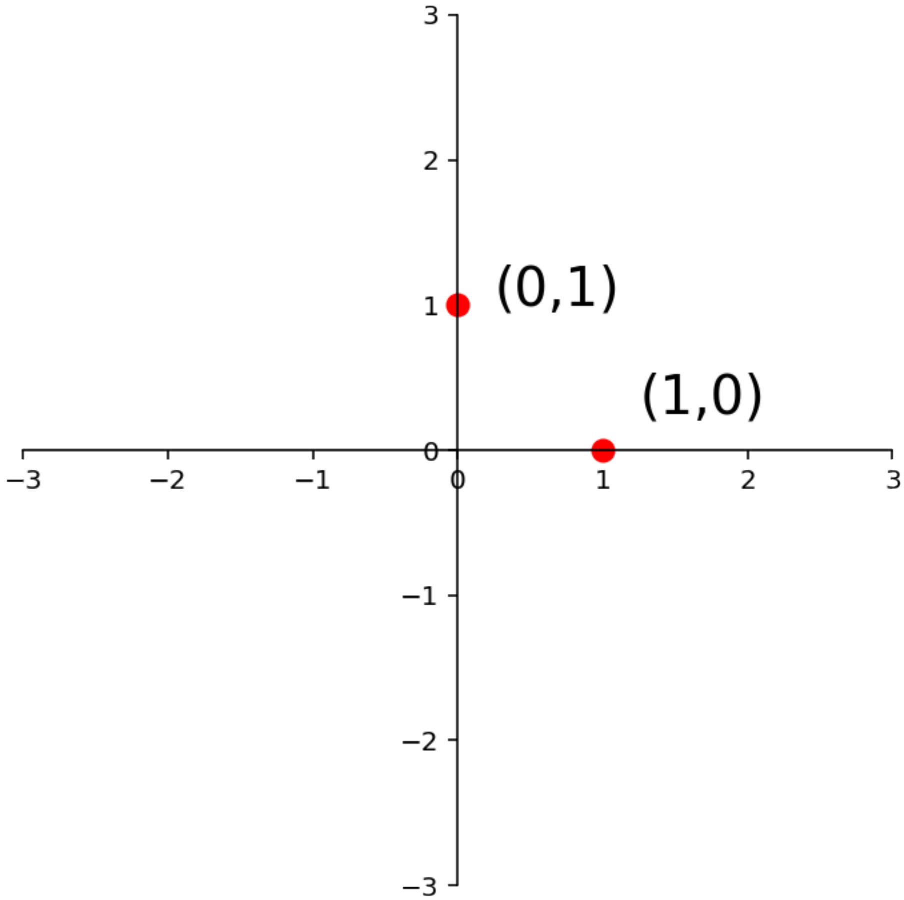
First, let's recall the linear transformation

$$T(\mathbf{x}) = r\mathbf{x}.$$

With $r > 1$, this is a dilation. It moves every vector further from the origin.

Let's say the dilation is by a factor of 2.5.

To construct the matrix A that implements this transformation, we ask: where do $\mathbf{e_1}$ and $\mathbf{e_2}$ go?



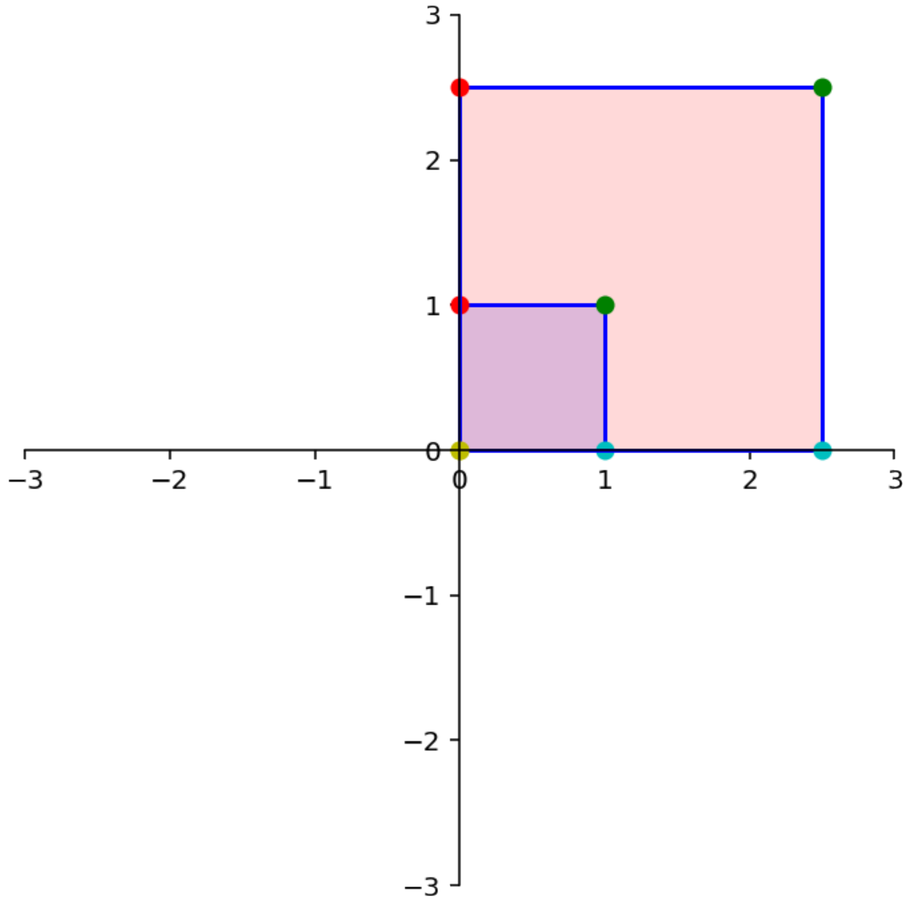
Under the action of A, $\mathbf{e_1}$ goes to $\begin{bmatrix} 2.5 \\ 0 \end{bmatrix}$ and $\mathbf{e_2}$ goes to $\begin{bmatrix} 0 \\ 2.5 \end{bmatrix}$.

So the matrix A must be $\begin{bmatrix} 2.5 & 0 \\ 0 & 2.5 \end{bmatrix}$.

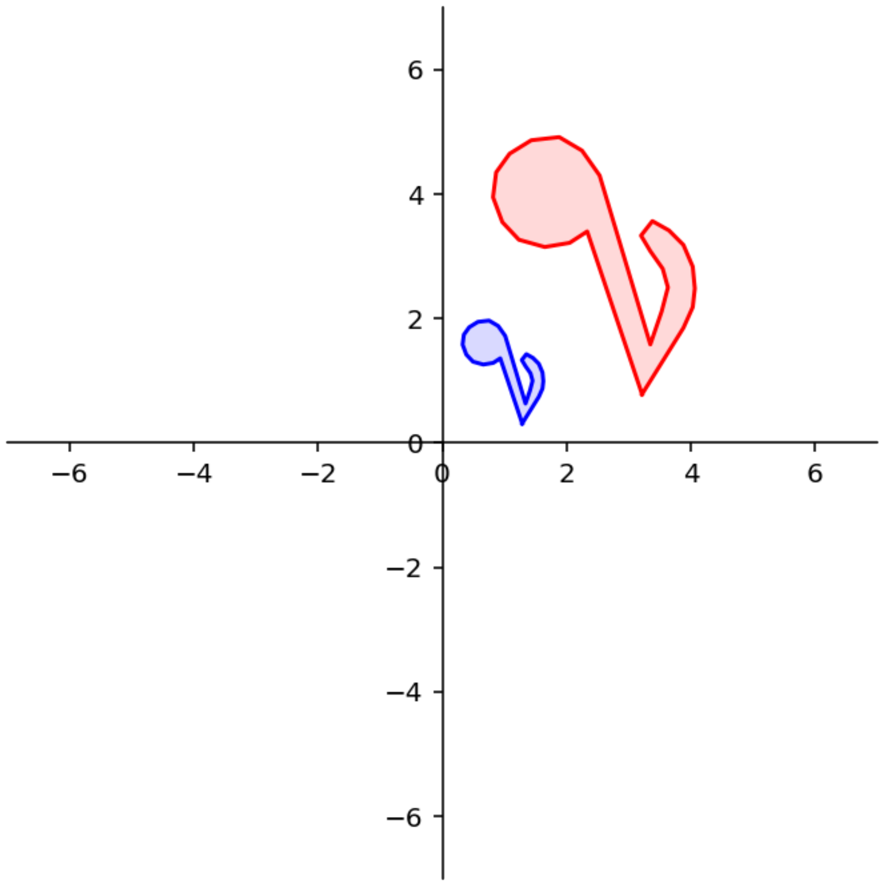
Let's test this out:

```
[10]: square = np.array(
          [[0,1,1,0],
           [1,1,0,0]])
      A = np.array(
          [[2.5, 0],
           [0, 2.5]])
      print('A = \n',A)
      dm.plotSetup()
      dm.plotSquare(square)
      dm.plotSquare(A @ square,'r')
```

```
A =
 [[2.5 0. ]
 [0.  2.5]]
```

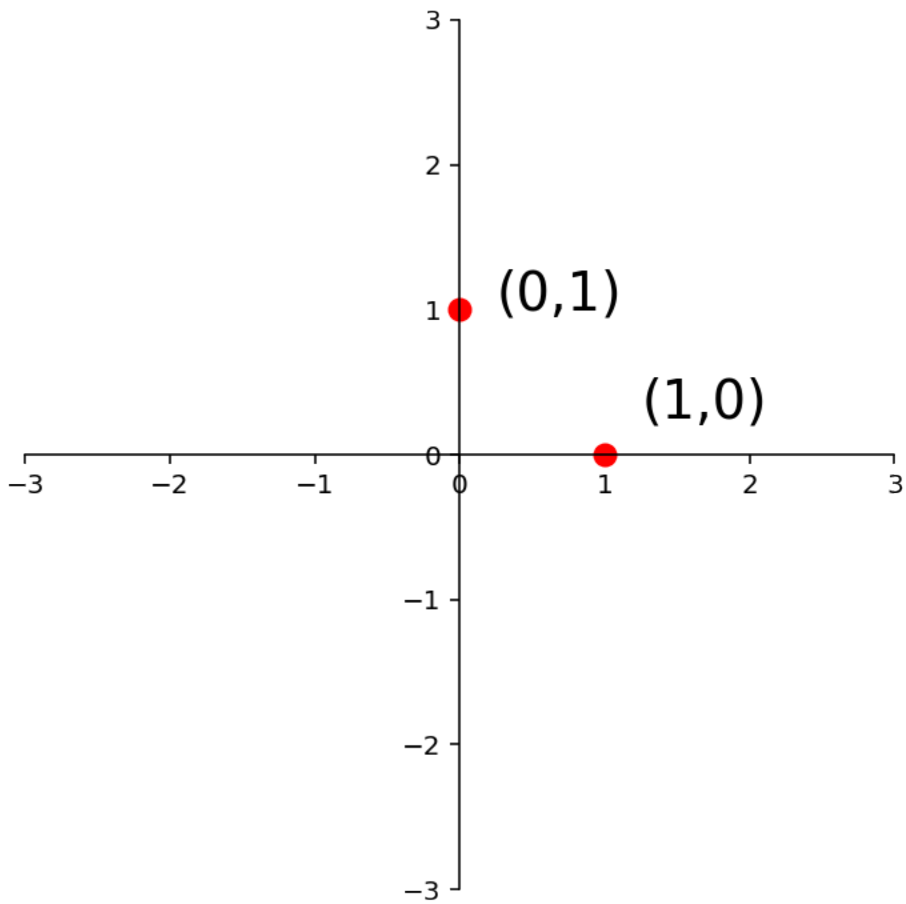



```
[11]: dm.plotSetup(-7,7,-7, 7)
      dm.plotShape(note)
      dm.plotShape(A @ note,'r')
```



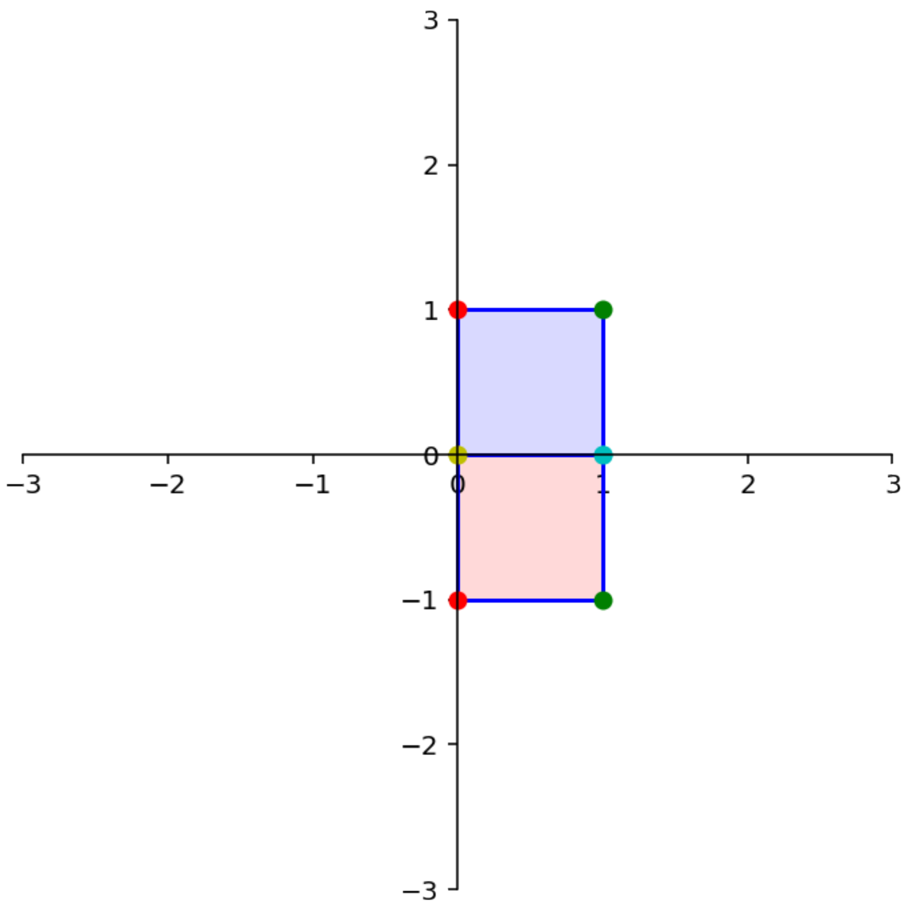
Question Time! Q8.1

OK, now let's reflect through the x_1 axis. Where do $\mathbf{e_1}$ and $\mathbf{e_2}$ go?

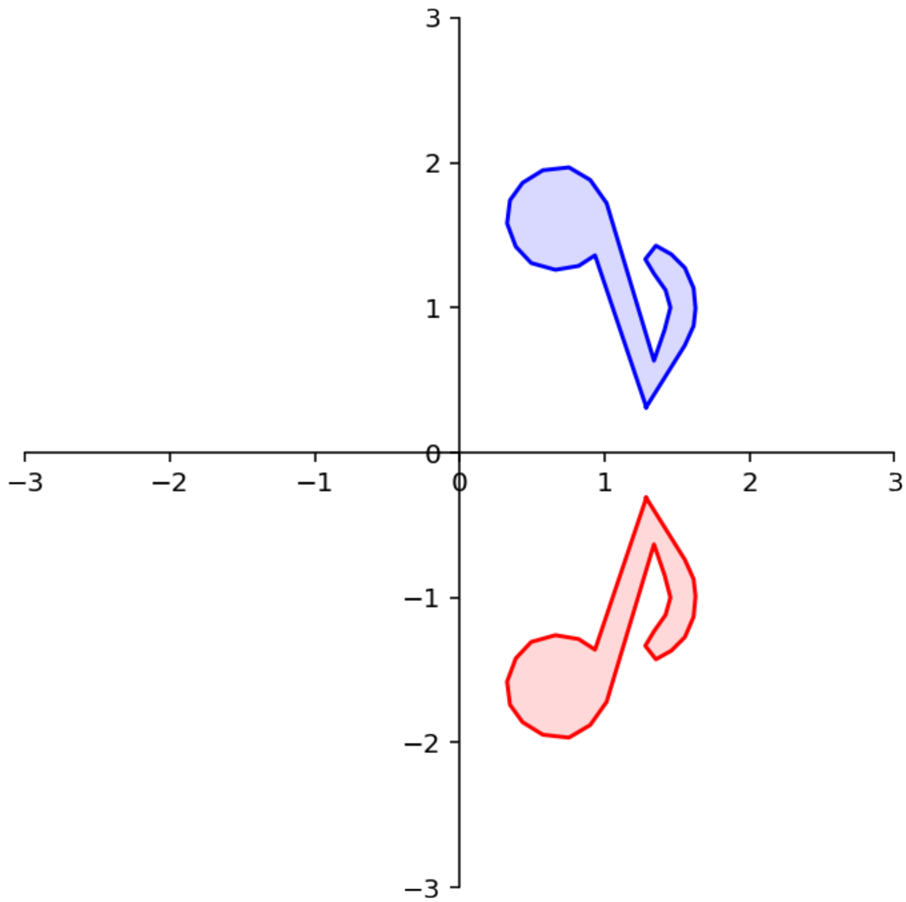



```
[13]: A = np.array(
          [[1, 0],
           [0,-1]])
      dm.plotSetup()
      dm.plotSquare(square)
      dm.plotSquare(A @ square,'r')
      Latex(r'Reflection through the $x_1$ axis')
```

Reflection through the x_1 axis



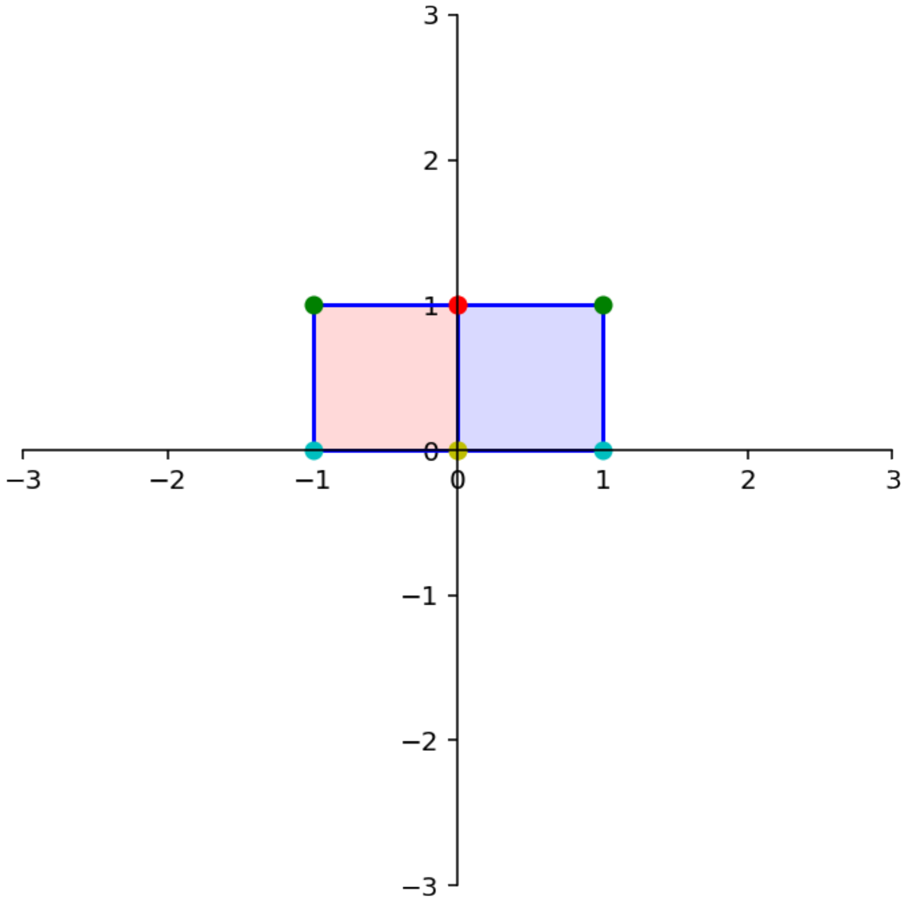

```
[14]: dm.plotSetup()
      dm.plotShape(note)
      dm.plotShape(A @ note,'r')
```



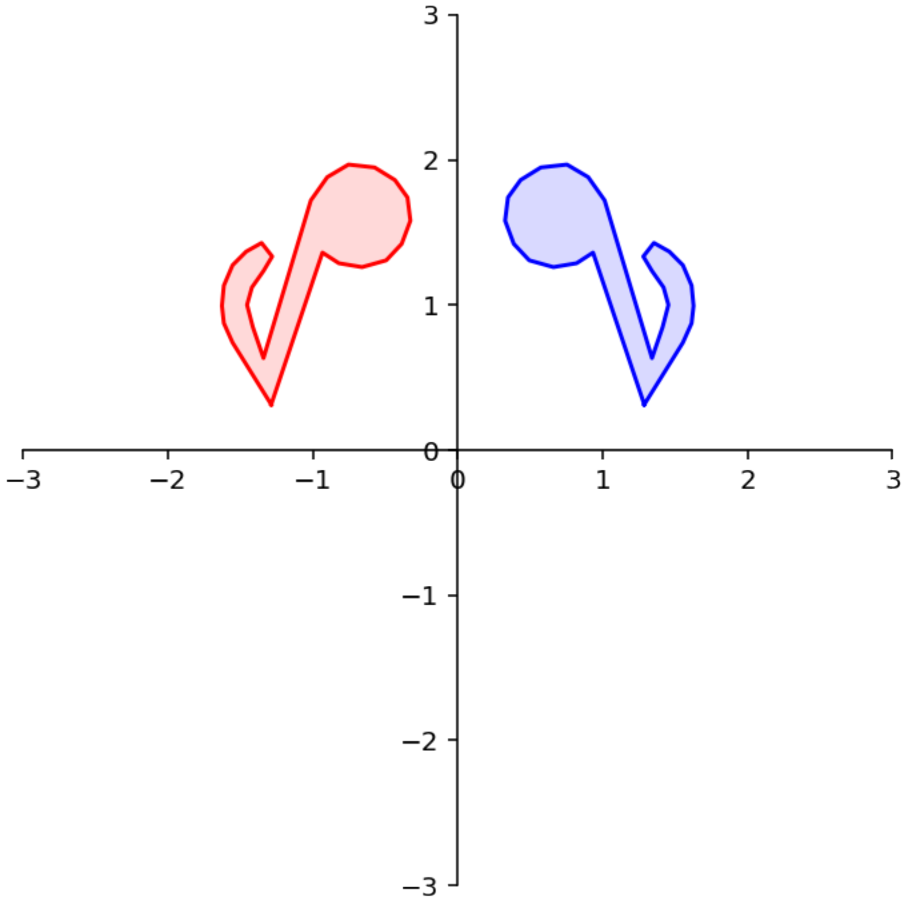
What about reflection through the x_2 axis?

```
[15]: A = np.array(
          [[-1,0],
           [0, 1]])
      dm.plotSetup()
      dm.plotSquare(square)
      dm.plotSquare(A @ square,'r')
      Latex(r'Reflection through the $x_2$ axis')
```

Reflection through the x_2 axis



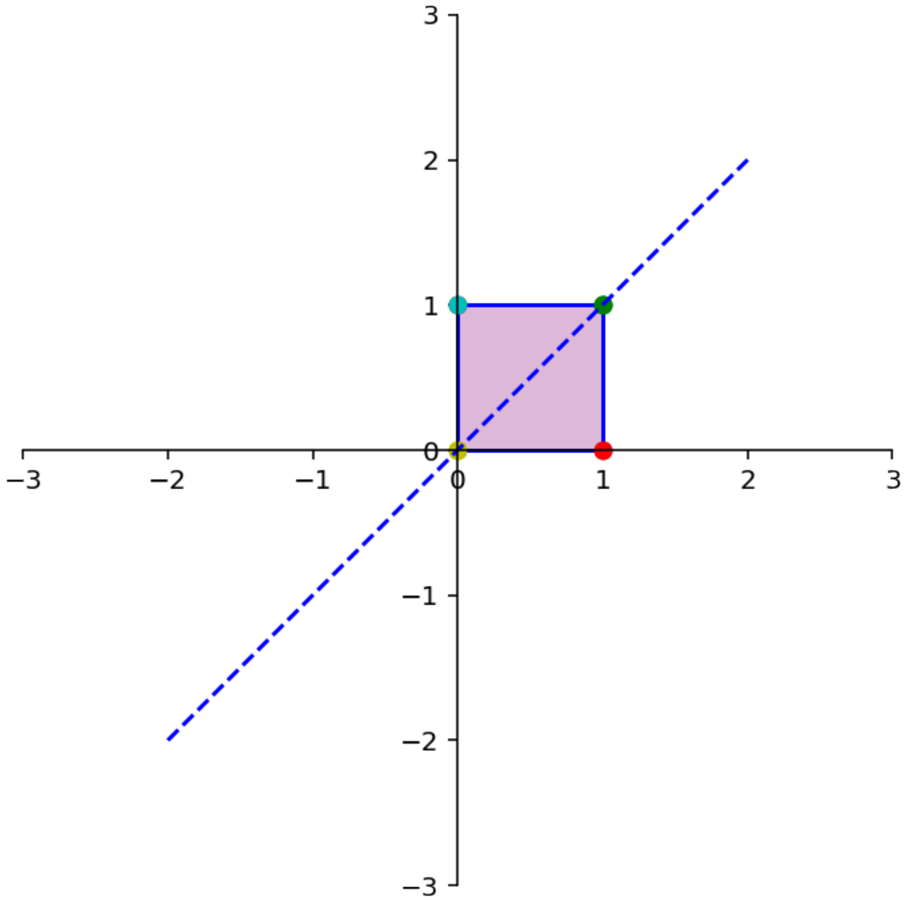

```
[16]: dm.plotSetup()
      dm.plotShape(note)
      dm.plotShape(A @ note,'r')
```



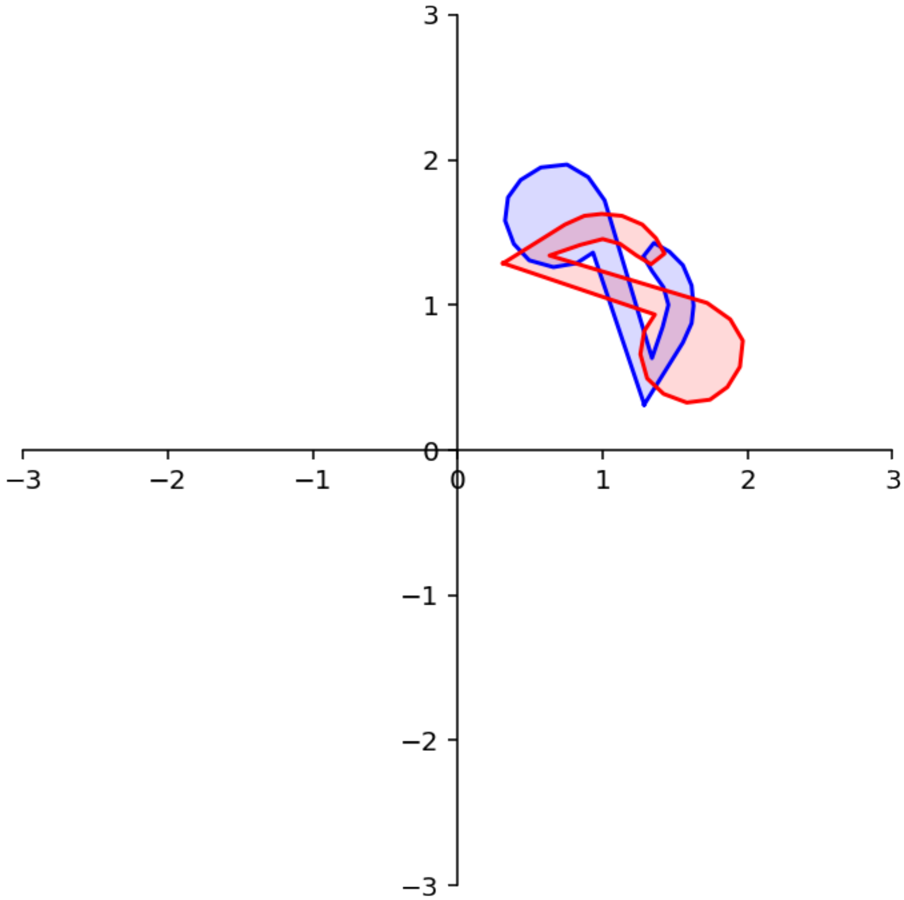
What about reflection through the line $x_1 = x_2$?

```
[17]: A = np.array(
          [[0,1],
           [1,0]])
      dm.plotSetup()
      dm.plotSquare(square)
      dm.plotSquare(A @ square,'r')
      plt.plot([-2,2],[-2,2],'b--')
      Latex(r'Reflection through the line $x_1 = x_2$')
```

Reflection through the line $x_1 = x_2$



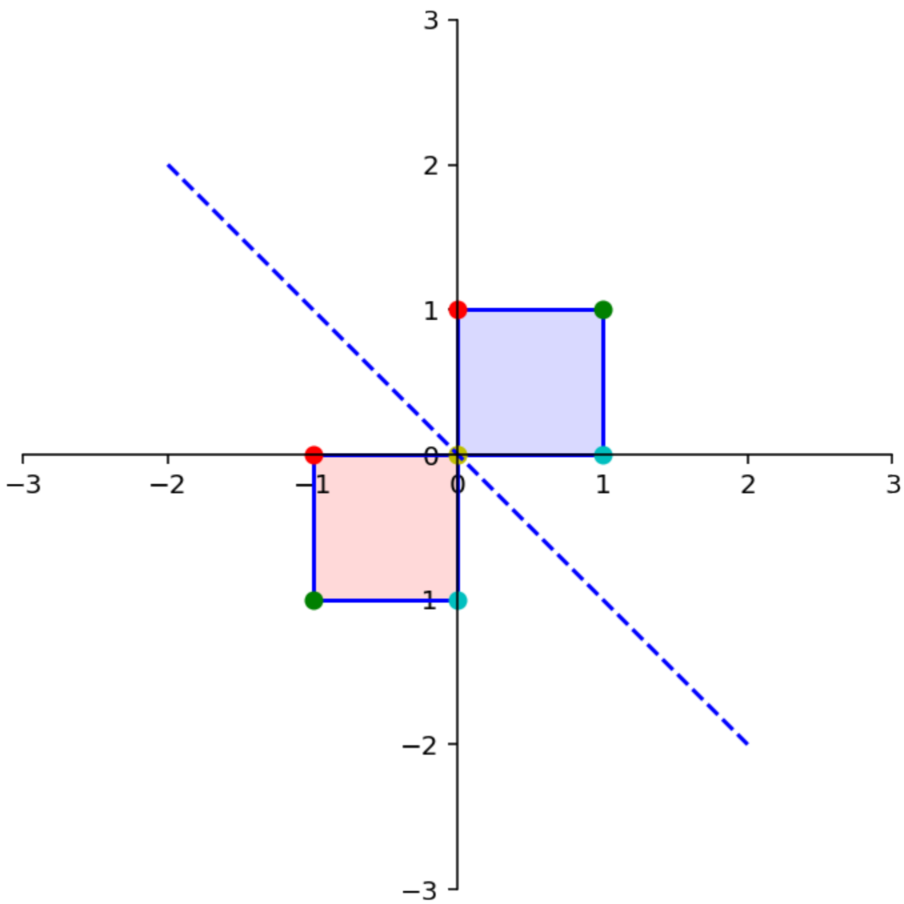

```
[18]: dm.plotSetup()
      dm.plotShape(note)
      dm.plotShape(A @ note,'r')
```



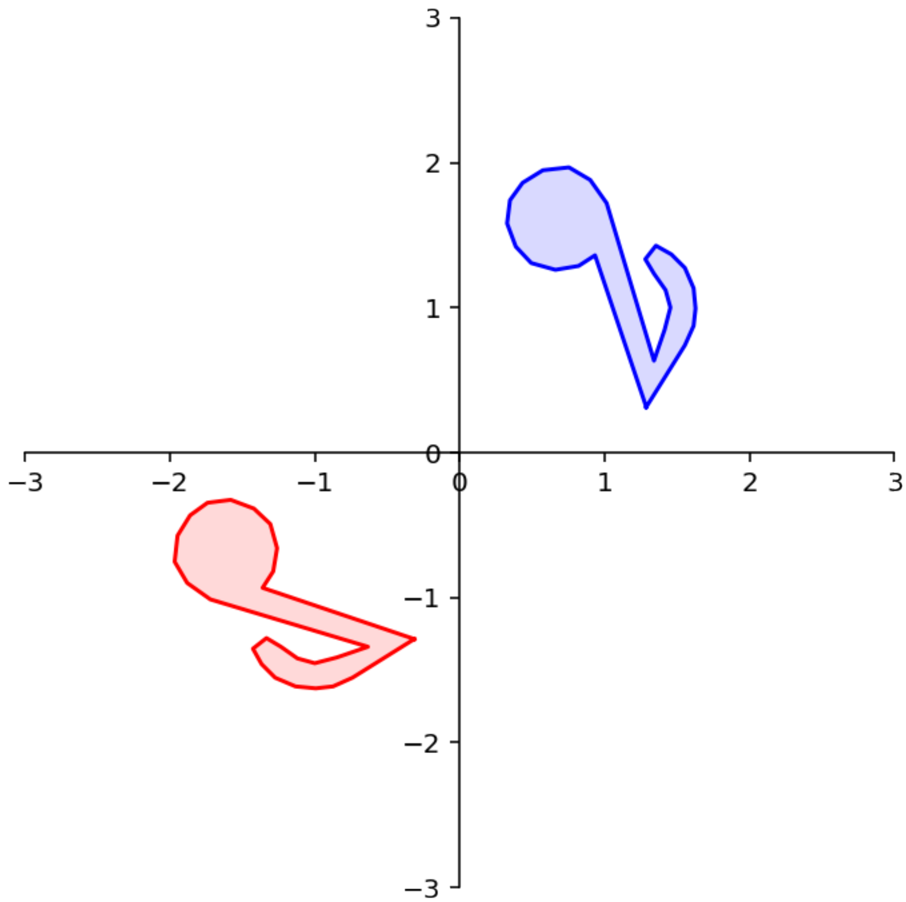
What about reflection through the line $x_1 = -x_2$?

```
[19]: A = np.array(
          [[ 0,-1],
           [-1, 0]])
      dm.plotSetup()
      dm.plotSquare(square)
      dm.plotSquare(A @ square,'r')
      plt.plot([-2,2],[2,-2],'b--')
      Latex(r'Reflection through the line $x_1 = -x_2$')
```

Reflection through the line $x_1 = -x_2$



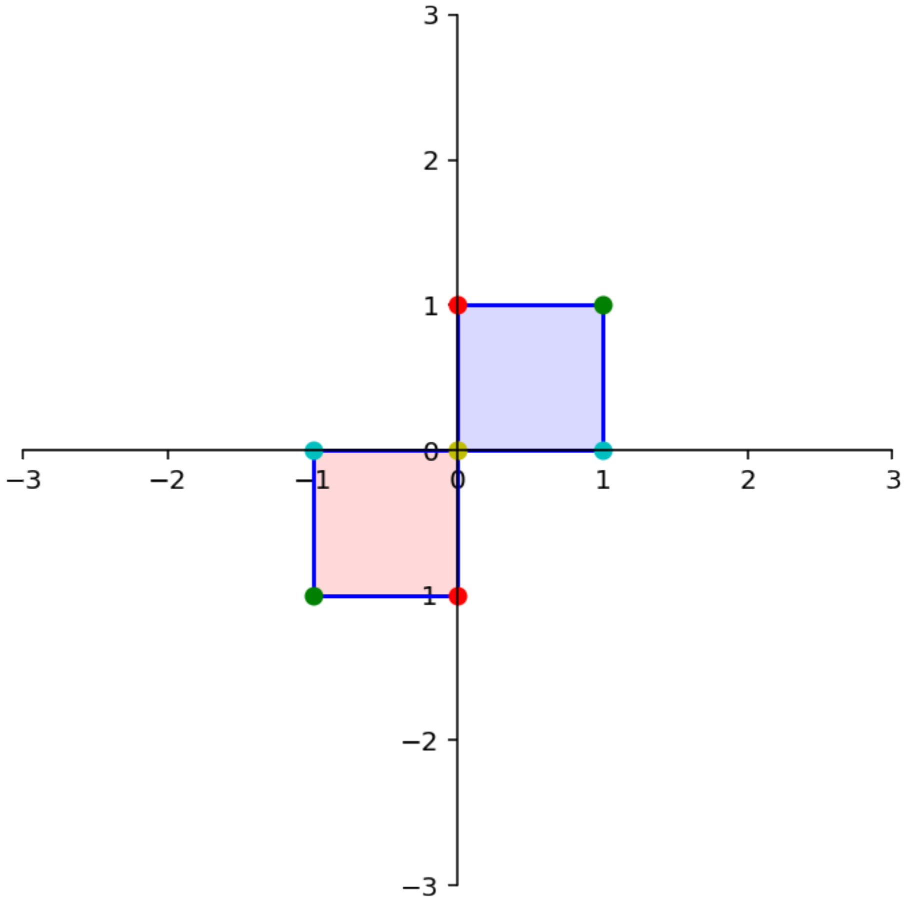

```
[20]: dm.plotSetup()
      dm.plotShape(note)
      dm.plotShape(A @ note,'r')
```



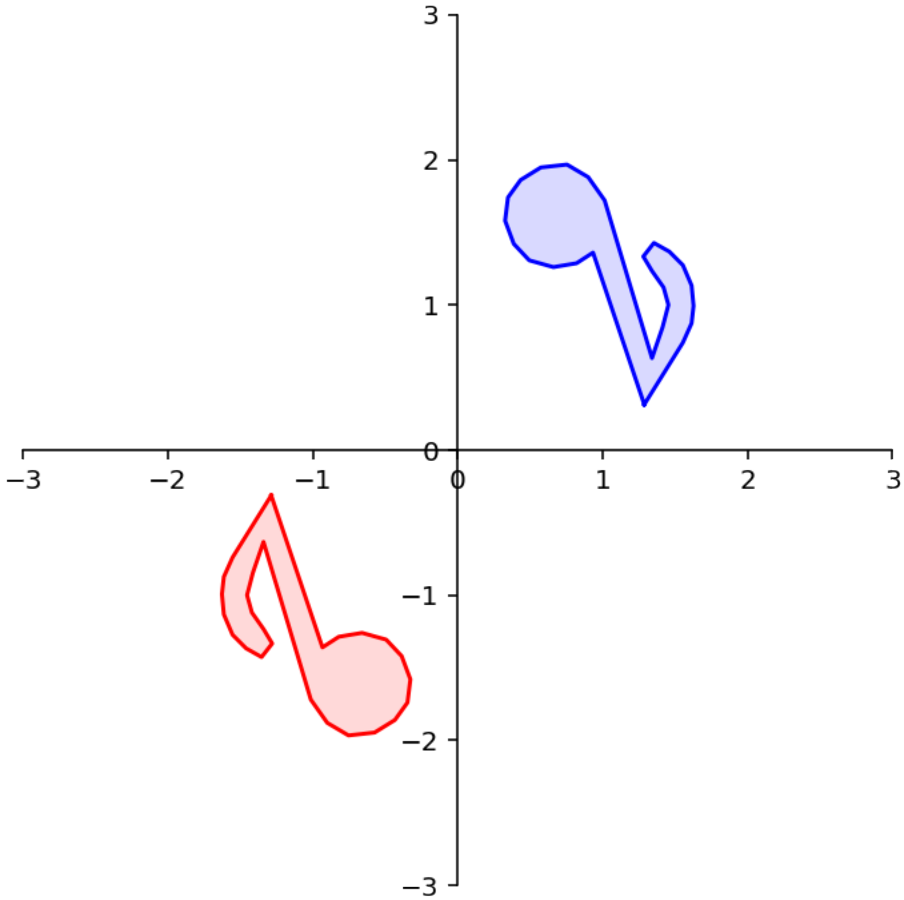
What about reflection through the origin?

```
[21]: A = np.array(
          [[-1, 0],
           [ 0,-1]])
      ax = dm.plotSetup()
      dm.plotSquare(square)
      dm.plotSquare(A @ square,'r')
      Latex(r'Reflection through the origin')
```

Reflection through the origin

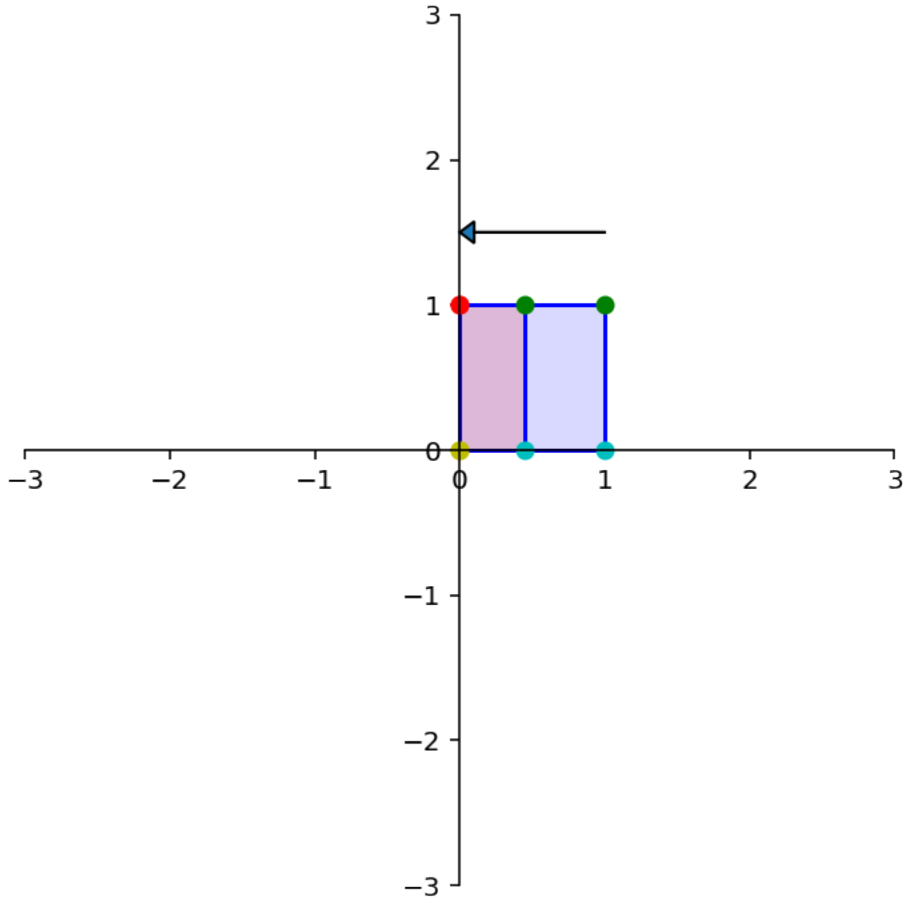



```
[22]: dm.plotSetup()
      dm.plotShape(note)
      dm.plotShape(A @ note,'r')
```

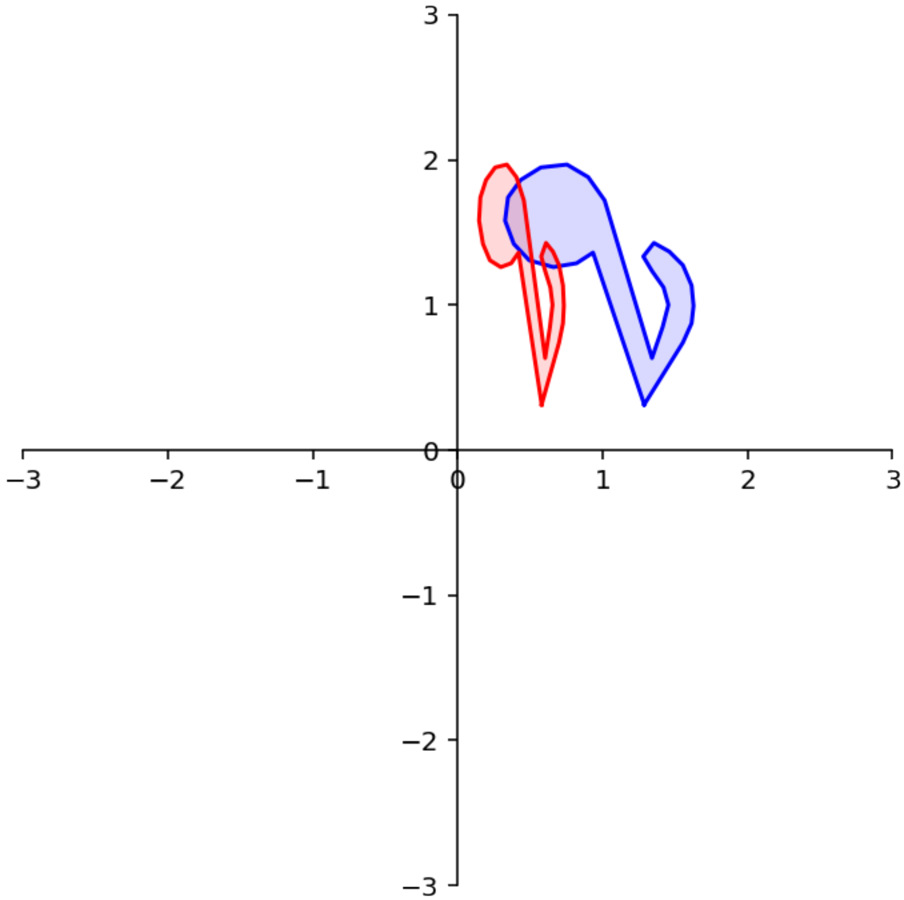



```
[23]: A = np.array(
          [[0.45, 0],
           [0,    1]])
      ax = dm.plotSetup()
      dm.plotSquare(square)
      dm.plotSquare(A @ square,'r')
      ax.arrow(1.0,1.5,-1.0,0,head_width=0.15, head_length=0.1, length_includes_head=True)
      Latex(r'Horizontal Contraction')
```

Horizontal Contraction

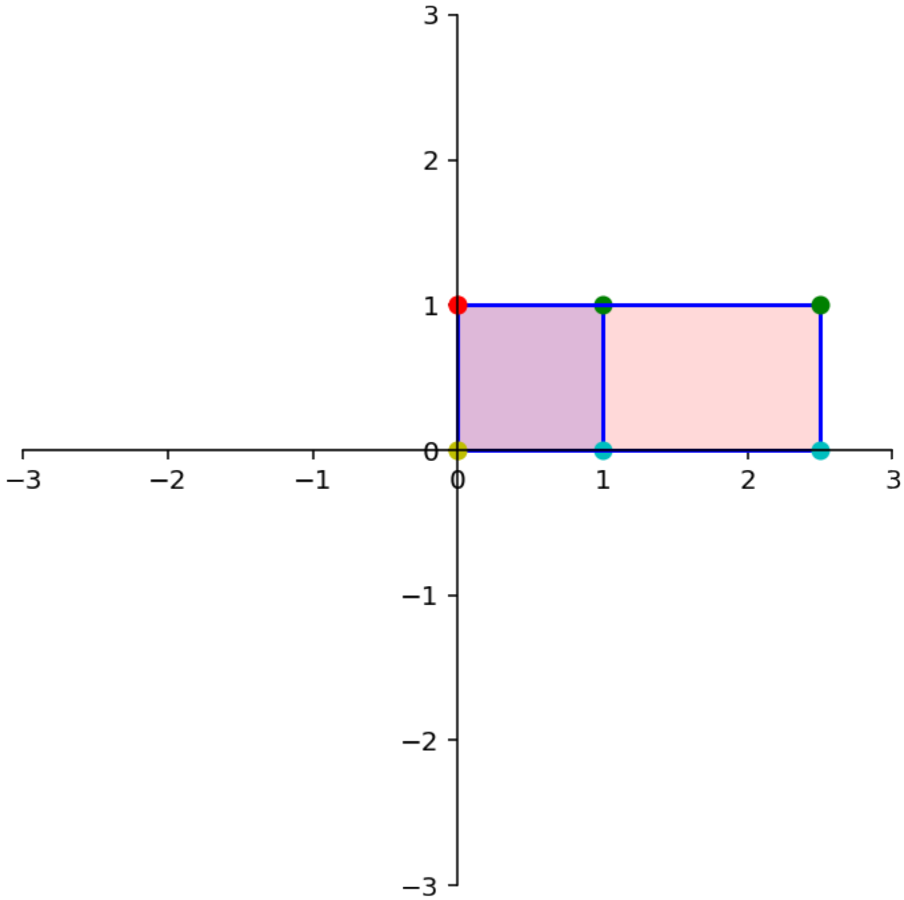



```
[24]: dm.plotSetup()
      dm.plotShape(note)
      dm.plotShape(A @ note,'r')
```



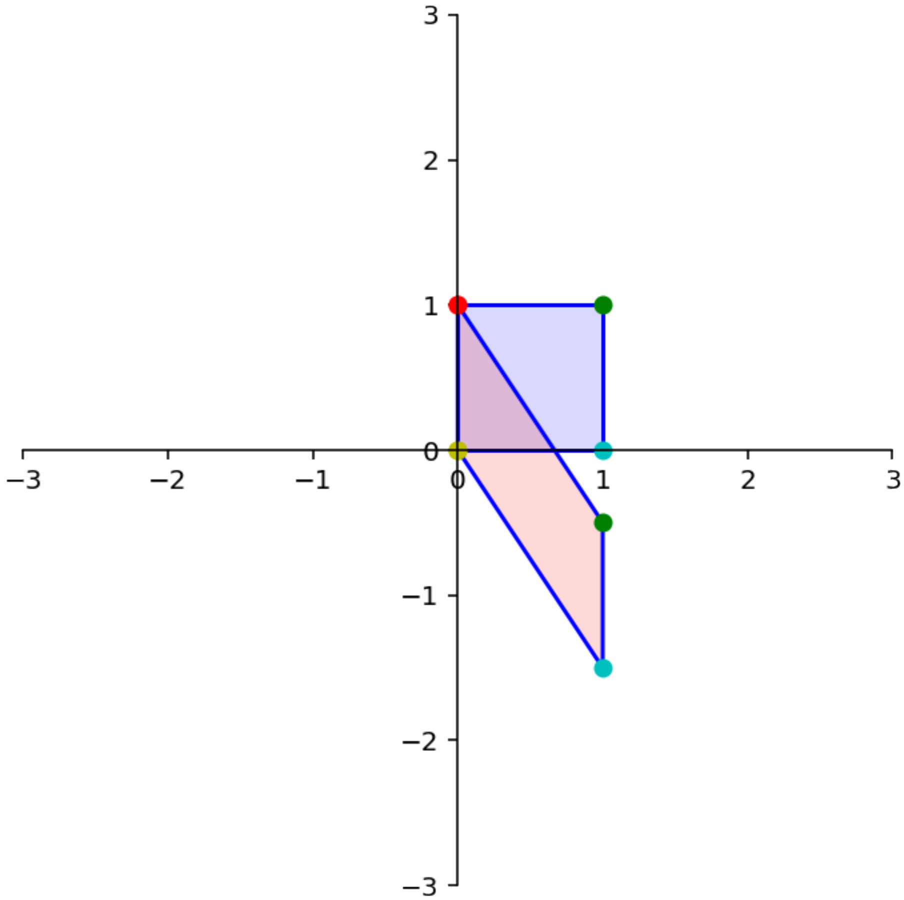

```
[25]: A = np.array(
          [[2.5,0],
           [0,  1]])
      dm.plotSetup()
      dm.plotSquare(square)
      dm.plotSquare(A @ square,'r')
      Latex(r'Horizontal Expansion')
```

Horizontal Expansion

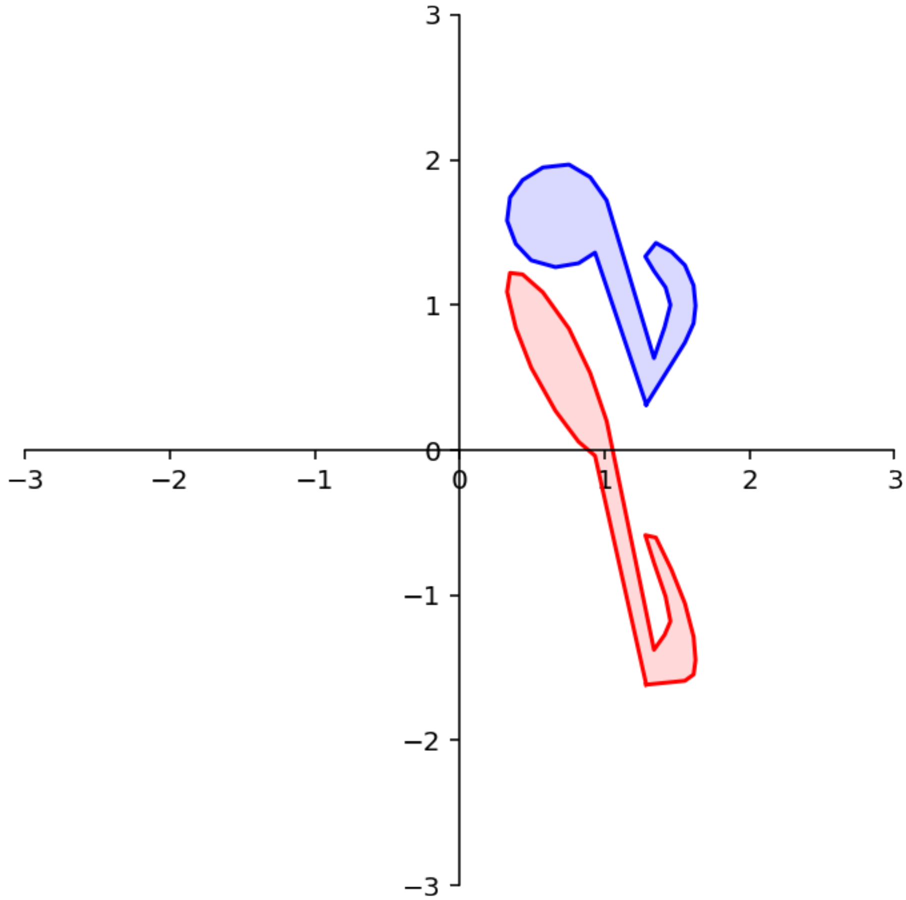



```
[26]: A = np.array(
          [[   1, 0],
           [-1.5, 1]])
      dm.plotSetup()
      dm.plotSquare(square)
      dm.plotSquare(A @ square,'r')
      Latex(r'Vertical Shear')
```

Vertical Shear




```
[27]: dm.plotSetup()
      dm.plotShape(note)
      dm.plotShape(A @ note,'r')
```



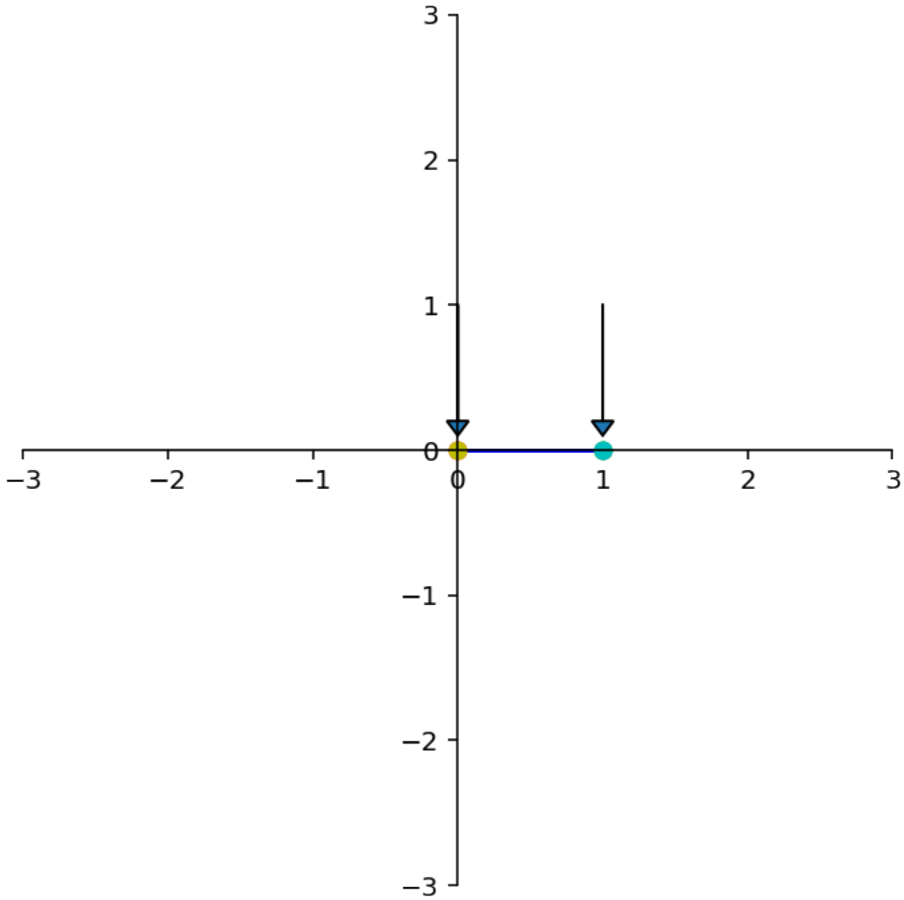
Question 8.2

Now let’s look at a particular kind of transformation called a **projection**.

Imagine we took any given point and ‘dropped’ it onto the x_1-axis.

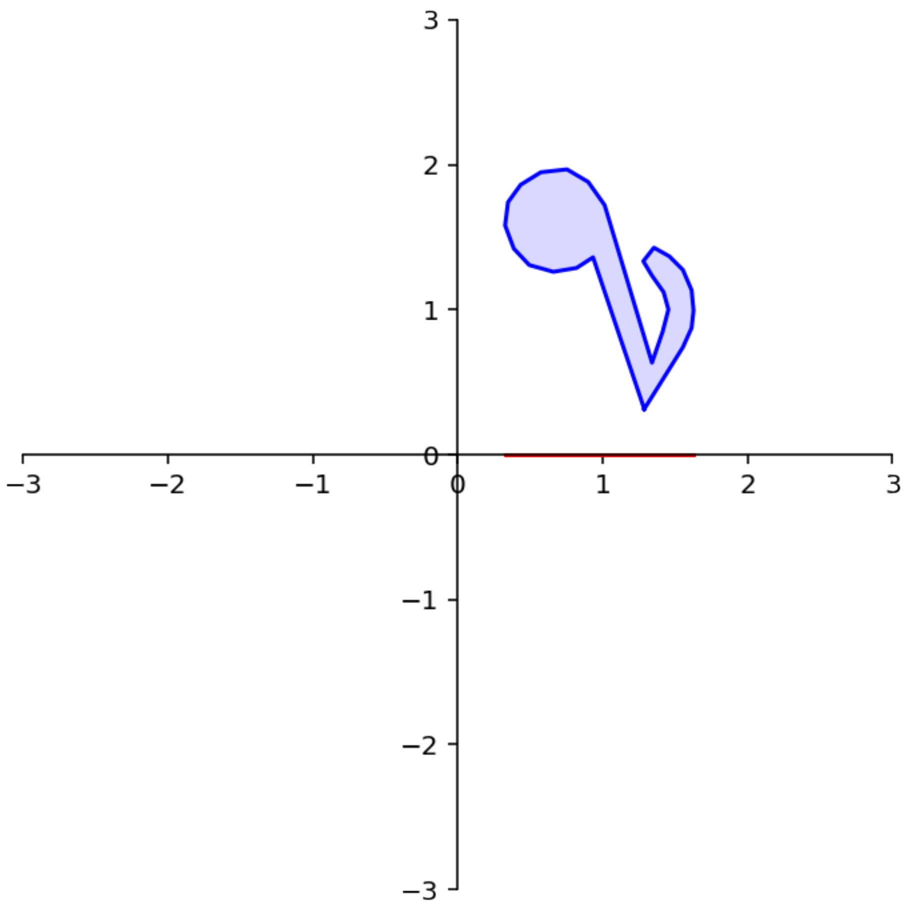
```
[28]: A = np.array(
          [[1,0],
           [0,0]])
      ax = dm.plotSetup()
      # dm.plotSquare(square)
      dm.plotSquare(A @ square,'r')
      ax.arrow(1.0,1.0,0,-0.9,head_width=0.15, head_length=0.1, length_includes_head=True)
      ax.arrow(0.0,1.0,0,-0.9,head_width=0.15, head_length=0.1, length_includes_head=True)
      Latex(r'Projection onto the $x_1$ axis')
```

Projection onto the x_1 axis



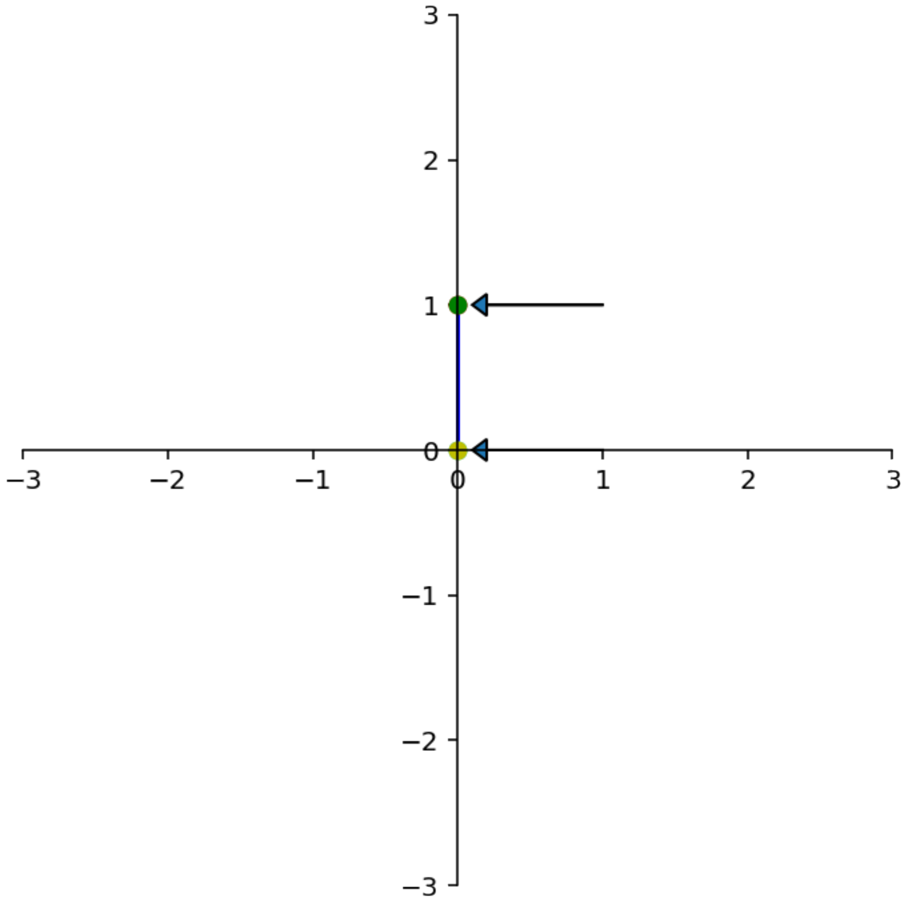
What happens to the **shape** of the point set?

```
[29]: dm.plotSetup()
      dm.plotShape(note)
      dm.plotShape(A @ note,'r')
```

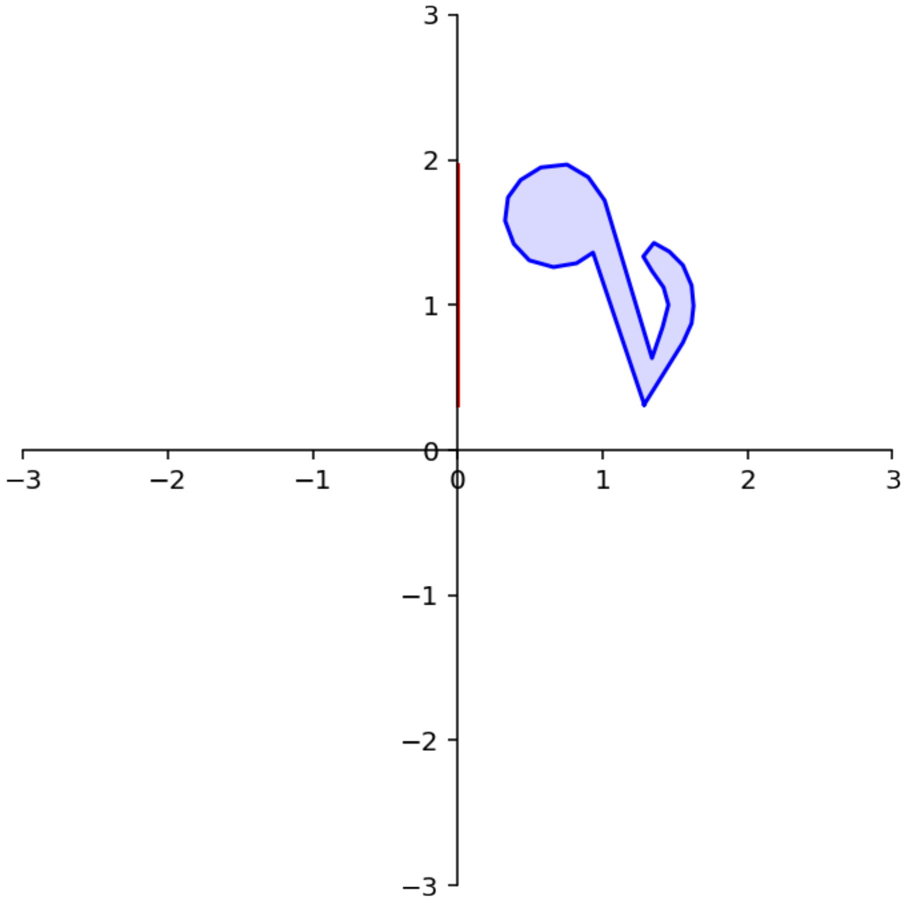



```
[30]: A = np.array(
          [[0,0],
           [0,1]])
      ax = dm.plotSetup()
      # dm.plotSquare(square)
      dm.plotSquare(A @ square)
      ax.arrow(1.0,1.0,-0.9,0,head_width=0.15, head_length=0.1, length_includes_head=True)
      ax.arrow(1.0,0.0,-0.9,0,head_width=0.15, head_length=0.1, length_includes_head=True)
      Latex(r'Projection onto the $x_2$ axis')
```

Projection onto the x_2 axis




```
[31]: dm.plotSetup()
      dm.plotShape(note)
      dm.plotShape(A @ note,'r')
```



Existence and Uniqueness

Notice that some of these transformations map multiple inputs to the same output, and some are incapable of generating certain outputs.

For example, the **projections** above can send multiple different points to the same point.

We need some terminology to understand these properties of linear transformations.

Definition. A mapping $T : \mathbb{R}^n \to \mathbb{R}^m$ is said to be **onto** $\mathbb{R}^m$ if each $\mathbf{b}$ in $\mathbb{R}^m$ is the image of *at least one* $\mathbf{x}$ in $\mathbb{R}^n$.

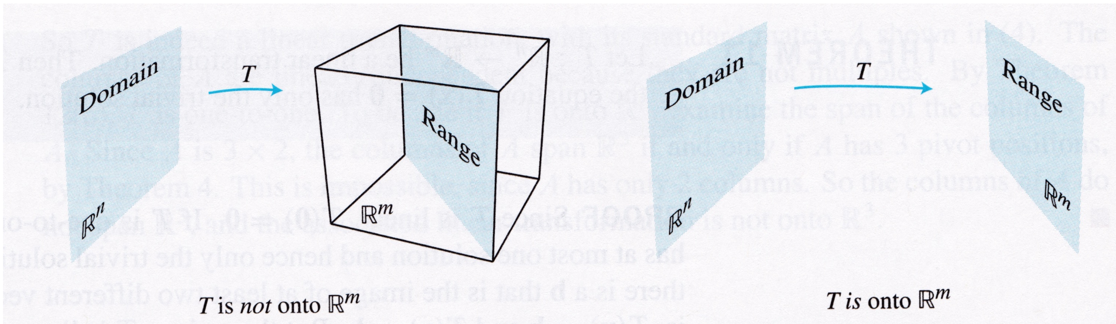
Informally, T is onto if every element of its codomain is in its range.

Another (important) way of thinking about this is that T is onto if there is a solution $\mathbf{x}$ of

$$T(\mathbf{x}) = \mathbf{b}$$

for all possible $\mathbf{b}$.

This is asking an **existence** question about a solution of the equation $T(\mathbf{x}) = \mathbf{b}$ for all $\mathbf{b}$.



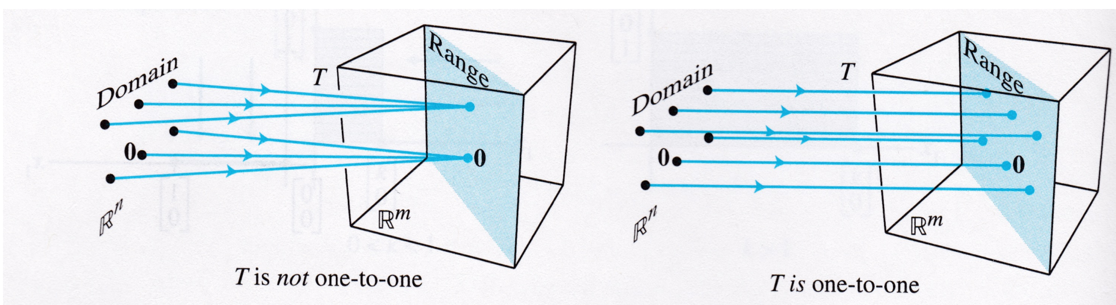
T is *not* onto $\mathbb{R}^m$ *T is* onto $\mathbb{R}^m$

Question Time! Q8.3

Definition. A mapping $T : \mathbb{R}^n \to \mathbb{R}^m$ is said to be **one-to-one** if each $\mathbf{b}$ in $\mathbb{R}^m$ is the image of *at most one* $\mathbf{x}$ in $\mathbb{R}^n$.

If T is one-to-one, then for each $\mathbf{b}$, the equation $T(\mathbf{x}) = \mathbf{b}$ has either a unique solution, or none at all.

This is asking an **existence** question about a solution of the equation $T(\mathbf{x}) = \mathbf{b}$ for all $\mathbf{b}$.



T is *not* one-to-one *T is* one-to-one

Let's examine the relationship between these ideas and some previous definitions.

If $A\mathbf{x} = \mathbf{b}$ is consistent for all $\mathbf{b}$, is $T(\mathbf{x}) = A\mathbf{x}$ onto? one-to-one?

$T(\mathbf{x})$ is onto. $T(\mathbf{x})$ may or may not be one-to-one. If the system has multiple solutions for some $\mathbf{b}$, $T(\mathbf{x})$ is not one-to-one.

If $A\mathbf{x} = \mathbf{b}$ is consistent and has a unique solution for all $\mathbf{b}$, is $T(\mathbf{x}) = A\mathbf{x}$ onto? one-to-one?

Yes to both.

If $A\mathbf{x} = \mathbf{b}$ is not consistent for all $\mathbf{b}$, is $T(\mathbf{x}) = A\mathbf{x}$ onto? one-to-one?

$T(\mathbf{x})$ is **not** onto. $T(\mathbf{x})$ may or may not be one-to-one.

If $T(\mathbf{x}) = A\mathbf{x}$ is onto, is $A\mathbf{x} = \mathbf{b}$ consistent for all $\mathbf{b}$? is the solution unique for all $\mathbf{b}$?

If $T(\mathbf{x}) = A\mathbf{x}$ is one-to-one, is $A\mathbf{x} = \mathbf{b}$ consistent for all $\mathbf{b}$? is the solution unique for all $\mathbf{b}$?