Zooming

The order in which a sequence of coordinate changes takes places is important. Here, for example, is what happens for each of these sequences:

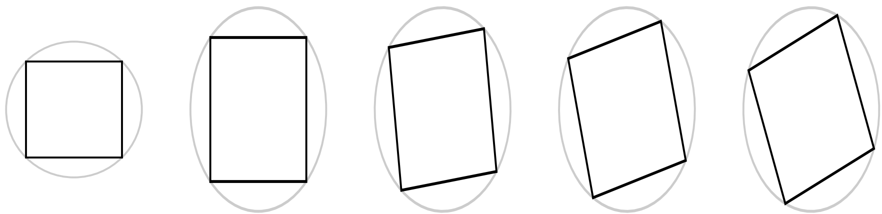
```
30 rotate
1 1.5 scale

newpath
0.5 square
stroke
```

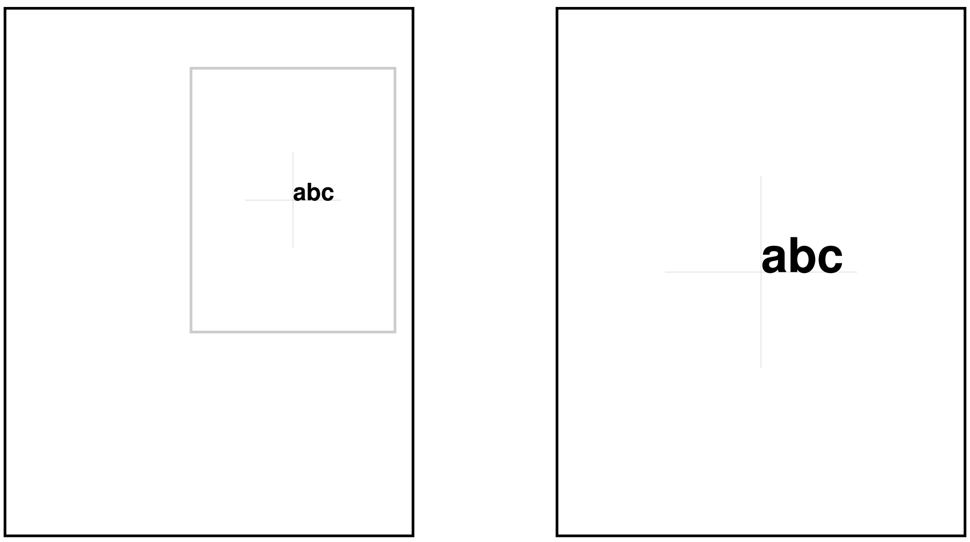
```
1 1.5 scale
30 rotate

newpath
0.5 square
stroke
```

The point to keep in mind is that after a coordinate change is made, all further coordinate changes take place with respect to that new system. Thus, in scaling a circle becomes an ellipse, and rotation takes place *around this ellipse* instead of around a circle. We can see what happens by doing it in stages:



To see a useful application of this idea, let's construct a procedure **`zoom`** which has the effect of zooming in at a point by a given scale. In other words, the original page and the zoomed one look like this



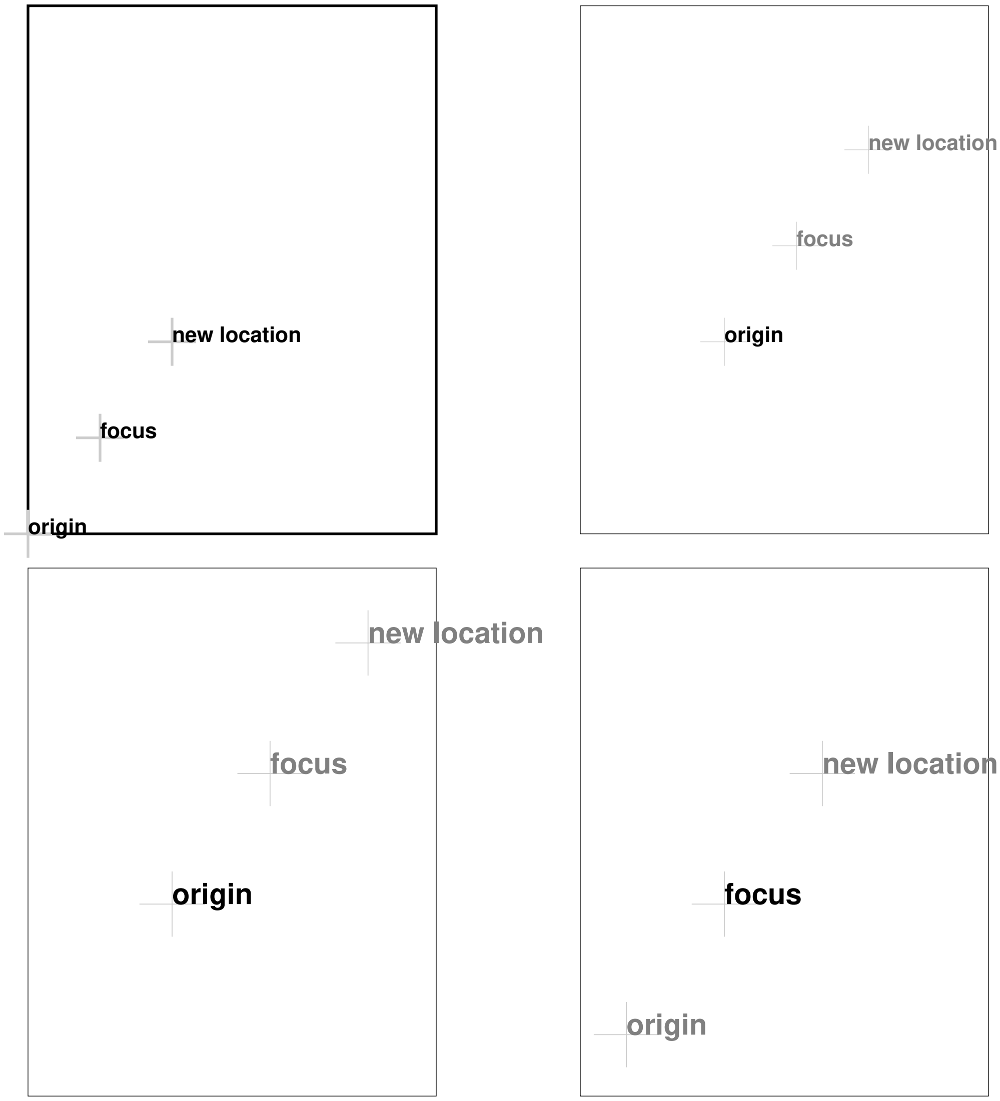
if we zoom into the centre of the square with a zoom factor of two.

How can we do this? We shall see more generally how to construct a sequence of commands that have this effect: the point that is located at (x, y) in the following drawing is drawn at the new location (c_x, c_y) instead, and the figure is in addition scaled by a factor of s. The sequence of commands we need here is

```
cx cy translate
s dup scale
x neg y neg translate
```

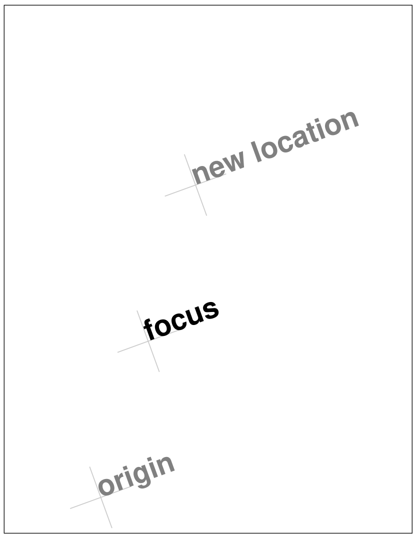
followed by the original drawing commands.

We can see how this works by tracking the effect line by line. In this picture, the point marked **focus** is at (x, y). It will be relocated at the **new location** (c_x, c_y), scaled by a certain factor s. The origin is also indicated. The four figures show the effect after none, one, two, and all three of the above lines are executed.



If you want to rotate the figure with the focus as the pivot, then add the correct line as here:

```
cx cy translate
20 rotate
s dup scale
x neg y neg translate
```



A useful procedure

```
% [cx cy] [x y] s:  the place that is now (x, y) is located at [cx cy]
% and lengths scaled by s

/zoom {
8 dict begin
/s exch def
aload pop
/y exch def
/x exch def
aload pop
translate
s dup scale
x neg y neg translate
currentlinewidth s div setlinewidth
end
} def
```

Finding the centre

Very often you want to make the new location (c_x, c_y) equal to the centre of the page. How do you find it? We assume, as usual, a page $8.5'' \times 11''$. The problem is to find the coordinates in the current coordinate system of the centre of the page. This means we have to use the affine transformation from page coordinates to user coordinates, as we did when drawing complete lines. In addition we have to apply various transformations to certain points. The command **`transform`** is applied when a matrix (in PostScript's sense) and two coordinates are on the stack. It leaves on the stack the effect of applying the matrix to the point, another pair of coordinates. Similarly **`itransform`** applies the **inverse transform**. So here is a procedure which returns as an array of two elements the centre of the page in user coordinates:

```
/page-centre {
[
  4.25 72 mul 5.5 72 mul
  matrix defaultmatrix transform
  matrix currentmatrix itransform
]
} def
```

In effect, we are transforming the centre first from page to physical coordinates and then back to user coordinates.

Playing around

Try this:

```
%!

(zoom.inc) run
% defines zoom

/draw {

gsave
1 0 0 setrgbcolor
newpath
x y moveto
-100 0 rlineto
 200 0 rlineto
x y moveto
 0 -100 rlineto
 0 200 rlineto
stroke
grestore

x y moveto
(Euclid) show

} def

/Helvetica-Bold findfont
25 scalefont
setfont

/s 1 def
/x 100 def
/y 100 def

{ gsave
  [x y] [x y] s zoom
  draw
  grestore
  /s s 1.1 mul def
  showpage
} loop
```

How would you get the text to rotate around the focus as the loop proceeds?