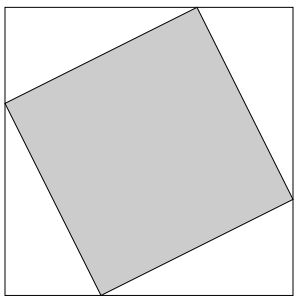
Mathematics 308—December 31, 1995

A sample PostScript program

Here is the figure:



Here is the program which drew it:

```
%!

0 setlinewidth

/a 0.50 def
/b 1.00 def
/c a a mul b b mul add sqrt def
/sum a b add def

72 72 scale

newpath
0 0 moveto
sum 0 lineto
sum sum lineto
0 sum lineto
0 0 lineto
closepath
stroke

a 0 translate
a b atan rotate

0.8 setgray
newpath
0 0 moveto
c 0 lineto
c c lineto
0 c lineto
0 0 lineto
closepath
fill
```

```
0 setgray
newpath
0 0 moveto
c 0 lineto
c c lineto
0 c lineto
0 0 lineto
closepath
stroke

closepath

showpage
```

Beginning the `PostScript` dictionary

As the course proceeds, we shall build up a dictionary of `PostScript` commands. Here are the ones used above, and a few more.

`%!`	must appear as the first two characters of every `PostScript` program
`newpath`	starts a path
`x y moveto`	moves the current point to (x, y)
`x y lineto`	draws line to (x, y)
`stroke`	draws the path
`fill`	fills the path (which must be closed)
`closepath`	closes up the path
`g setgray`	sets the current level of gray: 0 is black, 1 is white
`w setlinewidth`	sets the current line width. This must be set at the beginning of a program.
`sx sy scale`	multiplies the current x and y scales by sx and sy
`x y translate`	translates the origin
`theta rotate`	rotates the current drawing map by θ
`x y add`	puts $x + y$ on the stack
`x y sub`	puts $x - y$ on the stack
`x y mul`	puts xy on the stack
`x y div`	puts x/y on the stack
`x neg`	puts $-x$ on the stack
`y x atan`	puts the polar angle of (x, y)
`x sqrt`	puts $\sqrt{x}$ on the stack
`x sin`	puts $\sin x$ on the stack
`x cos`	puts $\cos x$ on the stack
`y x exp`	puts y^x on the stack
`x ln`	puts $\ln x$ on the stack
`x abs`	puts $\lvert x \rvert$ on the stack
`/x ... def`	defines `x` to be whatever is on the stack after the sequence `...`
`showpage`	causes the picture to be printed

`PostScript` is a complete programming language which operates on data situated on a stack, very much like Hewlett-Packard calculators. Nearly all the `PostScript` commands we shall use fall in one of two categories: either (1) they affect the graphics state (the way in which figures are drawn) or (2) they do arithmetic on the stack. All commands remove the relevant part of the stack when there are executed. For example, the command `x y add` puts x, y, and the command `add` on the stack, then removs them and puts $x + y$ on the top of the stack. The sequence

```
a a mul b b mul add sqrt
```

replaces a a `mul` by a^2

replaces b b `mul` by b^2

replaces a^2 b^2 by $a^2 + b^2$

replaces that in turn by $\sqrt{a^2 + b^2}$

The stack looks like this in succession:

a

a a

a^2

a^2 b

a^2 b b

a^2 b^2

$a^2 + b^2$

$\sqrt{a^2 + b^2}$

In the initial `PostScript` state, the unit of measurement is one **point**, and there are 72 points to an inch. Thus the effect of `72 72 scale` is to change the basic unit to an inch. The initial origin is at the lower left of the page.

A number or variable in a program just puts the number or the current value of the variable on the stack. If a is a variable then `/a` is the name of the variable, not its value. The name is used to assign values to a variable. Thus the effect of the lines

```
/a 0.50 def
/b 1.00 def
/c a a mul b b mul add sqrt def
/sum a b add def
```

is to introduce variables a, b, c, and sum, and to set $a = 0.5$, $b = 1.0$, $c = \sqrt{a^2 + b^2}$, $sum = a + b$. Probably most variables in `PostScript` are real numbers, but we shall see later that they can also be integers, arrays, or even strings or sub-routines.