Tutorial: Animation

To start, get the simple demonstration from class up and running. Create the following in a file called `tiny_anim.pro` (or grab it from github)

```
nframes = 100
imsize = 300
xinteranimate,set=[imsize,imsize,nframes],title='Lines',/showload

plot,[0,nframes*5],[0,nframes*5],/nodata,xstyle=4,ystyle=4
for iframe=0,nframes-1 do begin
    oplot,[iframe*5,0],[iframe*5,nframes*5]
    img = tvrd()
    xinteranimate, image=img, frame=iframe
endfor

xinteranimate,30
end
```

Once you've successfully completed this step, add it to git (of course). Try to run it twice. If it complains (and it should), use `Control-C` in IDL to make it quit, then `.reset_session` to reset things.

Pick either the `snake.pro` or your own `random_walk.pro` code to animate. I think random walk will probably look cooler.

Turn the looped plots in these programs into `xinteranimate` animations:

1. Add a `xinteranimate` command to set up the plot:
   `xinteranimate,set=[imsize,imsize,nframes],title='title',/showload`
2. Add a `tvrd()` line to grab the current image (in the loop)
3. Add another `xinteranimate` command to set the image in the frame:
   `xinteranimate, image=img, frame=iframe`
   (in the loop)
4. Add a final `xinteranimate` command to start the animation:
   `xinteranimate,rate`
   Look up `xinteranimate` in the online help (`?xinteranimate`) to find out what "rate" can and should be

HINT: This error:
`% CW_ANIMATE_LOAD: Frame number must be from 0 to nframes -1.`
indicates an attempt to add a frame past the last frame, which might indicate an issue with your loop variables.

Use a common block to get access to the insides of a procedure

(this section requires a `journal` and a new procedure called `commonsquare.pro`

Write this simple code:

```
pro commonsquare,y
    common stuff,x
    x = y^2
end
```

Run this with some number. Then, in IDL interactively, "recall" the common block:
`IDL> common stuff,x`
and get the value of `x`.

`.reset_session`, then re-write the program:

```
pro commonsquare,y
    common stuff,x
    x = x * y^2
end
```

Now try running `commonsquare` again:
`commonsquare,2`
Does it work? Why not? What happens if you give `x` a value first?

This is just a trivial demonstration of what `common` blocks do: they can be used to give you access to namespaces that are normally hidden.