Tutorial: Objects

Objects can be useful in many situations. The homework covers objects as representations of data. This tutorial will cover objects as they relate to graphics.

In IDL, there is a `plot` procedure and a `plot` function. The `plot` *function* returns an object, documented here: http://www.exelisvis.com/docs/PLOT.html

For this tutorial, please use the "IDL Workbench" or IDLDE. Just run `idlde &` on the command line instead of `idl`. The IDLDE allows for *tab completion*, which is awesome.

Let's start with some pretty simple things:

- Plot a line!
  `pl = plot([0,1])`
- See what data is in the plot
  `pl.getdata,x,y`
  `print,x,y`
- Make a silly animation:
  `for ii=0,50 do pl.setdata,x,(-1)^ii*x`
- Create a new surface object, using IDL's built-in test data:
  `surf = surface(/test)`
- Rotate that object 35 degrees:
  `surf.rotate,35`
- Rotate it back:
  `surf.rotate,-35`
- Rotate it around the z axis:
  `surf.rotate,35,/zaxis`
- Loop:
  `for ii=0,65 do surf.rotate,1,/zaxis`
- Type `surf.` and just wait a moment. An interactive drop-down menu should show up. Skim down towards the bottom, with the "C" icons. These are "methods" on the surface object, i.e. they tell the surface object to do something. The drop-down menu should tell you the syntax.
- Overplot something on your first plot.
  `p2 = plot([1,0],[0,1],overplot=pl,color='FF0000'x)`
  `p3 = plot([0.5,0.5],[0,1],overplot=pl,color='00FF00'x)`

Object-oriented programming is very useful for graphics, since you can manipulate a graphic (change its properties) after writing it, instead of re-making the whole thing from scratch every time you want to make a little change.

Now we'll go back to some old stuff: surface shading.

In your `ASTR2600_materials` directory, grab the latest new code.

Open the `animate_sinsin.pro` code in an editor. Run it. See if you can open the resulting `.avi` file using `mplayer sinsin_anim.avi`.

Driving

Compile the `carclass__DEFINE.pro` and `CarDrive.pro` code, in that order. If you do it in the reverse order you will get syntax errors when you try to make cars.

Create three cars, `car1`, `car2`, and `car3`. Make them start at different locations, each with some nonzero amount of gas. For example:

```
; make an "empty" car, then "populate" its properties
car1 = carclass()
car1.fill_gas,5
car1.set_location,6
; use the "init" to fill in the gas,location parameters
car2 = carclass(25,-5)
help,car1,car2,/struct
```

Drive the cars around some. Keep an eye on their gas levels and fill 'em up as needed. The cars drive at 1 'distance' per 'gas', which could be interpreted as very inefficient 1 mile-per-gallon cars... (though note that theoretical astrophysicists like to use a unit system in which $G = 1$, $c = 1$, etc., such that the numbers are meaningless until scaled to "real" physical quantities):

```
car1.drive,5
car2.drive,11
car1.fill_gas,3
```

What happens if you drive too far and run out of gas? What happens if you drive *backwards*? (answer these in a comment)

Modify the `car.drive` method (`pro car::drive`) so that you can't fill it up by driving backwards (that is a bug!). Instead, gas should decrease no matter which way you drive.

Also, modify the `car.drive` method so that you can't drive if the gas level is zero. This means you'll need to do two things:
1. Check if the `distance` is greater than the `self.gas`
2. If it is, you need to do something about it. Let the car drive until the gas is zero, then stop. It's probably a good idea to print an error message too.