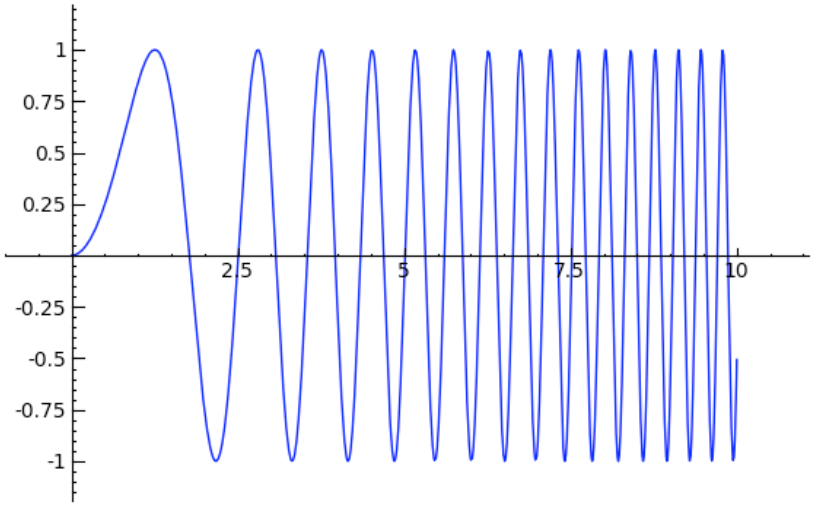
JPL09 - 2d graphics

2D Graphics

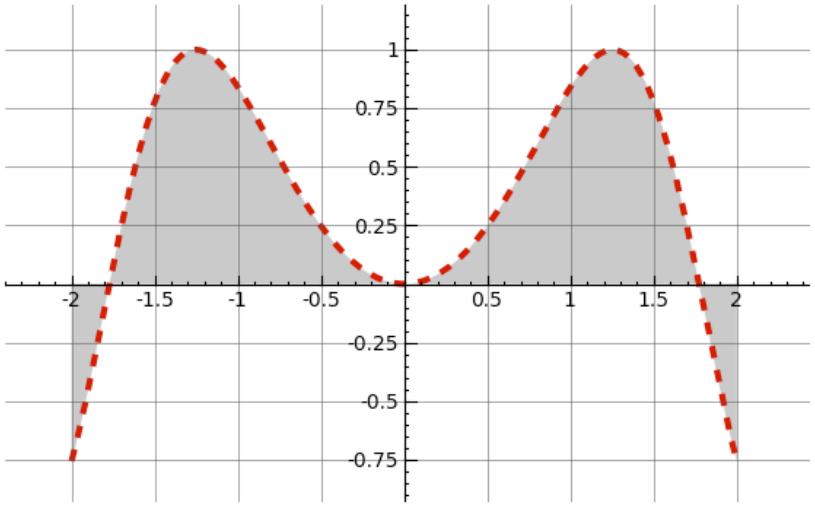
The **plot** command allows you to draw plots of functions. Type **plot(<tab key>** for the documentation (or look in the reference manual). We start by plotting the sin function. Try drawing the plot below.

```
var('x')   # make sure x is a symbolic variable
plot(sin(x^2), (x,0,10))
```



Here is a more complicated plot. Try to ***change every single input*** to the plot command in some way, evaluating to see what happens.

```
P = plot(sin(x^2), (x,-2,2), rgbcolor=(0.8,0,0), thickness=3, linestyle='--', fill='axis')
show(P, gridlines=True)
```

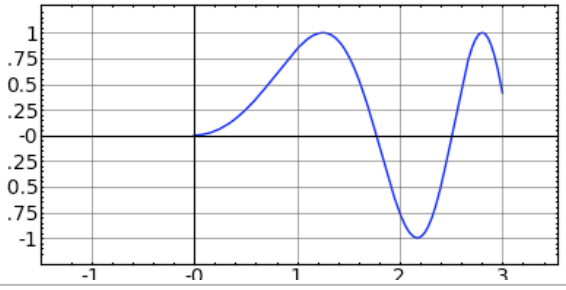


The **show command** which we used above allows us to show a plot after we create it.

P.show(options...)

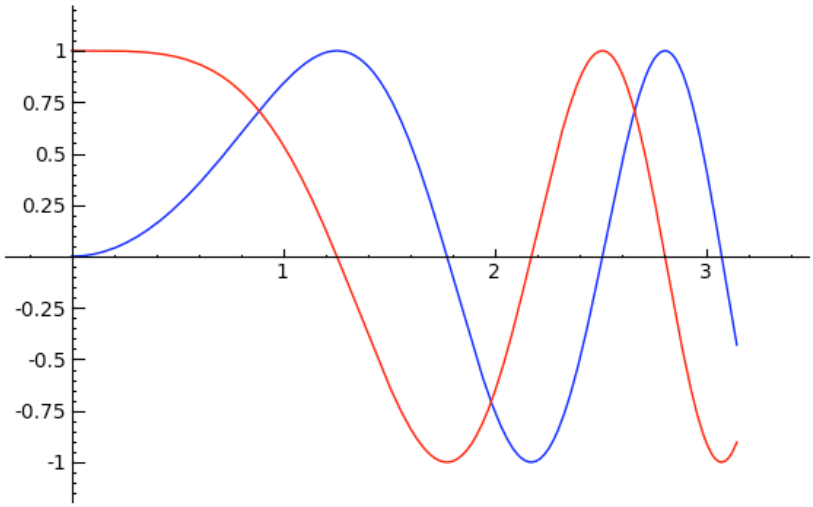
Here we use the show command to show a frame and gridlines instead of an axis. Try changing every single input to the show command below and seeing what impact this has. Try putting the cursor right after **P.show(** and pressing tab to get a list of the options for how you can change the values of the given inputs.

```
P = plot(sin(x^2), (x,0,3))
P.show(frame=True, gridlines=True, figsize=4, aspect_ratio=1, xmin=-1)
```



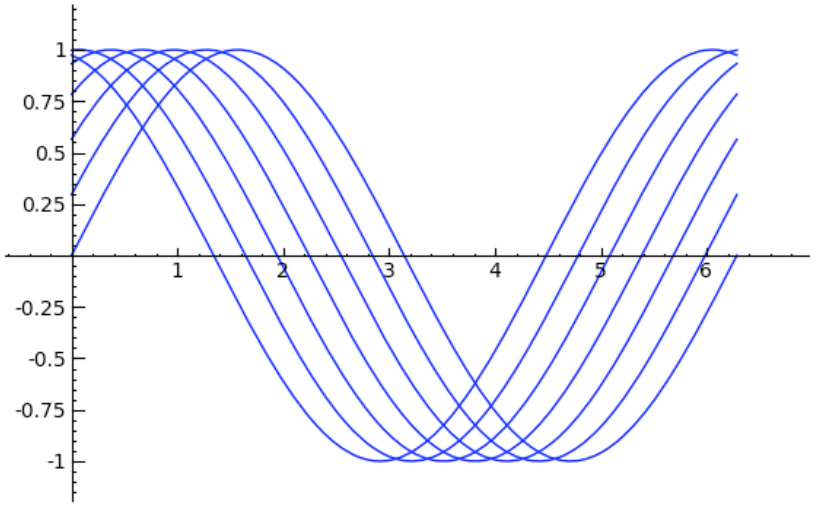
Plotting **multiple functions** at once is as easy as adding them together.

```
P1 = plot(sin(x^2), (x,0,pi))
P2 = plot(cos(x^2), (x,0,pi), rgbcolor='red')
P1 + P2
```



You can use the sum command to sum up many plots (try changing the range of values that n goes through). Notice below that the notation [a, a+eps, ..., b] makes a list of the values [a, a+eps, a+2*eps, a+3*eps, ..., b].

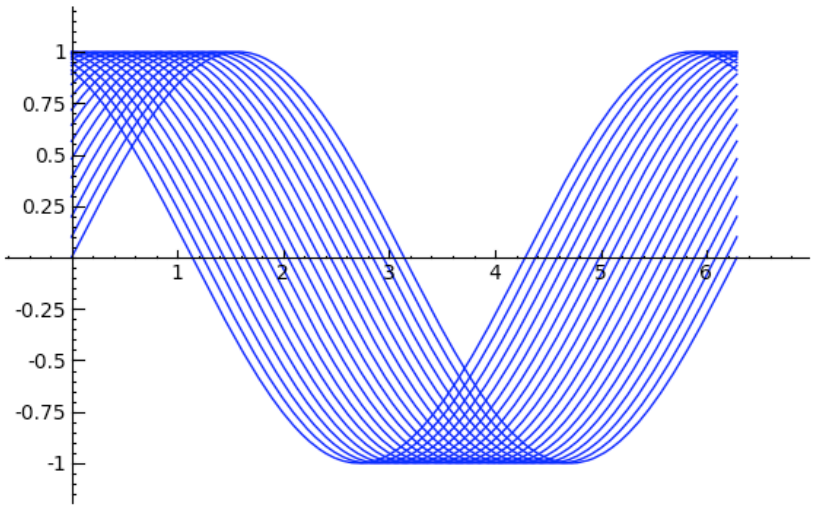
```
sum(plot(sin(x+n), (0,2*pi)) for n in [0,0.3,..,2])
```



We can also change eps ***interactively*:**

```
@interact
def f(eps=(0.1,1)):
    S = sum(plot(sin(x+n), (0,2*pi)) for n in [0,eps,..,2])
    S.show()
```

eps

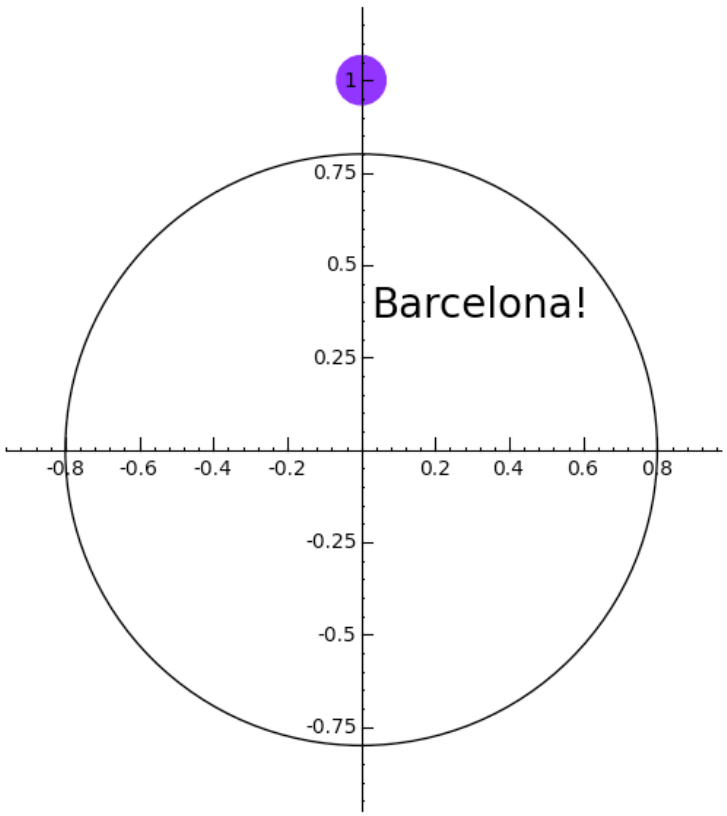


There are **many other 2d plotting commands** besides just "plot":

- line
- point (and points)
- polygon
- circle
- disk
- text
- arrow
- parametric_plot
- polar_plot
- implicit_plot
- contour_plot
- density_plot
- region_plot
- scatter_plot
- bar_chart
- plot_vector_field, plot_slope_field
- matrix_plot
- complex_plot
- graphics_array
- animate

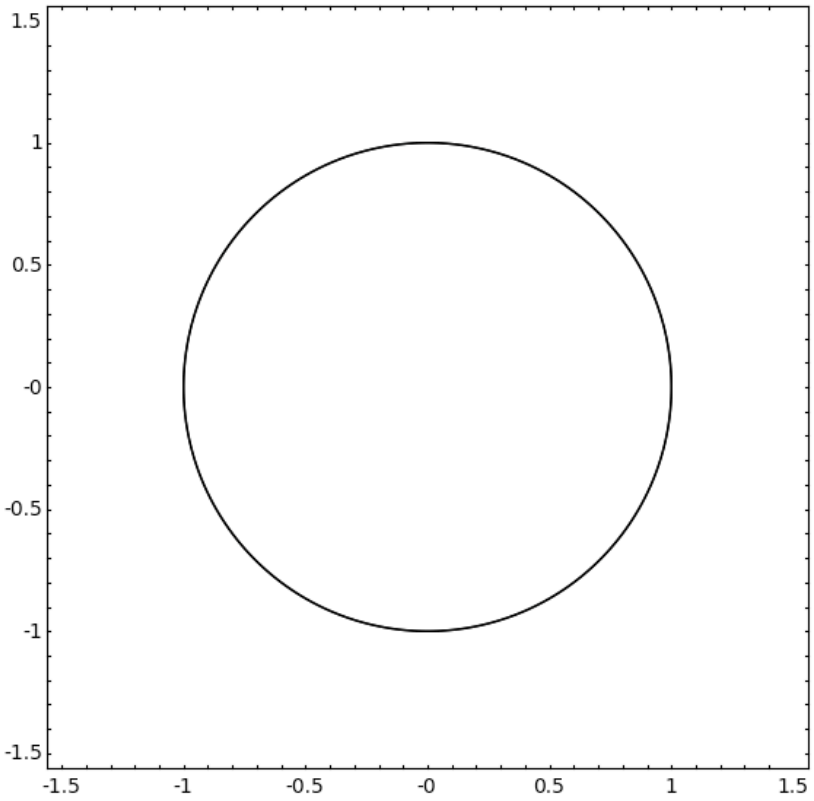
You can combine any of the above together in the same figure using + (try to move the purple dot below to the right by editing the input):

```
P = point((0,1),pointsize=500,rgbcolor='purple') + circle((0,0),0.8)
P += text("Barcelona!", (0.32,0.4), fontsize=20, rgbcolor='black')
P.show(aspect_ratio=1)
```



Drawing implicit plots is often useful. Here we make a small interact that draws an implicit plot. Try different input functions.

```
var('x,y')
P = implicit_plot(x^2 + y^2 == 1, (x,-1.5,1.5), (y,-1.5,1.5), plot_points=300)
P.show(aspect_ratio=1)
```



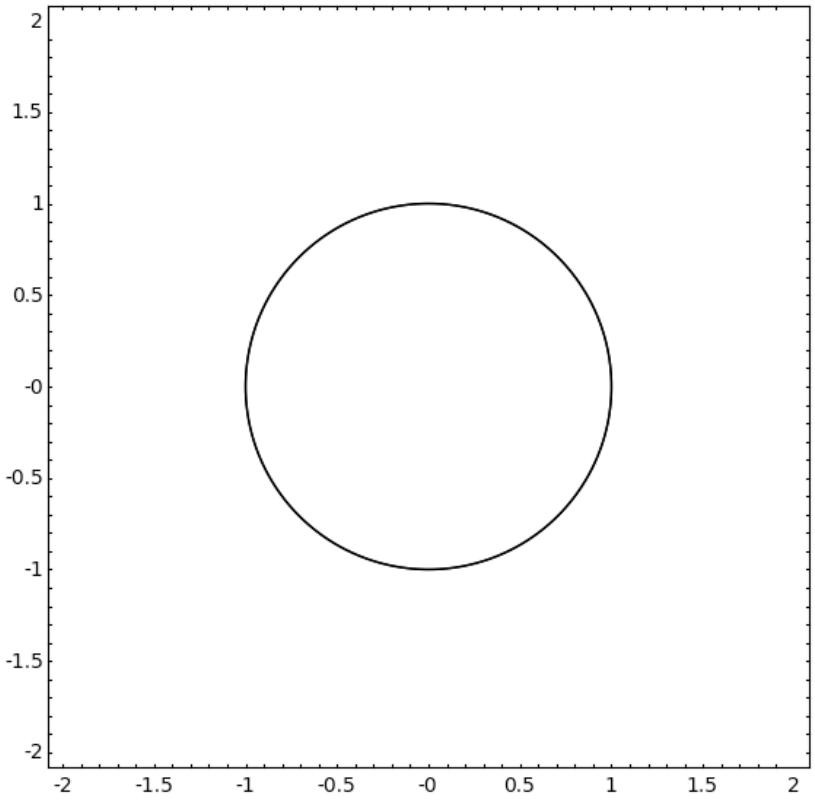
Here is an interact that you can use to play around more with the implicit_plot command.

```
var('x,y')
@interact
def demo(f=x^2 + y^2 == 1, x_range=range_slider(-2,2,default=(-2,2)),
                           y_range=range_slider(-2,2,default=(-2,2))):
   show(implicit_plot(f, (x,x_range[0],x_range[1]),(y,y_range[0],y_range[1]),
                      plot_points=300),aspect_ratio=1)
```

f `x^2 + y^2 == 1`

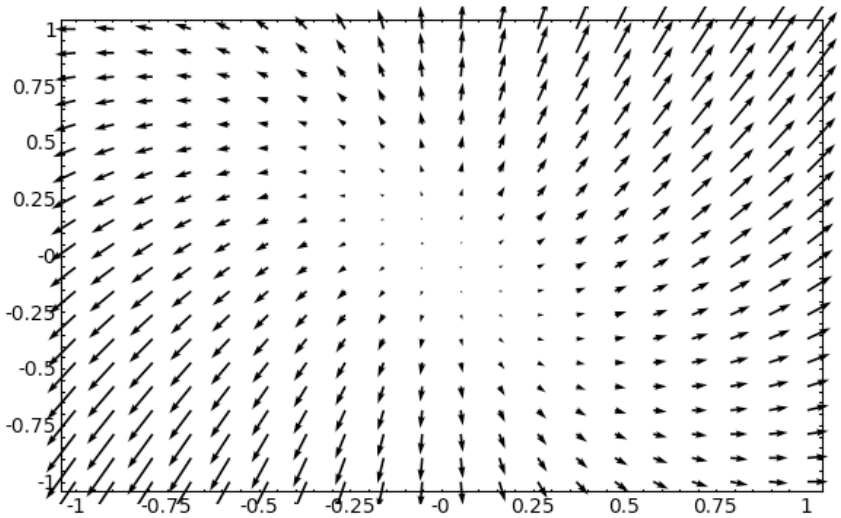
x_range

y_range



Here is an example of plotting a **vector field**. Try changing the function that gives the vector and re-evaluating.

```
var('x,y')
plot_vector_field((sin(x),x+y), (x,-1,1), (y,-1,1))
```



While you wait for everybody to catch up, try out as many of the other 2d plotting functions as you can before we move on to 3d graphics. In each case, remember that you can type **command_name(<tab key>** to get help.