Phys 238 : Getting Started With *Mathematica*

Note that Mathematica notebooks can contain a mix of commands and text, and can be organized into sections and subsections with appropriate titles. Click on the cell selection bar at the far right of a cell, and then select Format -> Style to change the style. Check the various cells in this notebook to see some examples.

Getting Started

Mathematica remembers definitions, even if you delete them from the notebook. This can sometimes be confusing, so it's often a good idea to simply clear everything and start anew. This is especially true if you use the Evaluation -> Evaluate Notebook menu item.

In[1]:=
```
Clear["Global`*"] (* Clear all variable and function definitions *)
```

(Text entered between (* and *) is a comment. It is ignored by Mathematica, but is useful to give the reader helpful hints.)

Variables

The assignment operator is the '=' sign. Anything inside (* *) is a comment.

In[2]:=
```
x = 7 (* Assign the value 7 to the varaiable x *)
```

Out[2]= 7

The next odd-looking expression takes the value of x (7), adds one to it, and then assigns the result back to the variable x:

In[3]:=
```
x = x + 1
```

Out[3]= 8

Now x is 8.

In[4]:=
```
x
```

Out[4]= 8

Variables can be more than one letter. All *Mathematica* functions start with an upper case letter (e.g. Cos, Sin, Log, Sqrt) so by convention, user variables usually start with lower case letters.

In[5]:=
```
mass = 0.080
```

Out[5]= 0.08

Mathematica prints out the result automatically. You can suppress that by ending a line with a semicolon:

In[6]:=
```
k = 4;
```

You test for equality with a double equals sign, '=='

```
In[7]:= x == 7
Out[7]= False

In[8]:= x == 8
Out[8]= True
```

You can also use Greek letters such as ω and Δ. Use the menu item Palettes -> Classroom Assistant to open up the Classroom Assistant Palette, which has handy ways to enter a variety of mathematical quantities. If you let the mouse hover over any item, Mathematica will also tell you the keyboard shortcut. For example, you can enter ω by pressing ESC w ESC.

Lists

A list is a collection of items in curly braces, separated by commas. (Search for the "Making Lists of Objects" Tutorial in the on-line Help.)

Here is a list of five items:

```
In[9]:= list1 = {1, 2, 3, 4, 5}
Out[9]= {1, 2, 3, 4, 5}
```

One handy way to create a list is with the Table[ ] function. The first example creates a list of successive values of 'i', where 'i' goes from 1 to 5. The second goes from 0 to 1 in steps of 0.2. See many more examples in the online help for Table.

```
In[10]:= list2 = Table[i, {i, 1, 5}]
Out[10]=
{1, 2, 3, 4, 5}

In[11]:= list3 = Table[i, {i, 0, 1, 0.2}]
Out[11]=
{0., 0.2, 0.4, 0.6, 0.8, 1.}
```

This last example shows that the Table can use any function of the variable. Here is a list of the squares of the first 5 integers:

```
In[12]:= list4 = Table[n^2, {n, 1, 5}]
Out[12]=
{1, 4, 9, 16, 25}
```

Nested Lists

Each item in a list can also be a list. Suppose we want to calculate position x as a function of time t. Each of our data points will be a list:

In[13]:=
```
p1 = {t1, x1} (* t1 and x1 are the initial time and position. *)
```
Out[13]=
```
{t1, x1}
```

In[14]:=
```
p2 = {t2, x2} (* t2 and x2 are the second time and position. *)
```
Out[14]=
```
{t2, x2}
```

We can then build up our data as a list of data points:

In[15]:=
```
data = {p1, p2}
```
Out[15]=
```
{{t1, x1}, {t2, x2}}
```

Sometimes, it is easier to visualize if you use TableForm or MatrixForm

In[16]:=
```
TableForm[data]
```
Out[16]//TableForm=
```
t1    x1
t2    x2
```

In[17]:=
```
MatrixForm[data]
```
Out[17]//MatrixForm=
$$\begin{pmatrix} t1 & x1 \\ t2 & x2 \end{pmatrix}$$

Finally, here we make a list of data for the function x = Cos[t], where t runs from 0 to 1 in steps of Δx = 0.2.

In[18]:=
```
data = Table[{t, Cos[t]}, {t, 0, 1, 0.2}]
```
Out[18]=
```
{{0., 1.}, {0.2, 0.980067}, {0.4, 0.921061},
 {0.6, 0.825336}, {0.8, 0.696707}, {1., 0.540302}}
```

In[19]:=
```
TableForm[data]
```
Out[19]//TableForm=
```
0.     1.
0.2    0.980067
0.4    0.921061
0.6    0.825336
0.8    0.696707
1.     0.540302
```

Functions

Functions are rules that act on patterns. Functions in *Mathematica* act on arguments that are enclosed in square brackets. Here is a simple function that calculates the square of a number. On the left hand side, use the underscore '_' to indicate a pattern, and use the := to use the 'SetDelayed' feature. Again, see the HowTo for more details.

In[20]:=
```
Clear[x] (* Erase the old value of x to avoid confusion *)
```

In[21]:=
```
f[x_] := x^2 (* Note how the 'x' here is green,
  indicating it is a "placeholder" variable in the function. *)
```

In[22]:=
```
f[x]
```

Out[22]=

x^2

The thing in the square braces can be anything.

In[23]:=
```
f[6]
```

Out[23]=

36

In[24]:=
```
f[banana]
```

Out[24]=

banana^2

In[25]:=
```
f[♪]
```

Out[25]=

$♪^2$

If you haven' t defined a function yet, *Mathematica* simply echos it back:

In[26]:=
```
g[x]
```

Out[26]=

g[x]

A function knows how many arguments it is supposed to have. If you call it with the wrong number of arguments, *Mathematica* assumes you meant a different function that you just haven't defined yet, and so just echos it back.

In[27]:=
```
f[a, b]
```

Out[27]=

f[a, b]

Simple Plots

If you have a list of data points, it is easy to plot them.

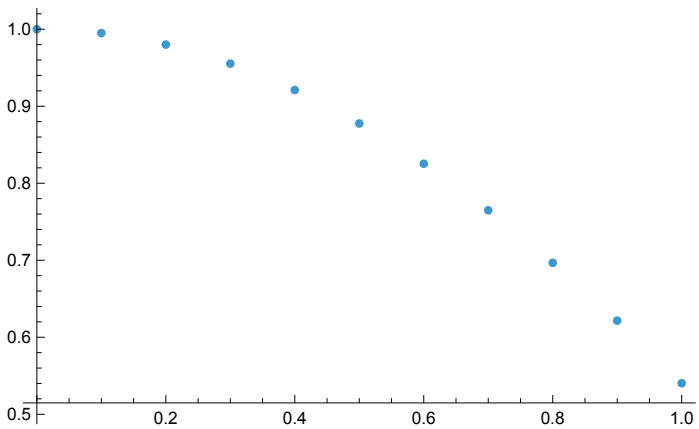
In[28]:=
```
data1 = Table[{t, Cos[t]}, {t, 0, 1, 0.1}];
TableForm[data1]
```

Out[29]//TableForm=

0.	1.
0.1	0.995004
0.2	0.980067
0.3	0.955336
0.4	0.921061
0.5	0.877583
0.6	0.825336
0.7	0.764842
0.8	0.696707
0.9	0.62161
1.	0.540302

In[30]:=
```
ListPlot[data1]
```

Out[30]=



If you have more than one data set, you can plot them both on the same graph. This is often convenient when comparing calculated values with a theoretical prediction.

In[31]:=
```
data2 = Table[{t, Sin[t]}, {t, 0, 1, 0.1}];
TableForm[data2]
```

Out[32]//TableForm=

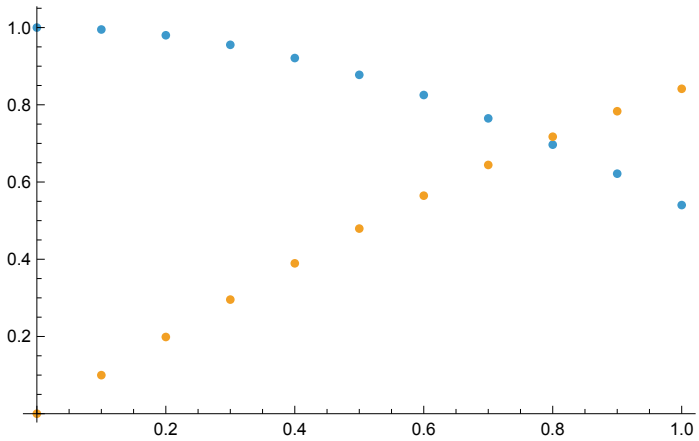
0.	0.
0.1	0.0998334
0.2	0.198669
0.3	0.29552
0.4	0.389418
0.5	0.479426
0.6	0.564642
0.7	0.644218
0.8	0.717356
0.9	0.783327
1.	0.841471

This next command plots both together. The first argument to ListPlot is always the data to plot. In this case, the first element is a list (in curly braces). That list contains two data sets,

data1 and data2. Each data set is, in turn, its own list of {t, x} pairs.

In[33]:=
```
ListPlot[{data1, data2}]
```

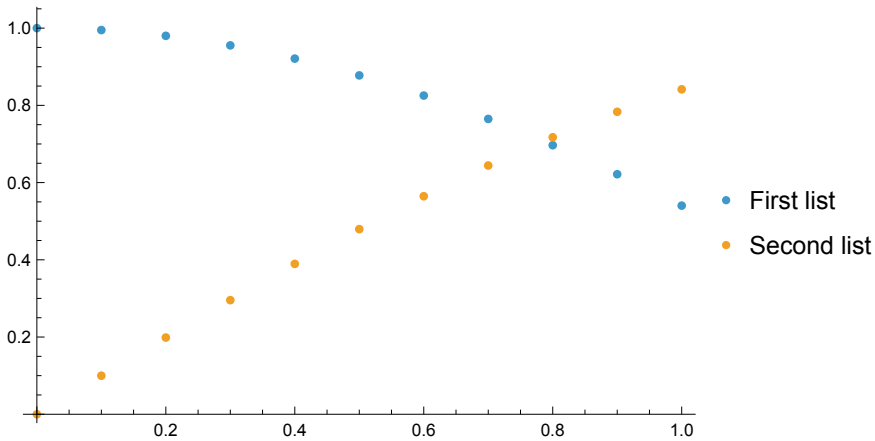
Out[33]=



It can be confusing to read such a graph unless you have a legend telling you which color goes with with data set. The PlotLegends option allows you to do just that. It takes a list (enclosed in curly braces) of names to apply to each data set. In this case, we also assign the plot to a variable name:

In[34]:=
```
dataplot = ListPlot[{data1, data2}, PlotLegends → {"First list", "Second list"}]
```

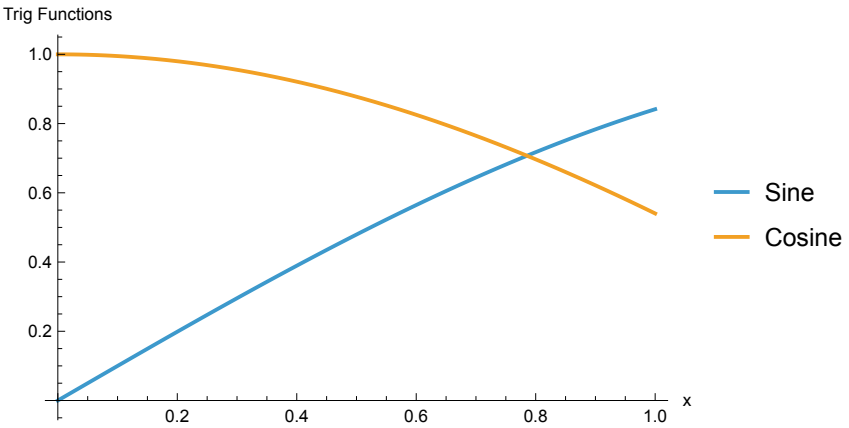
Out[34]=



Similarly, you can overlay two function plots. To compare the sine and cosine, for example, you could do

In[35]:=
```
Plot[{Sin[x], Cos[x]}, {x, 0, 1},
 PlotLegends → {"Sine", "Cosine"}, AxesLabel → {"x", "Trig Functions"}]
```

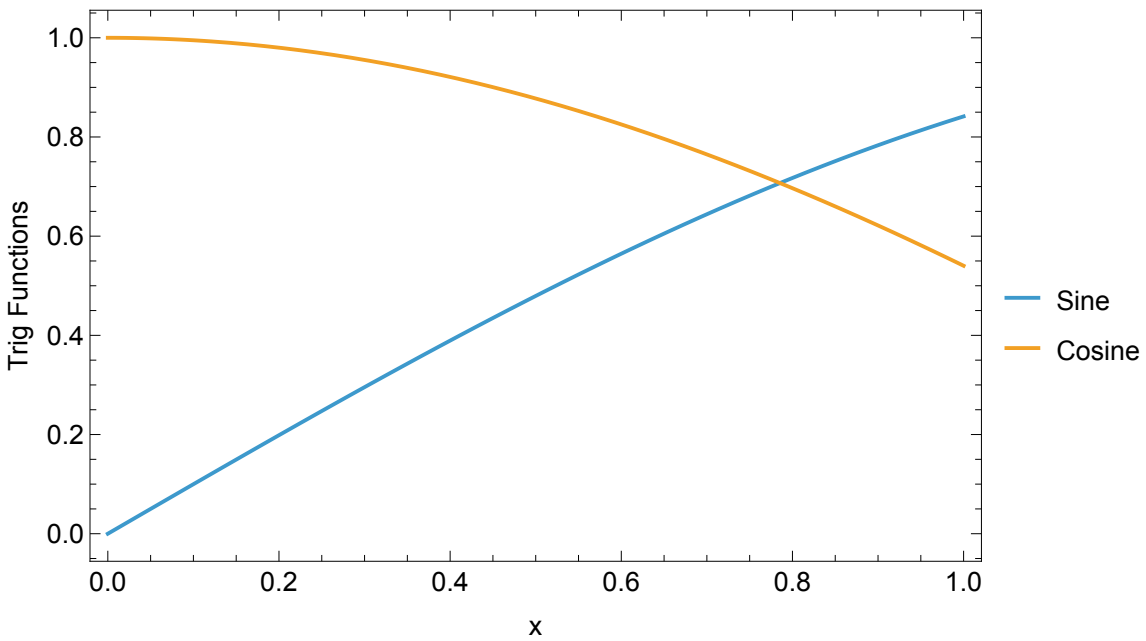
Out[35]=



Next, to make a nice plot, it is usually helpful to use a larger font for the axes labels, enclose the plot in a Frame, and make the image larger (e.g. this example uses 80% of the full screen).

In[36]:=
```
functionplot = Plot[{Sin[x], Cos[x]}, {x, 0, 1},
  PlotLegends → {"Sine", "Cosine"},
  Frame → True, FrameLabel → {"x", "Trig Functions"},
  LabelStyle → Larger, ImageSize → Scaled[0.80]]
```

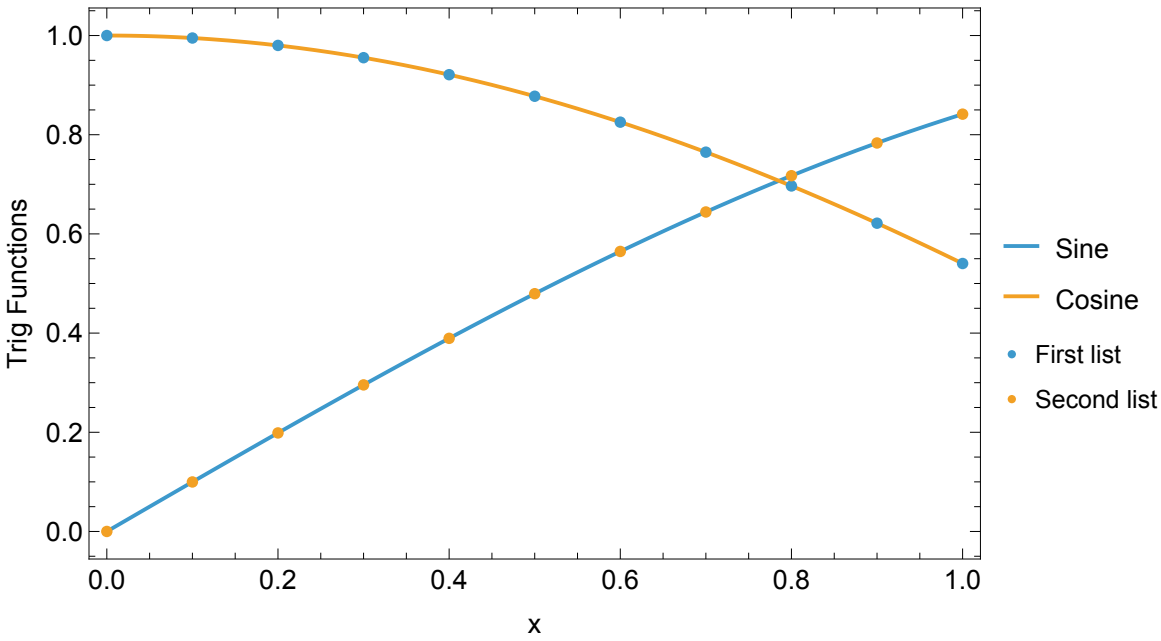
Out[36]=



Finally, you can superimpose the 'data' plot with the 'function' plot. This is usually useful when you are fitting an equation to data. The Show[ ] command does this. The first argument is a list (in curly braces) of the plots to show. The remaining arguments are any graphics options. In this case, I have chosen to make the graph larger, specifying it at 80% of full-scale.

In[37]:=
```
plotcomparison = Show[
  {functionplot, dataplot},
  ImageSize → Scaled[0.80]]
```

Out[37]=



Exporting Graphics

To use your graph in a LaTeX document, you should first Export it. Although LaTeX can import many different types of graphics (such as PNG or JPEG files) you often get the best results by using PDF files.

It is not always obvious where Mathematica will save your graph. It is often useful to save it in the same directory as the notebook. The following command sets the default director to be the same as the notebook directory. (This trick is also very useful for importing data.)

In[39]:=
```
SetDirectory[NotebookDirectory[]]
```

Out[39]=
```
/Users/doughera/notes/www/public_html/courses/phys238-2025/Mathematica
```

In[40]:=
```
Export["PlotComparison.pdf", plotcomparison]
```

Out[40]=
```
PlotComparison.pdf
```