```julia
begin
    using Plots
    using PlutoUI
    using PlutoTeachingTools
end
```

Table of Contents

Exercise Session 0: Basic Plotting in Julia

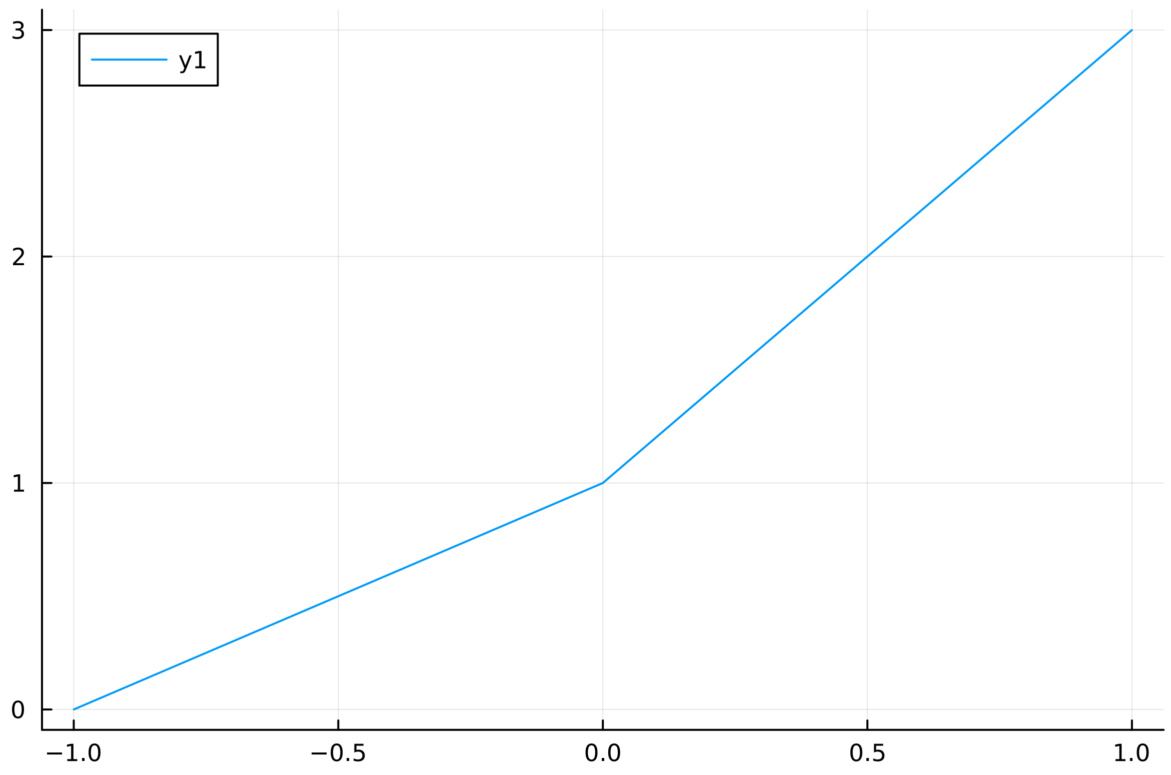
This notebook is meant to give you an introduction to the `Plots` Julia plotting library.

References:

- The reference source of information is the official Plots documentation.
- An alternative introduction is found in the Plots basics and Plots tutorial.
- Fancy examples are available on the official gallery.

Line plots

To plot a line, pass lists of `x` coordinates and `y` coordinates to `plot`:

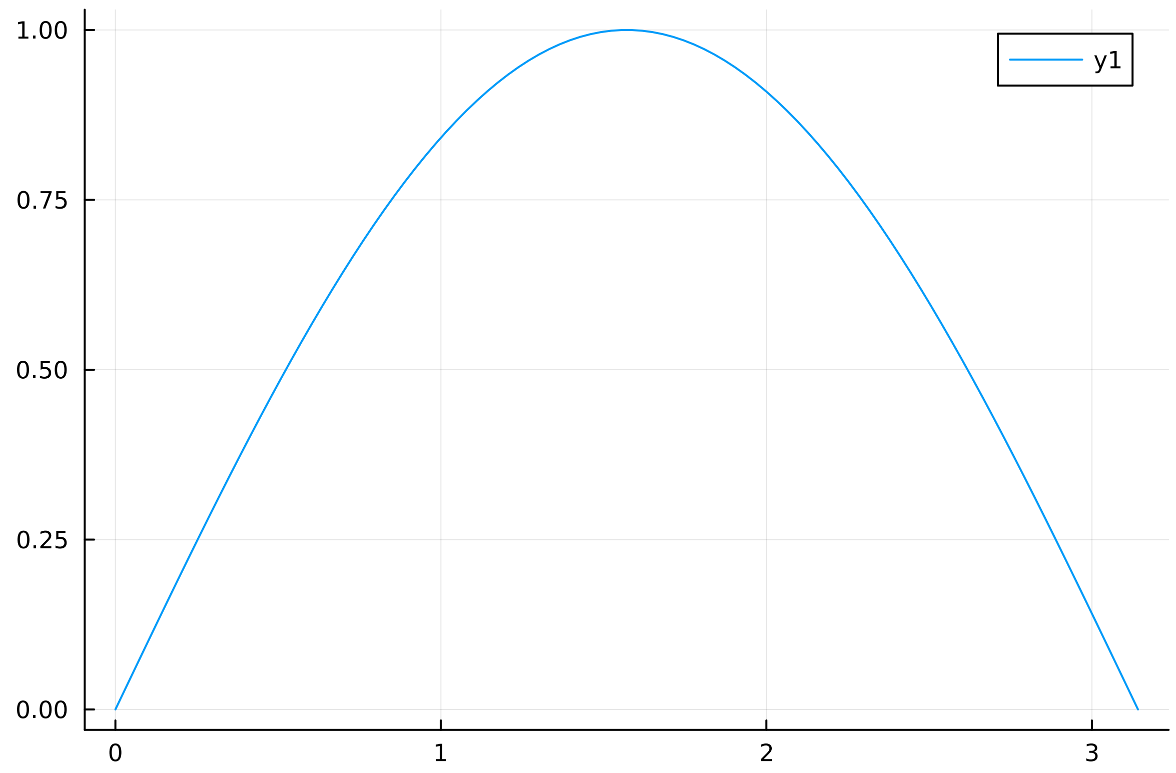


```julia
# Plot a line going from (-1, 0) to (0, 1) to (1, 3)
let
    xs = [-1, 0, 1]
    ys = [0, 1, 3]
    plot(xs, ys)
end
```

To plot a continuous function, we use the following principle:

- Generate many `x` coordinates with `range(a, b, length=N)`.
- Compute the corresponding `y` coordinates by calling the function.

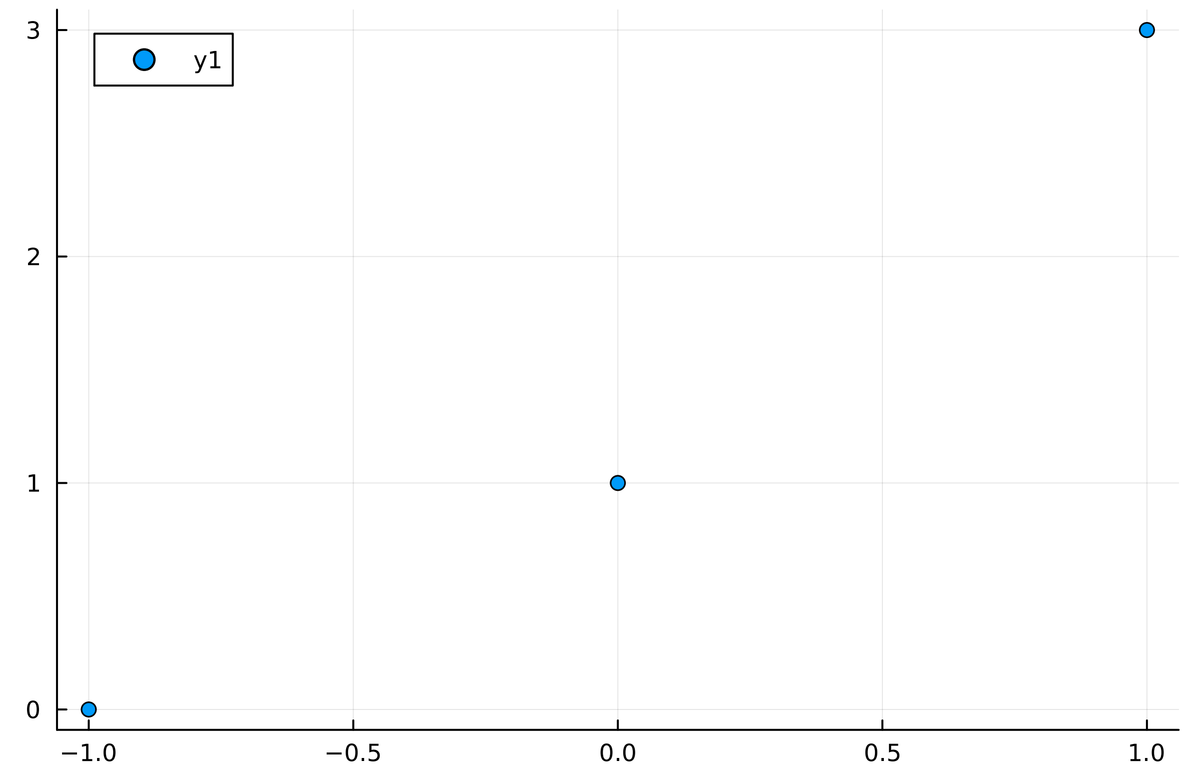
Here is a plot of the `sin` function between 0 and π:



```julia
let
    # x coordinates, here 100 points between 0 and π
    xs = range(0, π; length=100)
    # Compute corresponding y coordinates
    # Remember: calling a function with .() applies it to all elements of the vector
    ys = sin.(xs)
    plot(xs, ys)
end
```

Scatter plots

Scatter plots are also common. Each point is represented by a dot, and the points don't get connected together. For example:



```julia
# Plot the points (-1, 0), (0, 1), and (1, 3)
let
    xs = [-1, 0, 1]
    ys = [0, 1, 3]
    scatter(xs, ys)
end
```

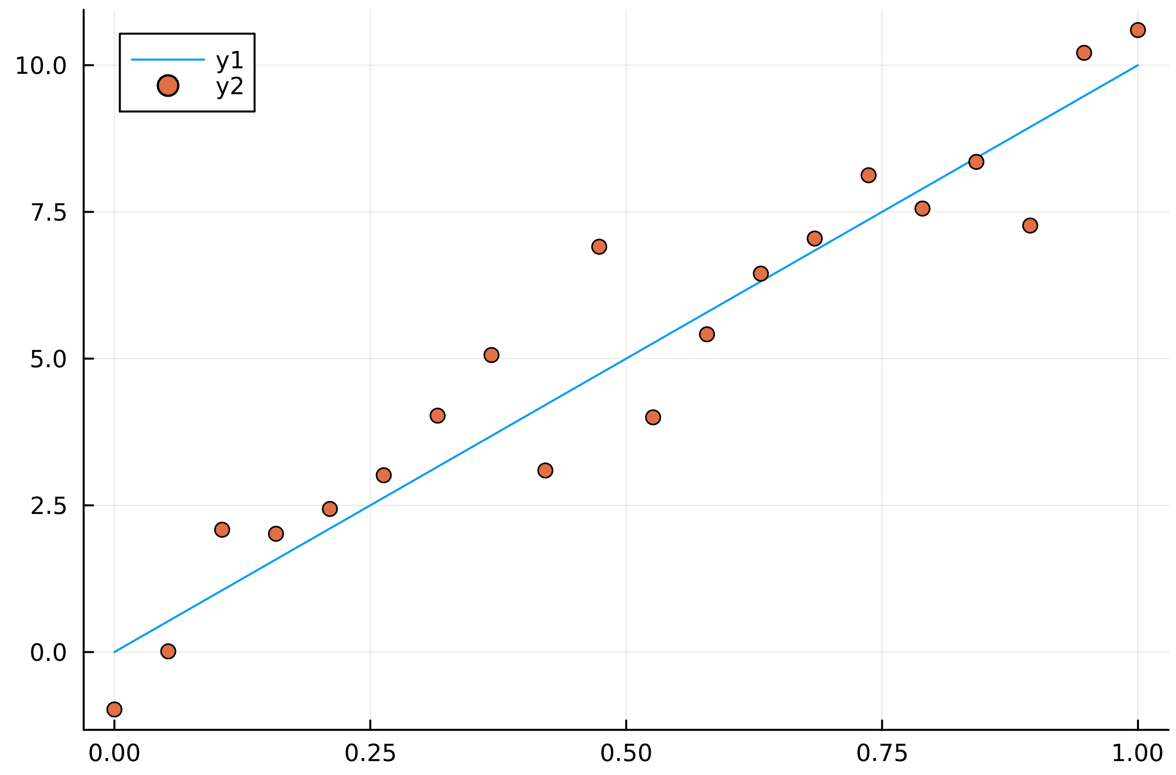
Multiple plots per figure

In Julia, functions that perform a modification often end with a `!` by convention. Plots follows this convention:

- `plot(...)`: Create a new plot then do something to it.
- `plot!(...)`: Modify an existing plot.

Similarly, `scatter(...)` creates a new plot while `scatter!(...)` modifies an existing plot.

Here is an example of plotting a new line with `plot` then adding some points with `scatter!`:



```julia
let
    xs = range(0, 1; length=20)
    # Plot line with equation y = 10x
    plot(xs, xs * 10)
    # Plot points with added (Gaussian) noise
    scatter!(xs, xs * 10 + randn(20))
end
```

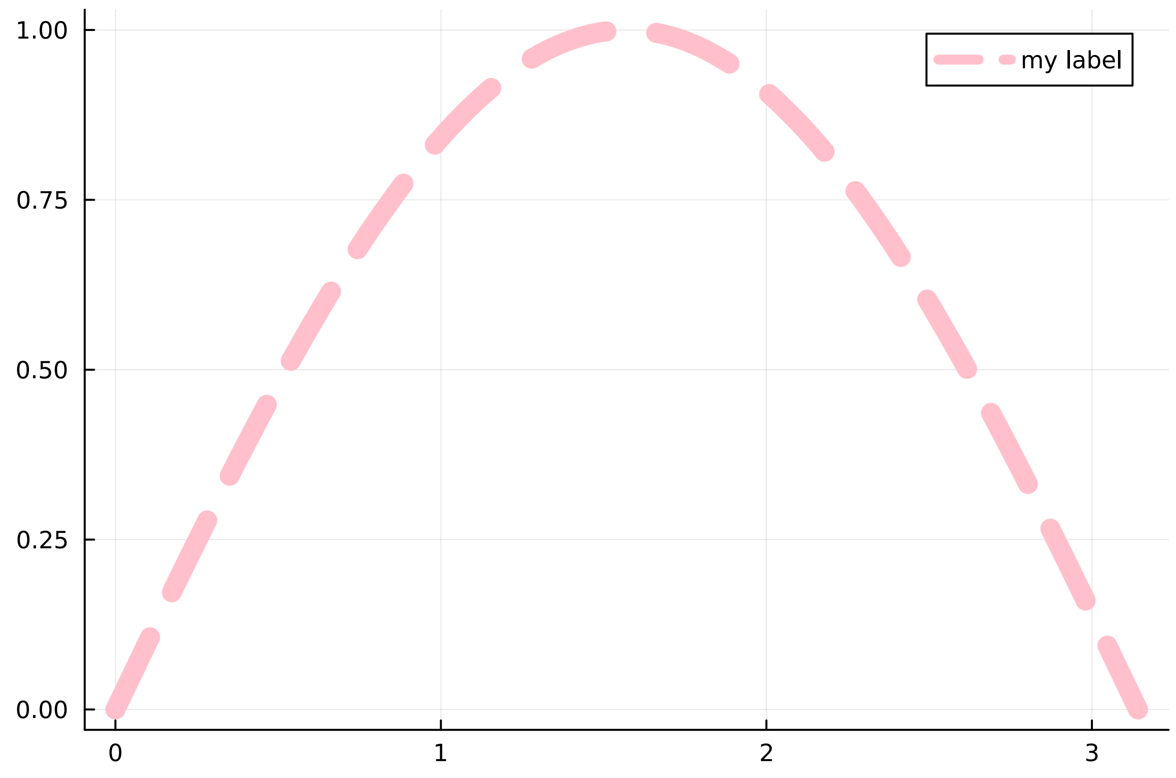
Series attributes

Each dataset added through `plot` or `scatter` is called a *series*. Additional options can be passed, called *attributes*. A full list is available in the official documentation. Here is a selection:

- `label`: Changes the name of the series in the legend.
- `color`: Changes the color of the series.
- `linestyle`: Changes the style of the line. For example `:solid` or `:dot`.
- `linewidth`: Width of the line in pixels.

These also apply to `scatter` (and other types of plots).

Here is a showcase of all of these options:



```julia
let
    xs = range(0, π; length=100)
    # Note how extra options are passed. Multiple options can be combined!
    plot(xs, sin.(xs); label="my label", color=:pink, linestyle=:dash, linewidth=10)
end
```

Exercise

Find the possible values of `linestyle` in the official documentation, and complete them in the following cell.

```
linestyle_values = ▸[:solid]
```

```julia
linestyle_values = [:solid]
```

Missing Response

Replace `[:solid]` by the list of allowed values for `linestyle`.

Plot attributes

Some attributes can be used to modify the entire plot.

A plot can contain multiple subplots, but we won't cover that in this tutorial. However this means that the official documentation has a page for plot attributes and a page for subplot attributes.

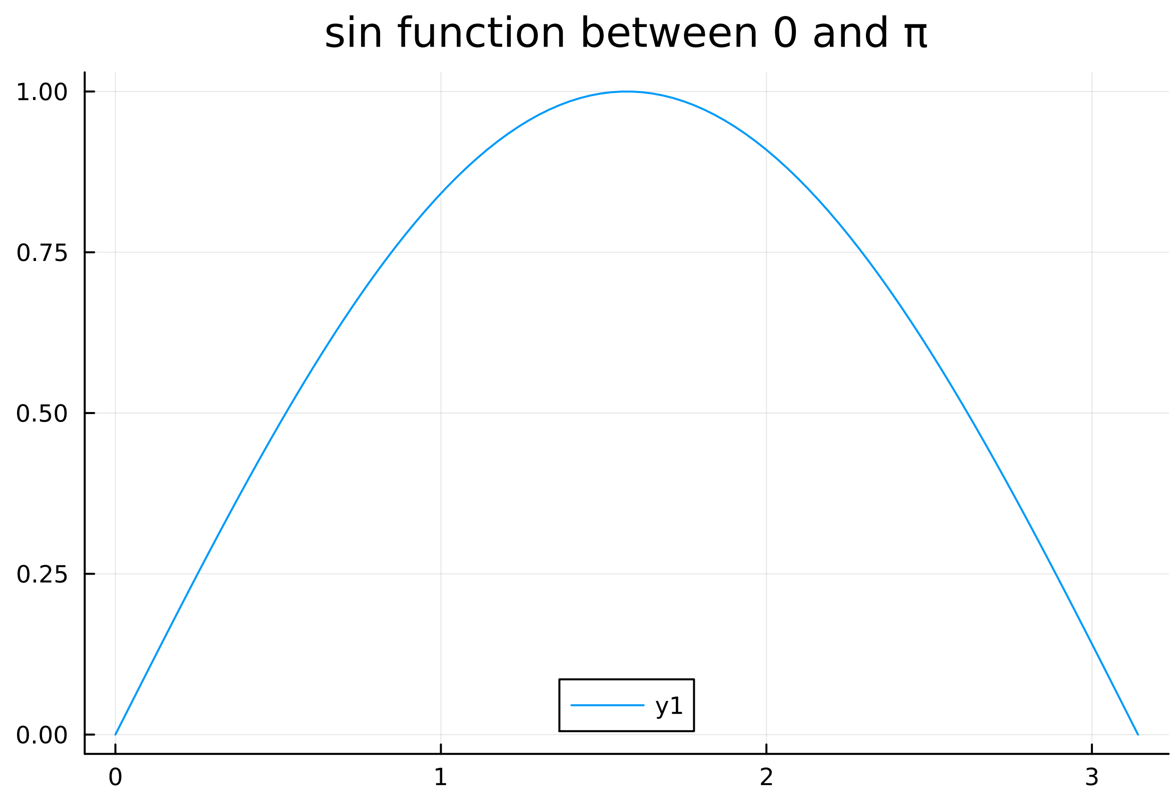
Here is a selection:

- `title`: Change the title displayed at the top of the plot.
- `legend`: Position of the legend. `false` can be used to disable the legend.
- `dpi`: "Dots Per Inch" of the output figure. Can be used to increase the quality of figures saved as an image. (See below). Default is `100`.

Plot attributes are also passed to `plot` / `plot!`. They can be passed at the same time as a series, or separately such as

```
plot!(; title="My title")
```

As usual, here is an example:



```julia
let
    xs = range(0, π; length=100)
    plot(xs, sin.(xs))
    # Change the title and move the legend to the bottom
    plot!(; title="sin function between 0 and π", legend=:bottom)
end
```

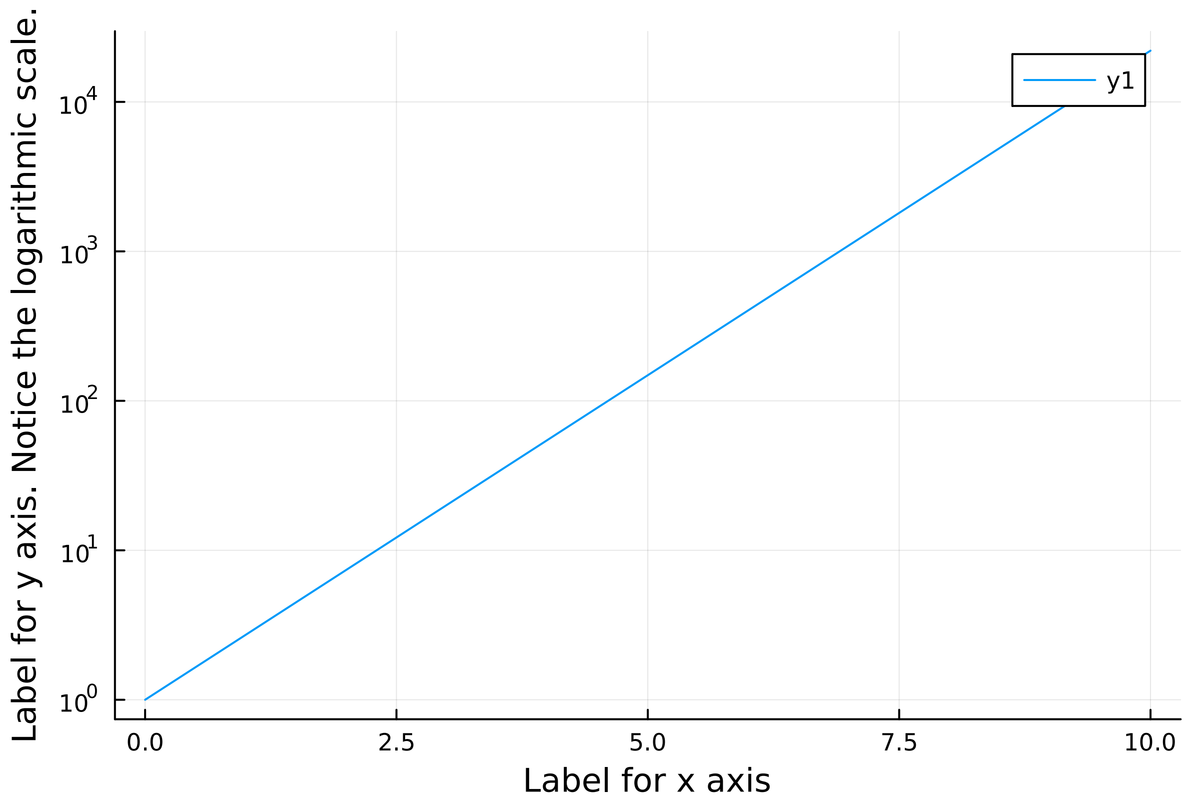
Axis attributes

Some attributes can also apply to the `x` axis (horizontal) or the `y` axis (vertical). Here is a selection:

- `xscale` and `yscale`: Scale of the axis. Most importantly, `:log10` can be passed to make the axis logarithmic.
- `xlabel` and `ylabel`: Name of the axis.
- `xlims` and `ylims`: Range of the axis. For example, `[5, 10]` to make the axis range from 5 to 10.

The full list is available in the official documentation, without the `x` or `y` prefix.

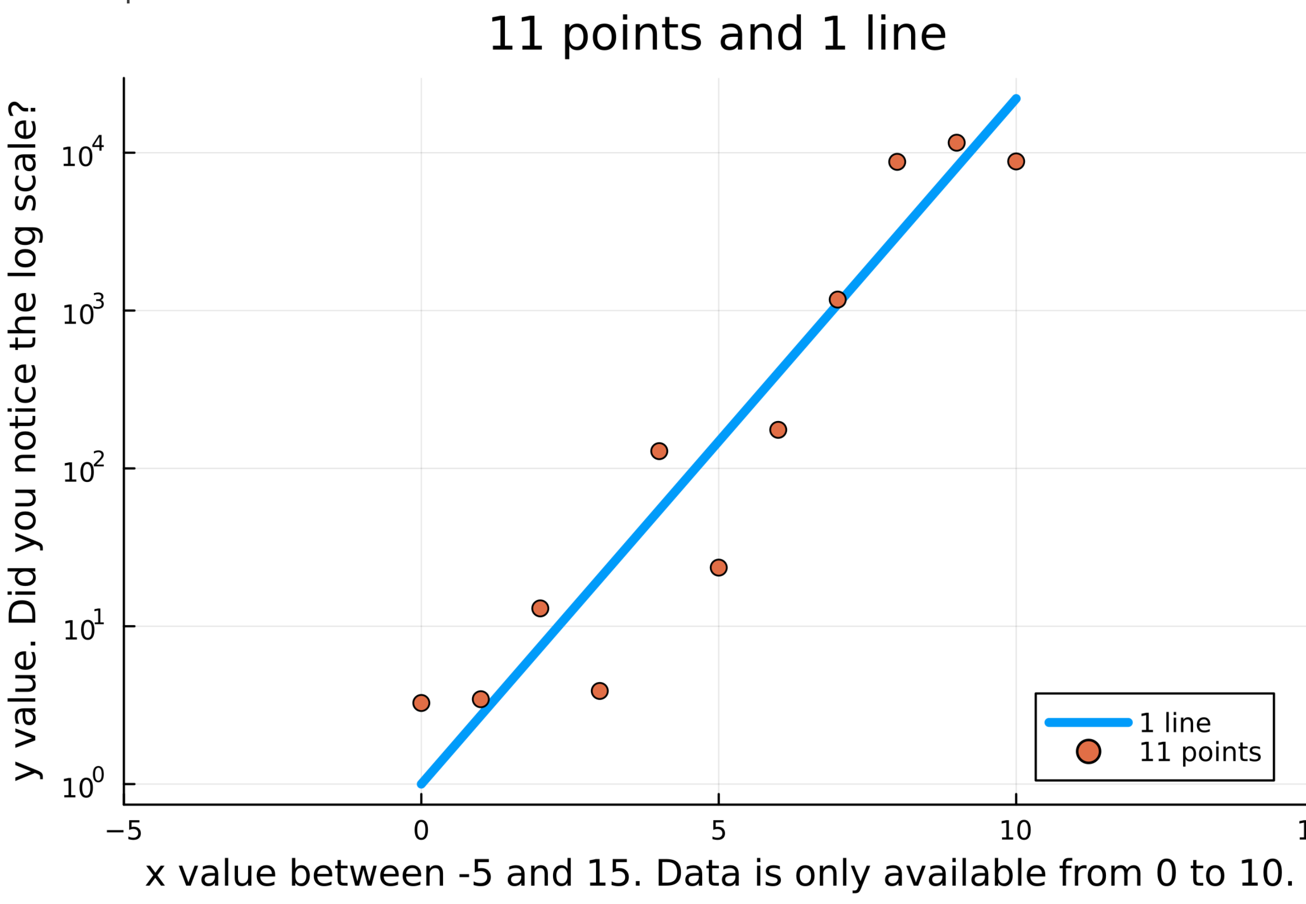
Here is an example:



```julia
let
    xs = range(0, 10; length=100)
    plot(xs, exp.(xs))
    plot!(; xlabel="Label for x axis")
    plot!(; yscale=:log10, ylabel="Label for y axis. Notice the logarithmic scale.")
end
```

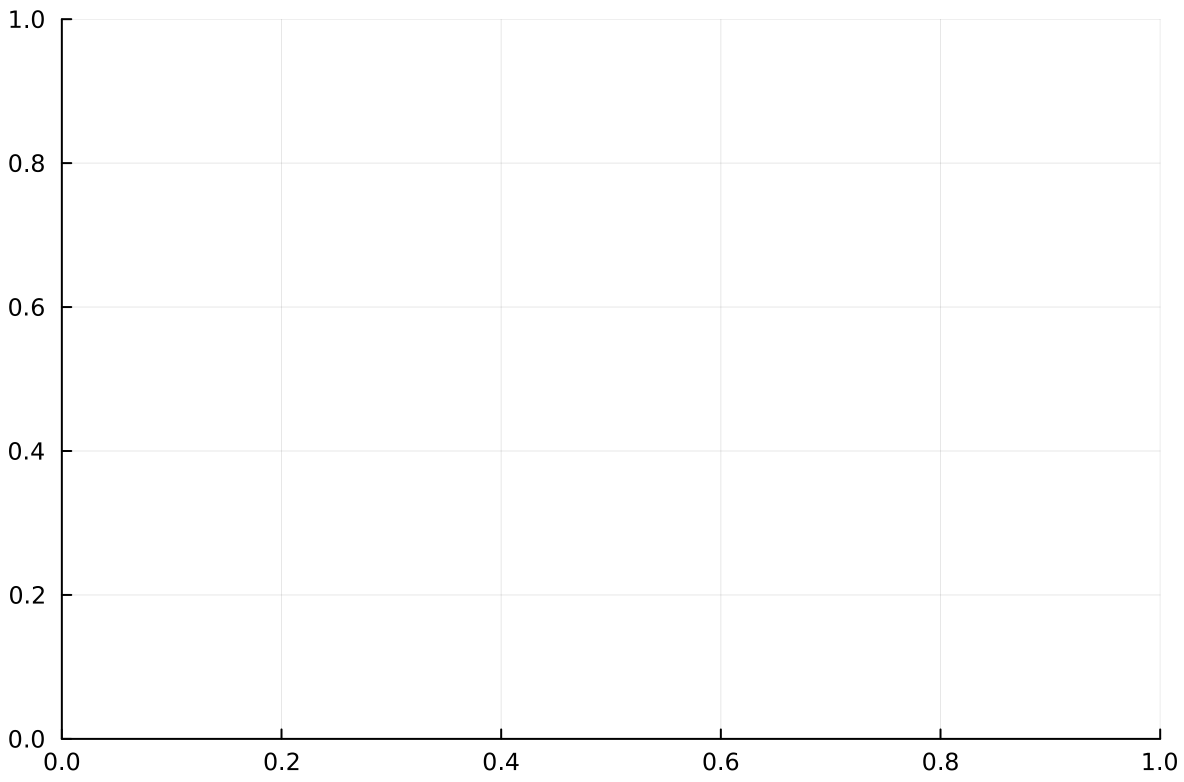
Practice Time

Here is a plot:



Exercise

Complete the cell below to produce a similar plot. The data for the line and the scatter plot is provided, and you have to plot it. Pay attention to the axes, the legend, the title, and so on.



```julia
let
    # The data points are given to you.
    xs = range(0, 10; length=11)
    ys_points = [3.264219326544821, 3.4489309156347425, 12.975323123905081,
    3.8904467103807927, 128.650270688119, 23.524783240700987, 175.52405606163614,
    1173.8330021528288, 8755.08696752278, 11573.569080380066, 8807.211902252335]
    ys_line = exp.(xs)
    # Fill in the plotting code here:
    plot()
end
```

The End

This concludes the tutorial on plots in Julia.

The next section explains how to save plots, for example to include them in your reports. Feel free to skip that section if you are not interested, you won't need it in this class!

In any case, the third and last notebook for this week awaits you on Moodle.

Optional: Saving plots

(This is not needed for this course, but could be useful to you in the future.)

The `plot` / `plot!` function returns the current plot, which Pluto renders. This is how we have been rendering plots so far. To save a plot, for example to include it in a report, we use the `savefig` function.

We have to pass the name of the file to `savefig`.

- Pass a file name ending with `.pdf` to save the plot as a PDF document. This will save the plot as vector graphics, which means that the plot will not be blurry even when zooming in a lot.
- Pass a file name ending with an image extension such as `.png` to save the plot as an image. Images can be easier to work with, but can look blurry when zoomed in.

In reports, prefer vector graphics... they look better. 😉

PDF

An example of saving a figure to PDF:

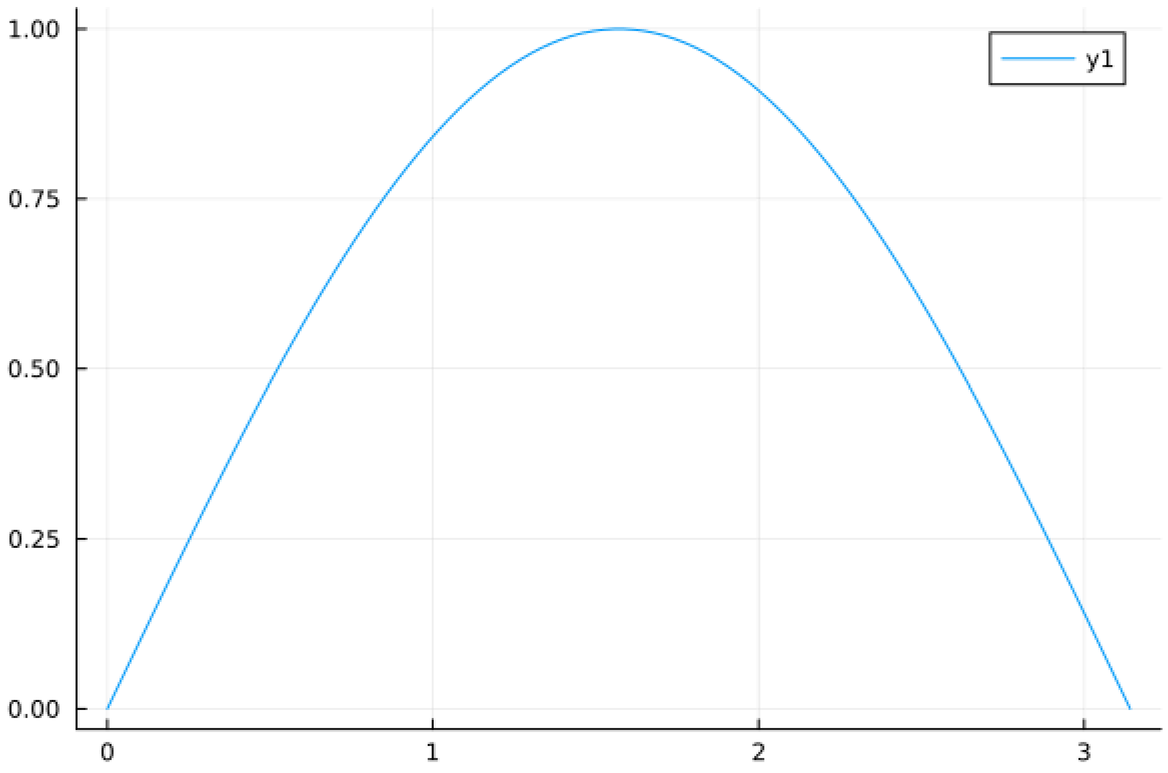
```
let
    xs = range(0, π; length=100)
    plot(xs, sin.(xs))
    savefig("sin_plot.pdf")
end;
```

Image

An example of saving a figure to PNG:

```
let
    xs = range(0, π; length=100)
    plot(xs, sin.(xs))
    savefig("sin_plot.png")
end;
```

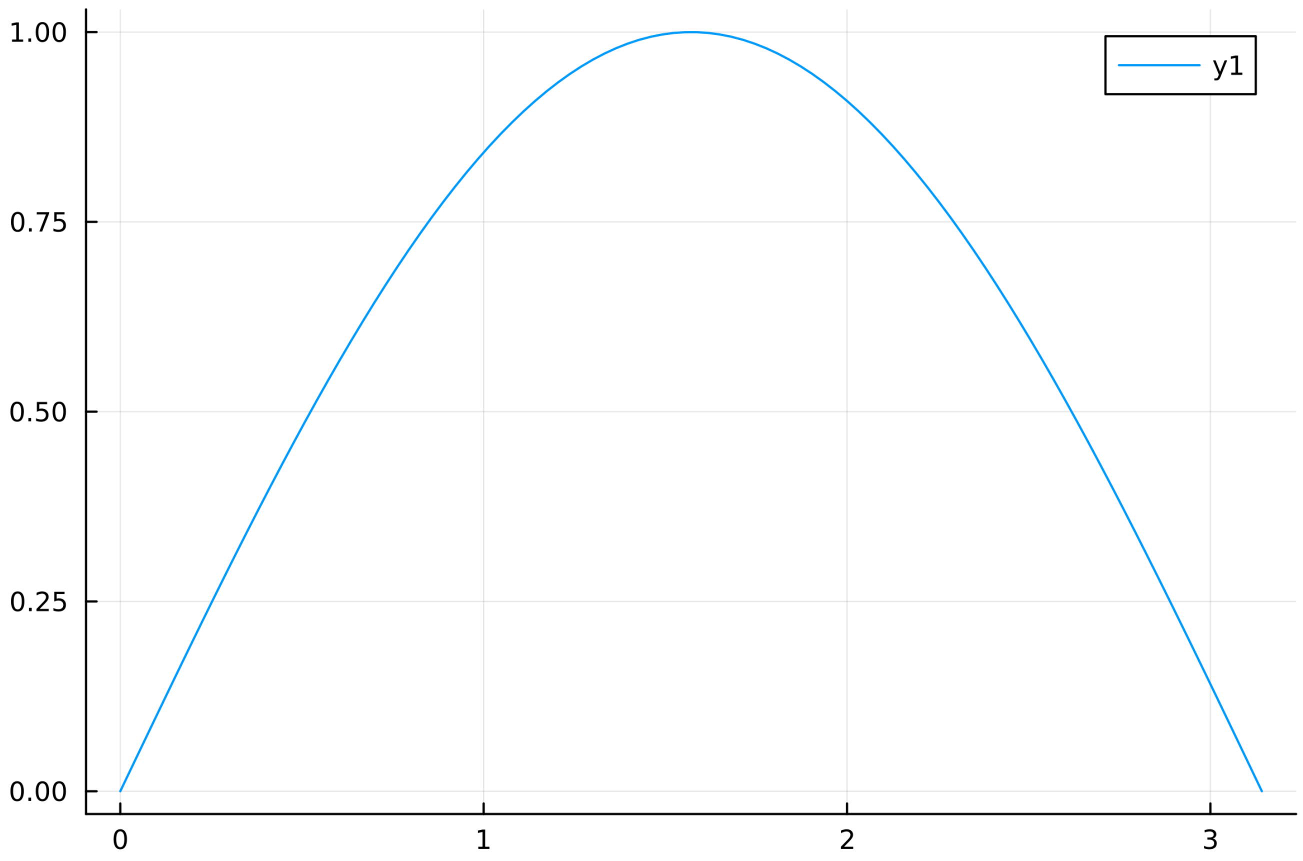
If you open the above image, you will notice that it looks blurry. For example it might look like this:



The number of pixels used by the image can be increased with the `dpi` attribute:

```julia
let
    xs = range(0, π; length=100)
    plot(xs, sin.(xs); dpi=400)
    savefig("sin_plot_high_dpi.png")
end;
```

Here is an example of the high DPI version:


1.00
0.75
0.50
0.25
0.00
0
1
2
3
y1