0.1 Basics of Mathematica

0.1.1 What is Mathematica?

Mathematica is an IDE for the Wolfram programming language. It is not a procedural programming language, where all commands must be written before they are run. On the contrary you can run/declare/write one command at a time and generate output as you go along.

Mathematica has many built in commands for different mathematical functions, plotting graphs and making other visuals. This immense library of pre-made options is what makes this ideal for data analysis in IAs.

You can start your free Mathematica trial ***here***.

0.1.2 Where to begin

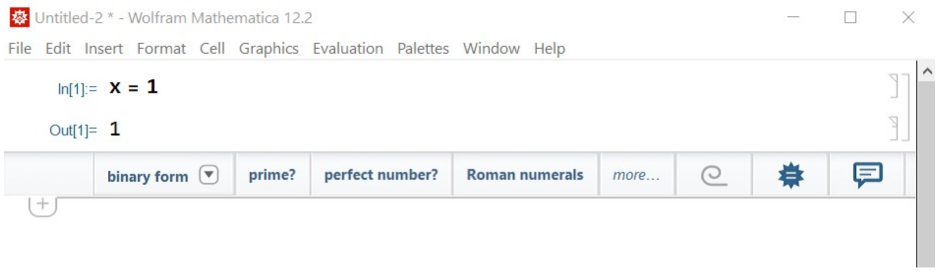
On a basic level, you can do two things. Declare variables and declare/run functions. To do either one of these things, you must open a blank Mathematica notebook, type your command/variable and hit shift enter.

0.1.2.1 Variables

Variables work in a similar way to variables in math. You can "declare" a variable and assign it a value.

You can assign a value to the variable, by using an equal sign symbol. A variable can hold any type of data (Text must be enclosed in quotation marks).

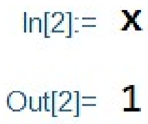
Here is an example of us declaring some variable called x and assigning it a value of 1:


Untitled-2 * - Wolfram Mathematica 12.2
File Edit Insert Format Cell Graphics Evaluation Palettes Window Help
binary form
prime?
perfect number?
Roman numerals
more...

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

Out[1]= 1
```

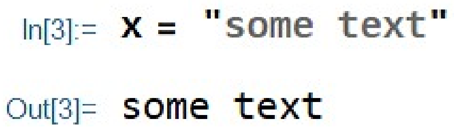
Figure 1: After typing $x = 1$, remember to hit shift enter for your command to actually execute. The program immediately gives an output of "1" as that is the value you just stored

If this variable is now entered as a command, it will give the output of 1.
The value of x can be replaced with quite literally anything else as well. Let's replace it with some text.



```
In[2]:= x

Out[2]= 1
```

(a) Shift entering x now gives an output of 1



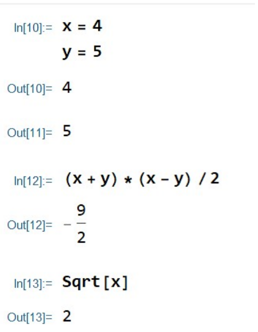
```
In[3]:= x = "some text"

Out[3]= some text
```

(b) We have replaced the value of x with text

Figure 2

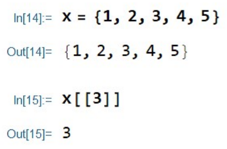
Mathematica also recognizes basic operations (both with numbers and variables). There are also commands for more complex functions (E.G. Square root).

It is also possible to have lists of things (arrays). These are enclosed in curly brackets. If you want to access a specific part of an array (and not the whole thing), type the order of the item you would like to access enclosed in double square brackets after calling on the array (indexing starts at 1).
You can also have nested arrays (an array within an array). Arrays do not need to contain only one type of data, they can have anything anywhere!



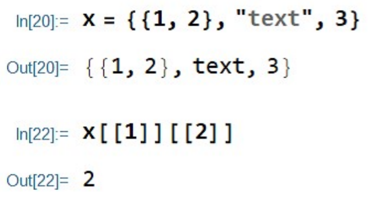
```
In[10]:= x = 4
         y = 5

Out[10]= 4

Out[11]= 5

In[12]:= (x + y) * (x - y) / 2

Out[12]= -9/2

In[13]:= Sqrt[x]

Out[13]= 2
```

Figure 3: In this example, x is passed as an argument to the Sqrt function. (Mathematica uses square brackets for functions



```
In[14]:= x = {1, 2, 3, 4, 5}

Out[14]= {1, 2, 3, 4, 5}

In[15]:= x[[3]]

Out[15]= 3
```

(a) We call on the 3d element of the array stored in x.



```
In[20]:= x = {{1, 2}, "text", 3}

Out[20]= {{1, 2}, text, 3}

In[22]:= x[[1]][[2]]

Out[22]= 2
```

(b) This initializes an array with 3 elements. The first element is an array of 2 elements. We then call on the second element of the first element of the array

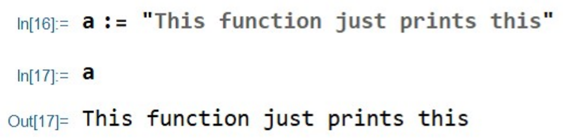
Figure 4

0.1.2.2 Functions

It is also possible to declare functions in Mathematica. These do not necessarily store data, but rather a rule of what should be done when they are called on. Functions must be assigned a name.

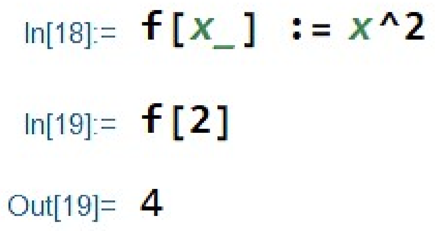
Unlike variables, you define a function with := and not =.

If you would like your function to take in arguments, place them in square brackets (followed by an underscore) right after the function name.



```
In[16]:= a := "This function just prints this"

In[17]:= a

Out[17]= This function just prints this
```

(a) This function just prints text when called.



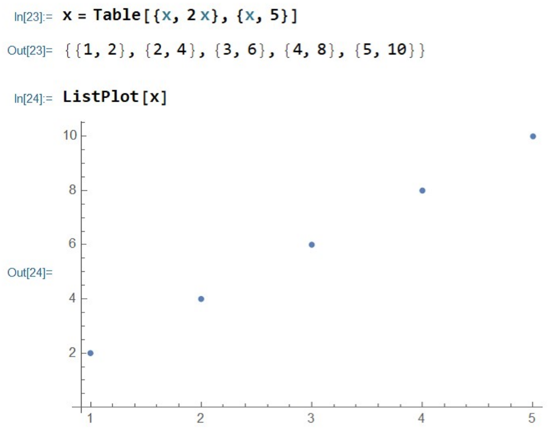
```
In[18]:= f[x_] := x^2

In[19]:= f[2]

Out[19]= 4
```

(b) This function squares the argument when called. It will not run if no argument is provided

Figure 5

0.1.2.3 Useful Pre-Set Functions

With Mathematica, it is possible to do virtually anything! For Example,
plot a list of points! For this, you must have an array of points (array of arrays). Pass it as an argument to the ListPlot function to get an output! An easy way to generate a list of points is to use the Table command.

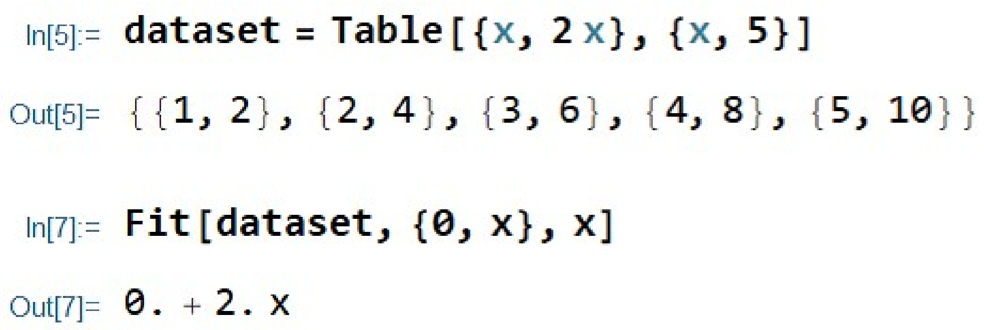


```
In[23]:= x = Table[{x, 2 x}, {x, 5}]

Out[23]= {{1, 2}, {2, 4}, {3, 6}, {4, 8}, {5, 10}}

In[24]:= ListPlot[x]

Out[24]=
```

Figure 6: With the table command I tell Mathematica to make a list of elements of the form {x, 2x}, for values of x from 1 to 5.

You can right click on this graph to save it as an image or pdf in the format of your choice. Note that graphs and images can also be stored in variables.

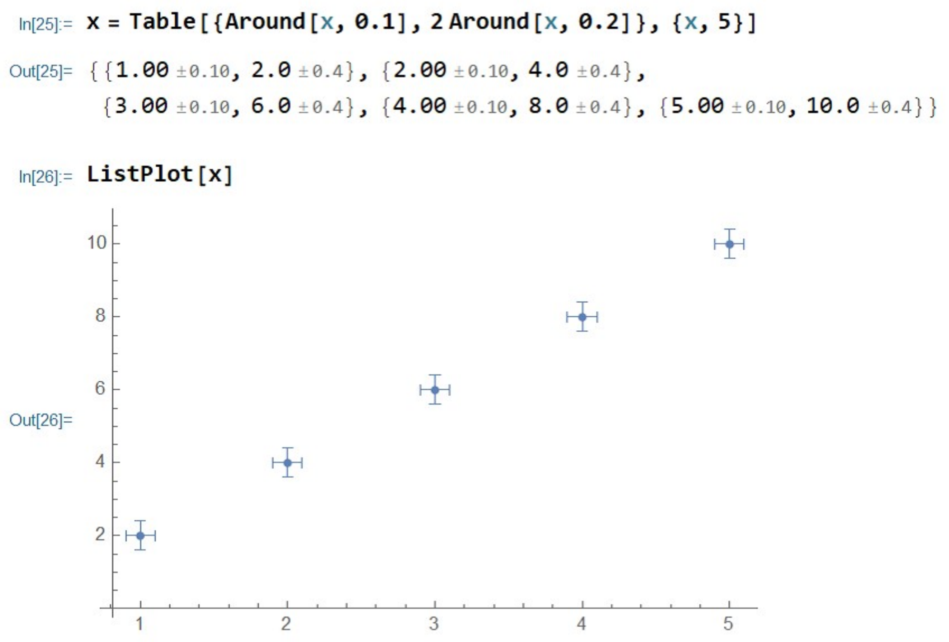
There is also the fit function, that finds a line/curve of best fit for a dataset.



```
In[5]:= dataset = Table[{x, 2 x}, {x, 5}]

Out[5]= {{1, 2}, {2, 4}, {3, 6}, {4, 8}, {5, 10}}

In[7]:= Fit[dataset, {0, x}, x]

Out[7]= 0. + 2. x
```

Figure 7: For data stored in variable "dataset", Mathematica here finds a fit with terms of x and some constant.

There is also a data type for measured numbers with uncertainty! It is the "Around" type, which takes two numbers as an argument. The first number is the actual value, and the second is its uncertainty! When operations are done on a "measured" value, Mathematica automatically propagates uncertainty! A list of points, where coordinates are "Around" type numbers, can be plotted with error bars using the same ListPlot method!

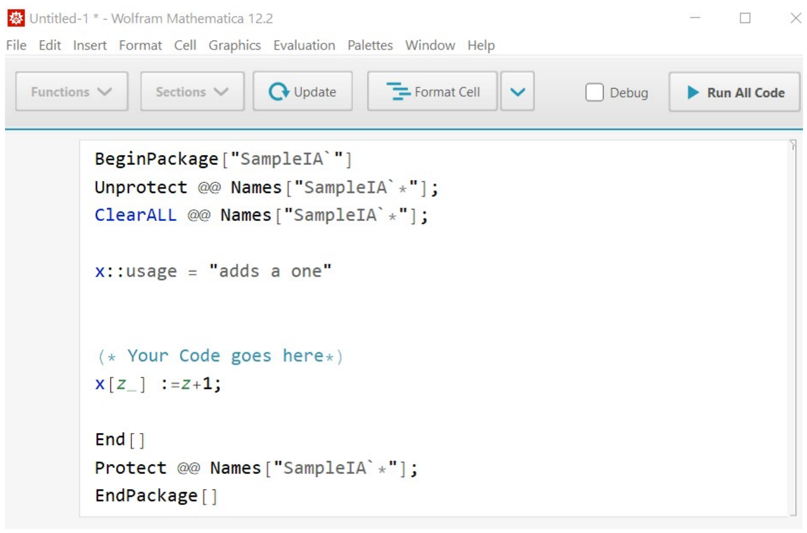


```
In[25]:= x = Table[{Around[x, 0.1], 2 Around[x, 0.2]}, {x, 5}]

Out[25]= {{1.00 ±0.10, 2.0 ±0.4}, {2.00 ±0.10, 4.0 ±0.4},
          {3.00 ±0.10, 6.0 ±0.4}, {4.00 ±0.10, 8.0 ±0.4}, {5.00 ±0.10, 10.0 ±0.4}}

In[26]:= ListPlot[x]

Out[26]=
```

Figure 8: Note that the uncertainty for the y coordinates is 0.4 and not 0.2, because the y coordinate is x multiplied by 2.

0.1.2.4 Packages

So far we have been initializing all of our variables in a notebook type file. If you have complex data analysis, this can become very cumbersome and difficult to keep track of. For this purpose, it is possible to write a mathematica package, which would contain code for functions. It could

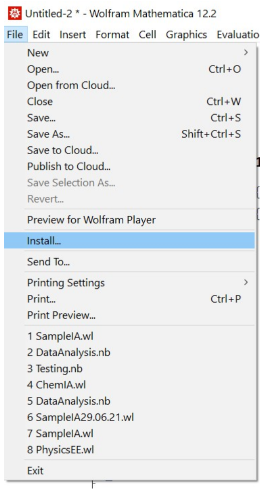

Untitled-1 * - Wolfram Mathematica 12.2
File Edit Insert Format Cell Graphics Evaluation Palettes Window Help
Functions
Sections
Update
Format Cell
Debug
Run All Code

```
BeginPackage["SampleIA`"]
Unprotect @@ Names["SampleIA`*"];
ClearALL @@ Names["SampleIA`*"];

x::usage = "adds a one"


(* Your Code goes here*)
x[z_] :=z+1;

End[]
Protect @@ Names["SampleIA`*"];
EndPackage[]
```

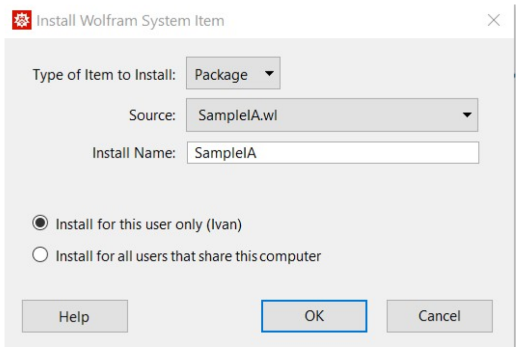
Figure 9: The function's name declared inside the file, must be the same as the file name. Note that all declared functions require a "usage statement", which is meant to be a description of the function's purpose.

then be loaded into a notebook file, making those functions available.

Mathematica packages will follow the following syntax:
The package must then be installed to your local mathematica repository.



(a) Hit install



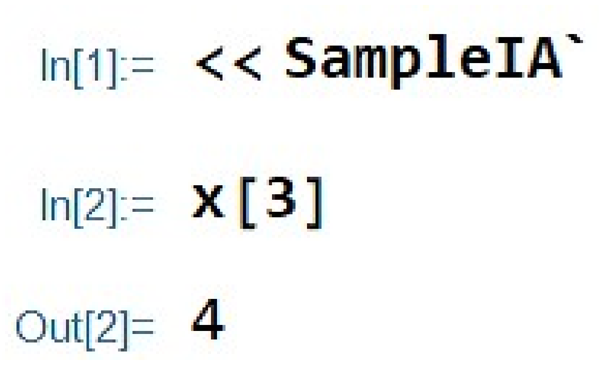
Type of Item to Install: Package
Source: SampleIA.wl
Install Name: SampleIA

(● Install for this user only (Ivan)) ○ Install for all users that share this computer

(b) Make sure the name you enter for the package is identical to the file name

Figure 10

You can now load in your function by typing its name preceded by <<and followed by ` symbols.



```
In[1]:= << SampleIA`

In[2]:= x[3]

Out[2]= 4
```

Figure 11: Now that we have loaded in our package, we can use the functions it contains (the x function from earlier).

0.1.3 What next?

You can find out a lot more about different built in functions from the ***Wolfram Language & System Official Documentation Center***.

You can also look at the package for data analysis for the sample IA in ***NFFC***.