Getting Started With Mathematica

What you will make

In this resource you will be introduced to Mathematica and the Wolfram language, which are available for free on the Raspberry Pi.

Mathematica is a computational programming tool used in science, maths, computing and engineering. It was first released in 1988. It is proprietary software that you can use for free on the Raspberry Pi and comes bundled for free with Raspbian. Mathematica is generally used for coding projects at university level and above.

What you will learn

By following the Getting Started with Mathematica worksheet, you will learn:

- How to launch the Mathematica notebook and run commands
- How to use variables
- How to use symbolic values for irrational numbers like Pi
- How to use lists, ranges, tables and loops
- How to use and and search for built-in functions
- How to save a Wolfram script and run it from the command line
- How to perform list operations
- How to use Matrices
- How to plot in 3D
- How to access GPIO pins and the camera module

This resource covers elements from the following strands of the Raspberry Pi Digital Making Curriculum (https://www.raspberrypi.org/curriculum/):

- Use basic programming constructs to create simple programs (https://www.raspberrypi.org/curriculum/programming/creator)

What you will need

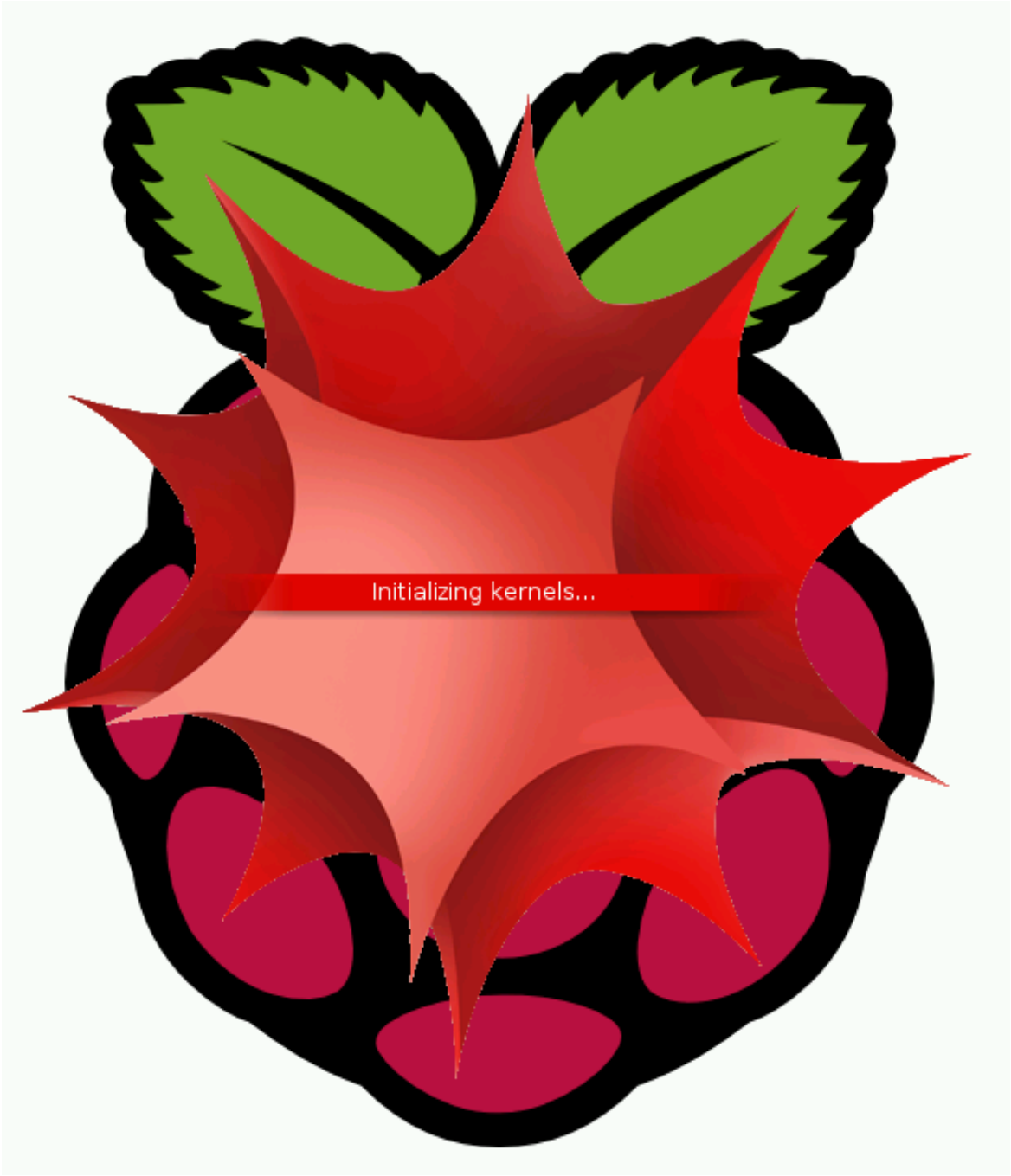
- A Raspberry Pi computer

Getting started with Mathematica

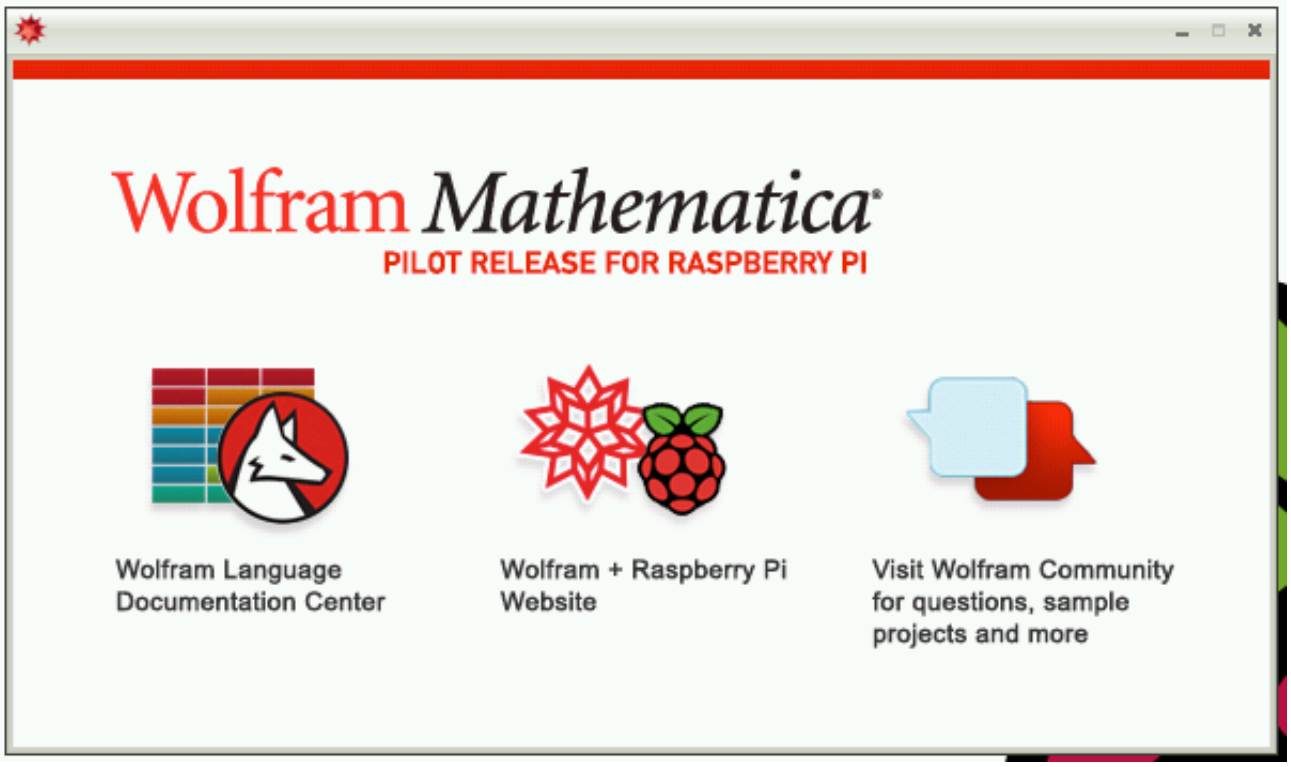
Mathematica is a computational programming tool used in science, maths, computing and engineering, first released in 1988. It is proprietary software that you can use for free on the Raspberry Pi and has been bundled with Raspbian and NOOBS since late 2013.

Starting Mathematica

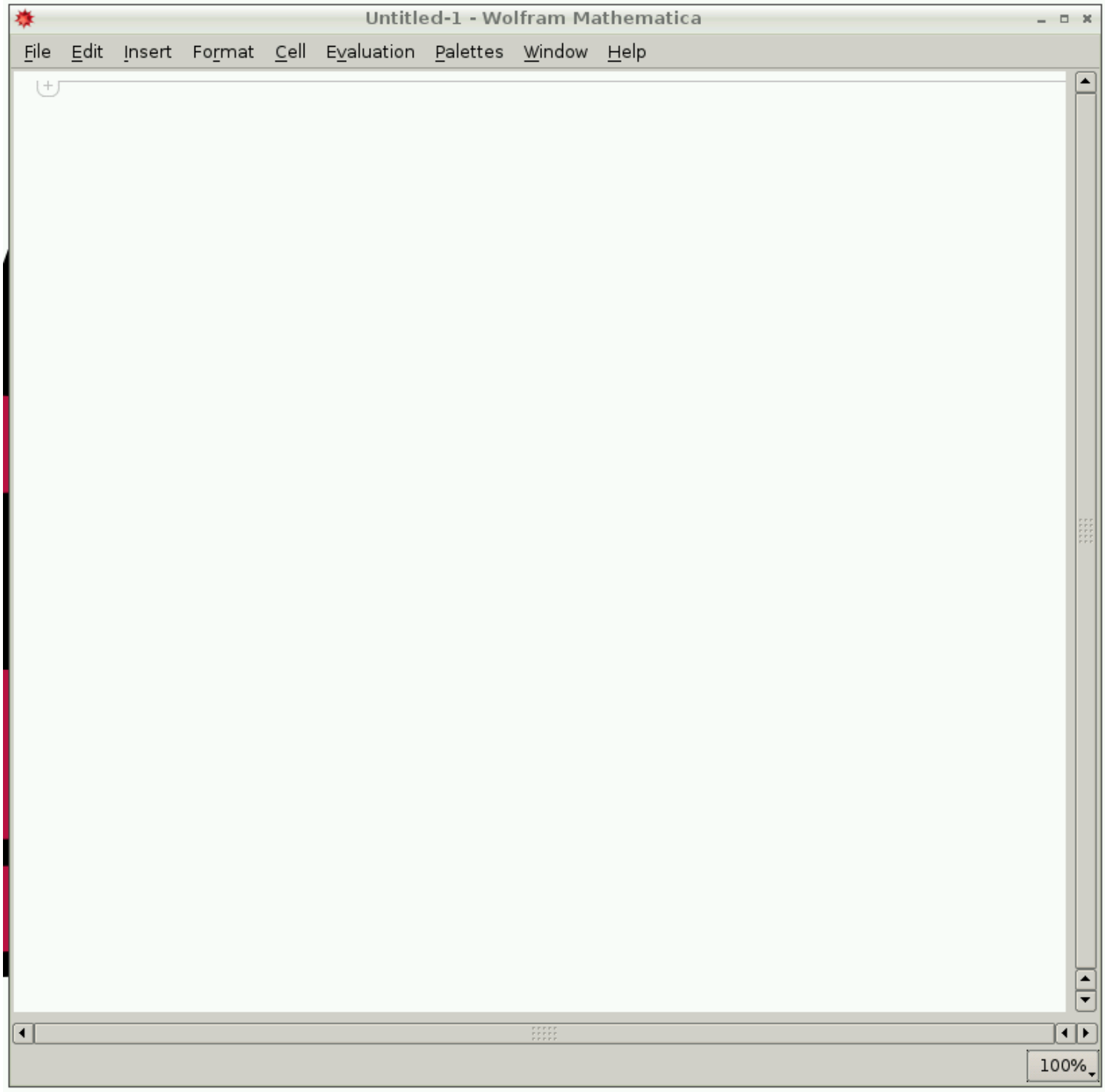
Double-click the `Mathematica` icon on the Desktop or open it from the applications menu to start. You'll see a splash screen with the red Mathematica logo while the program loads:



Once loaded, you'll see two windows. These are the Wolfram information dialogue:



and the Mathematica notebook:



The Wolfram information dialogue shows web links to:

- Wolfram Language Documentation Center (http://reference.wolfram.com/language/index.html)
- Wolfram + Raspberry Pi Website (http://www.wolfram.com/raspberry-pi)
- Wolfram Community (http://community.wolfram.com/content?curTag=raspberry%20pi)

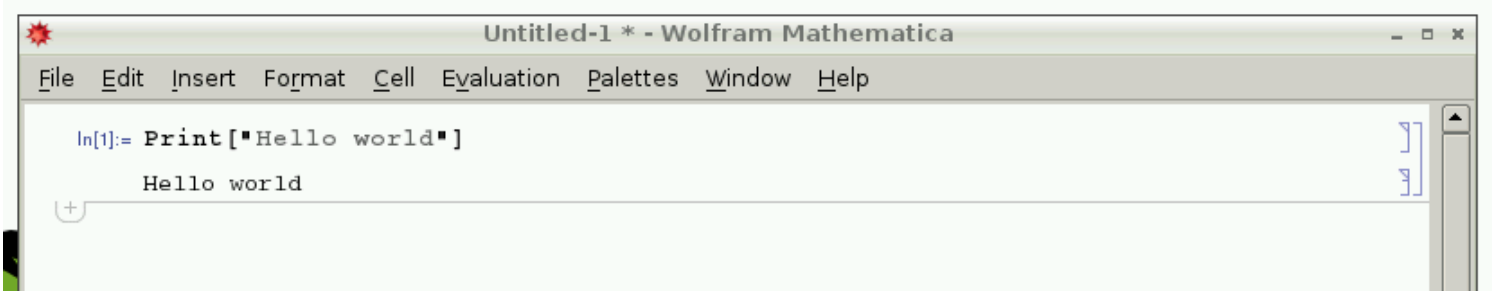
These links will open in the web browser on the Raspberry Pi, provided you're connected to the internet.

Programming in Mathematica

Click inside the notebook window and enter:

```
Print["Hello world"]
```

Press **`Shift + Enter`**; it will run the command and print “Hello world” to the screen like so:



You can perform mathematical calculations, for example:

```
In[2]:= 2 + 2

Out[2]= 4

In[3]:= 16254 / 32

Out[3]= 8127 / 16

In[4]:= 1024 * 32

Out[4]= 32768
```

Notebook editing

You can revisit a previously entered command by clicking it or moving the edit cursor with the keyboard. You can then delete, edit, or add something and press **`Shift + Enter`** to execute the new command in its place.

You can save a notebook and come back to it later, send it to a friend, post it online, or even hand it in as your homework! Just go to **`File > Save As`** in the notebook window.

When you open up a saved notebook, all the previous entries will be shown, including all the inputs and outputs. You can execute each cell again with **`Shift + Enter`**, or all at once by selecting **`Evaluation > Evaluate Notebook`** from the menu.

Variables

You can store the results of calculations in variables:

```
radius = 5;
diameter = 2 * radius;
circumference = 2 * Pi * radius;
area = Pi * radius^2;
```

Note the semicolon at the end of each line suppresses the output being printed.

Symbolic values

Note the use of the built-in symbol **`Pi`** which contains a symbolic value of Pi. This means that if you pass it into an equation the reference to the true value of *Pi* is preserved, not converted to decimal and rounded:

```
In[19]:= Pi

Out[19]: π

In[20]:= tau = 2 * Pi

Out[20]: 2 π
```

To get the decimal representation of a symbolic value, use the **`N`** function:

```
In[5]:= N[Pi]

Out[5]: 3.14159
```

The default number of significant figures given is 6, but more can be given by specifying the number in the second argument:

```
In[6]:= N[Pi, 10]

Out[6]: 3.141592654
```

Note this is the number of figures, not decimal places; so the 3 is included in the count, leaving 9 decimal places.

Lists

You can store collections of data in a list:

```
nums = {1, 2, 3, 5, 8}
people = {"Alice", "Bob", "Charlotte", "David"}
```

Range

The `Range` function can be used to produce a list of numbers:

```
Range[5] (*The numbers 1 to 5*)
Range[2, 5] (*The numbers 2 to 5*)
Range[2, 5, 2] (*The numbers 2 to 5, in steps of 2*)
```

Table

The `Table` function is a method of generating the values of a list with a function:

```
Table[i ^ 2, {i, 10}] (*Squares of the numbers 1 to 10*)
Table[i ^ 2, {i, 5, 10}] (*Squares of the numbers 5 to 10*)
Table[i ^ 2, {i, nums}] (*Squares of the items in the list nums*)
```

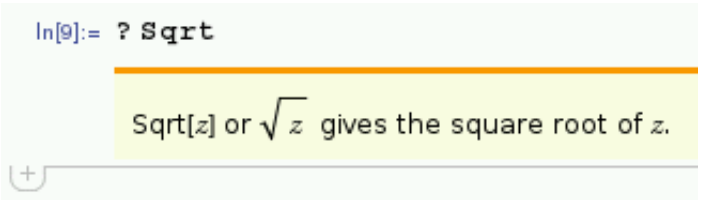
Looping

You can run a loop a number of times, or over the items in a list, with `Do`:

```
Do[Print["Hello"], {10}] (*Print "Hello" 10 times*)
Do[Print[i], {i, 5}] (*Print the numbers 1 to 5*)
Do[Print[i], {i, 3, 5}] (*Print the numbers 3 to 5*)
Do[Print[i], {i, 1, 5, 2}] (*Print the numbers 1 to 5, in steps of 2*)
Do[Print[i ^ 2], {i, nums}] (*Print the square of each item in the list nums*)
```

Function help

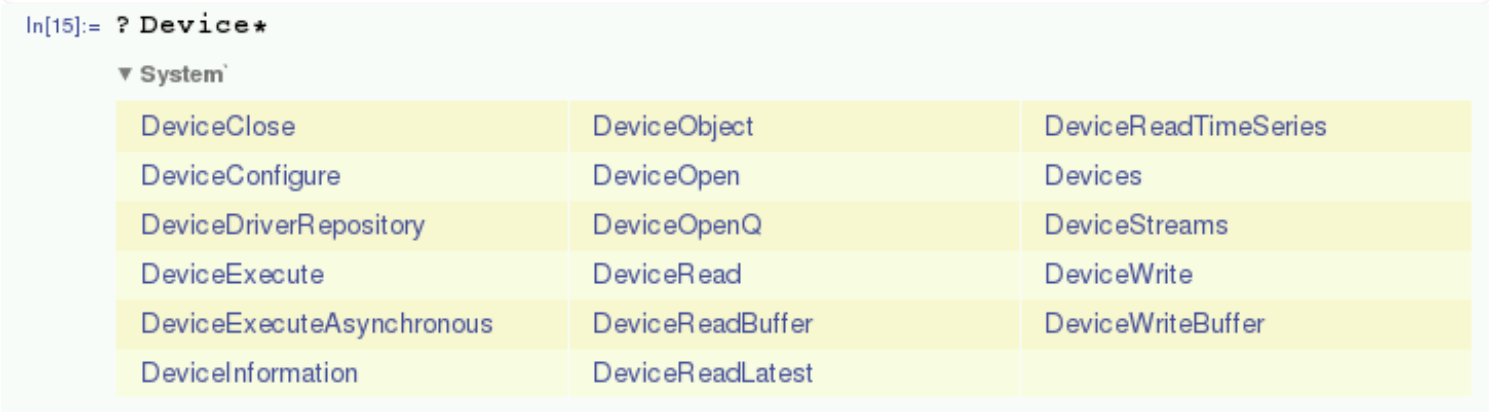
You can get usage help for a function by preceding the function name with a question mark (`?`) and pressing `Shift + Enter`:



Function search

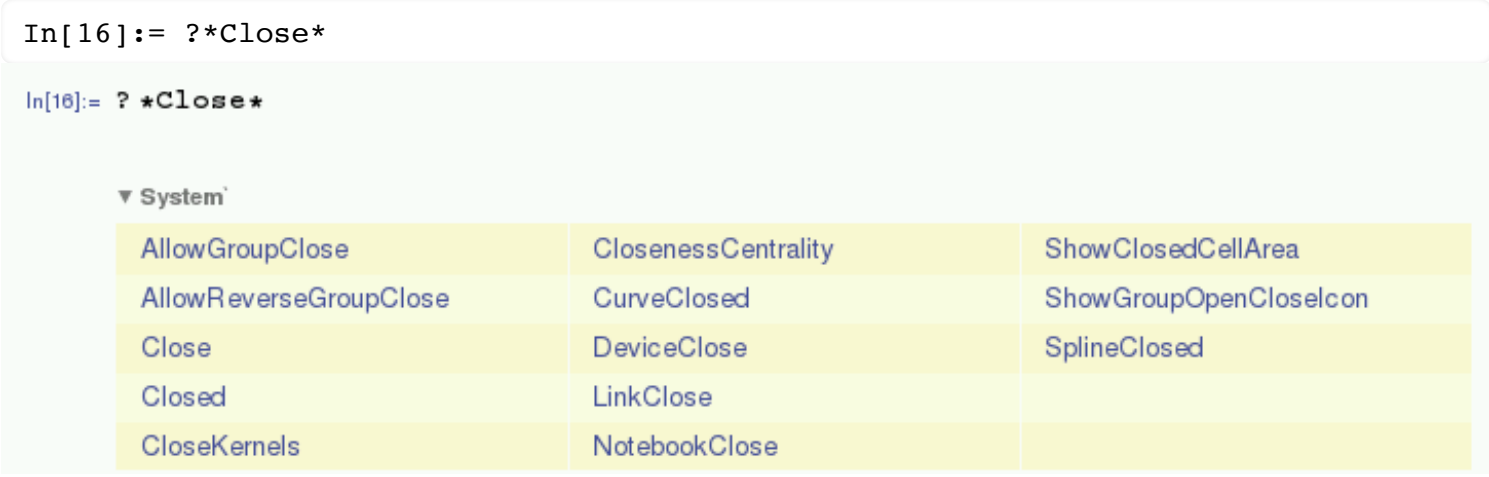
You can also search for functions by entering part of the function name to find matches. Just start with a (`?`) and add an asterisk (`*`) on the end as a wildcard:

```
In[15]:= ?Device*
```



You can use multiple wildcards:

```
In[16]:= ?*Close*
```



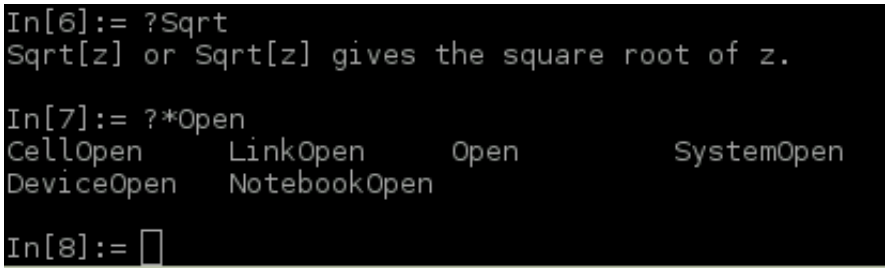
Comments

As seen in earlier examples, you can leave comments (notes that are ignored in the program) in scripts by using brackets (`(` & `)`) and asterisks (`*`):

```
Print["Hello"] (*Print "Hello" to the screen*)
```

Wolfram command line access

You can also access the Wolfram language from the command line by entering **`wolfram`** in the terminal, or double-clicking the Wolfram Desktop icon. This will give a text-only (non-graphical) programming environment with the `In[x]` / `Out[x]` style interface, but without interactive notebook functionality. The Mathematica functions will still work as expected:



You'll find the command line interface faster to use due to the lack of graphical processing required to run the notebook, however it lacks the graphical interface's interactivity and pretty printing.

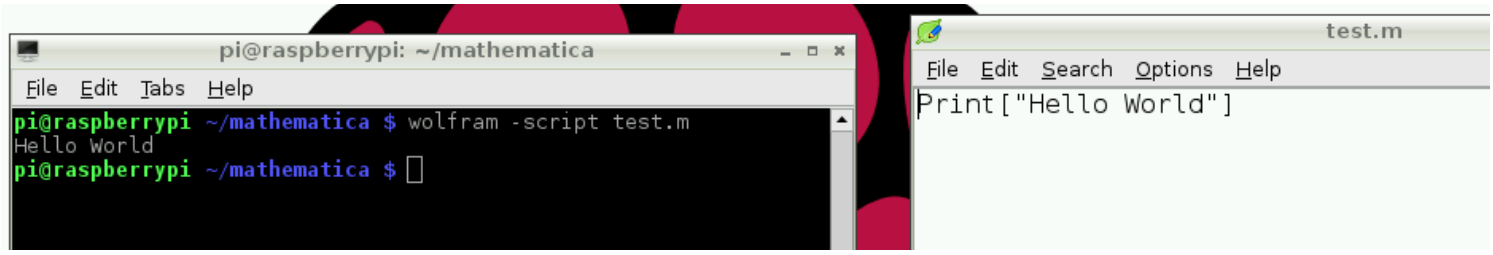
To exit, press `Ctrl + D`.

Running scripts with Wolfram

You can write a program, save it as a normal file (usually with a **`.m`** or **`.wl`** file extension), and execute the script from the command line by adding the **`-script`** flag.

To run **`test.m`**:

```
wolfram -script test.m
```



List operations

You can apply an operation or function to all items in a list:

```
In[21]:= 2 * {1, 2, 3, 4, 5}

Out[21]: {2, 4, 6, 8, 10}

In[22]:= {1, 2, 3, 4, 5} ^ 2

Out[22]: {1, 4, 9, 16, 25}

In[23]:= Sqrt[{1, 2, 3, 4, 5}]

Out[23]: {1, Sqrt[2], Sqrt[3], 2, Sqrt[5]}
```

Note that in the last example the square roots of `1` and `4` were given exactly, as they yield integer value; however the square roots of `2`, `3` and `5`, which are irrational, are given symbolically.

Matrices

One of the most useful additional components of a mathematical programming language is the ability to do matrix (https://en.wikipedia.org/wiki/Matrix_(mathematics)) operations. Of course, Mathematica has these available.

To create a matrix first enter the values as a list of lists, making sure the dimensions are rectangular, i.e. `n x m` where `n` and `m` are integers:

```
m = {{1, 2}, {3, 4}, {5, 6}};
```

You can view this list as a matrix by typing:

```
m // MatrixForm
```

```
In[21]:= m = {{1, 2}, {3, 4}, {5, 6}}

Out[21]= {{1, 2}, {3, 4}, {5, 6}}

In[22]:= m // MatrixForm

Out[22]//MatrixForm=
        1  2
        3  4
        5  6
```

You can perform matrix operations such as dot product (https://en.wikipedia.org/wiki/Dot_product):

```
m = {{1, 2}, {3, 4}, {5, 6}};
m2 = {{10, 20, 30}, {40, 50, 60}};
m . m2 // MatrixForm
```

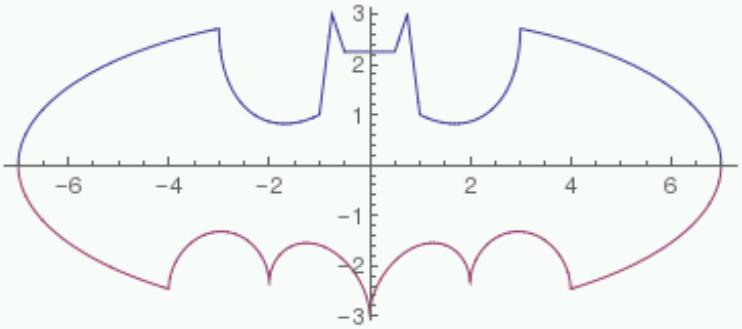
```
In[7]:= m = {{1, 2}, {3, 4}, {5, 6}};
        m2 = {{10, 20, 30}, {40, 50, 60}};
        m . m2 // MatrixForm

Out[9]//MatrixForm=
        90  120  150
       190  260  330
       290  400  510
```

Plotting

You can plot interesting things using Mathematica:

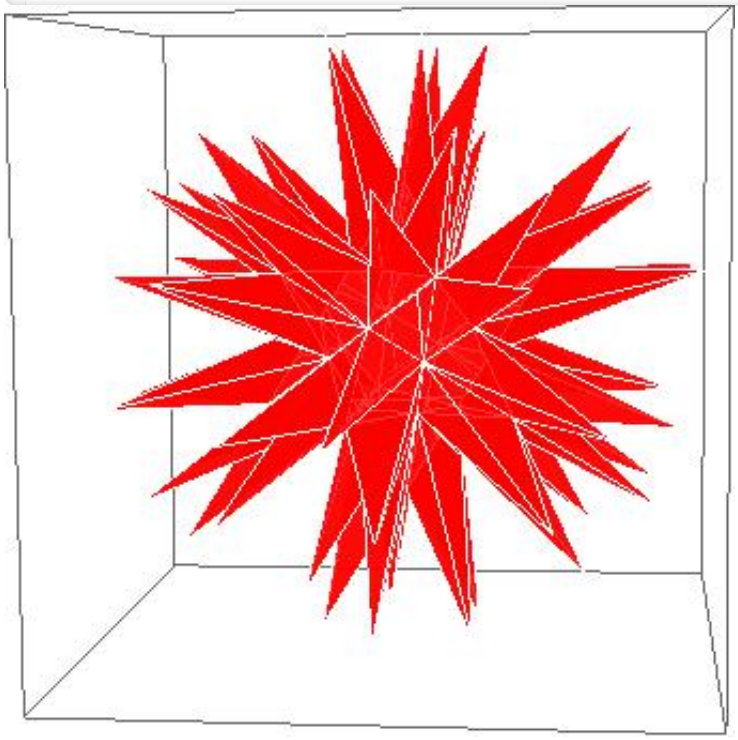
```
In[5]:= Plot[
 {With[{w = 3 * Sqrt[1 - (x / 7) ^ 2],
    l = (6 / 7) * Sqrt[10] + (3 + x) / 2 - (3 / 7) * Sqrt[10] * Sqrt[4 - (x + 1) ^ 2],
    h =
     (1 / 2) * (3 * (Abs[x - 1 / 2] + Abs[x + 1 / 2] + 6) -
        11 * (Abs[x - 3 / 4] + Abs[x + 3 / 4])),
    r = (6 / 7) * Sqrt[10] + (3 - x) / 2 - (3 / 7) * Sqrt[10] * Sqrt[4 - (x - 1) ^ 2]},
   w + (l - w) * UnitStep[x + 3] + (h - l) * UnitStep[x + 1] + (r - h) * UnitStep[x - 1] +
    (w - r) * UnitStep[x - 3]],
  (1 / 2) * (3 * Sqrt[1 - (x / 7) ^ 2] + Sqrt[1 - (Abs[Abs[x] - 2] - 1) ^ 2] +
     Abs[x / 2] - ((3 * Sqrt[33] - 7) / 112) * x ^ 2 - 3) *
   ((x + 4) / Abs[x + 4] - (x - 4) / Abs[x - 4]) - 3 * Sqrt[1 - (x / 7) ^ 2]},
 {x, -7, 7}, AspectRatio → Automatic]
```

Out[5]=



For example, plot an echidnahedron (http://mathworld.wolfram.com/Echidnahedron.html) with the following command:

```
Graphics3D[{Opacity[.8], Glow[RGBColor[1,0,0]], EdgeForm[White], Lighting ->
None, PolyhedronData["Echidnahedron", "Faces"]}]
```



GPIO

You can access the GPIO pins from Mathematica using the **`DeviceWrite`** and **`DeviceRead`** functions.

To access the GPIO pins you'll need to be running as root. To do this, run **`sudo wolfram`** from the terminal (this will run the command line **`wolfram`** environment), or **`sudo mathematica &`** to run the Mathematica notebook as root.

The following command turns on GPIO pin 14 (using BCM pin numbering):

```
DeviceWrite["GPIO", 14 -> 1]
```

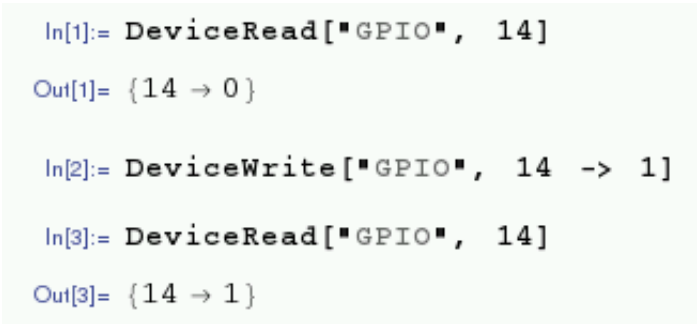
The following command turns pin 14 off:

```
DeviceWrite["GPIO", 14 -> 0]
```

You can also read the status of a GPIO input device (to check if a button is pressed, for example) with **`DeviceRead`**, in a similar way:

```
button = DeviceRead["GPIO", 14]
```

The variable **`button`** should now contain `0` for off or `1` for on.



Read more about GPIO in general on the GPIO usage page (https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/README.md).

Camera

You can take pictures with the camera using the **DeviceRead** function. First, attach your camera as directed in the camera setup guide (http://www.raspberrypi.org/help/camera-module-setup/).

To take a still picture with the camera, type the following command:

```
img = DeviceRead["RaspiCam"]
```

Then to save the image as a file, use **Export** and supply the save path and the variable containing the image:

```
Export["/home/pi/img.jpg", img]
```

Published by the Raspberry Pi Foundation – www.raspberrypi.org