A Mathematica Workshop

For Mathematicians

Sam Fearn

June 21st, 2019

Outline

What Is Mathematica?

- Mathematica is an example of a Computer Algebra System (CAS) developed by Wolfram.

What Is Mathematica?

- Mathematica is an example of a Computer Algebra System (CAS) developed by Wolfram.
- Unlike a simple calculator application, Mathematica can work symbolically.

```
 In[1]:=        D[Sin[x],x]
Out[1]:=        Cos[x]
```

What Is Mathematica?

- Mathematica is an example of a Computer Algebra System (CAS) developed by Wolfram.
- Unlike a simple calculator application, Mathematica can work symbolically.

```
 In[1]:=        D[Sin[x],x]
Out[1]:=        Cos[x]
```

- The combination of symbolic manipulation and a large number of built-in mathematical functions means that a large number of problems can be solved much easier with Mathematica than a general purpose programming language.

What Is Mathematica?

- Mathematica is an example of a Computer Algebra System (CAS) developed by Wolfram.
- Unlike a simple calculator application, Mathematica can work symbolically.

```
 In[1]:=        D[Sin[x],x]
Out[1]:=        Cos[x]
```

- The combination of symbolic manipulation and a large number of built-in mathematical functions means that a large number of problems can be solved much easier with Mathematica than a general purpose programming language.
- There are other CAS which work very similarly, though the exact syntax for commands may change. Other popular CAS include Maple, MATLAB and the open source SageMath. Also Cadabra, which is developed in Durham by Kasper Peeters.

A Notebook Interface

- You may already be familiar with Wolfram|Alpha, which understands many of the same commands as Mathematica – they are both built on the Wolfram Language.

A Notebook Interface

- You may already be familiar with Wolfram|Alpha, which understands many of the same commands as Mathematica – they are both built on the Wolfram Language.
- One way that Mathematica differs from Wolfram|Alpha, is that (by default) Mathematica uses a 'Notebook' interface – similar to Jupyter Notebooks or R Markdown documents/ R Notebooks.

A Notebook Interface

- You may already be familiar with Wolfram|Alpha, which understands many of the same commands as Mathematica – they are both built on the Wolfram Language.
- One way that Mathematica differs from Wolfram|Alpha, is that (by default) Mathematica uses a 'Notebook' interface – similar to Jupyter Notebooks or R Markdown documents/ R Notebooks.
- A Mathematica notebook may contain headings and subheadings, formatted text and even other languages (Python, JavaScript) alongside Mathematica code cells.

A Notebook Interface

- You may already be familiar with Wolfram|Alpha, which understands many of the same commands as Mathematica – they are both built on the Wolfram Language.
- One way that Mathematica differs from Wolfram|Alpha, is that (by default) Mathematica uses a 'Notebook' interface – similar to Jupyter Notebooks or R Markdown documents/ R Notebooks.
- A Mathematica notebook may contain headings and subheadings, formatted text and even other languages (Python, JavaScript) alongside Mathematica code cells.
- Mathematica cells can even be dynamic, changing automatically based on the execution of other cells, or have controls attached to them allowing dynamic modification.

Working In A Notebook

- In a notebook, content is arranged into *cells*.

Working In A Notebook

- In a notebook, content is arranged into *cells*.
- Cells which contain Mathematica code (as opposed to text) can be executed in any order you like, the output is then placed in an 'output' cell below the 'input' cell.

Working In A Notebook

- In a notebook, content is arranged into *cells*.
- Cells which contain Mathematica code (as opposed to text) can be executed in any order you like, the output is then placed in an 'output' cell below the 'input' cell.
- Cells are evaluated by pressing **Shift-Enter**.

Working In A Notebook

- In a notebook, content is arranged into *cells*.
- Cells which contain Mathematica code (as opposed to text) can be executed in any order you like, the output is then placed in an 'output' cell below the 'input' cell.
- Cells are evaluated by pressing **Shift-Enter**.
- The output from one calculation can be used as part of another calculation.

Working In A Notebook

- In a notebook, content is arranged into *cells*.
- Cells which contain Mathematica code (as opposed to text) can be executed in any order you like, the output is then placed in an 'output' cell below the 'input' cell.
- Cells are evaluated by pressing **Shift-Enter**.
- The output from one calculation can be used as part of another calculation.
- As in other programming languages, we can create *variables* and define our own *functions*.

Working In A Notebook

- In a notebook, content is arranged into *cells*.
- Cells which contain Mathematica code (as opposed to text) can be executed in any order you like, the output is then placed in an 'output' cell below the 'input' cell.
- Cells are evaluated by pressing **Shift-Enter**.
- The output from one calculation can be used as part of another calculation.
- As in other programming languages, we can create *variables* and define our own *functions*.
- There are also a large number of built in functions covering a wide range of mathematical (and non-mathematical) topics.

Working In A Notebook

- In a notebook, content is arranged into *cells*.
- Cells which contain Mathematica code (as opposed to text) can be executed in any order you like, the output is then placed in an 'output' cell below the 'input' cell.
- Cells are evaluated by pressing **Shift-Enter**.
- The output from one calculation can be used as part of another calculation.
- As in other programming languages, we can create *variables* and define our own *functions*.
- There are also a large number of built in functions covering a wide range of mathematical (and non-mathematical) topics.
- Using a notebook means that you can change the definition of a function – or add a new function – without having to rerun all the rest of your code. This can be very helpful if calculations take a long time to run, or if you want to store lots of results in a file you're working on.

Some Basic Functions

- The convention is that built in functions in Mathematica start with a capital letter.

Some Basic Functions

- The convention is that built in functions in Mathematica start with a capital letter.
- Square brackets are then used to give arguments to a function, with arguments separated by commas.

Some Basic Functions

- The convention is that built in functions in Mathematica start with a capital letter.
- Square brackets are then used to give arguments to a function, with arguments separated by commas.

```
 In[1]:=  Binomial[15,4]
Out[1]:=  1365
```

Some Basic Functions

- The convention is that built in functions in Mathematica start with a capital letter.
- Square brackets are then used to give arguments to a function, with arguments separated by commas.

```
 In[1]:=  Binomial[15,4]
Out[1]:=  1365
```

```
 In[2]:=  Simplify[(2x - 4)/(3x + 3) - (x - 1)/(x + 1)]
Out[2]:=  -1/3
```

Some Basic Functions

- The convention is that built in functions in Mathematica start with a capital letter.
- Square brackets are then used to give arguments to a function, with arguments separated by commas.

```
 In[1]:=  Binomial[15,4]
Out[1]:=  1365
```

```
 In[2]:=  Simplify[(2x - 4)/(3x + 3) - (x - 1)/(x + 1)]
Out[2]:=  -1/3
```

```
 In[3]:=  Coefficient[(x + 3)^7, x, 4]
Out[3]:=  945
```

Lists

```
 In[1]:=  Divisors[24]
Out[1]:=  {1, 2, 3, 4, 6, 8, 12, 24}
```

Lists

```
 In[1]:=  Divisors[24]
Out[1]:=  {1, 2, 3, 4, 6, 8, 12, 24}
```

Lists in Mathematica are denoted using curly braces. Some functions like Divisors[n] return lists.

Lists

```
 In[1]:=  Divisors[24]
Out[1]:=  {1, 2, 3, 4, 6, 8, 12, 24}
```

Lists in Mathematica are denoted using curly braces. Some functions like Divisors[n] return lists. Other functions may have lists for their arguments.

Lists

```
 In[1]:=  Divisors[24]
Out[1]:=  {1, 2, 3, 4, 6, 8, 12, 24}
```

Lists in Mathematica are denoted using curly braces. Some functions like Divisors[n] return lists. Other functions may have lists for their arguments.

```
 In[2]:=  Integrate[x^3 - 6 x + 3, {x, -5, 5}]
Out[2]:=  30
```

Lists

```
 In[1]:=  Divisors[24]
Out[1]:=  {1, 2, 3, 4, 6, 8, 12, 24}
```

Lists in Mathematica are denoted using curly braces. Some functions like Divisors[n] return lists. Other functions may have lists for their arguments.

```
 In[2]:=  Integrate[x^3 - 6 x + 3, {x, -5, 5}]
Out[2]:=  30
```

In Mathematica, vectors are given as lists, and matrices are given as lists of lists (where the first list corresponds to the top row and so on).

Lists

```
 In[1]:=  Divisors[24]
Out[1]:=  {1, 2, 3, 4, 6, 8, 12, 24}
```

Lists in Mathematica are denoted using curly braces. Some functions like Divisors[n] return lists. Other functions may have lists for their arguments.

```
 In[2]:=  Integrate[x^3 - 6 x + 3, {x, -5, 5}]
Out[2]:=  30
```

In Mathematica, vectors are given as lists, and matrices are given as lists of lists (where the first list corresponds to the top row and so on).

```
 In[3]:=  Det[{{2, 1}, {1, 2}}]
Out[3]:=  3
```

Chaining Functions

Often we may want to use one function inside another.

Chaining Functions

Often we may want to use one function inside another.

```
 In[1]:=  Exponent[Integrate[2 x^6 - 3x + 1, x], x]
Out[1]:=  7
```

Chaining Functions

Often we may want to use one function inside another.

```
 In[1]:=  Exponent[Integrate[2 x^6 - 3x + 1, x], x]
Out[1]:=  7
```

We can also use the percent sign '%' to refer to the output from the previous calculation – we have to be careful of the order here.

Chaining Functions

Often we may want to use one function inside another.

```
 In[1]:=  Exponent[Integrate[2 x^6 - 3x + 1, x], x]
Out[1]:=  7
```

We can also use the percent sign '%' to refer to the output from the previous calculation – we have to be careful of the order here.

```
 In[2]:=  RandomInteger[{1, 100}]
Out[2]:=  78
```

```
 In[3]:=  Prime[%]
Out[3]:=  397
```

We can also use '%n' to refer to the output on the n^{th} line.

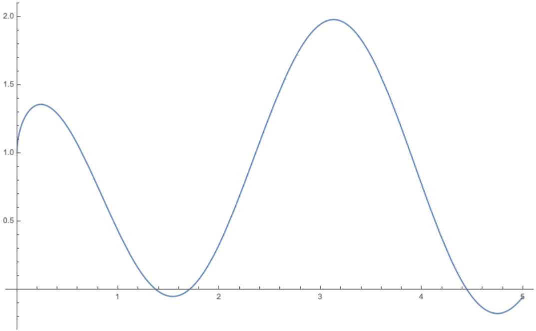
Plotting Basics

It can often be useful to plot a function in order to better understand its nature.

Plotting Basics

It can often be useful to plot a function in order to better understand its nature.

```
In[1]:=  Plot[Sin[Sqrt[x]] + Cos[2 x], {x, 0, 5}]
```



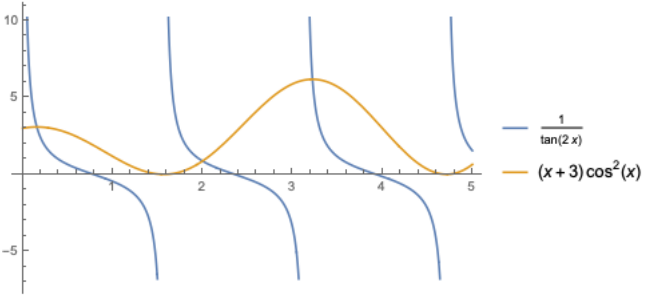
Plotting Basics

If we want to compare functions, we can put multiple functions in a list.

Plotting Basics

If we want to compare functions, we can put multiple functions in a list.

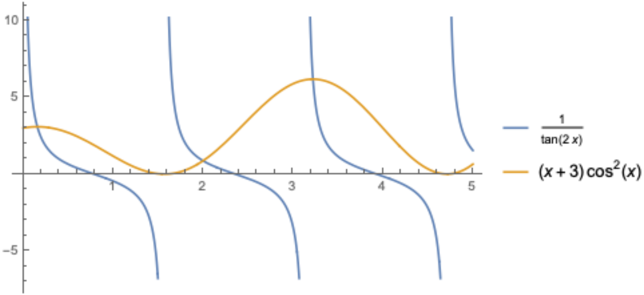
```
In[1]:=  Plot[{1/Tan[2 x], (x + 3) Cos[x]^2},
         {x, 0, 5}, PlotLegends -> "Expressions"]
```



Plotting Basics

If we want to compare functions, we can put multiple functions in a list.

```
In[1]:=  Plot[{1/Tan[2 x], (x + 3) Cos[x]^2},
         {x, 0, 5}, PlotLegends -> "Expressions"]
```

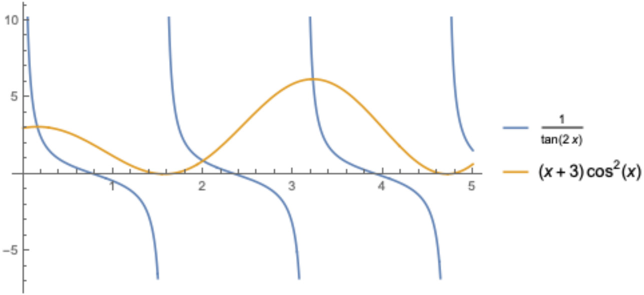


We have added an optional argument to add a legend.

Plotting Basics

If we want to compare functions, we can put multiple functions in a list.

```
In[1]:=  Plot[{1/Tan[2 x], (x + 3) Cos[x]^2},
         {x, 0, 5}, PlotLegends -> "Expressions"]
```



We have added an optional argument to add a legend.
Mathematica has automatically applied a *PlotStyle* to our plot.

The Documentation

Mathematica has so many functions, that one of the most important things to learn is how to look up the definitions of functions as you need them.

The Documentation

Mathematica has so many functions, that one of the most important things to learn is how to look up the definitions of functions as you need them. Luckily Mathematica has very good documentation.

The Documentation

Mathematica has so many functions, that one of the most important things to learn is how to look up the definitions of functions as you need them. Luckily Mathematica has very good documentation.

You can access the documentation either online (https://reference.wolfram.com/language/) or by opening the help menu and selecting *Wolfram Documentation*. From the documentation, you can view guides and tutorials which explain many common functions or search for functions by name.

The Documentation

Mathematica has so many functions, that one of the most important things to learn is how to look up the definitions of functions as you need them. Luckily Mathematica has very good documentation.

You can access the documentation either online (https://reference.wolfram.com/language/) or by opening the help menu and selecting *Wolfram Documentation*. From the documentation, you can view guides and tutorials which explain many common functions or search for functions by name.

Each function is described in the documentation with examples, possible options and related functions – searching for a function and then looking at the related functions is often a good way to find the function you need.

The Documentation

Mathematica has so many functions, that one of the most important things to learn is how to look up the definitions of functions as you need them. Luckily Mathematica has very good documentation.

You can access the documentation either online (https://reference.wolfram.com/language/) or by opening the help menu and selecting *Wolfram Documentation*. From the documentation, you can view guides and tutorials which explain many common functions or search for functions by name.

Each function is described in the documentation with examples, possible options and related functions – searching for a function and then looking at the related functions is often a good way to find the function you need.

As you start typing in a cell, Mathematica will also show a box of possible autocompletion options. This is another way of finding the function you need.

The Documentation

Mathematica has so many functions, that one of the most important things to learn is how to look up the definitions of functions as you need them. Luckily Mathematica has very good documentation.

You can access the documentation either online (https://reference.wolfram.com/language/) or by opening the help menu and selecting *Wolfram Documentation*. From the documentation, you can view guides and tutorials which explain many common functions or search for functions by name.

Each function is described in the documentation with examples, possible options and related functions – searching for a function and then looking at the related functions is often a good way to find the function you need.

As you start typing in a cell, Mathematica will also show a box of possible autocompletion options. This is another way of finding the function you need. Once you have typed the name of a function, Mathematica will also show a dropdown menu with links to the documentation if you hover your mouse over the function name.

Documentation Demo

Questions?

Practical:

- Download the Notebook for this workshop from my website
  www.maths.dur.ac.uk/~sxwc62/blog/
- Work through the 'Basic Mathematica' section of the notebook, using the documentation to look up the necessary functions.

Document Structure

- The notebook format is very good for helping to organise long documents, or for creating files that can be easily understood when you return to them at a later date.

Document Structure

- The notebook format is very good for helping to organise long documents, or for creating files that can be easily understood when you return to them at a later date.
- If you right click on your document you will have an option for inserting a new cell. The various levels of headings that can be added allow you to easily structure your document and have good default styling.

Document Structure

- The notebook format is very good for helping to organise long documents, or for creating files that can be easily understood when you return to them at a later date.
- If you right click on your document you will have an option for inserting a new cell. The various levels of headings that can be added allow you to easily structure your document and have good default styling.
- Adding headings also automatically puts cells into groups. These groups can be opened/closed with the shortcut **Ctrl+'** (or **Cmd+'** on Mac).

Document Structure

- The notebook format is very good for helping to organise long documents, or for creating files that can be easily understood when you return to them at a later date.
- If you right click on your document you will have an option for inserting a new cell. The various levels of headings that can be added allow you to easily structure your document and have good default styling.
- Adding headings also automatically puts cells into groups. These groups can be opened/closed with the shortcut **Ctrl+'** (or **Cmd+'** on Mac).
- You can also add cells which just contain plain (or formatted) text. This can be used to add explanations to your notebook, making it easier for someone else (or yourself at a later date) to understand.

Defining Your Own functions

To define your own function, you put the name and variables of the function first, then ':=', then the definition of your function.

```
 In[1]:=  myFunc[x_,y_] := x^2+x*y-3
 In[2]:=  myFunc[4,6]
Out[2]:=  37
```

Defining Your Own functions

To define your own function, you put the name and variables of the function first, then ':=', then the definition of your function.

```
 In[1]:=  myFunc[x_,y_] := x^2+x*y-3
 In[2]:=  myFunc[4,6]
Out[2]:=  37
```

Note that the variables for our function have an underscore after them on the left hand side of the definition **only**.

Defining Your Own functions

To define your own function, you put the name and variables of the function first, then ':=', then the definition of your function.

```
 In[1]:=  myFunc[x_,y_] := x^2+x*y-3
 In[2]:=  myFunc[4,6]
Out[2]:=  37
```

Note that the variables for our function have an underscore after them on the left hand side of the definition **only**.

Your functions can call any other function (including your own), and unless specified (see later) will try to evaluate on whatever input you give.

```
 In[3]:=  padToLength24[vec_] :=
         Join[Table[0, 24 - Length[vec]],vec]
 In[4]:=  toBinaryCodeword[n_] :=
         padToLength24[IntegerDigits[n,2]]
```

Prefix, Postfix, Infix?

All the functions we've used so far have been used as *Prefix* functions – the function name comes first, with the arguments after inside brackets.

Prefix, Postfix, Infix?

All the functions we've used so far have been used as *Prefix* functions – the function name comes first, with the arguments after inside brackets.

Sometimes it is convenient to put a function you want to apply at the end of the line. This is known as *postfix notation* and is often used for functions which change the display of the output.

```
In[1]:=  A = {{1, 2}, {3, 4}};
In[2]:=  B = {{5, 6}, {7, 8}};
In[3]:=  (A.B +2A) // MatrixForm
```

This displays the matrix as a rectangular array, rather than as a list of lists, making it easier to read the output.

Prefix, Postfix, Infix?

All the functions we've used so far have been used as *Prefix* functions – the function name comes first, with the arguments after inside brackets.

Sometimes it is convenient to put a function you want to apply at the end of the line. This is known as *postfix notation* and is often used for functions which change the display of the output.

```
 In[1]:=  A = {{1, 2}, {3, 4}};
 In[2]:=  B = {{5, 6}, {7, 8}};
 In[3]:=  (A.B +2A) // MatrixForm
```

This displays the matrix as a rectangular array, rather than as a list of lists, making it easier to read the output.

Some functions, including many standard mathematical operators ($+$,$-$,$*$,$/$), are used as *infix* operators.

```
 In[4]:=  Table[i, {i, 1, 5}] ~ Join ~ Divisors[6]
Out[4]:=  {1, 2, 3, 4, 5, 1, 2, 3, 6}
```

FullForm And Head

In Mathematica, everything is an expression. Everything is a nested series of functions operating on variables, even if Mathematica displays them in a more compact way.

FullForm And Head

In Mathematica, everything is an expression. Everything is a nested series of functions operating on variables, even if Mathematica displays them in a more compact way.

```
 In[1]:=  FullForm[x + z^6/y^(2/3)]
Out[1]:=  Plus[x,Times[Power[y,Rational[-2,3]]
               ,Power[z,6]]]
```

FullForm And Head

In Mathematica, everything is an expression. Everything is a nested series of functions operating on variables, even if Mathematica displays them in a more compact way.

```
 In[1]:=  FullForm[x + z^6/y^(2/3)]
Out[1]:=  Plus[x,Times[Power[y,Rational[-2,3]]
                ,Power[z,6]]]
```

Whenever we have an expression of the form f[x,y], f is known as the *head* of the expression.

```
 In[2]:=  Head[{1,2,3}]
Out[2]:=  List
```

FullForm And Head

In Mathematica, everything is an expression. Everything is a nested series of functions operating on variables, even if Mathematica displays them in a more compact way.

```
 In[1]:=  FullForm[x + z^6/y^(2/3)]
Out[1]:=  Plus[x,Times[Power[y,Rational[-2,3]]
                ,Power[z,6]]]
```

Whenever we have an expression of the form f[x,y], f is known as the *head* of the expression.

```
 In[2]:=  Head[{1,2,3}]
Out[2]:=  List
```

Understanding how Mathematica stores its expressions allows us to write much more powerful functions.

Map And Apply

The next two functions we're going to consider can be easily confused.

Map And Apply

The next two functions we're going to consider can be easily confused.

The function Apply[] lets us replace the head of an expression with another head.

Map And Apply

The next two functions we're going to consider can be easily confused.

The function Apply[] lets us replace the head of an expression with another head. If we didn't know about the function Total[], we might do the following

```
 In[1]:=  Apply[Plus,{1,2,3}]
Out[1]:=  6
```

Map And Apply

The next two functions we're going to consider can be easily confused.

The function Apply[] lets us replace the head of an expression with another head. If we didn't know about the function Total[], we might do the following

```
 In[1]:=  Apply[Plus,{1,2,3}]
Out[1]:=  6
```

The function Map[] lets us call a function on each value of a list in turn. The output is a list of the values.

```
 In[1]:=  f[x_]:= x - 3
 In[1]:=  Map[f,{1,2,3}]
Out[1]:=  {-2, -1, 0}
```

Patterns

We can use the Head of an expression to restrict whether our function tries to evaluate on the expression or not.

```
 In[1]:=  f[x_Integer]:= Mod[x,2]
 In[2]:=  Map[f, Table[i/2, {i, 0, 6}]]
Out[2]:=  {0, f[1/2], 1, f[3/2], 0, f[5/2], 1}
```

Patterns

We can use the Head of an expression to restrict whether our function tries to evaluate on the expression or not.

```
 In[1]:=  f[x_Integer]:= Mod[x,2]
 In[2]:=  Map[f, Table[i/2, {i, 0, 6}]]
Out[2]:=  {0, f[1/2], 1, f[3/2], 0, f[5/2], 1}
```

Here, we've only those x which have Head of Integer will be evaluated.

Patterns

We can use the Head of an expression to restrict whether our function tries to evaluate on the expression or not.

```
 In[1]:=  f[x_Integer]:= Mod[x,2]
 In[2]:=  Map[f, Table[i/2, {i, 0, 6}]]
Out[2]:=  {0, f[1/2], 1, f[3/2], 0, f[5/2], 1}
```

Here, we've only those x which have Head of Integer will be evaluated. We can use this to *overload* a function, making it do different things depending on the argument.

Patterns

We can use the Head of an expression to restrict whether our function tries to evaluate on the expression or not.

```
 In[1]:=  f[x_Integer]:= Mod[x,2]
 In[2]:=  Map[f, Table[i/2, {i, 0, 6}]]
Out[2]:=  {0, f[1/2], 1, f[3/2], 0, f[5/2], 1}
```

Here, we've only those x which have Head of Integer will be evaluated. We can use this to *overload* a function, making it do different things depending on the argument.

```
 In[3]:=  f[x_List]:= Map[f,List]
Out[3]:=  f[{1,3/2,2}]:= {1,f[3/2],0}
```

Patterns

Patterns

Patterns in Mathematica are very powerful for defining complex functions. Although we don't have time to discuss in detail, we can define functions only on arguments which match arbitrary functions rather than just by testing the Head.

One more type of pattern that you might need for the exercises is the following

Patterns

Patterns in Mathematica are very powerful for defining complex functions. Although we don't have time to discuss in detail, we can define functions only on arguments which match arbitrary functions rather than just by testing the Head.

One more type of pattern that you might need for the exercises is the following

```
In[3]:=  g[x_?EvenQ]:= x/2
In[4]:=  g[x_?Odd]:= 3x+1
```

Patterns

Patterns in Mathematica are very powerful for defining complex functions. Although we don't have time to discuss in detail, we can define functions only on arguments which match arbitrary functions rather than just by testing the Head.

One more type of pattern that you might need for the exercises is the following

```
In[3]:=  g[x_?EvenQ]:= x/2
In[4]:=  g[x_?Odd]:= 3x+1
```

This applies one definition of g if the function EvenQ returns True, and another definition if the function OddQ returns True.

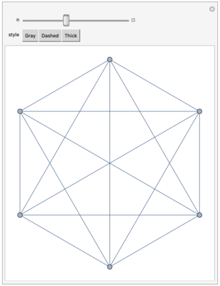
Dynamic Plots

Mathematica has a few ways to produce dynamic content – content that changes either in response to user input, or due to other ongoing computations.

Dynamic Plots

Mathematica has a few ways to produce dynamic content – content that changes either in response to user input, or due to other ongoing computations. Manipulate allows us to change one or more values inside a function and have the function automatically update.

```
In[1]:=  Manipulate[CompleteGraph[n, EdgeStyle ->
        style], {n, 2, 12, 1}, {style, {Gray -> "Gray",
        Dashed -> "Dashed", Thick -> "Thick"}}]
```



Questions?

Practical:

- Work through the ‘More Mathematica’ section of the notebook,