Brian’s Wolfram Language Cheat Sheet

A Wolfram Language notebook containing a compilation of fundamental, low-level syntax and functions (such as @@, @@@, /@ ./, Table, Array, Module, etc.)

Fundamental Functions and Syntax

These are functions and syntax that relate directly to the application of functions to symbols or lists.

Apply — Another way of Applying a Function to a List of Arguments

```
In[17]:= Apply[f, {a, {b1, b2}, {{c11, c12}, {c21, c22}}}]
Out[17]=
         f[a, {b1, b2}, {{c11, c12}, {c21, c22}}]
```

Apply — Can Take a Level Specification

```
In[18]:= Apply[f, {a, {b1, b2}, {{c11, c12}, {c21, c22}}}, {0}]
Out[18]=
         f[a, {b1, b2}, {{c11, c12}, {c21, c22}}]
```

```
In[19]:= Apply[f, {a, {b1, b2}, {{c11, c12}, {c21, c22}}}, {1}]
Out[19]=
         {a, f[b1, b2], f[{c11, c12}, {c21, c22}]}
```

```
In[20]:= Apply[f, {a, {b1, b2}, {{c11, c12}, {c21, c22}}}, {2}]
Out[20]=
         {a, {b1, b2}, {f[c11, c12], f[c21, c22]}}
```

The default level specification is {0}.

Apply — Behaves Strangely at Level 0 if you Don’t Give it a List

What is this good for:

```
         Apply[f, a]
Out[38]=
         a
```

@@ — A Shorthand for Apply

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

```
In[42]:= Apply[f, {x, y, z}]
Out[42]=
         f[x, y, z]
```

@ vs @@

```
In[43]:= f@x
Out[43]=
         f[x]

In[44]:= f @@ {x}
Out[44]=
         f[x]

In[45]:= Sin@{x, y}
Out[45]=
         {Sin[x], Sin[y]}

In[47]:= Sin @@ {{x, y}}
Out[47]=
         {Sin[x], Sin[y]}

In[49]:= f@{x, y}
Out[49]=
         f[{x, y}]

In[48]:= f @@ {{x, y}}
Out[48]=
         f[{x, y}]
```

Prefix — Has some Fundamental Relationship to @

```
In[24]:= Prefix[f[x]]
Out[24]=
         f@x

         f[x]
Out[25]=
         f@x
```

// — Apply as an Afterthought

```
In[∘]:= Array[Plus, {10, 10}] // Grid
Out[∘]=
         2  3  4  5  6  7  8  9 10 11
         3  4  5  6  7  8  9 10 11 12
         4  5  6  7  8  9 10 11 12 13
         5  6  7  8  9 10 11 12 13 14
         6  7  8  9 10 11 12 13 14 15
         7  8  9 10 11 12 13 14 15 16
         8  9 10 11 12 13 14 15 16 17
         9 10 11 12 13 14 15 16 17 18
        10 11 12 13 14 15 16 17 18 19
        11 12 13 14 15 16 17 18 19 20
```

Map — Make a New List by Applying a Function to Each Element in a List

```
In[∘]:= Map[f, {x, y, z}]
Out[∘]=
        {f[x], f[y], f[z]}
```

Map and /@ are Not Needed for Functions that Are Already Listable

```
In[∘]:= Map[Sin, {x, y, z}]
Out[∘]=
        {Sin[x], Sin[y], Sin[z]}

In[∘]:= Sin /@ {x, y, z}
Out[∘]=
        {Sin[x], Sin[y], Sin[z]}

In[∘]:= {x, y, z} // Sin
Out[∘]=
        {Sin[x], Sin[y], Sin[z]}
```

Since Sin is listable, just use:

```
In[∘]:= Sin[{x, y, z}]
Out[∘]=
        {Sin[x], Sin[y], Sin[z]}

In[∘]:= Sin@{x, y, z}
Out[∘]=
        {Sin[x], Sin[y], Sin[z]}
```

But interestingly, even though Sin is listable, you cannot use:

```
In[∘]:=  Apply[Sin, {x, y, z}]

         Sin: Sin called with 3 arguments; 1 argument is expected.
Out[∘]=
         Sin[x, y, z]
```

Apply vs @

So Apply with a list and @ are not identical, even though with one argument they are:

```
In[35]:=  Sin@1
Out[35]=
          Sin[1]

In[39]:=  Apply[Sin, {1}]
Out[39]=
          Sin[1]

In[40]:=  Sin@{1, 2}
Out[40]=
          {Sin[1], Sin[2]}

In[41]:=  Apply[Sin, {{1, 2}}]
Out[41]=
          {Sin[1], Sin[2]}
```

/@ — A Shorthand for Map

```
In[∘]:=  f /@ {x, y, z}
Out[∘]=
         {f[x], f[y], f[z]}
```

MapApply

```
In[∘]:=  MapApply[f, {{x, y}, {z}, {a, b, c}}]
Out[∘]=
         {f[x, y], f[z], f[a, b, c]}
```

@@@ — A Shorthand for MapApply

```
In[∘]:=  f @@@ {{x, y}, {z}, {a, b, c}}
Out[∘]=
         {f[x, y], f[z], f[a, b, c]}
```