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

Apply — Can Take a Level Specification

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

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

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

@@ — A Shorthand for Apply

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

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

@ vs @@

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

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

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

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

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

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

Prefix — Has some Fundamental Relationship to @

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

        f[x]
Out[ ]=
        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[•]:= Sin@1
Out[•]=
         Sin[1]

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

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

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

Datasets

The following from *EIWL3* Section 45 is both powerful and confusing:

“Anywhere you can ask for a part [in a dataset] you can also give a function that will be applied to all parts at that level.” For example:

```
In[21]:= data = Dataset[<|"a" → <|"x" → 1, "y" → 2, "z" → 3|>, "b" → <|"x" → 5, "y" → 10, "z" → 7|> |>]
         data[All, f]
Out[21]=
```

	x	y	z
a	1	2	3
b	5	10	7

```
data[All, f]
Out[22]=
```

a	f[<|"x" → 1, "y" → 2, "z" → 3|>]
b	f[<|"x" → 5, "y" → 10, "z" → 7|>]

```
In[23]:= data[All, All, f]
Out[23]=
```

	x	y	z
a	f[1]	f[2]	f[3]
b	f[5]	f[10]	f[7]

Furthermore, the function can be a Select statement, where it is intended that we use the “operator form” of Select[]. For example:

```
In[12]:= data[Select[#z > 5 &]]
Out[12]=
```

b	x	5
	y	10
	z	7

```
In[25]:= data[All, Select[# > 5 &]]
Out[25]=
```

a		
b	y	10
	z	7

```
In[31]:=
(* The following is illegal,
making it appear that you can't have a list of associations: *)
(* {<|"x"→1,"y"→2,"z"→3|>,"b"→<|"x"→5,"y"→10,"z"→7|>} *)

(* However, you can but you have to do it this way: *)

listOfAssociations =
 {Association["x" → 5, "y" → 2, "z" → 3], Association["x" → 1, "y" → 10, "z" → 7]}
Out[31]=
{<|x → 5, y → 2, z → 3|>, <|x → 1, y → 10, z → 7|>}

In[32]:= SortBy[#z &][listOfAssociations]
Out[32]=
{<|x → 5, y → 2, z → 3|>, <|x → 1, y → 10, z → 7|>}

In[33]:= SortBy[#x &][listOfAssociations]
Out[33]=
{<|x → 1, y → 10, z → 7|>, <|x → 5, y → 2, z → 3|>}
```