Jeremy’s Cool Shortcuts!

Using the @ symbol is the same as applying a function.

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

// applies a function in reverse.

```
In[•]:= x // f
Out[•]=
        f[x]
```

Applying a function to lists can be done using /@ (this is a shortcut for the Map[] function).

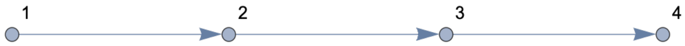
```
In[•]:= f /@ {x, y, z}
Out[•]=
        {f[x], f[y], f[z]}
```

A function can be made pure by using the & symbol. The pound sign (#) is used for slots.

```
In[•]:= f[#] &[x]
Out[•]=
        f[x]
```

NestList[] creates a list of iterative outputs. Nest[] shows just the final one. NestGraph[] fulfils an analogous purpose for graphs.

```
In[•]:= NestList[f, x, 3]
        Nest[f, x, 3]
        NestGraph[{# + 1} &, 1, 3, VertexLabels → All]
Out[•]=
        {x, f[x], f[f[x]], f[f[f[x]]]}
Out[•]=
        f[f[f[x]]]
Out[•]=
```



Array acts like table but can produce n-dimensional outputs. FoldList folds elements iteratively from a list.

```
In[∘]:= Array[f, {2, 2}] // Grid
        FoldList[Plus, 0, Range[3]]
Out[∘]=
        f[1, 1] f[1, 2]
        f[2, 1] f[2, 2]
Out[∘]=
        {0, 1, 3, 6}
```

Items from lists can be pulled by specifying the index or span.

```
In[∘]:= Range[10]⟦{1, 3, 5}⟧
        Range[10]⟦1 ;; 5⟧
Out[∘]=
        {1, 3, 5}
Out[∘]=
        {1, 2, 3, 4, 5}
```

The command /@ does not necessarily perform the same thing when the item a function is applied to is also a function.

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

A list can be used as the argument for a function using @@.

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

Three @s (@@@) applies a function to each sublist in a list.

```
In[∘]:= f @@@ {{1, 2, 3}, {4, 5, 6}}
Out[∘]=
        {f[1, 2, 3], f[4, 5, 6]}
```

Modules can be used while naming functions to create local variables that will not recognised outside of the function.

```
In[∘]:= Module[{x = 1}, x + 1]
        x
Out[∘]=
        2
Out[∘]=
        x
```