Rania's Wolfram Language Cheat Sheet

Syntax

Single List

```
f@ {a, b, c}
f @@ {a, b, c} (*apply*)
f /@ {a, b, c} (*map*)
f @@@ {a, b, c}
{a, b, c} // f
```

Out[•]=
```
f[{a, b, c}]
```

Out[•]=
```
f[a, b, c]
```

Out[•]=
```
{f[a], f[b], f[c]}
```

Out[•]=
```
{a, b, c}
```

Out[•]=
```
f[{a, b, c}]
```

List of List

In[•]:=
```
f@ {{a}, {b}, {c}}
f @@ {{a}, {b}, {c}}
f /@ {{a}, {b}, {c}}
f @@@ {{a}, {b}, {c}}
{{a}, {b}, {c}} // f
```

Out[•]=
```
f[{{a}, {b}, {c}}]
```

Out[•]=
```
f[{a}, {b}, {c}]
```

Out[•]=
```
{f[{a}], f[{b}], f[{c}]}
```

Out[•]=
```
{f[a], f[b], f[c]}
```

Out[•]=
```
f[{{a}, {b}, {c}}]
```

Pure Functions

In[•]:=
```
Power[#, 2] & /@ {1, 2, 3}
```
Out[•]=
```
{1, 4, 9}
```

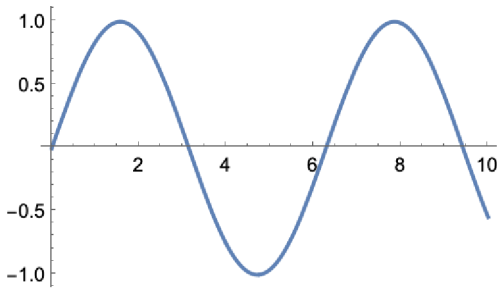
In[•]:=
```
Select[Range[26], EvenQ[#] &] (*Pay Attention to & notation*)
```
Out[•]=
```
{2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26}
```

Conditionals

In[•]:=
```
(*&& and
 || or
   ! not *)
If[8 == 1, "f is 1", "f is not 1"]
```
Out[•]=
```
f is not 1
```

Functions

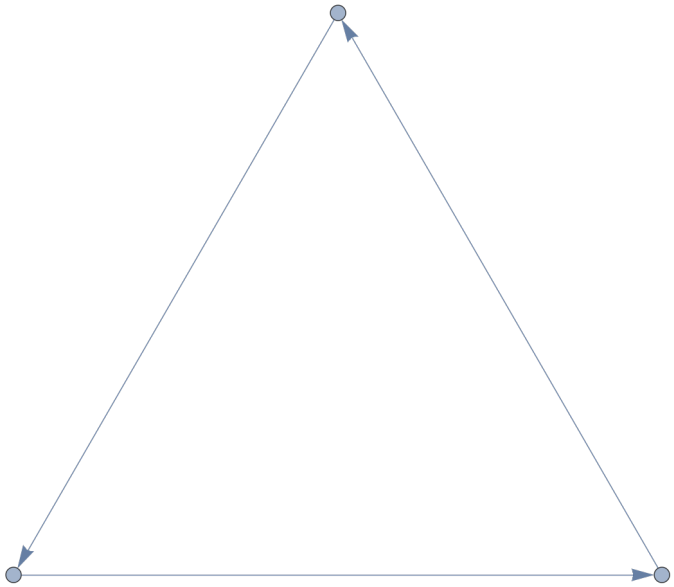
In[•]:=
```
(* Rasterize - Converts an expression to a rasterized image *)
Rasterize["Hello, world!"]
Rasterize[Plot[Sin[x], {x, 0, 10}]]
```
Out[•]=
```
Hello, world!
```
Out[•]=



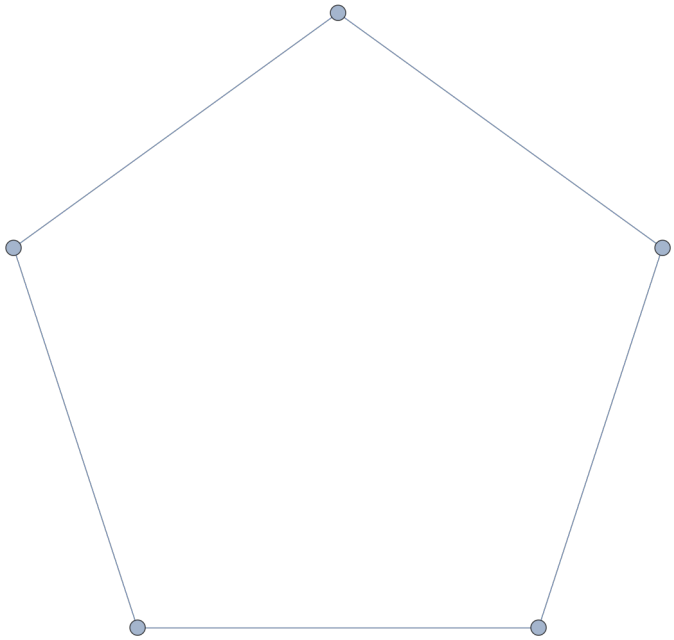
In[•]:=
```
(* Mod - Computes the remainder of division *)
Mod[10, 3] (* 10 mod 3 → 1 *)
Mod[-10, 3, 1] (* Shifts remainder into range centered around 1 *)
```
Out[•]=
```
1
```
Out[•]=
```
2
```

In[•]:=
```
(* Graph - Creates a graph from vertex and edge specifications *)
Graph[{1 → 2, 2 → 3, 3 → 1}]
Graph[Range[5], {1 ⟷ 2, 2 ⟷ 3, 3 ⟷ 4, 4 ⟷ 5, 5 ⟷ 1}]
```

Out[•]=

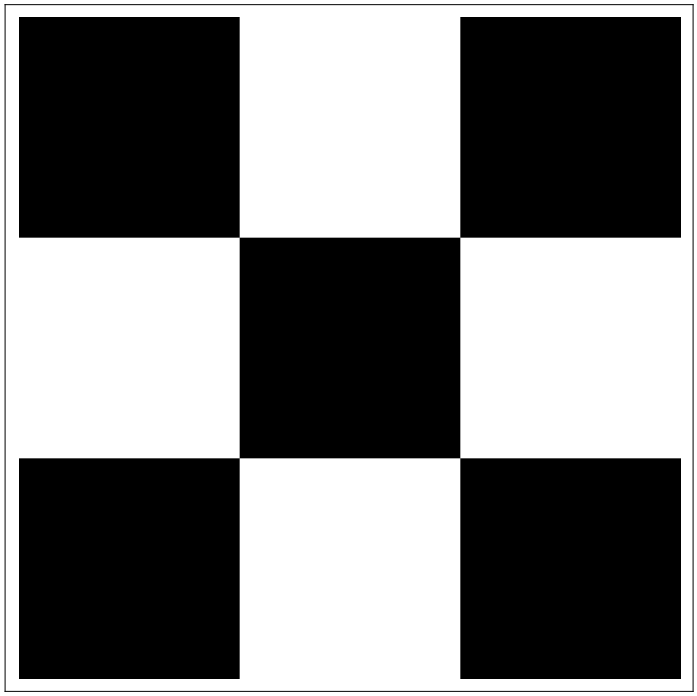


Out[•]=

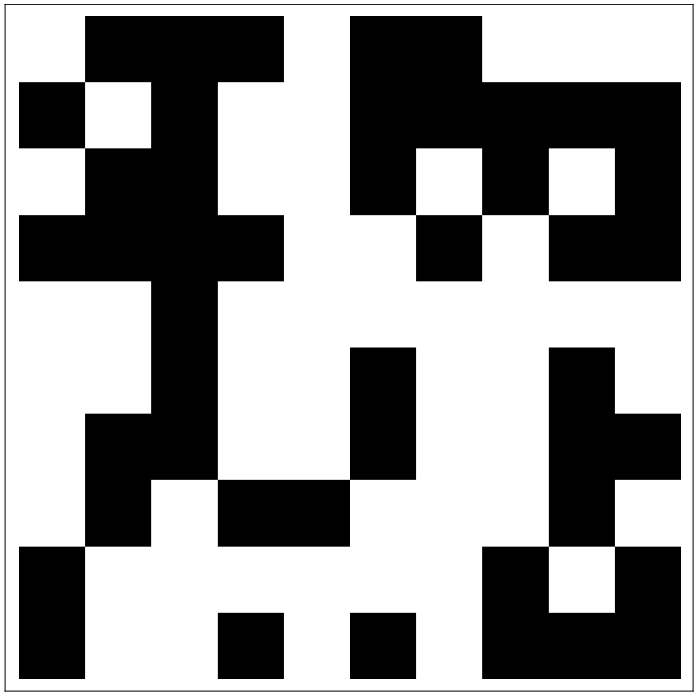


In[•]:=
```
(* ArrayPlot - Visualizes matrices as images *)
ArrayPlot[{{1, 0, 1}, {0, 1, 0}, {1, 0, 1}}]
ArrayPlot[RandomInteger[{0, 1}, {10, 10}]]
```

Out[•]=



Out[•]=



```
In[•]:= (* Array - Generates an array of values based on an expression *)
        Array[f, 5] (* {f[1], f[2], f[3], f[4], f[5]} *)
        Array[Prime, 10] (* First 10 prime numbers *)
Out[•]=
        {f[1], f[2], f[3], f[4], f[5]}
Out[•]=
        {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}

In[•]:= (* Nest - Applies a function repeatedly *)
        Nest[Sqrt, 256, 3] (* sqrt(sqrt(sqrt(256))) *)
        Nest[RotateRight, {a, b, c, d}, 2]
Out[•]=
        2
Out[•]=
        {c, d, a, b}

In[•]:= (* NestList - Like Nest, but returns intermediate results *)
        NestList[Sqrt, 256, 3]
        NestList[RotateRight, {a, b, c, d}, 2]
Out[•]=
        {256, 16, 4, 2}
Out[•]=
        {{a, b, c, d}, {d, a, b, c}, {c, d, a, b}}

In[•]:= (* PrimeQ - Tests if a number is prime *)
        PrimeQ[7] (* True *)
        PrimeQ[10] (* False *)
Out[•]=
        True
Out[•]=
        False

In[•]:= (* MemberQ - Checks if an element is in a list *)
        MemberQ[{a, b, c}, b] (* True *)
Out[•]=
        True

In[•]:= (* EvenQ - Tests if a number is even *)
        EvenQ[4] (* True *)
        EvenQ[3] (* False *)
Out[•]=
        True
Out[•]=
        False
```

```
In[•]:= (* Last & First - Get the last/first element of a list *)
        Last[{1, 2, 3, 4}]
        First[{1, 2, 3, 4}]
Out[•]=
        4
Out[•]=
        1

In[•]:= (* Select - Filters elements based on a condition *)
        Select[Range[10], PrimeQ]
Out[•]=
        {2, 3, 5, 7}

In[•]:= (* Total - Computes the sum of elements *)
        Total[{1, 2, 3, 4}]
        Total[{{1, 2}, {3, 4}}, {2}] (* Sum along second level *)
Out[•]=
        10
Out[•]=
        {3, 7}

In[•]:= (* Thread - Applies a function element-wise to lists *)
        Thread[{a, b, c} + {1, 2, 3}]
        Thread[Equal[{a, b, c}, {1, 2, 3}]]
Out[•]=
        {1 + a, 2 + b, 3 + c}
Out[•]=
        {a == 1, b == 2, c == 3}

In[•]:= (* Grid - Displays elements in a tabular format *)
        Grid[{{"A", "B"}, {1, 2}, {3, 4}}]
Out[•]=
        A B
        1 2
        3 4

In[•]:= (* Partition - Splits a list into sublists *)
        Partition[Range[10], 2]
        Partition[Range[10], 3, 1] (* Overlapping partitions *)
Out[•]=
        {{1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10}}
Out[•]=
        {{1, 2, 3}, {2, 3, 4}, {3, 4, 5}, {4, 5, 6}, {5, 6, 7}, {6, 7, 8}, {7, 8, 9}, {8, 9, 10}}

In[•]:= (* ArrayFlatten - Flattens nested arrays into a single matrix *)
        ArrayFlatten[{{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}}]
Out[•]=
        {{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}}
```

```
In[•]:= (* Flatten - Flattens nested lists *)
        Flatten[{{1, {2, 3}}, {4, 5}}]
        Flatten[{{1, {2, 3}}, {4, 5}}, 1] (* Flatten only one level *)
Out[•]=
        {1, 2, 3, 4, 5}
Out[•]=
        {1, {2, 3}, 4, 5}

In[•]:= (* Max - Finds the maximum element *)
        Max[3, 10, 7]
        Max[{3, 10, 7}]
Out[•]=
        10
Out[•]=
        10

In[•]:= (* Split - Groups consecutive identical elements *)
        Split[{1, 1, 2, 2, 2, 3, 3, 1}]
Out[•]=
        {{1, 1}, {2, 2, 2}, {3, 3}, {1}}

In[•]:= (* GatherBy - Groups elements based on a function *)
        GatherBy[{1, 2, 3, 4, 5, 6}, EvenQ]
Out[•]=
        {{1, 3, 5}, {2, 4, 6}}

In[•]:= (* RandomSample - Randomly selects elements from a list *)
        RandomSample[Range[10], 5]
Out[•]=
        {6, 7, 1, 5, 2}

In[•]:= (* Tuples - Generates all possible tuples of given length *)
        Tuples[{0, 1}, 3] (* All binary strings of length 3 *)
Out[•]=
        {{0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {0, 1, 1}, {1, 0, 0}, {1, 0, 1}, {1, 1, 0}, {1, 1, 1}}

In[•]:= (* TakeSmallest - Extracts the smallest elements *)
        TakeSmallest[{5, 1, 3, 9, 2}, 3]
Out[•]=
        {1, 2, 3}

In[•]:= (* Join - Concatenates lists *)
        Join[{1, 2}, {3, 4}]
Out[•]=
        {1, 2, 3, 4}
```

```
In[•]:= (* ReplacePart - Replaces specified parts of an expression *)
        ReplacePart[{a, b, c}, {2 → x}]
        ReplacePart[{{a, b}, {c, d}}, {{1, 2} → x, {2, 1} → y}]
Out[•]=
        {a, x, c}
Out[•]=
        {{a, x}, {y, d}}

In[•]:= (* Nothing - Removes elements in a list transformation *)
        DeleteCases[{a, Nothing, b}, Nothing]
Out[•]=
        {a, b}

In[•]:= (* /@ and & in Array *)
        Sin /@ Range[5] (* Apply Sin to each element *)
        Array[#^2 &, 5] (* Square each index *)
Out[•]=
        {Sin[1], Sin[2], Sin[3], Sin[4], Sin[5]}
Out[•]=
        {1, 4, 9, 16, 25}

In[•]:= (* [] vs. // *)
        f[x] (* Direct function application *)
        x // f (* Same as f[x] *)
Out[•]=
        f[x]
Out[•]=
        f[x]

In[•]:= (* Cases - Extracts elements matching a pattern *)
        Cases[{1, 2, 3, 4, 5}, _Integer?EvenQ]
        Cases[IntegerDigits[2^1000], 0 | 1]
        Cases[{{a, b, c}, {d, e, f}}, {x_, y_, z_} :↦ {y, x, z}]
Out[•]=
        {2, 4}
Out[•]=
        {1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0}
Out[•]=
        {{b, a, c}, {e, d, f}}
```

```
In[•]:= (* Head - Gets the type of an expression *)
        Head[3] (* Integer *)
        Head[{1, 2, 3}] (* List *)
Out[•]=
        Integer
Out[•]=
        List

In[•]:= (* IntegerQ - Checks if a number is an integer *)
        IntegerQ[5.0] (* False *)
        IntegerQ[5] (* True *)
Out[•]=
        False
Out[•]=
        True

In[•]:= (* Rule - Creates transformation rules *)
        {a, b, c} /. a → x
Out[•]=
        {x, b, c}

In[•]:= (* Keys - Extracts keys from an association *)
        Keys[<|"a" → 1, "b" → 2|>]
Out[•]=
        {a, b}
```

Precedence

Function Application (@): This is used for prefix function application, where f @ x is equivalent to f[x]
Apply (@@): This operator replaces the head of an expression. For example, f @@ {a, b, c} changes the head of {a, b, c} from List to f, resulting in f[a, b, c]
Map (/@): This applies a function to each element in a list. For instance, f /@ {a, b, c} yields {f[a], f[b], f[c]}

The precedence of these operators:
@
@@
/ (Various operators like division)
/; (Condition)
/= (UpSet)
/. (ReplaceAll)
// (Postfix)
/@

To determine the precedence of operators: Precedence.
Precedence[Apply] (* Output: 650 *)
Precedence[ReplaceAll] (* Output: 110 *)
Higher values indicate higher precedence.

```
Precedence[Apply]  (*Output:650*)
Precedence[ReplaceAll]  (*Output:110*)
```

Out[•]=

```
620.
```

Out[•]=

```
110.
```