21. Functional Operation (Apply and Map)

2016-07-08

```
$Version
```

```
10.0 for Mac OS X x86 (64-bit) (September 10, 2014)
```

21.1 Applying Functions to Lists or Other Expressions (Apply)

In **f[{a,b,c}]** one is giving a list as the argument to a function. Often one needs to apply a function directly to the elements of a list, rather than to the list as a whole. This can be done just by appying the command **Apply** to **f[{a,b,c}]. Apply** just replaces the head of the expression it is acting on.

```
Clear[f]; f[{a, b, c}]
```

```
f[{a, b, c}]
```

```
Apply[f, {a, b, c}]
```

```
f[a, b, c]
```

```
Apply[Plus, {a, b, c} ]
```

```
a + b + c
```

```
Exp[{a, b, c}]
```

$\{e^a, e^b, e^c\}$

```
Apply[Exp, {a, b, c}]
```

Exp::argx: Exp called with 3 arguments; 1 argument is expected. >>

```
Exp[a, b, c]
```

Exp::argx : Exp called with 3 arguments; 1 argument is expected. >>

The operator **Exp** cannot be used in **Apply[],** since the result would be an exponential function with 3
simultaneous arguments. Now an example of a simple program is given, in which **Apply** is used to advantage:

21.1.1 A definition for the arimthmic mean

```
mymean[list_] := Apply[Plus, list] / Length[list]
```

```
mymean[{a, b, c, d}]
```

$\frac{1}{4}(a + b + c + d)$

```
list = RandomReal[{0, 1}, 10]
```

```
{0.0665109, 0.887212, 0.673656, 0.721573, 0.250424,
 0.297352, 0.00180066, 0.666908, 0.0229811, 0.495532}
```

```
mymean[list]
```

```
0.408395
```

Using the second argument to RandomReal and related functions is generally quicker than creating lists of randoms using Table.

```
mymean[RandomReal[{0, 1}, 100]]
```

0.504773

21.1.2 A definition for the product of several matrices

There is a law for generating a set of matrices, e.g.

```
matlaw = {{1, k}, {0, k}}
```

{{1, k}, {0, k}}

```
Table[matlaw // MatrixForm, {k, 1, 3}]
```

$\left\{\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 0 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 3 \\ 0 & 3 \end{pmatrix}\right\}$

The function for performing the matrix product of these matrices is defined with the help of **Apply[]**:

```
MatrixProduct[matexpr_, {i_, imin_, imax_}] :=
 Apply[Dot, Table[matexpr, {i, imin, imax}]]
```

```
MatrixProduct[{{1, k}, {0, k}}, {k, 1, 3}]
```

{{1, 15}, {0, 6}}

21.1.3 General description of Apply

Apply[f, {a,b,...}]	apply **f** to a list, giving f[a,b,...]
Apply[f, *expr*]	apply **f** to the top level of expression ***expr***
f @@ *expr*	praefix form of **Apply**
Apply[f, *expr*, *levels*]	apply **f** at the specified *levels* in ***expr***

Apply replaces the head of the expression used as the second argument by the expression given in the first argument. In a longer version of this command the level (or levels) may be indicated at which the first argument should be applied.
If ***levels*** comprises just one number (within or without braces) then the operators are replaced at just this level.
If ***levels*** is a list *({n1,n2})* then the operators are replaced at all levels from level **n1** till level **n2**.

Input	Output / Representation
Apply[List, a + b + c]	{a, b, c}
FullForm[a + b + c]	Plus[a, b, c]
List @@ (a + b + c)	{a, b, c}

$\{a^2, b^2, c^2\}$

Apply[List, a b c]	{a, b, c}
Exp[{*a*, *b*, *c*}]	$\{e^a, e^b, e^c\}$
Apply[List, Exp[{*a*, *b*, *c*}]]	$\{e^a, e^b, e^c\}$
Apply[List, Exp[a+b+c]]	{*e*, a + b + c}

The last result can be understood by looking at the expression as which **Exp[a+b+c]** is stored:

FullForm[Exp[a+b+c]]	Power[E,Plus[a,b,c]]

```
lr = RandomReal[{0, 1}, 1 000 000];
```

```
st = Sum[lr[[k]], {k, Length[lr]}] // Timing
```

{0.269471, 499 900.}

```
Apply[Plus, lr] // Timing
```

```
{0.160472, 499 900.}
```

One sees that the summation runs about three times faster with **Apply[]** than with **Sum[]**. In prefix form the
summation is written as:

```
Plus @@ lr
```

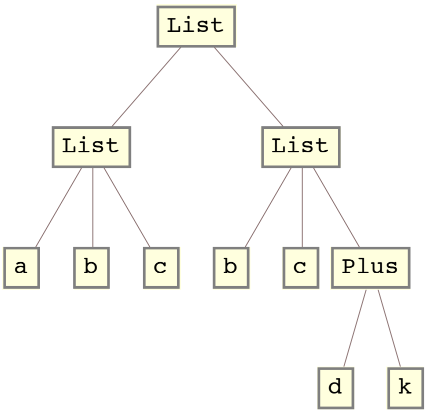
```
499 900.
```

In a longer version of this command the level (or levels) may be indicated at which the first argument should be applied.

```
m = { {a, b, c}, {b, c, d + k} }
```

```
{{a, b, c}, {b, c, d + k}}
```

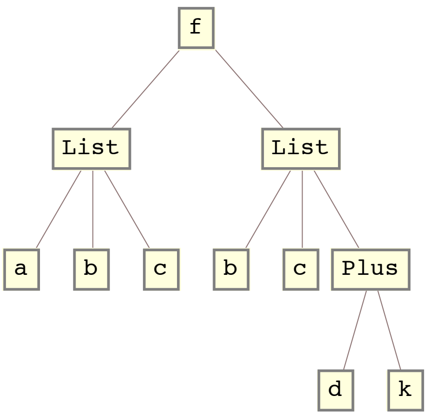
```
TreeForm[m, ImageSize -> 250]
```




```
Apply[f, m]
```

```
f[{a, b, c}, {b, c, d + k}]
```

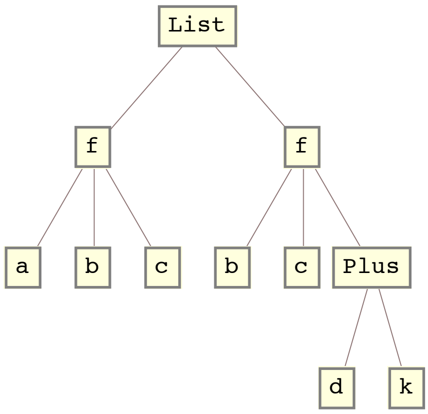
```
TreeForm[%,  ImageSize -> 250]
```




```
Apply[f, m, {1} ]
```

```
{f[a, b, c], f[b, c, d + k]}
```

```
TreeForm[%,  ImageSize -> 250]
```




```
Apply[f, m, 1 ]
```

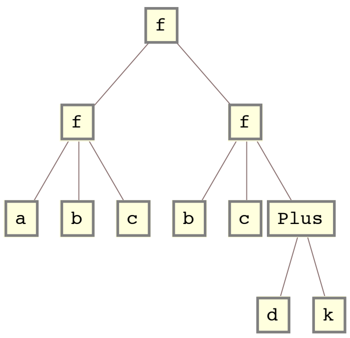
{f[a, b, c], f[b, c, d + k]}

This gives the same Treeform as above.

```
Apply[f, m, {0,1} ]
```

f[f[a, b, c], f[b, c, d + k]]

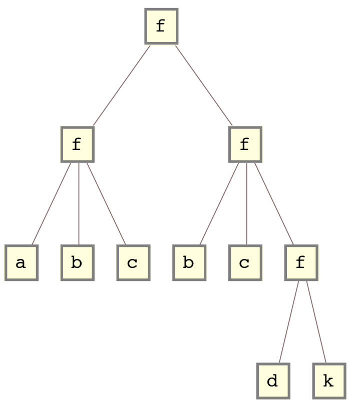
```
TreeForm[%, ImageSize → 200]
```




```
Apply[f, m, {0,2} ]
```

f[f[a, b, c], f[b, c, f[d, k]]]

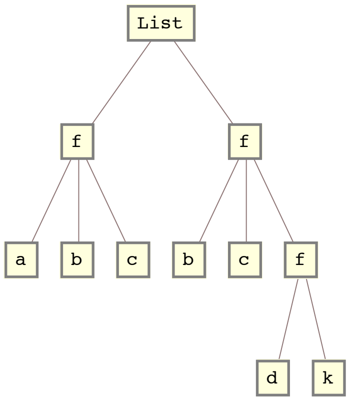
```
TreeForm[%, ImageSize → 200]
```




```
Apply[f, m, {1,2} ]
```

{f[a, b, c], f[b, c, f[d, k]]}

TreeForm[%, ImageSize → 200]



Apply[f, m, {0,1}]

f[f[a, b, c], f[b, c, d + k]]

m

{{a, b, c}, {b, c, d + k}}

Apply[Exp, m, {0,1}]

Exp::argx: Exp called with 3 arguments; 1 argument is expected. >>

Exp::argx: Exp called with 3 arguments; 1 argument is expected. >>

Exp::argx: Exp called with 2 arguments; 1 argument is expected. >>

General::stop: Further output of Exp::argx will be suppressed during this calculation. >>

Exp[Exp[a, b, c], Exp[b, c, d + k]]

Exp::argx : Exp called with 3 arguments; 1 argument is expected. >>

General::stop :
"Further output of Exp :: argx, will be suppressed during this calculation.>>"

xl = Array[x,{10}]

{x[1], x[2], x[3], x[4], x[5], x[6], x[7], x[8], x[9], x[10]}

yl = Array[y,{10}]

{y[1], y[2], y[3], y[4], y[5], y[6], y[7], y[8], y[9], y[10]}

xyl = Transpose[{xl, yl}]

{{x[1], y[1]}, {x[2], y[2]}, {x[3], y[3]}, {x[4], y[4]}, {x[5], y[5]},
{x[6], y[6]}, {x[7], y[7]}, {x[8], y[8]}, {x[9], y[9]}, {x[10], y[10]}}

Apply[Plus, xyl]

{x[1] + x[2] + x[3] + x[4] + x[5] + x[6] + x[7] + x[8] + x[9] + x[10],
y[1] + y[2] + y[3] + y[4] + y[5] + y[6] + y[7] + y[8] + y[9] + y[10]}

Apply[Plus, xyl, {1}]

{x[1] + y[1], x[2] + y[2], x[3] + y[3], x[4] + y[4], x[5] + y[5],
x[6] + y[6], x[7] + y[7], x[8] + y[8], x[9] + y[9], x[10] + y[10]}

xl + yl

{x[1] + y[1], x[2] + y[2], x[3] + y[3], x[4] + y[4], x[5] + y[5],
x[6] + y[6], x[7] + y[7], x[8] + y[8], x[9] + y[9], x[10] + y[10]}

Apply[Plus, xyl, {2}]

{{1, 1}, {2, 2}, {3, 3}, {4, 4}, {5, 5}, {6, 6}, {7, 7}, {8, 8}, {9, 9}, {10, 10}}

```
Apply[Plus, xyl, {3}]
```

```
{{x[1], y[1]}, {x[2], y[2]}, {x[3], y[3]}, {x[4], y[4]}, {x[5], y[5]},
 {x[6], y[6]}, {x[7], y[7]}, {x[8], y[8]}, {x[9], y[9]}, {x[10], y[10]}}
```

```
Depth[xyl]
```

```
4
```

```
Apply[Plus, xyl, {0, 1}]
```

```
x[1] + x[2] + x[3] + x[4] + x[5] + x[6] + x[7] + x[8] + x[9] +
 x[10] + y[1] + y[2] + y[3] + y[4] + y[5] + y[6] + y[7] + y[8] + y[9] + y[10]
```

```
Apply[Plus, xyl, {1, 2}]
```

```
{2, 4, 6, 8, 10, 12, 14, 16, 18, 20}
```

```
Apply[Plus, xyl, {0, 2}]
```

```
110
```

```
li = {{{0, 1}}, {{1, 1.5}}, {{2, 3}}}
```

```
{{{0, 1}}, {{1, 1.5}}, {{2, 3}}}
```

```
Apply[Flatten, li, {1}]
```

```
{{0, 1}, {1, 1.5}, {2, 3}}
```

```
Line[%]
```

```
Line[{{0, 1}, {1, 1.5}, {2, 3}}]
```

```
Apply[Point, li, {1}]
```

```
{Point[{0, 1}], Point[{1, 1.5}], Point[{2, 3}]}
```

There is an own infix expression for **Apply[*expr*, *list*, {1}]**, viz. **@@@**:

```
Point @@@ li
```

```
{Point[{0, 1}], Point[{1, 1.5}], Point[{2, 3}]}
```

```
Apply[Circle, {{{0, 0}, .5}, {{1, 0}, .4}, {{0, 1}, 2}}, {1}]
```

```
{Circle[{0, 0}, 0.5], Circle[{1, 0}, 0.4], Circle[{0, 1}, 2]}
```

```
cc = Circle @@@ {{{0, 0}, .5}, {{1, 0}, .4}, {{0, 1}, 2}}
```

```
{Circle[{0, 0}, 0.5], Circle[{1, 0}, 0.4], Circle[{0, 1}, 2]}
```

21.2 Applying Functions to Parts of Expressions (Map)

Map[f, {a,b,...}]	apply **f** to each element in a list, giving **{f[a], f[b], ...}**
Map[f, *expr*]	apply **f** to the first-level parts of ***expr***
f /@ *expr*	praefix form of **Map**
MapAll[f, *expr*]	apply **f** to all parts of ***expr***
f //@ *expr*	praefix form of MapAll

```
Map[f, a + b + c ]
```

```
f[a] + f[b] + f[c]
```

```
Map[f, {a,b,c} ]
```

```
{f[a], f[b], f[c]}
```

```
Map[f, a b c ]
```

f[a] f[b] f[c]

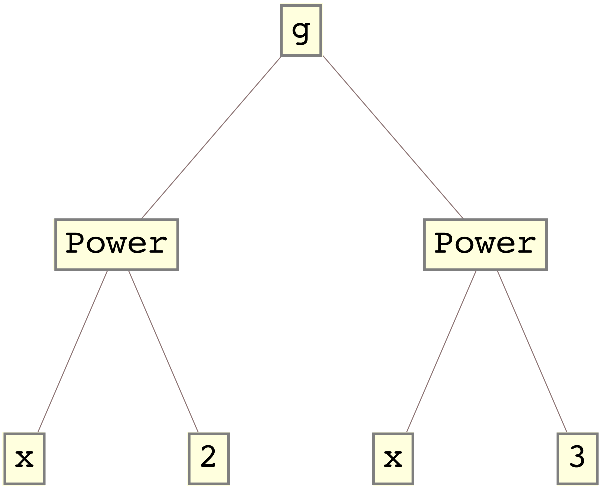
```
ls = {{0, 1}, {1, 1.5}, {2, 3}};
Map[Point, ls]
```

{Point[{0, 1}], Point[{1, 1.5}], Point[{2, 3}]}

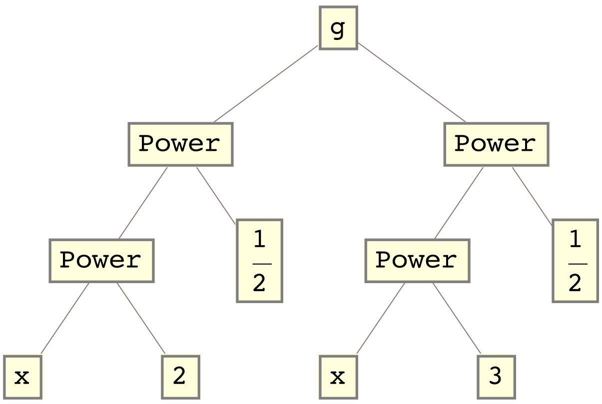
```
Map[Sqrt, g[x^2, x^3] ]
```

g[$\sqrt{x^2}$, $\sqrt{x^3}$]

```
TreeForm[g[x^2, x^3]]
```

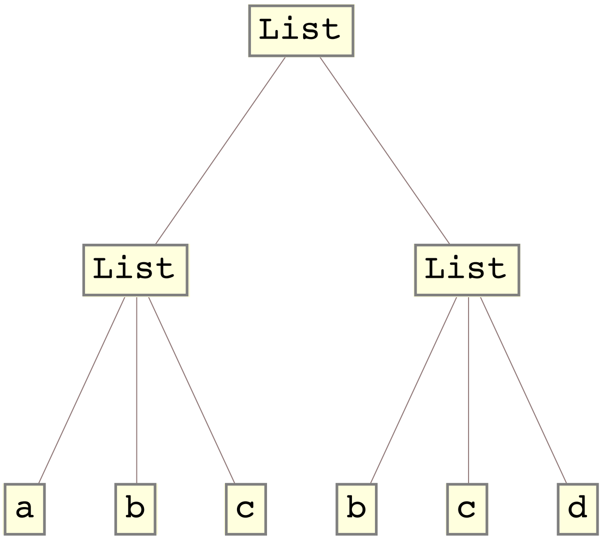



```
TreeForm[%%]
```




```
m = {{a,b,c},{b,c,d}};
```

```
TreeForm[m]
```




```
Map[f, m]
```

```
{f[{a, b, c}], f[{b, c, d}]}
```

```
Map[f, m, {2}]
```

```
{{f[a], f[b], f[c]}, {f[b], f[c], f[d]}}
```

```
MapAll[f, m]
```

```
f[{f[{f[a], f[b], f[c]}], f[{f[b], f[c], f[d]}]}]
```

```
f = Sqrt[3 x^(4j)+x^(12j)]/(x^(2j)Sqrt[3 + x^(8j)])
```

$$\frac{x^{-2 j} \sqrt{3 x^{4 j} + x^{12 j}}}{\sqrt{3 + x^{8 j}}}$$

```
PowerExpand[f]
```

$$\frac{x^{-2 j} \sqrt{3 x^{4 j} + x^{12 j}}}{\sqrt{3 + x^{8 j}}}$$

```
Factor[f]//PowerExpand
```

1

```
MapAll[Factor,f] //PowerExpand
```

1

f = $e^{-2 p \kappa - 2 q \kappa + z \kappa - zp \kappa}$ **;**

```
Factor[f]
```

$e^{-2 p \kappa - 2 q \kappa + z \kappa - zp \kappa}$

```
Map[Factor, f, {1}]
```

$e^{-(2 p + 2 q - z + zp) \kappa}$

```
ll = {{1, 2, 3, 4}, {5, 6}, {7, 8}, {9, 10}, {11, 12}};
```

```
Flatten[ll]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

```
l3 = Partition[%, 3]
```

{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}}

```
Reverse[l3]
```

{{10, 11, 12}, {7, 8, 9}, {4, 5, 6}, {1, 2, 3}}

```
Map[Reverse, %]
```

{{12, 11, 10}, {9, 8, 7}, {6, 5, 4}, {3, 2, 1}}

```
Map[Reverse, l3]
```

{{3, 2, 1}, {6, 5, 4}, {9, 8, 7}, {12, 11, 10}}

```
take2[list_] := Take[list, 2]
```

```
take2[{1,2,3}]
```

{1, 2}

```
take2[ {{1,2,3}, {4,5,6}, {7,8,9,11}} ]
```

{{1, 2, 3}, {4, 5, 6}}

```
take2[{1,2,3}, {4,5,6}]
```

take2[{1, 2, 3}, {4, 5, 6}]

```
Map[take2, {{1,2,3}, {5,6,7,}, {2,1,6,6}} ]
```

{{1, 2}, {5, 6}, {2, 1}}

```
take2[ {{1,2,3}, {5,6,7,}, {2,1,6,6}} ]
```

{{1, 2, 3}, {5, 6, 7, Null}}

```
FullForm[take2[list_]:= Take[list, 2] ]
```

Null

```
FullForm[take2[list_] ]
```

Pattern[list, Blank[]]

```
ne = 200;
```

```
vx = Array[x, ne];
```

```
ma = RandomReal[{1, ne}, {ne, ne}];
mb = RandomReal[{1, ne}, ne];
```

```
eq = ma.vx == mb // Thread;
```

```
so = Solve[eq, vx] // Flatten;
```

```
so[[{1, 2}]]
```

{x[1] → 2.1785, x[2] → 0.887553}

```
sx = Transpose[so /. Rule -> List][[2]];
```

```
sx[[{1, 2}]]
```

{2.1785, 0.887553}

so is a list containing the solutions as a substitution rule. **sx** contains only the numbers corresponding to the solutions. The operation performed below with **Map** runs much faster then by the common substitution command.

```
sxx = vx /. so // Timing;
```

```
sxx[[1]]
```

0.001321

```
sl  = Last /@ so // Timing;
sl[[1]]
```

0.000048

```
lx = {"x1", "x2", "x3", "x4", "x5"};
```

```
lc = Characters[lx]
```

{{x, 1}, {x, 2}, {x, 3}, {x, 4}, {x, 5}}

```
ld = lc /. "x" → "y"
```

{{y, 1}, {y, 2}, {y, 3}, {y, 4}, {y, 5}}

```
StringJoin[ld]
```

y1y2y3y4y5

```
Map[StringJoin, ld]
```

{y1, y2, y3, y4, y5}

```
Map[StringJoin, ld, {0}]
```

y1y2y3y4y5

```
mydata = RandomReal[{0, 1}, {3, 8}]
```

{{0.757396, 0.938414, 0.511784, 0.917408, 0.525658, 0.722486, 0.467552, 0.472751},
{0.308142, 0.250153, 0.113246, 0.130676, 0.339667, 0.748014, 0.334588, 0.738728},
{0.289612, 0.802057, 0.88714, 0.0348512, 0.530587, 0.227612, 0.560693, 0.335458}}

```
myfft = Map[Fourier, mydata] // Chop;
```

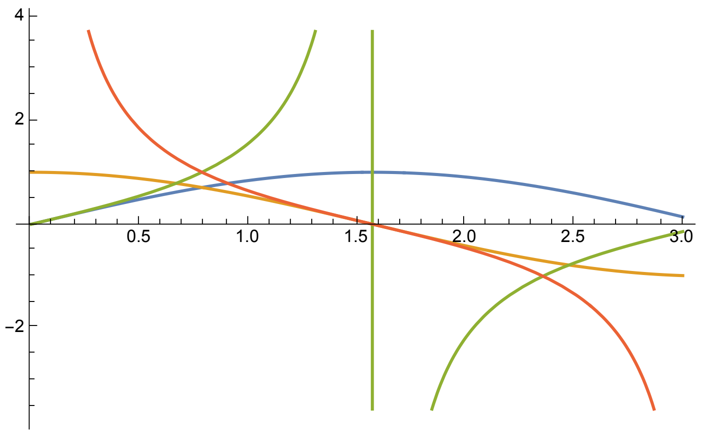
This does a FFT (= Fast Fourier Transform) on each row of the data.

```
myfft
```

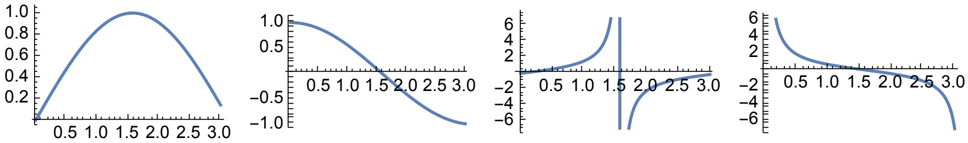
{{1.87859, 0.0247493 + 0.180785 ⅈ, 0.107381 + 0.0957216 ⅈ, 0.139114 + 0.149508 ⅈ,
-0.278837, 0.139114 - 0.149508 ⅈ, 0.107381 - 0.0957216 ⅈ, 0.0247493 - 0.180785 ⅈ},
{1.04765, 0.0164022 - 0.354734 ⅈ, 0.0707016 + 0.0455245 ⅈ,
-0.0386933 - 0.198222 ⅈ, -0.272918, -0.0386933 + 0.198222 ⅈ,
0.0707016 - 0.0455245 ⅈ, 0.0164022 + 0.354734 ⅈ},
{1.29684, 0.133565 + 0.183876 ⅈ, -0.221902 + 0.233119 ⅈ, -0.30396 - 0.0469567 ⅈ,
0.306904, -0.30396 + 0.0469567 ⅈ, -0.221902 - 0.233119 ⅈ, 0.133565 - 0.183876 ⅈ}}

Compare the action of **Map (= /@)** in the commands below:

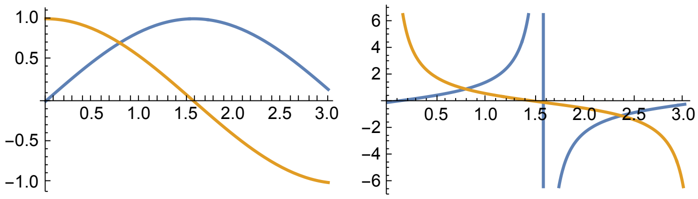
```
Plot[{Sin[x], Cos[x], Tan[x], Cot[x]}, {x, 0, 3}]
```



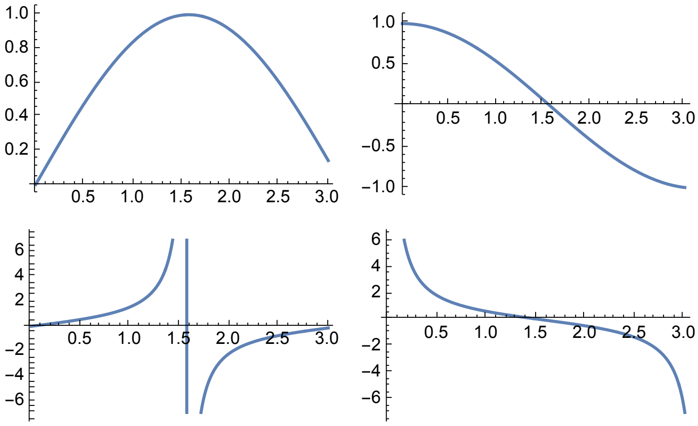
```
Show[GraphicsRow[(Plot[#1, {x, 0, 3}, DisplayFunction → Identity] &) /@
   {Sin[x], Cos[x], Tan[x], Cot[x]}],
 DisplayFunction → $DisplayFunction, ImageSize → 500]
```



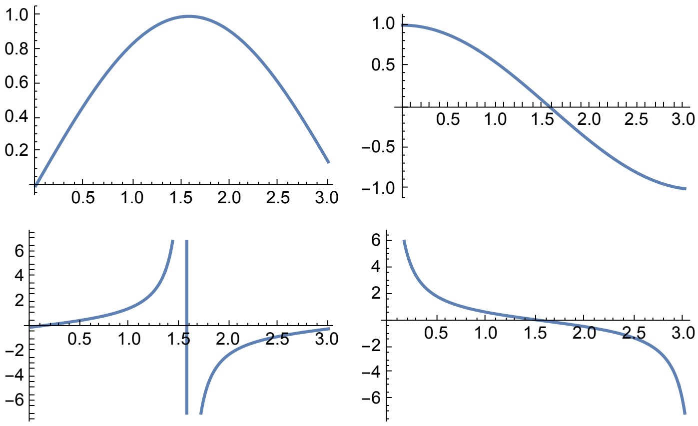
```
Show[GraphicsRow[(Plot[#1, {x, 0, 3}, DisplayFunction → Identity] &) /@
   {{Sin[x], Cos[x]}, {Tan[x], Cot[x]}}], DisplayFunction → $DisplayFunction]
```



```
Show[GraphicsGrid[
  {(Plot[#1, {x, 0, 3}, DisplayFunction → Identity] &) /@ {Sin[x], Cos[x]},
   (Plot[#1, {x, 0, 3}, DisplayFunction → Identity] &) /@ {Tan[x], Cot[x]}}],
 DisplayFunction → $DisplayFunction]
```



```
Show[GraphicsGrid[Partition[(Plot[#1, {x, 0, 3}, DisplayFunction → Identity] &) /@
    {Sin[x], Cos[x], Tan[x], Cot[x]}, 2]], DisplayFunction → $DisplayFunction]
```



This is the same picrture as above.

```
list = Table[RandomInteger[{1, 9}, 3], {5}]
```

```
{{8, 5, 2}, {3, 8, 8}, {6, 9, 9}, {7, 9, 5}, {8, 5, 7}}
```

For a function, let's say that we want the square root of the sum of

```
Clear[f]
f[{x_, y_, z_}] := Sqrt[x^2 + y^2 + z^2]
```

then just **Map** the function onto the elements of the list.

```
f /@ list
```

$\{\sqrt{93}, \sqrt{137}, 3\sqrt{22}, \sqrt{155}, \sqrt{138}\}$

```
Map[f, list]
```

$\{\sqrt{93}, \sqrt{137}, 3\sqrt{22}, \sqrt{155}, \sqrt{138}\}$

If your function is defined this way

```
Clear[g]
g[x_, y_, z_] := Sqrt[x^2 + y^2 + z^2]
```

then you have to **Apply** it to each triplet list and that function is mapped onto the list.

```
g @@ # & /@ list
```

$\{\sqrt{93}, \sqrt{137}, 3\sqrt{22}, \sqrt{155}, \sqrt{138}\}$

```
Map[Apply[g, #] &, list]
```

$\{\sqrt{93}, \sqrt{137}, 3\sqrt{22}, \sqrt{155}, \sqrt{138}\}$

```
g @@@ list
```

$\{\sqrt{93}, \sqrt{137}, 3\sqrt{22}, \sqrt{155}, \sqrt{138}\}$

MapAt[f, *expr*, *{part1, part2, ...}*]	Apply **f** to specified parts of ***expr***

```
Clear[a, b, c, d]
```

```
m = {{a, b, c}, {b, c, d}};
```

```
MapAt[f, m, {{1,2}, {2,3}} ]
```

```
{{a, f[b], c}, {b, c, f[d]}}
```

```
Position[m, b]
```

```
{{1, 2}, {2, 1}}
```

```
MapAt[f, m, %]
```

```
{{a, f[b], c}, {f[b], c, d}}
```

```
MapAt[f, {a,b,c,d}, { {2}, {3} } ]
```

```
{a, f[b], f[c], d}
```

```
t = 1 + (3 + x)^2 / x
```

$$1 + \frac{(3 + x)^2}{x}$$

```
FullForm[t]
```

```
Plus[1, Times[Power[x, -1], Power[Plus[3, x], 2]]]
```

```
MapAt[f, t, {{2, 1, 1}, {2, 2}} ]
```

$$1 + \frac{f\left[(3 + x)^2\right]}{f[x]}$$

Through[p[f_1, f_2]][x] gives **p[f_1[x], f_2[x]]**

Through serves to distribute an operand toseveral operators. More precisely:
Through distributes operators that appear inside the heads of expressions.

```
Through[(f + g + h)[x, y]]
```

```
f[x, y] + g[x, y] + h[x, y]
```

```
Through[{Re, Im}[(x + I y)^2]]
```

```
{Re[(x + i y)^2], Im[(x + i y)^2]}
```

```
ComplexExpand[%]
```

$\{x^2 - y^2, 2\,x\,y\}$

Scan[*f, expr*]	Evaluates the result of applying ***f*** to each element of ***expr*** without constructing a new expression
Scan[*f, expr, level*]	Same as above going down to given ***level***

```
sc = Scan[Print, {a, b, c}]
```

a

b

c

```
sc
```

```
sc = Scan[Print[#^3] &, {a, b, c}]
```

a^3

b^3

c^3

```
Scan[Print, a x^2 + b x + c]
```

c

b x

$a\,x^2$

```
Scan[Print, a x^2 + b x + c, 2]
```

c

b

x

b x

a

x^2

$a\,x^2$

```
?? Scan
```

Scan[*f*, *expr*] evaluates *f* applied to each element of *expr* in turn.
Scan[*f*, *expr*, *levelspec*] applies *f* to parts of *expr* specified by *levelspec*.
Scan[*f*] represents an operator form of Scan that can be applied to an expression. >>

```
Attributes[Scan] = {Protected}
```

```
Options[Scan] = {Heads → False}
```

By default, heads are not printed as shown in the examples above.

```
Scan[Print, {a, b}, Heads → True]
```

```
List
a
b
```

21.3 Pure Functions

Function[x, *body*]	a pure function in which **x** is replaced with any argument one provides
Function[{x1,x2,...}, *body*]	a pure function that takes several arguments
body &	a pure function in which arguments are specified as **#** or **#1,#2,#3**,... etc.

In the last type of command the ampersand **&** is obligatory.

When using functional operations as **Map** , one must specify a function to apply. In the examples above the "name" of a function was used to specify it. Pure functions allow one to give functions which can be applied to arguments, without having to define explicit names for the functions.

```
Clear[a,b,c,d,f,g,h,n,x]
h[x_] = f[x] + g[x]
f[x] + g[x]

Map[h, {a,b,c}]
{f[a] + g[a], f[b] + g[b], f[c] + g[c]}

Map[ f[#] + g[#] &, {a,b,c} ]
{f[a] + g[a], f[b] + g[b], f[c] + g[c]}

Function[x, x^2]
Function[x, x^2]

%[n]
n^2

Function[y^2, z^3]
Function::flpar: Parameter specification y^2 in Function[y^2, z^3] should be a symbol or a list of symbols >>
Function[y^2, z^3]
```

Parameter specification y^2 in Function[y^2, z^3] should be a symbol or a list of symbols. >>

```
Map[ Function[x, x^2], a + b + c ]
a^2 + b^2 + c^2

Map[ #^2 &, a + b + c ]
a^2 + b^2 + c^2

Map[ Take[#, 2] &, {{2,1,7},{4,1,5}, {3,1,2}} ]
{{2, 1}, {4, 1}, {3, 1}}
```

The postfix form of a command with several arguments may be realized with the help of a pure function.

```
Pi // N[#, 19] &
3.141592653589793238
```

The prefix form of a command may be realized with the help of a pure function.

```
list = RandomReal[{0, 1}, 5]
```

```
{0.730142, 0.285077, 0.936111, 0.416989, 0.362143}
```

```
Position[#, Max[#]] & @ list
```

```
{{3}}
```

Or the pure function may be treated as if it were a normal function, whose argument must stand between square brackets.

```
Position[#, Max[#]] &[list]
```

```
{{3}}
```

#0 is a rarely used “parameter,” which refers to the pure function itself and thus makes it possible to create
pure recursive functions:

```
fact = Function[If[#1 == 0, 1, #1 #0[#1 - 1]]] ;
```

```
fact[5]
```

```
120
```

But this works for nonnegative integers only; so safeguards are necessary.

```
fact = Function[If[Head[#1] == Integer && #1 ≥ 0,
    If[#1 == 0, 1, #1 #0[#1 - 1]], Print["Undefined"], Print["Undefined"]]] ;
```

```
fact[5]
```

```
120
```

```
fact[-2]
```

```
Undefined
```

```
fact[2.5]
```

```
Undefined
```

Tests whether a given function satisfies a differential equation mye be performed with the help of a pure function.

dg $= -y(x)\, k^2 + y''(x)$

$-k^2\, y[x] + y''[x]$

```
f = Exp[-k * #] &
```

```
Exp[-k #1] &
```

```
dg /. {y → f}
```

```
0
```

The pure function and `Map` may be used for removing factors in equations. In the example below the factor exp(i ωt) can be removed in an expression by simple multiplication. But in an equation this cannot be achieved in this simple way, but it can be done with `Map`.

```
Clear[a,b,c,ex,t,w]
ex = a E^(I w t ) (b + c x)
```

$a\, e^{i\, t\, w}\, (b + c\, x)$

```
 ex E^(- I w t)
```

$a\, (b + c\, x)$

```
eq = ex == 0
```

$a\, e^{i\, t\, w}\, (b + c\, x) == 0$

```
eq E^(- I w t) //Expand //Cancel
```

$e^{-i t w} \left(a e^{i t w} (b + c x) == 0\right)$

```
Map[Cancel[#/E^(I w t )]&, eq]
```

$a (b + c x) == 0$

Below all pairs where sin(x) >= 0 are taken from a list **l1** containing pairs of {cos(x), sin(x)} for x = n 2π/6, n ϵ ℕ.

```
l1 = Table[{Cos[k π / 3], Sin[k π / 3]}, {k, 6}] (* macher *)
```

$\left\{\left\{\frac{1}{2}, \frac{\sqrt{3}}{2}\right\}, \left\{-\frac{1}{2}, \frac{\sqrt{3}}{2}\right\}, \{-1, 0\}, \left\{-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right\}, \left\{\frac{1}{2}, -\frac{\sqrt{3}}{2}\right\}, \{1, 0\}\right\}$

```
Select[%, #[[2]] >= 0 &]
```

$\left\{\left\{\frac{1}{2}, \frac{\sqrt{3}}{2}\right\}, \left\{-\frac{1}{2}, \frac{\sqrt{3}}{2}\right\}, \{-1, 0\}, \{1, 0\}\right\}$

```
l4 = Cases[l1, {_,x_}/;N[x]>= 0,{1}]     (* Sakulin *)
```

$\left\{\left\{\frac{1}{2}, \frac{\sqrt{3}}{2}\right\}, \left\{-\frac{1}{2}, \frac{\sqrt{3}}{2}\right\}, \{-1, 0\}, \{1, 0\}\right\}$

```
Cases[1.22 A * x + 3 * b * y + Z^4, _Symbol, Infinity]
```

{A, x, b, y, Z}

```
Cases[1.22 A * x + 3 * b * y + Z^4, _?NumberQ, Infinity]
```

{1.22, 3, 4}

```
Cases[1.22 A * x + 3 * b * y + Z^4, _?IntegerQ, Infinity]
```

{3, 4}

```
Cases[1.22 A * x + 3 * b * y + Z^4, _Real, Infinity]
```

{1.22}

```
s1 = a u'[x] + a u''[x]
```

a u′[x] + a u″[x]

```
Cases[s1, Derivative[_][u][x], -1]
```

{u′[x], u″[x]}

```
s2 = a D[u[x, y], y] + c D[u[x, y], x, y]
```

$a\, u^{(0,1)}[x, y] + c\, u^{(1,1)}[x, y]$

```
FullForm[s2]
```

Plus[Times[a, Derivative[0, 1][u][x, y]], Times[c, Derivative[1, 1][u][x, y]]]

```
Cases[s2, Derivative[_][u][x, y], -1]
```

{}

```
Cases[s2, Derivative[__][u][x, y], -1]
```

$\left\{u^{(0,1)}[x, y], u^{(1,1)}[x, y]\right\}$

Above the argument of Derivative[] must contain two underscores since the list consists of several elements.

Below an index vector gives the order of the elements in a list of random numbers. The sorting is done with the number list.

```
n = 20;
b = RandomReal[{0, 1}, n]
```

```
{0.434101, 0.0672506, 0.724531, 0.379785, 0.730319, 0.179466,
 0.505932, 0.47035, 0.0889891, 0.162457, 0.29375, 0.267216, 0.779895,
 0.110323, 0.332028, 0.247772, 0.224051, 0.204674, 0.327716, 0.728271}
```

```
Range[n]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
```

```
iv = Sort[Range[n], b[[#1]] > b[[#2]] &]
```

```
{13, 5, 20, 3, 7, 8, 1, 4, 15, 19, 11, 12, 16, 17, 18, 6, 10, 14, 9, 2}
```

```
b[[iv]]
```

```
{0.779895, 0.730319, 0.728271, 0.724531, 0.505932, 0.47035,
 0.434101, 0.379785, 0.332028, 0.327716, 0.29375, 0.267216, 0.247772,
 0.224051, 0.204674, 0.179466, 0.162457, 0.110323, 0.0889891, 0.0672506}
```

```
Sort[b] // Reverse
```

```
{0.779895, 0.730319, 0.728271, 0.724531, 0.505932, 0.47035,
 0.434101, 0.379785, 0.332028, 0.327716, 0.29375, 0.267216, 0.247772,
 0.224051, 0.204674, 0.179466, 0.162457, 0.110323, 0.0889891, 0.0672506}
```

In the following example the list **data** is distributed over sublists according to the value of the first element of each pair; the intervals for this sublists are given in the list at the end of the **Select[]** command. The
Map[Union, ..., {1}] orders the sublists according to the value of the first element of each pair.

```
data = Table[{RandomReal[{0, 10}], x[i]}, {i, 15}]
```

```
{{4.03559, x[1]}, {3.28059, x[2]}, {1.18255, x[3]},
 {8.07051, x[4]}, {5.97835, x[5]}, {0.74464, x[6]}, {9.96614, x[7]},
 {4.97524, x[8]}, {2.12422, x[9]}, {1.11973, x[10]}, {4.21469, x[11]},
 {1.13396, x[12]}, {3.04189, x[13]}, {0.993193, x[14]}, {5.84686, x[15]}}
```

```
Map[Union,
 Select[data,
    Function[{sublist}, sublist[[1]] < #[[2]] && sublist[[1]] ≥ #[[1]]]] & /@
   {{0, 5}, {5, 7}, {7, 9}, {9, 10}}, {1}]
```

```
{{{0.74464, x[6]}, {0.993193, x[14]}, {1.11973, x[10]},
   {1.13396, x[12]}, {1.18255, x[3]}, {2.12422, x[9]}, {3.04189, x[13]},
   {3.28059, x[2]}, {4.03559, x[1]}, {4.21469, x[11]}, {4.97524, x[8]}},
 {{5.84686, x[15]}, {5.97835, x[5]}}, {{8.07051, x[4]}}, {{9.96614, x[7]}}}
```

```
teile = {5, 7, 9};
MapThread[Function[{low, high}, Select[Sort[data], low ≤ #[[1]] < high &]],
 {Prepend[teile, 0], Append[teile, 10]}]
```

```
{{{0.74464, x[6]}, {0.993193, x[14]}, {1.11973, x[10]},
   {1.13396, x[12]}, {1.18255, x[3]}, {2.12422, x[9]}, {3.04189, x[13]},
   {3.28059, x[2]}, {4.03559, x[1]}, {4.21469, x[11]}, {4.97524, x[8]}},
 {{5.84686, x[15]}, {5.97835, x[5]}}, {{8.07051, x[4]}}, {{9.96614, x[7]}}}
```

21.4 Exercises

21.1+ Define a function which computes the geometric mean of the numbers comprised in a list. The function must not contain loop commands.

21.2 The same as 21.1 for the harmonic mean.

21.3 + Transform the following operator (that of the spherical Bessel functions of order L) :

$$[d^2/dr^2 + (2/r)\, d/dr - L\,(L+1)/r^2 + k^2]\ u(r)$$

into that of Bessel' s differential equation

$$[d^2/dr^2 + (1/r)\, d/dr - L^2/r^2 + k^2]\ w(r)$$

by commands acting on lists.

21.4+ How can one flatten a list with multiple braces from the inside out, not from the outside in as is accomplished by **`Flatten[]`** ? For example, the list below

{ {{{a, b}}, {{c, d}}}, {{{p, q}}, {{r, s}}} }

should be transformed into:

```
{{a, b, c, d}, {p, q, r, s}};
```

or the list

```
{{{0.3, 0.5}}, {{0.6, 1.}}, {{0.9, 1.5}}, {{1.2, 2.}}}
```

into

```
{{0.3, 0.5}, {0.6, 1.}, {0.9, 1.5}, {1.2, 2.}} .
```

21.5 Transform the following list by replacing x with v in the names of the lists and of the elements:

lx = {x1,x2,x3,x4,x5} -> lv = {v1,v2,v3,v4,v5}.

These replacementrs should be done by working on lx, not by just rewriting it.

21.6+ Generate 2 lists. la contains cordinates of n 3D-points as elements:
la = {{ax1,ay1,az1}, {ax2,ay2,az2},...,{axn,ayn,azn}}.
lb is the same with a replaced by b everywhere. Interweave the 2 lists such that
ll = {{ax1,ay1,az1},{bx1,by1,bz1}, {ax2,ay2,az2},...,{axn,ayn,azn},{bxn,byn,bzn}}.

21.7 A list containing four sublists is given (in reality all the elements are numbers):

{{x1,y1,z1}, {{x2,y2,z2}, {{x3,y3,z3}, {{x4,y4,z4}}

1. The four sublists are regared as the coordinates of four points.
Transform the above list into that for the corresponding graphics command.
2. The four sublists are regared as the coordinates of four points of a line.
Transform the above list into that for the corresponding graphics command.

21.8 Two n x m matrices A = (a_{ik}) and U = (u_{ik}) are given. Find at least two different ways to get the system of equations, in which corresponding elements of U are set identical to those of A. At least one of these methods should work without loop commands.

21.9+ Four k x k matrices A,B,C,D are given. Form the 2k x 2k matrix given below without any loop commands: $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$.

21.10 Decompose $2^{67} + 1$ or the number given in ex.13.9 into prime factors.
Prepare a function which multiplies the factors again by acting directly on the list of prime factors.