Class 5 : Programming in Mathematica Part #3

Applying Functions Repeatedly

```
? Fold
```

Fold[*f*, *x*, *list*] gives the last element of FoldList[*f*, *x*, *list*]. >>

```
? FoldList
```

FoldList[*f*, *x*, {*a*, *b*, ...}] gives {*x*, *f*[*x*, *a*], *f*[*f*[*x*, *a*], *b*], ...}. >>

```
? Nest
```

Nest[*f*, *expr*, *n*] gives an expression with *f* applied *n* times to *expr*. >>

```
? NestList
```

NestList[*f*, *expr*, *n*] gives a list of the results of applying *f* to *expr* 0 through *n* times. >>

```
Nest[f, x, 4]
```

```
f[f[f[f[x]]]]
```

```
NestList[f, x, 4]
```

```
{x, f[x], f[f[x]], f[f[f[x]]], f[f[f[f[x]]]]}
```

```
recip[x_] := 1 / (1 + x)
```

```
Nest[recip, x, 3]
```

$$\frac{1}{1+\frac{1}{1+\frac{1}{1+x}}}$$

```
newton3[x_] := N[1 / 2 (x + 3 / x)]
```

```
NestList[newton3, 1.0, 5]
```

```
{1., 2., 1.75, 1.73214, 1.73205, 1.73205}
```

```
FixedPoint[newton3, 1.0]
```

```
1.73205
```

```
FixedPointList[newton3, 1.0]
```

```
{1., 2., 1.75, 1.73214, 1.73205, 1.73205, 1.73205}
```

```
divide2[n_] := n / 2
```

```
NestWhileList[divide2, 123 456, EvenQ]
```

{123 456, 61 728, 30 864, 15 432, 7716, 3858, 1929}

```
NestWhileList[newton3, 1.0, Unequal, 2]
```

{1., 2., 1.75, 1.73214, 1.73205, 1.73205, 1.73205}

```
NestWhileList[Mod[5 #, 7] &, 1, Unequal, All]
```

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

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

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

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

f[f[f[x, a], b], c]

```
FoldList[Plus, 0, {a, b, c}]
```

{0, a, a + b, a + b + c}

Testing and Searching List Elements

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

{{1}, {4}}

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

2

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

True

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

False

```
m = IdentityMatrix[3]
```

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

```
FreeQ[m, 0]
```

False

```
Position[m, 0]
```

{{1, 2}, {1, 3}, {2, 1}, {2, 3}, {3, 1}, {3, 2}}

Building Lists from Functions

```
Array[p, 5]
```

{p[1], p[2], p[3], p[4], p[5]}

```
Table[p[i], {i, 5}]
```

{p[1], p[2], p[3], p[4], p[5]}

```
Map[p, Range[5]]
```

{p[1], p[2], p[3], p[4], p[5]}

```
p[x_] := p^2
Array[p, 5]
Table[p[i], {i, 5}]
Map[p, Range[5]]
```

$\{p^2, p^2, p^2, p^2, p^2\}$

$\{p^2, p^2, p^2, p^2, p^2\}$

$\{p^2, p^2, p^2, p^2, p^2\}$

```
Array[# + #^2 &, 5]
```

{2, 6, 12, 20, 30}

```
Array[m, {2, 3}]
```

{{m[1, 1], m[1, 2], m[1, 3]}, {m[2, 1], m[2, 2], m[2, 3]}}

```
Array[Plus[##]^2 &, {3, 3}]
```

{{4, 9, 16}, {9, 16, 25}, {16, 25, 36}}

```
NestList[D[#, x] &, x^n, 3]
```

$\{x^n, n x^{-1+n}, (-1+n) n x^{-2+n}, (-2+n) (-1+n) n x^{-3+n}\}$

Selecting Parts of Expressions with Functions

```
Select[{2, 15, 1, a, 16, 17}, # > 4 &]
```

{15, 16, 17}

```
t = Expand[(x + y + z)^2]
```

$x^2 + 2 x y + y^2 + 2 x z + 2 y z + z^2$

```
Select[t, FreeQ[#, x] &]
```

$y^2 + 2 y z + z^2$

```
Select[{-1, 3, 10, 12, 14}, # > 3 &, 1]
```

{10}

Everything Is an Expression

```
x + y + z
```

x + y + z

```
FullForm[%]
```

Plus[x, y, z]

```
1 + x^2 + (y + z)^2
```

$1 + x^2 + (y + z)^2$

```
FullForm[%]
```

```
Plus[1, Power[x, 2], Power[Plus[y, z], 2]]
```

```
Head[f[x, y]]
```

```
f
```

```
? Head
```

Head[*expr*] gives the head of *expr*. ≫

```
Head[a + b + c]
```

```
Plus
```

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

```
List
```

```
Head[23 432]
```

```
Integer
```

```
Head[345.6]
```

```
Real
```

```
Part[{a, b, c}, 0]
```

```
List
```

```
Part[{a, b, c}, 1]
```

```
a
```

```
Part[{a, b, c}, 3]
```

```
c
```

```
Part[{a, b, c}, 4]
```

Part::partw : Part 4 of {a, b, c} does not exist. ≫

```
{a, b, c}〚4〛
```

```
Part[23 242, 0]
```

```
Integer
```

```
Part[23 242, 1]
```

Part::partd : Part specification 23242〚1〛 is longer than depth of object. ≫

```
23 242〚1〛
```

Immediate and Delayed Definitions

```
ex[x_] := Expand[(1 + x) ^2]
```

```
? ex
```

```
Global`ex
```

```
ex[x_] := Expand[(1 + x)^2]
```

```
iex[x_] = Expand[(1 + x)^2]
```

$1+2x+x^2$

```
?iex
```

Global`iex

iex[x_] = $1+2x+x^2$

```
ex[y + 2]
```

$9+6y+y^2$

```
iex[y + 2]
```

$1+2(2+y)+(2+y)^2$

```
r1 = Random[]
```

0.705655

```
r2 := Random[]
```

```
{r1, r2}
```

{0.705655, 0.0128323}

```
{r1, r2}
```

{0.705655, 0.270484}

Applying Transformation Rules

```
x + y /. x -> 3
```

3 + y

```
x + y /. {x -> a, y -> b}
```

a + b

```
x + y /. {{x -> 1, y -> 2}, {x -> 4, y -> 2}}
```

{3, 6}

```
Solve[x^3 - 5 x^2 + 2 x + 8 == 0, x]
```

{{x → -1}, {x → 2}, {x → 4}}

```
x^2 + 6 /. %
```

{7, 10, 22}

```
{x^2, x^3, x^4} /. {x^3 -> u, x^n_ -> p[n]}
```

{p[2], u, p[4]}

```
h[x + h[y]] /. h[u_] -> u^2
```

$(x+h[y])^2$

```
h[x + h[y]] //. h[u_] -> u^2
```

$(x+y^2)^2$

```
{x^2, y^3} /. {x -> y, y -> x}
```

$\{y^2, x^3\}$

```
x^2 /. {x -> (1 + y), y -> b}
```

$(1+y)^2$

```
x^2 /. x -> (1 + y) /. y -> b
```

$(1+b)^2$

```
x^2 + y^6 /. {x -> 2 + a, a -> 3}
```

$(2+a)^2 + y^6$

```
x^2 + y^6 //. {x -> 2 + a, a -> 3}
```

$25 + y^6$

```
log[a b c d] /. log[x_ y_] -> log[x] + log[y]
```

log[a] + log[b c d]

```
log[a b c d] //. log[x_ y_] -> log[x] + log[y]
```

log[a] + log[b] + log[c] + log[d]

Functions as Procedures

```
exprod[n_] := Expand[Product[x + i, {i, 1, n}]]
```

```
exprod[5]
```

$120 + 274\,x + 225\,x^2 + 85\,x^3 + 15\,x^4 + x^5$

```
cex[n_, i_] := (t = exprod[n]; Coefficient[t, x^i])
```

```
cex[5, 3]
```

85

```
t
```

$120 + 274\,x + 225\,x^2 + 85\,x^3 + 15\,x^4 + x^5$

```
ncex[n_, i_] := Module[{u}, u = exprod[n]; Coefficient[u, x^i]]
```

```
ncex[5, 3]
```

85

```
u
```

u

Special Forms of Assignment

```
t = 7 x
```

7 x

```
t += 18 x

25 x

t

25 x

t = 8; t *= 7; t

56

i = 5; Print[i++]; Print[i]

5

6

i = 5; Print[++i]; Print[i]

6

6

{x, y} = {5, 8}

{5, 8}

{x, y} = {y, x}

{8, 5}

x

8

y

5

{a, b, c} = {1, 2, 3}; {b, a, c} = {a, c, b}; {a, b, c}

{3, 1, 2}

v = {5, 7, 9}

{5, 7, 9}

Append[v, 11]

{5, 7, 9, 11}

v

{5, 7, 9}

AppendTo[v, 11] (*This is like doing v=Append[v,11] *)

{5, 7, 9, 11}

v

{5, 7, 9, 11}
```

Conditionals

```
If[7 > 8, x, y]
```

y

```
If[7 > 8, Print[x], Print[y]]
```

y

When you write programs in *Mathematica*,
you will often have a choice between making a single definition whose right-hand side
 involves several branches controlled by If functions, or making several definitions,
each controlled by an appropriate /; condition. By using several definitions,
you can often produce programs that are both clearer, and easier to modify.

```
?/;
```

patt /; *test* is a pattern which matches only if the evaluation of *test* yields True.
lhs :> *rhs* /; *test* represents a rule which applies only if the evaluation of *test* yields True.
lhs := *rhs* /; *test* is a definition to be used only if *test* yields True. ≫

```
f[x_] := If[x > 0, 1, -1]
g[x_] := 1 /; x > 0
g[x_] := -1 /; x <= 0
?g
```

Global`g

g[x_] := 1 /; x > 0

g[x_] := -1 /; x ≤ 0

```
?Switch
```

Switch[*expr*, $form_1$, $value_1$, $form_2$, $value_2$, …] evaluates *expr*, then compares it with each of
the $form_i$ in turn, evaluating and returning the $value_i$ corresponding to the first match found. ≫

```
?Which
```

Which[$test_1$, $value_1$, $test_2$, $value_2$, …] evaluates each of the $test_i$ in
turn, returning the value of the $value_i$ corresponding to the first one that yields True. ≫

```
h[x_] := Which[x < 0, x^2, x > 5, x^3, True, 0]
h[-5]
```

25

```
h[2]
```

0

```
r[x_] := Switch[Mod[x, 3], 0, a, 1, b, 2, c]
```

```
r[7]
```

```
b
```

```
Switch[17, 0, a, 1, b, _, q]
```

```
q
```

An important point about symbolic systems such as *Mathematica* is that the conditions you give may yield neither True nor False. Thus, for example, the condition x == y does not yield True or False unless x and y have specific values, such as numerical ones.

```
If[x == y, a, b]
```

```
If[x == y, a, b]
```

```
If[x == y, a, b, c]
```

```
c
```

```
TrueQ[x == y]
```

```
False
```

```
x === y
```

```
False
```

```
If[x === y, a, b]
```

```
b
```

```
If[TrueQ[x == y], a, b]
```

```
b
```

The main difference between *lhs* === *rhs* and *lhs* == *rhs* is that === always returns True or False, whereas == can leave its input in symbolic form, representing a symbolic equation, as discussed in "Equations". You should typically use === when you want to test the *structure* of an expression, and == if you want to test mathematical equality. The *Mathematica* pattern matcher effectively uses === to determine when one literal expression matches another.

```
Head[a + b + c] === Times
```

```
False
```

```
Head[a + b + c] == Times
```

```
Plus == Times
```

```
t[x_] := (x != 0 && 1 / x < 3)
```

```
t[2]
```

```
True
```

```
t[0]
```

```
False
```

The way that Mathematica evaluates logical expressions allows you to combine sequences of tests where later tests may make sense only if the earlier ones are satisfied. The behavior, which is analogous to that found in languages such as C, is

convenient in constructing many kinds of Mathematica programs.

Loops and Control Structures

```
?Do
```

Do[*expr*, {i_{max}}] evaluates *expr* i_{max} times.
Do[*expr*, {*i*, i_{max}}] evaluates *expr* with the variable *i* successively taking on the values 1 through i_{max} (in steps of 1).
Do[*expr*, {*i*, i_{min}, i_{max}}] starts with $i = i_{min}$.
Do[*expr*, {*i*, i_{min}, i_{max}, *di*}] uses steps *di*.
Do[*expr*, {*i*, {i_1, i_2, …}}] uses the successive values i_1, i_2, ….
Do[*expr*, {*i*, i_{min}, i_{max}}, {*j*, j_{min}, j_{max}}, …] evaluates *expr* looping over different values of *j*, etc. for each *i*. »

```
?For
```

For[*start*, *test*, *incr*, *body*] executes *start*, then repeatedly evaluates *body* and *incr* until *test* fails to give True. »

```
?While
```

While[*test*, *body*] evaluates *test*, then *body*, repetitively, until *test* first fails to give True. »

```
Do[Print[i^2], {i, 4}]
```

```
1
4
9
16
```

```
t = x; Do[t = 1/(1 + k t), {k, 2, 12, 2}]; t
```

$$\frac{1}{1+\frac{12}{1+\frac{10}{1+\frac{8}{1+\frac{6}{1+\frac{4}{1+2x}}}}}}$$

```
n = 17; While[(n = Floor[n/2]) != 0, Print[n]]
```

```
8
4
2
1
```

```
For[i = 1, i < 7, i++, Print[i]]
```

```
1
2
3
4
5
6
```

```
For[i = 1; t = x, i^2 < 10, i++, t = t^2 + i; Print[t]]
```

$1 + x^2$

$2 + \left(1 + x^2\right)^2$

$3 + \left(2 + \left(1 + x^2\right)^2\right)^2$

```
While[False, Print[x]]
```

```
t = 1; Do[t *= k; Print[t]; If[t > 19, Break[]], {k, 10}]
```

1

2

6

24

```
t = 1; Do[t *= k; Print[t]; If[k < 3, Continue[]]; t += 2, {k, 10}]
```

1

2

6

32

170

1032

7238

57 920

521 298

5 213 000

```
f[x_] := (If[x > 5, Return[big]]; t = x^3; Return[t - 7])
```

```
f[10]
```

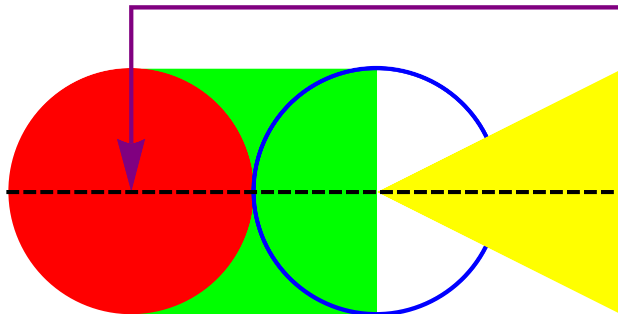
big

The Structure of Graphics

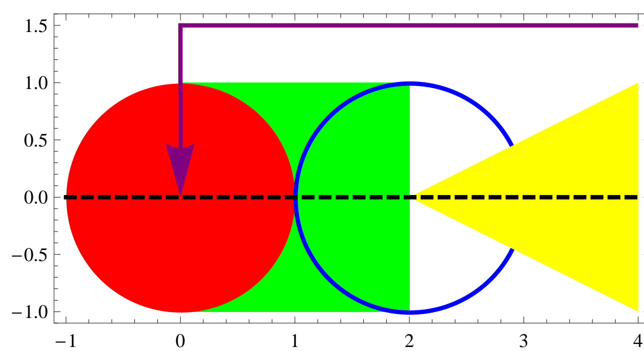
```
?Graphics
```

Graphics[*primitives*, *options*] represents a two-dimensional graphical image. ≫

```
Graphics[{Thick, Green, Rectangle[{0, -1}, {2, 1}], Red, Disk[], Blue, Circle[{2, 0}],
  Yellow, Polygon[{{2, 0}, {4, 1}, {4, -1}}], Purple, Arrowheads[Large],
  Arrow[{{4, 3/2}, {0, 3/2}, {0, 0}}], Black, Dashed, Line[{{-1, 0}, {4, 0}}]}]
```



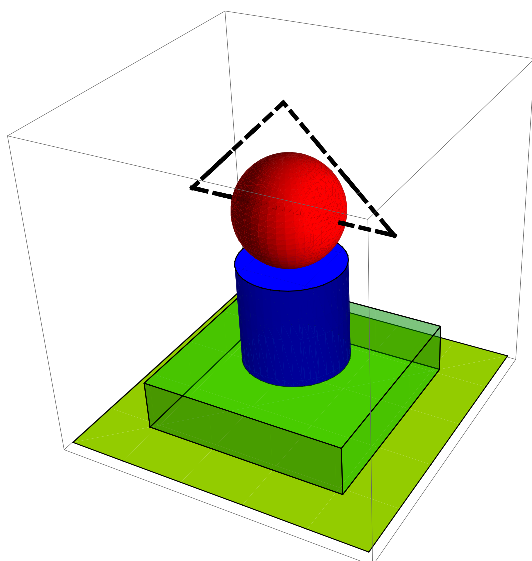
```
Show[%, Frame -> True]
```



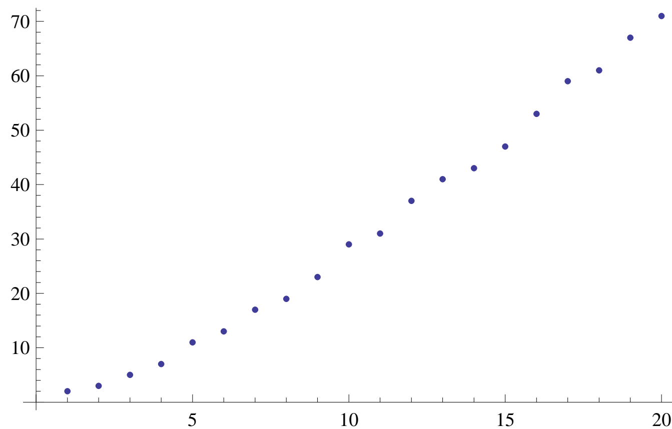

```
?Graphics3D
```

Graphics3D[*primitives*, *options*] represents a three-dimensional graphical image. ≫

```
Graphics3D[{Blue, Cylinder[], Red, Sphere[{0, 0, 2}], Black,
  Thick, Dashed, Line[{{-2, 0, 2}, {2, 0, 2}, {0, 0, 4}, {-2, 0, 2}}],
  Yellow, Polygon[{{-3, -3, -2}, {-3, 3, -2}, {3, 3, -2}, {3, -3, -2}}],
  Green, Opacity[.3], Cuboid[{-2, -2, -2}, {2, 2, -1}]}]
```



```
ListPlot[Table[Prime[n], {n, 20}]]
```



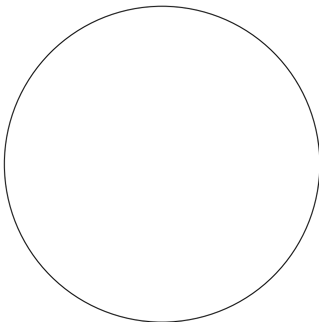

```
InputForm[%]
```

```
Graphics[{{{}, {Hue[0.67, 0.6, 0.6], Point[{{1., 2.},
     {2., 3.}, {3., 5.}, {4., 7.}, {5., 11.}, {6., 13.},
     {7., 17.}, {8., 19.}, {9., 23.}, {10., 29.}, {11.,
     31.}, {12., 37.}, {13., 41.}, {14., 43.}, {15.,
     47.}, {16., 53.}, {17., 59.}, {18., 61.}, {19.,
     67.}, {20., 71.}}]}, {}}},
 {AspectRatio -> GoldenRatio^(-1), Axes -> True,
  AxesOrigin -> {0, 0}, PlotRange ->
   {{0., 20.}, {0., 71.}}, PlotRangeClipping -> True,
  PlotRangePadding -> {Scaled[0.02], Scaled[0.02]}}]
```

```
Graphics[Circle[]];
2 + 2
```

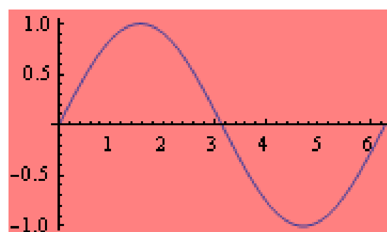
4

```
Print[Graphics[Circle[]]];
2 + 2
```



4

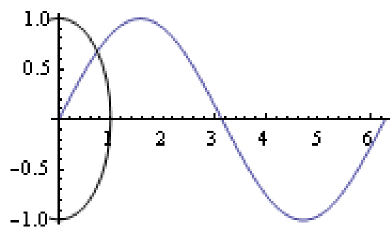
```
g1 = Plot[Sin[x], {x, 0, 2 Pi}];
Show[g1, Background -> Pink]
```



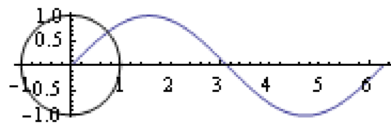
```
? Show
```

Show[*graphics*, *options*] shows graphics with the specified options added.
Show[g_1, g_2, …] shows several graphics combined. »

```
Show[{g1, Graphics[Circle[]]}]
```



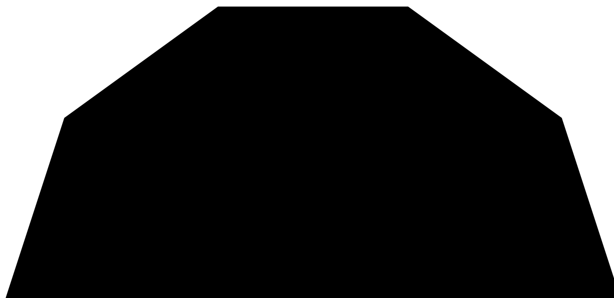
```
Show[{g1, Graphics[Circle[]]}, PlotRange -> All, AspectRatio -> Automatic]
```



```
? Polygon
```

Polygon[{pt_1, pt_2, …}] is a graphics primitive that represents a filled polygon.
Polygon[{{pt_{11}, pt_{12}, …}, {pt_{21}, …}, …}] represents a collection of polygons. »

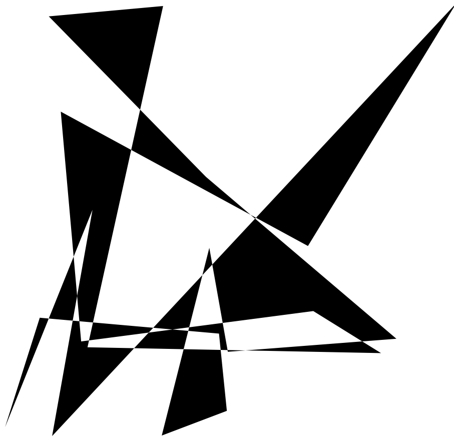
```
poly = Polygon[Table[N[{Cos[n Pi / 5], Sin[n Pi / 5]}], {n, 0, 5}]];
Graphics[poly]
```



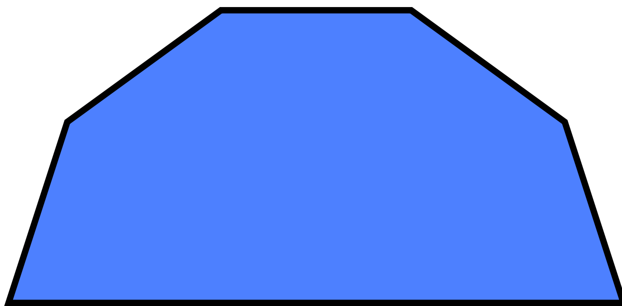
```
InputForm[%]
```

```
Graphics[Polygon[{{1., 0.}, {0.8090169943749475,
    0.5877852522924731}, {0.30901699437494745, 0.9510565162951535},
   {-0.30901699437494745, 0.9510565162951535},
   {-0.8090169943749475, 0.5877852522924731}, {-1., 0.}}]]
```

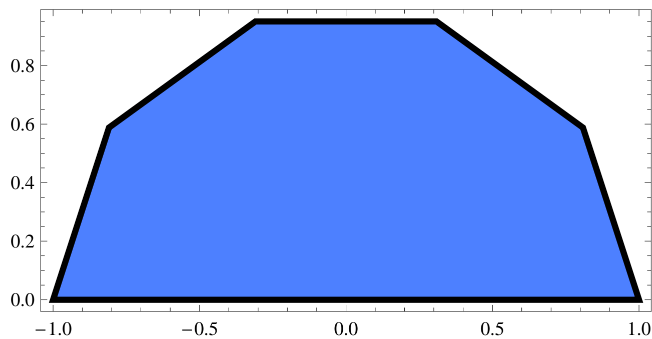
```
Graphics[Polygon[Table[{Random[], Random[]}, {20}]]]
```



```
Graphics[{RGBColor[0.3, 0.5, 1], EdgeForm[Thickness[0.01]], poly}]
```



```
Show[%, Frame -> True]
```



Two - Dimensional Graphics Elements

```
?Point
```

Point[*coords*] is a graphics primitive that represents a point.
Point[{$coords_1$, $coords_2$, …}] represents a collection of points. ≫

```
?Line
```

Line[{pt_1, pt_2, …}] is a graphics primitive which represents a line joining a sequence of points.
Line[{{pt_{11}, pt_{12}, …}, {pt_{21}, …}, …}] represents a collection of lines. ≫

```
?Rectangle
```

Rectangle[{x_{min}, y_{min}}, {x_{max}, y_{max}}] is a two-dimensional graphics primitive that represents a filled rectangle, oriented parallel to the axes.
Rectangle[{x_{min}, y_{min}}] corresponds to a unit square. »

```
?Polygon
```

Polygon[{pt_1, pt_2, …}] is a graphics primitive that represents a filled polygon.
Polygon[{{pt_{11}, pt_{12}, …}, {pt_{21}, …}, …}] represents a collection of polygons. »

```
?Circle
```

Circle[{x, y}, r] is a two-dimensional graphics primitive that represents a circle of radius r centered at the point x, y.
Circle[{x, y}] gives a circle of radius 1.
Circle[{x, y}, r, {θ_1, θ_2}] gives a circular arc.
Circle[{x, y}, {r_x, r_y}] gives an ellipse with semi-axes of lengths r_x and r_y, oriented parallel to the coordinate axes. »

```
?Disk
```

Disk[{x, y}, r] is a two-dimensional graphics primitive that represents a filled disk of radius r centered at the point x, y.
Disk[{x, y}] gives a disk of radius 1.
Disk[{x, y}, r, {θ_1, θ_2}] gives a segment of a disk.
Disk[{x, y}, {r_x, r_y}] gives an elliptical disk with semi-axes of lengths r_x and r_y, oriented parallel to the coordinate axes. »

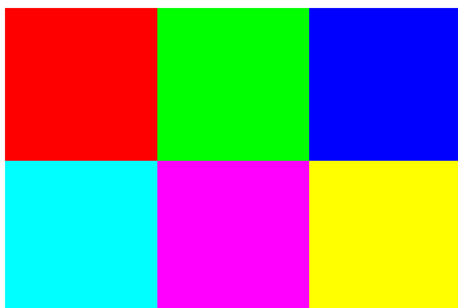
```
?Text
```

Text[*expr*] displays with *expr* in plain text format.
Text[*expr*, *coords*] is a graphics primitive that displays the textual form of *expr* centered at the point specified by *coords*. »

```
?Raster
```

Raster[{{a_{11}, a_{12}, …}, …}] is a two-dimensional graphics primitive which represents a rectangular array of gray cells.
Raster[{{{r_{11}, g_{11}, b_{11}}, …}, …}] represents an array of RGB color cells.
Raster[{{{r_{11}, g_{11}, b_{11}, α_{11}}, …}, …}] represents an array of color cells with opacity α_{ij}.
Raster[{{{a_{11}, α_{11}}, …}, …}] represents an array of gray cells with the specified opacities. »

```
Graphics[
 Raster[{{{0, 1, 1}, {1, 0, 1}, {1, 1, 0}}, {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}}]]
```

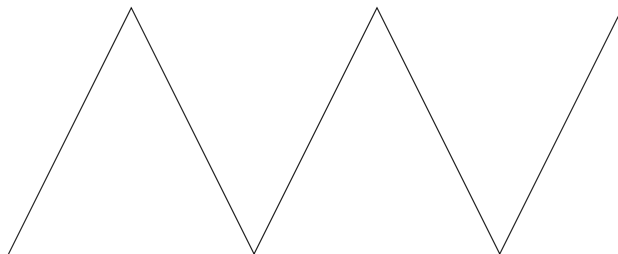


```
rgb = Reverse[ExampleData[{"TestImage", "Lena"}, "Data"] / 255.];
```

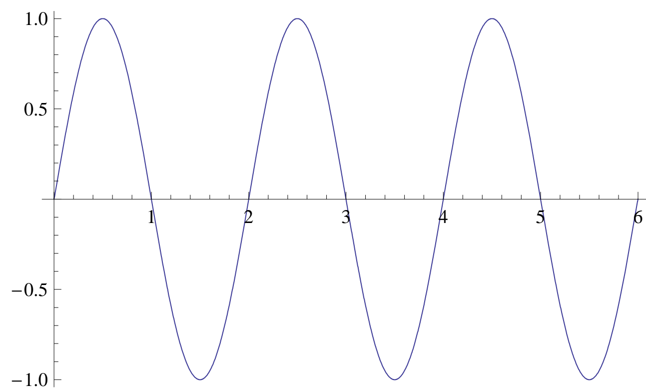
```
Graphics[Raster[rgb]]
```

```
gray = Map[{.3, .59, .11}.# &, rgb, {2}];

ker = {{1, 0, -1}, {1, 0, -1}, {1, 0, -1}};

f = ListCorrelate[ker, gray];

Graphics[Raster[f, Automatic, {Min[f], Max[f]} / 4, ColorFunction -> "TemperatureMap"]]
```

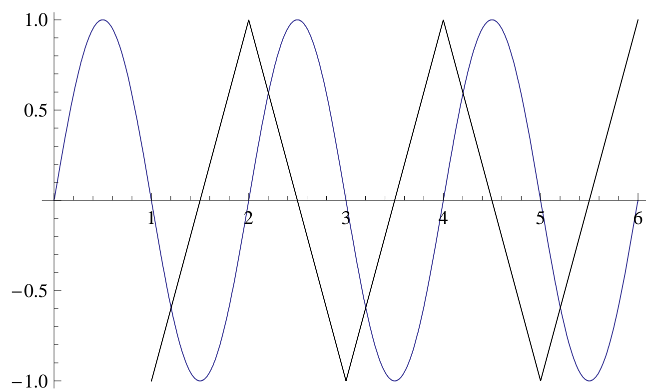
```
sawline = Line[Table[{n, (-1)^n}, {n, 6}]]

Line[{{1, -1}, {2, 1}, {3, -1}, {4, 1}, {5, -1}, {6, 1}}]

sawgraph = Graphics[sawline]
```



```
Plot[Sin[Pi x], {x, 0, 6}]
```



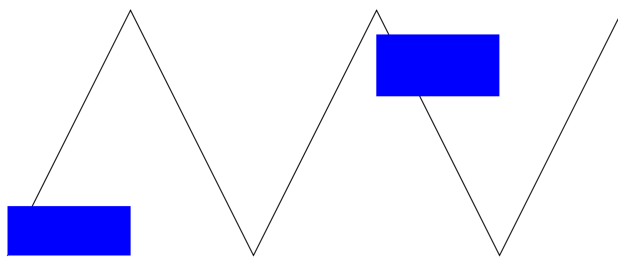

```
Show[%, sawgraph]
```




```
{Blue, Rectangle[{1, -1}, {2, -0.6}], Rectangle[{4, .3}, {5, .8}]}
```

```
{RGBColor[0, 0, 1], Rectangle[{1, -1}, {2, -0.6}], Rectangle[{4, 0.3}, {5, 0.8}]}
```

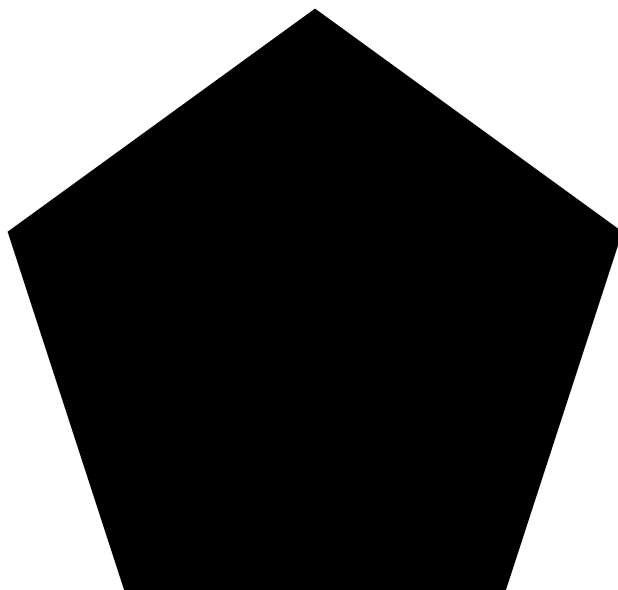
```
Graphics[{sawline, %}]
```



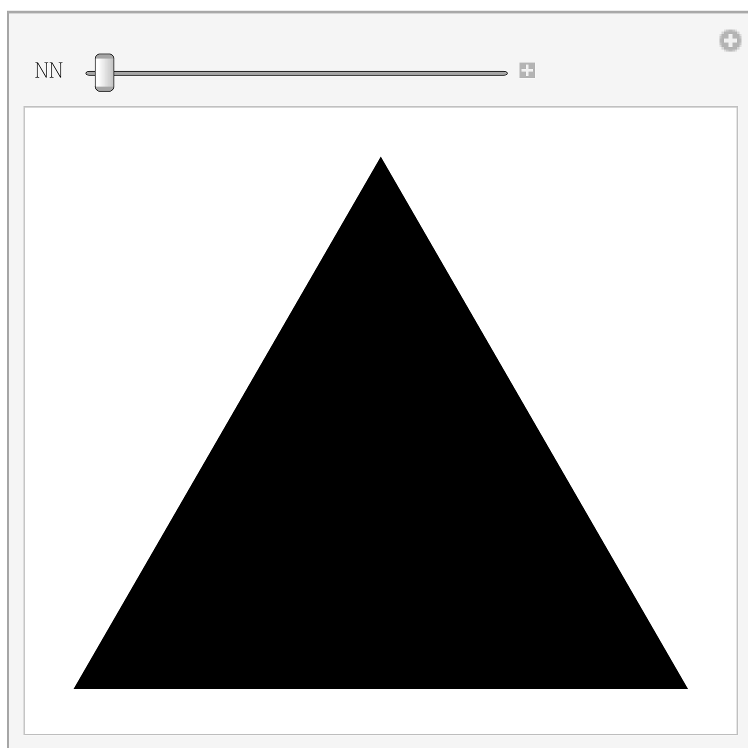
```
pentagon = Table[{Sin[2 Pi n / 5], Cos[2 Pi n / 5]}, {n, 5}]
```

$$\left\{\left\{\sqrt{\frac{5}{8}+\frac{\sqrt{5}}{8}}, \frac{1}{4}\left(-1+\sqrt{5}\right)\right\}, \left\{\sqrt{\frac{5}{8}-\frac{\sqrt{5}}{8}}, \frac{1}{4}\left(-1-\sqrt{5}\right)\right\}, \left\{-\sqrt{\frac{5}{8}-\frac{\sqrt{5}}{8}}, \frac{1}{4}\left(-1-\sqrt{5}\right)\right\}, \left\{-\sqrt{\frac{5}{8}+\frac{\sqrt{5}}{8}}, \frac{1}{4}\left(-1+\sqrt{5}\right)\right\}, \{0, 1\}\right\}$$

```
Graphics[Polygon[pentagon]]
```



```
Manipulate[
 Graphics[Polygon[Table[{Sin[2 Pi n / NN], Cos[2 Pi n / NN]}, {n, NN}]]], {NN, 3, 20, 1}]
```

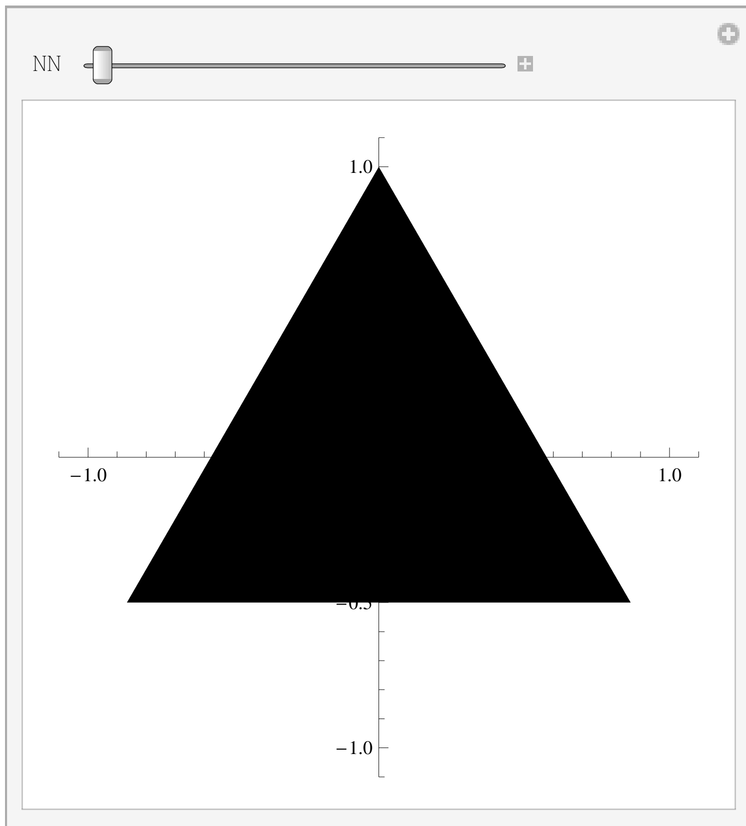



```
?PlotRange
```

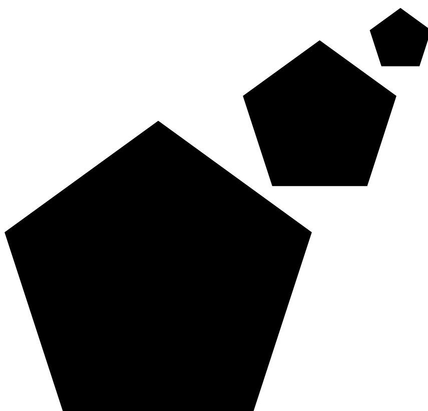
PlotRange is an option for graphics functions that specifies what range of coordinates to include in a plot. ≫

$\{\{x_{min}, x_{max}\}, \{y_{min}, y_{max}\}\}$

```
Manipulate[Graphics[Polygon[Table[{Sin[2 Pi n / NN], Cos[2 Pi n / NN]}, {n, NN}]],
  PlotRange -> {{-1.1, 1.1}, {-1.1, 1.1}}, Axes -> True], {NN, 3, 20, 1}]
```




```
Graphics[Polygon[{pentagon, 1 + .5 pentagon, 1.5 + .2 pentagon}]]
```



```
? Translate
```

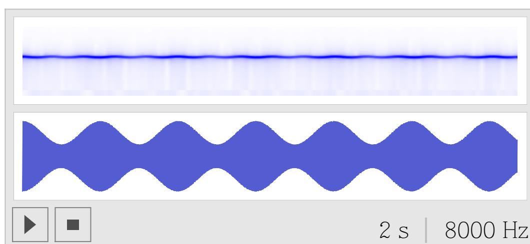
Translate[g, {x, y, …}] represents graphics primitives g translated by the vector {x, y, …}. ≫

```
?Rotate
```

Rotate[g, θ] represents 2D graphics primitives g
rotated counterclockwise by θ radians about the center of their bounding box.
Rotate[g, θ, {x, y}] rotates 2D graphics primitives about the point {x, y}.
Rotate[g, θ, w] rotates 3D graphics primitives by θ radians around the 3D vector w anchored at the origin.
Rotate[g, θ, w, p] rotates around the 3D vector w anchored at p.
Rotate[g, {u, v}] rotates around the origin transforming the 3D vector u to v.
Rotate[g, θ, {u, v}] rotates by angle θ in the plane spanned by 3D vectors u and v. ≫

The Representation of Sound

```
Play[(2 + Cos[20 t]) * Sin[3000 t + 2 Sin[50 t] ], {t, 0, 2}]
```



There is so much more to learn but now lets move on to demonstrations ...