# GRAPHICS

## Three-dimensional plots
## Table
## List
## List plots

## Three Dimensional Plotting

For creating three dimensional plots, use

**Plot3D**[ ]

The arguments of **Plot3D**[ ] are similar to those of the function **Plot**[ ].

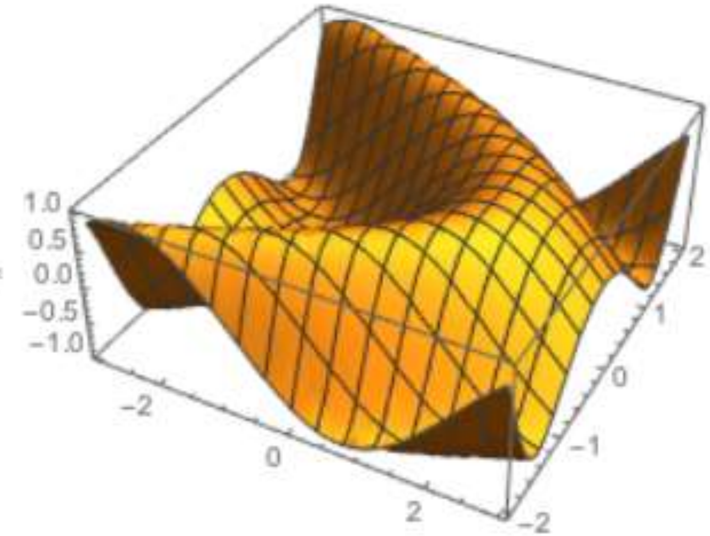
Plot3D[$f$, {$x$, $x_{min}$, $x_{max}$}, {$y$, $y_{min}$, $y_{max}$}]
generates a three-dimensional plot of $f$ as a function of $x$ and $y$.

Plot3D[{$f_1$, $f_2$, ...}, {$x$, $x_{min}$, $x_{max}$}, {$y$, $y_{min}$, $y_{max}$}]
plots several functions.

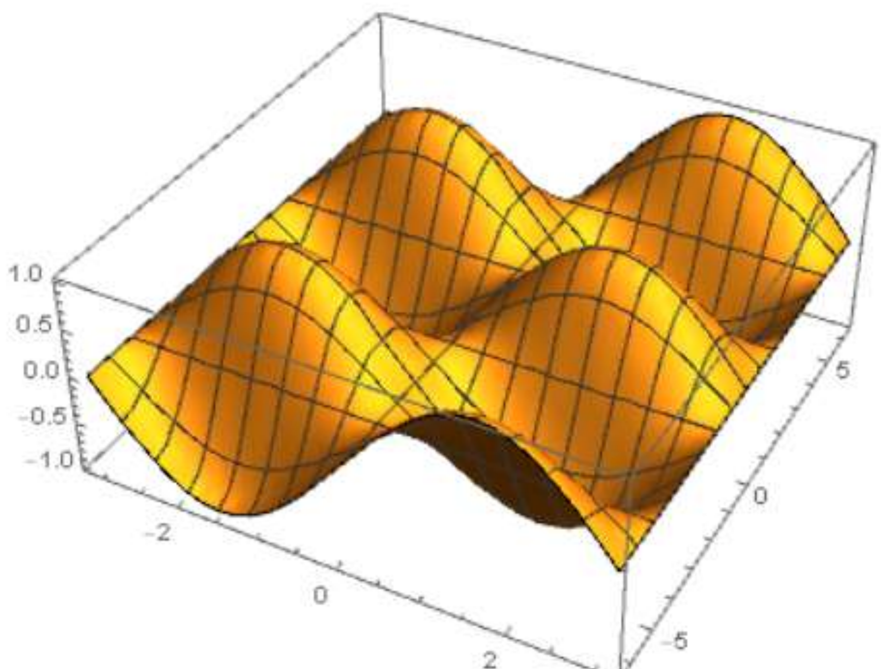
Plot3D[..., {$x$, $y$} ∈ $reg$]
takes variables {$x$, $y$} to be in the geometric region $reg$.

Plot a function:

In[1]:= `Plot3D[Sin[x + y^2], {x, -3, 3}, {y, -2, 2}]`

Out[1]= 


`Plot3D[Sin[x] Cos[y], {x, -Pi, Pi}, {y, -2 Pi, 2 Pi}]`



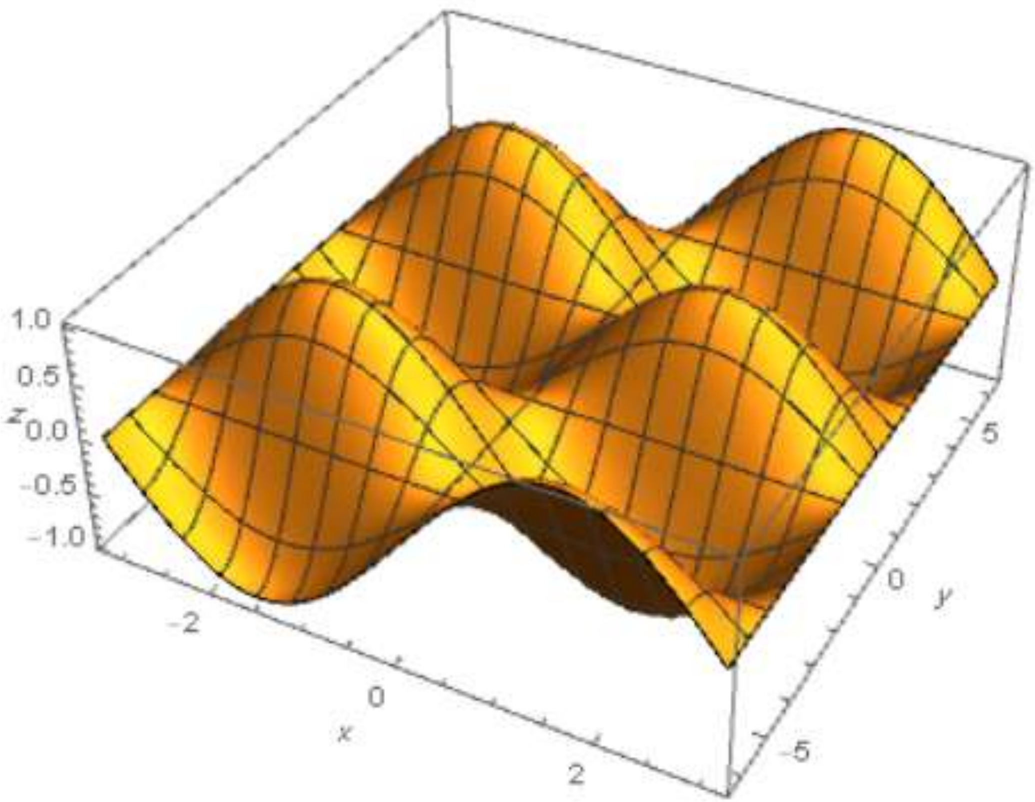
# PlotPoints

PlotPoints
is an option for plotting functions that specifies how many initial sample points to use.

## Details

- With a single variable, PlotPoints -> $n$ specifies the total number of initial sample points to use.
- With more than one variable, PlotPoints -> $n$ specifies that $n$ initial points should be used in each direction.
- PlotPoints -> $\{n_1, n_2, \ldots\}$ specifies different numbers of initial sample points for each successive direction.
- The initial sample points are usually equally spaced.

```
Plot3D[Sin[x] Cos[y], {x, -Pi, Pi}, {y, -2 Pi, 2 Pi}, AxesLabel -> {x, y, z}, PlotPoints -> 40]
```



# Mesh

Mesh

is an option for Plot3D, DensityPlot, and other plotting functions that specifies what mesh should be drawn.

## Details

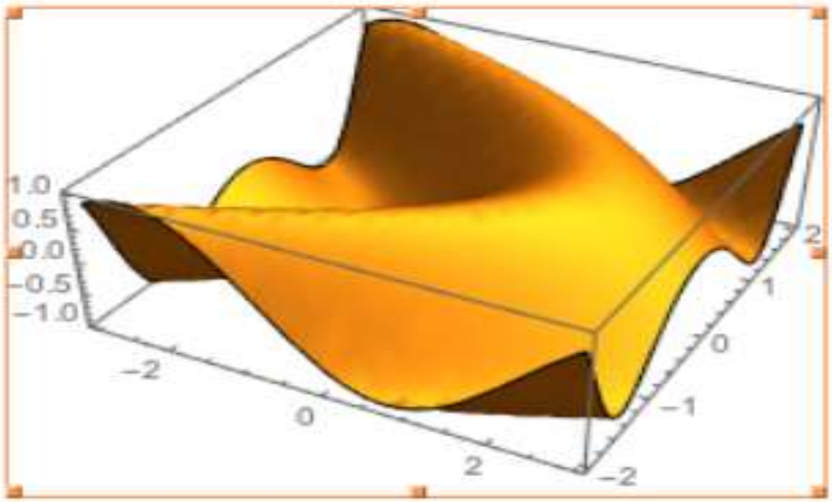
- The following settings can be given for Mesh:

| | |
|---|---|
| None | no mesh drawn |
| *n* | *n* equally spaced mesh divisions |
| All | mesh divisions between all elements |
| Full | mesh divisions between regular data points |
| $\{spec_1, spec_2, \ldots\}$ | separate specifications for each mesh function |

Use no mesh:

```
In[1]:= Plot3D[Sin[x + y^2], {x, -3, 3}, {y, -2, 2}, Mesh -> None]
```

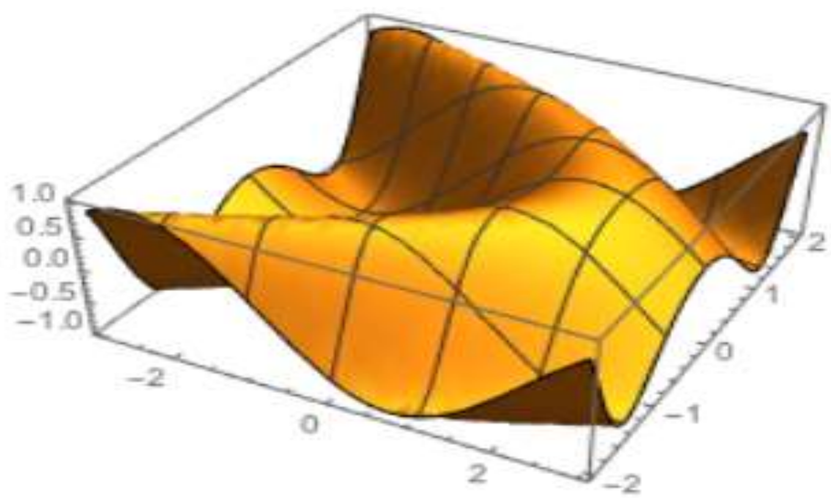
Out[1]=




Use 5 mesh lines in each direction:

```
In[1]:= Plot3D[Sin[x + y^2], {x, -3, 3}, {y, -2, 2}, Mesh → 5]
```

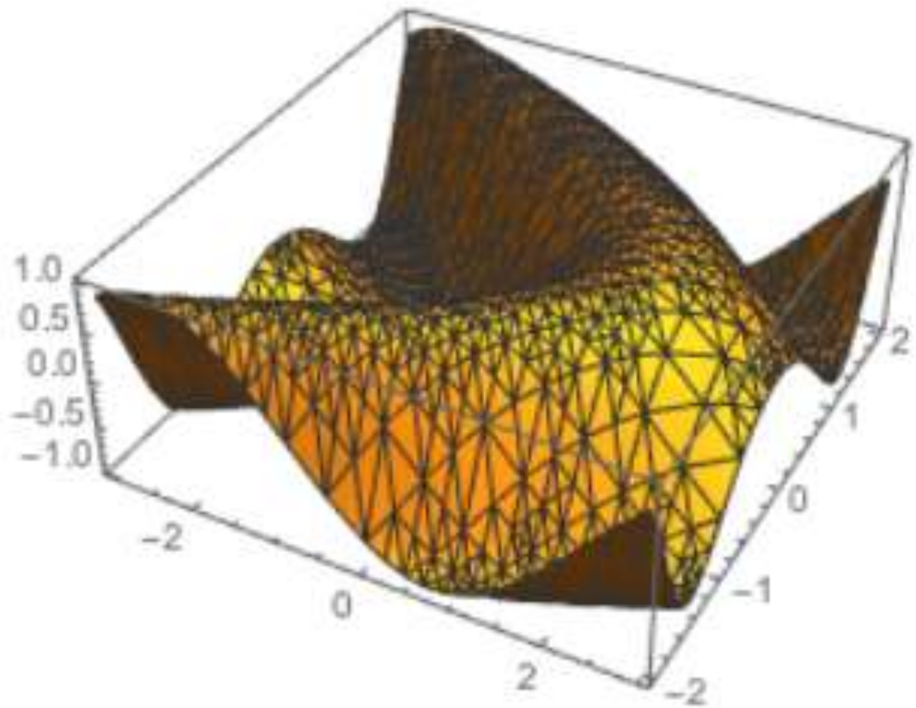
Out[1]=



Show all mesh elements used in the sampling:

In[1]:= 
```
Plot3D[Sin[x + y^2], {x, -3, 3}, {y, -2, 2}, Mesh -> All]
```

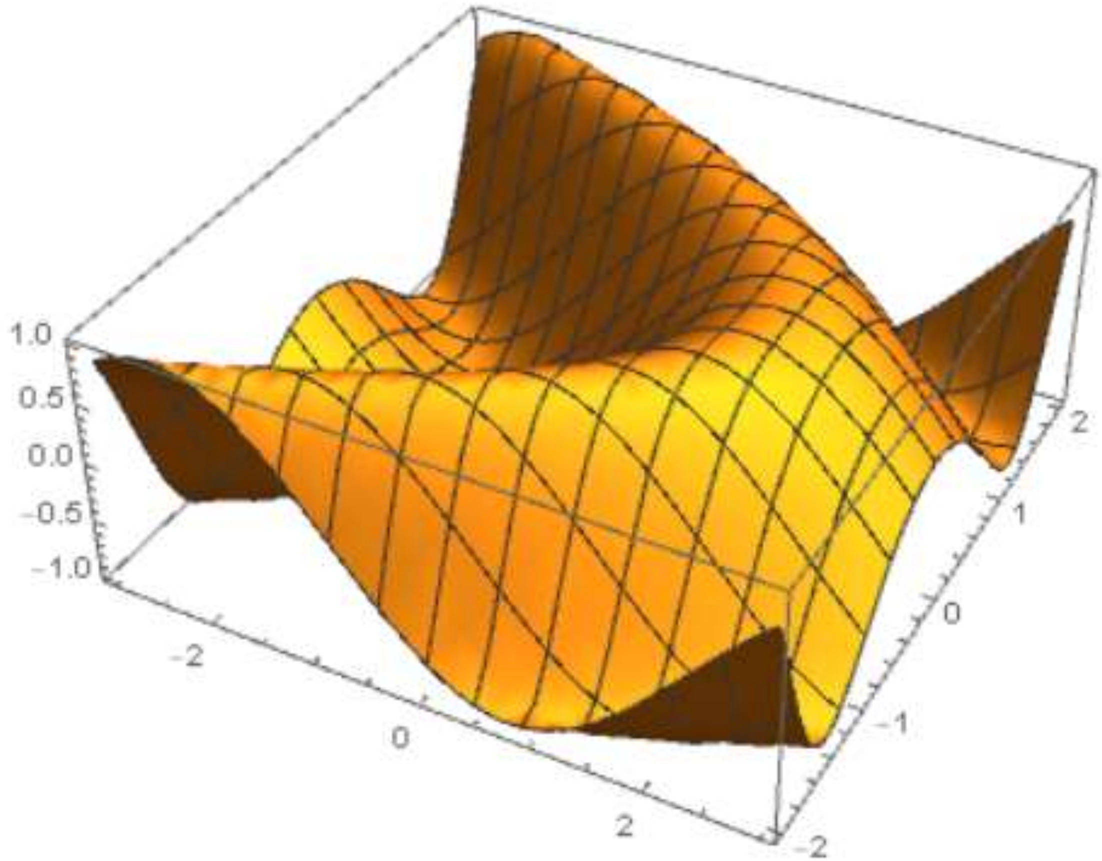
Out[1]=



In[49]:=

```
Plot3D[Sin[x + y^2], {x, -3, 3}, {y, -2, 2}, Mesh → Full]
```

Out[49]=



# MaxRecursion

MaxRecursion
is an option for functions like NIntegrate and Plot that specifies how many recursive subdivisions can be made.

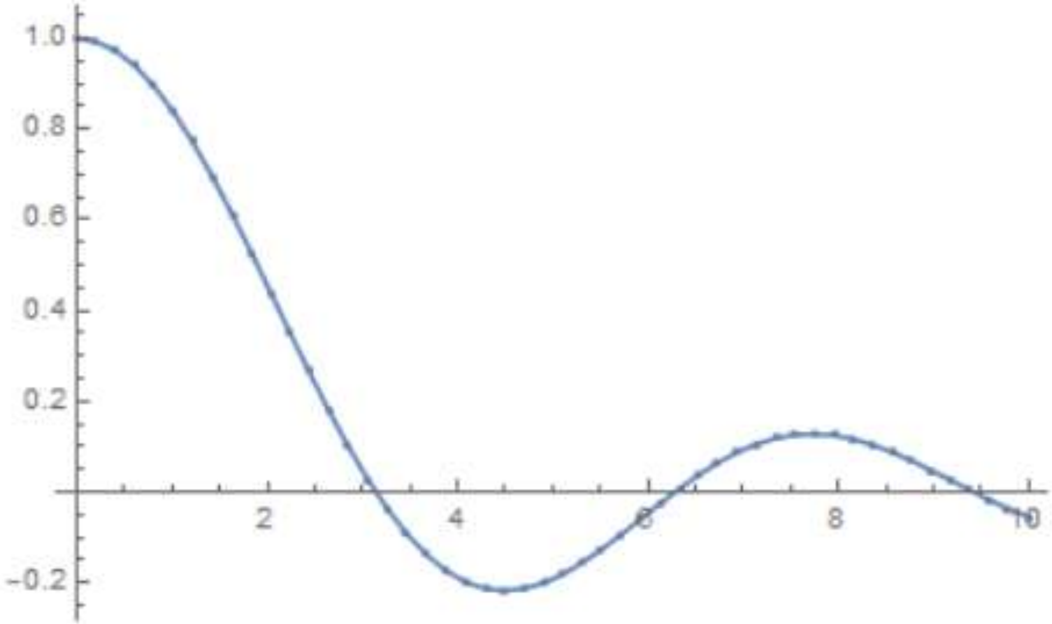
## Details

- MaxRecursion -> *n* specifies that up to *n* levels of recursion should be done.
- Recursive subdivision is done only in those places where more samples seem to be needed in order to achieve results with a certain level of quality.
- In *d* dimensions, each recursive subdivision increases the number of samples taken by a factor that increases roughly exponentially with *d*.

In[44]:=

```
Plot[Sinc[x], {x, 0, 10}, Mesh → All, MaxRecursion → 0, PlotPoints → 50]
```

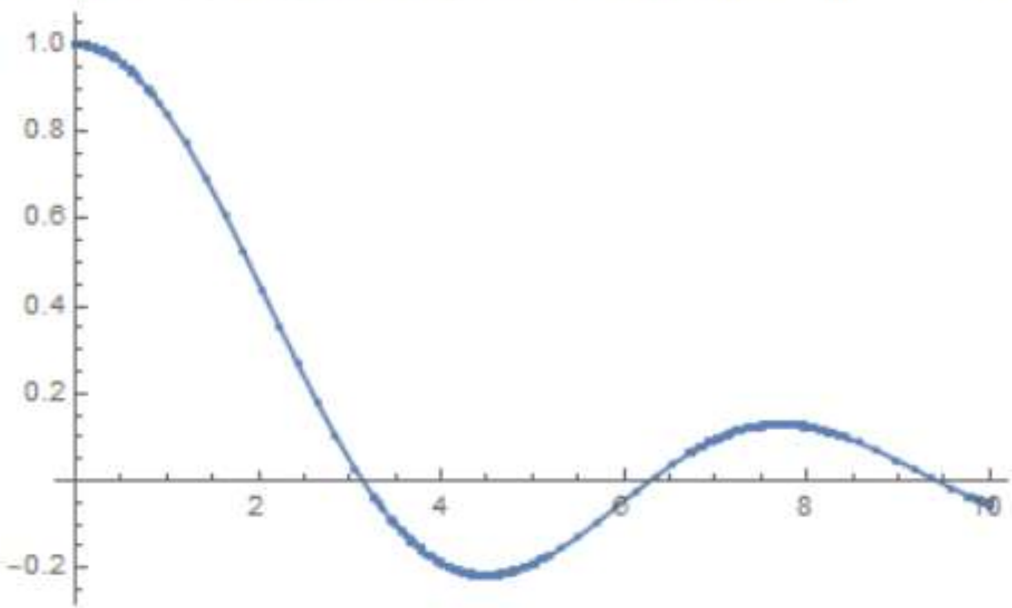
Out[44]=




In[43]:=

```
Plot[Sinc[x], {x, 0, 10}, Mesh → All, MaxRecursion → 15, PlotPoints → 50]
```

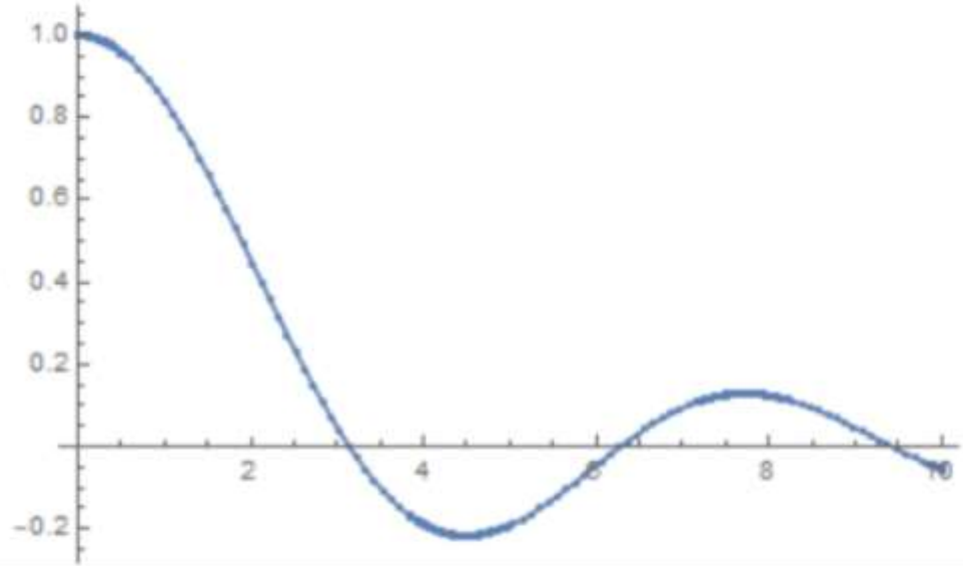
Out[43]=



By increasing the number of points, the line became heavy

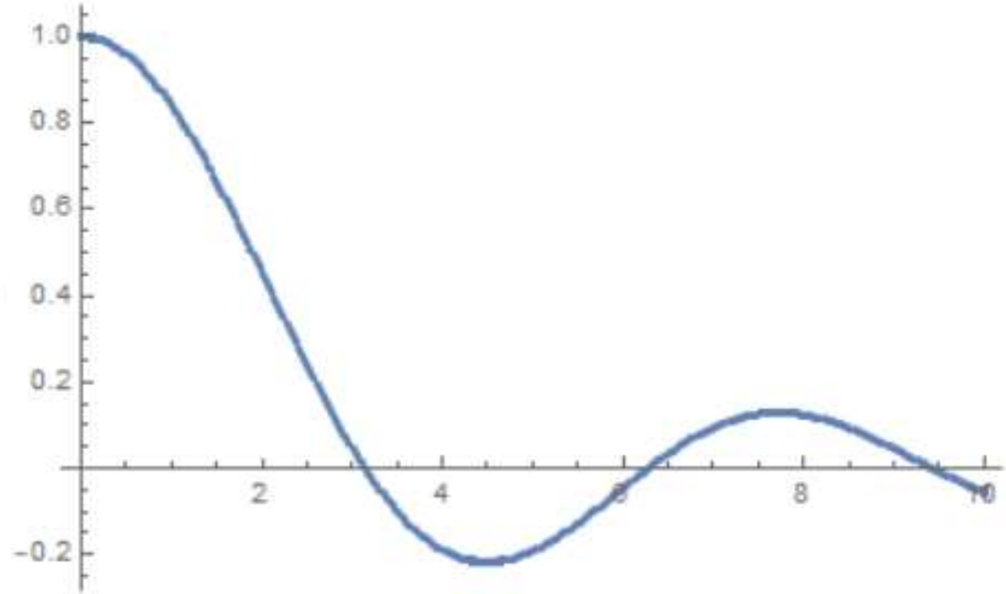
```
In[39]:= Plot[Sinc[x], {x, 0, 10}, Mesh → All, MaxRecursion → 15, PlotPoints → 100]
```

Out[39]=




```
In[41]:= Plot[Sinc[x], {x, 0, 10}, Mesh → All, MaxRecursion → 15, PlotPoints → 300]
```

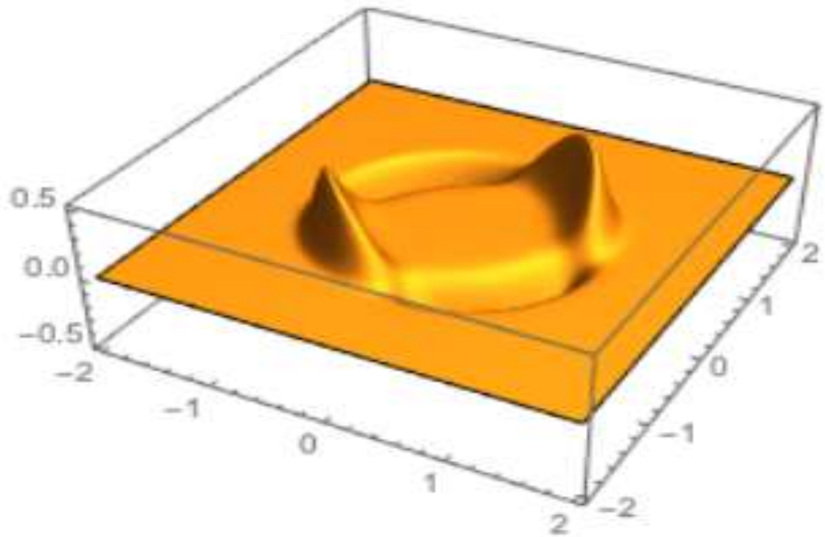
Out[41]=



Get a very high-quality plot of a sharp feature:

```
In[1]:= Plot3D[x y Exp[-10 (x^2 + y^2 - 1)^2], {x, -2, 2},
         {y, -2, 2}, PlotRange → All, Mesh → False, MaxRecursion → 5]
```

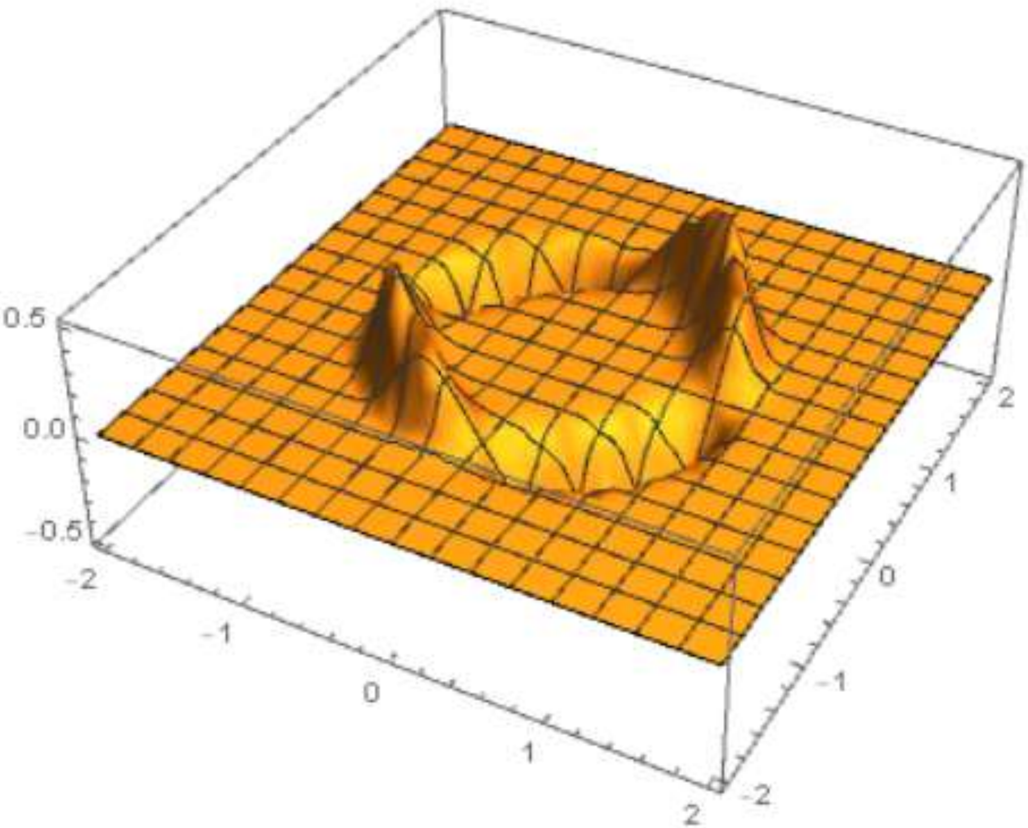
Out[1]=



In[48]:=

```
Plot3D[x y Exp[-10 (x^2 + y^2 - 1) ^2], {x, -2, 2}, {y, -2, 2}, PlotRange → All]
```

Out[48]=



# Table

Table[*expr*, $n$]
generates a list of $n$ copies of *expr*.

Table[*expr*, {$i$, $i_{max}$}]
generates a list of the values of *expr* when $i$ runs from 1 to $i_{max}$.

Table[*expr*, {$i$, $i_{min}$, $i_{max}$}]
starts with $i = i_{min}$.

Table[*expr*, {$i$, $i_{min}$, $i_{max}$, $di$}]
uses steps $di$.

Table[*expr*, {$i$, {$i_1$, $i_2$, ...}}]
uses the successive values $i_1$, $i_2$, ....

Table[*expr*, {$i$, $i_{min}$, $i_{max}$}, {$j$, $j_{min}$, $j_{max}$}, ...]
gives a nested list. The list associated with $i$ is outermost. »

## Details

- You can use Table to build up vectors, matrices, tensors, and other arrays.
- Table uses the standard Wolfram Language iteration specification.
- Table evaluates its arguments in a nonstandard way.
- Table[*expr*, *spec*] first evaluates *spec*, then localizes the variable specified and successively assigns values to it, each time evaluating *expr*.
- Table effectively uses Block to localize values or variables.
- Table[*expr*, $spec_1$, $spec_2$] is effectively equivalent to Table[Table[*expr*, $spec_2$], $spec_1$].

A table of the first 10 squares:

```
In[1]:= Table[i^2, {i, 10}]
Out[1]= {1, 4, 9, 16, 25, 36, 49, 64, 81, 100}
```

---

A table with *i* running from 0 to 20 in steps of 2:

```
In[1]:= Table[f[i], {i, 0, 20, 2}]
Out[1]= {f[0], f[2], f[4], f[6], f[8], f[10], f[12], f[14], f[16], f[18], f[20]}
```

**Example**

Use Mathematica to generate the list `{1,2,3,4,5,6,7,8,9,10}`.

**Table[*i*, {*i*, 10}]**

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

**Table[*i*, {*i*, 1, 10}]**

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

**Example**

Use Mathematica to define `listone` to be the list of numbers {1, 3/2, 2, 5/2, 3, 7/2, 4}.

$\mathbf{listone} = \left\{1, \frac{3}{2}, 2, \frac{5}{2}, 3, \frac{7}{2}, 4\right\}$ $\mathbf{listone} = \mathbf{Table}\left[i, \left\{i, 1, 4, \frac{1}{2}\right\}\right]$

Last, we define $i(n) = \frac{1}{2}n + \frac{1}{2}$ and use `Array` to create the table `listone`.

$i[\mathrm{n\_}] = \frac{n}{2} + \frac{1}{2};$

**listone = Array[*i*, 7]**

**Example**

Create a list of the first 25 prime numbers. What is the fifteenth prime number?

```
list = Table[{n, Prime[n]}, {n, 1, 25}];
```

```
Short[list]
```

{{1, 2}, {2, 3}, {3, 5}, {4, 7}, {5, 11}, {6, 13}, ⟨⟨13⟩⟩, {20, 71}, {21, 73}, {22, 79}, {23, 83},
{24, 89}, {25, 97}}

```
list[[15]]
```

{15, 47}

**Example**

(a) Generate a list consisting of five copies of the letter $a$. (b) Generate a list consisting of ten random integers between $-10$ and 10 and then a list of ten random real numbers between $-10$ and 10.

```
Clear[a]
```

```
Table[a, {5}]
```

$\{a, a, a, a, a\}$

```
RandomInteger[{-10, 10}, 10]
```

```
RandomReal[{-10, 10}, 10]
```

You can also use `Take` to extract elements of lists.

1. `Take[list,n]` returns the first $n$ elements of `list`;
2. `Take[list,-n]` returns the last $n$ elements of `list`; and
3. `Take[list,{n,m}]` returns the $n$th through $m$th elements of `list`.

```
Take[t1, 5]        Take[t1, -5]
```

**Example** **The Prime Difference Function and the Prime Number Theorem**

In `t1`, we use `Prime` and `Table` to compute a list of the first 25, 000 prime numbers.

```
t1 = Table[Prime[n], {n, 1, 25000}];

Length[t1]

Short[t1]

Take[t1, {12501, 12505}]
```

In `t2`, we compute the difference, $d_n$, between the successive prime numbers in `t1`. The result is plotted with `ListPlot` in Fig. 4.4.

```
t2 = Table[t1[[i + 1]] − t1[[i]], {i, 1, Length[t1] − 1}];

Short[t2]
```

**t1 = Table[{Sin[$x$ + $y$], Cos[$x$ − $y$]}, {$x$, 1, 5}, {$y$, 1, 5}]**

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

**Length[t1]**

5

**t1[[3]]**

{{Sin[4], Cos[2]}, {Sin[5], Cos[1]}, {Sin[6], 1}, {Sin[7], Cos[1]}, {Sin[8], Cos[2]}}

and the 2nd element of the third level (or the second part of the third part) is

**t1[[3, 2]]**

{Sin[5], Cos[1]}

**Example** **Dynamical Systems**

A sequence of the form $x_{n+1} = f(x_n)$ is called a **dynamical system**.

Sometimes, unusual behavior can be observed when working with dynamical systems. For example, consider the dynamical system with $f(x) = x + 2.5x(1 - x)$ and $x_0 = 1.2$. Note that we define $x_n$ using the form `x[n_]:=x[n]=...` so that Mathematica "remembers" the functional values it computes and thus avoids recomputing functional values previously computed. This is particularly advantageous when we compute the value of $x_n$ for large values of $n$.

```
Clear[f, x]

f[x_]:=x + 2.5x(1 - x)

x[n_]:=x[n] = f[x[n - 1]]

x[0] = 1.2;
```

```
In[1]:= Table[x, 10]

Out[1]= {x, x, x, x, x, x, x, x, x, x}
```

Make a 4×3 matrix:

```
In[1]:= Table[10 i + j, {i, 4}, {j, 3}]

Out[1]= {{11, 12, 13}, {21, 22, 23}, {31, 32, 33}, {41, 42, 43}}

In[2]:= MatrixForm[%]
```

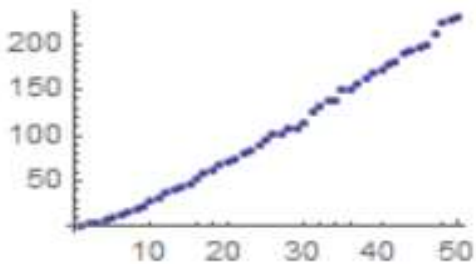
Out[2]//MatrixForm=

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \\ 41 & 42 & 43 \end{pmatrix}$$

Plot a table:

```
In[1]:= ListPlot[Table[Prime[i], {i, 50}]]
```

Out[1]=




Arrange a table in a column:

```
In[1]:= Column[Table[Prime[i], {i, 5}]]

Out[1]= 2
        3
        5
        7
        11
```

The index in the table can run backward:

```
In[1]:= Table[f[i], {i, 10, -5, -2}]

Out[1]= {f[10], f[8], f[6], f[4], f[2], f[0], f[-2], f[-4]}
```

---

Make a triangular array:

```
In[1]:= Table[10 i + j, {i, 5}, {j, i}]

Out[1]= {{11}, {21, 22}, {31, 32, 33}, {41, 42, 43, 44}, {51, 52, 53, 54, 55}}

In[2]:= TableForm[%]

                    11
                    21   22
Out[2]//TableForm=  31   32   33
                    41   42   43   44
                    51   52   53   54   55
```

```
Table[j + i, {i, 5}, {j, i}]

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

```
TableForm[%]

2
3   4
4   5   6
5   6   7   8
6   7   8   9   10
```

Make a 3×2×4 array, or tensor:

```
In[1]:= Table[100 i + 10 j + k, {i, 3}, {j, 2}, {k, 4}]

Out[1]= {{{111, 112, 113, 114}, {121, 122, 123, 124}},
   {{211, 212, 213, 214}, {221, 222, 223, 224}}, {{311, 312, 313, 314}, {321, 322, 323, 324}}}
```

Iterate over an existing list:

```
In[1]:= Table[Sqrt[x], {x, {1, 4, 9, 16}}]

Out[1]= {1, 2, 3, 4}
```

Make an array from existing lists:

```
In[1]:= Table[j^(1/i), {i, {1, 2, 4}}, {j, {1, 4, 9}}]
```

Out[1]= $\left\{\{1, 4, 9\}, \{1, 2, 3\}, \left\{1, \sqrt{2}, \sqrt{3}\right\}\right\}$

Table evaluates the expression separately each time:

```
In[1]:= Table[RandomInteger[10], 20]

Out[1]= {7, 7, 2, 7, 8, 5, 5, 4, 10, 6, 0, 4, 6, 8, 6, 1, 2, 1, 8, 9}
```

The table index can have symbolic values:

```
In[1]:= Table[2^x + x, {x, a, a + 5 n, n}]
```

Out[1]= $\{2^a + a,\ 2^{a+n} + a + n,\ 2^{a+2n} + a + 2n,\ 2^{a+3n} + a + 3n,\ 2^{a+4n} + a + 4n,\ 2^{a+5n} + a + 5n\}$

---

The variables need not just be symbols:

```
In[1]:= Table[a[x]!, {a[x], 6}]

Out[1]= {1, 2, 6, 24, 120, 720}

In[2]:= Table[x[1]^2 + x[2]^2, {x[1], 3}, {x[2], 3}]

Out[2]= {{2, 5, 10}, {5, 8, 13}, {10, 13, 18}}
```

```
In[1]:= Grid[Table[{i, Prime[i]}, {i, 10}]]
```

Out[1]=

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 5 |
| 4 | 7 |
| 5 | 11 |
| 6 | 13 |
| 7 | 17 |
| 8 | 19 |
| 9 | 23 |
| 10 | 29 |

Monitor the values by showing them in a temporary cell:

```
In[1]:= Monitor[Table[Pause[1]; i^i^i, {i, 3}], i]
Out[1]= {1, 16, 7625597484987}
```

Range gives the sequence of values of a table iterator:

```
In[1]:= Range[1, 10, 2]
Out[1]= {1, 3, 5, 7, 9}

In[2]:= Table[i, {i, 1, 10, 2}]
Out[2]= {1, 3, 5, 7, 9}
```

Do evaluates the same sequence of expressions as Table, but does not return them:

```
In[1]:= Do[Print[i^i], {i, 3}]
1
4
27

In[2]:= Table[i^i, {i, 3}]
Out[2]= {1, 4, 27}
```

Sum effectively applies Plus to results from Table:

```
In[1]:= Sum[x^i, {i, 3}]
```

Out[1]= $x + x^2 + x^3$

```
In[2]:= Table[x^i, {i, 3}]
```

Out[2]= $\{x, x^2, x^3\}$

```
In[2]:= Array[Function[{x, y}, x^y], {3, 4}]

Out[2]= {{1, 1, 1, 1}, {2, 4, 8, 16}, {3, 9, 27, 81}}

In[3]:= Table[x^y, {x, 3}, {y, 4}]

Out[3]= {{1, 1, 1, 1}, {2, 4, 8, 16}, {3, 9, 27, 81}}
```

Using multiple iteration specifications is equivalent to nesting Table functions:

```
In[1]:= Table[i + j, {i, 3}, {j, i}]

Out[1]= {{2}, {3, 4}, {4, 5, 6}}

In[2]:= Table[Table[i + j, {j, i}], {i, 3}]

Out[2]= {{2}, {3, 4}, {4, 5, 6}}
```

Formatting wrappers such as Grid give expressions that are no longer lists:

In[1]:= `Grid[Table[i j, {i, 4}, {j, 4}]]`

Out[1]=

```
1 2 3  4
2 4 6  8
3 6 9  12
4 8 12 16
```

In[2]:= `x + %`

Out[2]=

```
     1 2 3  4
     2 4 6  8
x +  3 6 9  12
     4 8 12 16
```

For some step sizes, output from Table may not include the upper limit given:

In[1]:= `Table[x, {x, 0, 10, 3}]`

Out[1]= `{0, 3, 6, 9}`

`MatrixForm[%]`

$$\begin{pmatrix} 0 \\ 3 \\ 6 \\ 9 \end{pmatrix}$$

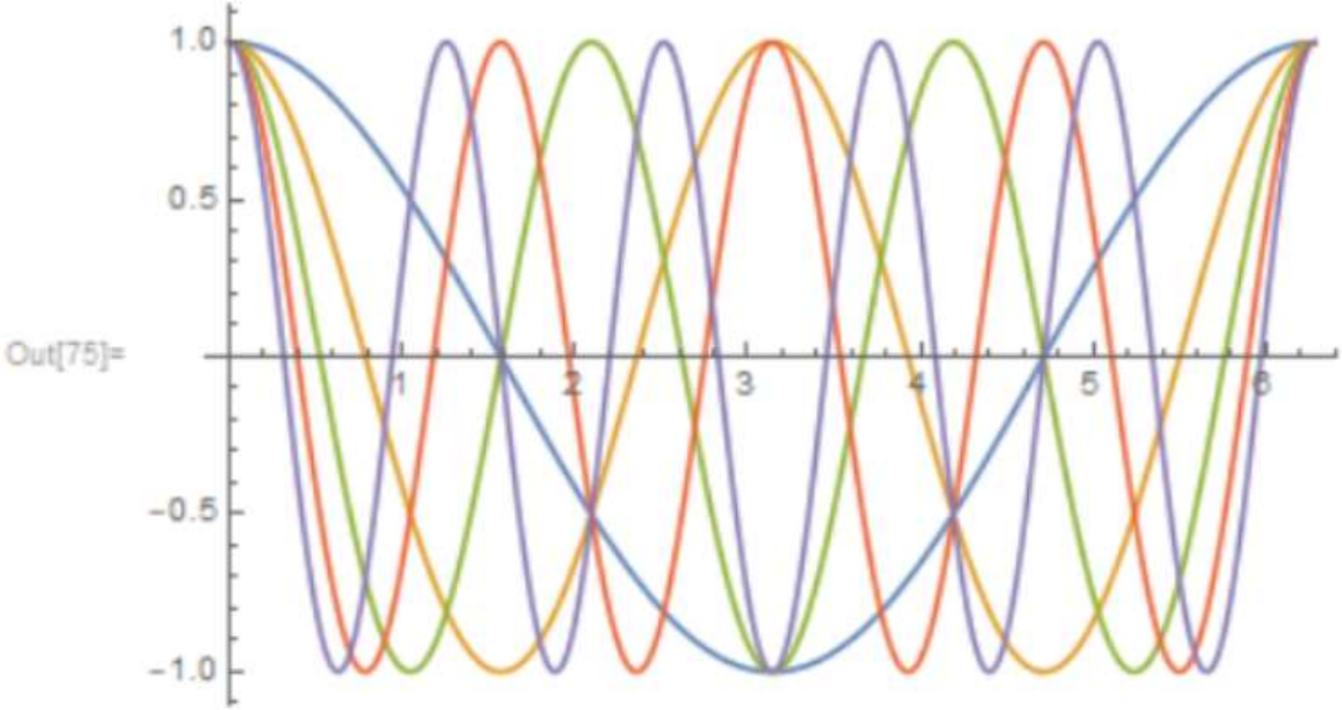
# Evaluate command

A list of expressions can be given to Mathematica to plot using the

**Table[ ]**

In such case, Evaluate command forces evaluation of the command.

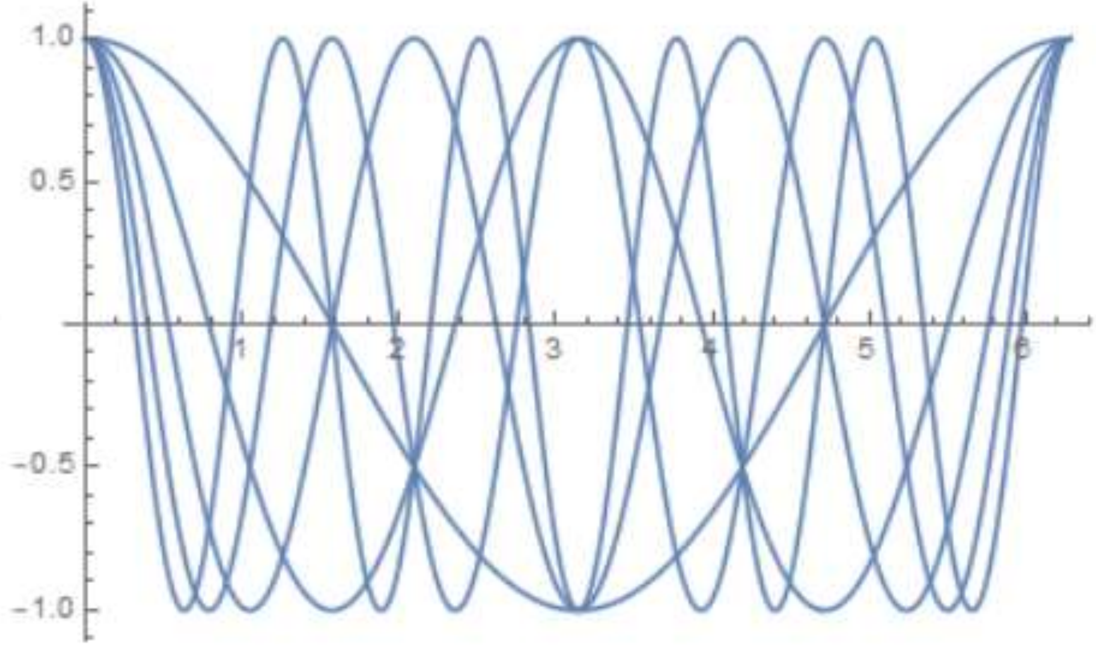
```
In[75]:= Plot[Evaluate[Table[Cos[a*x], {a, 1, 5}]], {x, 0, 2 Pi}]
```



# The output without applying Evaluate

```
In[94]:= Plot[Table[Cos[a*x], {a, 1, 5}], {x, 0, 2 Pi}]
```

Out[94]=




# The values of a and x:

```
Table[{a, x}, {a, 1, 5}, {x, 0, 2 Pi}]

{{{1, 0}, {1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}},
 {{2, 0}, {2, 1}, {2, 2}, {2, 3}, {2, 4}, {2, 5}, {2, 6}},
 {{3, 0}, {3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5}, {3, 6}},
 {{4, 0}, {4, 1}, {4, 2}, {4, 3}, {4, 4}, {4, 5}, {4, 6}},
 {{5, 0}, {5, 1}, {5, 2}, {5, 3}, {5, 4}, {5, 5}, {5, 6}}}
```

# The output values Cos [a*x]

```
In[93]:= Table[Cos[a*x], {a, 1, 5}, {x, 0, 2 Pi}]

Out[93]= {{1, Cos[1], Cos[2], Cos[3], Cos[4], Cos[5], Cos[6]},
  {1, Cos[2], Cos[4], Cos[6], Cos[8], Cos[10], Cos[12]},
  {1, Cos[3], Cos[6], Cos[9], Cos[12], Cos[15], Cos[18]},
  {1, Cos[4], Cos[8], Cos[12], Cos[16], Cos[20], Cos[24]},
  {1, Cos[5], Cos[10], Cos[15], Cos[20], Cos[25], Cos[30]}}
```

# The numeric values Cos [a*x]

```
In[92]:= Table[Cos[a*x], {a, 1, 5}, {x, 0, 2 Pi}] // N

Out[92]= {{1., 0.540302, -0.416147, -0.989992, -0.653644, 0.283662, 0.96017},
  {1., -0.416147, -0.653644, 0.96017, -0.1455, -0.839072, 0.843854},
  {1., -0.989992, 0.96017, -0.91113, 0.843854, -0.759688, 0.660317},
  {1., -0.653644, -0.1455, 0.843854, -0.957659, 0.408082, 0.424179},
  {1., 0.283662, -0.839072, -0.759688, 0.408082, 0.991203, 0.154251}}
```

# List ({...})

$\{e_1, e_2, \ldots\}$
is a list of elements.

## Details

- Lists are very general objects that represent collections of expressions.
- Functions with attribute Listable are automatically "threaded" over lists, so that they act separately on each list element. Most built-in mathematical functions are Listable.
- $\{a, b, c\}$ represents a vector.
- $\{\{a, b\}, \{c, d\}\}$ represents a matrix.
- Nested lists can be used to represent tensors.
- If Nothing appears in a list, it is automatically removed.

```
In[1]:= List[a, b, c, d]

Out[1]= {a, b, c, d}

In[2]:= FullForm[{a, b, c, d}]

Out[2]//FullForm= List[a, b, c, d]
```

Lists are very good for holding data since the elements can be anything:

```
In[1]:= data = {{"George", "Washington", 1789, False},
            {"John" "Adams", 1797, True}, {"Thomas", "Jefferson", 1801, True}};
```

---

Sine of successive squares:

```
In[1]:= ssq = N[Sin[Range[10]^2]]

Out[1]= {0.841471, -0.756802, 0.412118, -0.287903,
         -0.132352, -0.991779, -0.953753, 0.920026, -0.629888, -0.506366}
```

# ListPlot

ListPlot[{$y_1$, $y_2$, ...}]
plots points {1, $y_1$}, {2, $y_2$}, ....

ListPlot[{{$x_1$, $y_1$}, {$x_2$, $y_2$}, ...}]
plots a list of points with specified $x$ and $y$ coordinates.

ListPlot[{$data_1$, $data_2$, ...}]
plots data from all the $data_i$.

ListPlot[{..., $w$[$data_i$, ...], ...}]
plots $data_i$ with features defined by the symbolic wrapper $w$.

- Values $x_i$ and $y_i$ that are not of the form above are taken to be missing and are not shown.
- The $data_i$ have the following forms and interpretations:

| | |
|---|---|
| <\|"$k_1$" → $y_1$, "$k_2$" → $y_2$, ...\|> | values {$y_1$, $y_2$, ...} |
| <\|$x_1$ → $y_1$, $x_2$ → $y_2$, ...\|> | key-value pairs {{$x_1$, $y_1$}, {$x_2$, $y_2$}, ...} |
| {$y_1$ → "$lbl_1$", $y_2$ → "$lbl_2$", ...},<br>{$y_1$, $y_2$, ...} → {"$lbl_1$", "$lbl_2$", ...} | values {$y_1$, $y_2$, ...} with labels {$lbl_1$, $lbl_2$, ...} |

- DataRange determines how values $\{y_1, \ldots, y_n\}$ are interpreted into $\{\{x_1, y_1\}, \ldots, \{x_n, y_n\}\}$. Possible settings include:

| | |
|---|---|
| Automatic, All | uniform from 1 to $n$ |
| $\{x_{min}, x_{max}\}$ | uniform from $x_{min}$ to $x_{max}$ |

- In general a list of pairs $\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$ is interpreted as a list of points, but the setting DataRange → All forces it to be interpreted as multiple *data*$_i$ $\{\{y_{11}, y_{12}\}, \{y_{21}, y_{23}\}, \ldots\}$. »

- LabelingFunction -> *f* specifies that each point should have a label given by *f*[*value*, *index*, *lbls*], where *value* is the value associated with the point, *index* is its position in the *data*, and *lbls* is the list of relevant labels.

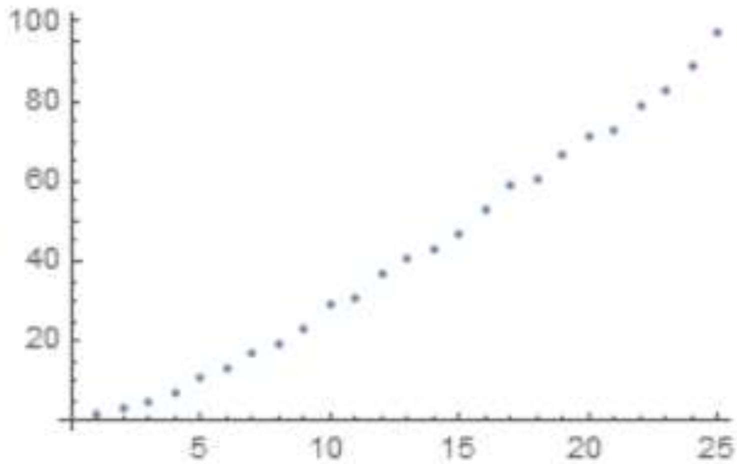
- Typical settings for PlotLegends include:

| | |
|---|---|
| None | no legend |
| Automatic | automatically determine legend |
| $\{lbl_1, lbl_2, \ldots\}$ | use $lbl_1, lbl_2, \ldots$ as legend labels |
| Placed[*lspec*, ...] | specify placement for legend |

Plot a list of $y$ values:

```
In[1]:= ListPlot[Prime[Range[25]]]
```

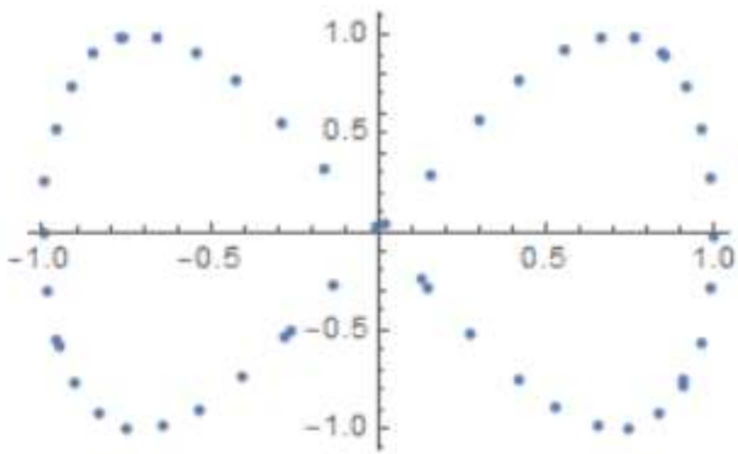
Out[1]=




Plot a list of $x$, $y$ pairs:

```
In[1]:= ListPlot[Table[{Sin[n], Sin[2 n]}, {n, 50}]]
```

Out[1]=

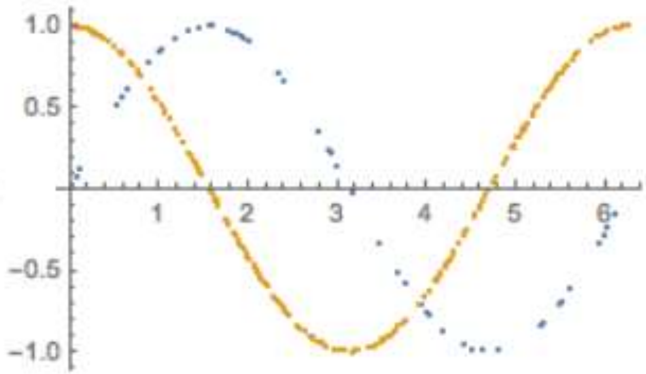


Plot multiple sets of irregular data:

```
In[1]:= data1 = Table[{x, Sin[x]}, {x, RandomReal[2 Pi, 50]}];
        data2 = Table[{x, Cos[x]}, {x, RandomReal[2 Pi, 250]}];
```

```
In[2]:= ListPlot[{data1, data2}]
```

Out[2]=

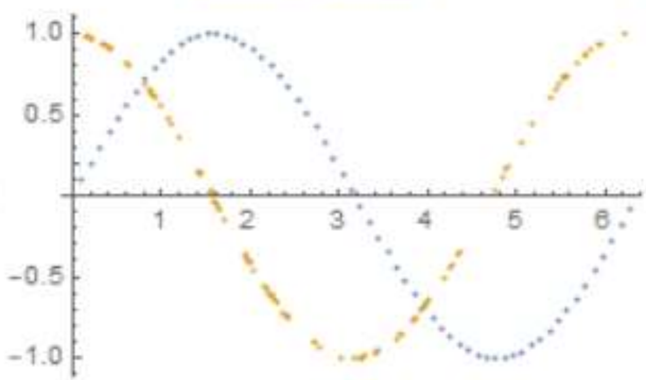



Plot multiple sets of data, regular or irregular, using DataRange to map them to the same *x* range:

```
In[1]:= data1 = Table[Sin[x], {x, 0, 2 Pi, 0.1}];
        data2 = Table[{x, Cos[x]}, {x, RandomReal[2 Pi, 100]}];
```

```
In[2]:= ListPlot[{data1, data2}, DataRange → {0, 2 Pi}]
```

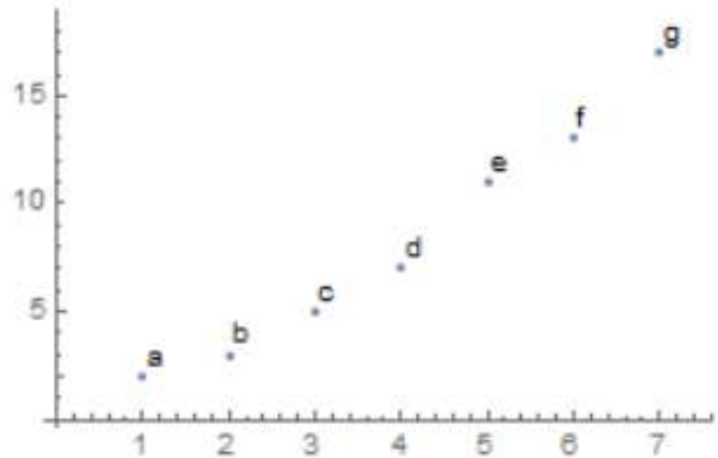
Out[2]=



Specify strings to use as labels:

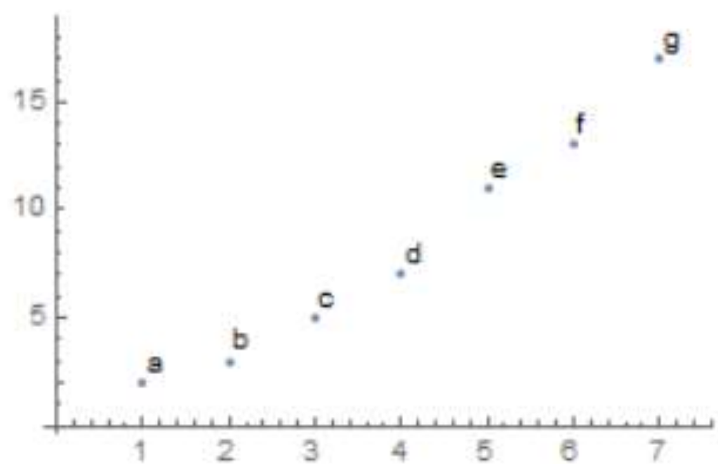
In[1]:= `ListPlot[{2 → "a", 3 → "b", 5 → "c", 7 → "d", 11 → "e", 13 → "f", 17 → "g"}]`

Out[1]=




In[2]:= `ListPlot[{2, 3, 5, 7, 11, 13, 17} → {"a", "b", "c", "d", "e", "f", "g"}]`

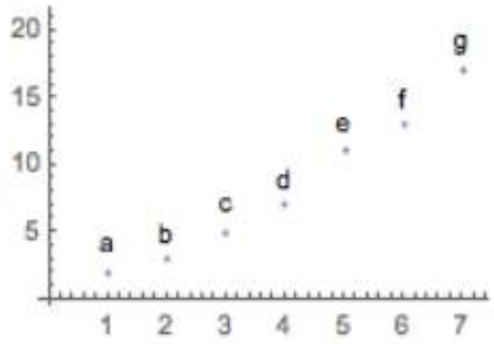
Out[2]=



Specify a location for labels:

```
In[1]:= ListPlot[{2, 3, 5, 7, 11, 13, 17} → {"a", "b", "c", "d", "e", "f", "g"}, LabelingFunction → Above]
```

Out[1]=

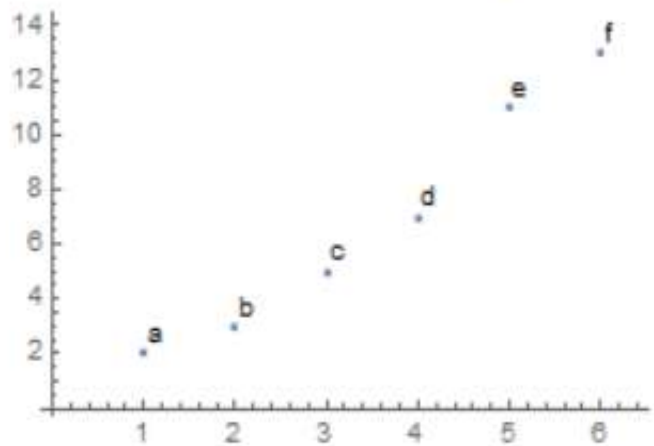


---

Numeric values in an Association are used as the $y$ coordinates:

```
In[1]:= ListPlot[<|"a" → 2, "b" → 3, "c" → 5, "d" → 7, "e" → 11, "f" → 13|>]
```

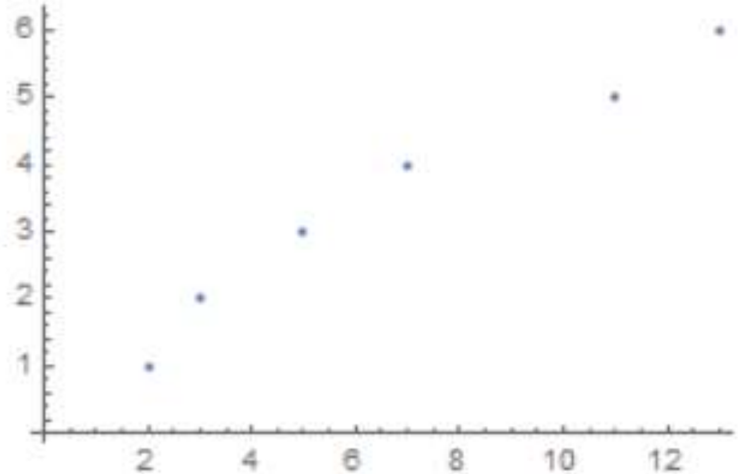
Out[1]=



Numeric keys and values in an Association are used as the $x$ and $y$ coordinates:

```
In[2]:= ListPlot[<|2 → 1, 3 → 2, 5 → 3, 7 → 4, 11 → 5, 13 → 6|>]
```

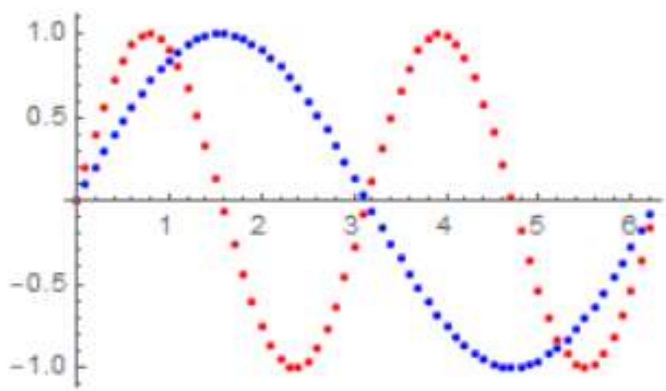
Out[2]=




Provide explicit styling to different sets:

```
In[1]:= data = Table[Table[{x, f}, {x, 0, 2 Pi, 0.1}], {f, {Sin[x], Sin[2 x]}}];
In[2]:= ListPlot[data, PlotStyle → {Blue, Red}]
```

Out[2]=



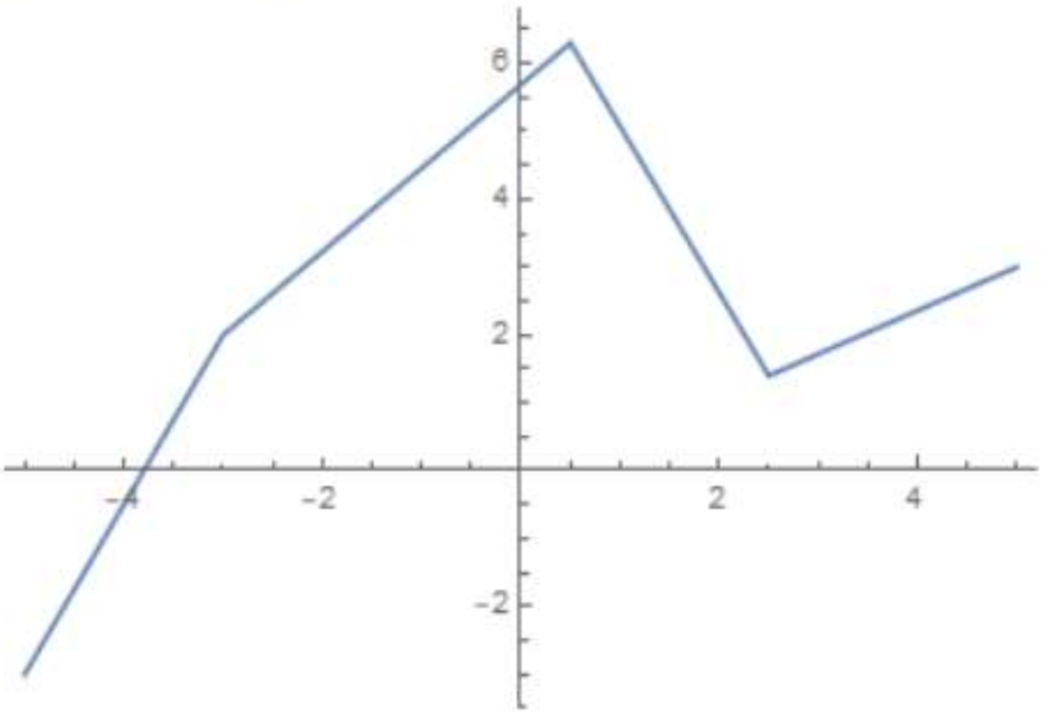
## ListPlot

ListPlot[ ] command draws a list of points, given as coordinate pairs (x,y)

In[106]:=

```
p3 = ListPlot[{{-5, -3}, {-3, 2}, {0.5, 6.3}, {2.5, 1.4}, {5, 3}}, PlotJoined -> True]
```

Out[106]=



To draw a plot joining the points (1, y1), (2,y2),..., (n , $y_n$).

```
In[107]:= ListPlot[{2.5, 3.7, -1.2, 7.0, 9.1, -2.3}, PlotJoined → True]
```

Out[107]=