

Calculator

1 Usage

1.1 Quick Examples

Enter the input into the upper window and press Shift+Enter to evaluate it. The return values appear in the window below. White spaces in the input are treated as multiplication where appropriate. The whole input window is evaluated, and it must be a nested combination of the functions defined in Section 3 below. The examples supplied in the “Examples” menu tab should provide enough information to grasp the syntax.

To calculate $2 + \frac{3!}{1+\pi}$ to at least 40 digits:

```
N[2+3!/(1+p\),40]
```

To define $f(x) = x^2 + \sin(x)$ and calculate the array $\{f(0), \dots, f(4)\}$:

```
f[x]:=x^2+Sin[x];Table[f[n],{n,5}]
```

To define f to be the factorial function in three different ways:

```
f[x]:=x!  
f[x]:=If[x<=1,1,x*f[x-1]]  
f[x]:=Product[n,{n,1,x}]
```

To make a plot of $z = \sin(xy)$:

```
ParametricPlot3D[x:y:Sin[x*y],{x,-4,4},{y,-4,4}]
```

To make a plot of $\rho = 2 + \frac{1}{5}\phi^2(\pi - \phi)^2 \sin(6\theta) \sin(7\phi)$ in spherical coordinates:

```
ParametricPlot3D[Local[r:=2+1/5 (ph\ (p\ - ph\))^2 Sin[6 th\] Sin[7 ph\];  
r\ Cos[th\] Sin[ph\] : r\ Sin[th\] Sin[ph\] : r\ Cos[ph\]],  
{ph\, 0, p\}, {th\, 0, 2p\}]
```

1.2 Windows and Options

Input bar: located at the very top; type the input here and press Shift+Enter to evaluate it

Output bar: located below the input bar; the last result is printed in red here

History window: shows a list of completed calculations

Debug window: shows the parsing of the input and the internal state of the math engine; toggle this window with F2

Button window: pastes the button contents in at the cursor; toggle this window with F4

Menu options: F1 or right click anywhere.

Zoom option: Pressing F3 will toggle between a 1x and 2x zoom of the whole window.

Optimization option: With optimizations disabled, straight-forward code is generated; with optimizations enabled, certain code simplifications are performed.

Ex: `ParametricPlot3D[Cos[θ]Sin[φ]:Sin[θ]Sin[φ]:Cos[φ],{φ,0,π},{θ,0,2π}]`

Optimization Off:	Optimization On:
5570560: &0 = 0;	5636096: &0 = 0;
5570572: &1 = 3.141592653589793;	5636108: &1 = 3.141592653589793;
5570584: &2 = 360;	5636120: &2 = 360;
5570596: &3 = 0;	5636132: &3 = 6.283185307179586;
5570608: &4 = 2;	5636144: Jump[5636260];
5570620: &5 = 3.141592653589793;	5636148: &6 : &4 = _CosSin[&4]; (1)
5570632: &4 = __Times[&4, &5]; (1)	5636176: &7 : &5 = _CosSin[&5]; (1)
5570660: &5 = 360;	5636204: &4 = __Times[&4, &7]; (1)
5570672: Jump[5570912];	5636232: &6 = __Times[&6, &7]; (1)
5570676: &8 = &7;	5636260: ParametricPlot3D[Jump[5636148], &5,&0,&1,&2, &4,&0,&3,&2, &4,&6,&5]
5570688: &8 = Cos[&8]; (1)	5636428: &0 = 1;
5570712: &9 = &6;	5636440: Halt[&0];
5570724: &9 = Sin[&9]; (1)	
5570748: &8 = __Times[&8, &9]; (1)	
5570776: &9 = &7;	
5570788: &9 = Sin[&9]; (1)	
5570812: &10 = &6;	
5570824: &10 = Sin[&10]; (1)	
5570848: &9 = __Times[&9, &10]; (1)	
5570876: &10 = &6;	
5570888: &10 = Cos[&10]; (1)	
5570912: ParametricPlot3D[Jump[5570676], &6,&0,&1,&2, &7,&3,&4,&5, &8,&9,&10]	
5571080: &0 = 1;	
5571092: Halt[&0];	

2 Data Types

Values are 256 bits in length. The top 16 bits are used to identify the type. These can be:

MachineComplex: stored as two 80-bit reals in the lower 160 bits

MachineInteger: a 192-bit signed integer

True: represented by any machine integer with a non-zero lowest QWORD

False: represented by any machine integer with a zero lowest QWORD

BigInteger: stores a GMP integer structure in the lowest 128 bits

BigReal: stores a GMP float structure in the lower 192 bits

Array: rank ≤ 3 array of machine complex/machine integer

Big numbers can go up to $\approx 2^{2^{30}}$

3 Built-in Functions

The calculator converts your input into a simple stack-based evaluation. For example, the evaluation of $1 + 2/3$ would push three constants onto the stack and call the divide and add functions in that order. With this architecture, only the function **PushExpr** (or **:**) might need an explanation. It simply push all of its arguments on the stack so that functions and inputs can return more than one value. It also has the effect of ‘splicing in’ arguments into functions so that **Plus[1, PushExpr[2, 3]]** (or $1 + (2 : 3)$) returns **6**. No checking is done to ensure that a function doesn’t attempt to redefine itself. Calling a plot function from inside a plot function will abort the calculation. **Functions in red have not been implemented yet.**

3.1 Basic Functions

Divide $[z_1, \dots, z_n]$ or $z_1 / \dots / z_n$: returns $(\dots((z_1/z_2)/z_3)\dots)/z_n$.

Equal $[z_1, z_2]$ or $z_1 == z_2$: returns $z_1 = z_2$.

Greater $[z_1, z_2]$ or $z_1 > z_2$: returns $z_1 > z_2$.

GreaterEqual $[z_1, z_2]$ or $z_1 \geq z_2$: returns $z_1 \geq z_2$.

Less $[z_1, z_2]$ or $z_1 < z_2$: returns $z_1 < z_2$.

LessEqual $[z_1, z_2]$ or $z_1 \leq z_2$: returns $z_1 \leq z_2$.

Minus $[z_1, \dots, z_n]$ or $z_1 - \dots - z_n$: returns $(\dots((z_1 - z_2) - z_3)\dots) - z_n$.

N $[z_1, \textit{digits}]$: set the precision of the evaluation of z_1 . *digits* must be an explicit integer.

Out $[n]$: the n^{th} output.

Out $[-n]$ or **%%...%**: the n^{th} from last output.

Plus $[z_1, \dots, z_n]$ or $z_1 + \dots + z_n$: returns $(\dots((z_1 + z_2) + z_3)\dots) + z_n$.

Power $[z_1, \dots, z_n]$ or $z_1^{\dots^{\wedge} z_n}$: returns $z_1^{\dots^{\wedge} z_n}$.

Sqrt $[z_1]$: returns $\sqrt{z_1}$.

Times $[z_1, \dots, z_n]$ or $z_1 * \dots * z_n$ or $z_1 z_2 \dots z_n$: returns $(\dots((z_1 \times z_2) \times z_3)\dots) \times z_n$.

Unequal $[z_1, z_2]$ or $z_1 \neq z_2$: returns $z_1 \neq z_2$.

Min $[z_1, \dots, z_n]$: returns the minimum.

Max $[z_1, \dots, z_n]$: returns the maximum.

3.2 Integer Functions

And $[z_1, \dots, z_n]$ or $z_1 \& \dots \& z_n$: returns the bitwise and of the z_i .

BitCount $[n]$: returns the number of ones in $|n|$.

BitShift $[n, e]$: returns $n * 2^e$ rounded towards 0.

Fibonacci $[z]$: Fibonacci numbers.

GCD $[z_1, \dots, z_n]$: returns the positive GCD of the z_i .

LCM $[z_1, \dots, z_n]$: returns the positive LCM of the z_i .

Or $[z_1, \dots, z_n]$ or $z_1 | \dots | z_n$: returns the bitwise or of the z_i .

QuotientMod $[n, d]$: returns $q = \lfloor n/d \rfloor$ and r such that $n = qd + r$.

Quotient $[n, d]$: returns q .

Mod $[n, d]$: returns r .

Xor $[z_1, \dots, z_n]$ or $z_1 \sim \dots \sim z_n$: returns the bitwise xor of the z_i .

Xor $[z_1]$ or $\sim z_1$: returns the bitwise not of z_1 .

3.3 Rounding Functions

Round[z]: closest integer to z with ties going to even.

Floor[z]: round towards $-\infty$.

Ceiling[z]: round towards ∞ .

IntegerPart[z]: round towards 0.

FractionalPart[z]: difference between z and the integer part of z .

3.4 Elementary Transcendental Functions

Log[z]: returns the inverse of e^z where $-\pi < \text{Im}(\log(z)) \leq \pi$.

Exp[z]: returns $e^z = \sum z^n/n!$.

ArcCos[z]: returns the inverse of $\cos(z)$.

ArcSin[z]: returns the inverse of $\sin(z)$.

ArcTan[z]: returns the inverse of $\tan(z)$.

ArcCosh[z]: returns the inverse of $\cosh(z)$.

ArcSinh[z]: returns the inverse of $\sinh(z)$.

ArcTanh[z]: returns the inverse of $\tanh(z)$.

Cos[z]: returns $\cos(z)$.

Sin[z]: returns $\sin(z)$.

Tan[z]: returns $\tan(z)$.

Cosh[z]: returns $\cosh(z)$.

Sinh[z]: returns $\sinh(z)$.

Tanh[z]: returns $\tanh(z)$.

3.5 Special Functions

Factorial[z] or $z!$: returns $z! = \prod \frac{(1+1/n)^z}{1+z/n}$.

3.6 Array Functions

ConstantArray[$z, u_{max}, v_{max}, w_{max}$]: creates a $u_{max} \times v_{max} \times w_{max}$ array filled with z .

List[z_1, \dots, z_n] or $\{z_1, \dots, z_n\}$: evaluates the z_i in turn and puts them into an array.

3.7 String Functions

HexString[z]: partition z into qwords. Integers are represented in 2's complement notation, and big reals are printed as: mantissa, exponent, sign. The most significant bit of the mantissa should be set for non-zero reals, and the implied radix point is just after this bit.

3.8 Operators

Do[$z, \{u, u_{low}, u_{hi}, u_{step}\}, \{v, v_{low}, v_{hi}, v_{step}\}, \dots$]: do loop. The loop associated with u is outermost. The u_{low}, \dots should each return one value.

Product[$z, \{u, u_{low}, u_{hi}, u_{step}\}, \{v, v_{low}, v_{hi}, v_{step}\}, \dots$]: accumulated product. The loop associated with u is outermost. The u_{low}, \dots should each return one value.

Sum[$z, \{u, u_{low}, u_{hi}, u_{step}\}, \{v, v_{low}, v_{hi}, v_{step}\}, \dots$]: accumulated sum. The loop associated with u is outermost. The u_{low}, \dots should each return one value.

Table[$z, \{u, u_{max}\}, \{v, v_{max}\}, \{w, w_{max}\}$]: creates a table of dimensions $u_{max} \times v_{max} \times w_{max}$. The loop associated with u is outermost. A rectangular array is generated from $0 \leq u < u_{max}, \dots$.

3.9 Definition Constructs

Local[$v_1 = z_1, \dots, v_n = z_n, z_{n+1}$] or $v_1 @ \dots @ v_n @ z_{n+1}$: creates space on the stack for the v_i and evaluates the z_i in turn.

Part[z_1, i, j, k] or $z_1[[i, j, k]]$: extract the part of an array z_1 .

Set[**Part**[z_1, i, j, k], z_2] or $z_1[[i, j, k]] = z_2$: set the part of an array z_1 .

Set[v_1, z_1] or $v_1 = z_1$: evaluates z_1 and assigns the top return value to v_1 .

Set[**PushExpr**[v_1, \dots, v_n, z_1] or $v_1 : \dots : v_n = z_1$]: evaluates z_1 and assigns the top n return value the v_i .

$f[v_1, \dots, v_m] = z_1$: compiles z_1 as a function of the m arguments v_i . This function may then be called via

$f[z_1, \dots, z_m]$ and is memoized for $m \leq 4$ and integer inputs. The number of return values is 1.
 $f[v_1, \dots, v_m] : n = z_1$: compiles z_1 as a function of the m arguments v_i . This function may then be called via $f[z_1, \dots, z_m]$, and the number of return values, n , is the number of $:$'s.

3.10 Flow Constructs

CmpdExpr $[z_1, \dots, z_n]$ or **$z_1; \dots; z_n$** : evaluates the z_i in turn, returning the stack in between.

If $[z_1, z_2]$: If the return value of z_1 (one value) is True, evaluates z_2 , else returns the same number of undefined values that z_2 would return.

If $[z_1, z_2, z_3]$: If the return value of z_1 (one value) is True, evaluates z_2 , else evaluates z_3 . z_2 and z_3 should return the same number of values.

Loop $[z_1]$: evaluates z_1 until the value is False.

PushExpr $[z_1, \dots, z_n]$ or **$z_1 : \dots : z_n$** : evaluates the z_i in turn.

While $[z_1, z_2]$: while z_1 returns (one value) True, evaluates z_2 . The stack pointer is returned to its original position between z_1 and z_2 .

3.11 Graphing Constructs

The graphics settings in the settings menu tab set a default number of divisions to use if the fourth argument of the iterators is not supplied.

ContourPlot2D $[f, \{x, x_{min}, x_{max}, x_{div}\}, \{y, y_{min}, y_{max}, y_{div}\}]$: Draws a plot of the (curve) $f(x, y) = 0$. f should return one value, and the curve is found with a distance estimate.

ContourPlot3D $[f, \{x, x_{min}, x_{max}, x_{div}\}, \{y, y_{min}, y_{max}, y_{div}\}, \{z, z_{min}, z_{max}, z_{div}\}]$: Draws a plot of the (surface) $f(x, y, z) = 0$. f should return one value, and the surface is found by looking for changes in the sign of f .

ParametricPlot2D $[F, \{u, u_{min}, u_{max}, u_{div}\}]$: Draws a plot of the curve $F([u_{min}, u_{max}])$. F should return two values (the coordinate functions $x[u, v] : y[u, v]$).

ParametricPlot3D $[F, \{u, u_{min}, u_{max}, u_{div}\}, \{v, v_{min}, v_{max}, v_{div}\}]$: Draws a plot of the surface $F([u_{min}, u_{max}] \times [v_{min}, v_{max}])$. F should return three values (the coordinate functions $x[u, v] : y[u, v] : z[u, v]$).

plot window controls:

w/a/s/d: movement i/j/k/l: look

c/x: change outside/inside color

n/b/m: toggle normals/box/mesh

4 Updates

v1.00

- fixed evaluation of $3^{\wedge}200$
- fixed parsing of $-x^{\wedge}y$ from $(-x)^{\wedge}y$ to $-(x^{\wedge}y)$
- fixed computation of E
- comparison functions work for all arg types
- logical functions implemented
- fixed printing of big floats
- function Factorial works everywhere except on multiprecision

v1.01

- fixed compilation of Product and Local and some issues with UI
- implemented rounding functions
- added plugin loader with demo.dll example
- added support for scientific notation, e.g. $1.2e10$ ect.

v1.02

- implemented Table and ConstantArray functions
- byte code optimizer started

v1.03

- bytecode optimizer finished: constant propagation, common subexpression elimination
- multiple argument Plus and Times are evaluated by binary splitting
- implemented BitShift, BitCount, and GCD

5 Formulas

Let n denote the number of bits of precision desired in the result. Let $AGM_k(a, b)$ denote the average of the k^{th} iterates of the classical arithmetic-geometric mean. Also, \cdot denotes a general associative operator whose long products are to be evaluated by binary splitting.

5.1 Constants

E: Let $\begin{pmatrix} p_1 \\ q_1 \end{pmatrix} \cdot \begin{pmatrix} p_2 \\ q_2 \end{pmatrix} = \begin{pmatrix} p_1 q_2 + p_2 \\ q_1 q_2 \end{pmatrix}$. Then, e is obtained via the series

$$e = \lim_{k \rightarrow \infty} \frac{q_k + p_k}{q_k}, \quad \begin{pmatrix} p_k \\ q_k \end{pmatrix} = \prod_{j=1}^k \begin{pmatrix} 1 \\ j \end{pmatrix}.$$

k must be chosen so that $\frac{1}{kk!} < 2^{-n}$. The initial guess and Newton-Raphson iterations are:

$$k \leftarrow \frac{n}{\log_2(n/\log_2(n))}; \quad k \leftarrow k - \frac{-k - \frac{1}{12k} + (k + \frac{3}{2}) \log(k) - n \log(2) + \frac{1}{2} \log(2\pi)}{-\frac{1}{12k^2} + \frac{3}{2k} + \log(k)}.$$

Two iterations followed by $k \leftarrow \lceil k \rceil$ will deduce the optimal k .

Pi: Let $\begin{pmatrix} p_1 \\ r_1 \\ q_1 \end{pmatrix} \cdot \begin{pmatrix} p_2 \\ r_2 \\ q_2 \end{pmatrix} = \begin{pmatrix} p_1 q_2 + r_1 p_2 \\ r_1 r_2 \\ q_1 q_2 \end{pmatrix}$. Then π is obtained as

$$\pi = \lim_{k \rightarrow \infty} \frac{q_k c / 12 \sqrt{c}}{p_k + a q_k}, \quad \begin{pmatrix} p_k \\ r_k \\ q_k \end{pmatrix} = \prod_{j=1}^k \begin{pmatrix} (-1)^j (6j-5)(2j-1)(6j-1)(a+bj) \\ (6j-5)(2j-1)(6j-1) \\ j^3 c^3 / 24 \end{pmatrix}, \quad \begin{matrix} a = 13591409 \\ b = 545140134 \\ c = 640320 \end{matrix}.$$

A good formula for k is

$$k \leftarrow \left\lfloor \frac{n}{3 \log_2(c/12)} \right\rfloor.$$

5.2 Elementary Functions

Exp[x]: the argument is first reduced to the range $2|x| < \log 2$. Then, $f(x) = e^x - 1$ is calculated via $n^{1/2}$ iterations of

$$f(2x) = f(x)^2 + 2f(x)$$

and $n^{1/2}$ terms of

$$f(x) = \sum_{m=1}^{\infty} \frac{x^m}{m!}.$$

The intermediate computations are performed with only $n + \log_2(n)$ bits (not $n + n^{1/2}$ bits).

Log[x]: $\log(x)$ for $x > 1$ is obtained from

$$\log(2^m x) = \frac{\pi}{2 \operatorname{AGM}_k(1, 2^{2-m}/x)} \left(1 + O\left(\frac{1}{2^{2m} x^2}\right) \right), \quad \begin{matrix} m = \lceil \log_2 n/x + n/2 \rceil \\ k = \lceil 2 \log_2 n \rceil \end{matrix}.$$

About $\log(n)$ bits are lost in the subtraction of $m \log(2)$ to obtain $\log(x)$, and the constant $\ln(2)$ is computed by setting $x = 1$.

ArcTan[x]: $\tan^{-1}(x)$ for $0 < x < 1$ is obtained from

$$-\tan^{-1}(x) = \operatorname{Im} \frac{\pi}{2 \operatorname{AGM}_k\left(1, \frac{1+ix}{2^{m-2}}\right)} \left(1 + O\left(\frac{1}{2^{2m}}\right) \right), \quad \begin{matrix} m = \lceil \log_2 n + n/2 \rceil \\ k = \lceil 2 \log_2 n \rceil \end{matrix}.$$

5.3 Other Functions

Factorial[x]: For machine precision computations the formulas

$$\begin{aligned}x! &= \frac{(x+1)!}{x+1}, \\x! &= \frac{\pi x}{\sin(\pi x)} \times \frac{1}{(-x)!}, \\x! &\sim \sqrt{2\pi} x^x e^{-x} \times \frac{x^8 + a_7 x^7 + a_6 x^6 + a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0}{x^8 - a_7 x^7 + a_6 x^6 - a_5 x^5 + a_4 x^4 - a_3 x^3 + a_2 x^2 - a_1 x + a_0},\end{aligned}$$

are used. The last one is accurate to double precision if $\text{Re}(x) > 10$.