The cool package*

nsetzer

September 24, 2006

This is the cool package: a COntent Oriented LaTeX package. That is, it is designed to give LaTeX commands the ability to contain the mathematical meaning while retaining the typesetting versatility.

This package requires the following, non-standard LaTeX packages (all of which are available on `www.ctan.org`): coolstr, coollist, forloop

1 Implementation

```
\newcounter{COOL@ct} %just a general counter
\newcounter{COOL@ct@}%just a general counter
```

1.1 Parenthesis

```
\newcommand{\inp}[2][0cm]{\left(#2\parbox[h][#1]{0cm}{}\right)}
        % in parentheses ()
\newcommand{\inb}[2][0cm]{\left[#2\parbox[h][#1]{0cm}{}\right]}
        % in brackets []
\newcommand{\inbr}[2][0cm]{\left\{#2\parbox[h][#1]{0cm}{}\right\}}
% in braces {}
\newcommand{\inap}[2][0cm]{\left<{#2}\parbox[h][#1]{0cm}{}\right>}
% in angular parentheses <>
\newcommand{\nop}[1]{\left.{#1}\right.}
% no parentheses
```

`\COOL@decide@paren` `\COOL@decide@paren[`⟨*parenthesis type*⟩`]{`⟨*function name*⟩`}{`⟨*contained text*⟩`}`. Since the handling of parentheses is something that will be common to many elements this function will take care of it.

If the optional argument is given, `\COOL@notation@`⟨*function name*⟩`Paren` is ignored and ⟨*parenthesis type*⟩ is used

⟨*parenthesis type*⟩ and `\COOL@notation@`⟨*function name*⟩`Paren` must be one of `none`, `p` for `()`, `b` for `[]`, `br` for `{}`, `ap` for ⟨⟩, `inv` for `\left.\right.`

```
\let\COOL@decide@paren@no@type=\relax
\newcommand{\COOL@decide@paren}[3][\COOL@decide@paren@no@type]{%
\ifthenelse{ \equal{#1}{\COOL@decide@paren@no@type} }%
{%
```

*This document corresponds to cool v1.2, dated 2006/09/17.

```
\def\COOL@decide@paren@type{\csname COOL@notation@#2Paren\endcsname}%
}%
% Else
{%
\def\COOL@decide@paren@type{#1}%
}%
\ifthenelse{ \equal{\COOL@decide@paren@type}{none} }%
{%
#3%
}%
% Else
{%
\ifthenelse{ \equal{\COOL@decide@paren@type}{p} }%
{%
\inp{#3}%
}%
% Else
{%
\ifthenelse{ \equal{\COOL@decide@paren@type}{b} }%
{%
\inb{#3}%
}%
% Else
{%
\ifthenelse{ \equal{\COOL@decide@paren@type}{br} }%
{%
\inbr{#3}%
}%
% Else
{%
\ifthenelse{ \equal{\COOL@decide@paren@type}{ap} }%
{%
\inap{#3}%
}%
% Else
{%
\ifthenelse{ \equal{\COOL@decide@paren@type}{inv} }%
{%
\nop{#3}%
}%
% Else
{%
\PackageError{cool}{Invalid Parenthesis Option}%
{*Paren can only be ‘none’, ‘p’, ‘b’, ‘br’, ‘ap’, ‘inv’}%
}%
}%
}%
}%
}%
```

```
}
```

1.2 Indicies

`\COOL@decide@indicies` `\COOL@decide@indicies{`⟨*function name*⟩`}{`⟨*local indication*⟩`}{`⟨*indicies*⟩`}`

Since up or down indicies can be as common as the parenthesis decision, this macro is the solution.

⟨*local indication*⟩ must be either `u` or `d`

⟨*indicies*⟩ is very likely to be required to be a comma separated list in the near future

the options for indicies are

`local`	allow the indicies to be decided by an optional argument to the function (such as `\LeviCivita[u]{i j}`)
`up`	force the indicies to appear as superscript
`down`	force the indicies to appear as subscript

```
\newcommand{\COOL@decide@indicies}[3]{%
\def\COOL@decide@indicies@placement%
{\csname COOL@notation@#1Indicies\endcsname}%
\ifthenelse{\equal{\COOL@decide@indicies@placement}{local}}%
{%
\ifthenelse{\equal{#2}{u}}%
{^{#3}}%
{_{#3}}%
}%
% Else
{%
\ifthenelse{\equal{\COOL@decide@indicies@placement}{up}}%
{%
{^{#3}}%
}%
% Else
{%
\ifthenelse{\equal{\COOL@decide@indicies@placement}{down}}%
{%
{_{#3}}%
}%
% else
{%
\PackageError{cool}{Invalid Option Sent}%
{#1Indices can only be 'up', 'down', or 'local'}%
}%
}%
}%
}
```

1.3 COntent Oriented LaTeX (COOL)

`\Style` `\Style{`⟨*options*⟩`}` sets the style of the output (how to notate particular functions). ⟨*options*⟩ is a comma delimited list of the form ⟨*key*⟩=⟨*value*⟩, where ⟨*key*⟩ is the *long* form of the command name without the preceeding backslash (i.e. `Integrate` and not `Int` or `\Int`). The list can be in any order and need only contain the styles that the user desires to set.

There can be multiple `\Style` commands within any document—the styled output of the command depends on the last `\Style` command to have specified its style.

For a list of styling options for a command, see the code where the command is defined

```
\newcommand{\Style}[1]{%
\COOL@keyeater#1,\COOL@keystop\COOL@keyend%
}
\newcommand{\COOL@keystop}{@@@}%
\def\COOL@keyeater#1=#2,#3\COOL@keyend{%
\ifx#3\COOL@keystop%
\expandafter\gdef\csname COOL@notation@#1\endcsname{#2}%
\else%
\expandafter\gdef\csname COOL@notation@#1\endcsname{#2}%
\COOL@keyeater#3\COOL@keyend%
\fi%
}
```

`\UseStyleFile` Since notational style should be kept consistent and will likely need to span several documents, use this command to input a notation style file that has previously been prepared. (to be implemented in a future release)

```
\newcommand{\UseStyleFile}[1]{}
```

1.3.1 Fundamental Constants

see `http://functions.wolfram.com/` for the definitions

`\I` The square root of minus 1 ($\sqrt{-1}$), i.

```
\newcommand{\I}{i}
```

`\E` Euler's constant and the base of the natural logarithm, e.

```
\newcommand{\E}{e}
```

`\PI` Pi—the ratio of the circumference of a circle to its diameter, π.

```
\newcommand{\PI}{\pi}
```

`\GoldenRatio` The Golden Ratio, φ

```
\newcommand{\GoldenRatio}{\varphi}
```

`\EulerGamma` Euler's Gamma constant, γ

```
\newcommand{\EulerGamma}{\gamma}
```

`\Catalan` Catalan constant, C

```
\newcommand{\Catalan}{C}
```

`\Glaisher` Glaisher constant, Glaisher

```
\newcommand{\Glaisher}{\mathord{\operatorname{Glaisher}}}
```

`\Khinchin` Khinchin constant, Khinchin

```
\newcommand{\Khinchin}{\mathord{\operatorname{Khinchin}}}
```

1.3.2 Symbols

`\Infinity` Infinity, ∞

```
\newcommand{\Infinity}{\infty}
```

`\Indeterminant` An indeterminant quantity

```
\newcommand{\Indeterminant}{%
\mathchoice%
{\mbox{\textrm>}}%
{\mbox{\small>}}%
{\mbox{\scriptsize>}}%
{\mbox{\tiny>}}%
}
```

`\DirectedInfinity` `\DirInfty` Directed Infinity `\DirectedInfinity{`⟨*complex number*⟩`}` or `\DirInfty{`⟨*complex number*⟩`}`

```
\newcommand{\DirectedInfinity}[1]{#1 \, \infty}
\newcommand{\DirInfty}[1]{\DirectedInfinity{#1}}
```

`\ComplexInfinity` `\CInfty` Complex infinity, $\tilde{\infty}$

```
\newcommand{\ComplexInfinity}{\tilde{\infty}}
\newcommand{\CInfty}{\ComplexInfinity}
```

1.3.3 Exponential Functions

`\Exp` Exponential—for use when e^x won't suffice, $\exp(x)$

```
\newcommand{\COOL@notation@ExpParen}{p}
\newcommand{\Exp}[1]
{%
\exp\COOL@decide@paren{Exp}{#1}%
}
```

`\Log` Logarithm, `\Log{x}`. This function has several options to be set. The usual parentheses, then some about the notation to be used for displaying the symbol.

```
\newcommand{\COOL@notation@LogParen}{none}
```

The following set the symbols:

`LogBaseESymb` can be `ln` or `log`, indicating what symbol should be used for the natural logarithm. If set to `log` then logarithms of base 10 are displayed as $\log_{10}$.

`LogShowBase` can be either `at will` or `always` and decides whether or not one should show the base, as in `log_b x`. If this option is set to `always` then `LogBaseESymb` is ignored.

`\Log{5}`	$\ln 5$	$\ln 5$
`\Log[10]{5}`	$\log 5$	$\log 5$
`\Log[4]{5}`	$\log_4 5$	$\log_4 5$
`\Style{LogBaseESymb=log}`		
`\Log{5}`	$\log 5$	$\log 5$
`\Log[10]{5}`	$\log_{10} 5$	$\log_{10} 5$
`\Log[4]{5}`	$\log_4 5$	$\log_4 5$
`\Style{LogShowBase=always}`		
`\Log{5}`	$\log_e 5$	$\log_e 5$
`\Log[10]{5}`	$\log_{10} 5$	$\log_{10} 5$
`\Log[4]{5}`	$\log_4 5$	$\log_4 5$
`\Style{LogShowBase=at will}`		
`\Log{5}`	$\ln 5$	$\ln 5$
`\Log[10]{5}`	$\log 5$	$\log 5$
`\Log[4]{5}`	$\log_4 5$	$\log_4 5$
`\Style{LogParen=p}`		
`\Log[4]{5}`	$\log_4 (5)$	$\log_4 (5)$

```
\newcommand{\COOL@notation@LogBaseESymb}{ln}% 'ln', 'log'
\newcommand{\COOL@notation@LogShowBase}{at will}% 'at will', 'always'
\newcommand{\Log}[2][\E]
{%
\ifthenelse{ \equal{\COOL@notation@LogShowBase}{at will} }%
{%
\ifthenelse{ \equal{#1}{\E} }%
{%
\ifthenelse{ \equal{\COOL@notation@LogBaseESymb}{ln} }%
{%
\ln \COOL@decide@paren{Log}{#2}%
}%
% Else
{%
\ifthenelse{ \equal{\COOL@notation@LogBaseESymb}{log} }%
{%
\log \COOL@decide@paren{Log}{#2}%
}%
% Else
{%
\PackageError{cool}{Invalid Option Sent}%
{LogBaseESymb can only be `ln' or `log'}%
}%
```

```
}%
}%
% Else
{%
\ifthenelse{ \equal{#1}{10} \AND
\NOT \equal{\COOL@notation@LogBaseESymb}{log}  }%
{%
\log \COOL@decide@paren{Log}{#2}%
}%
% Else
{%
\log_{#1} \COOL@decide@paren{Log}{#2}%
}%
}%
}%
% Else
{%
\ifthenelse{ \equal{\COOL@notation@LogShowBase}{always} }%
{%
\log_{#1}\COOL@decide@paren{Log}{#2}%
}%
% Else
{%
\PackageError{cool}{Invalid Option Sent}%
{LogShowBase can only be 'at will' or 'always'}%
}%
}%
}
```

1.3.4 Trigonometric Functions

`\Sin` The sine function, `\Sin{x}`, $\sin(x)$

```
\newcommand{\COOL@notation@SinParen}{p}
\newcommand{\Sin}[1]{\sin\COOL@decide@paren{Sin}{#1}}
```

`\Cos` The cosine function, `\Cos{x}`, $\cos(x)$

```
\newcommand{\COOL@notation@CosParen}{p}
\newcommand{\Cos}[1]{\cos\COOL@decide@paren{Cos}{#1}}
```

`\Tan` The tangent function, `\Tan{x}`, $\tan(x)$

```
\newcommand{\COOL@notation@TanParen}{p}
\newcommand{\Tan}[1]{\tan\COOL@decide@paren{Tan}{#1}}
```

`\Csc` The cosecant function, `\Csc{x}`, $\csc(x)$

```
\newcommand{\COOL@notation@CscParen}{p}
\newcommand{\Csc}[1]{\csc\COOL@decide@paren{Csc}{#1}}
```

`\Sec` The secant function, `\Sec{x}`, $\sec(x)$

```
\newcommand{\COOL@notation@SecParen}{p}
\newcommand{\Sec}[1]{\sec\COOL@decide@paren{Sec}{#1}}
```

`\Cot` The cotangent function, `\Cot{x}`, $\cot(x)$

```
\newcommand{\COOL@notation@CotParen}{p}
\newcommand{\Cot}[1]{\cot\COOL@decide@paren{Cot}{#1}}
```

1.3.5 Inverse Trigonometric Functions

`\COOL@notation@ArcTrig` The inverse trigoneometric functions style is governed by this global key. It's options are

`inverse` (default), this displays as $\sin^{-1}$
`arc`, this displays as arcsin

```
\def\COOL@notation@ArcTrig{inverse}
```

`\ArcSin` The inverse of the sine function, `\ArcSin{x}`, $\sin^{-1}(x)$

```
\newcommand{\COOL@notation@ArcSinParen}{p}
\newcommand{\ArcSin}[1]{%
\ifthenelse{ \equal{\COOL@notation@ArcTrig}{inverse} }%
{%
\sin^{-1}\COOL@decide@paren{ArcSin}{#1}%
}
% else
{
\ifthenelse{\equal{\COOL@notation@ArcTrig}{arc}}%
{%
\arcsin\COOL@decide@paren{ArcSin}{#1}%
}%
% else
{%
\PackageError{cool}{Invalid option sent}{}%
}%
}%
}
```

`\ArcCos` the inverse of the cosine function, `\ArcCos{x}`, $\cos^{-1}(x)$

```
\newcommand{\COOL@notation@ArcCosParen}{p}
\newcommand{\ArcCos}[1]{%
\ifthenelse{ \equal{\COOL@notation@ArcTrig}{inverse} }%
{%
\cos^{-1}\COOL@decide@paren{ArcCos}{#1}%
}%
% else
{%
\ifthenelse{\equal{\COOL@notation@ArcTrig}{arc}}%
{%
\arccos\COOL@decide@paren{ArcCos}{#1}%
}%
% else
{%
\PackageError{cool}{Invalid option sent}{}%
}%
```

```
}%
}
```

\ArcTan The inverse of the tangent function, \ArcTan{x}, $\tan^{-1}(x)$

```
\newcommand{\COOL@notation@ArcTanParen}{p}
\newcommand{\ArcTan}[1]{%
\ifthenelse{ \equal{\COOL@notation@ArcTrig}{inverse} }%
{%
\tan^{-1}\COOL@decide@paren{ArcTan}{#1}%
}%
% else
{%
\ifthenelse{\equal{\COOL@notation@ArcTrig}{arc}}%
{%
\arctan\COOL@decide@paren{ArcTan}{#1}%
}%
% else
{%
\PackageError{cool}{Invalid option sent}{}%
}%
}%
}
```

\ArcCsc The Inverse Cosecant function, \ArcCsc{x}, $\csc^{-1}(x)$

```
\newcommand{\COOL@notation@ArcCscParen}{p}
\newcommand{\ArcCsc}[1]{\csc^{-1}\COOL@decide@paren{ArcCsc}{#1}}
```

\ArcSec The inverse secant function, \ArcSec{x}, $\sec^{-1}(x)$

```
\newcommand{\COOL@notation@ArcSecParen}{p}
\newcommand{\ArcSec}[1]{\sec^{-1}\COOL@decide@paren{ArcSec}{#1}}
```

\ArcCot The inverse cotangent function, \ArcCot{x}, $\cot^{-1}(x)$

```
\newcommand{\COOL@notation@ArcCotParen}{p}
\newcommand{\ArcCot}[1]{\cot^{-1}\COOL@decide@paren{ArcCot}{#1}}
```

1.3.6 Hyperbolic Functions

\Sinh Hyperbolic sine, \Sinh{x}, $\sinh(x)$

```
\newcommand{\COOL@notation@SinhParen}{p}
\newcommand{\Sinh}[1]{\sinh\COOL@decide@paren{Sinh}{#1}}
```

\Cosh Hyperbolic cosine, \Cosh{x}, $\cosh(x)$

```
\newcommand{\COOL@notation@CoshParen}{p}
\newcommand{\Cosh}[1]{\cosh\COOL@decide@paren{Cosh}{#1}}
```

\Tanh Hyperbolic Tangent, \Tanh{x}, $\tanh(x)$

```
\newcommand{\COOL@notation@TanhParen}{p}
\newcommand{\Tanh}[1]{\tanh\COOL@decide@paren{Tanh}{#1}}
```

`\Csch` Hyperbolic cosecant `\Csch{x}`, $\operatorname{csch}(x)$

```
\newcommand{\COOL@notation@CschParen}{p}
\DeclareMathOperator{\csch}{csch}
\newcommand{\Csch}[1]{\csch\COOL@decide@paren{Csch}{#1}}
```

`\Sech` Hyperbolic secant, `\Sech{x}`, $\operatorname{sech}(x)$

```
\newcommand{\COOL@notation@SechParen}{p}
\DeclareMathOperator{\sech}{sech}
\newcommand{\Sech}[1]{\sech\COOL@decide@paren{Sech}{#1}}
```

`\Coth` Hyperbolic Cotangent, `\Coth{x}`, $\coth(x)$

```
\newcommand{\COOL@notation@CothParen}{p}
\newcommand{\Coth}[1]{\coth\COOL@decide@paren{Coth}{#1}}
```

1.3.7 Inverse Hyperbolic Functions

`\ArcSinh` Inverse hyperbolic sine, `\ArcSinh{x}`, $\sinh^{-1}(x)$

```
\newcommand{\COOL@notation@ArcSinhParen}{p}
\newcommand{\ArcSinh}[1]{\sinh^{-1}\COOL@decide@paren{ArcSinh}{#1}}
```

`\ArcCosh` Inverse hyperbolic cosine, `\ArcCosh{x}`, $\cosh^{-1}(x)$

```
\newcommand{\COOL@notation@ArcCoshParen}{p}
\newcommand{\ArcCosh}[1]{\cosh^{-1}\COOL@decide@paren{ArcCosh}{#1}}
```

`\ArcTanh` Inverse hyperbolic tangent, `\ArcTanh{x}`, $\tanh^{-1}(x)$

```
\newcommand{\COOL@notation@ArcTanhParen}{p}
\newcommand{\ArcTanh}[1]{\tanh^{-1}\COOL@decide@paren{ArcTanh}{#1}}
```

`\ArcCsch` Inverse hyperbolic cosecant, `\ArcCsch{x}`, $\operatorname{csch}^{-1}(x)$

```
\newcommand{\COOL@notation@ArcCschParen}{p}
\newcommand{\ArcCsch}[1]{\csch^{-1}\COOL@decide@paren{ArcCsch}{#1}}
```

`\ArcSech` Inverse hyperbolic secant, `\ArcSech{x}`, $\operatorname{sech}^{-1}(x)$

```
\newcommand{\COOL@notation@ArcSechParen}{p}
\newcommand{\ArcSech}[1]{\sech^{-1}\COOL@decide@paren{ArcSech}{#1}}
```

`\ArcCoth` Inverse hyperbolic cotangent, `\ArcCoth{x}`, $\coth^{-1}(x)$

```
\newcommand{\COOL@notation@ArcCothParen}{p}
\newcommand{\ArcCoth}[1]{\coth^{-1}\COOL@decide@paren{ArcCoth}{#1}}
```

1.3.8 Product Logarithms

`\LambertW` Lambert Function. `\LambertW` is an alias for `\ProductLog` and its properties are therefore set using that function

```
\newcommand{\LambertW}[1]{\ProductLog{#1}}
```

`\ProductLog` Generalized Lambert Function `\ProductLog{`[⟨*index*⟩`,`]⟨*variable*⟩`}`.

Lambert Function	`\ProductLog{x}`	$W(x)$
Generalized Lambert Function	`\ProductLog{k,x}`	$W_k(x)$

```
\newcommand{\COOL@notation@ProductLogParen}{p}
\newcommand{\ProductLog}[1]{%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer}=1}%
{%
W\!\COOL@decide@paren{ProductLog}{#1}%
}%
% else
{%
\ifthenelse{\value{COOL@listpointer}=2}%
{%
W_{\listval{#1}{1}}\!\COOL@decide@paren{ProductLog}{\listval{#1}{2}}%
}%
% else
{%
\PackageError{cool}{`ProductLog' Invaid Argument}%
{Must have a comma separated list of length 1 or 2}
}%
}%
}
```

1.3.9 Max and Min

`\Max` the maximum function, `\Max{x,y,z}`, $\max(x, y, z)$

```
\newcommand{\COOL@notation@MaxParen}{p}
\newcommand{\Max}[1]{\max\COOL@decide@paren{Max}{#1}}
```

`\Min` the minimum function, `\Min{x,y,z}`, $\min(x, y, z)$

```
\newcommand{\COOL@notation@MinParen}{p}
\newcommand{\Min}[1]{\min\COOL@decide@paren{Min}{#1}}
```

1.3.10 Bessel Functions

`\BesselJ` Bessel Function of the first kind, `\BesselJ{\nu}{x}`, $J_\nu(x)$

```
\newcommand{\COOL@notation@BesselJSymb}{J}
\newcommand{\COOL@notation@BesselJParen}{p}
\newcommand{\BesselJ}[2]%
{\COOL@notation@BesselJSymb_{#1}\!\COOL@decide@paren{BesselJ}{#2}}
```

`\BesselY` Bessel Function of the second kind, `\BesselY{\nu}{x}`, $Y_\nu(x)$

```
\newcommand{\COOL@notation@BesselYSymb}{Y}
\newcommand{\COOL@notation@BesselYParen}{p}
\newcommand{\BesselY}[2]%
{\COOL@notation@BesselYSymb_{#1}\!\COOL@decide@paren{BesselY}{#2}}
```

`\BesselI` Modified Bessel Function of the first kind, `\BesselI{\nu}{x}`, $I_\nu(x)$

```
\newcommand{\COOL@notation@BesselISymb}{I}
\newcommand{\COOL@notation@BesselIParen}{p}
\newcommand{\BesselI}[2]%
{\COOL@notation@BesselISymb_{#1}\!\COOL@decide@paren{BesselI}{#2}}
```

`\BesselK` Modified Bessel Function of the second kind, `\BesselK{\nu}{x}`, $K_\nu(x)$

```
\newcommand{\COOL@notation@BesselKSymb}{K}
\newcommand{\COOL@notation@BesselKParen}{p}
\newcommand{\BesselK}[2]%
{\COOL@notation@BesselKSymb_{#1}\!\COOL@decide@paren{BesselK}{#2}}
```

1.3.11 Airy Functions

`\AiryAi` Airy Ai Function, `\AiryAi{x}`, $\mathrm{Ai}(x)$

```
\newcommand{\COOL@notation@AiryAiParen}{p}
\DeclareMathOperator{\AiryAiSymb}{Ai}
\newcommand{\AiryAi}[1]{\AiryAiSymb\!\COOL@decide@paren{AiryAi}{#1}}
```

`\AiryBi` Airy Bi Function, `\AiryBi{x}`, $\mathrm{Bi}(x)$

```
\newcommand{\COOL@notation@AiryBiParen}{p}
\DeclareMathOperator{\AiryBiSymb}{Bi}
\newcommand{\AiryBi}[1]{\AiryBiSymb\!\COOL@decide@paren{AiryBi}{#1}}
```

1.3.12 Struve Functions

`\StruveH` Struve H function, `\StruveH{\nu}{z}`, $\mathbf{H}_\nu(z)$

```
\newcommand{\COOL@notation@StruveHParen}{p}
\newcommand{\StruveH}[2]{ {\bf H}_{#1}\!\COOL@decide@paren{StruveH}{#2}}
```

`\StruveL` Struve L function, `\StruveL{\nu}{z}`, $\mathbf{L}_\nu(z)$

```
\newcommand{\COOL@notation@StruveLParen}{p}
\newcommand{\StruveL}[2]{ {\bf L}_{#1}\!\COOL@decide@paren{StruveL}{#2}}
```

1.3.13 Integer Functions

`\Floor` floor, `\Floor{x}`, $\lfloor x \rfloor$

```
\newcommand{\Floor}[1]{\lfloor #1 \rfloor}
```

`\Ceiling` ceiling, `\Ceiling{x}`, $\lceil x \rceil$

```
\newcommand{\Ceiling}[1]{\lceil #1 \rceil}
```

`\Round` round, `\Round{x}`, $\lfloor x \rceil$

```
\newcommand{\Round}[1]{\lfloor #1 \rceil}
```

\iPart \IntegerPart The integer part of a real number, `\iPart{x}`, `\IntegerPart{x}`, $\mathrm{int}\,(x)$

```
\newcommand{\COOL@notation@IntegerPartParen}{p}
\DeclareMathOperator{\iPartSymb}{int}
\newcommand{\iPart}[1]{\iPartSymb\COOL@decide@paren{IntegerPart}{#1}}
\newcommand{\IntegerPart}[1]{\iPart{#1}}
```

\fPart \FractionalPart the fractional part of a real number, `\fPart{x}`, `\FractionalPart{x}`, $\mathrm{frac}\,(x)$

```
\newcommand{\COOL@notation@FractionalPartParen}{p}
\DeclareMathOperator{\fPartSymb}{frac}
\newcommand{\fPart}[1]{\fPartSymb\COOL@decide@paren{FractionalPart}{#1}}
\newcommand{\FractionalPart}[1]{\fPart{#1}}
```

\Mod Modulo, `\Mod{n}{m}`, $n \mod m$

```
\newcommand{\COOL@notation@ModDisplay}{mod}
\newcommand{\Mod}[2]{%
\ifthenelse{\equal{\COOL@notation@ModDisplay}{mod}}%
{%
#1 \mod #2%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@ModDisplay}{bmod}}%
{%
#1 \bmod #2%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@ModDisplay}{pmod}}%
{%
#1 \pmod #2%
}%
% ElseIf
{\ifthenelse{\equal{\COOL@notation@ModDisplay}{pod}}%
{%
#1 \pod #2%
}%
% Else
{%
\PackageError{cool}{Invalid Option Sent}%
{ModDisplay can only be ‘mod’, ‘bmod’, ‘pmod’, or ‘pod’}%
}}}}%
}
```

\Quotient quotient, `\Quotient{m}{n}`, $\mathrm{quotient}\,(m, n)$

```
\newcommand{\COOL@notation@QuotientParen}{p}
\DeclareMathOperator{\QuotientSymb}{quotient}
\newcommand{\Quotient}[2]%
{\QuotientSymb\COOL@decide@paren{Quotient}{#1,#2}}
```

\GCD greatest common divisor, `\GCD{n_1,n_2,\dots,n_m}`, $\gcd\,(n_1, n_2, \ldots, n_m)$

```
\newcommand{\COOL@notation@GCDParen}{p}
\newcommand{\GCD}[1]{\gcd\COOL@decide@paren{GCD}{#1}}
```

\ExtendedGCD Extended Greatest Common Divisor,
\EGCD \EGCD{n}{m}, \ExtendedGCD{n}{m}, $\gcd(n,m)$

```
\newcommand{\COOL@notation@ExtendedGCDParen}{p}
\DeclareMathOperator{\ExtendedGCDSymb}{egcd}
\newcommand{\ExtendedGCD}[2]%
{\ExtendedGCDSymb\COOL@decide@paren{ExtendedGCD}{#1,#2}}
\newcommand{\EGCD}[2]{\ExtendedGCD{#1}{#2}}
```

\LCM Least Common Multiple, \LCM{n_1,n_2,\ldots,n_m}, $\operatorname{lcm}(n_1, n_2, \ldots, n_m)$

```
\newcommand{\COOL@notation@LCMParen}{p}
\DeclareMathOperator{\LCMSymb}{lcm}
\newcommand{\LCM}[1]{\LCMSymb\COOL@decide@paren{LCM}{#1}}
```

\Fibonacci Fibonacci number, \Fibonacci{n}, F_n, and
Fibonacci Polynomial, \Fibonacci{n,x}, $F_n(x)$

```
\newcommand{\COOL@notation@FibonacciParen}{p}
\newcommand{\Fibonacci}[1]{%
\liststore{#1}{COOL@Fibonacci@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 1}%
{%
F_{#1}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 2}%
{%
F_{\COOL@Fibonacci@arg@i}\!%
\COOL@decide@paren{Fibonacci}{\COOL@Fibonacci@arg@ii}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`Fibonacci' can only accept a
comma separate list of length 1 or 2}%
}}%
}
```

\Euler Euler number, \Euler{n}, E_n, and Euler Polynomial, \Euler{n,x}, $E_n(x)$

```
\newcommand{\COOL@notation@EulerParen}{p}
\newcommand{\Euler}[1]{%
\liststore{#1}{COOL@Euler@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 1}%
{%
E_{#1}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 2}%
{%
```

```
E_{\COOL@Euler@arg@i}\!%
\COOL@decide@paren{Euler}{\COOL@Euler@arg@ii}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`Euler' can only accept a
comma separate list of length 1 or 2}%
}}%
}
```

`\Bernoulli` Bernoulli number, `\Bernoulli{n}`, B_n and Bernoulli Polynomial `\Bernoulli{n,x}`, $B_n(x)$

```
\newcommand{\COOL@notation@BernoulliParen}{p}
\newcommand{\Bernoulli}[1]{%
\liststore{#1}{COOL@Bernoulli@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 1}%
{%
B_{#1}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 2}%
{%
B_{\COOL@Bernoulli@arg@i}\!%
\COOL@decide@paren{Bernoulli}{\COOL@Bernoulli@arg@ii}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`Bernoulli' can only accept a
comma separate list of length 1 or 2}%
}}%
}
```

`\StirlingSOne` Stirling number of the first kind `\StirlingSOne{n}{m}`, $S_n^{(m)}$

```
\newcommand{\StirlingSOne}[2]{S_{#1}^{\inp{#2}}}
```

`\StirlingSTwo` Stirling number of the second kind, `\StirlingSTwo{n}{m}`, $\mathcal{S}_n^{(m)}$

```
\newcommand{\StirlingSTwo}[2]{{\cal S}_{#1}^{\inp{#2}}}
```

`\PartitionsP` Number of unrestricted partitions of an integer, `\PartitionsP{x}`, $p(x)$

```
\newcommand{\COOL@notation@PartitionsPParen}{p}
\newcommand{\PartitionsP}[1]{p\!\COOL@decide@paren{PartitionsP}{#1}}
```

`\PartitionsQ` number of partitions of an integer into distinct parts, `\PartitionsQ{x}`, $q(x)$

```
\newcommand{\COOL@notation@PartitionsQParen}{p}
\newcommand{\PartitionsQ}[1]{q\!\COOL@decide@paren{PartitionsQ}{#1}}
```

\DiscreteDelta Discrete delta function,
\DiscreteDelta{n_1,n_2,\ldots,n_m}, $\delta\left(n_1, n_2, \ldots, n_m\right)$

```
\newcommand{\COOL@notation@DiscreteDeltaParen}{p}
\newcommand{\DiscreteDelta}[1]%
{\delta\COOL@decide@paren{DiscreteDelta}{#1}}
```

\KroneckerDelta Kronecker Delta, \KroneckerDelta{n_1,n_2,\ldots,n_m}, $\delta^{n_1 n_2 \ldots n_m}$

```
\newcommand{\COOL@notation@KroneckerDeltaUseComma}{false}%
\newcommand{\COOL@notation@KroneckerDeltaIndicies}{local}
\newcommand{\KroneckerDelta}[2][u]{%
\liststore{#2}{COOL@arg@}%
\listval{#2}{0}%
\def\COOL@arg@temp{}%
\forLoop{1}{\value{COOL@listpointer}}{COOL@ct}%
{%
\ifthenelse{\equal{\COOL@notation@KroneckerDeltaUseComma}{true}}%
{%
\ifthenelse{\NOT \value{COOL@ct} = 1}
{%
\edef\COOL@arg@temp%
{\COOL@arg@temp, \csname COOL@arg@\roman{COOL@ct}\endcsname}%
}%
% Else
{%
\edef\COOL@arg@temp%
{\COOL@arg@temp \csname COOL@arg@\roman{COOL@ct}\endcsname}%
}%
}%
% Else
{%
\edef\COOL@arg@temp%
{\COOL@arg@temp \csname COOL@arg@\roman{COOL@ct}\endcsname}%
}%
}%
\delta\COOL@decide@indicies{KroneckerDelta}{#1}{\COOL@arg@temp}%
}
```

\LeviCivita Levi-Civita totally anti-symmetric Tensor density,
\Signature \LeviCivita{n_1,n_2,\ldots,n_m}, $\epsilon^{n_1 n_2 \ldots n_m}$

```
\newcommand{\COOL@notation@LeviCivitaUseComma}{false}
\newcommand{\COOL@notation@LeviCivitaIndicies}{local}
\newcommand{\LeviCivita}[2][u]{%
\liststore{#2}{COOL@arg@}%
\listval{#2}{0}%
\def\COOL@arg@temp{}%
\forLoop{1}{\value{COOL@listpointer}}{COOL@ct}%
{%
\ifthenelse{\equal{\COOL@notation@LeviCivitaUseComma}{true}}%
{%
```

```
\ifthenelse{\NOT \value{COOL@ct} = 1}{,}{}%
\edef\COOL@arg@temp%
{\COOL@arg@temp \csname COOL@arg@\roman{COOL@ct}\endcsname}%
}%
% Else
{%
\edef\COOL@arg@temp%
{\COOL@arg@temp \csname COOL@arg@\roman{COOL@ct}\endcsname}%
}%
}%
\epsilon\COOL@decide@indicies{LeviCivita}{#1}{\COOL@arg@temp}%
}%
\newcommand{\Signature}[2][u]{\LeviCivita[#1]{#2}}
```

1.3.14 Classical Orthogonal Polynomials

`\HermiteH` Hermite Polynomial, `\HermiteH{n}{x}`, $H_n(x)$

```
\newcommand{\COOL@notation@HermiteHParen}{p}
\newcommand{\COOL@notation@HermiteHSymb}{H}
\newcommand{\HermiteH}[2]%
{\COOL@notation@HermiteHSymb_{#1}\!\COOL@decide@paren{HermiteH}{#2}}
```

`\LaugerreL` Laugerre Polynomial, `\LaugerreL{\nu,x}`, $L_\nu(x)$ and
Generalized Laugerre Polynomial `\LaugerreL{\nu,\lambda,x}`, $L_\nu^\lambda(x)$

```
\newcommand{\COOL@notation@LaugerreLParen}{p}
\newcommand{\COOL@notation@LaugerreLSymb}{L}
\newcommand{\LaugerreL}[1]{%
\liststore{#1}{COOL@list@temp@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer}=2}%
{%
\COOL@notation@LaugerreLSymb_{\COOL@list@temp@i}%
\!\COOL@decide@paren{LaugerreL}{\COOL@list@temp@ii}%
}%
% Else If
{ \ifthenelse{\value{COOL@listpointer}=3}%
{%
\COOL@notation@LaugerreLSymb_{\COOL@list@temp@i}^{\COOL@list@temp@ii}%
\!\COOL@decide@paren{LaugerreL}{\COOL@list@temp@iii}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`LaugerrL' only accepts a comma separated list of length 2 or 3}%
}}%
}
```

`\LegendreP` Legendre Polynomials

Legendre Polynomial	`\LegendreP{n,x}`	$P_n(x)$
Associated Legendre Polynomial of the first kind of type 2	`\LegendreP{\ell,m,x}`	$P_\ell^m(x)$
	`\LegendreP{\ell,m,2,x}`	$P_\ell^m(x)$
Associated Legendre Function of the first kind of type 3	`\LegendreP{\ell,m,3,x}`	$\mathcal{P}_\ell^m(x)$

```
\newcommand{\COOL@notation@LegendrePParen}{p}
\newcommand{\COOL@notation@LegendrePSymb}{P}
\newcommand{\LegendreP}[1]{%
\liststore{#1}{COOL@LegendreP@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 2}%
{%
\COOL@notation@LegendrePSymb_{\COOL@LegendreP@arg@i}\!%
\COOL@decide@paren{LegendreP}{\COOL@LegendreP@arg@ii}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 3}%
{%
\COOL@notation@LegendrePSymb_{\COOL@LegendreP@arg@i}%
^{\COOL@LegendreP@arg@ii}\!%
\COOL@decide@paren{LegendreP}{\COOL@LegendreP@arg@iii}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 4}%
{%
\isint{\COOL@LegendreP@arg@iii}%
\ifthenelse{\boolean{COOL@isint}}%
{%
\ifcase\COOL@LegendreP@arg@iii\relax%
\PackageError{cool}{Invalid Argument}%
{‘LegendreP’ third argument must be $>$ 1}%
\or%
\PackageError{cool}{Invalid Argument}%
{‘LegendreP’ third argument must be $>$ 1}%
\or%
\COOL@notation@LegendrePSymb_{\COOL@LegendreP@arg@i}%
^{\COOL@LegendreP@arg@ii}\!%
\COOL@decide@paren{LegendreP}{\COOL@LegendreP@arg@iv}%
\or%
{\cal P}_{\COOL@LegendreP@arg@i}%
^{\COOL@LegendreP@arg@ii}\!%
\COOL@decide@paren{LegendreP}{\COOL@LegendreP@arg@iv}%
\else%
\PackageError{cool}{Invalid Argument}{unsupported}%
\fi%
}
% Else
{%
```

```
\PackageError{cool}{Invalid Argument}{third arg must be int}%
}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`LegendreP' can only accept a%
 comma separated list of length 2-4}%
}}}%
}
```

`\LegendreQ` Legendre Polynomials of the second kind

Legendre Polynomial	`\LegendreQ{n,x}`	$Q_n(x)$
Associated Legendre Polynomial of the second kind of type 2	`\LegendreQ{\ell,m,x}`	$Q_\ell^m(x)$
	`\LegendreQ{\ell,m,2,x}`	$Q_\ell^m(x)$
Associated Legendre Function of the second kind of type 3	`\LegendreQ{\ell,m,3,x}`	$\mathcal{Q}_\ell^m(x)$

```
\newcommand{\COOL@notation@LegendreQParen}{p}
\newcommand{\COOL@notation@LegendreQSymb}{Q}
\newcommand{\LegendreQ}[1]{%
\liststore{#1}{COOL@LegendreQ@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 2}%
{%
\COOL@notation@LegendreQSymb_{\COOL@LegendreQ@arg@i}\!%
\COOL@decide@paren{LegendreQ}{\COOL@LegendreQ@arg@ii}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 3}%
{%
\COOL@notation@LegendreQSymb_{\COOL@LegendreQ@arg@i}%
^{\COOL@LegendreQ@arg@ii}\!%
\COOL@decide@paren{LegendreQ}{\COOL@LegendreQ@arg@iii}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 4}%
{%
\isint{\COOL@LegendreQ@arg@iii}%
\ifthenelse{\boolean{COOL@isint}}%
{%
\ifcase\COOL@LegendreQ@arg@iii\relax%
\PackageError{cool}{Invalid Argument}%
{`LegendreQ' third argument must be $>$ 1}%
\or%
\PackageError{cool}{Invalid Argument}%
{`LegendreQ' third argument must be $>$ 1}%
\or%
\COOL@notation@LegendreQSymb_{\COOL@LegendreQ@arg@i}%
```

```
^{\COOL@LegendreQ@arg@ii}\!%
\COOL@decide@paren{LegendreQ}{\COOL@LegendreQ@arg@iv}%
\or%
{\cal Q}_{\COOL@LegendreQ@arg@i}%
^{\COOL@LegendreQ@arg@ii}\!%
\COOL@decide@paren{LegendreQ}{\COOL@LegendreQ@arg@iv}%
\else%
\PackageError{cool}{Invalid Argument}{unsupported}%
\fi%
}
% Else
{%
\PackageError{cool}{Invalid Argument}{third arg must be int}%
}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`LegendreQ' can only accept a%
 comma separated list of length 2-4}%
}}}%
}
```

`\ChebyshevT` Chebyshev Polynomial of the first kind, `ChebyshevT{n}{x}`, $ChebyshevTnx$

```
\newcommand{\COOL@notation@ChebyshevTParen}{p}
\newcommand{\COOL@notation@ChebyshevTSymb}{T}
\newcommand{\ChebyshevT}[2]%
{\COOL@notation@ChebyshevTSymb_{#1}\!\COOL@decide@paren{ChebyshevT}{#2}}
```

`\ChebyshevU` , `\ChebyshevU{n}{z}`, $U_n(z)$ Chebyshev Polynomial of the second kind

```
\newcommand{\COOL@notation@ChebyshevUParen}{p}
\newcommand{\COOL@notation@ChebyshevUSymb}{U}
\newcommand{\ChebyshevU}[2]%
{\COOL@notation@ChebyshevUSymb_{#1}\!\COOL@decide@paren{ChebyshevU}{#2}}
```

`\JacobiP` Jacobi Polynomial, `\JacobiP{n}{a}{b}{x}`, $P_n^{(a,b)}(x)$

```
\newcommand{\COOL@notation@JacobiPParen}{p}
\newcommand{\COOL@notation@JacobiPSymb}{P}
\newcommand{\JacobiP}[4]{%
\COOL@notation@JacobiPSymb_{#1}^{\inp{#2, #3}}%
\!\COOL@decide@paren{JacobiP}{#4}%
}
```

1.3.15 Associated Polynomials

`\AssocLegendreP` Associated Legendre Polynomial of the first kind of type 2 `\AssocLegendreP{\ell}{m}{x}`, $P_\ell^m(x)$

```
\newcommand{\AssocLegendreP}[3]{\LegendreP{#1,#2,#3}}
```

\AssocLegendreQ Associated Legendre Polynomial of the second kind of type 2 \AssocLegendreQ{\ell}{m}{x}, $Q_\ell^m(x)$

```
\newcommand{\AssocLegendreQ}[3]{\LegendreQ{#1,#2,#3}}
```

\GegenbauerC Gegenbauer Polynomial, \GegenbauerC{n}{\lambda}{x}, $C_n^\lambda(x)$

```
\newcommand{\COOL@notation@GegenbauerCParen}{p}
\newcommand{\COOL@notation@GegenbauerCSymb}{C}
\newcommand{\GegenbauerC}[3]{%
\COOL@notation@GegenbauerCSymb_{#1}^{#2}%
\!\COOL@decide@paren{GegenbauerC}{#3}%
}
```

\SphericalHarmonicY \SphericalHarmY \SpHarmY Spherical Harmonic, \SpHarmY{\ell}{m}{\theta}{\phi}, \SphericalHarmY{\ell}{m}{\theta}{\phi}, \SphericalHarmonicY{\ell}{m}{\theta}{\phi}, $Y_\ell^m(\theta,\phi)$

```
\newcommand{\COOL@notation@SphericalHarmonicYParen}{p}
\newcommand{\COOL@notation@SphericalHarmonicYSymb}{Y}
\newcommand{\SphericalHarmonicY}[4]{%
\COOL@notation@SphericalHarmonicYSymb_{#1}^{#2}%
\!\COOL@decide@paren{SphericalHarmonicY}{#3,#4}%
}
\newcommand{\SphericalHarmY}[4]{\SphericalHarmonicY{#1}{#2}{#3}{#4}}
\newcommand{\SpHarmY}[4]{\SphericalHarmonicY{#1}{#2}{#3}{#4}}
```

1.3.16 Other Polynomials

\CyclotomicC Cyclotomic Polynomial, \CyclotomicC{n}{z}, $C_n(z)$

```
\newcommand{\COOL@notation@CyclotomicCParen}{p}
\newcommand{\CyclotomicC}[2]%
{C_{#1}\!\COOL@decide@paren{CyclotomicC}{#2}}
```

\FibonacciF Fibonacci Polynomial, \FibonacciF{n}{z}, $F_n(z)$

```
\newcommand{\FibonacciF}[2]{\Fibonacci{#1,#2}}
```

\EulerE Euler Polynomial, \EulerE{n}{z}, $E_n(z)$

```
\newcommand{\EulerE}[2]{\Euler{#1,#2}}
```

\BernoulliB Bernoulli Polynomial, \BernoulliB{n}{z}, $B_n(z)$

```
\newcommand{\BernoulliB}[2]{\Bernoulli{#1,#2}}
```

1.3.17 Factorial Functions

\Factorial Factorial, \Factorial{n}, $n!$

```
\newcommand{\Factorial}[1]{#1!}
```

\DblFactorial Double Factorial, \DblFactorial{n}, $n!!$

```
\newcommand{\DblFactorial}[1]{#1!!}
```

`\Binomial` binomial, `\Binomial{n}{r}`, $\binom{n}{r}$

```
\newcommand{\Binomial}[2]{ \binom{#1}{#2} }
```

`\Multinomial` Multinomial, `\Multinomial{n_1,\ldots,n_m}`, $(n_1 + \ldots + n_m; n_1, \ldots, n_m)$

```
\newcommand{\Multinomial}[1]%
{%
\listval{#1}{0}% get the length of the list
\setcounter{COOL@listlen}{\value{COOL@listpointer}}% record length
\liststore{#1}{COOL@list@temp@}%
\isint{\COOL@list@temp@i}% check that the entries are integers
\setcounter{COOL@ct}{2}%
\whiledo{ \boolean{COOL@isint} \AND
\NOT \value{COOL@ct}>\value{COOL@listlen}  }%
{%
\def\COOL@Multinomial@tempa%
{\csname COOL@list@temp@\roman{COOL@ct}\endcsname}%
\isint{\COOL@Multinomial@tempa}%
\stepcounter{COOL@ct}%
}%
\ifthenelse{\boolean{COOL@isint}}%
{%
% all of them are integers
\setcounter{COOL@ct@}{ \COOL@list@temp@i }% records the sum
\forLoop{2}{\value{COOL@listlen}}{COOL@ct}%
{%
\addtocounter{COOL@ct@}%
{\csname COOL@list@temp@\roman{COOL@ct}\endcsname}%
}%
\left(\arabic{COOL@ct@}%
}%
% Else
{%
\left(%
\listval{#1}{1}%
\forLoop{2}{\value{COOL@listlen}}{COOL@ct}%
{%
+ \listval{#1}{\arabic{COOL@ct}}%
}%
}%
;#1\right)%
}
```

1.3.18 Gamma Functions

`\GammaFunc` Gamma Function

Gamma Function	`\GammaFunc{z}`	$\Gamma(z)$
Incomplete Gamma Function	`\GammaFunc{a,z}`	$\Gamma(a,z)$
Generalized Incomplete Gamma Function	`\GammaFunc{a,x,y}`	$\Gamma(a,x,y)$

```
\newcommand{\COOL@notation@GammaFuncParen}{p}
```

```
\newcommand{\GammaFunc}[1]{%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 1}%
{%
\Gamma\!\COOL@decide@paren{GammaFunc}{#1}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 2}%
{%
\Gamma\!\COOL@decide@paren{GammaFunc}{#1}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 3}%
{%
\Gamma\!\COOL@decide@paren{GammaFunc}{#1}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`GammaFunc' can only accept a comma separate list of length 1 to 3}%
}%
}}%
}
```

`\IncGamma` incomplete Gamma function, `\IncGamma{a}{x}`, $\Gamma(a,x)$

```
\newcommand{\IncGamma}[2]{\GammaFunc{#1,#2}}
```

`\GenIncGamma` Generalized Incomplete Gamma, `\GenIncGamma{a}{x}{y}`, $\Gamma(a,x,y)$

```
\newcommand{\GenIncGamma}[3]{\GammaFunc{#1, #2, #3}}
```

`\GammaRegularized` `\RegIncGamma` `\GammaReg` Regularized Incomplete Gamma

`\GammaRegularized{a,x}`	$Q(a,x)$
`\RegIncGamma{a}{x}`	$Q(a,x)$
`\GammaReg{a,x}`	$Q(a,x)$

```
\newcommand{\COOL@notation@GammaRegularizedParen}{p}%
\newcommand{\GammaRegularized}[1]{%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 2}%
{%
Q\!\COOL@decide@paren{GammaRegularized}{#1}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 3}%
{%
Q\!\COOL@decide@paren{GammaRegularized}{#1}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
```

```
{‘GammaRegularized’ can only accept comma%
 separated lists of length 2 or 3}%
}%
}%
}
\newcommand{\RegIncGamma}[2]{\GammaRegularized{#1, #2}}
\newcommand{\GammaReg}[1]{\GammaRegularized{#1}}
```

\RegIncGammaInv
\InverseGammaRegularized
\GammaRegInv

Inverse of Regularized Incomplete Gamma,

`\RegIncGammaInv{a}{x}`	$Q^{-1}(a,x)$
`\InverseGammaRegularized{a,x}`	$Q^{-1}(a,x)$
`\GammaRegInv{a,x}`	$Q^{-1}(a,x)$

```
\newcommand{\COOL@notation@InverseGammaRegularizedParen}{p}
\newcommand{\InverseGammaRegularized}[1]{%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 2}%
{%
Q^{-1}\!\COOL@decide@paren{InverseGammaRegularized}{#1}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 3}%
{%
Q^{-1}\!\COOL@decide@paren{InverseGammaRegularized}{#1}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{‘InverseGammaRegularized’ can only accept%
 a comma separated list of length 2 or 3}%
}%
}%
}
\newcommand{\RegIncGammaInv}[2]{\InverseGammaRegularized{#1, #2}}
\newcommand{\GammaRegInv}[1]{\InverseGammaRegularized{#1}}
```

\GenRegIncGamma

Generalized Regularized Incomplete Gamma

`\GenRegIncGamma{a}{x}{y}`	$Q(a,x,y)$
`\GammaRegularized{a,x,y}`	$Q(a,x,y)$

```
\newcommand{\GenRegIncGamma}[3]{\GammaRegularized{#1, #2, #3}}
```

\GenRegIncGammaInv

Inverse of Gen. Reg. Incomplete Gamma, `\GenRegIncGammaInv{a}{x}{y}`, $Q^{-1}(a,x,y)$

```
\newcommand{\GenRegIncGammaInv}[3]{\InverseGammaRegularized{#1, #2, #3}}
```

\Pochhammer

Pochhammer Symbol `\Pochhammer{a}{n}`, $(a)_n$

```
\newcommand{\Pochhammer}[2]{\inp{#1}_{#2}}
```

\LogGamma

Log Gamma Function, `\LogGamma{x}`, $\log\Gamma(x)$

```
\newcommand{\COOL@notation@LogGammaParen}{p}
\DeclareMathOperator{\LogGammaSymb}{log\Gamma}
\newcommand{\LogGamma}[1]{\LogGammaSymb\COOL@decide@paren{LogGamma}{#1}}
```

1.3.19 Derivatives of Gamma Functions

\DiGamma Digamma function, `\DiGamma{x}`, $\digamma(x)$

```
\newcommand{\COOL@notation@DiGammaParen}{p}
\newcommand{\DiGamma}[1]{\digamma\!\COOL@decide@paren{DiGamma}{#1}}
```

PolyGamma function, `\PolyGamma{\nu}{x}`, $\psi^{(\nu)}(x)$

\PolyGamma

```
\newcommand{\COOL@notation@PolyGammaParen}{p}
\newcommand{\PolyGamma}[2]%
{\psi^{\inp{#1}}\!\COOL@decide@paren{PolyGamma}{#2}}
```

\HarmNum Harmonic Number

Harmonic Number	`\HarmNum{x}`	H_x
General Harmonic Number	`\HarmNum{x,r}`	$H_x^{(r)}$

```
\newcommand{\HarmNum}[1]{%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer}=1}%
{%
H_{#1}
}%
% Else If
{ \ifthenelse{\value{COOL@listpointer}=2}%
{%
\liststore{#1}{COOL@list@temp@}%
H^{\inp{\COOL@list@temp@ii}}_{\COOL@list@temp@i}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{‘Harm Num’ can only accept a comma separated list of length 1 or 2}%
}}%
}
```

1.3.20 Beta Functions

\Beta

Beta Function	`\Beta{a,b}`	$B(a,b)$
Incomplete Beta Function	`\Beta{z,a,b}`	$B_z(a,b)$
Generalized Incomplete Beta Function	`\Beta{z_1,z_2,a,b}`	$B_{(z_1,z_2)}(a,b)$

```
\newcommand{\COOL@notation@BetaParen}{p}
\newcommand{\Beta}[1]{%
\liststore{#1}{COOL@Beta@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 2}%
```

```
{%
B\!\COOL@decide@paren{Beta}{\COOL@Beta@arg@i, \COOL@Beta@arg@ii}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 3}%
{%
B_{\COOL@Beta@arg@i}\!%
\COOL@decide@paren{Beta}{\COOL@Beta@arg@ii, \COOL@Beta@arg@iii}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 4}%
{%
B_{\inp{\COOL@Beta@arg@i,\COOL@Beta@arg@ii}}\!%
\COOL@decide@paren{Beta}{\COOL@Beta@arg@iii, \COOL@Beta@arg@iv}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`Beta' can only accept a comma separated list of length 2 to 4}%
}%
}}%
}
```

`\IncBeta` Incomplete Beta Function

`\IncBeta{z}{a}{b}`	$B_z(a,b)$
`\Beta{z,a,b}`	$B_z(a,b)$

```
\newcommand{\IncBeta}[3]{\Beta{#1,#2, #3}}
```

`\GenIncBeta` Generalized Incomplete Beta Function

`\GenIncBeta{x}{y}{a}{b}`	$B_{(x,y)}(a,b)$
`\Beta{x,y,a,b}`	$B_{(x,y)}(a,b)$

```
\newcommand{\GenIncBeta}[4]{\Beta{#1,#2,#3,#4}}
```

`\BetaRegularized` `\BetaReg` `\RegIncBeta` Regularized Incomplete Beta Function

`\BetaRegularized{z,a,b}`	$I_z(a,b)$
`\BetaReg{z,a,b}`	$I_z(a,b)$
`\RegIncBeta{z}{a}{b}`	$I_z(a,b)$

```
\newcommand{\COOL@notation@BetaRegularizedParen}{p}
\newcommand{\BetaRegularized}[1]{%
\liststore{#1}{COOL@BetaRegularized@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 3}%
{%
I_{\COOL@BetaRegularized@arg@i}\!%
\COOL@decide@paren{BetaRegularized}%
{\COOL@BetaRegularized@arg@ii, \COOL@BetaRegularized@arg@iii}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 4}%
```

```
{%
I_{\inp{\COOL@BetaRegularized@arg@i, \COOL@BetaRegularized@arg@ii}}\!%
\COOL@decide@paren{BetaRegularized}%
{\COOL@BetaRegularized@arg@iii, \COOL@BetaRegularized@arg@iv}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`BetaRegularized' can only accept%
 a comma separated list of length 3 or 4}%
}%
}%
}
\newcommand{\RegIncBeta}[3]{\BetaRegularized{#1,#2,#3}}
\newcommand{\BetaReg}[1]{\BetaRegularized{#1}}
```

\InverseBetaRegularized
\BetaRegInv
\RegIncBetaInv

Inverse of Regularized Incomplete Beta Function

`\InverseBetaRegularized{z,a,b}`	$I_z^{-1}(a,b)$
`\BetaRegInv{z,a,b}`	$I_z^{-1}(a,b)$
`\RegIncBetaInv{z}{a}{b}`	$I_z^{-1}(a,b)$

```
\newcommand{\COOL@notation@InverseBetaRegularizedParen}{p}
\newcommand{\InverseBetaRegularized}[1]{%
\liststore{#1}{COOL@InverseBetaRegularized@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 3}%
{%
I^{-1}_{\COOL@InverseBetaRegularized@arg@i}\!%
\COOL@decide@paren{InverseBetaRegularized}%
{\COOL@InverseBetaRegularized@arg@ii,%
 \COOL@InverseBetaRegularized@arg@iii}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 4}%
{%
I^{-1}_{\inp{ \COOL@InverseBetaRegularized@arg@i,%
\COOL@InverseBetaRegularized@arg@ii%
    }%
}\!%
\COOL@decide@paren{InverseBetaRegularized}%
{\COOL@InverseBetaRegularized@arg@iii,%
 \COOL@InverseBetaRegularized@arg@iv}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`InverseBetaRegularized' can only accept%
 a comma separated list of length 3 or 4}%
}%
}%
```

```
}
\newcommand{\RegIncBetaInv}[3]{\InverseBetaRegularized{#1,#2,#3}}
\newcommand{\BetaRegInv}[1]{\InverseBetaRegularized{#1}}
```

\GenRegIncBeta Generalized Regularized Incomplete Beta Func

`\GenRegIncBeta{x}{y}{a}{b}`	$B_{(x,y)}(a,b)$
`\Beta{x,y,a,b}`	$B_{(x,y)}(a,b)$

```
\newcommand{\GenRegIncBeta}[4]{\Beta{#1,#2,#3,#4}}
```

\GenRegIncBetaInv Inverse of Generalized Regularized Incomplete Beta Function

`\GenRegIncBetaInv{x}{y}{z}{b}`	$I^{-1}_{(x,y)}(z,b)$
`\InverseBetaRegularized{x,y,z,b}`	$I^{-1}_{(x,y)}(z,b)$

```
\newcommand{\GenRegIncBetaInv}[4]{\InverseBetaRegularized{#1,#2,#3,#4}}
```

1.3.21 Error Functions

\Erf

Error Function	`\Erf{x}`	$\mathrm{erf}(x)$
Generalized Error Function	`\Erf{x,y}`	$\mathrm{erf}(x,y)$

```
\newcommand{\COOL@notation@ErfParen}{p}
\DeclareMathOperator{\ErfSymb}{erf}
\newcommand{\Erf}[1]{%
\liststore{#1}{COOL@Erf@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 1}%
{%
\ErfSymb\!\COOL@decide@paren{Erf}{#1}
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 2}%
{%
\ErfSymb\!\COOL@decide@paren{Erf}{#1}
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{‘Erf’ can only accept a comma separated list of length 1 or 2}%
}%
}%
}
```

\ErfInv Inverse of Error Function

`\ErfInv{x}`	$\mathrm{erf}^{-1}(x)$
`\ErfInv{x,y}`	$\mathrm{erf}^{-1}(x,y)$

```
\newcommand{\COOL@notation@ErfInvParen}{p}
\newcommand{\ErfInv}[1]{%
\liststore{#1}{COOL@Erf@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 1}%
```

```
{%
\ErfSymb^{-1}\!\COOL@decide@paren{ErfInv}{#1}
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 2}%
{%
\ErfSymb^{-1}\!\COOL@decide@paren{ErfInv}{#1}
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`Erf' can only accept a comma separated list of length 1 or 2}%
}%
}%
}
```

\GenErf
\GenErfInv

Generalized Error Function and its inverse

`\GenErf{z_1}{z_2}`	$\mathrm{erf}(z_1, z_2)$
`\GenErfInv{z_1}{z_2}`	$\mathrm{erf}^{-1}(z_1, z_2)$

```
\newcommand{\GenErf}[2]{\Erf{#1,#2}}
\newcommand{\GenErfInv}[2]{\ErfInv{#1, #2}}
```

\Erfc

Complimentary Error Function and its inverse

`\Erfc{z}`	$\mathrm{erfc}(z)$
`\ErfcInv{z}`	$\mathrm{erfc}^{-1}(z)$

```
\newcommand{\COOL@notation@ErfcParen}{p}
\DeclareMathOperator{\ErfcSymb}{erfc}
\newcommand{\Erfc}[1]{\ErfcSymb\!\COOL@decide@paren{Erfc}{#1}}
\newcommand{\COOL@notation@ErfcInvParen}{p}
\newcommand{\ErfcInv}[1]%
{\ErfcSymb^{-1}\!\COOL@decide@paren{ErfcInv}{#1}}
```

\Erfi

Imaginary Error Function, `\Erfi{z}`, $\mathrm{erfi}(z)$

```
\newcommand{\COOL@notation@ErfiParen}{p}
\DeclareMathOperator{\ErfiSymb}{erfi}
\newcommand{\Erfi}[1]{\ErfiSymb\!\COOL@decide@paren{Erfi}{#1}}
```

1.3.22 Fresnel Integrals

\FresnelS

Fresnel Integral, `\FresnelS{z}`, $S(z)$

```
\newcommand{\COOL@notation@FresnelSParen}{p}
\newcommand{\FresnelS}[1]{S\!\COOL@decide@paren{FresnelS}{#1}}
```

\FresnelC

Fresnel Integral, `\FresnelC{z}`, $C(z)$

```
\newcommand{\COOL@notation@FresnelCParen}{p}
\newcommand{\FresnelC}[1]{C\!\COOL@decide@paren{FresnelC}{#1}}
```

1.3.23 Exponential Integrals

\ExpIntE Exponential Integral, \ExpIntE{\nu}{x}, $E_\nu(x)$

```
\newcommand{\COOL@notation@ExpIntEParen}{p}
\newcommand{\ExpIntE}[2]{E_{#1}\!\COOL@decide@paren{ExpIntE}{#2}}
```

\ExpIntEi Exponential Integral, \ExpIntEi{x}, $\mathrm{Ei}(x)$

```
\newcommand{\COOL@notation@ExpIntEiParen}{p}
\DeclareMathOperator{\ExpIntEiSymb}{Ei}
\newcommand{\ExpIntEi}[1]%
{\ExpIntEiSymb\!\COOL@decide@paren{ExpIntEi}{#1}}
```

\LogInt Logarithmic Integral, \LogInt{x}, $\mathrm{li}(x)$

```
\newcommand{\COOL@notation@LogIntParen}{p}
\DeclareMathOperator{\LogIntSymb}{li}
\newcommand{\LogInt}[1]{\LogIntSymb\!\COOL@decide@paren{LogInt}{#1}}
```

\SinInt Sine Integral, \SinInt{x}, $\mathrm{Si}(x)$

```
\newcommand{\COOL@notation@SinIntParen}{p}
\DeclareMathOperator{\SinIntSymb}{Si}
\newcommand{\SinInt}[1]{\SinIntSymb\!\COOL@decide@paren{SinInt}{#1}}
```

\CosInt Cosine Integral, \CosInt{x}, $\mathrm{Ci}(x)$

```
\newcommand{\COOL@notation@CosIntParen}{p}
\DeclareMathOperator{\CosIntSymb}{Ci}
\newcommand{\CosInt}[1]{\CosIntSymb\!\COOL@decide@paren{CosInt}{#1}}
```

\SinhInt Hyberbolic Sine Integral, \SinhInt{x}, $\mathrm{Shi}(x)$

```
\newcommand{\COOL@notation@SinhIntParen}{p}
\DeclareMathOperator{\SinhIntSymb}{Shi}
\newcommand{\SinhInt}[1]{\SinhIntSymb\!\COOL@decide@paren{SinhInt}{#1}}
```

\CoshInt Hyberbolic Cosine Integral, \CoshInt{x}, $\mathrm{Chi}(x)$

```
\newcommand{\COOL@notation@CoshIntParen}{p}
\DeclareMathOperator{\CoshIntSymb}{Chi}
\newcommand{\CoshInt}[1]{\CoshIntSymb\!\COOL@decide@paren{CoshInt}{#1}}
```

1.3.24 Hypergeometric Functions

\COOL@Hypergeometric@pq@ab@value This macro is a decision maker that decides what to return for the Hypergeometric function since its results vary based on the nature of the input. This macro is called as

\COOL@Hypergeometric@pq@ab@value {'p'|'q'} {⟨*p_input* | *q_input*⟩} {'a'|'b'} {⟨*a_input* | *b_input*⟩}

```
\newcommand{\COOL@Hypergeometric@pq@ab@value}[4]{%
\ifthenelse{\boolean{COOL@#1@isint} \AND \boolean{COOL@#3@islist}}%
{% #1 is an INT and #3 is a LIST
\ifthenelse{ #2 = 0 }%
```

```
{%
\PackageWarning{cool}{‘#3’-arg ignored}%
}%
% Else
{%
\ifthenelse{ #2 = 1 }%
{%
\PackageError{cool}{‘Hypergeometric’ ‘#1’-arg mismatch with ‘#3’-arg}{}%
}%
% Else
{%
#4%
}%
}%
}%
% Else
{}%
\ifthenelse{ \boolean{COOL@#1@isint} \AND
\NOT \boolean{COOL@#3@islist}  }%
{%
\ifthenelse{ #2 = 0 }%
{%
% return nothing
}%
% Else
{%
\ifthenelse{ #2 = 1 }%
{%
% return
#4%
}%
% Else
{%
\forLoop{1}{#2}{COOL@ct}
{%
\ifthenelse{ \value{COOL@ct} = 1 }{}{,}%
#4_{\arabic{COOL@ct}}%
}% end for loop
}%
}%
}%
% else
{}%
\ifthenelse{  \NOT \boolean{COOL@#1@isint} \AND
\boolean{COOL@#3@islist}  }%
{%
\PackageError{cool}{Invalid Argument}%
{‘Hypergeometric’: ‘#1’-arg is not int but ‘#3’-arg is list}
}%
% else
```

```
{}%
\ifthenelse{ \NOT \boolean{COOL@#1@isint} \AND
\NOT \boolean{COOL@#3@islist}  }%
{%
%return
#4_1,\ldots,#4_{#2}%
}%
% else
{}%
}%
```

\Hypergeometric Generalized Hypergeometric function. ${}_pF_q(a_1,\ldots,a_p;b_1,\ldots,b_q;x)$

```
\Hypergeometric{0}{0}{}{}{x}
```
${}_0F_0(;;x)$

```
\Hypergeometric{0}{1}{}{b}{x}
```
${}_0F_1(;b;x)$

```
\Hypergeometric{1}{1}{a}{b}{x}
```
${}_1F_1(a;b;x)$

```
\Hypergeometric{1}{1}{1}{1}{x}
```
${}_1F_1(1;1;x)$

```
\Hypergeometric{3}{5}{a}{b}{x}
```
${}_3F_5(a_1,a_2,a_3;b_1,b_2,b_3,b_4,b_5;x)$

```
\Hypergeometric{3}{5}{1,2,3}{1,2,3,4,5}{x}
```
${}_3F_5(1,2,3;1,2,3,4,5;x)$

```
\Hypergeometric{p}{5}{a}{b}{x}
```
${}_pF_5(a_1,\ldots,a_p;b_1,b_2,b_3,b_4,b_5;x)$

```
\Hypergeometric{p}{3}{a}{1,2,3}{x}
```
${}_pF_3(a_1,\ldots,a_p;1,2,3;x)$

```
\Hypergeometric{p}{q}{a}{b}{x}
```
${}_pF_q(a_1,\ldots,a_p;b_1,\ldots,b_q;x)$

```
\newcommand{\COOL@notation@HypergeometricParen}{p}
\newcommand{\COOL@notation@HypergeometricSymb}{F}
\newcommand{\Hypergeometric}[6][F]{%
\provideboolean{COOL@p@isint}%
\provideboolean{COOL@q@isint}%
\provideboolean{COOL@a@islist}%
\provideboolean{COOL@b@islist}%
\isint{#2}%
\ifthenelse{\boolean{COOL@isint}}%
{\setboolean{COOL@p@isint}{true}}%
% Else
{\setboolean{COOL@p@isint}{false}}%
\isint{#3}%
\ifthenelse{\boolean{COOL@isint}}%
{\setboolean{COOL@q@isint}{true}}%
% Else
{\setboolean{COOL@q@isint}{false}}%
\listval{#4}{0}%
\ifthenelse{\value{COOL@listpointer}>1}%
{\setboolean{COOL@a@islist}{true}}%
% Else
{\setboolean{COOL@a@islist}{false}}%
```

ensure that the submitted list is the same length as p

```
\ifthenelse{ \boolean{COOL@p@isint} \AND
\boolean{COOL@a@islist} \AND
\NOT \( #2 = \value{COOL@listpointer} \) }%
{%
\PackageError{cool}{‘Hypergeometric’ ‘p’-arg mismatch with ‘a’-arg}{}%
}%
% else
{}%
\listval{#5}{0}%
\ifthenelse{\value{COOL@listpointer}>1}%
{\setboolean{COOL@b@islist}{true}}%
% Else
{\setboolean{COOL@b@islist}{false}}%
```

ensure that the submitted ‘b’ list is the same length as q

```
\ifthenelse{ \boolean{COOL@q@isint} \AND
\boolean{COOL@b@islist} \AND
\NOT \( #3 = \value{COOL@listpointer} \) }%
{%
\PackageError{cool}{‘Hypergeometric’ ‘q’-arg mismatch with ‘b’-arg}%
{‘b’ list is not the same length as ‘q’}%
}%
% else
{}%
% troubleshoot
\ifthenelse{ \boolean{COOL@a@islist} \AND \NOT \boolean{COOL@p@isint} }%
{%
\PackageError{cool}{‘Hypergeometric’ ‘a’-arg mismatch with ‘p’-arg}%
{happens if ‘a’-arg is a list and ‘p’-arg isn’t an integer}%
}%
% else
{}%
\ifthenelse{ \boolean{COOL@b@islist} \AND \NOT \boolean{COOL@q@isint} }%
{%
\PackageError{cool}{‘Hypergeometric’ ‘b’-arg mismatch with ‘q’-arg}%
{happens if ‘b’-arg is a list and ‘q’-arg isn’t an integer}%
}%
% else
{}%
```

First print the ${}_pF_q$

```
{}_{#2}{\COOL@notation@HypergeometricSymb}_{#3}\!%
\COOL@decide@paren{Hypergeometric}%
{%
\COOL@Hypergeometric@pq@ab@value{p}{#2}{a}{#4};%
\COOL@Hypergeometric@pq@ab@value{q}{#3}{b}{#5};%
#6%
}%
}
```

\RegHypergeometric Regularized hypergeometric function ${}_p\tilde{F}_q(a_1,\ldots,a_p;b_1,\ldots,b_q;x)$

```
\newcommand{\COOL@notation@RegHypergeometricParen}{p}
\newcommand{\COOL@notation@RegHypergeometricSymb}{\tilde{F}}
\newcommand{\RegHypergeometric}[6][\tilde{F}]{%
\provideboolean{COOL@p@isint}%
\provideboolean{COOL@q@isint}%
\provideboolean{COOL@a@islist}%
\provideboolean{COOL@b@islist}%
\isint{#2}%
\ifthenelse{\boolean{COOL@isint}}%
{\setboolean{COOL@p@isint}{true}}%
% Else
{\setboolean{COOL@p@isint}{false}}%
\isint{#3}%
\ifthenelse{\boolean{COOL@isint}}%
{\setboolean{COOL@q@isint}{true}}%
% Else
{\setboolean{COOL@q@isint}{false}}%
\listval{#4}{0}%
\ifthenelse{\value{COOL@listpointer}>1}%
{\setboolean{COOL@a@islist}{true}}%
% Else
{\setboolean{COOL@a@islist}{false}}%
```

ensure that the submitted list is the same length as p

```
\ifthenelse{ \boolean{COOL@p@isint} \AND
\boolean{COOL@a@islist} \AND
\NOT \( #2 = \value{COOL@listpointer} \) }%
{%
\PackageError{cool}%
{`RegHypergeometric' `p'-arg mismatch with `a'-arg}{}%
}%
% else
{}%
\listval{#5}{0}%
\ifthenelse{\value{COOL@listpointer}>1}%
{\setboolean{COOL@b@islist}{true}}%
% Else
{\setboolean{COOL@b@islist}{false}}%
```

ensure that the submitted `b' list is the same length as q

```
\ifthenelse{ \boolean{COOL@q@isint} \AND
\boolean{COOL@b@islist} \AND
\NOT \( #3 = \value{COOL@listpointer} \) }%
{%
\PackageError{cool}%
{`RegHypergeometric' `q'-arg mismatch with `b'-arg}%
{`b' list is not the same length as `q'}%
}%
% else
```

```
{}%
% troubleshoot
\ifthenelse{ \boolean{COOL@a@islist} \AND \NOT \boolean{COOL@p@isint} }%
{%
\PackageError{cool}%
{‘RegHypergeometric’ ‘a’-arg mismatch with ‘p’-arg}%
{happens if ‘a’-arg is a list and ‘p’-arg isn’t an integer}%
}%
% else
{}%
\ifthenelse{ \boolean{COOL@b@islist} \AND \NOT \boolean{COOL@q@isint} }%
{%
\PackageError{cool}%
{‘RegHypergeometric’ ‘b’-arg mismatch with ‘q’-arg}%
{happens if ‘b’-arg is a list and ‘q’-arg isn’t an integer}%
}%
% else
{}%
```

First print the ${}_pF_q$

```
{}_{#2}{\COOL@notation@RegHypergeometricSymb}_{#3}\!%
\COOL@decide@paren{RegHypergeometric}%
{%
\COOL@Hypergeometric@pq@ab@value{p}{#2}{a}{#4};%
\COOL@Hypergeometric@pq@ab@value{q}{#3}{b}{#5};%
#6%
}%
}
```

`\AppellFOne` Appell Hypergeometric Function

`\AppellFOne{a}{b_1,b_2}{c}{z_1,z_2}` $F_1(a;b_1,b_2;c;z_1,z_2)$

```
\newcommand{\COOL@notation@AppellFOneParen}{p}
\newcommand{\AppellFOne}[4]%
{F_{1}\!\COOL@decide@paren{AppellFOne}{#1; #2; #3; #4}}
```

`\HypergeometricU` Tricomi confluent hypergeometric function

`\HypergeometricU{a}{b}{z}` $U(a,b,z)$

```
\newcommand{\COOL@notation@HypergeometricUSymb}{U}
\newcommand{\HypergeometricU}[3]%
{\COOL@notation@HypergeometricUSymb\!\inp{#1, #2, #3}}
```

`\COOL@MeijerG@anp@value` This macro is a decision maker for the `\MeijerG` macro. Despite the name it is used for both p and q. It is called as

`\COOL@MeijerG@anp@value {`$\langle a \mid b\rangle$`} {`$\langle n \mid m\rangle$`} {`$\langle p \mid q\rangle$`}`

```
\newcommand{\COOL@MeijerG@anp@value}[3]{%
\isint{#3}%
\ifthenelse{\boolean{COOL@isint}}%
{%
\isint{#2}%
```

```
\ifthenelse{\boolean{COOL@isint}}%
{%
\forLoop{1}{#3}{COOL@ct}%
{%
\ifthenelse{\value{COOL@ct}=1}{}{,}%
#1_{\arabic{COOL@ct}}%
}%
}%
% else
{%
#1_1,\ldots,#1_{#2},#1_{#2+1},\dots,#1_{#3}%
}%
}%
% else
{%
\isint{#2}%
\ifthenelse{\boolean{COOL@isint}}%
{%
\forLoop{1}{#2}{COOL@ct}%
{%
\ifthenelse{\value{COOL@ct}=1}{}{,}%
#1_{\arabic{COOL@ct}}%
}%
\setcounter{COOL@ct}{#2}%
\addtocounter{COOL@ct}{1}%
,#1_{\arabic{COOL@ct}}, \ldots, #1_{#3}%
}%
% else
{%
#1_1,\ldots,#1_{#2},#1_{#2+1},\dots,#1_{#3}%
}%
}%
}
```

`\MeijerG` `\MeijerG{`$a_1,\ldots,a_n$`}{`$a_{n+1},\ldots,a_p$`}{`$b_1,\ldots,b_m$`}{`$b_{m+1},\ldots,b_q$`}{`$\langle x\rangle$`}`
`\MeijerG[`$\langle a$ *list symbol*$\rangle$`,`$\langle b$ *list symbol*$\rangle$`]{`$\langle n\rangle$`}{`$\langle p\rangle$`}{`$\langle m\rangle$`}{`$\langle q\rangle$`}{`$\langle x\rangle$`}`
`\MeijerG[`$\langle a$ *list symbol*$\rangle$`]{`$\langle n\rangle$`}{`$\langle p\rangle$`}{`$b_1,\ldots,b_m$`}{`$b_{m+1},\ldots,b_q$`}{`$\langle x\rangle$`}`
`\MeijerG[,`$\langle b$ *list symbol*$\rangle$`]{`$a_1,\ldots,a_n$`}{`$a_{n+1},\ldots,a_p$`}{`$\langle m\rangle$`}{`$\langle q\rangle$`}{`$\langle x\rangle$`}`

Meijer G-Function

`\MeijerG[a,b]{n}{p}{m}{q}{z}` $G_{p,q}^{m,n}\left(z \left| \begin{smallmatrix} a_1,\ldots,a_n,a_{n+1},\ldots,a_p \\ b_1,\ldots,b_m,b_{m+1},\ldots,b_q \end{smallmatrix} \right.\right)$

Meijer G-Function

`\MeijerG{1,2}{3}{a,b}{c,d}{z}` $G_{3,4}^{2,2}\left(z \left| \begin{smallmatrix} 1,2,3 \\ a,b,c,d \end{smallmatrix} \right.\right)$

Generalized Meijer G-Function

`\MeijerG[a,b]{n}{p}{m}{q}{z,r}` $G_{p,q}^{m,n}\left(z,r \left| \begin{smallmatrix} a_1,\ldots,a_n,a_{n+1},\ldots,a_p \\ b_1,\ldots,b_m,b_{m+1},\ldots,b_q \end{smallmatrix} \right.\right)$

Generalized Meijer G-Function

`\MeijerG{1,2}{3}{a,b}{c,d}{z,r}` $G_{3,4}^{2,2}\left(z,r \left| \begin{smallmatrix} 1,2,3 \\ a,b,c,d \end{smallmatrix} \right.\right)$

```
\newcommand{\COOL@notation@MeijerGSymb}{G}
\newcommand{\MeijerG}[6][@,@]{%
\listval{#1}{0}
\ifthenelse{\value{COOL@listpointer}>2 \OR \value{COOL@listpointer}<1}%
{%
\PackageError{cool}{`MeijerG' Invalid Optional Argument}%
{Must be a comma separated list of length 1 or 2}%
}%
% else
{%
}%
\COOL@notation@MeijerGSymb%
\ifthenelse{\equal{#1}{@,@}}%
{%
\listval{#2}{0}% n
\setcounter{COOL@ct}{\value{COOL@listpointer}}%
\listval{#4}{0}% m
\setcounter{COOL@ct@}{\value{COOL@listpointer}}%
^{\arabic{COOL@ct@},\arabic{COOL@ct}}%
\listval{#3}{0}% p - n
\addtocounter{COOL@ct}{\value{COOL@listpointer}}%
\listval{#5}{0}% q - m
\addtocounter{COOL@ct@}{\value{COOL@listpointer}}%
_{\arabic{COOL@ct},\arabic{COOL@ct@}}%
\!\left(%
#6%
\left|%
{ {#2,#3} \@@atop {#4,#5} }%
\right)\right.%
}%
% else
{%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer}=2}%
{%
\providebooleanCOOL@MeijerG@opt@one@blank}%
\def\COOL@MeijerG@sniffer##1,##2\COOL@MeijerG@sniffer@end{%
\ifthenelse{\equal{##1}{}}%
{%
```

```
\setboolean{COOL@MeijerG@opt@one@blank}{true}%
}%
% else
{%
\setboolean{COOL@MeijerG@opt@one@blank}{false}%
}%
}%
\expandafter\COOL@MeijerG@sniffer#1\COOL@MeijerG@sniffer@end\relax%
\ifthenelse{\boolean{COOL@MeijerG@opt@one@blank}}%
{%
```

this is `\MeijerG[,b]{a_1,...,a_n}{a_{n++},...,a_p}{m}{q}{x}`

```
\listval{#2}{0}% n
\setcounter{COOL@ct}{\value{COOL@listpointer}}%
^{#4,\arabic{COOL@ct}}%
\listval{#3}{0}% p
\addtocounter{COOL@ct}{\value{COOL@listpointer}}%
_{\arabic{COOL@ct},#5}%
\!\left(%
#6%
\left|%
{%
{#2,#3} \@@atop {\COOL@MeijerG@anp@value{\listval{#1}{2}}{#4}{#5}}
}%
\right)\right.%
}%
% else
{%
^{#4,#2}_{#3,#5}%
\!\left(%
#6%
\left|%
{%
{\COOL@MeijerG@anp@value{\listval{#1}{1}}{#2}{#3}}%
\@@atop%
{\COOL@MeijerG@anp@value{\listval{#1}{2}}{#4}{#5}}
}%
\right)\right.%
}%
}%
% else
{%
```

this is `\MeijerG[a]{n}{p}{b_1,...,b_m}{b_{m++},...,a_p}{x}`

```
\listval{#4}{0}% m
\setcounter{COOL@ct}{\value{COOL@listpointer}}%
^{\arabic{COOL@ct}, #2}%
\listval{#5}{0}% q
\addtocounter{COOL@ct}{\value{COOL@listpointer}}%
_{#3, \arabic{COOL@ct}}%
\!\left(%
```

```
#6%
\left|%
{%
{\COOL@MeijerG@anp@value{#1}{#2}{#3}} \@@atop {#4,#5}
}%
\right)\right.%
}%
}%
}%
```

1.3.25 Angular Momentum Functions

`\ClebschGordon` Clebsch-Gordon Coefficients

`\ClebschGordon{j_1,m_1}{j_2,m_2}{j,m}` $\langle j_1, j_2; m_1, m_2 | j_1, j_2; j, m \rangle$

http://functions.wolfram.com/HypergeometricFunctions/ClebschGordan/

```
\newcommand{\ClebschGordon}[3]{%
\listval{#1}{0}%
\ifthenelse{\NOT \value{COOL@listpointer}=2}%
{%
\PackageError{cool}{`ClebschGordon' Invalid Argument}%
{Must have a comma separated list of length two}%
}%
% else
{}%
\listval{#2}{0}%
\ifthenelse{\NOT \value{COOL@listpointer}=2}%
{%
\PackageError{cool}{`ClebschGordon' Invalid Argument}%
{Must have a comma separated list of length two}%
}%
% else
{}%
\listval{#3}{0}%
\ifthenelse{\NOT \value{COOL@listpointer}=2}%
{%
\PackageError{cool}{`ClebschGordon' Invalid Argument}%
{Must have a comma separated list of length two}%
}%
% else
{}%
\left<%
\listval{#1}{1},\listval{#2}{1};%
\listval{#1}{2},\listval{#2}{2}%
\left|%
\listval{#1}{1},\listval{#2}{1};%
\listval{#3}{1},\listval{#3}{2}%
\right>\right.%
}
```

`\ThreeJSymbol` Wigner 3-j Symbol

\ThreeJSymbol{j_1,m_1}{j_2,m_2}{j_3,m_3} $\begin{pmatrix} j_1 & j_2 & j_3 \\ m_1 & m_2 & m_3 \end{pmatrix}$

http://functions.wolfram.com/HypergeometricFunctions/ThreeJSymbol/

```
\newcommand{\ThreeJSymbol}[3]{%
\listval{#1}{0}%
\ifthenelse{\NOT \value{COOL@listpointer}=2}%
{%
\PackageError{cool}{`ThreeJSymbol' Invalid Argument}%
{Must have comma separated list of length 2}%
}%
% else
{}%
\listval{#2}{0}%
\ifthenelse{\NOT \value{COOL@listpointer}=2}%
{%
\PackageError{cool}{`ThreeJSymbol' Invalid Argument}%
{Must have comma separated list of length 2}%
}%
% else
{}%
\listval{#3}{0}%
\ifthenelse{\NOT \value{COOL@listpointer}=2}%
{%
\PackageError{cool}{`ThreeJSymbol' Invalid Argument}%
{Must have comma separated list of length 2}%
}%
% else
{}%
\mathchoice{%
% displaystyle
\inp{\!%
\begin{array}{ccc}%
\listval{#1}{1} & \listval{#2}{1} & \listval{#3}{1} \\%
\listval{#1}{2} & \listval{#2}{2} & \listval{#3}{2}
\end{array}%
     \!}%
   }%
   {%
% inline
\inp{\!%
{\listval{#1}{1} \@@atop \listval{#1}{2}}%
{\listval{#2}{1} \@@atop \listval{#2}{2}}%
{\listval{#3}{1} \@@atop \listval{#3}{2}}%
     \!}%
   }%
   {%
% subscript
\inp{\!%
{\listval{#1}{1} \@@atop \listval{#1}{2}}%
{\listval{#2}{1} \@@atop \listval{#2}{2}}%
```

```
{\listval{#3}{1} \@@atop \listval{#3}{2}}%
      \!}%
   }%
   {%
% subsubscript
\inp{\!%
{\listval{#1}{1} \@@atop \listval{#1}{2}}%
{\listval{#2}{1} \@@atop \listval{#2}{2}}%
{\listval{#3}{1} \@@atop \listval{#3}{2}}%
      \!}%
   }%
}
```

\SixJSymbol Racah 6-j Symbol

\SixJSymbol{j_1,j_2,j_3}{j_4,j_5,j_6} $\begin{Bmatrix} j_1 & j_2 & j_3 \\ j_4 & j_5 & j_6 \end{Bmatrix}$

http://functions.wolfram.com/HypergeometricFunctions/SixJSymbol/

```
\newcommand{\SixJSymbol}[2]{%
\listval{#1}{0}%
\ifthenelse{\NOT \value{COOL@listpointer}=3}%
{%
\PackageError{cool}{`SixJSymbol' Invalid Argument}%
{Must have a comma separated list of length 3}%
}%
%else
{}%
\listval{#2}{0}%
\ifthenelse{\NOT \value{COOL@listpointer}=3}%
{%
\PackageError{cool}{`SixJSymbol' Invalid Argument}%
{Must have a comma separated list of length 3}%
}%
%else
{}%
\mathchoice{%
% displaystyle
\inbr{\!%
\begin{array}{ccc}%
\listval{#1}{1} & \listval{#1}{2} & \listval{#1}{3} \\%
\listval{#2}{1} & \listval{#2}{2} & \listval{#2}{3}%
\end{array}%
      \!}%
  }%
  {%
% inline
\inbr{\!%
{\listval{#1}{1} \@@atop \listval{#2}{1}}%
{\listval{#1}{2} \@@atop \listval{#2}{2}}%
{\listval{#1}{3} \@@atop \listval{#2}{3}}%
      \!}%
```

```
  }%
  {%
% superscript
\inbr{\!%
{\listval{#1}{1} \@@atop \listval{#2}{1}}%
{\listval{#1}{2} \@@atop \listval{#2}{2}}%
{\listval{#1}{3} \@@atop \listval{#2}{3}}%
    \!}%
  }%
  {%
% supersuperscript
\inbr{\!%
{\listval{#1}{1} \@@atop \listval{#2}{1}}%
{\listval{#1}{2} \@@atop \listval{#2}{2}}%
{\listval{#1}{3} \@@atop \listval{#2}{3}}%
    \!}%
  }%
}
```

1.3.26 Complete Elliptic Integrals

`\EllipticK` Complete Elliptic Integral of the First Kind

`\EllipticK{x}` $K(x)$

```
\newcommand{\COOL@notation@EllipticKParen}{p}
\newcommand{\COOL@notation@EllipticKSymb}{K}
\newcommand{\EllipticK}[1]%
{\COOL@notation@EllipticKSymb\!\COOL@decide@paren{EllipticK}{#1}}%
```

`\EllipticE` Complete Elliptic Integral of the Second Kind

`\EllipticE{x}` $E(x)$

```
\newcommand{\COOL@notation@EllipticEParen}{p}
\newcommand{\COOL@notation@EllipticESymb}{E}
\newcommand{\EllipticE}[1]{%
\liststore{#1}{COOL@EllipticE@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 1}%
{%
\COOL@notation@EllipticESymb\!\COOL@decide@paren{EllipticE}{#1}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 2}%
{%
\COOL@notation@EllipticESymb\!%
\COOL@decide@paren{EllipticE}%
{\COOL@EllipticE@arg@i \left| \, \COOL@EllipticE@arg@ii \!\!\right.}%
}%
% Else
{%
\PackageError{Invalid Argument}%
```

```
{‘EllipticE’ can only accept a comma separated list of length 1 or 2}%
}%
}%
}
```

\EllipticPi Complete Elliptic Integral of the Third Kind

\EllipticPi{n,m} $\Pi(n \,|\, m)$

```
\newcommand{\COOL@notation@EllipticPiParen}{p}
\newcommand{\COOL@notation@EllipticPiSymb}{\Pi}
\newcommand{\EllipticPi}[1]{%
\liststore{#1}{COOL@EllipticPi@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 2}%
{%
\COOL@notation@EllipticPiSymb%
\!\COOL@decide@paren{EllipticPi}%
{\COOL@EllipticPi@arg@i \left| \, \COOL@EllipticPi@arg@ii \!\!\right.}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 3}%
{%
\COOL@notation@EllipticPiSymb%
\!\COOL@decide@paren{EllipticPi}%
{ \COOL@EllipticPi@arg@i; \,%
\COOL@EllipticPi@arg@ii \left| \,%
\COOL@EllipticPi@arg@iii \!\!\right.%
}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{‘EllipticPi’ can only accept a comma separated list of length 2 or 3}%
}%
}%
}
```

1.3.27 Incomplete Elliptic Integrals

\EllipticF \IncEllipticF Incomplete Elliptic Integral of the First Kind

\EllipticF{z,m} $F(z \,|\, m)$

\IncEllipticF{z}{m} $F(z \,|\, m)$

```
\newcommand{\COOL@notation@EllipticFParen}{p}
\newcommand{\COOL@notation@EllipticFSymb}{F}
\newcommand{\EllipticF}[1]{%
\liststore{#1}{COOL@EllipticF@arg@}%
\listval{#1}{0}%
\ifthenelse{ \value{COOL@listpointer} = 2 }%
{%
\COOL@notation@EllipticFSymb%
```

```
\!\COOL@decide@paren{EllipticF}%
{\COOL@EllipticF@arg@i \left| \, \COOL@EllipticF@arg@ii \!\!\right.}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`EllipticF' can only accept a comma separated list of length 2}%
}%
}
\newcommand{\IncEllipticF}[2]{\EllipticF{#1,#2}}
```

\IncEllipticE Incomplete Elliptic Integral of the Second Kind

`\IncEllipticE{z}{m}`	$E(z \,|\, m)$
`\EllipticE{z,m}`	$E(z \,|\, m)$

```
\newcommand{\IncEllipticE}[2]{\EllipticE{#1,#2}}
```

\IncEllipticPi \EllipticPi Incomplete Elliptic Integral of the Third Kind

`\IncEllipticPi{n}{z}{m}`	$\Pi(n;\, z \,|\, m)$
`\EllipticPi{n,z,m}`	$\Pi(n;\, z \,|\, m)$

```
\newcommand{\IncEllipticPi}[3]{\EllipticPi{#1,#2,#3}}
```

\JacobiZeta Jacobi Zeta Function

`\JacobiZeta{z}{m}`	$Z(z \,|\, m)$

```
\newcommand{\COOL@notation@JacobiZetaParen}{p}
\newcommand{\COOL@notation@JacobiZetaSymb}{Z}
\newcommand{\JacobiZeta}[2]{%
\COOL@notation@JacobiZetaSymb
\!\COOL@decide@paren{JacobiZeta}{#1 \left| \, #2 \right.\!\!}%
}
```

1.3.28 Jacobi Theta Functions

\EllipticTheta \JacobiTheta Jacobi Theta Functions

`\JacobiTheta{1}{z}{q}`	$\vartheta_1(z,q)$
`\JacobiTheta{2}{z}{q}`	$\vartheta_2(z,q)$
`\JacobiTheta{3}{z}{q}`	$\vartheta_3(z,q)$
`\JacobiTheta{4}{z}{q}`	$\vartheta_4(z,q)$

```
\newcommand{\COOL@notation@EllipticThetaParen}{p}
\newcommand{\EllipticTheta}[3]%
{\vartheta_{#1}\!\COOL@decide@paren{EllipticTheta}{#2, #3}}
\newcommand{\JacobiTheta}[3]{\EllipticTheta{#1}{#2}{#3}}
```

1.3.29 Neville Theta Functions

\NevilleThetaC Neville Theta Function, `\NevilleThetaC{z}{m}`, $\vartheta_c(z \,|\, m)$

```
\newcommand{\COOL@notation@NevilleThetaCParen}{p}
\newcommand{\NevilleThetaC}[2]{%
\vartheta_{c}\!\COOL@decide@paren{NevilleThetaC}%
```

```
{#1 \left| \, #2 \right.\!\!}%
}
```

\NevilleThetaD Neville Theta Function, \NevilleThetaD{z}{m}, $\vartheta_d(z \,|\, m)$

```
\newcommand{\COOL@notation@NevilleThetaDParen}{p}
\newcommand{\NevilleThetaD}[2]{%
\vartheta_{d}\!\COOL@decide@paren{NevilleThetaD}%
{#1 \left| \, #2 \right.\!\!}%
}
```

\NevilleThetaN Neville Theta Function, \NevilleThetaN{z}{m}, $\vartheta_n(z \,|\, m)$

```
\newcommand{\COOL@notation@NevilleThetaNParen}{p}
\newcommand{\NevilleThetaN}[2]{%
\vartheta_{n}\!\COOL@decide@paren{NevilleThetaN}%
{#1 \left| \, #2 \right.\!\!}%
}
```

\NevilleThetaS Neville Theta Function, \NevilleThetaS{z}{m}, $\vartheta_s(z \,|\, m)$

```
\newcommand{\COOL@notation@NevilleThetaSParen}{p}
\newcommand{\NevilleThetaS}[2]{%
\vartheta_{s}\!\COOL@decide@paren{NevilleThetaS}%
{#1 \left| \, #2 \right.\!\!}%
}
```

1.3.30 Weierstrass Functions

\WeierstrassP \WeiP Weierstrass Elliptic Function

\WeierstrassP{z}{g_2,g_3}	$\wp(z; g_2, g_3)$	
\WeiP{z}{g_2,g_3}	$\wp(z; g_2, g_3)$	

```
\newcommand{\COOL@notation@WeierstrassPParen}{p}
\newcommand{\WeierstrassP}[2]{%
\liststore{#2}{COOL@WeiP@arg@g@}%
\listval{#2}{0}%
\ifthenelse{\NOT \value{COOL@listpointer} = 2}%
{%
\PackageError{cool}{Invalid Argument}%
{`WeierstrassP' second argument must be%
 a comma separated list of length 2}%
}
% Else
{%
\wp\!\COOL@decide@paren{WeierstrassP}{#1; #2}
}%
}
\newcommand{\WeiP}[2]{\WeierstrassP{#1}{#2}}
```

\WeierstrassPInv \WeiPInv Inverse of Weierstrass Elliptic Function

Inverse	\WeiPInv{z}{g_2,g_3}	$\wp^{-1}(z; g_2, g_3)$	
Generalized Inverse	\WeiPInv{z_1,z_2}{g_2,g_3}	$\wp^{-1}(z_1, z_2; g_2, g_3)$	

```
\newcommand{\COOL@notation@WeierstrassPInvParen}{p}
\newcommand{\WeierstrassPInv}[2]{%
\liststore{#1}{COOL@WeiPinv@arg@z@}%
\liststore{#1}{COOL@WeiPinv@arg@g@}%
\listval{#2}{0}%
\ifthenelse{\NOT \value{COOL@listpointer} = 2}%
{%
\PackageError{cool}{Invalid Argument}%
{`WeierstrassPInv' second argument must be%
 a comma separated list of length 2}%
}
% Else
{
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 1}%
{%
\wp^{-1}\!\COOL@decide@paren{WeierstrassPInv}{#1; #2}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 2}%
{%
\wp^{-1}\!\COOL@decide@paren{WeierstrassPInv}{#1; #2}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{`WeierstrassPInv' first argument must be%
 a comma separate list of length 1 or 2}%
}}%
}%
}
\newcommand{\WeiPInv}[2]{\WeierstrassPInv{#1}{#2}}
```

`\WeierstrassPGenInv` Generalized Inverse of Weierstrass Elliptic Function

`\WierstrassPGenInv{z_1}{z_2}{g_1}{g_2}`

```
\newcommand{\WeierstrassPGenInv}[4]{\WeierstrassPInv{#1,#2}{#3,#4}}
```

`\WeierstrassSigma` `\WeiSigma` Wierstrass Sigma Function

Sigma	`\WeierstrassSigma{z}{g_2,g_3}`	$\sigma(z; g_2, g_3)$
	`\WeiSigma{z}{g_2,g_3}`	$\sigma(z; g_2, g_3)$
Associated Sigma	`\WeierstrassSigma{n,z}{g_2,g_3}`	$\sigma_n(z; g_2, g_3)$
	`\WeiSigma{n,z}{g_2,g_3}`	$\sigma_n(z; g_2, g_3)$

```
\newcommand{\WeierstrassSigma}[2]{%
\liststore{#1}{COOL@WeiSigma@arg@z@}%
\liststore{#2}{COOL@WeiSigma@arg@g@}%
\listval{#2}{0}%
\ifthenelse{\NOT \value{COOL@listpointer} = 2}
{%
\PackageError{cool}{Invalid Argument}%
```

```
{‘WeierstrassSigma’ second argument must be%
 a comma separated list of length 2}%
}%
% Else
{%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 1}%
{%
\sigma\!\inp{#1; #2}%
}%
% ElseIf
{ \ifthenelse{\value{COOL@listpointer} = 2}%
{%
\sigma_{\COOL@WeiSigma@arg@z@i}\!\inp{\COOL@WeiSigma@arg@z@ii; #2}%
}%
% Else
{%
\PackageError{cool}{Invalid Argument}%
{‘WeierstrassSigma’ first argument must be%
 a comma separated list of length 1 or 2}%
}}%
}%
}
\newcommand{\WeiSigma}[2]{\WeierstrassSigma{#1}{#2}}
```

`\AssocWeierstrassSigma` Associated Weierstrass Sigma Function

`\AssocWeierstrassSigma{n}{z}{g_2}{g_3}`	$\sigma_n(z; g_2, g_3)$
`\WeiSigma{n,z}{g_2,g_3}`	$\sigma_n(z; g_2, g_3)$

```
\newcommand{\AssocWeierstrassSigma}[4]{\WeierstrassSigma{#1,#2}{#3,#4}}
```

`\WeierstrassZeta` `\WeiZeta` Weierstrass Zeta Function

`\WeierstrassZeta{z}{g_2,g_3}`	$\zeta(z; g_2, g_3)$
`\WeiZeta{z}{g_2,g_3}`	$\zeta(z; g_2, g_3)$

```
\newcommand{\COOL@notation@WeierstrassZetaParen}{p}%
\newcommand{\WeierstrassZeta}[2]{%
\listval{#2}{0}%
\ifthenelse{\NOT \value{COOL@listpointer} = 2}%
{%
\PackageError{cool}{Invalid Argument}%
{‘WeierstrassZeta’ second argument must be%
 a comma separated list of length 2}%
}%
% Else
{%
\zeta\!\COOL@decide@paren{WeierstrassZeta}{#1; #2}%
}%
}
\newcommand{\WeiZeta}[2]{\WeierstrassZeta{#1}{#2}}
```

\WeierstrassHalfPeriods Weierstrass half-periods
\WeiHalfPeriods

```
\WeierstrassHalfPeriods{g_2,g_3}   {ω1(g2,g3),ω3(g2,g3)}
\WeiHalfPeriods{g_2,g_3}           {ω1(g2,g3),ω3(g2,g3)}
```

```
\newcommand{\WeierstrassHalfPeriods}[1]{%
\listval{#1}{0}%
\ifthenelse{\NOT \value{COOL@listpointer} = 2}%
{%
\PackageError{cool}{Invalid Argument}%
{`WeierstrassHalfPeriods' can only accept%
 a comma separated list of length 2}%
}%
% Else
{%
\{ \omega_1\!\inp{#1}, \omega_3\!\inp{#1} \}%
}%
}
\newcommand{\WeiHalfPeriods}[1]{\WeierstrassHalfPeriods{#1}}
```

\WeierstrassInvariants Weierstrass Invariants

```
\WeierstrassInvariants{\omega_1,\omega_3}   {g2(ω1,ω3),g3(ω1,ω3)}
\WeiInvars{\omega_1,\omega_3}               {g2(ω1,ω3),g3(ω1,ω3)}
```

```
\newcommand{\WeierstrassInvariants}[1]{%
\listval{#1}{0}%
\ifthenelse{\NOT \value{COOL@listpointer} = 2}%
{%
\PackageError{cool}{Invalid Argument}%
{`WeierstrassInvariants' can only accept%
 a comma separated list of length 2}%
}%
% Else
{%
\{ g_2\!\inp{#1}, g_3\!\inp{#1} \}%
}%
}
\newcommand{\WeiInvars}[1]{\WeierstrassInvariants{#1}}
```

\COOL@hideOnSF Used to hide inputs or other when style is `sf`

`sf` short form
`ff` full form

```
\newcommand{\COOL@hideOnSF}[2]
{%
\ifthenelse{ \equal{\csname COOL@notation@#1\endcsname}{sf} }%
{}%
% Else
{#2}%
}
```

\WeierstrassPHalfPeriodValues Weierstrass elliptic function values at half-periods
\WeiPHalfPeriodVal

\Style{WeierstrassPHalfPeriodValuesDisplay=sf} (Default)

\WeierstrassPHalfPeriodValues{g_2,g_3}

\WeiPHalfPeriodVal{g_2,g_3}

$$\{e_1, e_2, e_3\}$$

\Style{WeierstrassPHalfPeriodValuesDisplay=ff}

\WeierstrassPHalfPeriodValues{g_2,g_3}

\WeiPHalfPeriodVal{g_2,g_3}

$$\{e_1(g_2, g_3), e_2(g_2, g_3), e_3(g_2, g_3)\}$$

```
\newcommand{\COOL@notation@WeierstrassPHalfPeriodValuesDisplay}{sf}
\newcommand{\WeierstrassPHalfPeriodValues}[1]
{%
\listval{#1}{0}%
\ifthenelse{\NOT \value{COOL@listpointer} = 2}%
{%
\PackageError{cool}{Invalid Argument}%
{`WeierstrassPHalfPeriodValues' can only accept%
 a comma separated list of length 2}%
}%
% Else
{%
\{  e_1\COOL@hideOnSF{WeierstrassPHalfPeriodValuesDisplay}{\!\inp{#1}},%
e_2\COOL@hideOnSF{WeierstrassPHalfPeriodValuesDisplay}{\!\inp{#1}},%
e_3\COOL@hideOnSF{WeierstrassPHalfPeriodValuesDisplay}{\!\inp{#1}}%
\}%
}%
}
\newcommand{\WeiPHalfPeriodVal}[1]{\WeierstrassPHalfPeriodValues{#1}}
```

\WeierstrassZetaHalfPeriodValues Weierstrass zeta function values at half-periods

\WeiZetaHalfPeriodVal

\Style{WeierstrassZetaHalfPeriodValuesDisplay=sf} (Default)

\WeierstrassZetaHalfPeriodValues{g_2,g_3}

\WeiZetaHalfPeriodVal{g_2,g_3}

$$\{\eta_1, \eta_2, \eta_3\}$$

\Style{WeierstrassZetaHalfPeriodValuesDisplay=ff}

\WeierstrassZetaHalfPeriodValues{g_2,g_3}

\WeiZetaHalfPeriodVal{g_2,g_3}

$$\{\eta_1(g_2, g_3), \eta_2(g_2, g_3), \eta_3(g_2, g_3)\}$$

```
\newcommand{\COOL@notation@WeierstrassZetaHalfPeriodValuesDisplay}{sf}
\newcommand{\WeierstrassZetaHalfPeriodValues}[1]
{%
\listval{#1}{0}%
\ifthenelse{\NOT \value{COOL@listpointer} = 2}%
{%
\PackageError{cool}{Invalid Argument}%
{`WeierstrassZetaHalfPeriodValues' can only accept%
 a comma separated list of length 2}%
```

```
}%
% Else
{%
\{%
\eta_1\COOL@hideOnSF%
{WeierstrassZetaHalfPeriodValuesDisplay}{\!\inp{#1}},%
\eta_2\COOL@hideOnSF%
{WeierstrassZetaHalfPeriodValuesDisplay}{\!\inp{#1}},%
\eta_3\COOL@hideOnSF%
{WeierstrassZetaHalfPeriodValuesDisplay}{\!\inp{#1}}%
\}%
}%
}
\newcommand{\WeiZetaHalfPeriodVal}[1]%
{\WeierstrassZetaHalfPeriodValues{#1}}
```

1.3.31 Jacobi Functions

`\JacobiAmplitude` Amplitude, `\JacobiAmplitude{z}{m}`, $\mathrm{am}(z \,|\, m)$

```
\newcommand{\COOL@notation@JacobiAmplitudeParen}{p}
\DeclareMathOperator{\JacobiAmplitudeSymb}{am}
\newcommand{\JacobiAmplitude}[2]{%
\JacobiAmplitudeSymb\!\COOL@decide@paren%
{JacobiAmplitude}{#1 \left| \, #2 \right.\!\!}%
}
```

`\JacobiCD` `\JacobiCDInv` Jacobi elliptic function and its inverse

`\JacobiCD{z}{m}` $\mathrm{cd}(z \,|\, m)$
`\JacobiCDInv{z}{m}` $\mathrm{cd}^{-1}(z \,|\, m)$

```
\newcommand{\COOL@notation@JacobiCDParen}{p}
\newcommand{\COOL@notation@JacobiCDInvParen}{p}
\DeclareMathOperator{\JacobiCDSymb}{cd}
\newcommand{\JacobiCD}[2]{%
\JacobiCDSymb\!\COOL@decide@paren%
{JacobiCD}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiCDInv}[2]{%
\JacobiCDSymb^{-1}\!\COOL@decide@paren%
{JacobiCDInv}{#1 \left| \, #2 \right.\!\!}%
}
```

`\JacobiCN` `\JacobiCNInv` Jacobi elliptic function and its inverse

`\JacobiCN{z}{m}` $\mathrm{cn}(z \,|\, m)$
`\JacobiCNInv{z}{m}` $\mathrm{cn}^{-1}(z \,|\, m)$

```
\newcommand{\COOL@notation@JacobiCNParen}{p}
\newcommand{\COOL@notation@JacobiCNInvParen}{p}
\DeclareMathOperator{\JacobiCNSymb}{cn}
\newcommand{\JacobiCN}[2]{%
\JacobiCNSymb\!\COOL@decide@paren%
```

```
{JacobiCN}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiCNInv}[2]{%
\JacobiCNSymb^{-1}\!\COOL@decide@paren%
{JacobiCNInv}{#1 \left| \, #2 \right.\!\!}%
}
```

\JacobiCS
\JacobiCSInv

Jacobi elliptic function and its inverse

`\JacobiCS{z}{m}`	$\mathrm{cs}(z \,|\, m)$
`\JacobiCSInv{z}{m}`	$\mathrm{cs}^{-1}(z \,|\, m)$

```
\newcommand{\COOL@notation@JacobiCSParen}{p}
\newcommand{\COOL@notation@JacobiCSInvParen}{p}
\DeclareMathOperator{\JacobiCSSymb}{cs}
\newcommand{\JacobiCS}[2]{%
\JacobiCSSymb\!\COOL@decide@paren%
{JacobiCS}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiCSInv}[2]{%
\JacobiCSSymb^{-1}\!\COOL@decide@paren%
{JacobiCSInv}{#1 \left| \, #2 \right.\!\!}%
}
```

\JacobiDC
\JacobiDCInv

Jacobi elliptic function and its inverse

`\JacobiDC{z}{m}`	$\mathrm{dc}(z \,|\, m)$
`\JacobiDCInv{z}{m}`	$\mathrm{dc}^{-1}(z \,|\, m)$

```
\newcommand{\COOL@notation@JacobiDCParen}{p}
\newcommand{\COOL@notation@JacobiDCInvParen}{p}
\DeclareMathOperator{\JacobiDCSymb}{dc}
\newcommand{\JacobiDC}[2]{%
\JacobiDCSymb\!\COOL@decide@paren%
{JacobiDC}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiDCInv}[2]{%
\JacobiDCSymb^{-1}\!\COOL@decide@paren%
{JacobiDCInv}{#1 \left| \, #2 \right.\!\!}%
}
```

\JacobiDN
\JacobiDNInv

Jacobi elliptic function and its inverse

`\JacobiDN{z}{m}`	$\mathrm{dn}(z \,|\, m)$
`\JacobiDNInv{z}{m}`	$\mathrm{dn}^{-1}(z \,|\, m)$

```
\newcommand{\COOL@notation@JacobiDNParen}{p}
\newcommand{\COOL@notation@JacobiDNInvParen}{p}
\DeclareMathOperator{\JacobiDNSymb}{dn}
\newcommand{\JacobiDN}[2]{%
\JacobiDNSymb\!\COOL@decide@paren%
{JacobiDN}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiDNInv}[2]{%
\JacobiDNSymb^{-1}\!\COOL@decide@paren%
```

```
{JacobiDNInv}{#1 \left| \, #2 \right.\!\!}%
}
```

\JacobiDS
\JacobiDSInv

Jacobi elliptic function and its inverse

`\JacobiDS{z}{m}`	$\mathrm{ds}(z \,|\, m)$
`\JacobiDSInv{z}{m}`	$\mathrm{ds}^{-1}(z \,|\, m)$

```
\newcommand{\COOL@notation@JacobiDSParen}{p}
\newcommand{\COOL@notation@JacobiDSInvParen}{p}
\DeclareMathOperator{\JacobiDSSymb}{ds}
\newcommand{\JacobiDS}[2]{%
\JacobiDSSymb\!\COOL@decide@paren%
{JacobiDS}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiDSInv}[2]{%
\JacobiDSSymb^{-1}\!\COOL@decide@paren%
{JacobiDSInv}{#1 \left| \, #2 \right.\!\!}%
}
```

\JacobiNC
\JacobiNCInv

Jacobi elliptic function and its inverse

`\JacobiNC{z}{m}`	$\mathrm{nc}(z \,|\, m)$
`\JacobiNCInv{z}{m}`	$\mathrm{nc}^{-1}(z \,|\, m)$

```
\newcommand{\COOL@notation@JacobiNCParen}{p}
\newcommand{\COOL@notation@JacobiNCInvParen}{p}
\DeclareMathOperator{\JacobiNCSymb}{nc}
\newcommand{\JacobiNC}[2]{%
\JacobiNCSymb\!\COOL@decide@paren%
{JacobiNC}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiNCInv}[2]{%
\JacobiNCSymb^{-1}\!\COOL@decide@paren%
{JacobiNCInv}{#1 \left| \, #2 \right.\!\!}%
}
```

\JacobiND
\JacobiNDinv

Jacobi elliptic function and its inverse

`\JacobiND{z}{m}`	$\mathrm{nd}(z \,|\, m)$
`\JacobiNDInv{z}{m}`	$\mathrm{nd}^{-1}(z \,|\, m)$

```
\newcommand{\COOL@notation@JacobiNDParen}{p}
\newcommand{\COOL@notation@JacobiNDInvParen}{p}
\DeclareMathOperator{\JacobiNDSymb}{nd}
\newcommand{\JacobiND}[2]{%
\JacobiNDSymb\!\COOL@decide@paren%
{JacobiND}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiNDInv}[2]{%
\JacobiNDSymb^{-1}\!\COOL@decide@paren%
{JacobiNDInv}{#1 \left| \, #2 \right.\!\!}%
}
```

\JacobiNS
\JacobiNSInv

Jacobi elliptic function and its inverse

```
\JacobiNS{z}{m}      ns(z | m)
\JacobiNSInv{z}{m}   ns^{-1}(z | m)
```

```
\newcommand{\COOL@notation@JacobiNSParen}{p}
\newcommand{\COOL@notation@JacobiNSInvParen}{p}
\DeclareMathOperator{\JacobiNSSymb}{ns}
\newcommand{\JacobiNS}[2]{%
\JacobiNSSymb\!\COOL@decide@paren%
{JacobiNS}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiNSInv}[2]{%
\JacobiNSSymb^{-1}\!\COOL@decide@paren%
{JacobiNSInv}{#1 \left| \, #2 \right.\!\!}%
}
```

`\JacobiSC` `\JacobiSCInv` Jacobi elliptic function and its inverse

```
\JacobiSC{z}{m}      sc(z | m)
\JacobiSCInv{z}{m}   sc^{-1}(z | m)
```

```
\newcommand{\COOL@notation@JacobiSCParen}{p}
\newcommand{\COOL@notation@JacobiSCInvParen}{p}
\DeclareMathOperator{\JacobiSCSymb}{sc}
\newcommand{\JacobiSC}[2]{%
\JacobiSCSymb\!\COOL@decide@paren%
{JacobiSC}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiSCInv}[2]{%
\JacobiSCSymb^{-1}\!\COOL@decide@paren%
{JacobiSCInv}{#1 \left| \, #2 \right.\!\!}%
}
```

`\JacobiSD` `\JacobiSDInv` Jacobi elliptic function and its inverse

```
\JacobiSD{z}{m}      sd(z | m)
\JacobiSDInv{z}{m}   sd^{-1}(z | m)
```

```
\newcommand{\COOL@notation@JacobiSDParen}{p}
\newcommand{\COOL@notation@JacobiSDInvParen}{p}
\DeclareMathOperator{\JacobiSDSymb}{sd}
\newcommand{\JacobiSD}[2]{%
\JacobiSDSymb\!\COOL@decide@paren%
{JacobiSD}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiSDInv}[2]{%
\JacobiSDSymb^{-1}\!\COOL@decide@paren%
{JacobiSDInv}{#1 \left| \, #2 \right.\!\!}%
}
```

`\JacobiSN` `\JacobiSNInv` Jacobi elliptic function and its inverse

```
\JacobiSN{z}{m}      sn(z | m)
\JacobiSNInv{z}{m}   sn^{-1}(z | m)
```

```
\newcommand{\COOL@notation@JacobiSNParen}{p}
```

```
\newcommand{\COOL@notation@JacobiSNInvParen}{p}
\DeclareMathOperator{\JacobiSNSymb}{sn}
\newcommand{\JacobiSN}[2]{%
\JacobiSNSymb\!\COOL@decide@paren%
{JacobiSN}{#1 \left| \, #2 \right.\!\!}%
}
\newcommand{\JacobiSNInv}[2]{%
\JacobiSNSymb^{-1}\!\COOL@decide@paren%
{JacobiSNInv}{#1 \left| \, #2 \right.\!\!}%
}
```

1.3.32 Modular Functions

`\DedekindEta` Dedekind eta modular function, `\DedekindEta{z}`, $\eta(z)$

```
\newcommand{\COOL@notation@DedekindEtaParen}{p}
\newcommand{\DedekindEta}[1]{\eta\!\COOL@decide@paren{DedekindEta}{#1}}
```

`\KleinInvariantJ` Klein invariant modular function, `\KleinInvariantJ{z}`, $J(z)$

```
\newcommand{\COOL@notation@KleinInvariantJParen}{p}
\newcommand{\KleinInvariantJ}[1]%
{J\!\COOL@decide@paren{KleinInvariantJ}{#1}}
```

`\ModularLambda` Modular lambda function, `\ModularLambda{z}`, $\lambda(z)$

```
\newcommand{\COOL@notation@ModularLambdaParen}{p}
\newcommand{\ModularLambda}[1]%
{\lambda\!\COOL@decide@paren{ModularLambda}{#1}}
```

`\EllipticNomeQ` `\EllipticNomeQInv` Nome and its inverse

`\EllipticNomeQ{m}`	$q(m)$
`\EllipticNomeQInv{m}`	$q^{-1}(m)$

```
\newcommand{\COOL@notation@EllipticNomeQParen}{p}
\newcommand{\COOL@notation@EllipticNomeQInvParen}{p}
\newcommand{\EllipticNomeQ}[1]%
{q\!\COOL@decide@paren{EllipticNomeQ}{#1}}
\newcommand{\EllipticNomeQInv}[1]%
{q^{-1}\!\COOL@decide@paren{EllipticNomeQ}{#1}}
```

1.3.33 Arithmetic Geometric Mean

`\ArithGeoMean` `\AGM` Arithmetic Geometric Mean

`\ArithGeoMean{a}{b}`	$\mathrm{agm}(a,b)$
`\AGM{a}{b}`	$\mathrm{agm}(a,b)$

```
\newcommand{\COOL@notation@ArithGeoMeanParen}{p}
\DeclareMathOperator{\ArithGeoMeanSymb}{agm}
\newcommand{\ArithGeoMean}[2]%
{\ArithGeoMeanSymb\!\COOL@decide@paren{ArithGeoMean}{#1, #2}}
\newcommand{\AGM}[2]{\ArithGeoMean{#1}{#2}}
```

1.3.34 Elliptic Exp and Log

\EllipticExp Elliptic exponential
\EExp

`\EllipticExp{z}{a,b}`	$\mathrm{eexp}(z;a,b)$
`\EExp{z}{a,b}`	$\mathrm{eexp}(z;a,b)$

```
\newcommand{\COOL@notation@EllipticExpParen}{p}
\DeclareMathOperator{\EllipticExpSymb}{eexp}
\newcommand{\EllipticExp}[2]{%
\liststore{#2}{COOL@EllipticExp@arg@}
\listval{#2}{0}%
\ifthenelse{\NOT \value{COOL@listpointer} = 2}%
{%
\PackageError{cool}{Invalid Argument}%
{`EllipticExp' second argument must be
 a comma separated list of length 2}%
}%
% Else
{%
\EllipticExpSymb\!\COOL@decide@paren{EllipticExp}{#1; #2}%
}%
}
\newcommand{\EExp}[2]{\EllipticExp{#1}{#2}}
```

\EllipticLog Elliptic logarithm
\ELog

`\EllipticLog{z_1,z_2}{a,b}`	$\mathrm{elog}(z_1,z_2;a,b)$
`\ELog{z_1,z_2}{a,b}`	$\mathrm{elog}(z_1,z_2;a,b)$

```
\newcommand{\COOL@notation@EllipticLogParen}{p}
\DeclareMathOperator{\EllipticLogSymb}{elog}
\newcommand{\EllipticLog}[2]{%
\liststore{#1}{COOL@EllipticLog@arg@z@}%
\liststore{#2}{COOL@EllipticLog@arg@a@}%
\listval{#1}{0}%
\ifthenelse{\NOT \value{COOL@listpointer} = 2}%
{%
\PackageError{cool}{Invalid Argument}%
{`EllipticLog' first argument must be
 a comma separated list of length 2}%
}%
% Else
{%
\listval{#2}{0}%
\ifthenelse{\NOT \value{COOL@listpointer} = 2}%
{%
\PackageError{cool}{Invalid Argument}%
{`EllipticLog' second argument must be%
 a comma separated list of length 2}%
}%
% Else
{%
```

```
\EllipticLogSymb\!\COOL@decide@paren{EllipticLog}{#1; #2}%
}%
}%
}
\newcommand{\ELog}[2]{\EllipticLog{#1}{#2}}
```

1.3.35 Zeta Functions

`\RiemannZeta` Riemann Zeta Function

`\RiemannZeta{s}`	$\zeta(s)$
`\Zeta{s}`	$\zeta(s)$

```
\newcommand{\RiemannZeta}[1]{\Zeta{#1}}
```

`\HurwitzZeta` Hurwitz Zeta Function

`\HurwitzZeta{s}{a}`	$\zeta(s,a)$
`\Zeta{s,a}`	$\zeta(s,a)$

```
\newcommand{\HurwitzZeta}[2]{\Zeta{#1,#2}}
```

`\Zeta` Riemann and Hurwitz Zeta

Riemann Zeta	`\Zeta{s}`	$\zeta(s)$
Hurwitz Zeta	`\Zeta{s,a}`	$\zeta(s,a)$

```
\newcommand{\COOL@notation@ZetaParen}{p}
\newcommand{\Zeta}[1]{%
\liststore{#1}{COOL@Zeta@arg@}%
\listval{#1}{0}% get the list length
\ifthenelse{\value{COOL@listpointer} = 2}%
{%
\zeta\!\COOL@decide@paren{Zeta}{\COOL@Zeta@arg@i,\COOL@Zeta@arg@ii}%
}%
% else
{%
\ifthenelse{\value{COOL@listpointer} = 1}%
{%
\zeta\!\COOL@decide@paren{Zeta}{#1}%
}%
% else
{%
\PackageError{cool}{`Zeta' Invalid Argument}%
{the Zeta function can only accept%
 a comma deliminated list of length 1 or 2}
}%
}%
}%
```

`\RiemannSiegelTheta` Riemann-Siegel Theta Function, `\RiemannSiegelTheta{z}`, $\vartheta(z)$

```
\newcommand{\COOL@notation@RiemannSiegelThetaParen}{p}
\newcommand{\RiemannSiegelTheta}[1]%
{\vartheta\!\COOL@decide@paren{RiemannSiegelTheta}{#1}}
```

\RiemannSiegelZ Riemann-Siegel Z Function, \RiemannSiegelZ{z}, $Z(z)$

```
\newcommand{\COOL@notation@RiemannSiegelZParen}{p}
\newcommand{\RiemannSiegelZ}[1]%
{Z\!\COOL@decide@paren{RiemannSiegelZ}{#1}}
```

\StieltjesGamma Stieltjes Constant, \StieltjesGamma{n}, γ_n

```
\newcommand{\StieltjesGamma}[1]{\gamma_{#1}}
```

\LerchPhi Lerch transcendent, \LerchPhi{z}{s}{a}, $\Phi(z,s,a)$

```
\newcommand{\COOL@notation@LerchPhiParen}{p}
\newcommand{\LerchPhi}[3]{\Phi\!\COOL@decide@paren{LerchPhi}{#1,#2,#3}}
```

1.3.36 Polylogarithms

\NielsenPolyLog Nielsen Polylogarithm, \NielsenPolyLog{\nu}{p}{z}, $S_\nu^p(z)$

```
\newcommand{\COOL@notation@NielsenPolyLogParen}{p}
\newcommand{%
\NielsenPolyLog}[3]{S_{#1}^{#2}\!%
\COOL@decide@paren{NielsenPolyLog}{#3}%
}
```

\PolyLog Polylogarithm

Nielsen PolyLog	\PolyLog{\nu,p,z}	$S_\nu^p(z)$
PolyLog	\PolyLog{\nu,z}	$\mathrm{Li}_\nu(z)$

```
\newcommand{\COOL@notation@PolyLogParen}{p}
\DeclareMathOperator{\PolyLogSymb}{Li}
\newcommand{\PolyLog}[1]{%
\liststore{#1}{COOL@PolyLog@arg@}%
\listval{#1}{0}%
\ifthenelse{\value{COOL@listpointer} = 3}%
{%
\NielsenPolyLog{\COOL@PolyLog@arg@i}%
{\COOL@PolyLog@arg@ii}{\COOL@PolyLog@arg@iii}%
}%
% else
{%
\ifthenelse{ \value{COOL@listpointer} = 2 }%
{%
\PolyLogSymb_{\COOL@PolyLog@arg@i}\!%
\COOL@decide@paren{PolyLog}{\COOL@PolyLog@arg@ii}%
}%
% else
{%
\PackageError{cool}{`PolyLog' Invalid Argument}%
{This function returns either the Polylogarithm or the%
 Nielsen Polylogarithm.  It therefore only accepts a comma%
 deliminated list of length two or three (1 or 2 commas)}%
}%
}%
```

```
}
```

`\DiLog` Dilogarithm (alias for `\PolyLog{2,x}`); `\DiLog{x}`, $\mathrm{Li}_2(x)$

```
\newcommand{\DiLog}[1]{\PolyLog{2,#1}}
```

1.3.37 Mathieu Functions

`\MathieuC` Even Mathieu Function, `\MathieuC{a}{q}{z}`, $\mathrm{Ce}(a,q,z)$

```
\newcommand{\COOL@notation@MathieuCParen}{p}
\DeclareMathOperator{\MathieuCSymb}{Ce}
\newcommand{\MathieuC}[3]%
{\MathieuCSymb\!\COOL@decide@paren{MathieuC}{#1,#2,#3}}
```

`\MathieuS` Odd Mathieu Function, `\MathieuS{a}{q}{z}`, $\mathrm{Se}(a,q,z)$

```
\newcommand{\COOL@notation@MathieuSParen}{p}
\DeclareMathOperator{\MathieuSSymb}{Se}
\newcommand{\MathieuS}[3]%
{\mathord{\MathieuSSymb}\!\COOL@decide@paren{MathieuS}{#1,#2,#3}}
```

1.3.38 Mathieu Characteristics

`\MathieuCharacteristicA` `\MathieuCharisticA` Characteristic Value of Even Mathieu Function

`\MathieuCharacteristicA{r}{q}`	$a_r(q)$
`\MathieuCharisticA{r}{q}`	$a_r(q)$

```
\newcommand{\COOL@notation@MathieuCharacteristicAParen}{p}
\newcommand{\MathieuCharacteristicA}[2]%
{a_{#1}\!\COOL@decide@paren{MathieuCharacteristicA}{#2}}
\newcommand{\MathieuCharisticA}[2]{\MathieuCharacteristicA{#1}{#2}}
```

`\MathieuCharacteristicB` `\MathieuCharisticB` Characteristic Value of Even Mathieu Fucntion

`\MathieuCharacteristicB{r}{q}`	$b_r(q)$
`\MathieuCharisticB{r}{q}`	$b_r(q)$

```
\newcommand{\COOL@notation@MathieuCharacteristicBParen}{p}
\newcommand{\MathieuCharacteristicB}[2]%
{b_{#1}\!\COOL@decide@paren{MathieuCharacteristicB}{#2}}
\newcommand{\MathieuCharisticB}[2]{\MathieuCharacteristicB{#1}{#2}}
```

`\MathieuCharacteristicExponent` `\MathieuCharisticExp` Characteristic Exponent of a Mathieu Fucntion

`\MathieuCharateristicExponent{a}{q}`	$r(a,q)$
`\MathieuCharisticExp{a}{q}`	$r(a,q)$

```
\newcommand{\COOL@notation@MathieuCharacteristicExponentParen}{p}
\newcommand{\MathieuCharacteristicExponent}[2]%
{r\!\COOL@decide@paren{MathieuCharacteristicExponent}{#1,#2}}
\newcommand{\MathieuCharisticExp}[2]%
{\MathieuCharacteristicExponent{#1}{#2}}
```

1.3.39 Complex variables

`\Abs` Absolute value, `\Abs{z}`, $|z|$

```
\newcommand{\Abs}[1]{ \left|#1\right| }
```

`\Arg` Argument, `\Arg{z}`, $\arg(z)$

```
\newcommand{\Arg}[1]{ \arg\!\inp{#1} }
```

`\Conjugate` `\Conj` Complex Conjugate

`\Conj{z}`	z^*
`\Conjugate{z}`	z^*

```
\def\COOL@notation@Conjugate{star}
\newcommand{\COOL@notation@ConjugateParen}{inv}
\newcommand{\Conjugate}[1]{\Conj{#1}}
\newcommand{\Conj}[1]{%
\ifthenelse{\equal{\COOL@notation@Conjugate}{bar}}%
{%
\bar{#1}%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@Conjugate}{overline}}%
{%
\overline{#1}%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@Conjugate}{star}}%
{%
\COOL@decide@paren{Conjugate}{#1}^*%
}%
% Else
{%
\PackageError{cool}{Invalid Option Sent}%
{`Conjugate' can only be set at `star', `bar', or `overline'}%
}%
}}%
}
```

`\Real` Real Part, `\Real{z}`, $\operatorname{Re} z$

```
\newcommand{\COOL@notation@RealParen}{none}
\DeclareMathOperator{\RealSymb}{Re}
\newcommand{\Real}[1]{%
```

we put a space if there is no parentheses, or leave it out if there are

```
\ifthenelse{\equal{\COOL@notation@ImagParen}{none}}%
{%
\RealSymb{#1}%
}%
% Else
{%
\RealSymb\!\COOL@decide@paren{Imag}{#1}%
```

```
}%
}
```

`\Imag` Imaginary Part, `\Imag{z}`, $\operatorname{Im} z$

```
\newcommand{\COOL@notation@ImagParen}{none}
\DeclareMathOperator{\ImagSymb}{Im}
\newcommand{\Imag}[1]{%
```

we put a space if there is no parentheses, or leave it out if there are

```
\ifthenelse{\equal{\COOL@notation@ImagParen}{none}}%
{%
\ImagSymb{#1}%
}%
% Else
{%
\ImagSymb\!\COOL@decide@paren{Imag}{#1}%
}%
}
```

`\Sign` Sign function, `\Sign{x}`, $\operatorname{sgn}(x)$

```
\newcommand{\COOL@notation@SignParen}{p}
\newcommand{\Sign}[1]{\operatorname{sgn}\!\COOL@decide@paren{Sign}{#1}}
```

1.3.40 Number Theory Functions

`\FactorInteger` Prime decomposition, `\Factors{n}`, $\operatorname{factors}(n)$

`\Factors`

```
\newcommand{\COOL@notation@FactorIntegerParen}{p}
\DeclareMathOperator{\FactorIntegerSymb}{factors}
\newcommand{\FactorInteger}[1]%
{\FactorIntegerSymb\!\COOL@decide@paren{FactorInteger}{#1}}
\newcommand{\Factors}[1]{\FactorInteger{#1}}
```

`\Divisors` Divisors, `\Divisors{n}`, $\operatorname{divisors}(n)$

```
\newcommand{\COOL@notation@DivisorsParen}{p}
\DeclareMathOperator{\DivisorsSymb}{divisors}
\newcommand{\Divisors}[1]%
{\mathord{\DivisorsSymb}\!\COOL@decide@paren{Divisors}{#1}}
```

`\Prime` The nth Prime, `\Prime{n}`, $\operatorname{prime}(n)$

```
\newcommand{\COOL@notation@PrimeParen}{p}
\DeclareMathOperator{\PrimeSymb}{prime}
\newcommand{\Prime}[1]%
{\mathord{\PrimeSymb}\!\COOL@decide@paren{Prime}{#1}}
```

`\PrimePi` Prime counting function, `\PrimePi{x}`, $\pi(x)$

```
\newcommand{\COOL@notation@PrimePiParen}{p}
\newcommand{\PrimePi}[1]{\pi\!\COOL@decide@paren{PrimePi}{#1}}
```

`\DivisorSigma` Sum of divisor powers, `\DivisorSigma{k}{n}`, $\sigma_k(n)$

```
\newcommand{\COOL@notation@DivisorSigmaParen}{p}
\newcommand{\DivisorSigma}[2]%
{\sigma_{#1}\!\COOL@decide@paren{DivisorSigma}{#2}}
```

`\EulerPhi` Euler Totient Function, `\EulerPhi{x}`, $\varphi(x)$

```
\newcommand{\COOL@notation@EulerPhiParen}{p}
\newcommand{\EulerPhi}[1]{\varphi\!\COOL@decide@paren{EulerPhi}{#1}}
```

`\MoebiusMu` Moebius Function, `\MoebiusMu{x}`, $\mu(x)$

```
\newcommand{\COOL@notation@MoebiusMuParen}{p}
\newcommand{\MoebiusMu}[1]{\mu\!\COOL@decide@paren{MoebiusMu}{#1}}
```

`\JacobiSymbol` Jacobi Symbol, `\JacobiSymbol{n}{m}`, $\left(\frac{n}{m}\right)$

```
\newcommand{\JacobiSymbol}[2]{\inp{\frac{#1}{#2}}}
```

`\CarmichaelLambda` Carmichael Lambda Function, `\CarmichaelLambda{x}`, $\lambda(x)$

```
\newcommand{\COOL@notation@CarmichaelLambdaParen}{p}
\newcommand{\CarmichaelLambda}[1]%
{\lambda\!\COOL@decide@paren{CarmichaelLambda}{#1}}
```

`\DigitCount` Count the digits of an integer n for a given base b

`\DigitCount{n}{b}`

$$\{s_b^{(1)}(n), s_b^{(2)}(n), \ldots, s_b^{(b)-1}(n), s_b^{(0)}(n)\}$$

```
\newcommand{\DigitCount}[2]{%
\isint{#2}%
\ifthenelse{\boolean{COOL@isint}}%
{%
\{%
\setcounter{COOL@ct@}{#2}%
\addtocounter{COOL@ct@}{-1}%
\forLoop{1}{\arabic{COOL@ct@}}{COOL@ct}%
{%
s^{\arabic{COOL@ct}}_{#2}\!\inp{#1},
}%
s^{\inp{0}}_{#2}\!\inp{#1}%
\}%
}%
% else
{%
\{%
s^{\inp{1}}_{#2}\!\inp{#1},%
s^{\inp{2}}_{#2}\!\inp{#1},%
\ldots,%
s^{\inp{#2} - 1}_{#2}\!\inp{#1},%
s^{\inp{0}}_{#2}\!\inp{#1}%
\}%
}%
}
```

1.3.41 Generalized Functions

\DiracDelta Dirac Delta Function, `\DiracDelta{x}`, $\delta(x)$

```
\newcommand{\COOL@notation@DiracDeltaParen}{p}
\newcommand{\DiracDelta}[1]{\delta\!\COOL@decide@paren{DiracDelta}{#1}}
```

\HeavisideStep Heaviside Step Function
\UnitStep

`\HeavisideStep{x}`	$\theta(x)$
`\UnitStep{x}`	$\theta(x)$

```
\newcommand{\COOL@notation@HeavisideStepParen}{p}
\newcommand{\HeavisideStep}[1]%
{\theta\!\COOL@decide@paren{HeavisideStep}{#1}}
\newcommand{\UnitStep}[1]{\HeavisideStep{#1}}
```

1.3.42 Calculus

\COOL@notation@DDisplayFunc Both `\D` and `\pderiv` are controlled by these keys.
\COOL@notation@DShorten `DDisplayFunc` controls how the function is displayed, it can take the values:

inset	Display as $\frac{df}{dx}$
outset	Display as $\frac{d}{dx}f$

`DShorten` is for multiple derivatives. it can take the values

true	force derivatives to be consolidated, as in $\frac{d^2}{dxdy}f$
false	expand derivatives as in $\frac{d}{dx}\frac{d}{dx}f$

```
\newcounter{COOL@multideriv}
\newcommand{\COOL@notation@DDisplayFunc}{inset}
\newcommand{\COOL@notation@DShorten}{true}
```

\COOL@derivative Both `\D` and `pderiv` have the same basic operation, so a macro is defined that does the internals

`\COOL@derivative{`⟨*derivative power(s)*⟩`}{`⟨*function*⟩`}{`⟨*wrt*⟩`}{`⟨*symbol*⟩`}`

⟨*wrt*⟩ is a comma separated list of length ≥ 1.

⟨*symbol*⟩ is passed by `\D` or `\pderiv` and is 'd' or '∂' respectively

```
\COOL@derivative{2,3}{f}{x,y,z}{d}
```
$\frac{d^8 f}{dx^2\,dy^3\,dz^3}$

```
\COOL@derivative{2,3,4,5}{f}{x,y,z}{d}
```
$\frac{d^9 f}{dx^2\,dy^3\,dz^4}$

```
\COOL@derivative{2,n,1}{f}{x,y,z}{d}
```
$\frac{d^{2+n+1} f}{dx^2\,dy^n\,dz}$

```
\COOL@derivative{2,n}{f}{x,y,z}{d}
```
$\frac{d^{2+n+n} f}{dx^2\,dy^n\,dz^n}$

```
\Style{DDisplayFunc=outset}
\COOL@derivative{2,n}{f}{x,y,z}{d}
```
$\frac{d^{2+n+n}}{dx^2\,dy^n\,dz^n} f$

```
\Style{DShorten=false,DDisplayFunc=inset}
\COOL@derivative{2,n}{f}{x,y,z}{d}
```
$\frac{d^2}{dx^2}\frac{d^n}{dy^n}\frac{d^n f}{dz^n}$

```
\COOL@derivative{2,3,4,5}{f}{x,y,z}{d}
```
$\frac{d^2}{dx^2}\frac{d^3}{dy^3}\frac{d^4 f}{dz^4}$

```
\Style{DShorten=false,DDisplayFunc=outset}
\COOL@derivative{2,n}{f}{x,y,z}{d}
```
$\frac{d^2}{dx^2}\frac{d^n}{dy^n}\frac{d^n}{dz^n} f$

```
\newcommand{\COOL@derivative}[4]{%
```

Get the length of ⟨*wrt*⟩ argument. `\listval{#3}{0}` gives the length of the list since lists begin indexing at 1.

```
\listval{#3}{0}%
\setcounter{COOL@listlen}{\value{COOL@listpointer}}%
```

Store the ⟨*wrt*⟩ list and get the length of ⟨*derivative power(s)*⟩.

```
\liststore{#3}{COOL@deriv@wrt@}%
\listval{#1}{0}%
\setcounter{COOL@ct}{\value{COOL@listpointer}}%
\ifthenelse{\value{COOL@ct}>\value{COOL@listlen}}%
{\setcounter{COOL@ct}{\value{COOL@listlen}}}{}%
\liststore{#1}{COOL@deriv@powers@}%
```

Check to see if all of the powers are integers—if they are, then we may sum them in the usual sense

```
\isint{\COOL@deriv@powers@i}%
\setcounter{COOL@multideriv}{2}%
\whiledo{ \boolean{COOL@isint} \AND
\NOT \value{COOL@multideriv}>\value{COOL@ct}  }%
{%
\def\COOL@tempd%
{\csname COOL@deriv@powers@\roman{COOL@multideriv}\endcsname}%
\isint{\COOL@tempd}%
\stepcounter{COOL@multideriv}%
}%
```

If the length of ⟨*derivative power(s)*⟩ is less than the length of ⟨*wrt*⟩, then we assume that the last value applies to *all* the remaining derivatives.

```
\ifthenelse{ \equal{\COOL@notation@DShorten}{true} \AND
\equal{\COOL@notation@DDisplayFunc}{inset} }%
```

```
{%
\ifthenelse{ \boolean{COOL@isint} }%
{%
\def\COOL@temp@D@bot{}%
\setcounter{COOL@ct@}{0}%
\forLoop{1}{\value{COOL@ct}}{COOL@multideriv}%
{%
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@multideriv}\endcsname}%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
\addtocounter{COOL@ct@}{\COOL@power@temp}%
\ifthenelse{ \value{COOL@multideriv}=1 }{}%
{\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
}%
}%
```

we're done with the length of the ⟨*derivative power(s)*⟩ argument, and we want to start at it + 1 to add the remainders

```
\ifthenelse{\value{COOL@ct}<\value{COOL@listlen}}%
{%
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
\stepcounter{COOL@ct}%
\forLoop{\value{COOL@ct}}{\value{COOL@listlen}}{COOL@multideriv}%
{%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
\addtocounter{COOL@ct@}{\COOL@power@temp}%
\ifthenelse{ \value{COOL@multideriv}=1 }{}%
{\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
}%
```

```
}%
}%
% Else
{}%
\ifthenelse{\value{COOL@ct@}=1}%
{%
\frac{#4 #2}{\COOL@temp@D@bot}%
}%
% Else
{%
\frac{#4^{\arabic{COOL@ct@}} #2}{\COOL@temp@D@bot}%
}%
}%
% Else
{%
```

Powers are not all Integers

```
\edef\COOL@temp@D@bot{}%
\def\COOL@temp@D@top@power{}%
\forLoop{1}{\value{COOL@ct}}{COOL@multideriv}%
{%
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@multideriv}\endcsname}%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
\ifthenelse{ \value{COOL@multideriv} = 1}%
{%
\edef\COOL@temp@D@top@power{\COOL@power@temp}%
}%
% Else
{%
\edef\COOL@temp@D@top@power%
{\COOL@temp@D@top@power + \COOL@power@temp}%
\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}%
}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
}%
}%
```

we're done with the length of the ⟨*derivative power(s)*⟩ argument, and we want to start at it + 1 to add the remainders

```
\ifthenelse{\value{COOL@ct}<\value{COOL@listlen}}%
{%
```

```
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
\stepcounter{COOL@ct}%
\forLoop{\value{COOL@ct}}{\value{COOL@listlen}}{COOL@multideriv}%
{%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
\ifthenelse{ \value{COOL@multideriv} = 1}%
{%
\edef\COOL@temp@D@top@power{\COOL@power@temp}%
}%
% Else
{%
\edef\COOL@temp@D@top@power%
{\COOL@temp@D@top@power + \COOL@power@temp}%
\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}%
}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
}%
}%
}%
% Else
{}%
\frac{#4^{\COOL@temp@D@top@power} #2}{\COOL@temp@D@bot}%
}%
}%
% Else If
{ \ifthenelse{ \equal{\COOL@notation@DShorten}{true} \AND
\equal{\COOL@notation@DDisplayFunc}{outset}  }%
{%
\ifthenelse{ \boolean{COOL@isint} }%
{%
\def\COOL@temp@D@bot{}%
\setcounter{COOL@ct@}{0}%
\forLoop{1}{\value{COOL@ct}}{COOL@multideriv}%
{%
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@multideriv}\endcsname}%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
\addtocounter{COOL@ct@}{\COOL@power@temp}%
```

```
\ifthenelse{ \value{COOL@multideriv}=1 }{}%
{\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
}%
}%
```

we're done with the length of the ⟨*derivative power(s)*⟩ argument, and we want to start at it + 1 to add the remainders

```
\ifthenelse{\value{COOL@ct}<\value{COOL@listlen}}%
{%
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
\stepcounter{COOL@ct}%
\forLoop{\value{COOL@ct}}{\value{COOL@listlen}}{COOL@multideriv}%
{%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
\addtocounter{COOL@ct@}{\COOL@power@temp}%
\ifthenelse{ \value{COOL@multideriv}=1 }{}%
{\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
}%
}%
}%
% Else
{}%
\ifthenelse{\value{COOL@ct@}=1}%
{%
\frac{#4}{\COOL@temp@D@bot} #2%
}%
% Else
{%
\frac{#4^{\arabic{COOL@ct@}}}{\COOL@temp@D@bot} #2%
}%
```

```
}%
% Else
{%
```

Powers are not all Integers

```
\edef\COOL@temp@D@bot{}%
\def\COOL@temp@D@top@power{}%
\forLoop{1}{\value{COOL@ct}}{COOL@multideriv}%
{%
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@multideriv}\endcsname}%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
\ifthenelse{ \value{COOL@multideriv} = 1}%
{%
\edef\COOL@temp@D@top@power{\COOL@power@temp}%
}%
% Else
{%
\edef\COOL@temp@D@top@power%
{\COOL@temp@D@top@power + \COOL@power@temp}%
\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}%
}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
}%
}%
```

we're done with the length of the ⟨*derivative power(s)*⟩ argument, and we want to start at it + 1 to add the remainders

```
\ifthenelse{\value{COOL@ct}<\value{COOL@listlen}}%
{%
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
\stepcounter{COOL@ct}%
\forLoop{\value{COOL@ct}}{\value{COOL@listlen}}{COOL@multideriv}%
{%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
\ifthenelse{ \value{COOL@multideriv} = 1}%
{%
\edef\COOL@temp@D@top@power{\COOL@power@temp}%
}%
% Else
```

```
{%
\edef\COOL@temp@D@top@power%
{\COOL@temp@D@top@power + \COOL@power@temp}%
\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}%
}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@bot%
{\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
}%
}%
}%
% Else
{}%
\frac{#4^{\COOL@temp@D@top@power} }{\COOL@temp@D@bot} #2%
}%
}%
% Else If
{ \ifthenelse{ \equal{\COOL@notation@DShorten}{false} \AND
\equal{\COOL@notation@DDisplayFunc}{inset} }%
{%
\def\COOL@temp@D@result{}%
\def\COOL@temp@D@bot{}%
\def\COOL@temp@D@top{}%
\setcounter{COOL@ct@}{\value{COOL@ct}}%
\addtocounter{COOL@ct@}{-1}
\forLoop{1}{\value{COOL@ct@}}{COOL@multideriv}%
{%
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@multideriv}\endcsname}%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@top{#4}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@top{#4^\COOL@power@temp}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
}%
\edef\COOL@temp@D@result%
{\COOL@temp@D@result \frac{\COOL@temp@D@top}{\COOL@temp@D@bot}}%
```

```
}%
```

we're done with the length of the ⟨*derivative power(s)*⟩ argument, and we want to start at it + 1 to add the remainders

```
\ifthenelse{\value{COOL@ct}<\value{COOL@listlen}}%
{%
```

Must pick up the one for `\value{COOL@ct}`

```
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@ct}\endcsname}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@top{#4}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@top{#4^\COOL@power@temp}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
}%
\edef\COOL@temp@D@result%
{\COOL@temp@D@result \frac{\COOL@temp@D@top}{\COOL@temp@D@bot}}%
```

Now add the ones beyond

```
\stepcounter{COOL@ct}%
\setcounter{COOL@ct@}{\value{COOL@listlen}}%
\addtocounter{COOL@ct@}{-1}%
\forLoop{\value{COOL@ct}}{\value{COOL@ct@}}{COOL@multideriv}%
{%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@top{#4}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@top{#4^\COOL@power@temp}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
}%
\edef\COOL@temp@D@result%
{\COOL@temp@D@result \frac{\COOL@temp@D@top}{\COOL@temp@D@bot}}%
}%
```

Must pick up the one for `\value{COOL@listlen}`

```
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@listlen}\endcsname}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@top{#4}%
```

```
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@top{#4^\COOL@power@temp}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
}%
\edef\COOL@temp@D@result%
{\COOL@temp@D@result \frac{\COOL@temp@D@top #2}{\COOL@temp@D@bot}}%
}%
% Else
{%
```

Must pick up the one for `\value{COOL@ct}`

```
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@ct}\endcsname}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@top{#4}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@top{#4^\COOL@power@temp}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
}%
\edef\COOL@temp@D@result%
{\COOL@temp@D@result \frac{\COOL@temp@D@top #2}{\COOL@temp@D@bot}}%
}%
\COOL@temp@D@result%
}%
% Else If
{ \ifthenelse{ \equal{\COOL@notation@DShorten}{false} \AND
\equal{\COOL@notation@DDisplayFunc}{outset} }%
{%
\def\COOL@temp@D@result{}%
\def\COOL@temp@D@bot{}%
\def\COOL@temp@D@top{}%
\forLoop{1}{\value{COOL@ct}}{COOL@multideriv}%
{%
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@multideriv}\endcsname}%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@top{#4}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
```

```
}%
% Else
{%
\edef\COOL@temp@D@top{#4^\COOL@power@temp}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
}%
\edef\COOL@temp@D@result%
{\COOL@temp@D@result \frac{\COOL@temp@D@top}{\COOL@temp@D@bot}}%
}%
```

we're done with the length of the ⟨*derivative power(s)*⟩ argument, and we want to start at it + 1 to add the remainders

```
\ifthenelse{\value{COOL@ct}<\value{COOL@listlen}}%
{%
\edef\COOL@power@temp%
{\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
\stepcounter{COOL@ct}%
\forLoop{\value{COOL@ct}}{\value{COOL@listlen}}{COOL@multideriv}%
{%
\edef\COOL@wrt@temp%
{\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
\ifthenelse{ \equal{\COOL@power@temp}{1} }%
{%
\edef\COOL@temp@D@top{#4}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
}%
% Else
{%
\edef\COOL@temp@D@top{#4^\COOL@power@temp}%
\edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
}%
\edef\COOL@temp@D@result%
{\COOL@temp@D@result \frac{\COOL@temp@D@top}{\COOL@temp@D@bot}}%
}%
}%
% Else
{%
}%
\COOL@temp@D@result #2
}%
% Else
{%
\PackageError{cool}{Invalid Option Sent}%
{DShorten can only be ‘true’ or ‘false’;%
 DDisplayFunc can only be ‘inset’ or ‘outset’}%
}%
}}}%
}
```

\D Derivatives

\pderiv

`\D{f}{x}`	$\frac{d}{dx} f$
`\D[n]{f}{x}`	$\frac{d^n}{dx^n} f$
`\D{f}{x,y,z}`	$\frac{d}{dx}\frac{d}{dy}\frac{d}{dz} f$
`\D[1,2,1]{f}{x,y,z}`	$\frac{d}{dx}\frac{d^2}{dy^2}\frac{d}{dz} f$
`\pderiv{f}{x}`	$\frac{\partial}{\partial x} f$
`\pderiv[n]{f}{x}`	$\frac{\partial^n}{\partial x^n} f$
`\pderiv{f}{x,y,z}`	$\frac{\partial}{\partial x}\frac{\partial}{\partial y}\frac{\partial}{\partial z} f$
`\pderiv[1,2,1]{f}{x,y,z}`	$\frac{\partial}{\partial x}\frac{\partial^2}{\partial y^2}\frac{\partial}{\partial z} f$

```
\newcommand{\D}[3][1]{\COOL@derivative{#1}{#2}{#3}{d}}
\newcommand{\pderiv}[3][1]{\COOL@derivative{#1}{#2}{#3}{\partial}}
```

`\Integrate` Integrate

`\Int` This has the option `IntegrateDisplayFunc` which can be `inset` or `outset`:

`\Style{IntegrateDisplayFunc=inset}` (Default)

`\Integrate{f}{x}`	$\int f\,dx$
`\Int{f}{x}`	$\int f\,dx$
`\Integrate{f}{x,A}`	$\int_A f\,dx$
`\Int{f}{x,A}`	$\int_A f\,dx$
`\Integrate{f}{x,a,b}`	$\int_a^b f\,dx$
`\Int{f}{x,a,b}`	$\int_a^b f\,dx$

`\Style{IntegrateDisplayFunc=outset}`

`\Integrate{f}{x}`	$\int dx\, f$
`\Int{f}{x}`	$\int dx\, f$
`\Integrate{f}{x,A}`	$\int_A dx\, f$
`\Int{f}{x,A}`	$\int_A dx\, f$
`\Integrate{f}{x,a,b}`	$\int_a^b dx\, f$
`\Int{f}{x,a,b}`	$\int_a^b dx\, f$

```
\newcommand{\COOL@notation@IntegrateDisplayFunc}{inset}
\newcommand{\Integrate}[2]{%
\listval{#2}{0}%
```

record the length of the list

```
\setcounter{COOL@listlen}{\value{COOL@listpointer}}%
\ifthenelse{ \value{COOL@listlen} = 1 }%
{%
\ifthenelse{\equal{\COOL@notation@IntegrateDisplayFunc}{outset}}%
{%
```

```
\int \! d#2 \, #1%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@IntegrateDisplayFunc}{inset}}%
{%
\int #1 \, d#2%
}%
% Else
{%
\PackageError{cool}{Invalid Option Sent}%
{‘DisplayFunc’ can only be ‘inset’ or ‘outset’}%
}}%
}%
% ElseIf
{ \ifthenelse{ \value{COOL@listlen} = 2 }%
{%
\ifthenelse{\equal{\COOL@notation@IntegrateDisplayFunc}{outset}}%
{%
\int_{\listval{#2}{2}} \! d{\listval{#2}{1}} \, #1%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@IntegrateDisplayFunc}{inset}}%
{%
\int_{\listval{#2}{2}} #1 \, d{\listval{#2}{1}}%
}%
% Else
{%
\PackageError{cool}{Invalid Option Sent}%
{‘DisplayFunc’ can only be ‘inset’ or ‘outset’}%
}}%
}%
% ElseIf
{ \ifthenelse{ \value{COOL@listlen} = 3 }%
{%
\ifthenelse{\equal{\COOL@notation@IntegrateDisplayFunc}{outset}}%
{%
\int_{\listval{#2}{2}}^{\listval{#2}{3}} \! d{\listval{#2}{1}} \, #1%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@IntegrateDisplayFunc}{inset}}%
{%
\int_{\listval{#2}{2}}^{\listval{#2}{3}} #1 \, d{\listval{#2}{1}}%
}%
% Else
{%
\PackageError{cool}{Invalid Option Sent}%
{‘DisplayFunc’ can only be ‘inset’ or ‘outset’}%
}}%
}%
% Else
```

```
{%
\PackageError{cool}{‘Integrate’ has invalid parameter list}%
   {this happens when the second argument has more than two commas}%
}}}%
}%
\newcommand{\Int}[2]{\Integrate{#1}{#2}}
```

\Sum Sum

```
\Sum{a_n}{n}        \sum_n a_n
\Sum{a_n}{n,1,N}    \sum_{n=1}^N a_n
```

```
\newcommand{\Sum}[2]{%
\listval{#2}{0}%
```

record the length of the list

```
\setcounter{COOL@listlen}{\value{COOL@listpointer}}
\ifthenelse{ \value{COOL@listlen} = 1 }%
{%
\sum_{#2} #1%
}%
% else
{%
\ifthenelse{ \value{COOL@listlen} = 3 }%
{%
\sum_{ \listval{#2}{1} = \listval{#2}{2} }^{ \listval{#2}{3} } #1
}%
% else
{%
\PackageError{cool}{Invalid list length for ‘Sum’}%
{can only have none or two commas for second argument}%
}%
}%
}
```

\Prod Product

```
\Prod{a_n}{n}        \prod_n a_n
\Prod{a_n}{n,1,N}    \prod_{n=1}^N a_n
```

```
\newcommand{\Prod}[2]{%
\listval{#2}{0}%
```

record the length of the list

```
\setcounter{COOL@listlen}{\value{COOL@listpointer}}
\ifthenelse{ \value{COOL@listlen} = 1 }%
{%
\prod_{#2} #1%
}%
% else
{%
\ifthenelse{ \value{COOL@listlen} = 3 }%
{%
```

```
\prod_{ \listval{#2}{1} = \listval{#2}{2} }^{ \listval{#2}{3} } #1
}%
% else
{%
\PackageError{cool}{Invalid list length for 'Prod'}%
{can only have none or two commas for second argument}%
}%
}%
}
```

1.3.43 Vector Operators

`\DotProduct` The dot product, `\DotProduct{\vec{A}}{\vec{B}}`, $\vec{A} \cdot \vec{B}$

```
\newcommand{\DotProduct}[2]{#1 \cdot #2}
```

`\Cross` The cross product, `\Cross{\vec{A}}{\vec{B}}`, $\vec{A} \times \vec{B}$

```
\newcommand{\Cross}[2]{#1 \times #2}
```

`\Div` the divergence, `\Div{\vec{A}}`, $\nabla \cdot \vec{A}$

```
\newcommand{\Div}[1]{\nabla \cdot #1}
```

`\Grad` The gradient, `\Grad{f}`, ∇f

```
\newcommand{\Grad}[1]{\nabla #1}
```

`\Curl` The curl, `\Curl{\vec{A}}`, $\nabla \times \vec{A}$

```
\newcommand{\Curl}[1]{\nabla \times #1}
```

`\Laplacian` The laplacian, `\Laplacian{f}`, $\nabla^2 f$

```
\newcommand{\Laplacian}[1]{\nabla^2 #1}
```

1.3.44 Matrix Operations

`\Transpose` Transpose of a matrix, `\Transpose{A}`, A^T

```
\newcommand{\COOL@notation@TransposeParen}{inv}
\newcommand{\Transpose}[1]{ \COOL@decide@paren{Transpose}{#1}^T }
```

`\Dagger` Conjugate Transpose of a matrix, `\Dagger{A}`, $A^\dagger$

```
\newcommand{\COOL@notation@DaggerParen}{inv}
\newcommand{\Dagger}[1]{ \COOL@decide@paren{Dagger}{#1}^\dagger }
```

`\Det` determinant of a matrix

`\Style{DetDisplay=det}` (Default)	
`\Det{A}`	$\det A$
`\Style{DetDisplay=barenc}`	
`\Det{A}`	$\|A\|$

```
\newcommand{\COOL@notation@DetParen}{none}
\newcommand{\COOL@notation@DetDisplay}{det}
\newcommand{\Det}[1]{%
```

```
\ifthenelse{\equal{\COOL@notation@DetDisplay}{det}}%
{%
\det\COOL@decide@paren{Det}{#1}%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@DetDisplay}{barenc}}%
{%
\left|#1\right|%
}%
% Else
{%
\PackageError{cool}{Invalid Option Sent}%
{‘DetDisplay’ can only be ‘det’ or ‘barenc’}%
}}%
}
```

`\Tr` Trace of a Matrix, `\Tr{A}`, $\operatorname{Tr} A$

```
\newcommand{\COOL@notation@TrParen}{none}
\newcommand{\Tr}[2][]{%
\ifthenelse{\equal{#1}{}}
{%
\operatorname{Tr}\COOL@decide@paren{Tr}{#2}%
}%
% Else
{%
\operatorname{Tr}_{#1}\COOL@decide@paren{Tr}{#2}%
}%
}
```

1.3.45 Matricies

`\IdentityMatrix` The Identity Matrix

`\IdentityMatrix` $\mathbb{I}$

`\IdentityMatrix[2]` $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

```
\newcommand{\COOL@notation@IdentityMatrixParen}{p}
\newcounter{COOL@row}%
\newcounter{COOL@col}%
\newcommand{\IdentityMatrix}[1][0]{%
\isint{#1}%
\ifthenelse{\boolean{COOL@isint}}%
{%
\ifthenelse{ #1=0 }%
{%
\mathbb{I}%
}%
% Else
{%
\setcounter{COOL@ct}{\value{MaxMatrixCols}}%
```

```
\setcounter{MaxMatrixCols}{#1}%
\ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{p}}%
{%
\begin{pmatrix}%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{b}}%
{%
\begin{bmatrix}%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{br}}%
{%
\begin{Bmatrix}%
}%
% Else
{%
\begin{matrix}%
}}}%
\forLoop{1}{#1}{COOL@row}%
{%
\ifthenelse{\NOT \value{COOL@row} = 1}{\\}{}%
\forLoop{1}{#1}{COOL@col}%
{%
\ifthenelse{ \NOT \value{COOL@col} = 1 }{&}{}%
\ifthenelse{ \value{COOL@row}=\value{COOL@col} }{1}{0}%
}%
}%
\ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{p}}%
{%
\end{pmatrix}%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{b}}%
{%
\end{bmatrix}%
}%
% ElseIf
{ \ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{br}}%
{%
\end{Bmatrix}%
}%
% Else
{%
\end{matrix}%
}}}%
\setcounter{MaxMatrixCols}{\value{COOL@ct}}%
}%
}%
% Else
```

```
{%
\mathbb{I}%
}%
}%
```

Change History

v0

General: pre-Initial version [tentative edition] 1

v1.0

General: Initial Release 1

v1.1

General: Added listlenstore to package to allow storing of the list length 1

v1.2

General: Split off the list, string, and forloop parts to separate packages 1

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Symbols

\{ 7, 1682, 1696, 1719, 1738, 2121, 2133
\} 7, 1682, 1696, 1722, 1745, 2129, 2139

A

\Abs 2038
\AGM 1902
\AiryAi 327
\AiryAiSymb 328, 329
\AiryBi 330
\AiryBiSymb 331, 332
\AppellFOne 1189
\ArcCos 218
\ArcCosh 276
\ArcCot 258
\ArcCoth 284
\ArcCsc 254
\ArcCsch 280
\ArcSec 256
\ArcSech 282
\ArcSin 200
\ArcSinh 274
\ArcTan 236
\ArcTanh 278
\Arg 2039
\ArithGeoMean 1902
\ArithGeoMeanSymb 1903, 1905
\AssocLegendreP 659
\AssocLegendreQ 660
\AssocWeierstrassSigma 1656

B

\Bernoulli 431, 680
\BernoulliB 680
\BesselI 319
\BesselJ 311
\BesselK 323
\BesselY 315
\Beta 820, 847, 848, 908
\BetaReg 849
\BetaRegInv 876
\BetaRegularized 849
\Binomial 683

C

\CarmichaelLambda 2114
\Catalan 115
\Ceiling 338
\ChebyshevT 645
\ChebyshevU 649
\CInfty 128

\ClebschGordon 1328
\ComplexInfinity 128
\Conj 2040
\Conjugate 2040
\COOL@arg@temp 466, 473, 474, 478, 479, 484, 485, 488, 495, 501, 502, 506, 507, 510
\COOL@Bernoulli@arg@i 442
\COOL@Bernoulli@arg@ii 443
\COOL@decide@indicies 68
\COOL@decide@indicies@placement 69, 71, 79, 85
\COOL@decide@paren 13
\COOL@decide@paren@no@type . . . 13–15
\COOL@decide@paren@type 17, 21, 23, 29, 35, 41, 47, 53
\COOL@deriv@powers@i 2160
\COOL@derivative . . . 2151, 2591, 2592
\COOL@hideOnSF 1700, 1719–1721, 1739, 1741, 1743
\COOL@Hypergeometric@pq@ab@value 987
\COOL@keyeater 98, 101, 106
\COOL@keyend 98, 101, 106
\COOL@keystop 98, 100, 102
\COOL@MeijerG@anp@value 1195, 1292, 1303, 1305, 1322
\COOL@notation@ArcTrig 199
\COOL@notation@DDisplayFunc . . . 2148
\COOL@notation@DShorten 2148
\COOL@tempd 2165, 2167
\Cos 189
\Cosh 262
\CoshInt 984
\CoshIntSymb 985, 986
\CosInt 978
\CosIntSymb 979, 980
\Cot 197
\Coth 272
\Cross 2698
\Csc 193
\Csch 266
\Curl 2701
\CyclotomicC 675

D

\D 2591
\Dagger 2705
\DblFactorial 682
\DedekindEta 1888
\Det 2707
\DiGamma 797
\DigitCount 2117
\DiLog 2016
\DiracDelta 2142
\DirectedInfinity 126
\DirInfty 126
\DiscreteDelta 458
\Div 2699
\Divisors 2096
\DivisorSigma 2106
\DivisorsSymb 2097, 2099
\DotProduct 2697

E

\E 111, 138, 142
\EExp 1907
\EGCD 381
\EllipticE 1475, 1544
\EllipticExp 1907
\EllipticExpSymb 1908, 1920
\EllipticF 1526
\EllipticK 1471
\EllipticLog 1924
\EllipticLogSymb 1925, 1947
\EllipticNomeQ 1896
\EllipticNomeQInv 1896
\EllipticPi 1498, 1545
\EllipticTheta 1552
\ELog 1924
\Erf 910, 951
\Erfc 953
\ErfcInv 957
\ErfcSymb 954, 955, 958
\Erfi 959
\ErfInv 931, 952
\ErfiSymb 960, 961
\ErfSymb 911, 917, 922, 937, 942
\Euler 410, 679
\EulerE 679
\EulerGamma 114
\EulerPhi 2109
\Exp 130
\ExpIntE 966
\ExpIntEi 968
\ExpIntEiSymb 969, 971
\ExtendedGCD 381
\ExtendedGCDSymb 382, 384

F

\Factorial 681

\FactorInteger 2091
\FactorIntegerSymb 2092, 2094
\Factors 2091
\Fibonacci 389, 678
\FibonacciF 678
\Floor 337
\fPart 344
\fPartSymb 345, 346
\FractionalPart 344
\FresnelC 964
\FresnelS 962

G

\GammaFunc 721, 745, 746
\GammaReg 747
\GammaRegInv 769
\GammaRegularized 747, 791
\GCD 379
\GegenbauerC 661
\GenErf 951
\GenErfInv 951
\GenIncBeta 848
\GenIncGamma 746
\GenRegIncBeta 908
\GenRegIncBetaInv 909
\GenRegIncGamma 791
\GenRegIncGammaInv 792
\Glaisher 116
\GoldenRatio 113
\Grad 2700

H

\HarmNum 802
\HeavisideStep 2144
\HermiteH 513
\HurwitzZeta 1953
\Hypergeometric 1051
\HypergeometricU 1192

I

\I 110
\IdentityMatrix 2736
\Imag 2077
\ImagSymb 2078, 2082, 2086
\IncBeta 847
\IncEllipticE 1544
\IncEllipticF 1526
\IncEllipticPi 1545
\IncGamma 745
\Indeterminant 119
\Infinity 118
\Int 2593
\IntegerPart 340
\Integrate 2593
\InverseBetaRegularized . . . 876, 909
\InverseGammaRegularized . . . 769, 792
\iPart 340
\iPartSymb 341, 342

J

\JacobiAmplitude 1750
\JacobiAmplitudeSymb 1751, 1753
\JacobiCD 1756
\JacobiCDInv 1756
\JacobiCDSymb 1758, 1760, 1764
\JacobiCN 1767
\JacobiCNInv 1767
\JacobiCNSymb 1769, 1771, 1775
\JacobiCS 1778
\JacobiCSInv 1778
\JacobiCSSymb 1780, 1782, 1786
\JacobiDC 1789
\JacobiDCInv 1789
\JacobiDCSymb 1791, 1793, 1797
\JacobiDN 1800
\JacobiDNInv 1800
\JacobiDNSymb 1802, 1804, 1808
\JacobiDS 1811
\JacobiDSInv 1811
\JacobiDSSymb 1813, 1815, 1819
\JacobiNC 1822
\JacobiNCInv 1822
\JacobiNCSymb 1824, 1826, 1830
\JacobiND 1833
\JacobiNDInv 1840
\JacobiNDinv 1833
\JacobiNDSymb 1835, 1837, 1841
\JacobiNS 1844
\JacobiNSInv 1844
\JacobiNSSymb 1846, 1848, 1852
\JacobiP 653
\JacobiSC 1855
\JacobiSCInv 1855
\JacobiSCSymb 1857, 1859, 1863
\JacobiSD 1866
\JacobiSDInv 1866
\JacobiSDSymb 1868, 1870, 1874
\JacobiSN 1877
\JacobiSNInv 1877
\JacobiSNSymb 1879, 1881, 1885

\JacobiSymbol 2113
\JacobiTheta 1552
\JacobiZeta 1546

K

\Khinchin 117
\KleinInvariantJ 1890
\KroneckerDelta 461

L

\LambertW 286
\Laplacian 2702
\LaugerreL 517
\LCM 386
\LCMSymb 387, 388
\LegendreP 539, 659
\LegendreQ 592, 660
\LerchPhi 1983
\LeviCivita 490
\Log 135
\LogGamma 794
\LogGammaSymb 795, 796
\LogInt 972
\LogIntSymb 973, 974

M

\MathieuC 2017
\MathieuCharacteristicA 2025
\MathieuCharacteristicB 2029
\MathieuCharacteristicExponent 2033
\MathieuCharisticA 2025
\MathieuCharisticB 2029
\MathieuCharisticExp 2033
\MathieuCSymb 2018, 2020
\MathieuS 2021
\MathieuSSymb 2022, 2024
\Max 307
\MeijerG 1233
\Min 309
\Mod 348
\ModularLambda 1893
\MoebiusMu 2111
\Multinomial 684

N

\NevilleThetaC 1556
\NevilleThetaD 1561
\NevilleThetaN 1566
\NevilleThetaS 1571
\NielsenPolyLog 1985, 1997

P

\PartitionsP 454
\PartitionsQ 456
\pderiv 2591
\PI 112
\Pochhammer 793
\PolyGamma 799
\PolyLog 1990, 2016
\PolyLogSymb 1991, 2004
\Prime 2100
\PrimePi 2104
\PrimeSymb 2101, 2103
\Prod 2677
\ProductLog 286, 287

Q

\Quotient 375
\QuotientSymb 376, 378

R

\Real 2065
\RealSymb 2066, 2070, 2074
\RegHypergeometric 1118
\RegIncBeta 849
\RegIncBetaInv 876
\RegIncGamma 747
\RegIncGammaInv 769
\RiemannSiegelTheta 1976
\RiemannSiegelZ 1979
\RiemannZeta 1952
\Round 339

S

\Sec 195
\Sech 269
\Sign 2089
\Signature 490
\Sin 187
\Sinh 260
\SinhInt 981
\SinhIntSymb 982, 983
\SinInt 975
\SinIntSymb 976, 977
\SixJSymbol 1420
\SpHarmY 667
\SphericalHarmonicY 667
\SphericalHarmY 667
\StieltjesGamma 1982
\StirlingSOne 452
\StirlingSTwo 453

\StruveH 333
\StruveL 335
\Style 97
\Sum 2657

T

\Tan 191
\Tanh 264
\ThreeJSymbol 1361
\Tr 2725
\Transpose 2703

U

\UnitStep 2144
\UseStyleFile 109

W

\WeierstrassHalfPeriods 1672
\WeierstrassInvariants 1686
\WeierstrassP 1576
\WeierstrassPGenInv 1624
\WeierstrassPHalfPeriodValues . 1707
\WeierstrassPInv 1592, 1624
\WeierstrassSigma 1625, 1656
\WeierstrassZeta 1657
\WeierstrassZetaHalfPeriodValues 1726
\WeiHalfPeriods 1672
\WeiInvars 1699
\WeiP 1576
\WeiPHalfPeriodVal 1707
\WeiPInv 1592
\WeiSigma 1625
\WeiZeta 1657
\WeiZetaHalfPeriodVal 1726

Z

\Zeta 1952, 1953, 1954