SymPy v1.8 Cheat Sheet

```
import sympy as sy
```

Create Symbols

define symbols:
```
> a, b = sy.symbols('a b')
```
define a range of symbols:
```
> a, b, c, d, e = sy.symbols('a:e')
```
include Greek symbols:
```
> alpha = sy.symbols(r'\alpha')
```
include subscripts:
```
> a1 = sy.symbols('a_1')
```
define a range of subscripted symbols:
```
> a1, a2, a3 = sy.symbols('a_(1:4)')
```
define symbols using assumptions:
```
> a = sy.symbols('a', [key]=True/False)
```
where [key] can be: even, odd, integer, rational, real, imaginary, complex, prime, positive, negative, nonpositive, nonnegative, commutative, ...

Mathematical Constants

return $\pi \approx 3.14159$:
```
> sy.pi
```
return Euler's number $\mathrm{e} \approx 2.71828$:
```
> sy.E
```
return the imaginary unit $\mathrm{i}^2 = -1$:
```
> sy.I
```
return infinity ∞:
```
> sy.oo
```

Mathematical Functions

square root $\sqrt{x}$:
```
> sy.sqrt(x)
```
absolute value $|x|$:
```
> sy.abs(x)
```
return the sign of a number $\mathrm{sgn}(x)$:
```
> sy.sign(x)
```
trigonometric functions (sin, cos, tan, cot, ...):
```
> sy.sin(x)
```
inverse trigonometric functions:
```
> sy.asin(x)
```
hyperbolic functions:
```
> sy.sinh(x)
```
area hyperbolic functions:
```
> sy.asinh(x)
```
inverse tangent with correct quadrant:
```
> sy.atan2(y, x)
```
exponential function e^x:
```
> sy.exp(x)
```
natural logarithm $\ln(x)$:
```
> sy.log(x)
```
base-b logarithm $\log_b(x)$:
```
> sy.log(x, b)
```

Algebra

return the greatest common divisor:
```
> sy.gcd(x, y)
```
return the least common multiple:
```
> sy.lcm(x, y)
```
return the real/imaginary part of x:
```
> sy.re(x)
> sy.im(x)
```
perform a polynomial division:
```
> sy.div(x**2 - 4 + x, x-2)
```

Solve Equations

solve $f(x) = 0$:
```
> sy.solve(f, x)
```
solve system of equ's $f(x, y) = 0$, $g(x, y) = 0$:
```
> sy.solve([f, g], [x, y])
```
solve differential equation:
```
> f = sy.Function('f')
> sy.dsolve(sy.diff(f(x), x) - x, f(x))
```

Linear Algebra: Vectors

create a vector via its components v_i:
```
> sy.Matrix([1, 2, 3])
```
inner dot product of two vectors $\boldsymbol{v} \cdot \boldsymbol{w}$:
```
> v.dot(w)
```
cross product of two 3-vectors $\boldsymbol{v} \times \boldsymbol{w}$:
```
> v.cross(w)
```
return the norm of a vector $|\boldsymbol{v}| = \sqrt{\boldsymbol{v} \cdot \boldsymbol{v}}$:
```
> v.norm()
```
return the normalized vector $\hat{\boldsymbol{v}} = \boldsymbol{v}/|\boldsymbol{v}|$:
```
> v.normalized()
```

Linear Algebra: Create Matrices

$n \times n$ identity matrix $\mathbb{1}_n$:
```
> sy.eye(n)
```
$m \times n$ empty matrix, $M_{ij} = 0\ \forall i, j$:
```
> sy.zeros(m, n)
```
$m \times n$ matrix filled with 1, $M_{ij} = 1\ \forall i, j$:
```
> sy.ones(m, n)
```
define a diagonal matrix via its entries:
```
> sy.diag(1, 2, 3)
```
define a matrix via its entries M_{ij}:
```
> sy.Matrix([[1, 2],
.           [3, 4]])
```
...via a lambda function, $M_{ij} = 2i + j$:
```
> sy.Matrix(m, n, lambda i,j: 2*i + j)
```
...via a dyadic product $M_{ij} = v_i w_j$:
```
> sy.Matrix(m, n, lambda i,j: v[i]*w[j])
```

Linear Algebra: Matrix Properties

return the n-th row/column of a matrix M:
```
> M.row(n) # n = 0, 1, ...
> M.col(n)
```
return the shape (i.e. $m \times n$) of a matrix M:
```
> M.shape
```
return the rank of a matrix M:
```
> M.rank()
```
return the trace of a matrix $\mathrm{Tr}\{M\}$:
```
> M.trace()
```
return the determinant of a matrix $\det\{M\}$:
```
> M.det()
```

Linear Algebra: Manipulate Matrices

return the matrix inverse M^{-1}:
```
> M.inv()
```
return the matrix transpose M^T:
```
> M.T
```
return the complex conjugate all entries M^*:
```
> M.C
```
return the Hermitian conjugate $M^\dagger = (M^T)^*$:
```
> M.H
```
delete the n-th row/column (nothing returned):
```
> M.row_del(n) # n = 0, 1, ...
> M.col_del(n)
```

Linear Algebra: Matrices and Vectors

return the matrix-vector product $M\boldsymbol{v}$:
```
> M * v
```
return the matrix-matrix product MN:
```
> M * N
```
diagonalize M such that $D = P^{-1}MP$:
```
> P, D = M.diagonalize()
```
return eigenvalues as a dict with multiplicities:
```
> M.eigenvals()
```
return eigenvalues as a list:
```
> M.eigenvals(multiple=True)
```
return eigenvalues, multiplicities, eigenvectors:
```
> M.eigenvects()
```

Calculus: Derivatives

take the derivative of f with respect to x:
```
> sy.diff(f, x)
```
take the n-th derivative of f with respect to x:
```
> sy.diff(f, x, n)
```
take the derivative of f with respect to x and y:
```
> sy.diff(f, x, y)
```

Calculus: Integrals

integrate f with respect to x:
```
> sy.integrate(f, x)
```
integrate f with respect to x from a to b:
```
> sy.integrate(f, (x, a, b))
```

Limits

take the limit of f where x goes to a:
```
> sy.limit(f, x, a)
```
take the limit of f where x goes to a_+:
```
> sy.limit(f, x, a, dir='+')
```

Taylor Series

expand $f(x)$ around a up to $\mathcal{O}(n)$:
```
> f.series(x, a, n)
```
...approaching the number from above:
```
> f.series(x, a, n, dir='+')
```
...and remove the $\mathcal{O}(n)$:
```
> f.series(x, a, n).removeO()
```

Discrete Mathematics

perform discrete sum $\sum_{n=a}^{b} f$:
```
> sy.summation(f, (n, a, b))
```
perform product $\prod_{n=a}^{b} f$:
```
> sy.product(f, (n, a, b))
```
return the factorial $n!$:
```
> sy.factorial(n)
```
return the binomial coefficient $\binom{n}{k}$:
```
> sy.binomial(n, k)
```
return the i-th prime:
```
> sy.prime(i)
```
return the next prime greater than n:
```
> sy.nextprime(n)
```
return the Kronecker delta δ_{ij}:
```
> sy.KroneckerDelta(i, j)
```
return the Levi–Civita symbol ϵ_{ijk}:
```
> sy.LeviCivita(i, j, k)
```

Lambdify

create a numerical function $f(x, y, z) = x + yz$:
```
> f = sy.lambdify([x, y, z], x + y*z)
```

Miscellaneous

get help:
```
> help(sy.asinh)
```
simplify an expression f:
```
> sy.simplify(f)
```
substitute x for a in f:
```
> f.subs(x, a)
```
define fraction $\frac{p}{q}$ analytically:
```
> sy.Rational(p, q)
```
test for equality $a = b$ at random points:
```
> a.equals(b)
```
force numerical evaluation of f:
```
> f.n()
```
...and set very small numbers to zero:
```
> f.n(chop=True)
```
...and round to d digits:
```
> f.n(d)
```