# Cheat Sheets for SymPy

The following page contains a cheat sheet for **SymPy**, the free and open source Computer Algebra System. Considering how big of a project SymPy is, only the most useful commands are included.

Cheat sheet are primarily intended to help new or occasional users. These come from my book, **Symbolic Computation with Python and SymPy**, which provides the fastest way to understand how to get the best out of SymPy thanks to in-depth guided exercises. If you are new to SymPy, please consider purchasing this book.

Table of content:



## Basic SymPy

| SYMBOLS |
|---|
| `x = Symbol("x", *asmpt)` |
| `x, y, z = symbols("x:z", *asmpt)` |
| `var("x, y", *asmpt)` |
| `from abc import a, A, alpha, …` |
| `d = Dummy(*asmpt)` |
| `w = Wild("w", exclude=[], properties=[])` |
| `d1, d2 = symbols("d1, d2", *asmpt, cls=Dummy)` |

| EXPRESSIONS |
|---|
| `Add(*args, evaluate=True)` |
| `Mul(*args, evaluate=True)` |
| `Pow(base, exponent, evaluate=True)` |

| COMPARISON | |
|---|---|
| Structural | `expr1 == expr2` |
| Mathematical | `simplify(expr1 - expr2 ) == 0` |
| Mathematical | `nsimplify(expr1 - expr2 ) == 0` |
| Mathematical | `expr1.equals(expr2)` |

| DERIVATIVES |
|---|
| $\frac{df}{dx}$ |
| `diff(f, x)` |

| FUNCTIONS | |
|---|---|
| `x, y = symbols("x, y", *asmpt)` | |
| `f = Function("f", *asmpt)` | |
| `f(x, y)` | |
| `g, h = symbols("g, h", *asmpt, cls=Function)` | |
| Absolute Value | `Abs(x)` |
| Square Root | `sqrt(x)` |
| N-th Root | `root(x, n)` |
| Exponential | `exp(x)` |
| Logarithm | `log(x), log(x, b)`<br>`ln(x)` |
| Minimum | `Min(*args)` |
| Maximum | `Max(*args)` |
| Trigonometric | `sin, cos, tan,`<br>`asin, acos, atan` |
| Hyperbolic | `sinh, cosh, tanh`<br>`asinh, acosh, atanh` |
| Complexes | `re, im, sign, abs, arg` |
| `l = Lambda((x, y), expr)` | |
| `w = WildFunction("w", *asmpt, nargs=)` | |
| `p = Piecewise((expr1, cond1), ...)` | |
| `piecewise_fold(expr)` | |

| LIMITS |
|---|

| NUMBERS | |
|---|---|
| Integers, $\mathbb{Z}$ | `Integer(n)` |
| Reals, $\mathbb{R}$ | `Float(n)` |
| Rationals, $\mathbb{Q}$ | `Rational(p, q)`<br>`(p) / q`<br>`p / S(q)` |
| Not a Number | `nan` |
| Infinity, $\infty$ | `oo` |
| Complex Infinity, $\tilde{\infty}$ | `zoo` |
| Euler's number, $e$ | `E` |
| Imaginary unit, $i$ | `I` |
| $\pi$ | `pi` |
| Complex Numbers, $\mathbb{C}$ | `n1 + I * n2` |

| WALKING EXPR TREE |
|---|
| `preorder_traversal(expr)` |
| `postorder_traversal(expr)` |

| INTEGRALS |
|---|
| $\int f(x)dx$ |
| `integrate(f, x)` |
| `f.integrate(x)` |
| `Integral(f, x).doit()` |
| $\int_c^d \int_a^b f(x,y)dxdy$ |
| `integrate(f, (x, a, b), (y, c, d))` |
| `f.integrate((x, a, b), (y, c, d))` |

`f.diff(x)`

`Derivative(f, x).doit()`

$$\frac{\partial^3 f}{\partial^2 x \partial y}$$

`diff(f, x, 2, y)`

`f.diff(x, 2, y)`

`Derivative(f, x, 2, y).doit()`

**SERIES**

`r = series(expr, x, x0, n, dir="+|-")`

`r = expr.series(x, x0=0, n=6, dir="+|-")`

`r.removeO()`

$$\lim_{x \to x_0} f(x)$$

`limit(f, x, x0, dir="+|-|+-")`

`f.limit(x, x0, dir="+|-|+-")`

`Limit(f, x, x0, dir="+|-|+-").doit()`

**FOURIER SERIES**

`fs = fourier_series(f, limits=None, finite=True)`

`fs = f.fourier_series(limits=None)`

`fs.scale(s), fs.scalex(s), fs.shift(s), fs.shiftx(s), fs.truncate(n=3) fs.sigma_approximation(n=3)`

`Integral(f, (x, a, b), (y, c, d)).doit()`

Transform and U-substitution:

`Integral.transform(x, u)`

**SOLVERS**

`solve()`

`solveset()`

`nsolve()`

`linsolve()`

`nonlinsolve()`

`dsolve()`

## Expression Manipulation

**SIMPLIFICATION**

`simplify`

`nsimplify`

`radsimp`

`ratsimp`

`trigsimp`

`combsimp`

`powsimp`

`powdenest`

`factor`

`together`

`cancel`

`logcombine`

**EXPANSION**

`expand`

`expand_mul`

`expand_log`

`expand_func`

`expand_trig`

`expand_complex`

`expand_multinomial`

`expand_power_exp`

`expand_power_base`

**SUBSTITUTION**

`expr.subs(old, new, simultaneous=False)`

`expr.xreplace({k_old: v_new})`

`expr.replace(query, value)`

**COLLECTION**

`collect(expr, syms, **kwargs)`

`rcollect(expr, evaluate=None)`

`collect_sqrt(expr, evaluate=True)`

`collect_const(expr, *vars, Numbers=True)`

`logcombine(expr, force=False)`

**SEARCH / FIND**

`expr.find(query, group=False)`

`expr.has(*patterns)`

`expr.match(pattern, old=False)`

**INFORMATION**

`expr.args`

`expr.atoms(*types)`

`expr.free_symbols`

`expr.func`

**OTHERS**

`fraction(expr, exact=False)`

`rewrite(*args, **hints)`

`sympify(obj, *args)`

**USEFUL FUNCTION BASE CLASSES**

```
from sympy.core.function import AppliedUndef
from sympy.functions.elementary.trigonometric import TrigonometricFunction, InverseTrigonometricFunction
from sympy.functions.elementary.hyperbolic import HyperbolicFunction, ReciprocalHyperbolicFunction
from sympy.functions.combinatorial.factorials import CombinatorialFunction
```

## Relationals, Logic Operators, Sets

**RELATIONALS**

| | |
|---|---|
| lhs = rhs | `Eq(lhs, rhs)` |
| lhs != rhs | `Ne(lhs, rhs)` |
| lhs > rhs | `Gt(lhs, rhs)` |
| lhs ≥ rhs | `Ge(lhs, rhs)` |
| lhs < rhs | `Lt(lhs, rhs)` |
| lhs ≤ rhs | `Le(lhs, rhs)` |
| Methods: | `inequality.as_set()` |
| Attributes: | `lhs, rhs, canonical,` |

**LOGICAL OPERATORS**

| | |
|---|---|
| `And(*args)` | $x \wedge y \wedge z$ |
| `Or(*args)` | $x \vee y \vee z$ |
| `Not(*args)` | $\neg x$ |
| `Xor(*args)` | $x \veebar y \veebar z$ |
| `Nand(*args)` | $x \veebar y \veebar z$ |
| `Nor(*args)` | $\neg (x \vee y \vee z)$ |
| `Xnor(*args)` | $\neg (x \veebar y \veebar z)$ |
| `Implies(x, y)` | $x \Rightarrow y$ |

**SETS**

| | |
|---|---|
| `Interval(0, 1, left_open=True)` | $(0, 1]$ |
| `Union(*set_args)` | $[0, 2] \cup [4, 5]$ |
| `Intersection(*set_args)` | $[0, 2] \cap [4, 5]$ |
| `EmptySet` | $\emptyset$ |
| `FiniteSet(*args)` | $\{x, y, z\}$ |

| | |
|---|---|
| | negated, reversed, |
| | reversedsign, lts, gts |

| | |
|---|---|
| Equivalent(x, y) | $x \iff y$ |
| Methods: | logic_expr.as_set() |

| | |
|---|---|
| ConditionSet(sym, condition, base_set=S.UniversalSet) | $\{x \mid x \in \mathbb{R} \land x > 0\}$ |
| Methods: | set.as_relational(sym) |

## Matrix and Linear Algebra

### MATRIX CREATION

| |
|---|
| A = Matrix([[1, 2], [3, 4]]) |
| A = Matrix(nr, nc, listElements) |
| A = Matrix(nr, nc, func) |
| Other classes: |
| ImmutableMatrix, SparseMatrix, ImmutableSparseMatrix |
| zeros(n), zeros(nr, nc) |
| ones(n), ones(nr, nc) |
| eye(n), eye(nr, nc) |
| diag(1, 2, 3), diag(*[1, 2, 3]) |
| hessian(f(x, y), (x, y)) |
| randMatrix(r, c=None, min=0, max=99) |
| A.as_immutable() |
| A.as_mutable() |

### INDEXING

| | |
|---|---|
| Element at (i, j) | A[i, j] |
| i-th row | A[i,:] or A.row(i) |
| i-th column | A[:,i] or A.col(i) |
| Rows from i to j | A[i:j, :] |
| Columns from i to j | A[:, i:j] |
| i-th and j-th row | A[[i:j], :] |
| i-th and j-th column | A[:, [i:j]] |
| Sub-matrix | A[i:j, k:l] |

### COMMON ATTR/MTDS

| |
|---|
| A.atoms(*types) |
| A.expand(**kwargs) |
| A.equals(other, failing_expression=False) |
| A.free_symbols |
| A.has(*patterns) |
| A.replace(F, G) |
| A.simplify(**kwargs) |
| A.subs(*args, **kwargs) |
| A.xreplace(rule) |
| A.is_anti_symmetric() |
| A.is_diagonal() |
| A.is_hermitian |
| A.is_lower |
| A.is_square |
| A.is_symbolic() |
| A.is_symmetric() |
| A.is_upper |
| A.is_zero_matrix |
| A.is_indefinite |
| A.is_negative_definite |

### MATRIX OPERATIONS

| | |
|---|---|
| Addition | A + B |
| Subtraction | A - B |
| Multiplication | A * B \| A * 2 |
| Scalar Division | A / 2 |
| Power (if A is square) | A**n |
| Element-wise Multiplication | |
| matrix_multiply_elementwise(A, B) A.multiply_elementwise(B) | |
| Element-wise operation | A.applyfunc() |
| Inverse | A**-1 or A.inv() |
| Transpose | A.T or A.transpose() |
| Determinant | A.det() |
| Trace | A.trace() |
| Adjoint | A.adjoint() |
| Conjugate | A.conjugate() |

### MATRIX SHAPING

| | |
|---|---|
| Shape | A.shape |
| Number of rows | A.rows |
| Number of columns | A.cols |
| Detele Column | A.col_del(i) |
| Detele Row | A.row_del(i) |
| Insert Column | A.col_insert(i, col) |
| Insert Row | A.row_insert(i, row) |
| Concatenate | A.col_join(col) |
| Concatenate | A.row_join(row) |
| Swap columns | A.col_swap(i, j) |
| Swap rows | A.row_swap(i, j) |
| Reshape Matrix | A.reshape(nrows, ncols) |
| Stack Horizontally | Matrix.hstack( *matrices) |
| Stack Vertically | Matrix.vstack( *matrices) |

### MATRIX CALCULUS

| | |
|---|---|
| A.diff(*args, **kwargs) | |
| A.integrate(*args, **kwargs) | |
| A, X are row or column vectors | A.jacobian(X) |
| A.limit(*args) | |

### SOLVERS – Ax = B

| |
|---|
| A, b = linear_eq_to_matrix([eq1, eq2, eq3], [x, y, z]) |
| linsolve(A*x - B, syms) |
| A.solve(B, method='GJ') |
| A.LDLsolve(B) |
| A.LUsolve(B) |
| A.QRsolve(B) |
| A.cholesky_solve(B) |
| A.gauss_jordan_solve(B) |
| A.pinv_solve(B) |
| A.diagonal_solve(B) |
| A.lower_triangular_solve(B) |
| A.upper_triangular_solve(B) |

### DECOMPOSITION

| |
|---|
| LDLdecomposition() |
| LUdecomposition() |
| LUdecompositionFF() |
| LUdecomposition_Simple() |
| QRdecomposition() |
| cholesky() |
| rank_decomposition() |

### OTHERS

| |
|---|
| A.condition_number() |
| A.copy() |
| A.exp() |
| A.log() |
| A.pinv(method='RD') |
| A.solve_least_squares(rhs, method='CH') |

### ROW/COLUMN VECTORS

| | |
|---|---|
| Cross Product | a.cross(b) |
| Dot Product | a.dot(b) |
| Magnitude | a.norm() |
| Normalized | a.normalized() |
| Projection | a.project(b) |
| Orthogonalize | Matrix.orthogona lize(*vecs, **kwargs) |

### CONVERT MATRIX TO

| | |
|---|---|
| Nested Python list | A.tolist() |
| Python list of non-zero values of A | A.values() |
| Column Matrix | A.vec() |

### TO NUMPY

| | |
|---|---|
| Python list of SymPy expressions to NumPy array | list2numpy(l, dtype=) |

| | |
|---|---|
| A.is_negative_semidefinite | |
| A.is_positive_definite | |
| A.is_positive_semidefinite | |

| ROTATION MATRICES | |
|---|---|
| Rotation about axis 1 | rot_axis1(theta) |
| Rotation about axis 2 | rot_axis2(theta) |
| Rotation about axis 3 | rot_axis3(theta) |

| EIGENVALUES |
|---|
| A.diagonalize() |
| A.eigenvals() |
| A.eigenvects() |
| A.is_diagonalizable() |
| A.jordan_form() |
| A.singular_values() |

| MATRIX SUBSPACES |
|---|
| A.rank() |
| A.columnspace() |
| A.nullspace() |
| A.rowspace() |
| A.orthogonalize(*vecs, **kwargs) |

| | |
|---|---|
| SymPy Matrix to NumPy array | matrix2numpy(m, dtype=) |
| Create ndarray of symbols | symarray(prefix, shape, **kwargs) |

## Matrix Expressions

| MATRIX EXPRESSIONS | |
|---|---|
| Matrix Symbol | A = MatrixSymbol("A", nrows, ncols) |
| Shape | A.shape |
| MatAdd(*args, evaluate=False) | |
| MatMul(*args, evaluate=False) | |
| MatPow(b, e, evaluate=False) | |
| Compare 2 MatrixSymbols | A.equals(B) |

| OPERATIONS | |
|---|---|
| Inverse | A.I, A.inv(), A.inverse(), Inverse(A) |
| Transposition | A.T, A.transpose(), Transpose(A) |
| Trace | Trace(A) |
| Determinant | Determinant(A) |
| Convert to Matrix | A.as_explicit() |

| SPECIAL MATRICES |
|---|
| Identity(n) |
| OneMatrix(nrows, ncols) |
| ZeroMatrix(nrows, ncols) |
| BlockMatrix([A, B], [C, D]) |
| FunctionMatrix(nrows, ncols, lambda) |

## Arrays: sympy.tensor.array

| ARRAY CREATION |
|---|
| A = Array([[1, 2, 3], [4, 5, 6]]) |
| A = Array(matrix) |
| A = Array(list, shape) |
| Other classes: |
| ImmutableDenseNDimArray, MutableSparseNDimArray, ImmutableSparseNDimArray |
| A.as_mutable() |
| A.as_immutable() |

| SHAPING |
|---|
| A.shape |
| A.reshape(nrow, ncols) |

| ARRAY OPERATIONS | |
|---|---|
| Addition | A + B |
| Subtraction | A - B |
| Scalar Multiplication | A * 2 |
| Scalar Division | A / 2 |
| Element-wise operation | A.applyfunc() |
| A.adjoint() | |
| A.conjugate() | |
| A.rank() | |
| A.transpose(), transpose(A) | |
| tensorproduct(A, B) | |
| tensorcontraction(A, *axes) | |

| ARRAY CALCULUS |
|---|
| A.diff(*args, **kwargs) |
| derive_by_array(expr, dx) |

| CONVERT ARRAY TO | |
|---|---|
| Nested Python list | A.tolist() |
| 2D Array to Matrix | A.tomatrix() |

## Vectors: sympy.vector

| COORDSYS3D |
|---|
| C = CoordSys3D(name, transformation='cartesian\|cylindrical\|spherical', parent=None, location=None, rotation_matrix=None, vector_names=None, variable_names=None) |
| C.create_new(name, transformation, variable_names=None, vector_names=None) |
| C.locate_new(name, position, vector_names=None, variable_names=None) |

| VECTOR | |
|---|---|
| Create new | i, j, k = C.base_vectors() <br> v = 2 * i + 3 * j + 4 * k |
| Addition and Subtraction | v1 + v2 and v1 - v2 |
| Scalar Multiplication and Division | v1 * 2 and v1 / 2 |
| Dot | v1.dot(v2), v1 & v2 |

| VECTOR FUNCTIONS |
|---|
| curl(vect, coord_sys=None, doit=True) |
| divergence(vect, coord_sys=None, doit=True) |
| gradient(scalar_field, coord_sys=None, doit=True) |
| is_conservative(field) |
| is_solenoidal(field) |
| scalar_potential(field, coord_sys) |
| scalar_potential_difference(field, coord_sys, point1, point2) |

| |
|---|
| C.orient_new(name, orienters, location=None, vector_names=None, variable_names=None) |
| C.position_wrt(**kwargs) |
| C.rotation_matrix(other) |
| C.scalar_map(other) |
| C.base_scalars() |
| C.base_vectors() |
| C.transformation_from_parent() |
| C.transformation_to_parent() |

**SCALAR FIELD**

| | |
|---|---|
| Create | x, y, z = C.base_vectors() field = x * y + z |

| | |
|---|---|
| Cross | v1.cross(v2), v1 ^ v2 |
| Outer | v1.outer(v2), v1 \| v2 |
| v1.components | |
| v1.magnitude() | |
| v1.normalize() | |
| v1.projection(v2) | |
| v1.separate() | |
| v1.to_matrix(system) | |

**DYADIC**

| | |
|---|---|
| Create | d = v1.outer(v2), d = v1 \| v2 |
| Dot | d.dot(other), d & other |
| Cross | d.cross(other), d ^ other |
| d.components | |
| d.to_matrix(system) | |

| |
|---|
| matrix_to_vector(matrix, system) |
| express(expr, system, system2=None, variables=False) |

**DEL**

| | |
|---|---|
| Create | del = Del(system=None) |
| Dot | del & v, del.dot(v, doit=False) |
| Cross | del ^ v, del.cross(v, doit=False) |
| Gradient | del.gradient(scalar_ field, doit=False) |

**ORIENTERS**

| |
|---|
| AxisOrienter(angle, axis) |
| BodyOrienter(angle1, angle2, angle3, rot_order) |
| SpaceOrienter(angle1, angle2, angle3, rot_order) |
| QuaternionOrienter(q0, q1, q2, q3) |