

Python – Summary

Built-in Functions

```
abs(x) ⇒ num
all(iterable) ⇒ bool
any(iterable) ⇒ bool
ascii(object) ⇒ str
bin(x) ⇒ str
bool([x]) ⇒ class
breakpoint(*args, **kws)
bytearray([source[, encoding[, errors]]) ⇒ class
bytes([source[, encoding[, errors]]) ⇒ class
callable(object) ⇒ bool
chr(i) ⇒ str
@classmethod
compile★
complex([real[, imag]]) ⇒ class
delattr(object, name)
dict(**kwarg) ⇒ class
dict(mapping, **kwarg) ⇒ class
dict(iterable, **kwarg) ⇒ class
dir([object]) ⇒ list
divmod(a, b) ⇒ num
enumerate(iterable, start=0) ⇒ enumerate object
eval(expression[, globals[, locals]]) ⇒ object
exec(object[, globals[, locals]])
filter(function, iterable)
float([x]) ⇒ class
format(value[, format_spec]) ⇒ str
frozenset([iterable]) ⇒ class
getattr(object, name[, default]) ⇒ object
globals() ⇒ dict
hasattr(object, name) ⇒ bool
hash(object) ⇒ int
help([object])
hex(x) ⇒ str
id(object) ⇒ int
input([prompt]) ⇒ str
int([x]) ⇒ class
int(x, base=10) ⇒ class
isinstance(object, classinfo) ⇒ bool
issubclass(class, classinfo) ⇒ bool

iter(object[, sentinel]) ⇒ iterator object
len(s) ⇒ int
list([iterable]) ⇒ class
locals() ⇒ dict
map(function, iterable, ...) ⇒ iterator
max(iterable, *[, key, default])
max(arg1, arg2, *args[, key]) ⇒ object
min(iterable, *[, key, default]) ⇒ object
min(arg1, arg2, *args[, key]) ⇒ object
next(iterator[, default]) ⇒ object
object ⇒ class
oct(x) ⇒ str
open★
ord(c) ⇒ int
pow(base, exp[, mod]) ⇒ num
print★
property★
range(stop) ⇒ class
range(start, stop[, step]) ⇒ class
repr(object) ⇒ str
reversed(seq) ⇒ iterator
round(number[, ndigits]) ⇒ num
set([iterable]) ⇒ class
setattr(object, name, value)
slice(stop) ⇒ class
slice(start, stop[, step]) ⇒ class
sorted(iterable, *, key=None, reverse=False) ⇒ list
@staticmethod
str(object=) ⇒ class
str★
sum(iterable, /, start=0) ⇒ num
super([type[, object-or-type]]) ⇒ proxy object
tuple([iterable]) ⇒ class
type(object) ⇒ class
type(name, bases, dict) ⇒ class
vars([object]) ⇒ __dict__ attr
zip(*iterables) ⇒ iterator
__import__★
```

```
★
__import__(name, globals=None, locals=None, fromlist=(), level=0)
compile(source, filename, mode, flags=0, dont_inherit=False, optimize=-1) ⇒ AST object
print(*objects, sep=',', end='\n', file=sys.stdout, flush=False)
property(fget=None, fset=None, fdel=None, doc=None) ⇒ class
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, ...) ⇒ file object
str(object=b, encoding='utf-8', errors='strict') ⇒ class
```

Dict Methods

```
dict.clear()
dict.copy() ⇒ dict
dict.fromkeys(iterable[, value]) ⇒ dict
dict.get(key[, default]) ⇒ object
dict.items() ⇒ dict_items
dict.keys() ⇒ dict_keys
dict.pop(key[, default]) ⇒ object
dict.popitem() ⇒ tuple
dict.setdefault(key[, default]) ⇒ object
dict.update([other])
dict.values() ⇒ dict_values
```

List Methods

<code>list.append(x)</code>	<code>list.extend(iterable)</code>	<code>list.remove(x)</code>
<code>list.clear()</code>	<code>list.index(x[, start[, end]])</code> ⇒ int	<code>list.reverse()</code>
<code>list.copy()</code> ⇒ list	<code>list.insert(i, x)</code>	<code>list.sort(key=None, reverse=False)</code>
<code>list.count(x)</code> ⇒ int	<code>list.pop([i])</code> ⇒ object	

String Methods

Testing

<code>str.endswith*</code>	<code>str.isdigit()</code> ⇒ bool	<code>str.isspace()</code> ⇒ bool
<code>str.isalnum()</code> ⇒ bool	<code>str.isidentifier()</code> ⇒ bool	<code>str.istitle()</code> ⇒ bool
<code>str.isalpha()</code> ⇒ bool	<code>str.islower()</code> ⇒ bool	<code>str.isupper()</code> ⇒ bool
<code>str.isascii()</code> ⇒ bool	<code>str.isnumeric()</code> ⇒ bool	<code>str.startswith*</code>
<code>str.isdecimal()</code> ⇒ bool	<code>str.isprintable()</code> ⇒ bool	

★

`str.endswith(suffix[, start[, end]])` ⇒ bool
`str.startswith(prefix[, start[, end]])` ⇒ bool

Formatting

<code>str.center(width[, fillchar])</code> ⇒ str	<code>str.rjust(width[, fillchar])</code> ⇒ str
<code>str.format(*args, **kwargs)</code> ⇒ str	<code>str.ljust(width[, fillchar])</code> ⇒ str
<code>str.format_map(mapping)</code> ⇒ str	<code>str.zfill(width)</code> ⇒ str

Transformation

<code>str.encode*</code>	<code>str.replace(old, new[, count])</code> ⇒ str
<code>str.expandtabs(tabsize=8)</code> ⇒ str	<code>str.rstrip([chars])</code> ⇒ str
<code>str.lstrip([chars])</code> ⇒ str	<code>str.strip([chars])</code> ⇒ str

★

`str.encode(encoding="utf-8", errors="strict")` ⇒ bytes object

Upper and Lower Cases

<code>str.capitalize()</code> ⇒ str	<code>str.lower()</code> ⇒ str	<code>str.title()</code> ⇒ str
<code>str.casefold()</code> ⇒ str	<code>str.swapcase()</code> ⇒ str	<code>str.upper()</code> ⇒ str

Split & Combine

<code>str.join(iterable)</code> ⇒ str	<code>str.split(sep=None, maxsplit=-1)</code> ⇒ str
<code>str.partition(sep)</code> ⇒ str	<code>str.rsplit(sep=None, maxsplit=-1)</code> ⇒ str
<code>str.rpartition(sep)</code> ⇒ str	<code>str.splitlines([keepends])</code> ⇒ str

Searching

<code>str.count(sub[, start[, end]])</code> ⇒ int	<code>str.rfind(sub[, start[, end]])</code> ⇒ int
<code>str.find(sub[, start[, end]])</code> ⇒ int	<code>str.rindex(sub[, start[, end]])</code> ⇒ int
<code>str.index(sub[, start[, end]])</code> ⇒ int	

String Formatting

<code>format_spec</code>	<code>[[fill]align][sign][#][0][width][grouping_option][.precision][type]</code>
<code>fill</code>	<any character>
<code>align</code>	"<" ">" "=" "^"
<code>sign</code>	"+" "-" " "
<code>width</code>	digit+
<code>grouping_option</code>	"_" ","
<code>precision</code>	digit+
<code>type</code>	"b" "c" "d" "e" "E" "f" "F" "g" "G" "n" "o" "s" "x" "X" "%"

Set Methods

<code>set.add(elem)</code>	<code>set.intersection(*others) ⇒ set</code>	<code>set.remove(elem)</code>
<code>set.clear()</code>	<code>set.intersection_update(*others)</code>	<code>set.symmetric_difference(other) ⇒ set</code>
<code>set.copy() ⇒ set</code>	<code>set.isdisjoint(bool)</code>	<code>set.symmetric_difference_update(other)</code>
<code>set.difference(*others) ⇒ set</code>	<code>set.issubset(other) ⇒ bool</code>	<code>set.union(*others) ⇒ set</code>
<code>set.difference_update(*others)</code>	<code>set.issuperset(other) ⇒ bool</code>	<code>set.update(*others)</code>
<code>set.discard(elem)</code>	<code>set.pop() ⇒ object</code>	

NumPy (import numpy as np)

Array Creation

`np.array(object, dtype=None, *, copy=True, order="K", subok=False, ndmin=0) ⇒ ndarray`
`np.arange([start,] stop[, step,] dtype=None) ⇒ ndarray`
`np.asarray(a[, dtype, order]) ⇒ ndarray`
`np.copy(a, order="K", subok=False) ⇒ ndarray`
`np.empty(shape, dtype=float, order="C") ⇒ int ⇒ ndarray`
`np.empty_like(prototype, dtype=None, order="K", subok=True, shape=None) ⇒ ndarray`
`np.eye(N, M=None, k=0, dtype=<class "float">, order="C") ⇒ ndarray`
`np.full(shape, fill_value, dtype=None, order="C") ⇒ ndarray`
`np.full_like(a, fill_value, dtype=None, order="K", subok=True, shape=None) ⇒ ndarray`
`np.fromfile(file, dtype=float, count=-1, sep=, offset=0) ⇒ ndarray`
`np.fromstring(string, dtype=float, count=-1, sep=) ⇒ ndarray`
`np.identity(n, dtype=None) ⇒ ndarray`
`np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0) ⇒ ndarray`
`np.loadtxt(fname, dtype=<class "float">, comments="#", delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0, encoding="bytes", max_rows=None) ⇒ ndarray`
`np.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0) ⇒ ndarray`
`np.meshgrid(*xi, copy=True, sparse=False, indexing="xy") ⇒ ndarray`
`np.ones(shape, dtype=None, order="C") ⇒ ndarray`
`np.ones_like(a, dtype=None, order="K", subok=True, shape=None) ⇒ ndarray`
`np.zeros(shape, dtype=float, order="C") ⇒ ndarray`
`np.zeros_like(a, dtype=None, order="K", subok=True, shape=None) ⇒ ndarray`

Array Manipulation

<code>numpy.append(arr, values, axis=None) ⇒ ndarray</code>	<code>np.repeat(a, repeats, axis=None) ⇒ ndarray</code>
<code>np.array_split(ary, indices_or_sections, axis=0) ⇒ list</code>	<code>np.reshape(a, newshape, order="C") ⇒ ndarray</code>
<code>np.atleast_1d(*arys) ⇒ ndarray</code>	<code>np.roll(a, shift, axis=None) ⇒ ndarray</code>
<code>np.atleast_2d(*arys) ⇒ ndarray</code>	<code>np.split(ary, indices_or_sections, axis=0) ⇒ list</code>
<code>np.atleast_3d(*arys) ⇒ ndarray</code>	<code>np.stack(arrays, axis=0, out=None) ⇒ ndarray</code>
<code>np.column_stack(tup) ⇒ ndarray</code>	<code>np.tile(A, reps) ⇒ ndarray</code>
<code>np.concatenate((a1, a2, ...), axis=0, out=None) ⇒ ndarray</code>	<code>np.transpose(a, axes=None) ⇒ ndarray</code>
<code>np.expand_dims(a, axis) ⇒ ndarray</code>	<code>ndarray.T ⇒ ndarray</code>
<code>np.ndarray.flatten([order]) ⇒ ndarray</code>	<code>np.unravel_index(indices, shape, order="C") ⇒ tuple</code>
<code>np.ravel(a, order="C") ⇒ ndarray ⇒ array_like</code>	

Linear Algebra

`np.cross(a, b, axisa=-1, axisb=-1, axisc=-1, axis=None) ⇒ ndarray`
`np.dot(a, b, out=None) ⇒ ndarray`
`np.linalg.det(a) ⇒ array_like`
`np.linalg.eig(a) ⇒ ndarray`
`np.inner(a, b) ⇒ ndarray`
`np.linalg.inv(a) ⇒ ndarray`
`np.matmul(x1, x2, **kwargs) ⇒ ndarray`
`np.linalg.matrix_rank(M, tol=None, hermitian=False) ⇒ array_like`
`np.linalg.norm(x, ord=None, axis=None, keepdims=False) ⇒ float|ndarray`
`np.outer(a, b, out=None) ⇒ ndarray`
`np.linalg.solve(a, b) ⇒ ndarray`
`M @ v ⇒ ndarray`
`np.trace(a, offset=0, axis1=0, axis2=1, dtype=None, out=None) ⇒ ndarray`

Matplotlib (import matplotlib.pyplot as plt)

Creation

```
plt.figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True,
           FigureClass=<class "matplotlib.figure.Figure">, clear=False, **kwargs) ⇒ Figure
plt.subplot(*args, **kwargs) ⇒ AxesSubplot
plt.subplots(nrows=1, ncols=1, *, sharex=False, sharey=False, squeeze=True, subplot_kw=None,
            gridspec_kw=None, **fig_kw) ⇒ Figure, axes.Axes
plt.twinx(ax=None) ⇒ AxesSubplot
plt.twiny(ax=None) ⇒ AxesSubplot
plt.tight_layout(*, pad=1.08, h_pad=None, w_pad=None, rect=None)
plt.savefig(*args, **kwargs)
plt.show(*, block=None)
```

Drawing

```
plt.annotate(text, xy, *args, **kwargs) ⇒ Annotation
plt.arrow(x, y, dx, dy, **kwargs) ⇒ FancyArrow
plt.contour(*args, data=None, **kwargs) ⇒ QuadContourSet
plt.contourf(*args, data=None, **kwargs) ⇒ QuadContourSet
plt.loglog(*args, **kwargs) ⇒ list of Line2D objects
plt.plot(*args, scalex=True, scaley=True, data=None, **kwargs) ⇒ list of Line2D objects
plt.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None,
            linewidths=None, verts=<deprecated parameter>, edgecolors=None, *, plotnonfinite=False, data=None,
            **kwargs) ⇒ PathCollection
plt.semilogx(*args, **kwargs) ⇒ list of Line2D objects
plt.semilogy(*args, **kwargs) ⇒ list of Line2D objects
plt.stem(*args, linefmt=None, markerfmt=None, basefmt=None, bottom=0, label=None,
         use_line_collection=True, data=None) ⇒ StemContainer
plt.step(x, y, *args, where="pre", data=None, **kwargs) ⇒ list of Line2D objects
plt.streamplot(x, y, u, v, density=1, linewidth=None, color=None, cmap=None, norm=None, arrowsize=1,
               arrowstyle="->", minlength=0.1, transform=None, zorder=None, start_points=None, maxlength=4.0,
               integration_direction="both", *, data=None) ⇒ StreamplotSet
plt.text(x, y, s, fontdict=None, **kwargs) ⇒ Text
```

Decoration

```
plt.axis(*args, emit=True, **kwargs) ⇒ Annotation
plt.axhline(y=0, xmin=0, xmax=1, **kwargs) ⇒ Line2D
plt.axvline(x=0, ymin=0, ymax=1, **kwargs) ⇒ Line2D
plt.colorbar(mappable=None, cax=None, ax=None, **kw) ⇒ Colorbar
plt.grid(b=None, which="major", axis="both", **kwargs)
plt.legend(*args, **kwargs)
plt.margins(*margins, x=None, y=None, tight=True) ⇒ tuple
plt.suptitle(t, **kwargs) ⇒ Text
plt.title(label, fontdict=None, loc=None, pad=None, *, y=None, **kwargs) ⇒ Text
plt.xlabel(xlabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)
plt.xlim(*args, **kwargs) ⇒ tuple
plt.xticks(ticks=None, labels=None, **kwargs) ⇒ tuple
plt.ylabel(ylabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)
plt.ylim(*args, **kwargs) ⇒ tuple
plt.yticks(ticks=None, labels=None, **kwargs) ⇒ tuple
```

Decoration OOP (The names of the listed OOP methods differ from the non-OOP method names.)

```
Axes.set_title(self, label, fontdict=None, loc=None, pad=None, *, y=None, **kwargs) ⇒ Text
Axes.set_xlabel(self, xlabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)
Axes.set_xlim(self, left=None, right=None, emit=True, auto=False, *, xmin=None, xmax=None) ⇒ tuple
Axes.set_xticks(self, ticks, *, minor=False) ⇒ tuple
Axes.set_ylabel(self, ylabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)
Axes.set_ylim(self, bottom=None, top=None, emit=True, auto=False, *, ymin=None, ymax=None) ⇒ tuple
Axes.set_yticks(self, ticks, *, minor=False) ⇒ tuple
```

Linestyle

Marker

Color

"-", "solid"	".", "point"	"o", "circle"	"v", "triangle_down"	"b", "blue"	"g", "green"	"r", "red"
":", "dotted"	"s", "square"	"p", "pentagon"	">", "triangle_right"	"c", "cyan"	"m", "magenta"	"k", "black"
"--", "dashed"	"*", "star"	"d", "D", "diamond"	"^", "triangle_up"	"w", "white"	"y", "yellow"	"#0F0F0F"
"-.", "dashdot"	"x", "X", "x"	"+", "P", "plus"	"<", "triangle_left"	(0.0, 0.5, 1.0)		