Python Memoire

[builtins: 2](#_Toc67181964)

[os: 4](#_Toc67181965)

[os.path: 6](#_Toc67181966)

[sys 7](#_Toc67181967)

[re 8](#_Toc67181968)

[datetime 9](#_Toc67181969)

[math 10](#_Toc67181970)

[random 11](#_Toc67181971)

[statistics 11](#_Toc67181972)

[itertools 12](#_Toc67181973)

[time 13](#_Toc67181974)

[glob 13](#_Toc67181975)

[site 13](#_Toc67181976)

[subprocess 13](#_Toc67181977)

[functools 13](#_Toc67181978)

[collections 13](#_Toc67181979)

[inspect 14](#_Toc67181980)

[operator 14](#_Toc67181981)

[requests 14](#_Toc67181982)

[textwrap 14](#_Toc67181983)

[multiprocessing 15](#_Toc67181984)

[cProfile 15](#_Toc67181985)

[socket 15](#_Toc67181986)

[hashlib 15](#_Toc67181987)

[traceback 15](#_Toc67181988)

[http.client 15](#_Toc67181989)

[numpy 16](#_Toc67181990)

[numpy.linalg 16](#_Toc67181991)

[sympy 16](#_Toc67181992)

[sympy.solvers.solveset 16](#_Toc67181993)

[set 17](#_Toc67181994)

[dictionary 18](#_Toc67181995)

[string 19](#_Toc67181996)

[list 22](#_Toc67181997)

[bytes 23](#_Toc67181998)

[bytearray 23](#_Toc67181999)

[memoryview 23](#_Toc67182000)

# [builtins](https://docs.python.org/3/library/functions.html)

range(*stop*) *or* range(*start, stop*[*, step=1*])

The arguments to the range constructor must be integers

zip(*\*iterables*)

Make an iterator that aggregates elements from each of the iterables.

len(*s*)

Return length (number of items) of an object.

abs(*x*)

Return the absolute value of a number.

sum(*iterable, /, start=0*)

Sums *start* and the items of an iterable from left to right, and returns total.

max(*iterable \**[*, key, default*]) *or* max(*arg1, arg2, \*args*[*, key*])

min(*iterable \**[*, key, default*]) *or* min(*arg1, arg2, \*args*[*, key*])

Return the smallest/largest item in an iterable or the smallest of two or more arguments

isinstance(*object, classinfo*)

Return True if the *object* argument is an instance of the *classinfo* argument.

issubclass(*class, classinfo*)

Return True if *class* is a subclass of *classinfo*. A class is considered a subclass of itself.

dir([*object*])

Without an argument, return the list of names in the current local scope. With an argument, return a list of valid attributes for that object.

all(*iterable*)

Return True if all elements of the iterable are True (or the iterable is empty).

any(*iterable*)

Return True if any of the elements in iterable is True. If the iterable is empty, return False.

ord(*c*)

Return integer representing Unicode of chararacter *c*.

chr(*i*)

Return the string representing character whose Unicode is integer *i*.

id(*object*)

Return identity of *object*; integer which is unique and constant for the lifetime of the object.

type(*object*) *or* type(*name, bases, dict*)

With one argument, return the type of *object*. With three arguments, return a new type object.

bin(*x*)

Convert an integer to a binary string prefixed with “*0b”*.

hex(*x*)

Convert an integer to a hex string prefixed with *“0x”*.

enumerate(*iterable, start=0*)

Return a enumerate object. Yields a tuple containing a count, and the next value of the iterable.

eval(*expression*[*, globals*[*, locals*]])

The expression argument is parsed and evaluated as a Python expression (technically speaking, a condition list) using the globals and locals dictionaries as global and local namespace.

sorted(*iterable, \*, key=None, reverse=False*)

Return a new sorted list from the items in *iterable*. *key* specifies a function of one argument that extracts the comparison key for each element.

del

Deletion of a name removes the binding of that name from the local or global namespace.

next(*iterator*[*, default*])

Retrieve the next item from the iterator by calling its *\_\_next\_\_()* method. If *default* is given, it is returned if the iterator is exhausted, otherwise StopIteration is raised.

map(*function, iterable, ...*)

Return an iterator that applies function to every item of iterable, yielding the results. If additional iterable arguments are passed, function must take that many arguments and is applied to the items from all iterables in parallel. With multiple iterables, the iterator stops when the shortest iterable is exhausted.

iter(*object*[*, sentinel*])

Returns an iterator object. Without the second argument, *object* must be a collection object which supports iteration. If the *sentinel* argument is given, the object must be callable; the created iterator will call object with no arguments until value returned is equal to *sentinel*.

setattr(*object, name, value*)

Assign *value* to the attribute *name*, provided the object allows it. setattr(x, 'foobar', 123) is equivalent to x.foobar = 123

getattr(*object, name*[*, default*])

Return the value of the named attribute of the object. getattr(x, 'foobar') is equivalent to x.foobar

hasattr(*object, name*)

True if *name* is the string of the name of one of the objects attributes, False if not.

delattr(*object, name*)

del attr(x, 'foobar') is equivalent to del x.foobar

reversed(*seq*)

Return a reverse iterator.

round(*number*[*, ndigits*])

Return number rounded to ndigits precision after the decimal point. If ndigits is omitted or is None, it returns the nearest integer to its input.

callable(*object*)

Return True if the *object* appears callable, False if not.

complex([*real*[, *imag*]])

Returns a complex number with the value *real* + *imag* \* j, or convert a string or number to a complex number.

breakpoint(*\*args, \*\*kws*)

This function drops you into the debugger at the call site.

filter(*function, iterable*)

Construct an iterator from those elements of *iterable* for which *function* returns True.

slice(*stop*) or slice(*start, stop*[*, step*])

Return a slice object representing the set of indices specified by *range(start, stop, step)*. *values[slice(a,b,c)]* is equivalent to *values[a:b:c]*

property(*fget=None, fset=None, fdel=None, doc=None*)

Return a property attribute. *fget* is a function for getting an attribute value, *fset* is a function for setting an attribute value. *fdel* is a function for deleting an attribute value.

vars(*[object]*)

Return the *\_\_dict\_\_* attribute for a module, class, instance, or other object. Without argument, like *locals()*

classmethod()

staticmethod()

super()

# [os](https://docs.python.org/3/library/os.html)

name

environ

system(*command*)

Execute string *command* in a subshell. Implemented by calling C function system(). On Unix, returns the exit status of the process in the format for wait(). On Windows, return the value returned by system shell.

replace(*src, dst, \*, src\_dir\_fd=None, dst\_dir\_fd=None*)

Rename the file or directory *src* to *dst*. Raises error if *dst* is a directory, if it is a file, said file is replaced.

scandir(*path=’.’*)

Returns an iterator corresponding to the entries in the directory given by *path*. In arbitrary order, and does not include *‘.’* or *‘..’*.

listdir(*path=’.’*)

Returns a list containing the names of the entries in the directory given by *path*. List is in arbitrary order, and does not include *‘.’ or ‘..’*.

walk(*top, topdown=True, onerror=None, followlinks=False*)

Generate the file names in a directory tree by wakling the tree, either top-down or bottom-up. For each directory in the tree rooted at directory *top*, yields a 3-tuple, (dirpath, dirnames, filenames).

stat(*path, \*, dir\_fd=None, follow\_symlinks=True*)

Get the status of a file or file descriptor – performs equivalent of a stat() system call on given *path*.

remove(*path, \*, dir\_fd=None*)

Remove (delete) the file *path*. Raises *IsADirectoryError* if *path* is a directory.

rmdir(*path, \*, dir\_fd=None*)

Remove (delete) the directory *path*.

readlink(*path, \*, dir\_fd=None*)

Return a string representing the path to which the symbolic link points.

link(*src, dst, \*, src\_dir\_fd=None, dst\_dir\_fd=None, follow\_symlinks=True*)

Create a hard link pointing to *src* named *dst*.

symlink(*src, dst, target\_is\_directory=False, \*, dir\_fd=None*)

Create a symbolic link pointing to *src* named *dst*.

chown(*path, uid, gid, \*, dir\_fd=None, follow\_symlinks=True*)

Change the owner and group id of path to the numeric *uid* and *gid.* Leave unchanged for -1.

chmod(*path, mode, \*, dir\_fd=None, follow\_symlinks=True*)

Change the mode of *path* to the numeric *mode*.

chroot(*path*)

Change the root directory of the current process to *path*.

sync()

Force write of everything to disk.

getlogin()

getpid()

getpgrp()

getppid()

getuid()

# [os.path](https://docs.python.org/3/library/os.path.html)

realpath(*path*)

Return the canonical path of the specified filename, eliminating any symbolic links in the path.

abspath(*path*)

Return a normalized absolutized version of the pathname path.

isfile(*path*)

Return True if path is an existing regular file. This follows symbolic links, so both islink() and isfile() can be true for the same path.

isdir(*path*)

Return True if path is an existing directory. This follows symbolic links, so both islink() and isdir() can be true for the same path.

spitext(*path*)

Split the pathname path into a pair (root, ext) such that root + ext == path, and ext is empty or begins with a period and contains at most one period.

getmtime(*path*)

Return the time of last modification of path. The return value is a floating point number giving the number of seconds since the epoch (see the time module). Raise OSError if the file does not exist or is inaccessible.

getctime(*path*)

Return the system’s ctime which, on some systems (like Unix) is the time of the last metadata change, and, on others (like Windows), is the creation time for path. The return value is a number giving the number of seconds since the epoch

dirname(*path*)

Return the directory name of pathname path.

basename(*path*)

Return the base name of pathname path.

getsize(*path*)

Return the size, in bytes, of path.

expanduser(*path*)

On Unix and Windows, return the argument with an initial component of ~ or ~user replaced by that user’s home directory.

# [sys](https://docs.python.org/3/library/sys.html)

argv

The list of arguments passed to a python script. *argv[0]* is the script name.

getsizeof(*object*[*, default*])

Return the size of an object in bytes. The object can be any type of object. All built-in objects will return correct results, but this does not have to hold true for third-party extensions as it is implementation specific.

getrecursionlimit()

Return the current value of the recursion limit, the maximum depth of the Python interpreter stack.

setrecursionlimit(*limit*)

Set the maximum depth of the Python interpreter stack to *limit*.

float\_info

A named tuple holding information about the float type.

path

A list of strings that specifies the search path for modules. Initialized from the environment variable PYTHONPATH, plus an installation-dependent default.

# [re](https://docs.python.org/3/library/re.html)

findall(*pattern, string, flags=0*)

Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left-to-right, and matches are returned in the order found. If one or more groups are present in the pattern, return a list of groups; this will be a list of tuples if the pattern has more than one group. Empty matches are included in the result.

finditer(*pattern, string, flags=0*)

Return an iterator yielding match objects over all non-overlapping matches for the RE pattern in string. The string is scanned left-to-right, and matches are returned in the order found. Empty matches are included in the result.

match(*pattern, string, flags=0*)

If zero or more characters at the beginning of string match the regular expression pattern, return a corresponding match object. Return None if the string does not match the pattern; note that this is different from a zero-length match.

search(*pattern, string, flags=0*)

Scan through string looking for the first location where the regular expression pattern produces a match, and return a corresponding match object. Return None if no position in the string matches the pattern; note that this is different from finding a zero-length match at some point in the string.

sub(*pattern, repl, string, count=0, flags=0*)

Return the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement *repl*. If the pattern isn’t found, string is returned unchanged. *repl* can be a string or a function; if it is a string, any backslash escapes in it are processed. Backreferences, such as *\6*, are replaced with the substring matched by group 6 in the pattern.

split(*pattern, string, maxsplit=0, flags=0*)

Split string by the occurrences of pattern. If capturing parentheses are used in pattern, then the text of all groups in the pattern are also returned as part of the resulting list. If maxsplit is nonzero, at most maxsplit splits occur, and the remainder of the string is returned as the final element of the list.

## Match Object

group(*[group1, ...]*)

Returns one or more subgroups of the match. If there is a single argument, the result is a single string; if there are multiple arguments, the result is a tuple with one item per argument. Without arguments, group1 defaults to zero (the whole match is returned). If a groupN argument is zero, the corresponding return value is the entire matching string; if it is in the inclusive range [1..99], it is the string matching the corresponding parenthesized group.

groups(*default=None*)

Return a tuple containing all the subgroups of the match, from 1 up to however many groups are in the pattern. The default argument is used for groups that did not participate in the match; it defaults to None.

groupdict(*default=None*)

Return a dictionary containing all the named subgroups of the match, keyed by the subgroup name. The default argument is used for groups that did not participate in the match; it defaults to None.

start([*group*])

end([*group*])

Return the indices of the start and end of the substring matched by group; group defaults to zero (meaning the whole matched substring). Return -1 if group exists but did not contribute to the match.

re

The regex objects whose match() or search() method produced this instance

string

The string passed to match() or search()

# [datetime](https://docs.python.org/3/library/datetime.html)

datetime(*year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None, \*, fold=0*)

The year, month and day arguments are required. tzinfo may be None, or an instance of a tzinfo subclass.

timedelta(*days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0*)

All arguments are optional and default to 0. Arguments may be integers or floats, and may be positive or negative.

strftime(*format*)

Return a string representing the date, controlled by an explicit format string.

strptime(*date\_string, format*)

Return a datetime corresponding to date\_string, parsed according to format.

fromtimestamp()

fromisoformat()

combine()

# [math](https://docs.python.org/3/library/math.html)

pi

exp()

expm1()

log()

log1p()

log2()

log10()

pow()

sqrt()

acos()

asin()

atan()

atan2()

sin()

cos()

tan()

degrees()

radians()

acosh()

asinh()

atanh()

cosh()

sinh()

tanh()

dist()

ceil()

floor()

comb()

copysign()

fabs()

factorial()

fmod()

fsum()

frexp()

gcd()

lcm()

isqrt()

modf()

nextafter()

perm()

prod()

remainder()

trunc()

ulp()

isclose()

isfinite()

isinf()

isnan()

erf()

# [random](https://docs.python.org/3/library/random.html)

seed(*a=None, version=2*)

Initialise rng. If *a* is *None*, seed is provided by OS, or if unavailable, the current time is used.

random()

Return a random float in the range *[0, 1)*.

uniform(*a, b*)

Return a randomly selected float in the range *[a, b)*.

randrange(*stop*) *or* randrange(*start, stop*[*, step*])

Randomly selected element from *range(start, stop, step)*.

randint(*a, b*)

Random integer between *[a, b]*.

randbytes(*n*)

Get *n* random bytes.

sample(*population, k, \*, counts=None*)

Return a *k* sized list of elements chosen from *population* without replacement.

choice(*seq*)

Return a random element from the non-empty sequence *seq*.

choices(*population, weights=None, \*, cum\_weights=None, k=1*)

Return a *k* sized list of elements chosen from *population* with replacement.

shuffle(*x*[*, random*])

Shuffle sequence *x* in-place.

getstate()

Return object capturing internal rng state.

setstate(*state*)

Restore rng state to that obtained from *getstate()*.

# [statistics](https://docs.python.org/3/library/statistics.html)

mean()

fmean()

geometric\_mean()

harmonic\_mean()

median()

mode()

multimode()

quantiles()

pstdev()

pvariance()

stdev()

variance()

# [itertools](https://docs.python.org/3/library/itertools.html)

repeat(*object*[*, times*])

Make an iterator that returns object over and over again. Runs indefinitely unless the times argument is specified.

cycle(*iterable*)

Make an iterator returning elements from the iterable and saving a copy of each. When the iterable is exhausted, return elements from the saved copy. Repeats indefinitely.

count(*start=0, step=1*)

Make an iterator that returns evenly spaced values starting with number start.

zip\_longest(*iterables, fillvalue=None*)

Make an iterator that aggregates elements from each of the iterables. If the iterables are of uneven length, missing values are filled-in with fillvalue. Iteration continues until the longest iterable is exhausted.

permutations(*iterable, r=None*)

Return successive r-length permutations of elements in iterable. If r is not specified or is None, then r defaults to the length of the iterable. The permutation tuples are emitted in lexicographic ordering according to the order of the input iterable.

product(*\*iterables, repeat=1*)

Cartesian product of input iterables. Roughly equivalent to nested for-loops in a generator expression. For example, product(A, B) returns the same as ((x,y) for x in A for y in B). The nested loops cycle like an odometer with the rightmost element advancing on every iteration.

combinations(*iterable, r*) *or* combinations\_with\_replacement*(iterable, r*)

Return r length subsequences of elements from the input iterable. The combination tuples are emitted in lexicographic ordering according to the order of the input iterable. Elements are treated as unique based on their position, not on their value. With replacement: return r length subsequences of elements from the input iterable allowing individual elements to be repeated more than once.

tee(*iterable, n=2*)

Return n independent iterators from a single iterable.

islice(*iterable, stop*) *or*islice(*iterable, start, stop*[*, step*])

Make an iterator that returns selected elements from the iterable.

starmap(*function, iterable*)

Make an iterator that computes the function using arguments obtained from the iterable. The difference between map() and starmap() parallels the distinction between function(a,b) and function(\*c)

filterfalse(*predicate, iterable*)

Make an iterator that filters elements from iterable returning only those for which the predicate is False. If predicate is None, return the items that are false.

groupby(*iterable, key=None*)

Make an iterator that returns consecutive keys and groups from the iterable. The key is a function computing a key value for each element.

chain(*\*iterables*)

Make an iterator that returns elements from the first iterable until it is exhausted, then proceeds to the next iterable, until all of the iterables are exhausted. Used for treating consecutive sequences as a single sequence.

accumulate(*iterable*[*, func, \*, initial=None*])

Make an iterator that returns accumulated sums, or accumulated results of other binary functions (specified via the optional func argument). If func is supplied, it should be a function of two arguments.

# [time](https://docs.python.org/3/library/time.html)

time()

asctime()

ctime()

gmtime()

localtime()

mktime()

get\_clock\_info()

sleep()

strftime()

strptime()

thread\_time()

# [subprocess](https://docs.python.org/3/library/subprocess.html)

run()

check\_output()

poll()

wait()

communicate()

send\_signal()

terminate()

kill()

# [functools](https://docs.python.org/3/library/functools.html)

reduce(*function, iterable*[*, initializer*])

Apply *function* of two arguments cumulatively to the items of *iterable*, LR, so as to reduce the iterable to a single value.

partial(*func, /, \*args, \*\*kw*)

Return a new partial object, which when called will behave like *func* called with the positional arguments *args* and keyword arguments *kw*.

partialmethod(*func, /, \*args, \*\*kw*)

Return a partialmethod descriptior, which behaves like *partial*(), except it is to be used as a method definition rather than directly called.

@cache(*user\_function*)

Lightweight unbounded function cache.

@cached\_property(*func*)

Transform a method of a class into a property whose value is computed once and then cached as a normal attribute for the life of the insance. Like *property()*, wit the addition of caching.

@lru\_cache(*user\_function*) or @lru\_cache(*maxsize=128, typed=False*)

Decorator to wrap a function with a memorizing callable that saves up to *maxsize* most recent calls.

@total\_ordering()

Given a class defining one or more rich comparison ordering methods, this class decorator supplies the rest. The class must define one of: *\_\_lt\_\_(), \_\_le\_\_(), \_\_gt\_\_(), or \_\_ge\_\_()*, in addition to *\_\_eq\_\_()*.

# [collections](https://docs.python.org/3/library/collections.html)

Counter([*iterable-or-mapping*])

A counter is a dict subclass for counting hashable objects. It is a collection where elements are stored as dict keys, and counts are stored as dict values.

Counter.elements()

Return an iterator over elements, repeating each as many times as its count. Elements are returned in the order first encountered.

Counter.most\_common([*n*])

Return a list of the *n* most common elements and their counts from the most common to least. If *n* is omitted or None, all elements are returned.

Counter.subtract([*iterable-or-mapping*])

Elements are subtracted from an iterable or another mapping, but subtracts counts instead of replacing them.

Counter.update([*iterable-or-mapping*])

Elements are counted from an iterable or added-in from another mapping or counter. Like *dict.update()*, but adds counts instead of replacing them. *Iterable* is expected to be a sequence of elements (not *k-v* pairs)

defaultdict(*default\_factor*[*, ...*])

Return a dict like object (subclass of dict). If *default\_factor* is not None, it is called without arguments to provide a default value for the given *key*.

deque([*iterable*[*, maxlen*]])

Returns deque object, initialized LR with data from *iterable*. Deques are a generalization of stacks and queues.

deque.append(*x*)

Add *x* to RHS of deque

deque.appendleft(*x*)

Add *x* to LHS of deque

deque.clear()

Remove all elements from deque

deque.copy()

Create shallow copy of deque

deque.count(*x*)

Count number of deque elements equal to *x*

deque.extend(*iterable*)

Append elements from *iterable* to RHS of deque

deque.extendleft(*iterable*)

Append elements from *iterable* to LHS of deque in reversing order

deque.index(*x*[*, start*[*, stop*]])

Return the position of first *x* in the deque (between indexes *start* and *stop*)

deque.insert(*i, x*)

Insert *x* into the deque at position *i*

deque.pop()

Remove and return element from RHS, raise *IndexError* if empty

deque.popleft()

Remove and return element from LHS, raise *IndexError* if empty

deque.remove(*value*)

Remove first occurrence of *value*. If not found, raise *ValueError*

deque.reverse()

Reverse elements in-place

deque.rotate(*n=1*)

Rotate the deque *n* steps right. Rotate left for negative value.

namedtuple()

OrderedDict([*items*])

Return instance of dict subclass with methods for rearranging order.

popitem(*last=True*)

Removes and returns a *k-v* pair. Order is LIFO if *last* is True, or FIFO if it is False.

move\_to\_end(*key, last=True*)

Move an existing *key* to either end of ordered dict, right (end) if *last* is True, left (start) if it is False. Raises *KeyError* if *key* does not exist.

ChainMap(*\*maps*)

Group multiple dicts or other mappings together to create a single, updateable view. Order is that of a series of *dict.update()* calls, starting at the last mapping.

ChainMap.new\_child(*m=None*)

Return a new ChainMap containing a new map, followed by all of the maps in the current instance. If *m* is specified, it becomes the new map at the front of the list of mappings (an empty dict is used if None).

# [heapq](https://docs.python.org/3/library/heapq.html)

Note: heapsort is not a stable sort.

heappush(*heap, item*)

Push the value item onto the *heap*, maintaining the heap invariant

heappop(*heap*)

Pop and return the smallest item from the *heap*, maintaining the heap invariant. Raise *IndexError* if empty. Use *heap[0]* to access smallest element without removing it.

heappushpop(*heap, item*)

Push *item* on the heap, then pop and return the smallest item from *heap*.

heapify(*x*)

Transform list *x* into a heap.

heapreplace(*heap, item*)

Pop and return the smallest item from *heap*, and also push the new *item*. Raise *IndexError* if empty.

merge(*\*iterables, key=None, reverse=False*)

Merge multiple sorted inputs into a single sorted output, returning iterator over sorted values.

nlargest(*n, iterable, key=None*)

Return a list with the *n* largest elements from *iterable*.

nsmallest(*n, iterable, key=None*)

Return a list with the *n* smallest elements from *iterable*.

# [bisect](https://docs.python.org/3/library/bisect.html)

bisect\_left(*a, x, lo=0, hi=len(a)*)

Locate the insertion point for *x* in *a* to maintain sorted order. Parameters *lo* and *hi* may specify a subset of the list to be considered. The returned insertion point *i* partitions the array *a* into two halves so that   
*all(val < x for val in a[lo:i])* and *all(val >= x for val in a[i:hi])*

bisect\_right(*a, x, lo=0, hi=len(a)*)

bisect(*a, x, lo=0, hi=len(a)*)

As per *bisect\_left()*, but returns an insertion point which comes after any existing entries of *x* in *a*. The returned insertion *i* point partitions the array *a* so that:  
*all(val <= x for val in a[lo:i])* and *all(val > x for val in a[i:hi])*

insort\_left(*a, x, lo=0, hi=len(a)*)

Insert *x* in *a* in sorted order. Assuming *a* is sorted, equivalent to: *a.insert(bisect.bisect\_left(a, x, lo, hi), x)*

insort\_right(*a, x, lo=0, hi=len(a)*)

insort(*a, x, lo=0, hi=len(a)*)

As per *insort\_left()*, but insert *x* in *a* after any existing entries of *x*.

# [queue](https://docs.python.org/3/library/queue.html)

Queue(*maxsize=0*)

FIFO queue. Infinite for *maxsize=0*

LifoQueue(*maxsize=0*)

LIFO queue. Infinite for *maxsize=0*

PriorityQueue(*maxsize=0*)

Priority queue. Infinite for *maxsize=0*

*Queue elements are accessible through attribute*

queue.queue

*Queue Methods:*

qsize()

Approximate size of queue

empty()

True if queue is empty, False otherwise

full()

True if queue is full, False otherwise

put(*item, block=True, timeout=None*)

Put *item* into queue. If *block* is True and *timeout* is None, block until free spot is available. If *timeout* is given, block the given value of seconds before raising *Full* exception.

put\_nowait(*item*)

Equivalent to *put(item, False)*

get(*block=True, timeout=None*)

Remove and return an item from queue. If *block* is True and *timeout* is None, block until an item is available. If *timeout* is given, block the given number of seconds before raising *Empty* exception.

get\_nowait()

Equivalent to *get(False)*

task\_done()

Indicate that a formerly enqueued task is complete. Used by consumer threads. For each *get()*, a subsequent *task\_done()* tells queue processing is complete. If *join()* is blocking, resume when all items have been processed.

join()

Block until all items in the queue have been gotten and processed.

# [inspect](https://docs.python.org/3/library/inspect.html)

getmembers()

getmodule()

signature()

getclasstree()

ismodule()

isclass()

ismethod()

isfunction()

isgenerator()

isawaitable()

isbuiltin()

getmro(*cls*)

Return a tuples of the base classes of class *cls*, including *cls*, in method resolution order. No class appears more than once in this tuple. In most cases, *cls* will be the first element of the tuple.

getmembers(*sample, inspect.isclass*)

Get all classes present in a module

getmembers(*sample.X, inspect.isfunction*)

Get all methods in function

# [textwrap](https://docs.python.org/3/library/textwrap.html)

wrap(*text, width=70, \*\*kwargs*)

Wraps the single paragraph in *text* so every line is at most *width* characters long. Returns a list of lines, without final newlines.

fill(*text, width=70, \*\*kwargs*)

Wraps the single paragraph in *text*, and returns a single string containing the wrapped paragraph. Equivalent to *“\n”.join(wrap(text, ...))*

shorten(*text, width, \*\*kwargs*)

Collapse and truncate the given *text* to fit the given *width*. kwargs are those of TextWrapper (*placeholder, …*).

dedent(*text*)

Remove any common leading whitespace from every line in *text*.

indent(*text, prefix, predicate=None*)

Add *prefix* to the beginning of the selected lines in *text*.

# [requests](https://requests.readthedocs.io/en/master/api/)

get()

post()

put()

delete()

head()

options()

# [operator](https://docs.python.org/3/library/operator.html)

add(*a, b*)

concat(*seq1, seq2*)

contains(*seq, obj*)

truediv(*a, b*)

floordiv(*a, b*)

and\_(*a, b*)

xor(*a, b*)

invert(*a*)

or\_(*a, b*)

pow(*a, b*)

is\_(*a, b*)

is\_not(*a, b*)

setitem(*obj, k, v*)

delitem(*obj, k*)

getitem(*obj, k*)

lshift(*a, b*)

mod(*a, b*)

mul(*a, b*)

matmul(*a, b*)

pos(*a*)

neg(*a*)

not\_(*a*)

rshift(*a, b*)

setitem(*seq, slice(i,j), values*)

delitem(*seq, slice(i,j)*)

getitem(*seq, slice(i,j))*

mod(*s, obj*)

sub(*a, b*)

truth(*obj*)

lt(*a, b*)

le(*a, b*)

eq(*a, b*)

ne(*a, b*)

ge(*a, b*)

gt(*a, b*)

attrgetter(*attr*) *or* attrgetter(*\*attrs*)

Return a callable object that fetches *attr* from its operand. If more than one attribute requested, returns tuple.

itemgetter(*item*) *or* itemgetter(*\*items*)

Return a callable object that fetches *item* from its operand, using *\_\_getitem\_\_()* method. Returns a tuple if multiple items given.

methodcaller(*name, /, \*args, \*\*kwargs*)

Return a callable object that calls the method *name* on its operand. If additional arguments / keywords are given, they are passed to method.

# [json](https://docs.python.org/3/library/json.html)

dumps(*obj*, *\**, *skipkeys=False*, *ensure\_ascii=True*, *check\_circular=True*, *allow\_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort\_keys=False*, *\*\*kw*)

Serialize *obj* to a JSON formatted str. Arguments as per *dump()*.

loads(*s*, *\**, *cls=None*, *object\_hook=None*, *parse\_float=None*, *parse\_int=None*, *parse\_constant=None*, *object\_pairs\_hook=None*, *\*\*kw*)

Deserialize *s* (str, bytes, or bytearray) containing JSON document to a python object. Arguments as per *load()*.

load(*fp*, *\**, *cls=None*, *object\_hook=None*, *parse\_float=None*, *parse\_int=None*, *parse\_constant=None*, *object\_pairs\_hook=None*, *\*\*kw*)

Deserialize file-like *fp* containing JSON document to python object.

dump(*obj*, *fp*, *\**, *skipkeys=False*, *ensure\_ascii=True*, *check\_circular=True*, *allow\_nan=True*, *cls=None*, *indent=None*, *separators=None*, *default=None*, *sort\_keys=False*, *\*\*kw*)

Serialize *obj* as JSON formatted stream to file-like object *fp*. Produces str (not bytes), *fp* must support str input.

Translation: JSON Python

object dict

array list

string str

number (int) int

number (real) float

true True

false False

null None

# [pickle](https://docs.python.org/3/library/pickle.html)

# [tempfile](https://docs.python.org/3/library/tempfile.html)

TemporaryFile()

TemporaryDirectory()

gettempdir()

mkstemp()

mkdtemp()

# [difflib](https://docs.python.org/3/library/difflib.html)

SequenceMatcher(*isjunk=None, a=’’, b=’’, autojunk=True*)

SequenceMatcher.find\_longest\_match(*alo=0, ahi=None, blo=0, bhi=None*)

SequenceMatcher.get\_opcodes()

SequenceMatcher.get\_grouped\_opcodes(*n=3*)

SequenceMatcher.ratio()

# [glob](https://docs.python.org/3/library/glob.html)

glob()

iglob()

escape()

# [site](https://docs.python.org/3/library/site.html)

getsitepackages()

getuserbase()

# [multiprocessing](https://docs.python.org/3/library/multiprocessing.html)

cpu\_count()

# [cProfile](https://docs.python.org/3/library/profile.html)

run()

# socket

# hashlib

sha256()

# traceback

print\_stack()

# http.client

HTTPConnection()

# [array](https://docs.python.org/3/tutorial/datastructures.html)

array(*typecode*[*, initalizer*])

A new array whose items are restricted by typecode, and initalised from *initalizer*, which must be a list, bytes-like object, or iterable.

append(*x*)

Append a new item with value *x* to the end of the array

buffer\_info()

Return a tuple (*address, base*) giving memory address and length of the array buffer

byteswap()

Byteswap all items of the array

count(*x*)

Return the number of occurrences of *x* in the array

extend(*iterable*)

Append items from *iterable* to end of the array. If *iterable* is an array, it must have the same typecode.

frombytes(*s*)

Append items from the string, interpreting the string as an array of machine values.

fromfile(*f, n*)

Read n items from file object *f* and append them to the end of the array

fromlist(*list*)

Append items from *list*.

fromunicode(*s*)

Extend this array with data from the given Unicode string. Array must be type *‘u’*

index(*x*)

Return the smallest *i* such that *i* is the index of the first occurrence of *x* in array.

insert(*i, x*)

Insert a new item with value *x* in the array before position *i*

pop([*i*])

Remove the item with the index *i* from the array and returns it. *i* defaults to *-1*.

remove(*x*)

Remove the first occurrence of *x* from the array

reverse()

tobytes()

tofile(*f*)

tolist()

tounicode()

*typecodes:*

b signed char l signed long

B unsigned char L unsigned long

u wchar\_t q signed long long

h signed short Q unsigned long long

H unsigned short f float

i signed int d double

I unsigned int

# [set](https://docs.python.org/3/library/stdtypes.html#set)

set([*iterable*])

frozenset([*iterable*])

add(*elem*)

Add element *elem* to set.

update(*\*others*) *or* set |= other ...

Add elements from all *others* to set.

remove(*elem*)

Remove element *elem* from set. Raises *KeyError* if not found.

discard(*elem*)

Remove element *elem* from set if it is present.

pop()

Remove and return an arbitrary element from the set. Raises KeyError if the set is empty.

clear()

Remove all elements from the set.

copy()

Return a shallow copy of the set.

isdisjoint(*other*)

Return True if the set has no elements in common with other.

issubset(*other*) *or* set <= other

Test whether every element in the set is in other.

issuperset(*other*) *or* set >= other

Test whether every element in other is in the set.

union(*\*others*) *or* set | other ...

Return a new set with elements from the set and all others.

intersection(*\*others*) *or* set & other ...

Return a new set with elements common to the set and all others.

difference(*\*others*) *or* set – other ...

Return a new set with elements in the set that are not in the others.

symmetric\_difference(*other*) *or* set ^ other

Return a new set with elements in either the set or other but not both.

intersection\_update(\*others) *or* set &= other ...

difference\_update(\*others) *or* set -= other ...

symmetric\_difference\_update(\*others) *or* set -= other ...

# [dictionary](https://docs.python.org/3/library/stdtypes.html#dict)

dict(*\*\*kwarg*) *or* dict(*mapping, \*\*kwarg*) *or* dict(*iterable, \*\*kwarg*)

clear()

Remove all items from dict.

copy()

Return shallow copy of the dict.

get(*key*[*, default=None*])

Return value for *key* if key is in the dict, else *default*.

pop(*key*[*, default*])

Return and remove value for key from dict. Raises exception if key not in dict and default not given.

popitem()

Remove and return a (*key, value*) pair. Pairs are returned in LIFO order.

items()

Return view of the dictionaries items (*key, value*) pairs.

keys()

Return view of the dictionaries keys.

values()

Return view of the dictionaries values.

setdefault(*key*[*, default=None*])

If *key* is in the dict, return its value. If not, insert *key* with value of *default.*

update([*other*])

Update the dictionary with the key/value pairs from *other*, (overwrites existing keys).

d | other

Create new dict with merged keys and value of *d* and *other*. The values of *other* take priority if there are conflicts.

d |= other

Update the dict *d* with keys and values from *other*. Values from *other* take priority if there are conflicts.

# [string](https://docs.python.org/3/library/stdtypes.html#string-methods)

str(*object=’’*) *or* str(*object=b’’, encoding=’utf-8’, errors=’strict’*)

Return a string version of object. If neither *encoding*, nor *errors* is given, str(*object*) returns *object.\_\_str\_\_()*. If *encoding* or *errors* is given, *object* should be a bytes-like object.

join(*iterable*)

Return a string which is the concatenation of the strings in iterable, using string providing join as separator.

count(*sub*[*, start*[*, end*]])

Return the number of non-overlapping occurrences of substring *sub*, in the range [*start, end*] (if given).

replace(*old, new*[*, count*])

Return a copy of the string with all occurrence of substring *old* replaced by *new*. If *count* is given, only the first *count* occurrences are replaced.

Simulating rreplace()

*new*.join(\_*str*.rsplit(*old, count*))

strip([*chars*])

Return a copy of the string with leading and trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If None is given, defaults to removing whitespace. The chars argument is not a prefix or suffix; rather, all combinations of its values are stripped.

find(*sub*[*, start*[*, end*]])

Return the lowest index in the string where substring *sub* is found within range [*start, end*] (if given). If not found, return *-1*.

rfind(*sub*[*, start*[*, end*]])

Return the highest index in the string where substring *sub* is found within range [*start, end*] (if given). If not found, return *-1*.

index(*sub*[*, start*[*, end*]])

Like *find()*, but raise *ValueError* if substring not found.

rindex()

Like *rfind()*, but raise *ValueError* if substring not found.

split(*sep=None, maxsplit=-1*)

Return a list of words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done (resulting in at most maxsplit+1 elements). If *sep* is given, consecutive delimiters are deemed to delimit empty strings. If *sep* is None, runs of consecutive whitespace are regarded as seperator.

format(*\*args, \*\*kwargs*)

Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces {}. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument.

capitalize()

Return a copy of the string with its first character capitalized and the rest lowercased.

center(*width*[*, fillchar*])

Return centred in a string of length *width*. Padding is done using *fillchar* (default is space). Origional string is returned if *width* *<= len(s)*.

ljust(*width*[*, fillchar*])

Return the string left-justified in a string of length *width*. Padding is done using *fillchar* (default is space). Original string is returned if *width <= len(s)*.

rjust(*width*[*, fillchar*])

As per *ljust()*, except right-justified.

encode(*encoding=’utf8’, errors=’strict’*)

Return an encoded version of the string as bytes object.

startswith(*prefix*[*, start*[*, end*]])

Return True if string starts with the *prefix*, otherwise return False. *prefix* can also be a tuple of prefixes to look for. With optional *start*, test string beginning at that position. With optional *end*, stop comparing string at that position.

endswith(*suffix*[*, start*[*, end*]])

Return True if the string ends with the specified *suffix*, otherwise return False. *suffix* can also be a tuple of suffixes to look for. With optional *start*, test beginning at that position. With optional *end*, stop comparing at that position.

rstrip([*chars*])

*strip()* on RHS of string.

lstrip([*chars*])

*strip()* on LHS of string.

lower()

Return a copy of the string with all the cased characters converted to lowercase.

upper()

Return a copy of the string with all cased characters converted to uppercase.

islower()

Return True if all cased characters are lower case (and there is at least one cased character).

isupper()

Return True if all cased characters are upper case (and there is at least one cased character).

swapcase()

Return a copy of the string with uppercase characters converted to lowercase, and vice-versa.

title()

Return a titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase

expandtabs(*tabsize=8*)

Return a copy of the string where all tab characters are replaced by one or more spaces, depending on the current column and the given tab size.

splitlines([*keepends*])

Return a list of the lines in the string, breaking at line boundaries. Line breaks are not included in the resulting list unless *keepends* is given and true.

zfill(*width*)

Return a copy of the string left-filled with *‘0’* digits to make a string of length *width*. A leading sign is handled by inserting the padding after the sign character. The origional string is returned if *width <= len(s)*.

partition(*sep*)

Split the string at the first occurrence of *sep*, and return a 3-tuple containing the string before, at, and after the separator. If the separator is not found, return a 3-tuple containing the string, followed by two empty strings.

rpartition(*sep*)

As per *partition()*, but split at the last occurrence of *sep*.

removeprefix(*prefix, /*)

If the string starts with the *prefix* string, return string with prefix removed, otherwise remove original string.

removesuffix(*suffix, /*)

If the string ends with the *suffix* string, return string with suffix removed, otherwise remove original string.

isprintable()

Return True if all characters in the string are printable, or the string is empty, otherwise False.

isnumeric()

Return True if all characters in the string are numeric characters, and there is at least one character, False otherwise.

isalpha()

Return True if all characters in the string are alphabetic and there is at least one character, False otherwise.

isalnum()

Return True if all characters in the string are alphanumeric and there is at least one character, False otherwise.

isascii()

Return True if the string is empty or all characters in the string are ASCII, False otherwise.

translate(*table*)

Return a copy of the string in which each character has been mapped through the given translation table.

maketrans(*x*[, *y*[*, z*]])

(static) returns a translation table usable for *translate()*.

# [list](https://docs.python.org/3/library/stdtypes.html#list)

list(*[iterable]*)

The constructor builds a list whose items are the same and in the same order as iterable’s items. iterable may be either a sequence, a container that supports iteration, or an iterator object. If iterable is already a list, a copy is made and returned.

append(*x*)

Appends *x* to the end of the sequence. Equivalent to *s[len(s):len(s)] = [x]*.

pop([*i*])

Retrieves the item at *i*, and also removes it (Defaults to *-1*).

sort(*\*, key=None, reverse=False*)

This method sorts the list in place, using only < comparisons between items. Exceptions are not suppressed, if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state). This method sorts the sequence in place. It is a stable sort.

clear()

Remove all items from list. Equivalent to *del s[:]*

copy()

Create a shallow copy of the list. Equivalent to *s[:]*

extend(*t*)

Extends the list with the contents of *t*. Equivalent to *s[len(s):len(s)] = t*

insert(*i, x*)

Inserts *x* into list at index given by *i*. Equivalent to *s[i:i] = [x]*

remove(*x*)

Remove the first item from list where *s[i]* is equal to *x*.

reverse()

Reverses list in-place.

index(*x*[*, i*[*, j*]])

Index of the first occurrence of *x* in list (at or after index *I* and before index *j*).

count(*x*)

Total number of occurrences of *x* in list.

[i for i, c in enumerate(\_vals) if c == n]

List of indexes of all occurrences of *n* in list *\_vals*.

# bytes

# bytearray

count()

removeprefix()

removesuffix()

decode()

find()

index()

join()

replace()

startswith()

endswith()

strip()

split()

# memoryview

# [numpy](https://numpy.org/doc/stable/reference/index.html)

sum(*a, axis=None, dtype=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>*)

Sum of array elements over a given axis

ndarray(*shape, dtype=float, buffer=None, offset=0, strides=None, order=None*)

A multidimensional array of homogenous, fixed-size items. Should be constructed using *array, zeros,* or *empty*.

zeros(*shape, dtype=float, order=’C’, like=None*)

Return new array of given shape and type, fill with 0.

ones(*shape, dtype=None, order=’C’, \*, like=None*)

Return new array of given shape and type, fill with 1.

empty(*shape, dtype=float, order=’C’, like=None*)

Return a new array of given shape and type, without initializing entries.

full(*shape, fill\_value, dtype=None, order=’C’, like=None*)

Return a new array of given shape and type, filled with *fill\_value*

save(*file, arr, allow\_pickle=True, fix\_imports=True*)

Save an array to a binary file in Numpy *.npy* format.

savez(*file, \*args, \*\*kwds*)

Save several arrays into a single binary file in *.npz* format.

load(*file, mmap\_mode=None, allow\_pickle=False, fix\_imports=True, encoding='ASCII*)

Load arrays or pickled objects from *.npy*, *.npz*, or pickled files. Note: use False for *allow\_pickle* for any untrusted data sources.

diag(*v, k=0*)

If *v* is a 2d array, return a copy of its *k*-th diagonal. If *v* is a 1d array, return a 2d array with *v* on the *k-*th diagonal.

diagonal(*a, offset=0, axis1=0, axis2=1*)

Return specified diagonals.

diagflat(*v, k=0*)

Create a 2d array with the flattened input as diagonal

identity(*n, dtype=None, \*, like=None*)

Return the identity array.

eye(*N, M=None, k=0, dtype=<class 'float'>, order='C', \*, like=None*)

Return a 2d array with ones on the diagonal and zeros elsewhere.

arange([*start,*] *stop,* [*step,*] *dtype=None, \*, like=None*)

Return evenly spaced values within a given interval. Equivalent to *range()* for integer arguments. For non-integer arguments, use *linspace()*

linspace(*start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0*)

Returns *num* evenly spaced numbers over a specified interval. If *endpoint* is True, *stop* is the last sample, otherwise it is excluded. If *retstep* is True, return *(samples, step)*, where *step* is the spacing between samples.

logspace(*start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0*)

Return array of values spaced evenly on a log scale

nditer(***op*, *flags=None*, *op\_flags=None*, *op\_dtypes=None*, *order='K'*, *casting='safe'*, *op\_axes=None*, *itershape=None*, *buffersize=0***)

Iterator object to iterate over arrays. See: [Iterating over arrays](https://numpy.org/doc/stable/reference/arrays.nditer.html#arrays-nditer). Note that iteration order is that of the memory layout of the array.

dot(*a, b, out=None*)

Dot product of two arrays.

vdot(*a, b*)

Dot product of two vectors. Flattens matrices. Uses complex conjugate of first argument.

tensordot(*a, b, axes=2*)

Tensor dot product along specified axes.

einsum(*subscripts, \*operands, out=None, dtype=None, order=’K’, casting=’safe’, optimize=False*)

Evaluate Einstein summation convention on the operands

inner(*a, b*)

Compute the inner product of two arrays

outer(*a, b, out=None*)

Compute the outer product of two vectors

cross(*a, b, axisa=-1, axisb=-1, axisc=-1, axis=None*)

Return the cross product of two (arrays of) vectors. If the dimension of *a* or *b* is 2, the third component is assumed to be zero. If the dimension of both is 2, only the z-component (only non-zero component) of the result is returned.

cumsum(*a, axis=None, dtype=None, out=None*)

Return the cumulative sum of the elements along a given axis

set\_printoptions(*precision=None, threshold=None, edgeitems=None, linewidth=None, suppress=None, nanstr=None, infstr=None, formatter=None, sign=None, floatmode=None, \*, legacy=None*)

Set options determining how floats, arrays, and other Numpy objects are displayed.

sort(*a, axis=-1, kind=None, order=None*)

Return a sorted copy of an array

where(*condition*[*, x, y*])

Return elements chosen from x or y depending on condition. If *x* and *y* are omitted, indexes are returned (as tuple).

full\_like(*a, fill\_value, dtype=None, order=’K’, subok=True, shape=None*)

Return a full array with the same shape and type as a given array

ones\_like(*a, dtype=None, order=’K’, subok=True, shape=None*)

Return an array of ones with the same shape and type as a given array

empty\_like(*prototype, dtype=None, order=’K’, subok=True, shape=None*)

Return a new array with the same shape and type as a given array

zeros\_like(*a, dtype=None, order=’K’, subok=True, shape=None*)

Return an array of zeros with the same shape and type as a given array

asfarray(*a, dtype=<class ‘numpy.double’>*)

Return an array converted to float type

asarray(*a, dtype=None, order=None, \*, like=None*)

Convert the input to an array

multiply(*x1, x2, /, out=None, \*, where=True, casting=’same\_kind’, …*)

Multiply arguments elementwise

add(*x1, x2, /*[*, out, where, casting, order, …*])

Add arguments elementwise

subtract(*x1, x2, /*[*, out, where, casting, …*])

Subtract arguments elementwise

divide(*x1, x2, /*[*,out, where, casting, …*])

Divide arguments elementwise

power(*x1, x2, /*[*, out, where, casting*)

First array elements raised to powers from second array, elementwise

in1d(*ar1, ar2, assume\_unique=False, invert=False*)

Test whether each element of a 1d array is also present in a second array

isin(*element, test\_elements, assume\_unique=False, invert=False*)

Calculates *element* in *test\_elements*, broadcasting over *element* only. Returns Boolean array of the shape of *element* that is True when an element of *element* is in *test\_elements* and False otherwise.

intersect1d(*ar1, ar2, assume\_unique=False, return\_indicies=False*)

Find the intersection of two arrays. Return the sorted, unique values that are in both input arrays.

unique(*ar, return\_index=False, return\_inverse=False, return\_counts=False, axis=None*)

Find the unique elements of an array (that is, each element once, elements with multiple occurrences will be included once). Return the sorted unique elements of an array.

setdiff1d(*ar1, ar2, assume\_unique=False*)

Find the set difference of two arrays. Return the unique values in *ar1* that are not in *ar2*.

setxor1d(*ar1, ar2, assume\_unique=False*)

Find the set exclusive-or of two arrays. Return the sorted, unique values that are in only one of the input arrays.

union1d(*ar1, ar2*)

Find the union of two arrays. Return the unique, sorted array of values that are in either of the two input arrays.

all(*a, axis=None, out=None, keepdims=<no value>, \*, where=<no value>*)

Test whether all array elements along a given axis evaluate to True

any(*a, axis=None, out=None, keepdims=<no value>, \*, where=<no value>*)

Test whether any array element along a given axis evaluates to True

tile(*A, reps*)

Construct an array by repeating *A* the number of times given by *reps*

repeat(*a, repeats, axis=None*)

Repeat elements of an array

argmax(*a, axis=None, out=None*)

Return the indicies of the maximum values along an axis

argmin(*a, axis=None, out=None*)

Returns the indicies of the minimum values along an axis

greater(*x1, x2, /[, out, where, casting, …])*

greater\_equal(*x1, x2, /[, out, where, …]*)

less(*x1, x2, /[, out, where, casting,* …])

less\_equal(*x1, x2, /[, out, where, casting, …]*)

equal(*x1, x2, /[, out, where, casting, …])*

not\_equal(*x1, x2, /[, out, where, casting, …]*)

Elementwise comparisons

allclose(*a, b[, rtol, atol, equal\_nan]*)

True if two arrays are equal elementwise within a tolerance.

lexsort(*keys, axis=-1*)

Perform an indirect stable sort using a sequence of *keys*. Returns an array of integer indices that describe the sort order by multiple columns. Sort is done using *keys* in reverse order as columns.

nonzero(*a*)

Return the indices of the elements that are non-zero. Returns a tuple of arrays, one of each dimension of *a*.

argwhere(*a*)

Find the indices of array elements that are non-zero, grouped by element.

count\_nonzero(*a, axis=None, \*, keepdims=False*)

Counts the number of non-zero values in the array *a*.

extract(*condition, arr*)

Return the elements of an array that satisfy some condition.   
Equivalent to: *compress(ravel(condition), ravel(arr)).*

ravel(*a, order=’C’*)

Return a contiguous flattened array.

tri(*N, M=None, k=0, dtype=<class ‘float’>, \*, like=None*)

Return an array with ones at and below the given diagonal *k*, and zeros elsewhere.

tril(*m, k=0*)

Return an array with elements above the *k*-th diagonal zeroed.

triu(*m, k=0*)

Return an array with elements below the *k*-th diagonal zeroed.

swapaxes(*a, axis1, axis2*)

Interchange two axes of an array

moveaxis(*a, source, destination*)

Move axes of an array to new positions. Other axes remain in their original order.

atleast\_1d(*\*arys*)

atleast\_2d(*\*arys*)

atleast\_3d(*\*arys*)

Convert inputs to arrays with at least 1/2/3 dimensions.

require(*a, dtype=None, requirements=None, \*, like=None*)

Return an ndarray of the provided type that satisfies requirements.

expand\_dims(*a, axis*)

Expand the shape of an array

squeeze(*a, axis=None*)

Remove axes of length one from *a*

reshape(*a, newshape, order=’C’*)

Gives a new shape to an array without changing its data

concatenate(*(a1, a2, …), axis=0, out=None, dtype=None, castring=’same\_kind’*)

Join a sequence of arrays along an existing axis

block(*arrays*)

Assemble an n-d array from nested list of blocks. Blocks in the innermost lists are concatenated along the last dimension (-1), then these are concatenated along second last dimension, and so on until outermost list.

split(*ary, indicies\_or\_sections, axis=0*)

Split an array into multiple sub-arrays as views into *ary*. Raises *ValueError* if *indicies\_or\_sections* is an integer, but split does not result in equal division. If *indicies\_or\_sections* is a 1d sorted list, use values as indexes of locations to split at.

array\_split(*ary, indicies\_or\_sections, axis=0*)

Split an array into multiple sub-arrays.

vsplit(*ary, indices\_or\_sections*)

Split an array into multiple sub-arrays vertically (row-wise). Equivalent to *split()* with *axis=0*

hsplit(*ary, indices\_or\_sections*)

Split array into subarrays horizontally (column-wise). Equivalent to *split()* with *axis=1*.

dsplit(*ary, indices\_or\_sections*)

Split array into multiple sub-arrays along 3rd axis (depth). Equivalent to *split()* with *axis=2*

stack(*arrays, axis=0, out=None*)

Join a sequence of arrays along a new axis. *axis* specifies the index of the new axis in the dimension of result

hstack(*tup*)

Stack arrays in sequence horizontally (column-wise). Equivalent to concatenation along *axis=1*

vstack(*tup*)

Stack arrays in sequence vertically (row-wise). Equivalent to concatenation along *axis=0* after 1d arrays have been reshaped to (1,N).

dstack(*tup*)

Stack arrays in sequence depth wise. Equivalent to concatenation along <third-axis> <*axis=2*> after 2d arrays have been reshaped to (M,N,1), and 1d arrays have been reshaped to (1,N,1).

column\_stack(*tup*)

Stack 1d arrays as columns into a 2d array

row\_stack(*tup*)

Stack arrays in sequence vertically (row-wise)

take(*a, indices, axis=None, out=None, mode=’raise’*)

Take elements from an array along an axis.

put(*a, ind, v, mode=’raise’*)

Replaces specified elements of an array with given values. Indexing works on flattened array. Roughly equivalent to: *a.flat[ind] = v*

copyto(*dst, src, casting=’same\_kind’, where=True*)

Copies values from one array to another, broadcasting as necessary.

compress(*condition, a, axis=None, out=None*)

Return selected slices of an array along a given axis. When working on 1d array, equivalent to *extract()*.

place(*arr, mask, vals*)

Change elements of an array based on conditional and input values.

meshgrid(*\*xi, copy=True, sparse=False, indexing=’xy’*)

Return N-d coordinate array, given 1-d coordinate arrays *x1, x2, …, xn*.

delete(*arr, obj, axis=None*)

Return an array, with sub-arrays along an axis deleted. For a 1d array, return elements not returned by *arr[obj]*

insert(*arr, obj, values, axis=None*)

Inset *values* along the given *axis* before the given indices *obj*.

append(*arr, values, axis=None*)

Append values to the end of an array

frombuffer(*buffer, dtype=float, count=-1, offset=0, \*, like=None*)

Interpret a buffer as a 1-d array

percentile(*a, q, axis=None, out=None, overwrite\_input=False, interpolation=’linear’, keepdims=False*)

Compute the *q-*th percentile(s) of the data along the specified axis.

genfromtxt()

logical\_not()

logical\_and()

logical\_xor()

unravel\_index(*indices, shape, order=’C’*)

Converts a flat index, or array of flat indices, into a tuple of coordinate arrays

ravel\_multi\_index(*multi\_index, dims, mode=’raise’, order=’C’*)

Converts a tuple of index arrays into an array of flat indices, applying boundary modes to the multi-index

triu\_indicies(*n, k=0, m=None*)

Return the indices for the upper-triangle of an *(n, m)* array

tril\_indicies(*n, k=0, m=None*)

Return the indices for the lower-triangle of an *(n, m)* array

mask\_indicies(*n, mask\_func, k=0*)

Return the indicies to access *(n, n)* arrays, given a masking function

ascontiguousarray(*a, dtype=None, \*, like=None*)

Return a contiguous array *(ndim >= 1)* in memory (C order)

broadcast(*in1, in2, …*)

Produce an object that mimics broadcasting

broadcast\_arrays(*\*args, subok=False*)

Broadcast any number of arrays against each other

broadcast\_to(*array, shape, subok=False*)

Broadcast an array to a new shape

broadcast\_shapes(*\*args*)

Broadcast the input shapes into a single shape

fromiter(*iterable, dtype, count=-1, like=None*)

Create a new 1d array from an iterable object

isnan(*x, /, out=None, \*, where=True, casting=’same\_kind’, order=’K’, dtype=None, subok=True[, signature, extobj]*)

Test elementwise for NaNs and return result as boolean array.

mean(*a, axis=None, dtype=None, out=None, keepdims=<no value>, \*, where=<no value>*)

Compute the arithmetic mean along the specified axis

nanmean(*a, axis=None, dtype=None, out=None, keepdims=<no value>*)

Compute arithmetic mean along specified axis, ignoring NaNs

round(*a, decimals=0, out=None*)

Round an array to the given number of decimals

shares\_memory(*a, b, max\_work=None*)

Determine if two arrays share memory

ndenumerate(*arr*)

Multidimensional index iterator, yields pairs of array coordinates and values

ndindex(\**shape*)

An n-dimensional iterator object to index arrays.

nditer(*op, flags=None, op\_flags=None, op\_dtypes=None, order=’K’, casting=’safe’, op\_axes=None, itershape=None, buffersize=0*)

Efficient multi-dimensional iterator to iterate ober arrays.

select(*condlist, choicelist, default=0*)

Return array drawn from elements in *choicelist*, depending on *condlist*.

partition(*a, kth, axis=-1, kind=’introselect’, order=None*)

Return a copy of the array, with elements rearranged such that the value of the element in *k*-th position is in the position it would be in in a sorted array. Elements larger/smaller are moved behind/before. *k-*th may be an array.

argpartition(*a, kth, axis=-1, kind=’introselect’, order=None*)

Perform an indirect partition along the given axis. Returns an array of indices of the same shape as *a* that index data along the given axis in partitioned order. *k*-th may be an array.

take\_along\_axis(*arr, indices, axis*)

Take values from the input array by matching 1d index and data slices. *argsort()* and *argpartition()* produce suitable indices for this function.

put\_along\_axis(*arr, indices, values, axis*)

Put values into destination array by matching 1d index and data slices. *argsort()* and *argpartition()* produce suitable indices for this function.

bincount(*x, weights=None, minlength=0*)

Count number of occurrences of each value in array of non-negative integers

amax(*a, axis=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>*)

Return the maximum of an array, or maximum along an axis

amin(*a, axis=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>*)

Return the minimum of an array, or minimum along an axis

average(*a, axis=None, weights=None, returned=False*)

Compute the weighted average along the specified axis

cov(*m, y=None, rowvar=True, bias=False, ddof=None, fweights=one, aweights=None, \*, dtype=None*)

Estimate a covariance matrix, given data and weights

corrcoef(*x, y=None, rowvar=True, bias=<no value>, ddof=<no value>, \*, dtype=None*)

Return Pearson product-moment correlation coefficents

lib.stride\_tricks.sliding\_window\_view(*x, window\_shape, axis=None, \*, subok=False, writeable=False*)

Create a sliding window view into the array with the given window shape. Window slides across all dimensions of the array, and extracts subsets of the array at all window positions.

lib.stride\_tricks.as\_strided(*x, shape=None, strides=None, subok=False, writable=True*)

Create a view into the array with the given shape and strides. This function manipulates internal data structure of ndarray, and can result in corruption/crashes if used incorrectly.

core.records.fromarrays(*arrayList, dtype=None, shape=None, formats=None, names=None, titles=None, aligned=False, byteorder=None*)

Create a record array from a (flat) list of arrays

array\_equiv(*a1, a2*)

Returns True if input arrays are shape consistent, and all elements are equal. Shape consistent meaning one array can be broadcast into the shape of the other.

sort\_complex(*a*)

Sort a complex array using the real part first, then the imaginary part

around(*a, decimals=0, out=None*)

Evenly round to the given number of decimals

diff(*a, n=1, axis=-1, prepend=<no value>, append=<no value>*)

Calculate the n-th discrete difference along the given axis

ediff1d(*ary, to\_end=None, to\_begin=None*)

Differences between consecutive elements of an array

gradient(*f, \*varargs, axis=None, edge\_order=1*)

Gradient of an n-dimensional array

log(*x*, */*, *out=None*, *\**, *where=True*, *casting='same\_kind'*, *order='K'*, *dtype=None*, *subok=True*[, *signature*, *extobj*])

Natural logarithm, element-wise

logaddexp(*x1, x2, /, out=None, \*, where=True, casting='same\_kind', order='K', dtype=None, subok=True[, signature, extobj])*

Natural logarithm of the sum of exponentials of the inputs

logaddexp2(*x1, x2, /, out=None, \*, where=True, casting='same\_kind', order='K', dtype=None, subok=True[, signature, extobj])*

Base-2 Logarithm of the sum of exponentials of the input

mgrid

ogrid

r\_

Translates slice objects to concatenation along the first axis. If the index expression contains comma-seperated-arrays, stack them along their first axis. Or, if the index expression contains slice notation or scalars, then create a 1d array with a range indicated by slice notation.

c\_

Translate slice objects to concatenation along the second axis

s\_

A nicer way to build up index tuples for arrays

ix\_

Construct an open mesh from multiple sequences

# [numpy.char](https://numpy.org/doc/stable/reference/routines.char.html)

|  |  |
| --- | --- |
| [**add**](https://numpy.org/doc/stable/reference/generated/numpy.char.add.html#numpy.char.add)(x1, x2) | Return element-wise string concatenation for two arrays of str or unicode. |
| [**multiply**](https://numpy.org/doc/stable/reference/generated/numpy.char.multiply.html#numpy.char.multiply)(a, i) | Return (a \* i), that is string multiple concatenation, element-wise. |
| [**mod**](https://numpy.org/doc/stable/reference/generated/numpy.char.mod.html#numpy.char.mod)(a, values) | Return (a % i), that is pre-Python 2.6 string formatting (interpolation), element-wise for a pair of array\_likes of str or unicode. |
| [**capitalize**](https://numpy.org/doc/stable/reference/generated/numpy.char.capitalize.html#numpy.char.capitalize)(a) | Return a copy of *a* with only the first character of each element capitalized. |
| [**center**](https://numpy.org/doc/stable/reference/generated/numpy.char.center.html#numpy.char.center)(a, width[, fillchar]) | Return a copy of *a* with its elements centered in a string of length *width*. |
| [**decode**](https://numpy.org/doc/stable/reference/generated/numpy.char.decode.html#numpy.char.decode)(a[, encoding, errors]) | Calls *str.decode* element-wise. |
| [**encode**](https://numpy.org/doc/stable/reference/generated/numpy.char.encode.html#numpy.char.encode)(a[, encoding, errors]) | Calls *str.encode* element-wise. |
| [**expandtabs**](https://numpy.org/doc/stable/reference/generated/numpy.char.expandtabs.html#numpy.char.expandtabs)(a[, tabsize]) | Return a copy of each string element where all tab characters are replaced by one or more spaces. |
| [**join**](https://numpy.org/doc/stable/reference/generated/numpy.char.join.html#numpy.char.join)(sep, seq) | Return a string which is the concatenation of the strings in the sequence *seq*. |
| [**ljust**](https://numpy.org/doc/stable/reference/generated/numpy.char.ljust.html#numpy.char.ljust)(a, width[, fillchar]) | Return an array with the elements of *a* left-justified in a string of length *width*. |
| [**lower**](https://numpy.org/doc/stable/reference/generated/numpy.char.lower.html#numpy.char.lower)(a) | Return an array with the elements converted to lowercase. |
| [**lstrip**](https://numpy.org/doc/stable/reference/generated/numpy.char.lstrip.html#numpy.char.lstrip)(a[, chars]) | For each element in *a*, return a copy with the leading characters removed. |
| [**partition**](https://numpy.org/doc/stable/reference/generated/numpy.char.partition.html#numpy.char.partition)(a, sep) | Partition each element in *a* around *sep*. |
| [**replace**](https://numpy.org/doc/stable/reference/generated/numpy.char.replace.html#numpy.char.replace)(a, old, new[, count]) | For each element in *a*, return a copy of the string with all occurrences of substring *old* replaced by *new*. |
| [**rjust**](https://numpy.org/doc/stable/reference/generated/numpy.char.rjust.html#numpy.char.rjust)(a, width[, fillchar]) | Return an array with the elements of *a* right-justified in a string of length *width*. |
| [**rpartition**](https://numpy.org/doc/stable/reference/generated/numpy.char.rpartition.html#numpy.char.rpartition)(a, sep) | Partition (split) each element around the right-most separator. |
| [**rsplit**](https://numpy.org/doc/stable/reference/generated/numpy.char.rsplit.html#numpy.char.rsplit)(a[, sep, maxsplit]) | For each element in *a*, return a list of the words in the string, using *sep* as the delimiter string. |
| [**rstrip**](https://numpy.org/doc/stable/reference/generated/numpy.char.rstrip.html#numpy.char.rstrip)(a[, chars]) | For each element in *a*, return a copy with the trailing characters removed. |
| [**split**](https://numpy.org/doc/stable/reference/generated/numpy.char.split.html#numpy.char.split)(a[, sep, maxsplit]) | For each element in *a*, return a list of the words in the string, using *sep* as the delimiter string. |
| [**splitlines**](https://numpy.org/doc/stable/reference/generated/numpy.char.splitlines.html#numpy.char.splitlines)(a[, keepends]) | For each element in *a*, return a list of the lines in the element, breaking at line boundaries. |
| [**strip**](https://numpy.org/doc/stable/reference/generated/numpy.char.strip.html#numpy.char.strip)(a[, chars]) | For each element in *a*, return a copy with the leading and trailing characters removed. |
| [**swapcase**](https://numpy.org/doc/stable/reference/generated/numpy.char.swapcase.html#numpy.char.swapcase)(a) | Return element-wise a copy of the string with uppercase characters converted to lowercase and vice versa. |
| [**title**](https://numpy.org/doc/stable/reference/generated/numpy.char.title.html#numpy.char.title)(a) | Return element-wise title cased version of string or unicode. |
| [**translate**](https://numpy.org/doc/stable/reference/generated/numpy.char.translate.html#numpy.char.translate)(a, table[, deletechars]) | For each element in *a*, return a copy of the string where all characters occurring in the optional argument *deletechars* are removed, and the remaining characters have been mapped through the given translation table. |
| [**upper**](https://numpy.org/doc/stable/reference/generated/numpy.char.upper.html#numpy.char.upper)(a) | Return an array with the elements converted to uppercase. |
| [**zfill**](https://numpy.org/doc/stable/reference/generated/numpy.char.zfill.html#numpy.char.zfill)(a, width) | Return the numeric string left-filled with zeros |
| [**equal**](https://numpy.org/doc/stable/reference/generated/numpy.char.equal.html#numpy.char.equal)(x1, x2) | Return (x1 == x2) element-wise. |
| [**not\_equal**](https://numpy.org/doc/stable/reference/generated/numpy.char.not_equal.html#numpy.char.not_equal)(x1, x2) | Return (x1 != x2) element-wise. |
| [**greater\_equal**](https://numpy.org/doc/stable/reference/generated/numpy.char.greater_equal.html#numpy.char.greater_equal)(x1, x2) | Return (x1 >= x2) element-wise. |
| [**less\_equal**](https://numpy.org/doc/stable/reference/generated/numpy.char.less_equal.html#numpy.char.less_equal)(x1, x2) | Return (x1 <= x2) element-wise. |
| [**greater**](https://numpy.org/doc/stable/reference/generated/numpy.char.greater.html#numpy.char.greater)(x1, x2) | Return (x1 > x2) element-wise. |
| [**less**](https://numpy.org/doc/stable/reference/generated/numpy.char.less.html#numpy.char.less)(x1, x2) | Return (x1 < x2) element-wise. |
| [**compare\_chararrays**](https://numpy.org/doc/stable/reference/generated/numpy.char.compare_chararrays.html#numpy.char.compare_chararrays)(a, b, cmp\_op, rstrip) | Performs element-wise comparison of two string arrays using the comparison operator specified by *cmp\_op*. |
| [**count**](https://numpy.org/doc/stable/reference/generated/numpy.char.count.html#numpy.char.count)(a, sub[, start, end]) | Returns an array with the number of non-overlapping occurrences of substring *sub* in the range [*start*, *end*]. |
| [**endswith**](https://numpy.org/doc/stable/reference/generated/numpy.char.endswith.html#numpy.char.endswith)(a, suffix[, start, end]) | Returns a boolean array which is *True* where the string element in *a* ends with *suffix*, otherwise *False*. |
| [**find**](https://numpy.org/doc/stable/reference/generated/numpy.char.find.html#numpy.char.find)(a, sub[, start, end]) | For each element, return the lowest index in the string where substring *sub* is found. |
| [**index**](https://numpy.org/doc/stable/reference/generated/numpy.char.index.html#numpy.char.index)(a, sub[, start, end]) | Like [**find**](https://numpy.org/doc/stable/reference/generated/numpy.char.find.html#numpy.char.find), but raises *ValueError* when the substring is not found. |
| [**isalpha**](https://numpy.org/doc/stable/reference/generated/numpy.char.isalpha.html#numpy.char.isalpha)(a) | Returns true for each element if all characters in the string are alphabetic and there is at least one character, false otherwise. |
| [**isalnum**](https://numpy.org/doc/stable/reference/generated/numpy.char.isalnum.html#numpy.char.isalnum)(a) | Returns true for each element if all characters in the string are alphanumeric and there is at least one character, false otherwise. |
| [**isdecimal**](https://numpy.org/doc/stable/reference/generated/numpy.char.isdecimal.html#numpy.char.isdecimal)(a) | For each element, return True if there are only decimal characters in the element. |
| [**isdigit**](https://numpy.org/doc/stable/reference/generated/numpy.char.isdigit.html#numpy.char.isdigit)(a) | Returns true for each element if all characters in the string are digits and there is at least one character, false otherwise. |
| [**islower**](https://numpy.org/doc/stable/reference/generated/numpy.char.islower.html#numpy.char.islower)(a) | Returns true for each element if all cased characters in the string are lowercase and there is at least one cased character, false otherwise. |
| [**isnumeric**](https://numpy.org/doc/stable/reference/generated/numpy.char.isnumeric.html#numpy.char.isnumeric)(a) | For each element, return True if there are only numeric characters in the element. |
| [**isspace**](https://numpy.org/doc/stable/reference/generated/numpy.char.isspace.html#numpy.char.isspace)(a) | Returns true for each element if there are only whitespace characters in the string and there is at least one character, false otherwise. |
| [**istitle**](https://numpy.org/doc/stable/reference/generated/numpy.char.istitle.html#numpy.char.istitle)(a) | Returns true for each element if the element is a titlecased string and there is at least one character, false otherwise. |
| [**isupper**](https://numpy.org/doc/stable/reference/generated/numpy.char.isupper.html#numpy.char.isupper)(a) | Returns true for each element if all cased characters in the string are uppercase and there is at least one character, false otherwise. |
| [**rfind**](https://numpy.org/doc/stable/reference/generated/numpy.char.rfind.html#numpy.char.rfind)(a, sub[, start, end]) | For each element in *a*, return the highest index in the string where substring *sub* is found, such that *sub* is contained within [*start*, *end*]. |
| [**rindex**](https://numpy.org/doc/stable/reference/generated/numpy.char.rindex.html#numpy.char.rindex)(a, sub[, start, end]) | Like **[rfind](https://numpy.org/doc/stable/reference/generated/numpy.char.rfind.html" \l "numpy.char.rfind" \o "numpy.char.rfind)**, but raises *ValueError* when the substring *sub* is not found. |
| [**startswith**](https://numpy.org/doc/stable/reference/generated/numpy.char.startswith.html#numpy.char.startswith)(a, prefix[, start, end]) | Returns a boolean array which is *True* where the string element in *a* starts with *prefix*, otherwise *False*. |
| [**str\_len**](https://numpy.org/doc/stable/reference/generated/numpy.char.str_len.html#numpy.char.str_len)(a) | Return len(a) element-wise. |

|  |  |
| --- | --- |
| [**array**](https://numpy.org/doc/stable/reference/generated/numpy.char.array.html#numpy.char.array)(obj[, itemsize, copy, unicode, order]) | Create a **[chararray](https://numpy.org/doc/stable/reference/generated/numpy.char.chararray.html" \l "numpy.char.chararray" \o "numpy.char.chararray)**. |
| [**asarray**](https://numpy.org/doc/stable/reference/generated/numpy.char.asarray.html#numpy.char.asarray)(obj[, itemsize, unicode, order]) | Convert the input to a **[chararray](https://numpy.org/doc/stable/reference/generated/numpy.char.chararray.html" \l "numpy.char.chararray" \o "numpy.char.chararray)**, copying the data only if necessary. |
| [**chararray**](https://numpy.org/doc/stable/reference/generated/numpy.char.chararray.html#numpy.char.chararray)(shape[, itemsize, unicode, …]) | Provides a convenient view on arrays of string and unicode values. |

# numpy topics

## broadcasting

Extends <>

Rules of broadcasting:

1. All input arrays with ndim smaller than the input array of largest ndim, have 1’s prepended to their shapes.
2. The size in each dimension of the output shape is the maximum of all the input sizes in that dimension.
3. An input can be used in the calculation if its size in a particular dimension either matches the output size in that dimension, or has value exactly 1.
4. If an input has a dimension size of 1 in its shape, the first data entry in that dimension will be used for all calculations along that dimension. In other words, the stepping machinery of the ufunc will simply not step along that dimension (the stride will be 0 for that dimension).

That is, arrays are broadcastable to the same shape if one of the following is true:

1. The arrays all have exactly the same shape.
2. The arrays all have the same number of dimensions and the length of each dimensions is either a common length or 1.
3. The arrays that have too few dimensions can have their shapes prepended with a dimension of length 1 to satisfy property 2.

b[:, None] *or* b[:, np.newaxis]

Expand dimensions of resulting selection by one dimension.

...

As many ‘:’ as needed

# [numpy.ufunc](https://numpy.org/doc/stable/reference/ufuncs.html)

(Universal Functions)

Function that operates on *ndarray* in an element-by-element fashion. Performs standard broadcasting on input arrays. These are vectorized wrappers. In numpy, ufuncs are instances of *numpy.ufunc*.

## Attributes

ufunc.nin Number of inputs

ufunc.nout Number of outputs

ufunc.nargs Number of arguments

ufunc.ntypes Number of types

ufunc.types List of types grouped *input->output*

ufunc.identity

ufunc.signature How the dimensions of each input/output are split into core/loop dims

## Methods

ufuncs which take two input arguments and return one output argument have the five methods (see below)

ufunc.reduce(*array[, axis, dtype, out, …]*)

Reduce arrays dimensions by one, by applying *ufunc* along one axis

ufunc.accumulate(*array[, axis, dtype, out]*)

Accumulate the results of applying the operator to all elements

ufunc.reduceat(*array, indices[, axis, …]*)

Perform a (local) reduce with specified slices over a single axis

ufunc.outer(*A, B, /, \*\*kwargs*)

Apply the *ufunc* to all pairs *(a, b)* with *a* in *A* and *b* in *B*

ufunc.at(*a, indices[, b]*)

Performs unbuffered in-place operation on operand *a* for elements specified by *indices*

## List of ufuncs

|  |  |
| --- | --- |
| [**add**](https://numpy.org/doc/stable/reference/generated/numpy.add.html#numpy.add)(x1, x2, /[, out, where, casting, order, …]) | Add arguments element-wise. |
| [**subtract**](https://numpy.org/doc/stable/reference/generated/numpy.subtract.html#numpy.subtract)(x1, x2, /[, out, where, casting, …]) | Subtract arguments, element-wise. |
| [**multiply**](https://numpy.org/doc/stable/reference/generated/numpy.multiply.html#numpy.multiply)(x1, x2, /[, out, where, casting, …]) | Multiply arguments element-wise. |
| [**matmul**](https://numpy.org/doc/stable/reference/generated/numpy.matmul.html#numpy.matmul)(x1, x2, /[, out, casting, order, …]) | Matrix product of two arrays. |
| [**divide**](https://numpy.org/doc/stable/reference/generated/numpy.divide.html#numpy.divide)(x1, x2, /[, out, where, casting, …]) | Returns a true division of the inputs, element-wise. |
| [**logaddexp**](https://numpy.org/doc/stable/reference/generated/numpy.logaddexp.html#numpy.logaddexp)(x1, x2, /[, out, where, casting, …]) | Logarithm of the sum of exponentiations of the inputs. |
| [**logaddexp2**](https://numpy.org/doc/stable/reference/generated/numpy.logaddexp2.html#numpy.logaddexp2)(x1, x2, /[, out, where, casting, …]) | Logarithm of the sum of exponentiations of the inputs in base-2. |
| [**true\_divide**](https://numpy.org/doc/stable/reference/generated/numpy.true_divide.html#numpy.true_divide)(x1, x2, /[, out, where, …]) | Returns a true division of the inputs, element-wise. |
| [**floor\_divide**](https://numpy.org/doc/stable/reference/generated/numpy.floor_divide.html#numpy.floor_divide)(x1, x2, /[, out, where, …]) | Return the largest integer smaller or equal to the division of the inputs. |
| [**negative**](https://numpy.org/doc/stable/reference/generated/numpy.negative.html#numpy.negative)(x, /[, out, where, casting, order, …]) | Numerical negative, element-wise. |
| [**positive**](https://numpy.org/doc/stable/reference/generated/numpy.positive.html#numpy.positive)(x, /[, out, where, casting, order, …]) | Numerical positive, element-wise. |
| [**power**](https://numpy.org/doc/stable/reference/generated/numpy.power.html#numpy.power)(x1, x2, /[, out, where, casting, …]) | First array elements raised to powers from second array, element-wise. |
| [**float\_power**](https://numpy.org/doc/stable/reference/generated/numpy.float_power.html#numpy.float_power)(x1, x2, /[, out, where, …]) | First array elements raised to powers from second array, element-wise. |
| [**remainder**](https://numpy.org/doc/stable/reference/generated/numpy.remainder.html#numpy.remainder)(x1, x2, /[, out, where, casting, …]) | Return element-wise remainder of division. |
| [**mod**](https://numpy.org/doc/stable/reference/generated/numpy.mod.html#numpy.mod)(x1, x2, /[, out, where, casting, order, …]) | Return element-wise remainder of division. |
| [**fmod**](https://numpy.org/doc/stable/reference/generated/numpy.fmod.html#numpy.fmod)(x1, x2, /[, out, where, casting, …]) | Return the element-wise remainder of division. |
| [**divmod**](https://numpy.org/doc/stable/reference/generated/numpy.divmod.html#numpy.divmod)(x1, x2[, out1, out2], / [[, out, …]) | Return element-wise quotient and remainder simultaneously. |
| [**absolute**](https://numpy.org/doc/stable/reference/generated/numpy.absolute.html#numpy.absolute)(x, /[, out, where, casting, order, …]) | Calculate the absolute value element-wise. |
| [**fabs**](https://numpy.org/doc/stable/reference/generated/numpy.fabs.html#numpy.fabs)(x, /[, out, where, casting, order, …]) | Compute the absolute values element-wise. |
| [**rint**](https://numpy.org/doc/stable/reference/generated/numpy.rint.html#numpy.rint)(x, /[, out, where, casting, order, …]) | Round elements of the array to the nearest integer. |
| [**sign**](https://numpy.org/doc/stable/reference/generated/numpy.sign.html#numpy.sign)(x, /[, out, where, casting, order, …]) | Returns an element-wise indication of the sign of a number. |
| [**heaviside**](https://numpy.org/doc/stable/reference/generated/numpy.heaviside.html#numpy.heaviside)(x1, x2, /[, out, where, casting, …]) | Compute the Heaviside step function. |
| [**conj**](https://numpy.org/doc/stable/reference/generated/numpy.conj.html#numpy.conj)(x, /[, out, where, casting, order, …]) | Return the complex conjugate, element-wise. |
| [**conjugate**](https://numpy.org/doc/stable/reference/generated/numpy.conjugate.html#numpy.conjugate)(x, /[, out, where, casting, …]) | Return the complex conjugate, element-wise. |
| [**exp**](https://numpy.org/doc/stable/reference/generated/numpy.exp.html#numpy.exp)(x, /[, out, where, casting, order, …]) | Calculate the exponential of all elements in the input array. |
| [**exp2**](https://numpy.org/doc/stable/reference/generated/numpy.exp2.html#numpy.exp2)(x, /[, out, where, casting, order, …]) | Calculate *2\*\*p* for all *p* in the input array. |
| [**log**](https://numpy.org/doc/stable/reference/generated/numpy.log.html#numpy.log)(x, /[, out, where, casting, order, …]) | Natural logarithm, element-wise. |
| [**log2**](https://numpy.org/doc/stable/reference/generated/numpy.log2.html#numpy.log2)(x, /[, out, where, casting, order, …]) | Base-2 logarithm of *x*. |
| [**log10**](https://numpy.org/doc/stable/reference/generated/numpy.log10.html#numpy.log10)(x, /[, out, where, casting, order, …]) | Return the base 10 logarithm of the input array, element-wise. |
| [**expm1**](https://numpy.org/doc/stable/reference/generated/numpy.expm1.html#numpy.expm1)(x, /[, out, where, casting, order, …]) | Calculate exp(x) - 1 for all elements in the array. |
| [**log1p**](https://numpy.org/doc/stable/reference/generated/numpy.log1p.html#numpy.log1p)(x, /[, out, where, casting, order, …]) | Return the natural logarithm of one plus the input array, element-wise. |
| [**sqrt**](https://numpy.org/doc/stable/reference/generated/numpy.sqrt.html#numpy.sqrt)(x, /[, out, where, casting, order, …]) | Return the non-negative square-root of an array, element-wise. |
| [**square**](https://numpy.org/doc/stable/reference/generated/numpy.square.html#numpy.square)(x, /[, out, where, casting, order, …]) | Return the element-wise square of the input. |
| [**cbrt**](https://numpy.org/doc/stable/reference/generated/numpy.cbrt.html#numpy.cbrt)(x, /[, out, where, casting, order, …]) | Return the cube-root of an array, element-wise. |
| [**reciprocal**](https://numpy.org/doc/stable/reference/generated/numpy.reciprocal.html#numpy.reciprocal)(x, /[, out, where, casting, …]) | Return the reciprocal of the argument, element-wise. |
| [**gcd**](https://numpy.org/doc/stable/reference/generated/numpy.gcd.html#numpy.gcd)(x1, x2, /[, out, where, casting, order, …]) | Returns the greatest common divisor of |x1| and |x2| |
| [**lcm**](https://numpy.org/doc/stable/reference/generated/numpy.lcm.html#numpy.lcm)(x1, x2, /[, out, where, casting, order, …]) | Returns the lowest common multiple of |x1| and |x2| |

|  |  |
| --- | --- |
| [**sin**](https://numpy.org/doc/stable/reference/generated/numpy.sin.html#numpy.sin)(x, /[, out, where, casting, order, …]) | Trigonometric sine, element-wise. |
| [**cos**](https://numpy.org/doc/stable/reference/generated/numpy.cos.html#numpy.cos)(x, /[, out, where, casting, order, …]) | Cosine element-wise. |
| [**tan**](https://numpy.org/doc/stable/reference/generated/numpy.tan.html#numpy.tan)(x, /[, out, where, casting, order, …]) | Compute tangent element-wise. |
| [**arcsin**](https://numpy.org/doc/stable/reference/generated/numpy.arcsin.html#numpy.arcsin)(x, /[, out, where, casting, order, …]) | Inverse sine, element-wise. |
| [**arccos**](https://numpy.org/doc/stable/reference/generated/numpy.arccos.html#numpy.arccos)(x, /[, out, where, casting, order, …]) | Trigonometric inverse cosine, element-wise. |
| [**arctan**](https://numpy.org/doc/stable/reference/generated/numpy.arctan.html#numpy.arctan)(x, /[, out, where, casting, order, …]) | Trigonometric inverse tangent, element-wise. |
| [**arctan2**](https://numpy.org/doc/stable/reference/generated/numpy.arctan2.html#numpy.arctan2)(x1, x2, /[, out, where, casting, …]) | Element-wise arc tangent of x1/x2 choosing the quadrant correctly. |
| [**hypot**](https://numpy.org/doc/stable/reference/generated/numpy.hypot.html#numpy.hypot)(x1, x2, /[, out, where, casting, …]) | Given the “legs” of a right triangle, return its hypotenuse. |
| [**sinh**](https://numpy.org/doc/stable/reference/generated/numpy.sinh.html#numpy.sinh)(x, /[, out, where, casting, order, …]) | Hyperbolic sine, element-wise. |
| [**cosh**](https://numpy.org/doc/stable/reference/generated/numpy.cosh.html#numpy.cosh)(x, /[, out, where, casting, order, …]) | Hyperbolic cosine, element-wise. |
| [**tanh**](https://numpy.org/doc/stable/reference/generated/numpy.tanh.html#numpy.tanh)(x, /[, out, where, casting, order, …]) | Compute hyperbolic tangent element-wise. |
| [**arcsinh**](https://numpy.org/doc/stable/reference/generated/numpy.arcsinh.html#numpy.arcsinh)(x, /[, out, where, casting, order, …]) | Inverse hyperbolic sine element-wise. |
| [**arccosh**](https://numpy.org/doc/stable/reference/generated/numpy.arccosh.html#numpy.arccosh)(x, /[, out, where, casting, order, …]) | Inverse hyperbolic cosine, element-wise. |
| [**arctanh**](https://numpy.org/doc/stable/reference/generated/numpy.arctanh.html#numpy.arctanh)(x, /[, out, where, casting, order, …]) | Inverse hyperbolic tangent element-wise. |
| [**degrees**](https://numpy.org/doc/stable/reference/generated/numpy.degrees.html#numpy.degrees)(x, /[, out, where, casting, order, …]) | Convert angles from radians to degrees. |
| [**radians**](https://numpy.org/doc/stable/reference/generated/numpy.radians.html#numpy.radians)(x, /[, out, where, casting, order, …]) | Convert angles from degrees to radians. |
| [**deg2rad**](https://numpy.org/doc/stable/reference/generated/numpy.deg2rad.html#numpy.deg2rad)(x, /[, out, where, casting, order, …]) | Convert angles from degrees to radians. |
| [**rad2deg**](https://numpy.org/doc/stable/reference/generated/numpy.rad2deg.html#numpy.rad2deg)(x, /[, out, where, casting, order, …]) | Convert angles from radians to degrees. |

|  |  |
| --- | --- |
| [**bitwise\_and**](https://numpy.org/doc/stable/reference/generated/numpy.bitwise_and.html#numpy.bitwise_and)(x1, x2, /[, out, where, …]) | Compute the bit-wise AND of two arrays element-wise. |
| [**bitwise\_or**](https://numpy.org/doc/stable/reference/generated/numpy.bitwise_or.html#numpy.bitwise_or)(x1, x2, /[, out, where, casting, …]) | Compute the bit-wise OR of two arrays element-wise. |
| [**bitwise\_xor**](https://numpy.org/doc/stable/reference/generated/numpy.bitwise_xor.html#numpy.bitwise_xor)(x1, x2, /[, out, where, …]) | Compute the bit-wise XOR of two arrays element-wise. |
| [**invert**](https://numpy.org/doc/stable/reference/generated/numpy.invert.html#numpy.invert)(x, /[, out, where, casting, order, …]) | Compute bit-wise inversion, or bit-wise NOT, element-wise. |
| [**left\_shift**](https://numpy.org/doc/stable/reference/generated/numpy.left_shift.html#numpy.left_shift)(x1, x2, /[, out, where, casting, …]) | Shift the bits of an integer to the left. |
| [**right\_shift**](https://numpy.org/doc/stable/reference/generated/numpy.right_shift.html#numpy.right_shift)(x1, x2, /[, out, where, …]) | Shift the bits of an integer to the right. |

|  |  |
| --- | --- |
| [**greater**](https://numpy.org/doc/stable/reference/generated/numpy.greater.html#numpy.greater)(x1, x2, /[, out, where, casting, …]) | Return the truth value of (x1 > x2) element-wise. |
| [**greater\_equal**](https://numpy.org/doc/stable/reference/generated/numpy.greater_equal.html#numpy.greater_equal)(x1, x2, /[, out, where, …]) | Return the truth value of (x1 >= x2) element-wise. |
| [**less**](https://numpy.org/doc/stable/reference/generated/numpy.less.html#numpy.less)(x1, x2, /[, out, where, casting, …]) | Return the truth value of (x1 < x2) element-wise. |
| [**less\_equal**](https://numpy.org/doc/stable/reference/generated/numpy.less_equal.html#numpy.less_equal)(x1, x2, /[, out, where, casting, …]) | Return the truth value of (x1 <= x2) element-wise. |
| [**not\_equal**](https://numpy.org/doc/stable/reference/generated/numpy.not_equal.html#numpy.not_equal)(x1, x2, /[, out, where, casting, …]) | Return (x1 != x2) element-wise. |
| [**equal**](https://numpy.org/doc/stable/reference/generated/numpy.equal.html#numpy.equal)(x1, x2, /[, out, where, casting, …]) | Return (x1 == x2) element-wise. |
| [**logical\_and**](https://numpy.org/doc/stable/reference/generated/numpy.logical_and.html#numpy.logical_and)(x1, x2, /[, out, where, …]) | Compute the truth value of x1 AND x2 element-wise. |
| [**logical\_or**](https://numpy.org/doc/stable/reference/generated/numpy.logical_or.html#numpy.logical_or)(x1, x2, /[, out, where, casting, …]) | Compute the truth value of x1 OR x2 element-wise. |
| [**logical\_xor**](https://numpy.org/doc/stable/reference/generated/numpy.logical_xor.html#numpy.logical_xor)(x1, x2, /[, out, where, …]) | Compute the truth value of x1 XOR x2, element-wise. |
| [**logical\_not**](https://numpy.org/doc/stable/reference/generated/numpy.logical_not.html#numpy.logical_not)(x, /[, out, where, casting, …]) | Compute the truth value of NOT x element-wise. |
| [**maximum**](https://numpy.org/doc/stable/reference/generated/numpy.maximum.html#numpy.maximum)(x1, x2, /[, out, where, casting, …]) | Element-wise maximum of array elements. |
| [**minimum**](https://numpy.org/doc/stable/reference/generated/numpy.minimum.html#numpy.minimum)(x1, x2, /[, out, where, casting, …]) | Element-wise minimum of array elements. |
| [**fmax**](https://numpy.org/doc/stable/reference/generated/numpy.fmax.html#numpy.fmax)(x1, x2, /[, out, where, casting, …]) | Element-wise maximum of array elements. |
| [**fmin**](https://numpy.org/doc/stable/reference/generated/numpy.fmin.html#numpy.fmin)(x1, x2, /[, out, where, casting, …]) | Element-wise minimum of array elements. |

|  |  |
| --- | --- |
| [**isfinite**](https://numpy.org/doc/stable/reference/generated/numpy.isfinite.html#numpy.isfinite)(x, /[, out, where, casting, order, …]) | Test element-wise for finiteness (not infinity or not Not a Number). |
| [**isinf**](https://numpy.org/doc/stable/reference/generated/numpy.isinf.html#numpy.isinf)(x, /[, out, where, casting, order, …]) | Test element-wise for positive or negative infinity. |
| [**isnan**](https://numpy.org/doc/stable/reference/generated/numpy.isnan.html#numpy.isnan)(x, /[, out, where, casting, order, …]) | Test element-wise for NaN and return result as a boolean array. |
| [**isnat**](https://numpy.org/doc/stable/reference/generated/numpy.isnat.html#numpy.isnat)(x, /[, out, where, casting, order, …]) | Test element-wise for NaT (not a time) and return result as a boolean array. |
| [**fabs**](https://numpy.org/doc/stable/reference/generated/numpy.fabs.html#numpy.fabs)(x, /[, out, where, casting, order, …]) | Compute the absolute values element-wise. |
| [**signbit**](https://numpy.org/doc/stable/reference/generated/numpy.signbit.html#numpy.signbit)(x, /[, out, where, casting, order, …]) | Returns element-wise True where signbit is set (less than zero). |
| [**copysign**](https://numpy.org/doc/stable/reference/generated/numpy.copysign.html#numpy.copysign)(x1, x2, /[, out, where, casting, …]) | Change the sign of x1 to that of x2, element-wise. |
| [**nextafter**](https://numpy.org/doc/stable/reference/generated/numpy.nextafter.html#numpy.nextafter)(x1, x2, /[, out, where, casting, …]) | Return the next floating-point value after x1 towards x2, element-wise. |
| [**spacing**](https://numpy.org/doc/stable/reference/generated/numpy.spacing.html#numpy.spacing)(x, /[, out, where, casting, order, …]) | Return the distance between x and the nearest adjacent number. |
| [**modf**](https://numpy.org/doc/stable/reference/generated/numpy.modf.html#numpy.modf)(x[, out1, out2], / [[, out, where, …]) | Return the fractional and integral parts of an array, element-wise. |
| [**ldexp**](https://numpy.org/doc/stable/reference/generated/numpy.ldexp.html#numpy.ldexp)(x1, x2, /[, out, where, casting, …]) | Returns x1 \* 2\*\*x2, element-wise. |
| [**frexp**](https://numpy.org/doc/stable/reference/generated/numpy.frexp.html#numpy.frexp)(x[, out1, out2], / [[, out, where, …]) | Decompose the elements of x into mantissa and twos exponent. |
| [**fmod**](https://numpy.org/doc/stable/reference/generated/numpy.fmod.html#numpy.fmod)(x1, x2, /[, out, where, casting, …]) | Return the element-wise remainder of division. |
| [**floor**](https://numpy.org/doc/stable/reference/generated/numpy.floor.html#numpy.floor)(x, /[, out, where, casting, order, …]) | Return the floor of the input, element-wise. |
| [**ceil**](https://numpy.org/doc/stable/reference/generated/numpy.ceil.html#numpy.ceil)(x, /[, out, where, casting, order, …]) | Return the ceiling of the input, element-wise. |
| [**trunc**](https://numpy.org/doc/stable/reference/generated/numpy.trunc.html#numpy.trunc)(x, /[, out, where, casting, order, …]) | Return the truncated value of the input, element-wise. |

# einsum() Examples

Let A and B be two 1D arrays of compatible shapes (meaning the lengths of the axes we pair together either equal, or one of them has length 1):

|  |  |  |
| --- | --- | --- |
| **Call signature** | **NumPy equivalent** | **Description** |
| ('i', A) | A | returns a view of A |
| ('i->', A) | sum(A) | sums the values of A |
| ('i,i->i', A, B) | A \* B | element-wise multiplication of A and B |
| ('i,i', A, B) | inner(A, B) | inner product of A and B |
| ('i,j->ij', A, B) | outer(A, B) | outer product of A and B |

Now let A and B be two 2D arrays with compatible shapes:

|  |  |  |
| --- | --- | --- |
| **Call signature** | **NumPy equivalent** | **Description** |
| ('ij', A) | A | returns a view of A |
| ('ji', A) | A.T | view transpose of A |
| ('ii->i', A) | diag(A) | view main diagonal of A |
| ('ii', A) | trace(A) | sums main diagonal of A |
| ('ij->', A) | sum(A) | sums the values of A |
| ('ij->j', A) | sum(A, axis=0) | sum down the columns of A (across rows) |
| ('ij->i', A) | sum(A, axis=1) | sum horizontally along the rows of A |
| ('ij,ij->ij', A, B) | A \* B | element-wise multiplication of A and B |
| ('ij,ji->ij', A, B) | A \* B.T | element-wise multiplication of A and B.T |
| ('ij,jk', A, B) | dot(A, B) | matrix multiplication of A and B |
| ('ij,kj->ik', A, B) | inner(A, B) | inner product of A and B |
| ('ij,kj->ikj', A, B) | A[:, None] \* B | each row of A multiplied by B |
| ('ij,kl->ijkl', A, B) | A[:, :, None, None] \* B | each value of A multiplied by B |

# numpy.random

rand(*d0, d1, …, dn*)

Random value(s) in a given shape.

randn(*d0, d1, …, dn*)

Value(s) from the standard normal distribution in a given shape.

randint(*low*[*, high, size*])

Random integers in range *[low, high)*

random([*size*])

Random floats in the interval *[0, 1)*

choice(*a*[*, size=None, replace=True, p=None*])

Random sample from a given 1d array

bytes(*length*)

Random bytes

shuffle(*x*)

Shuffle contents of sequence in-place. (Use from instance of *default\_rng*)

permutations(*x*)

Randomly permute a sequence, or return a permuted range. If *x* is multi-dimensional, only the first index is shuffled. (Use from instance of *default\_rng*)

normal([*loc, scale, size*])

Random samples from a normal (Gaussian) distribution. Mean = *loc*, standard deviation = *scale*.

seed()

get\_state()

set\_state()

# [numpy.random.default\_rng](https://numpy.org/doc/stable/reference/random/generator.html#numpy.random.default_rng)

default\_rng()

Use methods provided by *default\_rng* in preference to those of *numpy.random*

|  |  |
| --- | --- |
| [**integers**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.integers.html#numpy.random.Generator.integers)(low[, high, size, dtype, endpoint]) | Return random integers from low (inclusive) to high (exclusive), or if endpoint=True, low (inclusive) to high (inclusive). |
| [**random**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.random.html#numpy.random.Generator.random)([size, dtype, out]) | Return random floats in the half-open interval [0.0, 1.0). |
| [**choice**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.choice.html#numpy.random.Generator.choice)(a[, size, replace, p, axis, shuffle]) | Generates a random sample from a given 1-D array |
| [**bytes**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.bytes.html#numpy.random.Generator.bytes)(length) | Return random bytes |

|  |  |
| --- | --- |
| [**shuffle**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.shuffle.html#numpy.random.Generator.shuffle)(x[, axis]) | Modify an array or sequence in-place by shuffling its contents. |
| [**permutation**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.permutation.html#numpy.random.Generator.permutation)(x[, axis]) | Randomly permute a sequence, or return a permuted range. |
| [**permuted**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.permuted.html#numpy.random.Generator.permuted)(x[, axis, out]) | Randomly permute x along axis axis. |

|  |  |
| --- | --- |
| [**beta**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.beta.html#numpy.random.Generator.beta)(a, b[, size]) | Draw samples from a Beta distribution. |
| [**binomial**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.binomial.html#numpy.random.Generator.binomial)(n, p[, size]) | Draw samples from a binomial distribution. |
| [**chisquare**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.chisquare.html#numpy.random.Generator.chisquare)(df[, size]) | Draw samples from a chi-square distribution. |
| [**dirichlet**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.dirichlet.html#numpy.random.Generator.dirichlet)(alpha[, size]) | Draw samples from the Dirichlet distribution. |
| [**exponential**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.exponential.html#numpy.random.Generator.exponential)([scale, size]) | Draw samples from an exponential distribution. |
| [**f**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.f.html#numpy.random.Generator.f)(dfnum, dfden[, size]) | Draw samples from an F distribution. |
| [**gamma**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.gamma.html#numpy.random.Generator.gamma)(shape[, scale, size]) | Draw samples from a Gamma distribution. |
| [**geometric**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.geometric.html#numpy.random.Generator.geometric)(p[, size]) | Draw samples from the geometric distribution. |
| [**gumbel**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.gumbel.html#numpy.random.Generator.gumbel)([loc, scale, size]) | Draw samples from a Gumbel distribution. |
| [**hypergeometric**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.hypergeometric.html#numpy.random.Generator.hypergeometric)(ngood, nbad, nsample[, size]) | Draw samples from a Hypergeometric distribution. |
| [**laplace**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.laplace.html#numpy.random.Generator.laplace)([loc, scale, size]) | Draw samples from the Laplace or double exponential distribution with specified location (or mean) and scale (decay). |
| [**logistic**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.logistic.html#numpy.random.Generator.logistic)([loc, scale, size]) | Draw samples from a logistic distribution. |
| [**lognormal**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.lognormal.html#numpy.random.Generator.lognormal)([mean, sigma, size]) | Draw samples from a log-normal distribution. |
| [**logseries**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.logseries.html#numpy.random.Generator.logseries)(p[, size]) | Draw samples from a logarithmic series distribution. |
| [**multinomial**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.multinomial.html#numpy.random.Generator.multinomial)(n, pvals[, size]) | Draw samples from a multinomial distribution. |
| [**multivariate\_hypergeometric**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.multivariate_hypergeometric.html#numpy.random.Generator.multivariate_hypergeometric)(colors, nsample) | Generate variates from a multivariate hypergeometric distribution. |
| [**multivariate\_normal**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.multivariate_normal.html#numpy.random.Generator.multivariate_normal)(mean, cov[, size, …]) | Draw random samples from a multivariate normal distribution. |
| [**negative\_binomial**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.negative_binomial.html#numpy.random.Generator.negative_binomial)(n, p[, size]) | Draw samples from a negative binomial distribution. |
| [**noncentral\_chisquare**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.noncentral_chisquare.html#numpy.random.Generator.noncentral_chisquare)(df, nonc[, size]) | Draw samples from a noncentral chi-square distribution. |
| [**noncentral\_f**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.noncentral_f.html#numpy.random.Generator.noncentral_f)(dfnum, dfden, nonc[, size]) | Draw samples from the noncentral F distribution. |
| [**normal**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.normal.html#numpy.random.Generator.normal)([loc, scale, size]) | Draw random samples from a normal (Gaussian) distribution. |
| [**pareto**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.pareto.html#numpy.random.Generator.pareto)(a[, size]) | Draw samples from a Pareto II or Lomax distribution with specified shape. |
| [**poisson**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.poisson.html#numpy.random.Generator.poisson)([lam, size]) | Draw samples from a Poisson distribution. |
| [**power**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.power.html#numpy.random.Generator.power)(a[, size]) | Draws samples in [0, 1] from a power distribution with positive exponent a - 1. |
| [**rayleigh**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.rayleigh.html#numpy.random.Generator.rayleigh)([scale, size]) | Draw samples from a Rayleigh distribution. |
| [**standard\_cauchy**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.standard_cauchy.html#numpy.random.Generator.standard_cauchy)([size]) | Draw samples from a standard Cauchy distribution with mode = 0. |
| [**standard\_exponential**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.standard_exponential.html#numpy.random.Generator.standard_exponential)([size, dtype, method, out]) | Draw samples from the standard exponential distribution. |
| [**standard\_gamma**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.standard_gamma.html#numpy.random.Generator.standard_gamma)(shape[, size, dtype, out]) | Draw samples from a standard Gamma distribution. |
| [**standard\_normal**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.standard_normal.html#numpy.random.Generator.standard_normal)([size, dtype, out]) | Draw samples from a standard Normal distribution (mean=0, stdev=1). |
| [**standard\_t**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.standard_t.html#numpy.random.Generator.standard_t)(df[, size]) | Draw samples from a standard Student’s t distribution with df degrees of freedom. |
| [**triangular**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.triangular.html#numpy.random.Generator.triangular)(left, mode, right[, size]) | Draw samples from the triangular distribution over the interval [left, right]. |
| [**uniform**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.uniform.html#numpy.random.Generator.uniform)([low, high, size]) | Draw samples from a uniform distribution. |
| [**vonmises**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.vonmises.html#numpy.random.Generator.vonmises)(mu, kappa[, size]) | Draw samples from a von Mises distribution. |
| [**wald**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.wald.html#numpy.random.Generator.wald)(mean, scale[, size]) | Draw samples from a Wald, or inverse Gaussian, distribution. |
| [**weibull**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.weibull.html#numpy.random.Generator.weibull)(a[, size]) | Draw samples from a Weibull distribution. |
| [**zipf**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.zipf.html#numpy.random.Generator.zipf)(a[, size]) | Draw samples from a Zipf distribution. |

# [numpy.ndarray](https://numpy.org/doc/stable/reference/arrays.ndarray.html)

## attributes

flags

*dict*, Information about memory layout of array

shape

*tuple* of array dimensions

strides

*tuple* of bytes to step in each direction when traversing array

ndim

*int*, number of array dimensions

data

*buffer* object, pointing to start of array data

size

*int*, number of elements in array

itemsize

*int*, length of one element in bytes

nbytes

*int*, total bytes consumed by array

base

*ndarray*, base object if memory is from some other object

T

*ndarray*, transposed array

real

*ndarray*, real part of the array

imag

*ndarray,* imaginary part of the array

flat

*numpy.flatiter* object, 1d iterator over array

ctypes

*ctypes* object, for interaction with ctypes module

## conversion

item(*\*args*)

Copy an element of an array to a python scalar and return it

tolist()

Return the array as an *a.ndim* levels deep nested list of python scalars

itemset(*\*args*)

Insert scalar into array. If last item of *args* is *item*, then equivalent to *a[args]=item*. *args* may only be size 1 if array is size one.

tostring([*order*])

deprecated.

tobytes(*order=’C’*)

Construct python bytes containing the raw data bytes in the array.

tofile(*fid[, sep=””, format=”%s”]*)

Write array to file. Binary format if *sep* is empty. Result can be read with *fromfile()*.

dump(*file*)

Dump a pickle of the array to a file.

dumps()

Return the pickle of the array as string.

astype(*dtype, order=’K’, casting=’unsafe’, subok=True, copy=True*)

Copy of the array, cast to a specific type.

byteswrap(*inplace=False*)

Swap the bytes of the array elements. Toggle between little/big-endian. re/im components swapped separately.

copy(*order=’C’*)

Return a copy of the array

view(*[dtype][, type]*)

New view of array with same data

getfield(*dtype, offset=0*)

Returns the field of the given array as a certain type.

setflags(*write=None, align=None, uic=None*)

Set array flags.

fill(*value*)

Fill the array with a scalar value

## manipulation

reshape(*shape, order=’C’*)

Return an array containing the same data with a new shape. One shape dimension can be *-1*, it will be inferred from the other dimensions.

resize(*new\_shape, refcheck=True*)

Change shape and size of array in-place

transpose(*\*axes*)

Returns a view of the array with axes transposed

swapaxes(*axis1, axis2*)

Return a view of the array with *axis1* and *axis2* interchanged

flatten(*order=’C’*)

Return a copy of the array collapsed into one dimension.

ravel(*order=’C’*)

Return a flattened array

squeeze(*axis=None*)

Remove axes of length one from *a*

## selection and manipulation

take(*indices, axis=None, out=None, mode=’raise’*)

Return an array formed from the elements of *a* at the given indices.

put(*indices, values, mode=’raise’*)

Set *a.flat[n] = values[n]* for all *n* in *indices*

repeat(*repeats, axis=None*)

Repeat elements of an array

choose(*choices, out=None, mode=’raise’*)

Use an index array to construct a new array from a set of choices.

sort(*axis=-1, kind=None, order=None*)

Sort array in-place

argsort(*axis=-1, kind=None, order=None*)

Return the indices that would sort this array

partition(*kth, axis=-1, kind=’introselect’, order=None*)

Rearranges the elements in the array such that the value of the element in *kth* position is in the position it would be in a sorted array. All elements smaller are moved before, and all greater or equal are moved after it. Order of partitioned elements is undefined.

argpartition(*kth, axis=-1, kind=’introselect’, order=None*)

Return the indices that would partition this array

searchsorted(*v, side=’left’, sorter=None*)

Find indices where elements of *v* should be inserted in *a* to maintain order

nonzero()

Return the indices of the elements that are non-zero

compress(*condition, axis=None, out=None*)

Return selected slices of this array along a given axis, for each index where *condition* evaluates to True

diagonal(*offset=0, axis1=0, axis2=1*)

Return specified diagonals

## calculation

max(*axis=None, out=None, keepdims=False, initial=<no value>, where=True*)

Return the maximum along a given axis

argmax(*axis=None, out=None*)

Return indices of the maximum values along the given axis

min(*axis=None, out=None, keepdims=False, initial=<no value>, where=True*)

Return the minimum along a given axis

ptp(*axis=None, out=None, keepdims=False*)

Peak to peak (max-min) value along a given axis.

clip(*min=None, max=None, out=None, \*\*kwargs*)

Return an array whose values are limited to *[min, max]*. One of *max, min* must be given.

conj()

Complex conjugate of all elements

round(*decimals=0, out=None*)

Return *a* with each element rounded to the given number of decimals

trace(*offset=0, axis1=0, axis2=1, dtype=None, out=None*)

Return the sum along diagonals of the array

sum(*axis=None, dtype=None, out=None, keepdims=False, initial=0, where=True*)

Return the sum of the array elements over the given axis

cumsum(*axis=None, dtype=None, out=None*)

Return the cumulative sum of the elements along the given axis

mean(*axis=None, dtype=None, out=None, keepdims=False, \*, where=True*)

Returns the average of the array elements along a given axis

var(*axis=None, dtype=None, out=None, ddof=0, keepdims=False, \*, where=True*)

Returns the variance of the array elements, along given axis

std(*axis=None, dtype=None, out=None, ddof=0, keepdims=False, \*, where=True*)

Returns the standard deviation of the array elements along given axis

prod(*axis=None, dtype=None, out=None, keepdims=False, initial=1, where=True*)

Return the product of the array elements over the given axis

cumprod(*axis=None, dtype=None, out=None*)

Return the cumulative product of the elements along the given axis

all(*axis=None, out=None, keepdims=False, \*, where=True*)

Returns True if all elements evaluate to True

any(*axis=None, out=None, keepdims=False, \*, where=True*)

Returns True if any of the elements evaluate to True

## arithmetic, matrix multiplication, comparison

\_\_lt\_\_()

\_\_le\_\_()

<...>

# [numpy.linalg](https://numpy.org/doc/stable/reference/routines.linalg.html)

## matrix and vector products

dot(*a, b*[*, out*])

Dot product of two arrays

multi\_dot(*arrays, \**[*, out*])

Dot product of multiple arrays in single function call

vdot(*a, b*)

Dot product of two vectors

inner(*a, b*)

Inner product of two arrays

outer(*a, b*[*, out*])

Outer product of two vectors

matmul(*x1, x2, /*[*, out, casting, order, …*])

Matrix product of two arrays

tensordot(*a, b*[*, axes*])

Tensor dot product along specified axis.

einsum(*subscripts, \*operands*[*, out, dtype, …*])

Evaluates the Einstein summation convention on the operands.

einsum\_path(*subscripts, \*operands*[*, optimize*])

Evaluates the lowest cost contraction order for an einsum expression.

matrix\_power(*a, n*)

Raise a square matrix to the (integer) power *n*

kron(*a, b*)

Kronecker product of two arrays

## decompositions

cholseky(*a*)

Cholseky decomposition

qr(*a*[*, mode*])

qr factorization of a matrix

svd(*a*[*, full\_matricies, compute\_uv, …*])

Single value decomposition

## eigenvalues

eig(*a*)

Compute the eigenvalues and right eigenvectors of a square array

eigh(*x*[*, p*])

Return the eigenvalues and eigenvectors of a complex Hermitian or real symmetric matrix

eigvals(*a*)

Compute the eigenvalues of a general matrix

eigvalsh(*a*[*, UPLO*])

Compute the eigenvalues of a complex Hermitian or real symmetric matrix

## norms and other numbers

norm(*x*[*, ord, axis, keepdims*])

Matrix or vector norm

cond(*x*[*, p*])

Compute the conditional number of a matrix

det(*a*)

Compute the determinant of an array

matrix\_rank(*M*[*, tol, hermitian*])

Return the matrix rand of an array uing SVD method

slogdet(*a*)

Compute the sign and (natural) logarithm of the determinant of an array

trace(*a*[*, offset, axis1, axis2, dtype, out*])

Return the sum along diagonals of the array

## solving, inverting

solve(*a, b*)

Solve a linear matrix equation, or system of linear scalar equations

tensorsolve(*a, b*[*, axes*])

Solve the tensor equation *ax = b* for *x*

lstsq(*a, b*[*, rcond*])

Return the least-squares solution to a linear matrix equation

inv(*a*)

Compute the (multiplicative) inverse of a matrix

pinv(*a*[*, rcond, hermitian*])

Compute the (Moore-Penrose) pseudo-inverse of a matrix

tensorinv(*a*[*, ind*])

Compute the inverse of an N-dimensional array

# [numpy.polynomial](https://numpy.org/doc/stable/reference/routines.polynomials.package.html#module-numpy.polynomial)

This package provides *convenience classes* for each of six different kinds of polynomials:

|  |  |
| --- | --- |
| Polynomial | Power series |
| Chebyshev | Chebyshev series |
| Legendre | Legendre series |
| Laguerre | Laguerre series |
| Hermite | Hermite series |
| HermiteE | HermiteE series |

The following lists the various constants and methods common to all of the classes representing the various kinds of polynomials. In the following, the term Poly represents any one of the convenience classes (e.g. Polynomial, Chebyshev, Hermite, etc.) while the lowercase p represents an **instance** of a polynomial class.

## Constants

* Poly.domain – Default domain
* Poly.window – Default window
* Poly.basis\_name – String used to represent the basis
* Poly.maxpower – Maximum value n such that p\*\*n is allowed
* Poly.nickname – String used in printing

## Creating polynomial instances.

* Poly.basis(degree) – Basis polynomial of given degree
* Poly.identity() – p where p(x) = x for all x
* Poly.fit(x, y, deg) – p of degree deg with coefficients determined by the least-squares fit to the data x, y
* Poly.fromroots(roots) – p with specified roots
* p.copy() – Create a copy of p

## Converting polynomial instance kind

* p.cast(Poly) – Convert p to instance of kind Poly
* p.convert(Poly) – Convert p to instance of kind Poly or map between domain and window

## Calculus

* p.deriv() – Take the derivative of p
* p.integ() – Integrate p

## Validation

* Poly.has\_samecoef(p1, p2) – Check if coefficients match
* Poly.has\_samedomain(p1, p2) – Check if domains match
* Poly.has\_sametype(p1, p2) – Check if types match
* Poly.has\_samewindow(p1, p2) – Check if windows match

## Misc

* p.linspace() – Return x, p(x) at equally-spaced points in domain
* p.mapparms() – Return the parameters for the linear mapping between domain and window.
* p.roots() – Return the roots of p.
* p.trim() – Remove trailing coefficients.
* p.cutdeg(degree) – Truncate p to given degree
* p.truncate(size) – Truncate p to given size

## Creation of Polynomial instance

a = [1, -12, 10, 7, -10]

# Note: cooefficents in reverse order to np.roots(), hence index reversal

p = np.polynomial.Polynomial(a[::-1])

# [numpy datetime functions](https://numpy.org/doc/stable/reference/routines.datetime.html)

|  |  |
| --- | --- |
| [**datetime\_as\_string**](https://numpy.org/doc/stable/reference/generated/numpy.datetime_as_string.html#numpy.datetime_as_string)(arr[, unit, timezone, …]) | Convert an array of datetimes into an array of strings. |
| [**datetime\_data**](https://numpy.org/doc/stable/reference/generated/numpy.datetime_data.html#numpy.datetime_data)(dtype, /) | Get information about the step size of a date or time type. |

|  |  |
| --- | --- |
| [**busdaycalendar**](https://numpy.org/doc/stable/reference/generated/numpy.busdaycalendar.html#numpy.busdaycalendar)([weekmask, holidays]) | A business day calendar object that efficiently stores information defining valid days for the busday family of functions. |
| [**is\_busday**](https://numpy.org/doc/stable/reference/generated/numpy.is_busday.html#numpy.is_busday)(dates[, weekmask, holidays, …]) | Calculates which of the given dates are valid days, and which are not. |
| [**busday\_offset**](https://numpy.org/doc/stable/reference/generated/numpy.busday_offset.html#numpy.busday_offset)(dates, offsets[, roll, …]) | First adjusts the date to fall on a valid day according to the roll rule, then applies offsets to the given dates counted in valid days. |
| [**busday\_count**](https://numpy.org/doc/stable/reference/generated/numpy.busday_count.html#numpy.busday_count)(begindates, enddates[, …]) | Counts the number of valid days between *begindates* and *enddates*, not including the day of *enddates*. |

# [numpy.ma](https://numpy.org/doc/stable/reference/maskedarray.generic.html)

(Masked array)

# sympy

Symbol()

# sympy.solvers.solveset

nonlinsolve()

# [pandas](https://pandas.pydata.org/docs/reference/index.html)

## Data manipulations

melt(*frame[, id\_vars, value\_vars, var\_name, …]*)

Unpivot a DataFrame from wide to long format, optionally leaving identifiers set. Inverse of *pivot().*

pivot(*data[, index, columns, values]*)

Return reshaped DataFrame, organised by given index/column values. Inverse of *melt()*.

pivot\_table(*data[, values, index, columns, …]*)

Create a spreadsheet-style pivot table as a DataFrame

crosstab(*index, columns[, values, rownames, …]*)

Compute a simple cross tabulation of two (or more) factors

cut(*x, bins[, right, labels, retbins, …]*)

Bin values into discrete intervals

qcut(*x, q[, labels, retbins, precision, …]*)

Quantile-based discretization function

merge(*left, right[, how, on, left\_on, …]*)

Merge DataFrame or named Series objects with a database-style join

merge\_ordered(*left, right[, on, left\_on, …]*)

Perform merge with optional filling/interpolation

merge\_asof(*left, right[, on, left\_on, …]*)

Perform asof merge – similar to left-join, except match on nearest (rather than equal) key

concat(*objs[, axis, join, ignore\_index, …]*)

Concatenate list of pandas objects, *objs*, along a particular axis, with optional set logic along other axes

get\_dummies(*data[, prefix, prefix\_sep, …]*)

Convert categorical variable into dummy/indicator variable

factorize(*values[, sort, na\_sentinel, size\_hint]*)

Encode the object as an enumerated type or categorical variable

unique(*values*)

Hash table-based unique

wide\_to\_long(*df, stubnames, i, j[, sep, suffix]*)

Wide panel to long format

## Missing data

isna(*obj*)

Detect missing values for an array-like object

isnull(*obj*)

Detect missing values for an array-like object

notna(*obj*)

Detect non-missing values for an array-like object

notnull(*obj*)

Detect non-missing values for an array-like object

## Conversions

to\_numeric(*arg[, errors, downcast]*)

Convert argument to numeric type

## Datetime-like

to\_datetime(*arg[, errors, dayfirst, …]*)

Convert argument to datetime

to\_timedelta(*arg[, unit, errors]*)

Convert argument to timedelta

date\_range(*[start, end, periods, freq, tz, …]*)

Return a fixed frequency DatetimeIndex. Must specify exactly three of: (*start, end, periods, freq*)

bdate\_range(*[start, end, periods, freq, tz, …]*)

Return a fixed frequency DatetimeIndex, with business day as the default frequency

period\_range(*[start, end, periods, freq, name]*)

Return a fixed frequency PeriodIndex

timedelta\_range(*[start, end, periods, freq, …]*)

Return a fixed frequency TimdeltaIndex, with day as the default frequency

infer\_freq(*index[, warn]*)

Infer the most likely frequency given the input index

## Intervals

interval\_range(*[start, end, periods, freq, …]*)

Return a fixed frequency IntervalIndex

## Evaluation

eval(*expr[, parser, engine, truediv, …]*)

Evaluate a Python expression as a string using various backends

## Hashing

util.hash\_array(*vals[, encoding, hash\_key, …]*)

Given a 1d array, return an array of deterministic integers

util.hash\_pandas\_object(*obj[, index, …]*)

Return a data hash of the Index/Series/DataFrame

## Testing

test(*[extra\_args]*)

# pandas topics

## [indexing](https://pandas.pydata.org/docs/user_guide/indexing.html#indexing)

at[*row[, column]*]

Label based single value indexing

iat[*row[, column]*]

Integer-location single value indexing

loc[*row[, column]*]

Label based indexing, allows: (*label, list-of-labels, slice-with-labels, boolean-array, alignable-index*). Returns Series if *row* is a scalar, or a DataFrame if *row* is a list

iloc[*row[, column]*]

Integer-location based indexing, allows: (*int, list-of-ints, slice, boolean-array, callable*). Returns Series if *row* is a sclar, or a DataFrame if *row* is a list.

df.columns.get\_loc(*label*)

Get column/index position from column/index label

df.columns[*position*]

Get column/index label from *position*

## [reshaping data](https://pandas.pydata.org/docs/user_guide/reshaping.html)

stack() <-> unstack()

melt() <-> pivot()

pivot\_table()

# [pandas.Series](https://pandas.pydata.org/docs/reference/series.html)

## Constructor

Series(*[data, index, dtype, name, copy, …]*)

One-dimensional ndarray with axis labels

## Attributes

index

The index (axis labels) of the Series

array

The *ExtensionArray* of the data backing this Series or Index

values

Return Series or ndarray or ndarray-like

dtype

Return the dtype object of the underlying data

shape

Return a tuple of the shape of the underlying data

nbytes

Return the number of bytes in the underlying data

ndim

Number of dimensions of the underlying data (1 for a Series)

size

Number of elements in the underlying data

T

Transpose (self for a Series)

memory\_usage(*[index, deep]*)

Return the memory usage of the Series

hasnans

Return if Series has any NaNs

empty

Indicator whether *DataFrame* is empty

dtypes

Return dtype object of the underlying data

name

Return the name of the Series

flags

Properties associated with this pandas object

set\_flags(*\*[, copy, …]*)

Return a new object with updated flags

## Conversion

astype(*dtype[, copy, errors]*)

Cast a pandas object to a specified dtype

convert\_dtypes(*[infer\_objects, …]*)

Convert columns to best possible dtypes using dtypes supporting *pd.NA*

infer\_objects()

Attempt to infer better dtypes for object columns

copy(*[deep]*)

Make a copy of this objects indices and data

bool()

Return the bool of a single element Series or DataFrame

to\_numpy(*[dtype, copy, na\_value]*)

A NumPy *ndarray* representing values in this Series or Index

to\_period(*[freq, copy]*)

Convert Series from *DatetimeIndex* to *PeriodIndex*

to\_timestamp(*freq, how, copy]*)

Cast to DatetimeIndex of Timestamps, at beginning of period

to\_list()

Return a list of the values

\_\_array\_\_(*[dtype]*)

Return the values as a NumPy array

## Indexing, iteration

get(*key[, default]*)

Get item from object for given key

[*row*]

Access a single element by label

at[*row*]

Access a single value for a row/column label pair

iat[*row*]

Access a single value for a row/column pair by integer position

loc[*row*]

Access a group of rows and columns by label(s) ora Boolean array

iloc[*row*]

Purely integer-location based indexing for selection by position

\_\_iter\_\_()

Return an iterator of the values

items()

Lazily iterate over (index, value) tuples

iteritems()

Lazily iterate over (index, value) tuples

keys()

Return alias for index

pop(*item*)

Return item and drops from series

item()

Return the first element of the underlying data as a Python scalar

xs(*key[, axis, level, drop\_level]*)

Return cross-section from Series/DataFrame

## Binary operator functions

add(*other[, level, fill\_value, axis]*)

Return addition of series and *other*, elementwise

sub(*other[, level, fill\_value, axis]*)

Return subtraction of series and *other*, elementwise

mul(*other[, level, fill\_value, axis]*)

Return multiplication of series and *other*, elementwise

div(*other[, level, fill\_value, axis]*)

Return floating division of series and *other*, elementwise (*truediv*)

truediv(*other[, level, fill\_value, axis]*)

Return floating division of series and *other*, elementwise

floordiv(*other[, level, fill\_value, axis]*)

Return integer division of series and *other*, elementwise

mod(*other[, level, fill\_value, axis]*)

Return Modulo of series and *other*, elementwise

pow(*other[, level, fill\_value, axis]*)

Return exponential power of series and *other*, elementwise

radd(*other[, level, fill\_value, axis]*)

Return addition of series and *other*, elementwise

rsub(*other[, level, fill\_value, axis]*)

Return subtraction of series and *other*, elementwise

rmul(*other[, level, fill\_value, axis]*)

Return multiplication of series and *other*, elementwise

rdiv(*other[, level, fill\_value, axis]*)

Return floating division of series and *other*, elementwise

rtruediv(*other[, level, fill\_value, axis]*)

Return floating division of series and *other*, elementwise (*rtruediv*)

rfloordiv(*other[, level, fill\_value, …]*)

Return integer division of series and *other*, elementwise

rmod(*other[, level, fill\_value, axis]*)

Return modulo of series and *other*, elementwise

rpow(*other[, level, fill\_value, axis]*)

Return exponential power of series and *other*, elementwise

combine(*other, func[, fill\_value]*)

Combine the Series with a Series or scalar, according to *func*

combine\_first(*other*)

Combine Series values, choose the calling Series’s values first

round(*[decimals]*)

Round each value in a Series to the given number of decimals

lt(*other[, level, fill\_value, axis]*)

Return less-than of series and *other*, elementwise

gt(*other[, level, fill\_value, axis]*)

Return greater-than of series and *other*, elementwise

le(*other[, level, fill\_value, axis]*)

Return less-than of series and *other*, elementwise

ge(*other[, level, fill\_value, axis]*)

Return greater-than-equal-to of series and *other*, elementwise

ne(*other[, level, fill\_value, axis]*)

Return not-equal of series and *other*, elementwise

eq(*other[, level, fill\_value, axis]*)

Return equal-to of series and *other*, elementwise

product(*[axis, skipna, level, …]*)

Return the product of the values over the requested axis

dot(*other*)

Compute the dot product between the series and columns of *other*

## Func-application, GroupBy, and Window

apply(*func[, convert\_dtype, args]*)

Invoke function *func* on values of series

agg(*[func, axis]*)

Aggregate using one or more operations over the specified axis

aggregate(*[func, axis]*)

Aggregated using one or more operations over the specified axis

transform(*func[, axis]*)

Call *func* on self, producing a Series with transformed values

map(*arg[, na\_action]*)

Map values of Series according to input correspondence

groupby(*[by, axis, level, as\_index, …]*)

Group Series using a mapper or by a Series of columns

rolling(*window[, min\_periods, …]*)

Provide rolling window calculations

expanding(*[min\_periods, center, axis]*)

Provide expanding transformations

ewm(*[com, span, halflife, alpha, …]*)

Provide exponential weighted functions

pipe(*func, \*args, \*\*kwargs*)

Apply *func(self, \*args, \*\*kwargs)*

## Computations / descriptive stats

abs()

Return a Series/DataFrame with absolution numeric value of each element

all(*[axis, bool\_only, skipna, level]*)

Return whether all elements are True, potentially over an axis

any(*[axis, bool\_only, skipna, level]*)

Return whether any element is True, potentially over an axis

autocorr(*[lag]*)

Compute the lag-N autocorrelation

between(*left, right[, inclusive]*)

Return Boolean Series equivalent to *left <=* series <= *right*

clip(*[lower, upper, axis, inplace]*)

Trim values at input threshold

corr(*other[, method, min\_periods]*)

Compute correlation with *other*, excluding missing values

count(*[level]*)

Return number of non-NA/null observations in the Series

cov(*other[, min\_periods, ddof]*)

Compute covariance with series, excluding missing values

cummax(*[axis, skipna]*)

Return cumulative maximum over a DataFrame or Series axis

cummin(*[axis, skipna]*)

Return cumulative minimum over a DataFrame or Series axis

cumprod(*[axis, skipna]*)

Return cumulative product over a DataFrame or Series axis

cumsum(*[axis, skipna]*)

Return cumulative sum over a DataFrame or Series axis

describe(*[percentiles, include, …]*)

Generate descriptive statistics

diff(*[periods]*)

First discrete difference of element

factorize(*[sort, na\_sentinel]*)

Encode the object as an enumerated type or categorical variable

kurt(*[axis, skipna, level, numeric\_only]*)

Return unbiased kurtosis over requested axis

mad(*[axis, skipna, level]*)

Return mean absolute deviation of the values over the requested axis

max(*axis, skipna, level, numeric\_only]*)

Return the maximum of the values over the requested axis

min(*[axis, skipna, level, numeric\_only]*)

Return the minimum of the values over the requested axis

mean(*[axis, skipna, level, numeric\_only]*)

Return the mean of the values over the requested axis

median(*[axis, skipna, level, …]*)

Return the median of the values over the requested axis

mode(*[dropna]*)

Return the mode(s) of the series

nlargest(*[n, keep]*)

Return the largest *n* elements

nsmallest(*[n, keep]*)

Return the smallest *n* elements

pct\_change(*[periods, fill\_method, …]*)

Percentage change between the current and a prior element

prod(*[axis, skipna, level, …]*)

Return the product of the values over the requested axis

quantile(*[q, interpolation]*)

Return value at the given quantile

rank(*[axis, method, numeric\_only, …]*)

Compute numerical data ranks [*1, n*] along axis

sem(*axis, skipna, level, ddof, …]*)

Return unbiased standard error of the mean over requested axis

skew(*[axis, skipna, level, numeric\_only]*)

Return unbiased skew over requested axis

std(*[axis, skipna, level, ddof, …]*)

Return sample standard deviation over requested axis

sum(*[axis, skipna, level, …]*)

Return the sum of the values over the requested axis

var(*[axis, skipna, level, …]*)

Return unbiased variance over requested axis

kurtosis(*[axis, skipna, level, …]*)

Return unbiased kurtosis over requested axis

unique()

Return unique values of series

nunique(*[dropna]*)

Return number of unique elements in series

is\_unique

Return Boolean, whether values in series are unique

is\_monotonic

Return Boolean if values are monotonic\_increasing

is\_monotonic\_increasing

Alias for *is\_monotonic*

is\_monotonic\_decreasing

Return Boolean if values are monotonic\_decreasing

value\_counts(*[normalize, sort, …]*)

Return series containing counts of unique values

## Reindexing / selection / label manipulation

align(*other[, join, axis, level, …]*)

Algin two objects on their axes with the specified join method

drop(*labels, axis, index, columns, …]*)

Return series with rows given by *labels* removed

droplevel(*level[, axis]*)

Return DataFrame with requested index / column level(s) removed

drop\_duplicates(*[keep, inplace]*)

Return series with duplicate values removed

duplicated(*[keep]*)

Indicate duplicate series values

equals(*other*)

Test whether two objects contain the same elements

first(*offset*)

Select initial periods of time series data based on a date *offset*

last(*offset*)

Select final periods of time series data based on a date *offset*

head(*[n]*)

Return the first *n* rows

idxmax(*[axis, skipna]*)

Return the row label of the maximum value

idxmin(*[axis, skipna]*)

Return the row label of the minimum value

isin(*values*)

Boolean array, whether elements in series are contained in *values*, of same length as series

reindex(*[index]*)

Conform series to new index with optional filling logic

reindex\_like(*other[, method, copy, …]*)

Return an object with matching indices as other object

rename(*[index, axis, copy, inplace, …]*)

Alter series index labels or name

rename\_axis(*[mapper, index, columns, …]*)

Set the name of the axis for the index or columns

reset\_index(*[level, drop, name, inplace]*)

Generate a new DataFrame or Series with the index reset. If *drop=False*, add old index as new column.

sample(*[n, frac, replace, weights, …]*)

Return a random sample of items from an axis of object

set\_axis(*labels[, axis, inplace]*)

Assign desired index to given axis

take(*indices[, axis, is\_copy]*)

Return elements in the given positional indices along an axis

tail(*[n]*)

Return the last *n* rows

truncate(*[before, after, axis, copy]*)

Truncate a Series or DataFrame before and after some index value

where(*cond[, other, inplace, axis, …]*)

Replace values where the condition *cond* is False

mask(*cond[, other, inplace, axis, …]*)

Replace values where the condition *cond* is True

add\_prefix(*prefix*)

Prefix labels with string *prefix*

add\_suffix(*suffix*)

Suffix labels with string *suffix*

filter(*[items, like, regex, axis]*)

Subset the DataFrame rows or columns according to the specified index labels

## Missing data handling

backfill(*[axis, inplace, limit, downcast]*)

Synonym for *fillna()* with *method=’bfill’*

bfill(*[axis, inplace, limit, downcast]*)

Synonym for *fillna()* with *method=’bfill’*

dropna(*[axis, inplace, how]*)

Return a new Series with missing values removed

ffill(*[axis, inplace, limit, downcast]*)

Synonym for *fillna()* with *method=’ffill’*

fillna(*[value, method, axis, …]*)

Fill NA/NaN values using specified method

interpolate(*[method, axis, limit, …]*)

Fill NaN values using the specified method

isna()

Detect missing values

isnull()

Detect missing values

notna()

Detect existing (non-missing) values

notnull()

Detect existing (non-missing) values

pad(*[axis, inplace, limit, downcast]*)

Synonym for *fillna()* with *method=’ffill’*

replace(*[to\_replace, value, inplace, …]*)

Replace values given in *to\_replace* with *value*

## Reshaping, sorting

argsort(*[axis, kind, order]*)

Return the integer indices that would sort the series values

argmin(*[axis, skipna]*)

Return integer position of the smallest value in the series

argmax(*[axis, skipna]*)

Return integer position of the largest value in the series

reorder\_levels(*order*)

Rearrange index levels using input *order*

sort\_values(*[axis, ascending, …]*)

Sort by the values

sort\_index(*[axis, level, ascending, …]*)

Sort series by index labels

swaplevel(*[i, j, copy]*)

Swap levels *i* and *j* in a MultiIndex

unstack(*[level, fill\_value]*)

Unstack (or pivot), series with MultiIndex to produce DataFrame

explode(*[ignore\_index]*)

Transform each element of a list-like to a row

searchsorted(*value[, side, sorter]*)

Find indices where elements should be inserted to maintain order

ravel(*[order]*)

Return the flattened underlying data as an *ndarray*

repeat(*repeats[, axis]*)

Repeat elements of series

squeeze(*[axis]*)

Squeeze 1-dimensional axis objects into scalars

view(*[dtype]*)

Create a new view of the Series

## Combining / comparing / joining / merging

append(*to\_append[, ignore\_index, …]*)

Concatenate two or more series

compare(*other[, align\_axis, …]*)

Compare to another series and show the differences

update(*other*)

Modify series in-place using values from passed series *other*

## Time Series-related

asfreq(*freq[, method, how, …]*)

Convert TimeSeries to specified frequency

asof(*where[, subset]*)

Return the last row(s) without any NaNs before *where*

shift(*[periods, freq, axis, fill\_value]*)

Shift index by desired number of periods with optional time *freq*

first\_valid\_index()

Return index for first non-NA/null value

last\_valid\_index()

Return index for last non-NA/null value

resample(*rule[, axis, closed, label, …]*)

Resample time-series data

tz\_convert(*tz[, axis, level, copy]*)

Convert tz-aware axis to target timezone

tz\_localize(*tz[, axis, level, copy, …]*)

Localize tz-naïve index of a Series or DataFrame to target timezone

at\_time(*time[, asof, axis]*)

Select values a particular time of day

between\_time(*start\_time, end\_time[, …]*)

Select values between particular times of the day

tshift(*[periods, freq, axis]*)

Shift the time index, using the index frequency if available (Deprecated)

slice\_shift(*[periods, axis]*)

Equivalent to *shift()* without copying data (Deprecated)

## Accessors

dt

str

cat

sparse

## Datetime properties

dt.date

dt.time

dt.timetz

dt.year

dt.month

dt.day

dt.hour

dt.minute

dt.second

dt.microsecond

dt.nanosecond

dt.isocalendar().week

dt.week (Deprecated)

dt.weekofyear (Deprecated)

dt.dayofweek

dt.day\_of\_week

dt.weekday

dt.dayofyear

dt.day\_of\_year

dt.quarter

dt.is\_month\_start

dt.is\_quarter\_start

dt.is\_quarter\_end

dt.is\_year\_start

dt.is\_year\_end

dt.is\_leap\_year

dt.daysinmonth

dt.days\_in\_month

dt.tz

dt.freq

## Datetime methods

dt.to\_period(*\*args, \*\*kwargs*)

Cast PeriodArray/Index at a particular frequency

dt.to\_pydatetime()

Return the data as an array of native Python datetimes

dt.tz\_localize(*\*args, \*\*kwargs*)

Localize tz-naïve Datetime Array/Index to tz-aware

dt.tz\_convert(*\*args, \*\*kwargs*)

Convert tz-aware Datetime Array/Index from one timezone to another

dt.normalize(*\*args, \*\*kwargs*)

Convert times to midnight

dt.strftime(*\*args, \*\*kwargs*)

Convert to Index using specified date format

dt.round(*\*args, \*\*kwargs*)

Perform round operation on data to specified *freq*

dt.floor(*\*args, \*\*kwargs*)

Perform floor operation on the data to specified *freq*

dt.ceil(*\*args, \*\*kwargs*)

Perform ceil operation on the data to specified *frreq*

dt.month\_name(*\*args, \*\*kwargs*)

Return the month names of the DateTimeIndex with specified locale

dt.day\_name(*\*args, \*\*kwargs*)

Return the day names of the DateTimeIndex with specified locale

## Period properties

dt.qyear

dt.start\_time

dt.end\_time

## Timedelta properties

dt.days

dt.seconds

dt.microseconds

dt.nanoseconds

dt.components

## Timedelta methods

dt.to\_pytimedelta()

Return an array of native *datetime.timedelta* objects

dt.total\_seconds()

Return total duration of each element, expressed in seconds

## String handling

str.capitalize()

Convert strings to be capitalized

str.casefold()

Convert strings to be casefolded

str.cat(*[others, sep, na\_rep, join]*)

Concatenate strings with given seperator

str.center(*width[, fillchar]*)

Pad left and right sides of strings

str.contains(*pat[, case, flags, na, …]*)

Test if pattern or regex is contained within string

str.count(*pat[, flags]*)

Count occurrences of pattern in each string

str.decode(*encoding[, errors]*)

Decode character string using indicated encoding

str.encode(*encoding[, errors]*)

Encode character string using indicated encoding

str.endswith(*pat[, na]*)

Test if the end of string element matches pattern

str.extract(*pat[, flags, expand]*)

Extract capture groups in the regex *pat* as columns in a DataFrame

str.extractall(*pat[, flags]*)

Extract capture groups in the regex *pat* as columns in DataFrame

str.find(*sub[, start, end]*)

Return lowest indexes in each strings

str.findall(*pat[, flags]*)

Find all occurences of pattern or regex

str.get(*i*)

Extract element from each component at specified postition

str.index(*sub[, start, end]*)

Return lowest indexes in each string

str.join(*sep*)

Join lists contained as elements with passed delimiter

str.len()

Compute the length of each element

str.ljust(*width[, fillchar]*)

Pad right side of strings

str.lower()

Convert strings to lowercase

str.lstrip(*[to\_strip]*)

Remove leading characters

str.match(*pat[, case, flags, na]*)

Determine if each strings starts with a match of a regex

str.normalize(*form*)

Return the unicode normal form for strings

str.pad(*width[, side, fillchar]*)

Pad strings up to *width*

str.partition(*[sep, expand]*)

Split string at the first occurrence of *sep*

str.repeat(*repeats*)

Duplicate each string

str.replace(*pat, repl[, n, case, …]*)

Replace each occurrence of patten/regex

str.rfind(*sub[, start, end]*)

Return highest indexes in each string

str.rindex(*sub[, start, end]*)

Return highest indexes in each string

str.rjust(*width[, fillchar]*)

Pad left side of strings

str.rpartition(*[sep, expand]*)

Split the string at the last occurrence of *sep*

str.rstip(*[to\_strip]*)

Remove trailing characters

str.slice(*[start, stop, step]*)

Slice substrings from each element

str.slice\_replace(*[start, stop, repl]*)

Replace a positional slice of a string with another value

str.split(*[pat, n, expand]*)

Split strings around given separator/delim

str.rsplit(*[pat, n, expand]*)

Split strings around given separator/delim

str.startswith(*pat[, na]*)

Test if the start of each string element matches a pattern

str.strip(*[to\_strip]*)

Split strings around given separator/delim

str.swapcase()

Convert strings to be swapcased

str.title()

Convert strings to titlecase

str.translate(*table*)

Map all characters in string through given mapping *table*

str.upper()

Convert string to uppercase

str.wrap(*width, \*\*kwargs*)

Wrap strings at specified line width

str.zfill(*width*)

Pad strings by prepending ‘0’

str.isalnum()

Check whether all characters are alphanumeric

str.isalpha()

Check whether all characters are alphabetical

str.isdigit()

Check whether all characters are digits

str.isspace()

Check whether all characters are <whitespace>

str.islower()

Check whether all characters are lowercase

str.isupper()

Check whether all characters are uppercase

str.istitle()

Check whether all characters are titlecase

str.isnumeric()

Check whether all characters are <numeric>

str.isdecimal()

Check whether all characters are decimal

str.get\_dummies(*[sep]*)

Return DataFrame of dummy/indicator variables

## Categorical accessor

cat.categorise

Categories of categorical

cat.ordered

Whether categories have an ordered relationship

cat.codes

Return series of codes as well as index

cat.rename\_categories(*\*args, \*\*kwargs*)

Rename categories

cat.reorder\_categories(*\*args, \*\*kwargs*)

Reorder categories as specified in new\_categories

cat.add\_categories(*\*args, \*\*kwargs*)

Add new categories

cat.remove\_categories(*\*args, \*\*kwargs*)

Remove the specified categories

cat.remove\_unused\_categories(*\*args, …*)

Remove categories which are not used

cat.set\_categories(*\*args, \*\*kwargs*)

Set the categories to the specified *new\_categories*

cat.as\_ordered(*\*args, \*\*kwargs*)

Set the categorical to be ordered

cat.as\_unordered(*\*args, \*\*kwargs*)

Set the categorical to be unordered

## Sparse accessor

sparse.npoints

The number of non-*fill\_value* points

sparse.density

The percentage of non-*fill\_value* points, as decimal

sparse.fill\_value

Elements in *data* that are *fill\_value* (that are) are not stored

sparse.sp\_values

An ndarray containing the non-*fill\_value* values

sparse.from\_coo

Create a Series with spares values from a *scipy.sparse.coo\_matrix*

sparse.to\_coo

Create a *scipy.sparse.coo\_matrix* from a Series with MultiIndex

## Flags

Flags(*obj, \*, allows\_duplicate\_labels*)

Flags refer to attributes of the pandas object

## Metadata

attrs

Dictionary for storing global attributes of dataset

## Plotting

plot(*[kind, ax, figsize, …]*)

Series plotting accessor and method

plot.area(*[x, y]*)

Draw as stacked area plot

plot.bar(*[x, y]*)

Vertical bar plot

plot.barh(*[x, y]*)

Horizontal bar plot

plot.box(*[by]*)

Box plot of the DataFrame columns

plot.density(*[bw\_method, ind]*)

Kernel density estimate plot

plot.hist(*[by, bins]*)

Draw histogram of the DataFrame columns

plot.kde(*[bw\_method, ind]*)

Kernel density estimate plot

plot.line(*[x, y]*)

Plot Series or DataFrame as lines

plot.pie(*\*\*kwargs*)

Generate a pie plot

hist(*[by, ax, grid, xlabelsize, …]*)

Draw histogram of the input Series

## Serialization/IO/Conversion

to\_pickle(*path[, compression, …]*)

Pickle (serialize) object to file

to\_csv(*[path\_or\_buf, sep, na\_rep, …]*)

Write object to a comma-separated-values file

to\_dict(*[into]*)

Convert Series to {*k: v*} dict or dict-like object

to\_excel(*excel\_writer[, sheet\_name, …]*)

Write object to an Excel sheet

to\_frame(*[name]*)

Convert Series to DataFrame

to\_xarray()

Return an xarray object

to\_hdf(*path\_or\_buf, key[, mode, …]*)

Write contained data to an HDF5 file

to\_sql(*name, con[, schema, …]*)

Write records stored in DataFrame to SQL database

to\_json(*[path\_or\_buf, orient, …]*)

Convert object to JSON string

to\_string(*[buf, na\_rep, …]*)

Render a string representation of Series

to\_clipboard(*[excel, sep]*)

Copy object to the system clipboard

to\_latex(*buf, columns, col\_space, …]*)

Render object to LaTeX tabular, longtable, or nested table

to\_markdown(*[buf, mode, index, …]*)

Print Series in Markdown friendly format

# [pandas.DataFrame](https://pandas.pydata.org/docs/reference/frame.html)

## Constructor

DataFrame(*[data, index, columns, dtype, copy]*)

2d, size-mutable, potentially heterogeneous tabular data

## Attributes

index

Index (row) labels of the DataFrame

columns

Column labels of the DataFrame

dtypes

Return dtypes in the DataFrame

info(*[verbose, buf, max\_cols, …]*)

Print a concise summary of the DataFrame

select\_dtypes(*[include, exclude]*)

Return a subset of the DataFrame’s columns based on the column dtypes

values

Return a NumPy representation of the DataFrame

axes

Return a list representing the axes of the DataFrame

ndim

Return an int representing the number of axes/array-dimensions

size

Return a int representing the number of elements in object

shape

Return a tuple representing the dimensionality of the DataFrame

memory\_usage(*[index, deep]*)

Return the memory usage of each column in bytes

empty

Indicator whether DataFrame is empty

set\_flags(*\*[, copy, …]*)

Return a new object with updated flags

## Conversion

astype(*dtype[, copy, errors]*)

Cast a pandas object to a specific dtype

convert\_dtypes(*[infer\_objects, …]*)

Convert columns to best possible dtypes using dtypes supporting *pd.NA*

infer\_objects()

Attempt to infer better dtypes for object columns

copy(*[deep]*)

Make a copy of this objects indices and data

bool()

Return the bool of a single element Series or DataFrame

## Indexing, iteration

head(*[n]*)

Return the first *n* rows

[*column*]

Access a single column by label

at[*row[, column]*]

Access a single value for a row/column label pair

iat[*row[, column]*]

Access a single value for a row/column pair by integer position

loc[*row[, column]*]

Access a group of rows and columns by label(s), or a Boolean array

iloc[*row[, column]*]

Purely integer-location based indexing for selection by position

insert(*loc, columns, values[, …]*)

Insert column into DataFrame at specified location

\_\_iter\_\_()

Iterate over info axis

items()

Iterate over (*column name, Series*) pairs

iteritems()

Iterate over (*column name, Series*) pairs

keys()

Get the ‘info’ axis

iterrows()

Iterate over DataFrame rows as (*index, Series*) pairs

itertuples(*[index, name]*)

Iterate over DataFrame rows as *namedtuples*

lookup(*row\_labels, col\_labels*)

Label-based ‘fancy’ indexing function (Deprecated)

pop(*item*)

Return item and drop from frame

tail(*[n]*)

Return the last *n* rows

xs(*key[, axis, level, drop\_level]*)

Return cross-section from the DataFrame

get(*key[, default]*)

Get item from object for given key

isin(*values*)

Boolean array, whether each element in the DataFrame is contained in values, same length as DataFrame

where(*cond[, other, inplace, …]*)

Replace values where condition *cond* is False

mask(*cond[, other, inplace, axis, …]*)

Replace values where the condition *cond* is True

query(*expr[, inplace]*)

Query the columns of a DataFrame with a Boolean expression

## Binary operator functions

add(*other[, axis, level, fill\_value]*)

Get addition of dataframe and *other*, elementwise (+)

sub(*other[, axis, level, fill\_value]*)

Get subtraction of dataframe and *other*, elementwise (-)

mul(*other[, axis, level, fill\_value]*))

Get multiplication of dataframe and *other*, elementwise (\*)

div(*other[, axis, level, fill\_value]*)

Get floating division of dataframe and *other*, elementwise (/)

truediv(*other[, axis, level, …]*)

Get floating division of dataframe and *other*, elementwise (/)

floordiv(*other[, axis, level, …]*)

Get integer division of dataframe and *other*, elementwise (//)

mod(*other[, axis, level, fill\_value]*)

Get modulo of dataframe and *other*, elementwise (%)

pow(*other[, axis, level, fill\_value]*)

Get exponential power of dataframe and *other*, elementwise (\*\*)

dot(*other*)

Compute the matrix multiplication between the dataframe and *other* (@)

radd(*other[, axis, level, fill\_value]*)

Get addition of dataframe and *other*, elementwise (+)

rsub(*other[, axis, level, fill\_value]*)

Get subtraction of dataframe and *other*, elementwise (-)

rmul(*other[, axis, level, fill\_value]*)

Get multiplication of dataframe and *other*, elementwise (\*)

rdiv(*other[, axis, level, fill\_value]*)

Get floating division of dataframe and *other*, elementwise (/)

rtruediv(*other[, axis, level, …]*)

Get floating division of dataframe and *other*, elementwise (/)

rfloordiv(*other[, axis, level, …]*)

Get integer division of dataframe and *other*, elementwise (//)

rmod(*other[, axis, level, fill\_value]*)

Get modulo of dataframe and *other*, elementwise (%)

rpow(*other[, axis, level, fill\_value]*)

Get exponential power of dataframe and *other*, elementwise (\*\*)

lt(*other[, axis, level]*)

Less-than of dataframe and *other*, elementwise (<)

gt(*other[, axis, level]*)

Greater-than of dataframe and *other*, elementwise (>)

le(*other[, axis, level]*)

Less-than-equal of dataframe and *other*, elementwise (<=)

ge(*other[, axis, level]*)

Greater-than-equal of dataframe and *other*, elementwise (>=)

ne(*other[, axis, level]*)

Not-equal of dataframe and *other*, elementwise (!=)

eq(*other[, axis, level]*)

Equal of dataframe and *other*, elementwise (==)

combine(*other, func[, fill\_value, …]*)

Perform column-wise combine with another DataFrame

combine\_first(*other*)

Update null elements with *value* in the same location as *other*

## Func-application, GroupBy, and Window

apply(*func[, axis, raw, …]*)

Apply a function *func* along an axis of the dataframe

applymap(*func[, na\_action]*)

Apply a function *func* to a dataframe elementwise

pipe(*func, \*args, \*\*kwargs*)

Apply *func(self, \*args, \*\*kwargs)*

agg(*[func, axis]*)

Aggregaate using one or more operations over the specified axis

aggregate(*[func, axis]*)

Aggregate using one or more operations over the specified axis

transform(*func[, axis]*)

Call *func* on self, producing a DataFrame with transformed values

groupby(*[by, axis, level, …]*)

Group dataframe using a mapper, or by a series of columns

rolling(*window[, min\_periods, …]*)

Provide rolling window calculations

expanding(*[min\_periods, center, axis]*)

Provide expanding transformations

ewm(*[com, span, halflife, alpha, …]*)

Provide exponential weighted functions

## Computations, descriptive stats

abs()

Return a DataFrame with absolution numeric value of each element

all(*[axis, bool\_only, skipna, level]*)

Return whether all elements are True, potentially over an axis

any(*[axis, bool\_only, skipna, level]*)

Return whether any element is True, potentially over an axis

clip(*[lower, upper, axis, inplace]*)

Trim values at input threshold(s)

corr(*[method, min\_periods]*)

Compute pairwise correlation of columns, excluding NA/null values

corrwith(*other[, axis, drop, method]*)

Compute pairwise correlation

count(*[axis, level, numeric\_only]*)

Count non-NA cells for each column or row

cov(*[min\_periods, ddof]*)

Compute pairwise covariance of columns, excluding NA/null values

cummax(*[axis, skipna]*)

Return cumulative maximum over a DataFrame axis

cummin(*[axis, skipna]*))

Return cumulative minimum over a DataFrame axis

cumprod(*[axis, skipna]*))

Return cumulative product over a DataFrame axis

cumsum(*[axis, skipna]*))

Return cumulative sum over a DataFrame axis

describe(*[percentiles, include, …]*)

Generate descriptive statistics

diff(*[periods, axis]*)

First discrete difference of element

eval(*expr[, inplace]*)

Evaluate a string describing operations on DataFrame columns

kurt(*[axis, skipna, level, …]*)

Return unbiased kurtosis over requested axis

kurtosis(*[axis, skipna, level, …]*)

Return unbiased kurtosis over requested axis

mad(*[axis, skipna, level]*)

Return the mean absolute deviation of the values over the requested axis

max(*[axis, skipna, level, …]*)

Return the max of the values over the requested axis

min(*[axis, skipna, level, …]*)

Return the min of the values over the requested axis

mean(*[axis, skipna, level, …]*)

Return the mean of the values over the requested axis

median(*[axis, skipna, level, …]*)

Return the median of the values over the requested axis

mode(*[axis, numeric\_only, dropna]*)

Get the mode(s) of each element along the selected axis

pct\_change(*[periods, fill\_method, …]*)

Percentage change between the current and a prior element

prod(*[axis, skipna, level, …]*)

Return the product of the values over the requested axis

product(*[axis, skipna, level, …]*)

Return the product of the values over the requested axis

quantile(*[q, axis, numeric\_only, …]*)

Return values at the given quantile over requested axis

rank(*[axis, method, numeric\_only, …]*)

Compute numerical data ranks [*1, n*] along axis

round(*[decimals]*)

Round a dataframe to a variable number of decimal places

sem(*[axis, skipna, level, ddof, …]*)

Return unbiased standard error of the mean over requested axis

skew(*[axis, skipna, level, …]*)

Return unbiased skew over requested axis

sum(*[axis, skipna, level, …]*)

Return the sum of the values over the requested axis

std(*[axis, skipna, level, ddof, …]*)

Return sample standard deviation over requested axis

var(*[axis, skipna, level, ddof, …]*)

Return unbiased variance over requested axis

nunique(*[axis, dropna]*)

Count distinct observations over requested axis

value\_counts([*subset, normalize, …]*)

Return a Series containing counts of unique rows in the dataframe

## Reindexing / selection / label-manipulation

add\_prefix(*prefix*)

Prefix labels with string *prefix*

add\_suffix(*suffix*)

Suffix labels with string *suffix*

align(*other[, join, axis, level, …]*)

Align two objects on their axes with the specified join method

at\_time(*time[, asof, axis]*)

Select values at particular time of day

between\_time(*start\_time, end\_time*)

Select values between particular times of the day

drop(*[labels, axis, index, …]*)

Drop rows or columns specified by *labels*

drop\_duplicates(*[subset, keep, …]*)

Return DataFrame with duplicate rows removed

duplicated(*[subset, keep]*)

Return Boolean Series denoting duplicate rows

equals(*other*)

Test whether two objects contain the same elements

filter(*[items, like, regex, axis]*)

Subset the dataframe rows or columns according to the specified index labels

first(*offset*)

Select initial periods of time series data based on a date offset

last(*offset*)

Select final periods of time series data based on a date offset

head(*[n]*)

Return the first *n* rows

tail(*[n]*)

Return the last *n* rows

idxmax(*[axis, skipna]*)

Return index of first occurrence of maximum over requested axis

idxmin(*[axis, skipna]*)

Return index of first occurrence of minimum over requested axis

reindex(*[labels, index, columns, …]*)

Conform DataFrame to new index with optional filling logic

reindex\_like(*other[, method, …]*)

Return an object with matching indices as another object

rename(*[mapper, index, columns, …]*)

Alter axes labels

rename\_axis(*[mapper, index, …]*)

Set the name of the axis for the index or columns

reset\_index(*[level, drop, …]*)

Reset the index, or a level of it

sample(*[n, frac, replace, …]*)

Return a random sample of items from an axis of object

set\_axis(*labels[, axis, inplace]*)

Assign desired index to given axis

set\_index(*keys[, drop, append, …]*)

Set the dataframe index using existing columns

take(*indices[, axis, is\_copy]*)

Return the elements in the given positional indices along an axis

truncate(*[before, after, axis, copy]*)

Truncate dataframe before and after some index value

## Missing data handling

backfill(*[axis, inplace, limit, …]*)

Synonym for *fillna()* with *method=’bfill’*

bfill(*[axis, inplace, limit, downcast]*)

Synonym for *fillna()* with *method=’bfill’*

dropna(*[axis, how, thresh, …]*)

Remove missing values

ffill(*[axis, inplace, limit, downcast]*)

Synonym for *fillna()* with *method=’ffill’*

fillna(*[value, method, axis, …]*)

Fill NA/NaN values using the specified method

interpolate(*[method, axis, limit, …]*)

Fill NaN values using an interpolation method

isna()

Detect missing values

isnull()

Detect missing values

notna()

Detect existing (non-missing) values

notnull()

Detect existing (non-missing) values

pad(*[axis, inplace, limit, downcast]*)

Synonym for *fillna()* with *method=’ffill’*

replace(*[to\_replace, value, …]*)

Replace values given in *to\_replace* with *value*

## Reshaping / sorting / transposing

droplevel(*level[, axis]*)

Return DataFrame with requested index/column level(s) removed

pivot(*[index, columns, values]*)

Return reshaped DataFrame organized by given index/column values

pivot\_table(*[values, index, …]*)

Create a spreadsheet-style pivot table as a DataFrame

reorder\_levels(*order[, axis]*)

Rearrange index levels using input order

sort\_values(*by[, axis, ascending, …]*)

Sort by the values along either axis

sort\_index(*[axis, level, …]*)

Sort object by labels (along an axis)

nlargest(*n, columns[, keep]*)

Return the first *n* rows ordered by *columns* in descending order

nsmallest(*n, columns[, keep]*)

Return the first *n* rows ordered by *columns* in ascending order

swaplevel(*[i, j, axis]*)

Swap levels *i* and *j* in a MultiIndex on a particular axis

stack(*[level, dropna]*)

Stack the prescribed level(s) from columns to index

unstack(*[level, fill\_value]*)

Pivot a level of the (necessarily hierarchical) index labels

swapaxes(*axis1, axis2[, copy]*)

Interchange axes and swap values axes appropriately

melt(*[id\_vars, value\_vars, …]*)

Unpivot dataframe from wide to long format, optionally leaving identifiers set

explode(*column[, ignore\_index]*)

Transform each element of a list-like to a row, replicating index values

squeeze(*[axis]*)

Squeeze 1-dimensional axis objects into scalars

to\_xarray()

Return an xarray object from the pandas object

T

Transpose index and columns

transpose(*\*args[, copy]*)

Transpose index and columns

## Combining / comparing / joining / merging

append(*other[, ignore\_index, …]*)

Append rows of *other* to the end of caller, returning a new object

assign(*\*\*kwargs*)

Assign new columns to a dataframe

compare(*other[, align\_axis, …]]*)

Compare to another dataframe and show the differences

join(*other[, on, how, lsuffix, …]*)

Join columns of another dataframe

merge(*right[, how, on, left\_on, …]*)

Merge dataframe or named series objects with a database-style join

update(*other[, join, overwrite, …]*)

Modify in-place using non-NA values from another dataframe

## Time Series related

asfreq(*freq[, method, how, …]*)

Convert TimeSeries to specified frequency

asof(*where[, subset]*)

Return the last row(s) without any NaNs before *where*

shift(*[periods, freq, axis, …]*)

Shift index by desired number of periods with an optional time *freq*

slice\_shift(*[periods, axis]*)

Equivalent to *shift()* without copying data (Deprecated)

tshift(*[periods, freq, axis]*)

Shift the time index, using the index’s frequency if available (Deprecated)

first\_valid\_index()

Return index for first non-NA/null value

last\_valid\_index()

Return index for last non-NA/null value

resample(*rule[, axis, closed, …]*)

Resample time-series data

to\_period(*[freq, axis, copy]*)

Convert dataframe from DatatimeIndex to PeriodIndex

to\_timestamp(*[freq, how, axis, copy]*)

Cast to DatetimeIndex of timestamps, at *beginning* of period

tz\_convert(*tz[, axis, level, copy]*)

Convert tz-aware axis to target timezone

tz\_localize(*tz[, axis, level, …]*)

Localize tz-naïve index of dataframe to target timezone

## Flags

Flags(*obj, \*, allows\_duplicate\_labels*)

Flags that apply to pandas objects

## Metadata

attrs

Dictionary of global attributes of dataset

## Plotting

plot(*[x, y, kind, ax, …]*)

DataFrame plotting accessor and method

plot.area(*[x, y]*)

Draw a stacked area plot

plot.bar(*[x, y]*)

Vertical bar plot

plot.barh(*[x, y]*)

Horizontal bar plot

plot.box(*[by]*)

Box plot of dataframe columns

plot.density(*[bw\_method, ind]*)

Kernel Density Estimate plot

plot.hexbin(*x, y[, C, …]*)

Hexagonal binning plot

plot.hist(*[by, bins]*)

Histogram of dataframe’s columns

plot.kde(*[bw\_method, ind]*)

Kernel Density Estimate plot

plot.line(*[x, y]*)

Plot dataframe as lines

plot.pie(*\*\*kwargs*)

Pie plot

plot.scatter(*x, y[, s, c]*)

Scatter plot with varying marker point size and color

boxplot(*[column, by, ax, …]*)

Box plot from dataframe columns

hist(*[column, by, grid, …]*)

Histogram of dataframe

## Sparse accessor

sparse.density

Ratio of non-sparse points to total (dense) data points

sparse.from\_spmatrix(*data[, …]*)

Create new DataFrame from SciPy sparse matrix

sparse.to\_coo()

Return the contents of the frame as a sparse SciPy COO matrix

sparse.to\_dense()

Convert dataframe with sparse values to dense

## Serialization / IO / Conversion

from\_dict(*data[, orient, dtype, …]*)

Construct DataFrame from dict of array-like or dicts

from\_records(*data[, index, …]*)

Convert structured or record ndarray to DataFrame

to\_parquet(*[path, engine, …]*)

Write dataframe to binary parquet format

to\_pickle(*path[, compression, …]*)

Pickle (serialize) object to file

to\_csv(*path\_or\_buf, sep, na\_rep, …]*)

Write object to comma-separated-values file

to\_hdf(*path\_or\_buf, key[, mode, …]*)

Write contained data to HDF5 file

to\_sql(*name, con[, schema, …]*)

Write records stored in dataframe to SQL database

to\_dict(*[orient, into]*)

Convert DataFrame to dictionary

to\_excel(*excel\_writer[, …]*)

Write object to an Excel sheet

to\_json([*path\_or\_buf, orient, …]*)

Convert the object to a JSON string

to\_html(*[buf, columns, col\_space, …]*)

Render a dataframe as an HTML table

to\_feather(*path, \*\*kwargs*)

Write a dataframe to the binary Feather format

to\_latex(*[buf, columns, …]*)

Render object to a LaTeX tabular, longtable, or nested table

to\_stata(*path[, convert\_dates, …]*)

Export dataframe to Stata dta format

to\_gbq(*destination\_table[, …]*)

Write dataframe to Google BigQuery table

to\_records(*[index, column\_dtypes, …]*)

Convert dataframe to NumPy record array

to\_string(*[buf, columns, …]*)

Render dataframe to console-friendly tabular output

to\_clipboard(*[excel, sep]*)

Copy object to the system clipboard

to\_markdown(*[buf, mode, index, …]*)

Print dataframe in markdown-friendly format

style

Return a Styler object

# [pandas.Index](https://pandas.pydata.org/docs/reference/api/pandas.Index.html)

An index instance can contain only hashable objects. Many Index methods are also provided by the pandas objects which contain an Index (Series/DataFrame) -- those versions should generally be preferred. Since Indexes are immutable, all methods return a new Index as opposed to modifying the current Index in-place. Other index types inherit attributes/methods of Index.

## constructor

Index(*[data, dtype, copy, name, tupleize\_cols]*)

Immutable sequence used for indexing and alignment. Basic object storing axis labels for all pandas objects

## attributes

T

Transpose, which is by definition self

array

*ExtensionArray* of the data backing Index

asi8

Integer representation of values

dtype

dtype object of underlying data

has\_duplicates

Check if index has duplicate values

hasnans

Return whether index has any NaNs

inferred\_type

Return a string of the type inferred from the values

is\_all\_dates

Whether or not the index values only consists of dates

is\_monotonic

Alias for *is\_monotonic\_increasing*

is\_monotonic\_decreasing

Return if the index is monotonic decreasing (not increasing anywhere)

is\_monotonic\_increasing

Return if the index is monotonic increasing (not decreasing anywhere)

is\_unique

Return if the index has all unique values

name

Return Index or MultiIndex name

nbytes

Return number of bytes in underlying data

ndim

Number of dimensions of the underlying data (by definition 1)

nlevels

Number of levels

shape

Return a tuple of the shape of the underlying data

size

Return the number of elements in the underlying data

values

Return an array representing the data in the Index (*array* / *to\_numpy()* are recommended instead)

memory\_usage(*[deep]*)

Memory usage of the values

## Modifying and computations

all()

Whether all elements are Truthy

any(*\*args, \*\*kwargs*)

Whether any element is Truthy

argmin(*[axis, skipna]*)

Return int position of the smallest value

argmax(*[axis, skipna]*)

Return int position of the largest value

copy(*[name, deep, dtype, names]*)

Make a copy of this object

delete(*loc*)

Make new Index with passed location(s) deleted

drop(*labels[, errors]*)

Make new Index with passed list of labels deleted

drop\_duplicates(*[keep]*)

Return Index with duplicate values removed

duplicated(*[keep]*)

Indicate duplicate index values

equals(*other*)

Determine if two index objects is equal to *other*

factorize(*[sort, na\_sentinal]*)

Encode the object as an enumerated type or categorical variable

identical(*other*)

Similar to *equals()*, but check that object attributes and types are also equal

insert(*loc, item*)

Make new Index inserting new *item* at location *loc*

is\_(*other*)

Like *is()*, but works through views

is\_boolean()

Check if index only consists of Booleans

is\_categorical()

Check if the index holds categorical data

is\_floating()

Check if the index is a floating type

is\_integer()

Check if index only consists of integers

is\_interval()

Check if the index holds Interval objects

is\_mixed()

Check if the Index holds data with mixed types

is\_numeric()

Check if the index only consists of numeric data

is\_object()

Check if the index is of the *object* dtype

min(*[axis, skipna]*)

Return minimum value of index

max(*[axis, skipna]*)

Return maximum value of index

reindex(*target[, method, level, …]*)

Create index with *target* values

rename(*name[, inplace]*)

Alter Index or MultiIndex name

repeat(*repeats[, axis]*)

Repeat elements of an index

where(*cond[, other]*)

Replace values where the condition *cond* is False

take(*indices[, axis, allow\_fill, …]*)

Return a new Index of the values selected by *indices*

putmask(*mask, value*)

Return a new Index of the values set with *mask*

unique(*[level]*)

Return unique values in index (in order of appearance)

nunique(*[dropna]*)

Return number of unique elements in index

value\_counts(*[normalize, sort, …]*)

Return Series containing counts of unique values

## MultiIndex compatibility

set\_names(*names[, level, inplace]*)

Set Index or MultiIndex name

droplevel(*[level]*)

Return index with requested level(s) removed

## Missing values

fillna(*[value, downcast]*)

Fill NA/NaN values with specified value

dropna(*[how]*)

Return Index without NA/NaN values

isna()

Detect missing values (NaN, NaT, None)

notna()

Detect existing (non-missing) values

## Conversion

astype(*dtype[, copy]*)

Create an Index with values cast to dtypes

item()

Return the first element of the underlying data as a Python scalar

map(*mapper[, na\_action]*)

Map values using input correspondence (dict, series, or function)

ravel(*[order]*)

Return an *ndarray* of the flattened values of the underlying data

to\_list()

Return a list of the values

to\_native\_types(*[slicer]*)

Return formatted specified values of *self* (Deprecated)

to\_series(*[index, name]*)

Create a Series with both index and values equal to the index keys

to\_frame(*[index, name]*)

Create a DataFrame with a column containing index

view(*[cls]*)

Create a new view of the index

## Sorting

argsort(*\*args, \*\*kwargs*)

Return the integer indices that would sort the index

searchsorted(*value[, side, sorter]*)

Find indices where elements should be inserted to maintain order

sort\_values(*[return\_indexer, …]*)

Return a sorted copy of the index

## Time-specific operations

shift(*[periods, freq]*)

Shift index by desired number of time frequency increments

## Combining / joining / set operations

append(*other*)

Append a collection of Index values together

join(*other[, how, level, …]*)

Compute *join\_index* and indexers to conform data structures to the new index <meaing?>

intersection(*other[, sort]*)

Form the intersection of index and *other*

union(*other[, sort]*)

Form the union of index and *other*

difference(*other[, sort]*)

Return a new Index with elements of index not in *other*

symmetric\_difference(*other[, …]*)

Compute elements either in index or *other* but not in both

## Selecting

asof(*label*)

Return the label from the index, or, if not present, the previous one

asof\_locs(*where, mask*)

Return the locations (indices) of labels in the index

get\_indexer(*target[, method, limit, …]*)

Compute indexer and mask for new index given the current index

get\_indexer\_for(*target, \*\*kwargs*)

Guaranteed return of an indexer even when non-unique

get\_indexer\_non\_unique(*target*)

Compute indexer and mask for new index, given the current index

get\_level\_values(*level*)

Return an Index of values for requested level

get\_loc(*key[, method, tolerance]*)

Get integer location, slice, or Boolean mask for requested label

get\_slice\_bound(*label, side, kind*)

Calculate slice bound that corresponds to given label

get\_value(*series, key*)

Fast lookup of value from 1-dimensional *ndarray* (only use if one knows what one is doing)

isin(*values[, level]*)

Return a boolean array where index elements are in *values*, with same length as index

slice\_indexer(*[start, end, step, kind]*)

Compute the slice indexer for input labels and step. Index must be ordered and unique

slice\_locs(*[start, end, step, kind]*)

Compute slice locations for input labels. Index must be monotonic or unique.

## special cases

Float64Index

Int64Index

UInt64Index

# [pandas.RangeIndex](https://pandas.pydata.org/docs/reference/api/pandas.RangeIndex.html)

RangeIndex is a special case of Int64Index limited to representing monotonic ranges. It is the default Index used by DataFrame and Series when no explicit index is specified.

## constructor

RangeIndex(*[start, stop, step, dtype, copy, …]*)

## attributes

start

stop

step

## methods

from\_range(*data[, name, dtype]*)

Create RangeIndex from a range object

# [pandas.CategoricalIndex](https://pandas.pydata.org/docs/reference/api/pandas.CategoricalIndex.html)

## constructor

CategoricalIndex(*[data, categories, …]*)

Index based on underlying Categorical. Can only take limited values (categories)

## attributes

codes

categories

ordered

## methods

rename\_categories(*\*args, \*\*kwargs*)

Rename categories

reorder\_categories(*\*args, \*\*kwargs*)

Reorder categories as specified in *new\_categories*

add\_categories(*\*args, \*\*kwargs*)

Add new categories

remove\_categories(*\*args, \*\*kwargs*)

Remove the specified categories

remove\_unused\_categories(*\*args, \*\*kwargs*)

Remove categories which are not used

set\_categories(*\*args, \*\*kwargs*)

Set the categories to the specified *new\_categories*

as\_ordered(*\*args, \*\*kwargs*)

Set the categorical to be ordered

as\_unordered(*\*args, \*\*kwargs*)

Set the categorical to be unordered

map(*mapper*)

Map values using input correspondence (dict, Series, or function)

# [pandas.IntervalIndex](https://pandas.pydata.org/docs/reference/api/pandas.IntervalIndex.html#pandas.IntervalIndex)

## constructor

IntervalIndex(*data, closed=None, dtype=None, copy=False, name=None, verify\_integrity=True*)

Index of intervals that are closed on the same side.

## attributes

closed

length

is\_empty

is\_non\_overlapping\_monotonic

values

left

right

mid

## methods

from\_arrays(*left, right[, closed, name, …]*)

Construct from two arrays defining the left and right bounds

from\_tuples(*data[, closed, name, copy, dtype]*)

Construct an IntervalIndex from an array-like of tuples

from\_breaks(*breaks[, closed, name, copy, dtype]*)

Construct an IntervalIndex from an array of splits

contains(*\*args, \*\*kwargs*)

Check elementwise if the intervals contain the value

overlaps(*\*args, \*\*kwargs*)

Check elementwise if an interval overlaps the values in the IntervalArray

set\_closed(*\*args, \*\*kwargs*)

Return an IntervalArray identical to the current one, but closed on the specified side

to\_tuples(*\*args, \*\*kwargs*)

Return an ndarray of tuples of the form (left, right)

# [pandas.MultiIndex](https://pandas.pydata.org/docs/reference/api/pandas.MultiIndex.html)

A multi-level, or hierarchical, index for pandas objects

## attributes

names

nlevels

levshape

levels

codes

## methods

from\_arrays(*arrays[, sortorder, names]*)

Convert arrays to MultiIndex

from\_tuples(*tuples[, sortorder, names]*)

Convert list of tuples to MultiIndex

from\_product(*iterables[, sortorder, names]*)

Make MultiIndex from cartesian product of multiple iterables

from\_frame(*df[, sortorder, names]*)

Make MultiIndex from a DataFrame

set\_levels(*levels[, level, inplace, …]*)

Set new levels on MultiIndex

set\_codes(*codes[, level, inplace, …]*)

Set new codes on MultiIndex

to\_frame(*[index, name]*)

Create a DataFrame with the levels of the MultiIndex as columns

to\_flat\_index()

Convert MultiIndex to an Index of tuples containing the level values

is\_lexsorted()

Return True if the codes are lexicographically sorted

sortlevel(*[level, ascending, sort\_remaining]*)

Sort MultiIndex at the requested level

droplevel(*[level]*)

Return index with requested level(s) removed

swaplevel(*[i, j]*)

Swap level *i* with level *j*

reorder\_levels(*order*)

Rearrange levels using input *order*

remove\_unused\_levels()

Create MultiIndex from current that removes unused levels

get\_locs(*seq*)

Get location for a sequence of labels

get\_loc\_level(*key[, level, …]*)

Get location and sliced index for requested label(s)/level(s)

get\_indexer(*target[, method, …]*)

Compute indexer and mask for new index given the current index

get\_level\_values(*level*)

Return vector of label values for requested level

# [pandas.DatetimeIndex](https://pandas.pydata.org/docs/reference/api/pandas.DatetimeIndex.html)

## constructor

DatetimeIndex(*[data, freq, tz, normalize, …]*)

Immutable ndarray-like of datetime64 data

## attributes

year

month

day

hour

minute

second

microsecond

nanosecond

date

time

timetz

dayofyear

day\_of\_year

weekofyear

week

dayofweek

day\_of\_week

weekday

quarter

tz

freq

freqstr

is\_month\_start

is\_month\_end

is\_quarter\_start

is\_quarter\_end

is\_year\_start

is\_year\_end

is\_leap\_year

inferred\_freq

## selecting

indexer\_at\_time(*time[, asof]*)

Return index locations of values at particular time of day

indexer\_between\_time(*…[, …]*)

Return index locations of values between particular times of day

## time-specific operations

normalize(*\*args, \*\*kwargs*)

Convert times to midnight

strftime(*\*args, \*\*kwargs*)

Convert to Index using specified *date\_format*

sanp(*[freq]*)

Snap timestamps to nearest occurring frequency

tz\_convert(*tz*)

Convert tz-aware Datetime Array/Index from one timezone to another

tz\_localize(*tz[, ambiguous, …]*)

Localize tz-naïve Datetime Array/Index to tz-aware

round(*\*args, \*\*kwargs*)

Perform round operation on the data to the specified *freq*

floor(*\*args, \*\*kwargs*)

Perform floor operation on the data to the specified *freq*

ceil(*\*args, \*\*kwargs*)

Perform ceil operation on the data to the specified *freq*

month\_name(*\*args, \*\*kwargs*)

Return the month names of the DatetimeIndex with specified *locale*

day\_name(*\*args, \*\*kwargs*)

Return the day names of the DatetimeIndex with specified *locale*

## conversion

to\_period(*\*args, \*\*kwargs*)

Cast to PeriodArray/Index at a particular frequency

to\_perioddelta(*freq*)

Calculate TimedeltaArray of difference between values and index convert to PeriodArray at specified *freq*

to\_pydatetime(*\*args, \*\*kwargs*)

Return DatetimeArray/Index as object ndarray of datetime.datetime objects

to\_series(*[keep\_tz, index, name]*)

Create a Series with both index and values equal to the index keys, useful with map for returning an indexer based on an index

to\_frame(*[index, name]*)

Create a DataFrame with a column containing the index

## methods

mean(*\*args, \*\*kwargs*)

Return the mean value of the array

# [pandas.TimedeltaIndex](https://pandas.pydata.org/docs/reference/api/pandas.TimedeltaIndex.html)

## constructor

TimedeltaIndex(*[data, unit, freq, closed, …]*)

Immutable ndarray of timedelta64 data, can be boxed to timedelta objects

## attributes

days

seconds

microseconds

nanoseconds

components

inferred\_freq

## conversion

to\_pytimedelta(*\*args, \*\*kwargs*)

Return Timedelta Array/Index as object ndarray of datetime.timedelta objects

to\_series(*[index, name]*)

Create a Series with both index and values equal to the index keys

round(*\*args, \*\*kwargs*)

Perform round operation on the data to the specified *freq*

floor(*\*args, \*\*kwargs*)

Perform floor operation on the data to the specified *freq*

ceil(*\*args, \*\*kwargs*)

Perform ceil operation on the data to the specified *freq*

to\_frame(*[index, name]*)

Create a DataFrame with a column containing the index

## methods

mean(*\*args, \*\*kwargs*)

Return mean value of the array

# [pandas.PeriodIndex](https://pandas.pydata.org/docs/reference/api/pandas.PeriodIndex.html)

constructor

PeriodIndex(*[data, ordinal, freq, tz, …]*)

Immutable ndarray holding ordinal values indicating regular periods in time

## attributes

day

dayofweek

day\_of\_week

dayofyear

day\_of\_year

days\_in\_month

daysinmonth

end\_time

freq

freqstr

hour

is\_leap\_year

minute

month

quarter

qyear

second

start\_time

week

weekday

weekofyear

year

## methods

as\_freq(*[freq, how]*)

Convert PeriodArray/Index to the specified frequency *freq*

strftime(*\*args, \*\*kwargs*)

Convert to Index using specified *date\_format*

to\_timestamp(*[freq, how]*)

Cast to DatetimeArray/Index

# [pandas.IndexSlice](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.IndexSlice.html)

Create an object to more easily perform multi-index slicing

dfmi.loc[(slice(**None**), slice('B0', 'B1')), :]

Is equivalent to

idx = pd.IndexSlice

dfmi.loc[idx[:, 'B0':'B1'], :]

# pd.DateOffset

# [pandas GroupBy](https://pandas.pydata.org/pandas-docs/stable/reference/groupby.html)

## indexing, iteration

Gropuer()

\_\_iter\_\_()

groups

indices

get\_group(*name[, obj]*)

## function application

apply(*func, \*args, \*\*kwargs*)

Apply function *func* group-wise and combine results

agg(*func, \*args, \*\*kwargs*)

aggregate(*[func, engine, …]*)

Aggregate using one or more operations over the specified axis

transform(*func, \*args[, …]*)

Call function *func* producing a like-indexed Series/DataFrame on each group, and return a Series/DataFrame with same indexes as original object filled with transformed values

pipe(*func, \*args, \*\*kwargs*)

Apply a function *func* with arguments to this GroupBy object and return the result

## computations / descriptive stats

all(*[skipna]*)

Return True if all values in group are truthful

any(*[skipna]*)

Return True if any value in group is truthful

bfill(*[limit]*)

Backward fill the values

backfill(*[limit]*)

Backward fill the values

count()

Compute count of group, excluding missing values

cumcount(*[ascending]*)

Number each item in each group from 0 to (length-1)

cummax(*[axis]*)

Cumulative max for each group

cumin(*[axis]*)

Cumulative minimum for each group

cumprod(*[axis]*)

Cumulative product for each group

cumsum(*[axis]*)

Cumulative sum for each group

ffill(*[limit]*)

Forward fill the values

first(*numeric\_only, min\_count]*)

Compute first of group values

head(*[n]*)

Return first *n* rows of each group

last(*[numeric\_only, min\_count]*)

Compute last group of values

max(*[numeric\_only, min\_count]*)

Compute max group of values

mean(*[numeric\_only]*)

Compute mean of groups, excluding missing values

median(*[numeric\_only]*)

Compute median of groups, excluding missing values

min(*[numeric\_only, min\_count]*)

Compute min of group values

ngroup(*[ascending]*)

Number each group from 0 to n

nth(*n[, dropna]*)

Take the *n-*th row from each group if *n* is an int, or subset of rows if *n* is list of ints

ohlc()

Compute open, high, low, and close values of group, excluding missing values

pad(*[limit]*)

Forward fill the values

prod(*[numeric\_only, min\_count]*)

Product of group values

rank(*[method, asceding, na\_option, …]*)

Provide the rank of values within each group

pct\_change(*[periods, fill\_method, …]*)

Calculate percentage-change of each value to previous entry in group

size()

Compute group sizes

sem(*[ddof]*)

Compute standard error of the mean of groups, excluding missing values

std(*[ddof]*)

Compute standard deviation of groups, excluding missing values

sum(*[numeric\_only, min\_count]*)

Compute sum of group values

var(*[ddof]*)

Compute variance of groups, excluding missing values

tail(*[n]*)

Return last *n* rows of each group

## DF/Series Shared

SeriesGroupBy versions lacks axis argument

corr(*[method, min\_periods]*)

Compute pairwise correlation of columns, excluding null values

cov(*[min\_periods, ddof]*)

Compute pairwise covariance of columns, excluding null values

describe(*\*\*kwargs*)

Generate descriptive statistics

diff(*[periods, axis]*)

First discrete difference of elements

fillna(*value[, method, axis, …]*)

Fill NaN/Null values using specified method

filter(*func[, dropna]*)

Return a copy of a DataFrame excluding elements which do not satisfy boolean criterion of *func*

hist(*…*)

Make a histogram of the DataFrame

idxmax(*[axis, skipna]*)

Return index of first occurrence of maximum over given axis

idxmin(*[axis, skipna]*)

Return index of first occurrence of minimum over given axis

mad(*[axis, skipna, level]*)

Return mean absolute deviation of values over the given axis

nunique(*[dropna]*)

Return DataFrame with counts of unique elements in each position

plot()

quantile(*[q, interpolation]*)

Return group values at the given quantile (like *np.percentile*)

resample(*rule, \*args, \*\*kwargs*)

Provide resampling when using a TimeGrouper

sample(*[n, frac, replace, …]*)

Return a random sample of items from each group

shift(*[periods, freq, …]*)

Shift each group by *periods* observations

skew(*[axis, skipna, level, numeric\_only, …]*)

Return unbiased skew over requested axis

take(*indices[, axis, …]*)

Return the elements in given positional *indices* along an axis

tshift(*periods[, freq, axis]*)

Shift the time index, using index’s frequency (Deprecated)

## Only Series

nlargest(*n[, keep]*)

Return the largest *n* elements

nsmallest(*n[, keep]*)

Return the smallest *n* elements

nunique(*[dropna]*)

Number of unique elements in group

unique

Return unique values of Series object

value\_counts(*[normalize, sort, ascending, bins, dropna]*)

is\_monotonic\_increasing

is\_monotonic\_decreasing

## Only DataFrame

corrwith(*other[, axis, drop, method]*)

Compute pairwise correlation

boxplot(*[subplots, column, …]*)

Make box plots from DataFrameGroupBy data

# [pandas Window](https://pandas.pydata.org/pandas-docs/stable/reference/window.html)

## rolling window functions

count()

sum(*\*args, \*\*kwargs*)

mean(*\*args, \*\*kwargs*)

median(*\*\*kwargs*)

var(*[ddof]*)

min(*[ddof]*)

min(*\*args, \*\*kwargs*)

max(*\*args, \*\*kwargs*)

corr(*[other, pairwise]*)

cov(*[other, pairwise, ddof]*)

skew(*\*\*kwargs*)

kurt(*\*\*kwargs*)

apply(*func[, raw, engine, …]*)

aggregate(*func, \*args, \*\*kwargs*)

quantile(*quantile[, interpolation]*)

sem(*[ddof]*)

## weighted window functions

mean(*\*args, \*\*kwargs*)

sum(*\*args, \*\*kwargs*)

var(*[ddof]*)

std(*[ddof]*)

## expanding window functions

count()

sum(*\*args, \*\*kwargs*)

mean(*\*args, \*\*kwargs*)

median(*\*\*kwargs*)

var(*[ddof]*)

min(*[ddof]*)

min(*\*args, \*\*kwargs*)

max(*\*args, \*\*kwargs*)

corr(*[other, pairwise]*)

cov(*[other, pairwise, ddof]*)

skew(*\*\*kwargs*)

kurt(*\*\*kwargs*)

apply(*func[, raw, engine, …]*)

aggregate(*func, \*args, \*\*kwargs*)

quantile(*quantile[, interpolation]*)

sem(*[ddof]*)

## exponentially-weighed window functions

mean(*\*args, \*\*kwargs*)

std(*[bias]*)

var(*[bias]*)

corr(*[other, pairwise]*)

cov(*[other, …]*)

## window indexer

BaseIndexer(*[index\_array, …]*)

FixedForwardWindowIndexer(*[…]*)

VariableOffsetWindowIndexer(*[…]*)

# [pandas Resampling](https://pandas.pydata.org/pandas-docs/stable/reference/resampling.html)

## Indexing, iteration

\_\_iter\_\_()

groups

indices

get\_group()

## Function application

apply(*func, \*args, \*\*kwargs*)

aggregate(*func, \*args, \*\*kwargs*)

transform(*func, \*args, \*\*kwargs*)

pipe(*func, \*args, \*\*kwargs*)

## Upsampling

ffill(*[limit]*)

backfill(*[limit]*)

bfill(*[limit]*)

pad(*[limit]*)

nearest(*[limit]*)

fillna(*method[, limit]*)

asfreq(*[fill\_value]*)

interpolate(*[method, axis, limit, …]*)

## Computations / descriptive stats

count()

nunique(*[\_method]*)

first(*[\_method, min\_count]*)

last(*[\_method, min\_count]*)

max(*[\_method, min\_count]*)

mean(*[\_method]*)

median(*[\_method]*)

min(*[\_method, min\_count]*)

ohlc(*[\_method]*)

prod(*[\_method, min\_count]*)

size()

sem(*[\_method]*)

std(*[ddof]*)

sum(*[\_method, min\_count]*)

var(*[ddof]*)

quantile(*[q]*)