In this tutorial, you will learn about **imports** in Python, get some tips for working with unfamiliar libraries (and the objects they return), and dig into **operator overloading**.

Imports

So far we've talked about types and functions which are built-in to the language.

But one of the best things about Python (especially if you're a data scientist) is the vast number of high-quality custom libraries that have been written for it.

Some of these libraries are in the "standard library", meaning you can find them anywhere you run Python. Other libraries can be easily added, even if they aren't always shipped with Python.

Either way, we'll access this code with **imports**.

We'll start our example by importing `math` from the standard library.

In [1]:
```python
import math

print("It's math! It has type {}".format(type(math)))
```

```
It's math! It has type <class 'module'>
```

`math` is a module. A module is just a collection of variables (a *namespace*, if you like) defined by someone else. We can see all the names in `math` using the built-in function `dir()`.

In [2]:
```python
print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt', 'ceil',
'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'e
xp', 'exp2', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'is
qrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'na
n', 'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

We can access these variables using dot syntax. Some of them refer to simple values, like `math.pi`:

In [3]:
```python
print("pi to 4 significant digits = {:.4}".format(math.pi))
```

```
pi to 4 significant digits = 3.142
```

But most of what we'll find in the module are functions, like `math.log`:

Loading [MathJax]/extensions/Safe.js

In [4]:
```python
math.log(32, 2)
```

Out[4]: 5.0

Of course, if we don't know what `math.log` does, we can call `help()` on it:

In [5]:
```
help(math.log)
```

```
Help on built-in function log in module math:

log(...)
    log(x, [base=math.e])
    Return the logarithm of x to the given base.

    If the base not specified, returns the natural logarithm (base e) of x.
```

We can also call `help()` on the module itself. This will give us the combined documentation for *all* the functions and values in the module (as well as a high-level description of the module). Click the "output" button to see the whole `math` help page.

In [6]:
```
help(math)
```

Loading [MathJax]/extensions/Safe.js

```
Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.11/library/math.html

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

        The result is between 0 and pi.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

        The result is between -pi/2 and pi/2.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

        The result is between -pi/2 and pi/2.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    cbrt(x, /)
        Return the cube root of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.
```

```
    comb(n, k, /)
        Number of ways to choose k items from n items without repetition and
without order.

        Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates
        to zero when k > n.

        Also called the binomial coefficient because it is equivalent
        to the coefficient of k-th term in polynomial expansion of the
        expression (1 + x)**n.

        Raises TypeError if either of the arguments are not integers.
        Raises ValueError if either of the arguments are negative.

    copysign(x, y, /)
        Return a float with the magnitude (absolute value) of x but the sign
of y.

        On platforms that support signed zeros, copysign(1.0, -0.0)
        returns -1.0.

    cos(x, /)
        Return the cosine of x (measured in radians).

    cosh(x, /)
        Return the hyperbolic cosine of x.

    degrees(x, /)
        Convert angle x from radians to degrees.

    dist(p, q, /)
        Return the Euclidean distance between two points p and q.

        The points should be specified as sequences (or iterables) of
        coordinates.  Both inputs must have the same dimension.

        Roughly equivalent to:
            sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))

    erf(x, /)
        Error function at x.

    erfc(x, /)
        Complementary error function at x.

    exp(x, /)
        Return e raised to the power of x.

    exp2(x, /)
        Return 2 raised to the power of x.

    expm1(x, /)
        Return exp(x)-1.

        This function avoids the loss of precision involved in the direct ev
aluation of exp(x)-1 for small x.
```

Loading [MathJax]/extensions/Safe.js

```
fabs(x, /)
    Return the absolute value of the float x.

factorial(n, /)
    Find n!.

    Raise a ValueError if x is negative or non-integral.

floor(x, /)
    Return the floor of x as an Integral.

    This is the largest integer <= x.

fmod(x, y, /)
    Return fmod(x, y), according to platform C.

    x % y may differ.

frexp(x, /)
    Return the mantissa and exponent of x, as pair (m, e).

    m is a float and e is an int, such that x = m * 2.**e.
    If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

fsum(seq, /)
    Return an accurate floating point sum of values in the iterable seq.

    Assumes IEEE-754 floating point arithmetic.

gamma(x, /)
    Gamma function at x.

gcd(*integers)
    Greatest Common Divisor.

hypot(...)
    hypot(*coordinates) -> value

    Multidimensional Euclidean distance from the origin to a point.

    Roughly equivalent to:
        sqrt(sum(x**2 for x in coordinates))

    For a two dimensional point (x, y), gives the hypotenuse
    using the Pythagorean theorem:  sqrt(x*x + y*y).

    For example, the hypotenuse of a 3/4/5 right triangle is:

        >>> hypot(3.0, 4.0)
        5.0

isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)
    Determine whether two floating point numbers are close in value.

      rel_tol
```

```
                maximum difference for being considered "close", relative to the
                magnitude of the input values
              abs_tol
                maximum difference for being considered "close", regardless of t
he
                magnitude of the input values

            Return True if a is close in value to b, and False otherwise.

            For the values to be considered close, the difference between them
            must be smaller than at least one of the tolerances.

            -inf, inf and NaN behave similarly to the IEEE 754 Standard.  That
            is, NaN is not close to anything, even itself.  inf and -inf are
            only close to themselves.

        isfinite(x, /)
            Return True if x is neither an infinity nor a NaN, and False otherwi
se.

        isinf(x, /)
            Return True if x is a positive or negative infinity, and False other
wise.

        isnan(x, /)
            Return True if x is a NaN (not a number), and False otherwise.

        isqrt(n, /)
            Return the integer part of the square root of the input.

        lcm(*integers)
            Least Common Multiple.

        ldexp(x, i, /)
            Return x * (2**i).

            This is essentially the inverse of frexp().

        lgamma(x, /)
            Natural logarithm of absolute value of Gamma function at x.

        log(...)
            log(x, [base=math.e])
            Return the logarithm of x to the given base.

            If the base not specified, returns the natural logarithm (base e) of
x.

        log10(x, /)
            Return the base 10 logarithm of x.

        log1p(x, /)
            Return the natural logarithm of 1+x (base e).

            The result is computed in a way which is accurate for x near zero.
```

```
    log2(x, /)
        Return the base 2 logarithm of x.

    modf(x, /)
        Return the fractional and integer parts of x.

        Both results carry the sign of x and are floats.

    nextafter(x, y, /)
        Return the next floating-point value after x towards y.

    perm(n, k=None, /)
        Number of ways to choose k items from n items without repetition and
with order.

        Evaluates to n! / (n - k)! when k <= n and evaluates
        to zero when k > n.

        If k is not specified or is None, then k defaults to n
        and the function returns n!.

        Raises TypeError if either of the arguments are not integers.
        Raises ValueError if either of the arguments are negative.

    pow(x, y, /)
        Return x**y (x to the power of y).

    prod(iterable, /, *, start=1)
        Calculate the product of all the elements in the input iterable.

        The default start value for the product is 1.

        When the iterable is empty, return the start value.  This function i
s
        intended specifically for use with numeric values and may reject
        non-numeric types.

    radians(x, /)
        Convert angle x from degrees to radians.

    remainder(x, y, /)
        Difference between x and the closest integer multiple of y.

        Return x - n*y where n*y is the closest integer multiple of y.
        In the case where x is exactly halfway between two multiples of
        y, the nearest even value of n is used. The result is always exact.

    sin(x, /)
        Return the sine of x (measured in radians).

    sinh(x, /)
        Return the hyperbolic sine of x.

    sqrt(x, /)
        Return the square root of x.
```

Loading [MathJax]/extensions/Safe.js

```
    tan(x, /)
        Return the tangent of x (measured in radians).

    tanh(x, /)
        Return the hyperbolic tangent of x.

    trunc(x, /)
        Truncates the Real x to the nearest Integral toward 0.

        Uses the __trunc__ magic method.

    ulp(x, /)
        Return the value of the least significant bit of the float x.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    /Users/akarate/.pyenv/versions/3.11.5/Library/Frameworks/Python.framewor
k/Versions/3.11/lib/python3.11/lib-dynload/math.cpython-311-darwin.so
```

Other import syntax

If we know we'll be using functions in `math` frequently we can import it under a shorter alias to save some typing (though in this case "math" is already pretty short).

In [7]:

```python
import math as mt
mt.pi
```

Out[7]: 3.141592653589793

> You may have seen code that does this with certain popular libraries like Pandas, Numpy, Tensorflow, or Matplotlib. For example, it's a common convention to `import numpy as np` and `import pandas as pd`.

The `as` simply renames the imported module. It's equivalent to doing something like:

In [8]:

```python
import math
mt = math
```

Wouldn't it be great if we could refer to all the variables in the `math` module by themselves? i.e. if we could just refer to `pi` instead of `math.pi` or `mt.pi`? Good news: we can do that.

Loading [MathJax]/extensions/Safe.js

In [9]:
```
from math import *
print(pi, log(32, 2))
```

```
3.141592653589793 5.0
```

`import *` makes all the module's variables directly accessible to you (without any dotted prefix).

Bad news: some purists might grumble at you for doing this.

Worse: they kind of have a point.

In [10]:
```
from math import *
from numpy import *
print(pi, log(32, 2))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[10], line 3
      1 from math import *
      2 from numpy import *
----> 3 print(pi, log(32, 2))

TypeError: return arrays must be of ArrayType
```

What has happened? It worked before!

These kinds of "star imports" can occasionally lead to weird, difficult-to-debug situations.

The problem in this case is that the `math` and `numpy` modules both have functions called `log`, but they have different semantics. Because we import from `numpy` second, its `log` overwrites (or "shadows") the `log` variable we imported from `math`.

A good compromise is to import only the specific things we'll need from each module:

In [11]:
```
from math import log, pi
from numpy import asarray
```

Submodules

We've seen that modules contain variables which can refer to functions or values. Something to be aware of is that they can also have variables referring to *other modules*.

In [12]:
```
import numpy
print("numpy.random is a", type(numpy.random))
print("it contains names such as...",
      dir(numpy.random)[-15:]
     )
```

Loading [MathJax]/extensions/Safe.js

```
numpy.random is a <class 'module'>
it contains names such as... ['set_bit_generator', 'set_state', 'shuffle',
'standard_cauchy', 'standard_exponential', 'standard_gamma', 'standard_norma
l', 'standard_t', 'test', 'triangular', 'uniform', 'vonmises', 'wald', 'weib
ull', 'zipf']
```

So if we import `numpy` as above, then calling a function in the `random` "submodule" will require *two* dots.

In [13]:
```
# Roll 10 dice
rolls = numpy.random.randint(low=1, high=6, size=10)
rolls
```

Out[13]:
```
array([1, 1, 4, 3, 2, 1, 5, 1, 2, 2])
```

Oh the places you'll go, oh the objects you'll see

So after 6 lessons, you're a pro with ints, floats, bools, lists, strings, and dicts (right?).

Even if that were true, it doesn't end there. As you work with various libraries for specialized tasks, you'll find that they define their own types which you'll have to learn to work with. For example, if you work with the graphing library `matplotlib`, you'll be coming into contact with objects it defines which represent Subplots, Figures, TickMarks, and Annotations. `pandas` functions will give you DataFrames and Series.

In this section, I want to share with you a quick survival guide for working with strange types.

Three tools for understanding strange objects

In the cell above, we saw that calling a `numpy` function gave us an "array". We've never seen anything like this before (not in this course anyways). But don't panic: we have three familiar builtin functions to help us here.

1: `type()` (what is this thing?)

In [14]:
```
type(rolls)
```

Out[14]:
```
numpy.ndarray
```

2: `dir()` (what can I do with it?)

In [15]:
```
print(dir(rolls))
```

Loading [MathJax]/extensions/Safe.js

```
['T', '__abs__', '__add__', '__and__', '__array__', '__array_finalize__', '_
_array_function__', '__array_interface__', '__array_prepare__', '__array_pri
ority__', '__array_struct__', '__array_ufunc__', '__array_wrap__', '__bool_
_', '__class__', '__class_getitem__', '__complex__', '__contains__', '__copy
__', '__deepcopy__', '__delattr__', '__delitem__', '__dir__', '__divmod__',
'__dlpack__', '__dlpack_device__', '__doc__', '__eq__', '__float__', '__floo
rdiv__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getst
ate__', '__gt__', '__hash__', '__iadd__', '__iand__', '__ifloordiv__', '__il
shift__', '__imatmul__', '__imod__', '__imul__', '__index__', '__init__', '_
_init_subclass__', '__int__', '__invert__', '__ior__', '__ipow__', '__irshif
t__', '__isub__', '__iter__', '__itruediv__', '__ixor__', '__le__', '__len_
_', '__lshift__', '__lt__', '__matmul__', '__mod__', '__mul__', '__ne__', '_
_neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__',
'__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__',
'__rlshift__', '__rmatmul__', '__rmod__', '__rmul__', '__ror__', '__rpow__',
'__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__seta
ttr__', '__setitem__', '__setstate__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', '__truediv__', '__xor__', 'all', 'any', 'argmax', 'argmi
n', 'argpartition', 'argsort', 'astype', 'base', 'byteswap', 'choose', 'cli
p', 'compress', 'conj', 'conjugate', 'copy', 'ctypes', 'cumprod', 'cumsum',
'data', 'diagonal', 'dot', 'dtype', 'dump', 'dumps', 'fill', 'flags', 'fla
t', 'flatten', 'getfield', 'imag', 'item', 'itemset', 'itemsize', 'max', 'me
an', 'min', 'nbytes', 'ndim', 'newbyteorder', 'nonzero', 'partition', 'pro
d', 'ptp', 'put', 'ravel', 'real', 'repeat', 'reshape', 'resize', 'round',
'searchsorted', 'setfield', 'setflags', 'shape', 'size', 'sort', 'squeeze',
'std', 'strides', 'sum', 'swapaxes', 'take', 'tobytes', 'tofile', 'tolist',
'tostring', 'trace', 'transpose', 'var', 'view']
```

In [16]:
```python
# If I want the average roll, the "mean" method looks promising...
rolls.mean()
```

Out[16]:
```
2.2
```

In [17]:
```python
# Or maybe I just want to turn the array into a list, in which case I can us
rolls.tolist()
```

Out[17]:
```
[1, 1, 4, 3, 2, 1, 5, 1, 2, 2]
```

3: `help()` (tell me more)

In [18]:
```python
# That "ravel" attribute sounds interesting. I'm a big classical music fan.
help(rolls.ravel)
```

Loading [MathJax]/extensions/Safe.js

```
Help on built-in function ravel:

ravel(...) method of numpy.ndarray instance
    a.ravel([order])

    Return a flattened array.

    Refer to `numpy.ravel` for full documentation.

    See Also
    --------
    numpy.ravel : equivalent function

    ndarray.flat : a flat iterator on the array.
```

In [19]:
```
# Okay, just tell me everything there is to know about numpy.ndarray
# (Click the "output" button to see the novel-length output)
help(rolls)
```

Loading [MathJax]/extensions/Safe.js

```
Help on ndarray object:

class ndarray(builtins.object)
 |  ndarray(shape, dtype=float, buffer=None, offset=0,
 |          strides=None, order=None)
 |
 |  An array object represents a multidimensional, homogeneous array
 |  of fixed-size items.  An associated data-type object describes the
 |  format of each element in the array (its byte-order, how many bytes it
 |  occupies in memory, whether it is an integer, a floating point number,
 |  or something else, etc.)
 |
 |  Arrays should be constructed using `array`, `zeros` or `empty` (refer
 |  to the See Also section below).  The parameters given here refer to
 |  a low-level method (`ndarray(...)`) for instantiating an array.
 |
 |  For more information, refer to the `numpy` module and examine the
 |  methods and attributes of an array.
 |
 |  Parameters
 |  ----------
 |  (for the __new__ method; see Notes below)
 |
 |  shape : tuple of ints
 |      Shape of created array.
 |  dtype : data-type, optional
 |      Any object that can be interpreted as a numpy data type.
 |  buffer : object exposing buffer interface, optional
 |      Used to fill the array with data.
 |  offset : int, optional
 |      Offset of array data in buffer.
 |  strides : tuple of ints, optional
 |      Strides of data in memory.
 |  order : {'C', 'F'}, optional
 |      Row-major (C-style) or column-major (Fortran-style) order.
 |
 |  Attributes
 |  ----------
 |  T : ndarray
 |      Transpose of the array.
 |  data : buffer
 |      The array's elements, in memory.
 |  dtype : dtype object
 |      Describes the format of the elements in the array.
 |  flags : dict
 |      Dictionary containing information related to memory use, e.g.,
 |      'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.
 |  flat : numpy.flatiter object
 |      Flattened version of the array as an iterator.  The iterator
 |      allows assignments, e.g., ``x.flat = 3`` (See `ndarray.flat` for
 |      assignment examples; TODO).
 |  imag : ndarray
 |      Imaginary part of the array.
 |  real : ndarray
 |      Real part of the array.
 |  size : int
```

```
 |      Number of elements in the array.
 |  itemsize : int
 |      The memory use of each array element in bytes.
 |  nbytes : int
 |      The total number of bytes required to store the array data,
 |      i.e., ``itemsize * size``.
 |  ndim : int
 |      The array's number of dimensions.
 |  shape : tuple of ints
 |      Shape of the array.
 |  strides : tuple of ints
 |      The step-size required to move from one element to the next in
 |      memory. For example, a contiguous ``(3, 4)`` array of type
 |      ``int16`` in C-order has strides ``(8, 2)``.  This implies that
 |      to move from element to element in memory requires jumps of 2 bytes.
 |      To move from row-to-row, one needs to jump 8 bytes at a time
 |      (``2 * 4``).
 |  ctypes : ctypes object
 |      Class containing properties of the array needed for interaction
 |      with ctypes.
 |  base : ndarray
 |      If the array is a view into another array, that array is its `base`
 |      (unless that array is also a view).  The `base` array is where the
 |      array data is actually stored.
 |
 |  See Also
 |  --------
 |  array : Construct an array.
 |  zeros : Create an array, each element of which is zero.
 |  empty : Create an array, but leave its allocated memory unchanged (i.e.,
 |          it contains "garbage").
 |  dtype : Create a data-type.
 |  numpy.typing.NDArray : An ndarray alias :term:`generic <generic type>`
 |                         w.r.t. its `dtype.type <numpy.dtype.type>`.
 |
 |  Notes
 |  -----
 |  There are two modes of creating an array using ``__new__``:
 |
 |  1. If `buffer` is None, then only `shape`, `dtype`, and `order`
 |     are used.
 |  2. If `buffer` is an object exposing the buffer interface, then
 |     all keywords are interpreted.
 |
 |  No ``__init__`` method is needed because the array is fully initialized
 |  after the ``__new__`` method.
 |
 |  Examples
 |  --------
 |  These examples illustrate the low-level `ndarray` constructor.  Refer
 |  to the `See Also` section above for easier ways of constructing an
 |  ndarray.
 |
 |  First mode, `buffer` is None:
 |
 |  >>> np.ndarray(shape=(2,2), dtype=float, order='F')
```

```
 |  array([[0.0e+000, 0.0e+000], # random
 |         [     nan, 2.5e-323]])
 |
 |  Second mode:
 |
 |  >>> np.ndarray((2,), buffer=np.array([1,2,3]),
 |  ...            offset=np.int_().itemsize,
 |  ...            dtype=int) # offset = 1*itemsize, i.e. skip first element
 |  array([2, 3])
 |
 |  Methods defined here:
 |
 |  __abs__(self, /)
 |      abs(self)
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __array__(...)
 |      a.__array__([dtype], /)
 |
 |      Returns either a new reference to self if dtype is not given or a ne
w array
 |      of provided data type if dtype is different from the current dtype o
f the
 |      array.
 |
 |  __array_finalize__(...)
 |      a.__array_finalize__(obj, /)
 |
 |      Present so subclasses can call super. Does nothing.
 |
 |  __array_function__(...)
 |
 |  __array_prepare__(...)
 |      a.__array_prepare__(array[, context], /)
 |
 |      Returns a view of `array` with the same type as self.
 |
 |  __array_ufunc__(...)
 |
 |  __array_wrap__(...)
 |      a.__array_wrap__(array[, context], /)
 |
 |      Returns a view of `array` with the same type as self.
 |
 |  __bool__(self, /)
 |      True if self else False
 |
 |  __complex__(...)
 |
 |  __contains__(self, key, /)
 |      Return key in self.
```

Loading [MathJax]/extensions/Safe.js

```
 |
 |  __copy__(...)
 |      a.__copy__()
 |
 |      Used if :func:`copy.copy` is called on an array. Returns a copy of t
he array.
 |
 |      Equivalent to ``a.copy(order='K')``.
 |
 |  __deepcopy__(...)
 |      a.__deepcopy__(memo, /)
 |
 |      Used if :func:`copy.deepcopy` is called on an array.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __divmod__(self, value, /)
 |      Return divmod(self, value).
 |
 |  __dlpack__(...)
 |      a.__dlpack__(*, stream=None)
 |
 |      DLPack Protocol: Part of the Array API.
 |
 |  __dlpack_device__(...)
 |      a.__dlpack_device__()
 |
 |      DLPack Protocol: Part of the Array API.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __float__(self, /)
 |      float(self)
 |
 |  __floordiv__(self, value, /)
 |      Return self//value.
 |
 |  __format__(...)
 |      Default object formatter.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Return self+=value.
 |
 |  __iand__(self, value, /)
 |      Return self&=value.
```

```
 |  
 |  __ifloordiv__(self, value, /)
 |      Return self//=value.
 |  
 |  __ilshift__(self, value, /)
 |      Return self<<=value.
 |  
 |  __imatmul__(self, value, /)
 |      Return self@=value.
 |  
 |  __imod__(self, value, /)
 |      Return self%=value.
 |  
 |  __imul__(self, value, /)
 |      Return self*=value.
 |  
 |  __index__(self, /)
 |      Return self converted to an integer, if self is suitable for use as
an index into a list.
 |  
 |  __int__(self, /)
 |      int(self)
 |  
 |  __invert__(self, /)
 |      ~self
 |  
 |  __ior__(self, value, /)
 |      Return self|=value.
 |  
 |  __ipow__(self, value, /)
 |      Return self**=value.
 |  
 |  __irshift__(self, value, /)
 |      Return self>>=value.
 |  
 |  __isub__(self, value, /)
 |      Return self-=value.
 |  
 |  __iter__(self, /)
 |      Implement iter(self).
 |  
 |  __itruediv__(self, value, /)
 |      Return self/=value.
 |  
 |  __ixor__(self, value, /)
 |      Return self^=value.
 |  
 |  __le__(self, value, /)
 |      Return self<=value.
 |  
 |  __len__(self, /)
 |      Return len(self).
 |  
 |  __lshift__(self, value, /)
 |      Return self<<value.
 |  
```

Loading [MathJax]/extensions/Safe.js

```
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __matmul__(self, value, /)
 |      Return self@value.
 |
 |  __mod__(self, value, /)
 |      Return self%value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __neg__(self, /)
 |      -self
 |
 |  __or__(self, value, /)
 |      Return self|value.
 |
 |  __pos__(self, /)
 |      +self
 |
 |  __pow__(self, value, mod=None, /)
 |      Return pow(self, value, mod).
 |
 |  __radd__(self, value, /)
 |      Return value+self.
 |
 |  __rand__(self, value, /)
 |      Return value&self.
 |
 |  __rdivmod__(self, value, /)
 |      Return divmod(value, self).
 |
 |  __reduce__(...)
 |      a.__reduce__()
 |
 |      For pickling.
 |
 |  __reduce_ex__(...)
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __rfloordiv__(self, value, /)
 |      Return value//self.
 |
 |  __rlshift__(self, value, /)
 |      Return value<<self.
 |
 |  __rmatmul__(self, value, /)
 |      Return value@self.
 |
```

Loading [MathJax]/extensions/Safe.js

```
 |  __rmod__(self, value, /)
 |      Return value%self.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  __ror__(self, value, /)
 |      Return value|self.
 |
 |  __rpow__(self, value, mod=None, /)
 |      Return pow(value, self, mod).
 |
 |  __rrshift__(self, value, /)
 |      Return value>>self.
 |
 |  __rshift__(self, value, /)
 |      Return self>>value.
 |
 |  __rsub__(self, value, /)
 |      Return value-self.
 |
 |  __rtruediv__(self, value, /)
 |      Return value/self.
 |
 |  __rxor__(self, value, /)
 |      Return value^self.
 |
 |  __setitem__(self, key, value, /)
 |      Set self[key] to value.
 |
 |  __setstate__(...)
 |      a.__setstate__(state, /)
 |
 |      For unpickling.
 |
 |      The `state` argument must be a sequence that contains the following
 |      elements:
 |
 |      Parameters
 |      ----------
 |      version : int
 |          optional pickle version. If omitted defaults to 0.
 |      shape : tuple
 |      dtype : data-type
 |      isFortran : bool
 |      rawdata : string or list
 |          a binary string with the data (or a list if 'a' is an object arr
ay)
 |
 |  __sizeof__(...)
 |      Size of object in memory, in bytes.
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  __sub__(self, value, /)
```

Loading [MathJax]/extensions/Safe.js

```
 |      Return self-value.
 |  
 |  __truediv__(self, value, /)
 |      Return self/value.
 |  
 |  __xor__(self, value, /)
 |      Return self^value.
 |  
 |  all(...)
 |      a.all(axis=None, out=None, keepdims=False, *, where=True)
 |  
 |      Returns True if all elements evaluate to True.
 |  
 |      Refer to `numpy.all` for full documentation.
 |  
 |      See Also
 |      --------
 |      numpy.all : equivalent function
 |  
 |  any(...)
 |      a.any(axis=None, out=None, keepdims=False, *, where=True)
 |  
 |      Returns True if any of the elements of `a` evaluate to True.
 |  
 |      Refer to `numpy.any` for full documentation.
 |  
 |      See Also
 |      --------
 |      numpy.any : equivalent function
 |  
 |  argmax(...)
 |      a.argmax(axis=None, out=None, *, keepdims=False)
 |  
 |      Return indices of the maximum values along the given axis.
 |  
 |      Refer to `numpy.argmax` for full documentation.
 |  
 |      See Also
 |      --------
 |      numpy.argmax : equivalent function
 |  
 |  argmin(...)
 |      a.argmin(axis=None, out=None, *, keepdims=False)
 |  
 |      Return indices of the minimum values along the given axis.
 |  
 |      Refer to `numpy.argmin` for detailed documentation.
 |  
 |      See Also
 |      --------
 |      numpy.argmin : equivalent function
 |  
 |  argpartition(...)
 |      a.argpartition(kth, axis=-1, kind='introselect', order=None)
 |  
 |      Returns the indices that would partition this array.
```

Loading [MathJax]/extensions/Safe.js

```
 |
 |      Refer to `numpy.argpartition` for full documentation.
 |
 |      .. versionadded:: 1.8.0
 |
 |      See Also
 |      --------
 |      numpy.argpartition : equivalent function
 |
 |  argsort(...)
 |      a.argsort(axis=-1, kind=None, order=None)
 |
 |      Returns the indices that would sort this array.
 |
 |      Refer to `numpy.argsort` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.argsort : equivalent function
 |
 |  astype(...)
 |      a.astype(dtype, order='K', casting='unsafe', subok=True, copy=True)
 |
 |      Copy of the array, cast to a specified type.
 |
 |      Parameters
 |      ----------
 |      dtype : str or dtype
 |          Typecode or data-type to which the array is cast.
 |      order : {'C', 'F', 'A', 'K'}, optional
 |          Controls the memory layout order of the result.
 |          'C' means C order, 'F' means Fortran order, 'A'
 |          means 'F' order if all the arrays are Fortran contiguous,
 |          'C' order otherwise, and 'K' means as close to the
 |          order the array elements appear in memory as possible.
 |          Default is 'K'.
 |      casting : {'no', 'equiv', 'safe', 'same_kind', 'unsafe'}, optional
 |          Controls what kind of data casting may occur. Defaults to 'unsaf
e'
 |          for backwards compatibility.
 |
 |            * 'no' means the data types should not be cast at all.
 |            * 'equiv' means only byte-order changes are allowed.
 |            * 'safe' means only casts which can preserve values are allowe
d.
 |            * 'same_kind' means only safe casts or casts within a kind,
 |              like float64 to float32, are allowed.
 |            * 'unsafe' means any data conversions may be done.
 |      subok : bool, optional
 |          If True, then sub-classes will be passed-through (default), othe
rwise
 |          the returned array will be forced to be a base-class array.
 |      copy : bool, optional
 |          By default, astype always returns a newly allocated array. If th
is
 |          is set to false, and the `dtype`, `order`, and `subok`
```

Loading [MathJax]/extensions/Safe.js

```
 |          requirements are satisfied, the input array is returned instead
 |          of a copy.
 |
 |      Returns
 |      -------
 |      arr_t : ndarray
 |          Unless `copy` is False and the other conditions for returning th
e input
 |          array are satisfied (see description for `copy` input paramete
r), `arr_t`
 |          is a new array of the same shape as the input array, with dtype,
order
 |          given by `dtype`, `order`.
 |
 |      Notes
 |      -----
 |      .. versionchanged:: 1.17.0
 |         Casting between a simple data type and a structured one is possib
le only
 |         for "unsafe" casting.  Casting to multiple fields is allowed, but
 |         casting from multiple fields is not.
 |
 |      .. versionchanged:: 1.9.0
 |         Casting from numeric to string types in 'safe' casting mode requi
res
 |         that the string dtype length is long enough to store the max
 |         integer/float value converted.
 |
 |      Raises
 |      ------
 |      ComplexWarning
 |          When casting from complex to float or int. To avoid this,
 |          one should use ``a.real.astype(t)``.
 |
 |      Examples
 |      --------
 |      >>> x = np.array([1, 2, 2.5])
 |      >>> x
 |      array([1. ,  2. ,  2.5])
 |
 |      >>> x.astype(int)
 |      array([1, 2, 2])
 |
 |  byteswap(...)
 |      a.byteswap(inplace=False)
 |
 |      Swap the bytes of the array elements
 |
 |      Toggle between low-endian and big-endian data representation by
 |      returning a byteswapped array, optionally swapped in-place.
 |      Arrays of byte-strings are not swapped. The real and imaginary
 |      parts of a complex number are swapped individually.
 |
 |      Parameters
 |      ----------
 |      inplace : bool, optional
```

Loading [MathJax]/extensions/Safe.js

```
 |          If ``True``, swap bytes in-place, default is ``False``.
 |
 |      Returns
 |      -------
 |      out : ndarray
 |          The byteswapped array. If `inplace` is ``True``, this is
 |          a view to self.
 |
 |      Examples
 |      --------
 |      >>> A = np.array([1, 256, 8755], dtype=np.int16)
 |      >>> list(map(hex, A))
 |      ['0x1', '0x100', '0x2233']
 |      >>> A.byteswap(inplace=True)
 |      array([  256,     1, 13090], dtype=int16)
 |      >>> list(map(hex, A))
 |      ['0x100', '0x1', '0x3322']
 |
 |      Arrays of byte-strings are not swapped
 |
 |      >>> A = np.array([b'ceg', b'fac'])
 |      >>> A.byteswap()
 |      array([b'ceg', b'fac'], dtype='|S3')
 |
 |      ``A.newbyteorder().byteswap()`` produces an array with the same valu
es
 |        but different representation in memory
 |
 |      >>> A = np.array([1, 2, 3])
 |      >>> A.view(np.uint8)
 |      array([1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0,
0, 0,
 |             0, 0], dtype=uint8)
 |      >>> A.newbyteorder().byteswap(inplace=True)
 |      array([1, 2, 3])
 |      >>> A.view(np.uint8)
 |      array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0,
0, 0,
 |             0, 3], dtype=uint8)
 |
 |  choose(...)
 |      a.choose(choices, out=None, mode='raise')
 |
 |      Use an index array to construct a new array from a set of choices.
 |
 |      Refer to `numpy.choose` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.choose : equivalent function
 |
 |  clip(...)
 |      a.clip(min=None, max=None, out=None, **kwargs)
 |
 |      Return an array whose values are limited to ``[min, max]``.
 |      One of max or min must be given.
```

Loading [MathJax]/extensions/Safe.js

```
 |
 |      Refer to `numpy.clip` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.clip : equivalent function
 |
 |  compress(...)
 |      a.compress(condition, axis=None, out=None)
 |
 |      Return selected slices of this array along given axis.
 |
 |      Refer to `numpy.compress` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.compress : equivalent function
 |
 |  conj(...)
 |      a.conj()
 |
 |      Complex-conjugate all elements.
 |
 |      Refer to `numpy.conjugate` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.conjugate : equivalent function
 |
 |  conjugate(...)
 |      a.conjugate()
 |
 |      Return the complex conjugate, element-wise.
 |
 |      Refer to `numpy.conjugate` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.conjugate : equivalent function
 |
 |  copy(...)
 |      a.copy(order='C')
 |
 |      Return a copy of the array.
 |
 |      Parameters
 |      ----------
 |      order : {'C', 'F', 'A', 'K'}, optional
 |          Controls the memory layout of the copy. 'C' means C-order,
 |          'F' means F-order, 'A' means 'F' if `a` is Fortran contiguous,
 |          'C' otherwise. 'K' means match the layout of `a` as closely
 |          as possible. (Note that this function and :func:`numpy.copy` are
very
 |          similar but have different default values for their order=
 |          arguments, and this function always passes sub-classes through.)
 |
```

Loading [MathJax]/extensions/Safe.js

```
 |      See also
 |      --------
 |      numpy.copy : Similar function with different default behavior
 |      numpy.copyto
 |
 |      Notes
 |      -----
 |      This function is the preferred method for creating an array copy.  T
he
 |      function :func:`numpy.copy` is similar, but it defaults to using ord
er 'K',
 |      and will not pass sub-classes through by default.
 |
 |      Examples
 |      --------
 |      >>> x = np.array([[1,2,3],[4,5,6]], order='F')
 |
 |      >>> y = x.copy()
 |
 |      >>> x.fill(0)
 |
 |      >>> x
 |      array([[0, 0, 0],
 |             [0, 0, 0]])
 |
 |      >>> y
 |      array([[1, 2, 3],
 |             [4, 5, 6]])
 |
 |      >>> y.flags['C_CONTIGUOUS']
 |      True
 |
 |  cumprod(...)
 |      a.cumprod(axis=None, dtype=None, out=None)
 |
 |      Return the cumulative product of the elements along the given axis.
 |
 |      Refer to `numpy.cumprod` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.cumprod : equivalent function
 |
 |  cumsum(...)
 |      a.cumsum(axis=None, dtype=None, out=None)
 |
 |      Return the cumulative sum of the elements along the given axis.
 |
 |      Refer to `numpy.cumsum` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.cumsum : equivalent function
 |
 |  diagonal(...)
 |      a.diagonal(offset=0, axis1=0, axis2=1)
```

Loading [MathJax]/extensions/Safe.js

```
 |
 |      Return specified diagonals. In NumPy 1.9 the returned array is a
 |      read-only view instead of a copy as in previous NumPy versions.  In
 |      a future version the read-only restriction will be removed.
 |
 |      Refer to :func:`numpy.diagonal` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.diagonal : equivalent function
 |
 |  dot(...)
 |
 |  dump(...)
 |      a.dump(file)
 |
 |      Dump a pickle of the array to the specified file.
 |      The array can be read back with pickle.load or numpy.load.
 |
 |      Parameters
 |      ----------
 |      file : str or Path
 |          A string naming the dump file.
 |
 |          .. versionchanged:: 1.17.0
 |              `pathlib.Path` objects are now accepted.
 |
 |  dumps(...)
 |      a.dumps()
 |
 |      Returns the pickle of the array as a string.
 |      pickle.loads will convert the string back to an array.
 |
 |      Parameters
 |      ----------
 |      None
 |
 |  fill(...)
 |      a.fill(value)
 |
 |      Fill the array with a scalar value.
 |
 |      Parameters
 |      ----------
 |      value : scalar
 |          All elements of `a` will be assigned this value.
 |
 |      Examples
 |      --------
 |      >>> a = np.array([1, 2])
 |      >>> a.fill(0)
 |      >>> a
 |      array([0, 0])
 |      >>> a = np.empty(2)
 |      >>> a.fill(1)
 |      >>> a
```

Loading [MathJax]/extensions/Safe.js

```
 |      array([1.,  1.])
 |
 |      Fill expects a scalar value and always behaves the same as assigning
 |      to a single array element.  The following is a rare example where th
is
 |      distinction is important:
 |
 |      >>> a = np.array([None, None], dtype=object)
 |      >>> a[0] = np.array(3)
 |      >>> a
 |      array([array(3), None], dtype=object)
 |      >>> a.fill(np.array(3))
 |      >>> a
 |      array([array(3), array(3)], dtype=object)
 |
 |      Where other forms of assignments will unpack the array being assigne
d:
 |
 |      >>> a[...] = np.array(3)
 |      >>> a
 |      array([3, 3], dtype=object)
 |
 |  flatten(...)
 |      a.flatten(order='C')
 |
 |      Return a copy of the array collapsed into one dimension.
 |
 |      Parameters
 |      ----------
 |      order : {'C', 'F', 'A', 'K'}, optional
 |          'C' means to flatten in row-major (C-style) order.
 |          'F' means to flatten in column-major (Fortran-
 |          style) order. 'A' means to flatten in column-major
 |          order if `a` is Fortran *contiguous* in memory,
 |          row-major order otherwise. 'K' means to flatten
 |          `a` in the order the elements occur in memory.
 |          The default is 'C'.
 |
 |      Returns
 |      -------
 |      y : ndarray
 |          A copy of the input array, flattened to one dimension.
 |
 |      See Also
 |      --------
 |      ravel : Return a flattened array.
 |      flat : A 1-D flat iterator over the array.
 |
 |      Examples
 |      --------
 |      >>> a = np.array([[1,2], [3,4]])
 |      >>> a.flatten()
 |      array([1, 2, 3, 4])
 |      >>> a.flatten('F')
 |      array([1, 3, 2, 4])
 |
```

Loading [MathJax]/extensions/Safe.js

```
 |  getfield(...)
 |      a.getfield(dtype, offset=0)
 |
 |      Returns a field of the given array as a certain type.
 |
 |      A field is a view of the array data with a given data-type. The valu
es in
 |      the view are determined by the given type and the offset into the cu
rrent
 |      array in bytes. The offset needs to be such that the view dtype fits
in the
 |      array dtype; for example an array of dtype complex128 has 16-byte el
ements.
 |      If taking a view with a 32-bit integer (4 bytes), the offset needs t
o be
 |      between 0 and 12 bytes.
 |
 |      Parameters
 |      ----------
 |      dtype : str or dtype
 |          The data type of the view. The dtype size of the view can not be
larger
 |          than that of the array itself.
 |      offset : int
 |          Number of bytes to skip before beginning the element view.
 |
 |      Examples
 |      --------
 |      >>> x = np.diag([1.+1.j]*2)
 |      >>> x[1, 1] = 2 + 4.j
 |      >>> x
 |      array([[1.+1.j,  0.+0.j],
 |             [0.+0.j,  2.+4.j]])
 |      >>> x.getfield(np.float64)
 |      array([[1.,  0.],
 |             [0.,  2.]])
 |
 |      By choosing an offset of 8 bytes we can select the complex part of t
he
 |      array for our view:
 |
 |      >>> x.getfield(np.float64, offset=8)
 |      array([[1.,  0.],
 |             [0.,  4.]])
 |
 |  item(...)
 |      a.item(*args)
 |
 |      Copy an element of an array to a standard Python scalar and return i
t.
 |
 |      Parameters
 |      ----------
 |      \*args : Arguments (variable number and type)
 |
 |          * none: in this case, the method only works for arrays
```

Loading [MathJax]/extensions/Safe.js

```
 |              with one element (`a.size == 1`), which element is
 |              copied into a standard Python scalar object and returned.
 |
 |            * int_type: this argument is interpreted as a flat index into
 |              the array, specifying which element to copy and return.
 |
 |            * tuple of int_types: functions as does a single int_type argume
nt,
 |              except that the argument is interpreted as an nd-index into th
e
 |              array.
 |
 |      Returns
 |      -------
 |      z : Standard Python scalar object
 |          A copy of the specified element of the array as a suitable
 |          Python scalar
 |
 |      Notes
 |      -----
 |      When the data type of `a` is longdouble or clongdouble, item() retur
ns
 |      a scalar array object because there is no available Python scalar th
at
 |      would not lose information. Void arrays return a buffer object for i
tem(),
 |      unless fields are defined, in which case a tuple is returned.
 |
 |      `item` is very similar to a[args], except, instead of an array scala
r,
 |      a standard Python scalar is returned. This can be useful for speedin
g up
 |      access to elements of the array and doing arithmetic on elements of
the
 |      array using Python's optimized math.
 |
 |      Examples
 |      --------
 |      >>> np.random.seed(123)
 |      >>> x = np.random.randint(9, size=(3, 3))
 |      >>> x
 |      array([[2, 2, 6],
 |             [1, 3, 6],
 |             [1, 0, 1]])
 |      >>> x.item(3)
 |      1
 |      >>> x.item(7)
 |      0
 |      >>> x.item((0, 1))
 |      2
 |      >>> x.item((2, 2))
 |      1
 |
 |  itemset(...)
 |      a.itemset(*args)
 |
```

Loading [MathJax]/extensions/Safe.js

```
 |      Insert scalar into an array (scalar is cast to array's dtype, if pos
sible)
 |      
 |      There must be at least 1 argument, and define the last argument
 |      as *item*.  Then, ``a.itemset(*args)`` is equivalent to but faster
 |      than ``a[args] = item``.  The item should be a scalar value and `arg
s`
 |      must select a single item in the array `a`.
 |      
 |      Parameters
 |      ----------
 |      \*args : Arguments
 |          If one argument: a scalar, only used in case `a` is of size 1.
 |          If two arguments: the last argument is the value to be set
 |          and must be a scalar, the first argument specifies a single arra
y
 |          element location. It is either an int or a tuple.
 |      
 |      Notes
 |      -----
 |      Compared to indexing syntax, `itemset` provides some speed increase
 |      for placing a scalar into a particular location in an `ndarray`,
 |      if you must do this.  However, generally this is discouraged:
 |      among other problems, it complicates the appearance of the code.
 |      Also, when using `itemset` (and `item`) inside a loop, be sure
 |      to assign the methods to a local variable to avoid the attribute
 |      look-up at each loop iteration.
 |      
 |      Examples
 |      --------
 |      >>> np.random.seed(123)
 |      >>> x = np.random.randint(9, size=(3, 3))
 |      >>> x
 |      array([[2, 2, 6],
 |             [1, 3, 6],
 |             [1, 0, 1]])
 |      >>> x.itemset(4, 0)
 |      >>> x.itemset((2, 2), 9)
 |      >>> x
 |      array([[2, 2, 6],
 |             [1, 0, 6],
 |             [1, 0, 9]])
 |  
 |  max(...)
 |      a.max(axis=None, out=None, keepdims=False, initial=<no value>, where
=True)
 |      
 |      Return the maximum along a given axis.
 |      
 |      Refer to `numpy.amax` for full documentation.
 |      
 |      See Also
 |      --------
 |      numpy.amax : equivalent function
 |  
 |  mean(...)
```

```
 |      a.mean(axis=None, dtype=None, out=None, keepdims=False, *, where=Tru
e)
 |
 |      Returns the average of the array elements along given axis.
 |
 |      Refer to `numpy.mean` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.mean : equivalent function
 |
 |  min(...)
 |      a.min(axis=None, out=None, keepdims=False, initial=<no value>, where
=True)
 |
 |      Return the minimum along a given axis.
 |
 |      Refer to `numpy.amin` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.amin : equivalent function
 |
 |  newbyteorder(...)
 |      arr.newbyteorder(new_order='S', /)
 |
 |      Return the array with the same data viewed with a different byte ord
er.
 |
 |      Equivalent to::
 |
 |          arr.view(arr.dtype.newbytorder(new_order))
 |
 |      Changes are also made in all fields and sub-arrays of the array data
 |      type.
 |
 |
 |
 |      Parameters
 |      ----------
 |      new_order : string, optional
 |          Byte order to force; a value from the byte order specifications
 |          below. `new_order` codes can be any of:
 |
 |          * 'S' - swap dtype from current to opposite endian
 |          * {'<', 'little'} - little endian
 |          * {'>', 'big'} - big endian
 |          * {'=', 'native'} - native order, equivalent to `sys.byteorder`
 |          * {'|', 'I'} - ignore (no change to byte order)
 |
 |          The default value ('S') results in swapping the current
 |          byte order.
 |
 |
 |      Returns
 |      -------
```

Loading [MathJax]/extensions/Safe.js

```
 |      new_arr : array
 |          New array object with the dtype reflecting given change to the
 |          byte order.
 |
 |  nonzero(...)
 |      a.nonzero()
 |
 |      Return the indices of the elements that are non-zero.
 |
 |      Refer to `numpy.nonzero` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.nonzero : equivalent function
 |
 |  partition(...)
 |      a.partition(kth, axis=-1, kind='introselect', order=None)
 |
 |      Rearranges the elements in the array in such a way that the value of
the
 |      element in kth position is in the position it would be in a sorted a
rray.
 |      All elements smaller than the kth element are moved before this elem
ent and
 |      all equal or greater are moved behind it. The ordering of the elemen
ts in
 |      the two partitions is undefined.
 |
 |      .. versionadded:: 1.8.0
 |
 |      Parameters
 |      ----------
 |      kth : int or sequence of ints
 |          Element index to partition by. The kth element value will be in
its
 |          final sorted position and all smaller elements will be moved bef
ore it
 |          and all equal or greater elements behind it.
 |          The order of all elements in the partitions is undefined.
 |          If provided with a sequence of kth it will partition all element
s
 |          indexed by kth of them into their sorted position at once.
 |
 |          .. deprecated:: 1.22.0
 |              Passing booleans as index is deprecated.
 |      axis : int, optional
 |          Axis along which to sort. Default is -1, which means sort along
the
 |          last axis.
 |      kind : {'introselect'}, optional
 |          Selection algorithm. Default is 'introselect'.
 |      order : str or list of str, optional
 |          When `a` is an array with fields defined, this argument specifie
s
 |          which fields to compare first, second, etc. A single field can
 |          be specified as a string, and not all fields need to be specifie
```

Loading [MathJax]/extensions/Safe.js

```
d,
 |          but unspecified fields will still be used, in the order in which
 |          they come up in the dtype, to break ties.
 |
 |      See Also
 |      --------
 |      numpy.partition : Return a partitioned copy of an array.
 |      argpartition : Indirect partition.
 |      sort : Full sort.
 |
 |      Notes
 |      -----
 |      See ``np.partition`` for notes on the different algorithms.
 |
 |      Examples
 |      --------
 |      >>> a = np.array([3, 4, 2, 1])
 |      >>> a.partition(3)
 |      >>> a
 |      array([2, 1, 3, 4])
 |
 |      >>> a.partition((1, 3))
 |      >>> a
 |      array([1, 2, 3, 4])
 |
 |  prod(...)
 |      a.prod(axis=None, dtype=None, out=None, keepdims=False, initial=1, w
here=True)
 |
 |      Return the product of the array elements over the given axis
 |
 |      Refer to `numpy.prod` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.prod : equivalent function
 |
 |  ptp(...)
 |      a.ptp(axis=None, out=None, keepdims=False)
 |
 |      Peak to peak (maximum - minimum) value along a given axis.
 |
 |      Refer to `numpy.ptp` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.ptp : equivalent function
 |
 |  put(...)
 |      a.put(indices, values, mode='raise')
 |
 |      Set ``a.flat[n] = values[n]`` for all `n` in indices.
 |
 |      Refer to `numpy.put` for full documentation.
 |
 |      See Also
```

Loading [MathJax]/extensions/Safe.js

```
 |      --------
 |      numpy.put : equivalent function
 |
 |  ravel(...)
 |      a.ravel([order])
 |
 |      Return a flattened array.
 |
 |      Refer to `numpy.ravel` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.ravel : equivalent function
 |
 |      ndarray.flat : a flat iterator on the array.
 |
 |  repeat(...)
 |      a.repeat(repeats, axis=None)
 |
 |      Repeat elements of an array.
 |
 |      Refer to `numpy.repeat` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.repeat : equivalent function
 |
 |  reshape(...)
 |      a.reshape(shape, order='C')
 |
 |      Returns an array containing the same data with a new shape.
 |
 |      Refer to `numpy.reshape` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.reshape : equivalent function
 |
 |      Notes
 |      -----
 |      Unlike the free function `numpy.reshape`, this method on `ndarray` a
llows
 |      the elements of the shape parameter to be passed in as separate argu
ments.
 |      For example, ``a.reshape(10, 11)`` is equivalent to
 |      ``a.reshape((10, 11))``.
 |
 |  resize(...)
 |      a.resize(new_shape, refcheck=True)
 |
 |      Change shape and size of array in-place.
 |
 |      Parameters
 |      ----------
 |      new_shape : tuple of ints, or `n` ints
 |          Shape of resized array.
```

```
 |      refcheck : bool, optional
 |          If False, reference count will not be checked. Default is True.
 |
 |      Returns
 |      -------
 |      None
 |
 |      Raises
 |      ------
 |      ValueError
 |          If `a` does not own its own data or references or views to it ex
ist,
 |          and the data memory must be changed.
 |          PyPy only: will always raise if the data memory must be changed,
since
 |          there is no reliable way to determine if references or views to
it
 |          exist.
 |
 |      SystemError
 |          If the `order` keyword argument is specified. This behaviour is
a
 |          bug in NumPy.
 |
 |      See Also
 |      --------
 |      resize : Return a new array with the specified shape.
 |
 |      Notes
 |      -----
 |      This reallocates space for the data area if necessary.
 |
 |      Only contiguous arrays (data elements consecutive in memory) can be
 |      resized.
 |
 |      The purpose of the reference count check is to make sure you
 |      do not use this array as a buffer for another Python object and then
 |      reallocate the memory. However, reference counts can increase in
 |      other ways so if you are sure that you have not shared the memory
 |      for this array with another Python object, then you may safely set
 |      `refcheck` to False.
 |
 |      Examples
 |      --------
 |      Shrinking an array: array is flattened (in the order that the data a
re
 |      stored in memory), resized, and reshaped:
 |
 |      >>> a = np.array([[0, 1], [2, 3]], order='C')
 |      >>> a.resize((2, 1))
 |      >>> a
 |      array([[0],
 |             [1]])
 |
 |      >>> a = np.array([[0, 1], [2, 3]], order='F')
 |      >>> a.resize((2, 1))
```

Loading [MathJax]/extensions/Safe.js

```
 |      >>> a
 |      array([[0],
 |             [2]])
 |
 |      Enlarging an array: as above, but missing entries are filled with ze
ros:
 |
 |      >>> b = np.array([[0, 1], [2, 3]])
 |      >>> b.resize(2, 3) # new_shape parameter doesn't have to be a tuple
 |      >>> b
 |      array([[0, 1, 2],
 |             [3, 0, 0]])
 |
 |      Referencing an array prevents resizing...
 |
 |      >>> c = a
 |      >>> a.resize((1, 1))
 |      Traceback (most recent call last):
 |      ...
 |      ValueError: cannot resize an array that references or is referenced
...
 |
 |      Unless `refcheck` is False:
 |
 |      >>> a.resize((1, 1), refcheck=False)
 |      >>> a
 |      array([[0]])
 |      >>> c
 |      array([[0]])
 |
 |  round(...)
 |      a.round(decimals=0, out=None)
 |
 |      Return `a` with each element rounded to the given number of decimal
s.
 |
 |      Refer to `numpy.around` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.around : equivalent function
 |
 |  searchsorted(...)
 |      a.searchsorted(v, side='left', sorter=None)
 |
 |      Find indices where elements of v should be inserted in a to maintain
order.
 |
 |      For full documentation, see `numpy.searchsorted`
 |
 |      See Also
 |      --------
 |      numpy.searchsorted : equivalent function
 |
 |  setfield(...)
 |      a.setfield(val, dtype, offset=0)
```

```
 |
 |      Put a value into a specified place in a field defined by a data-typ
e.
 |
 |      Place `val` into `a`'s field defined by `dtype` and beginning `offse
t`
 |      bytes into the field.
 |
 |      Parameters
 |      ----------
 |      val : object
 |          Value to be placed in field.
 |      dtype : dtype object
 |          Data-type of the field in which to place `val`.
 |      offset : int, optional
 |          The number of bytes into the field at which to place `val`.
 |
 |      Returns
 |      -------
 |      None
 |
 |      See Also
 |      --------
 |      getfield
 |
 |      Examples
 |      --------
 |      >>> x = np.eye(3)
 |      >>> x.getfield(np.float64)
 |      array([[1.,  0.,  0.],
 |             [0.,  1.,  0.],
 |             [0.,  0.,  1.]])
 |      >>> x.setfield(3, np.int32)
 |      >>> x.getfield(np.int32)
 |      array([[3, 3, 3],
 |             [3, 3, 3],
 |             [3, 3, 3]], dtype=int32)
 |      >>> x
 |      array([[1.0e+000, 1.5e-323, 1.5e-323],
 |             [1.5e-323, 1.0e+000, 1.5e-323],
 |             [1.5e-323, 1.5e-323, 1.0e+000]])
 |      >>> x.setfield(np.eye(3), np.int32)
 |      >>> x
 |      array([[1.,  0.,  0.],
 |             [0.,  1.,  0.],
 |             [0.,  0.,  1.]])
 |
 |  setflags(...)
 |      a.setflags(write=None, align=None, uic=None)
 |
 |      Set array flags WRITEABLE, ALIGNED, WRITEBACKIFCOPY,
 |      respectively.
 |
 |      These Boolean-valued flags affect how numpy interprets the memory
 |      area used by `a` (see Notes below). The ALIGNED flag can only
 |      be set to True if the data is actually aligned according to the typ
```

```
e.
 |      The WRITEBACKIFCOPY and flag can never be set
 |      to True. The flag WRITEABLE can only be set to True if the array own
s its
 |      own memory, or the ultimate owner of the memory exposes a writeable
buffer
 |      interface, or is a string. (The exception for string is made so that
 |      unpickling can be done without copying memory.)
 |
 |      Parameters
 |      ----------
 |      write : bool, optional
 |          Describes whether or not `a` can be written to.
 |      align : bool, optional
 |          Describes whether or not `a` is aligned properly for its type.
 |      uic : bool, optional
 |          Describes whether or not `a` is a copy of another "base" array.
 |
 |      Notes
 |      -----
 |      Array flags provide information about how the memory area used
 |      for the array is to be interpreted. There are 7 Boolean flags
 |      in use, only four of which can be changed by the user:
 |      WRITEBACKIFCOPY, WRITEABLE, and ALIGNED.
 |
 |      WRITEABLE (W) the data area can be written to;
 |
 |      ALIGNED (A) the data and strides are aligned appropriately for the h
ardware
 |      (as determined by the compiler);
 |
 |      WRITEBACKIFCOPY (X) this array is a copy of some other array (refere
nced
 |      by .base). When the C-API function PyArray_ResolveWritebackIfCopy is
 |      called, the base array will be updated with the contents of this arr
ay.
 |
 |      All flags can be accessed using the single (upper case) letter as we
ll
 |      as the full name.
 |
 |      Examples
 |      --------
 |      >>> y = np.array([[3, 1, 7],
 |      ...                [2, 0, 0],
 |      ...                [8, 5, 9]])
 |      >>> y
 |      array([[3, 1, 7],
 |             [2, 0, 0],
 |             [8, 5, 9]])
 |      >>> y.flags
 |        C_CONTIGUOUS : True
 |        F_CONTIGUOUS : False
 |        OWNDATA : True
 |        WRITEABLE : True
 |        ALIGNED : True
```

```
 |          WRITEBACKIFCOPY : False
 |      >>> y.setflags(write=0, align=0)
 |      >>> y.flags
 |          C_CONTIGUOUS : True
 |          F_CONTIGUOUS : False
 |          OWNDATA : True
 |          WRITEABLE : False
 |          ALIGNED : False
 |          WRITEBACKIFCOPY : False
 |      >>> y.setflags(uic=1)
 |      Traceback (most recent call last):
 |        File "<stdin>", line 1, in <module>
 |      ValueError: cannot set WRITEBACKIFCOPY flag to True
 |
 |  sort(...)
 |      a.sort(axis=-1, kind=None, order=None)
 |
 |      Sort an array in-place. Refer to `numpy.sort` for full documentatio
n.
 |
 |      Parameters
 |      ----------
 |      axis : int, optional
 |          Axis along which to sort. Default is -1, which means sort along
the
 |          last axis.
 |      kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, optional
 |          Sorting algorithm. The default is 'quicksort'. Note that both 's
table'
 |          and 'mergesort' use timsort under the covers and, in general, th
e
 |          actual implementation will vary with datatype. The 'mergesort' o
ption
 |          is retained for backwards compatibility.
 |
 |          .. versionchanged:: 1.15.0
 |             The 'stable' option was added.
 |
 |      order : str or list of str, optional
 |          When `a` is an array with fields defined, this argument specifie
s
 |          which fields to compare first, second, etc.  A single field can
 |          be specified as a string, and not all fields need be specified,
 |          but unspecified fields will still be used, in the order in which
 |          they come up in the dtype, to break ties.
 |
 |      See Also
 |      --------
 |      numpy.sort : Return a sorted copy of an array.
 |      numpy.argsort : Indirect sort.
 |      numpy.lexsort : Indirect stable sort on multiple keys.
 |      numpy.searchsorted : Find elements in sorted array.
 |      numpy.partition: Partial sort.
 |
 |      Notes
 |      -----
```

Loading [MathJax]/extensions/Safe.js

```
 |      See `numpy.sort` for notes on the different sorting algorithms.
 |
 |      Examples
 |      --------
 |      >>> a = np.array([[1,4], [3,1]])
 |      >>> a.sort(axis=1)
 |      >>> a
 |      array([[1, 4],
 |             [1, 3]])
 |      >>> a.sort(axis=0)
 |      >>> a
 |      array([[1, 3],
 |             [1, 4]])
 |
 |      Use the `order` keyword to specify a field to use when sorting a
 |      structured array:
 |
 |      >>> a = np.array([('a', 2), ('c', 1)], dtype=[('x', 'S1'), ('y', in
t)])
 |      >>> a.sort(order='y')
 |      >>> a
 |      array([(b'c', 1), (b'a', 2)],
 |            dtype=[('x', 'S1'), ('y', '<i8')])
 |
 |  squeeze(...)
 |      a.squeeze(axis=None)
 |
 |      Remove axes of length one from `a`.
 |
 |      Refer to `numpy.squeeze` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.squeeze : equivalent function
 |
 |  std(...)
 |      a.std(axis=None, dtype=None, out=None, ddof=0, keepdims=False, *, wh
ere=True)
 |
 |      Returns the standard deviation of the array elements along given axi
s.
 |
 |      Refer to `numpy.std` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.std : equivalent function
 |
 |  sum(...)
 |      a.sum(axis=None, dtype=None, out=None, keepdims=False, initial=0, wh
ere=True)
 |
 |      Return the sum of the array elements over the given axis.
 |
 |      Refer to `numpy.sum` for full documentation.
 |
```

Loading [MathJax]/extensions/Safe.js

```
 |      See Also
 |      --------
 |      numpy.sum : equivalent function
 |
 |  swapaxes(...)
 |      a.swapaxes(axis1, axis2)
 |
 |      Return a view of the array with `axis1` and `axis2` interchanged.
 |
 |      Refer to `numpy.swapaxes` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.swapaxes : equivalent function
 |
 |  take(...)
 |      a.take(indices, axis=None, out=None, mode='raise')
 |
 |      Return an array formed from the elements of `a` at the given indice
s.
 |
 |      Refer to `numpy.take` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.take : equivalent function
 |
 |  tobytes(...)
 |      a.tobytes(order='C')
 |
 |      Construct Python bytes containing the raw data bytes in the array.
 |
 |      Constructs Python bytes showing a copy of the raw contents of
 |      data memory. The bytes object is produced in C-order by default.
 |      This behavior is controlled by the ``order`` parameter.
 |
 |      .. versionadded:: 1.9.0
 |
 |      Parameters
 |      ----------
 |      order : {'C', 'F', 'A'}, optional
 |          Controls the memory layout of the bytes object. 'C' means C-orde
r,
 |          'F' means F-order, 'A' (short for *Any*) means 'F' if `a` is
 |          Fortran contiguous, 'C' otherwise. Default is 'C'.
 |
 |      Returns
 |      -------
 |      s : bytes
 |          Python bytes exhibiting a copy of `a`'s raw data.
 |
 |      See also
 |      --------
 |      frombuffer
 |          Inverse of this operation, construct a 1-dimensional array from
Python
```

Loading [MathJax]/extensions/Safe.js

```
 |          bytes.
 |
 |      Examples
 |      --------
 |      >>> x = np.array([[0, 1], [2, 3]], dtype='<u2')
 |      >>> x.tobytes()
 |      b'\x00\x00\x01\x00\x02\x00\x03\x00'
 |      >>> x.tobytes('C') == x.tobytes()
 |      True
 |      >>> x.tobytes('F')
 |      b'\x00\x00\x02\x00\x01\x00\x03\x00'
 |
 |  tofile(...)
 |      a.tofile(fid, sep="", format="%s")
 |
 |      Write array to a file as text or binary (default).
 |
 |      Data is always written in 'C' order, independent of the order of `a
`.
 |      The data produced by this method can be recovered using the function
 |      fromfile().
 |
 |      Parameters
 |      ----------
 |      fid : file or str or Path
 |          An open file object, or a string containing a filename.
 |
 |          .. versionchanged:: 1.17.0
 |              `pathlib.Path` objects are now accepted.
 |
 |      sep : str
 |          Separator between array items for text output.
 |          If "" (empty), a binary file is written, equivalent to
 |          ``file.write(a.tobytes())``.
 |      format : str
 |          Format string for text file output.
 |          Each entry in the array is formatted to text by first converting
 |          it to the closest Python type, and then using "format" % item.
 |
 |      Notes
 |      -----
 |      This is a convenience function for quick storage of array data.
 |      Information on endianness and precision is lost, so this method is n
ot a
 |      good choice for files intended to archive data or transport data bet
ween
 |      machines with different endianness. Some of these problems can be ov
ercome
 |      by outputting the data as text files, at the expense of speed and fi
le
 |      size.
 |
 |      When fid is a file object, array contents are directly written to th
e
 |      file, bypassing the file object's ``write`` method. As a result, tof
ile
```

Loading [MathJax]/extensions/Safe.js

```
 |      cannot be used with files objects supporting compression (e.g., Gzip
File)
 |      or file-like objects that do not support ``fileno()`` (e.g., BytesI
O).
 |
 |  tolist(...)
 |      a.tolist()
 |
 |      Return the array as an ``a.ndim``-levels deep nested list of Python
scalars.
 |
 |      Return a copy of the array data as a (nested) Python list.
 |      Data items are converted to the nearest compatible builtin Python ty
pe, via
 |      the `~numpy.ndarray.item` function.
 |
 |      If ``a.ndim`` is 0, then since the depth of the nested list is 0, it
will
 |      not be a list at all, but a simple Python scalar.
 |
 |      Parameters
 |      ----------
 |      none
 |
 |      Returns
 |      -------
 |      y : object, or list of object, or list of list of object, or ...
 |          The possibly nested list of array elements.
 |
 |      Notes
 |      -----
 |      The array may be recreated via ``a = np.array(a.tolist())``, althoug
h this
 |      may sometimes lose precision.
 |
 |      Examples
 |      --------
 |      For a 1D array, ``a.tolist()`` is almost the same as ``list(a)``,
 |      except that ``tolist`` changes numpy scalars to Python scalars:
 |
 |      >>> a = np.uint32([1, 2])
 |      >>> a_list = list(a)
 |      >>> a_list
 |      [1, 2]
 |      >>> type(a_list[0])
 |      <class 'numpy.uint32'>
 |      >>> a_tolist = a.tolist()
 |      >>> a_tolist
 |      [1, 2]
 |      >>> type(a_tolist[0])
 |      <class 'int'>
 |
 |      Additionally, for a 2D array, ``tolist`` applies recursively:
 |
 |      >>> a = np.array([[1, 2], [3, 4]])
 |      >>> list(a)
```

Loading [MathJax]/extensions/Safe.js

```
 |      [array([1, 2]), array([3, 4])]
 |      >>> a.tolist()
 |      [[1, 2], [3, 4]]
 |
 |      The base case for this recursion is a 0D array:
 |
 |      >>> a = np.array(1)
 |      >>> list(a)
 |      Traceback (most recent call last):
 |        ...
 |      TypeError: iteration over a 0-d array
 |      >>> a.tolist()
 |      1
 |
 |  tostring(...)
 |      a.tostring(order='C')
 |
 |      A compatibility alias for `tobytes`, with exactly the same behavior.
 |
 |      Despite its name, it returns `bytes` not `str`\ s.
 |
 |      .. deprecated:: 1.19.0
 |
 |  trace(...)
 |      a.trace(offset=0, axis1=0, axis2=1, dtype=None, out=None)
 |
 |      Return the sum along diagonals of the array.
 |
 |      Refer to `numpy.trace` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.trace : equivalent function
 |
 |  transpose(...)
 |      a.transpose(*axes)
 |
 |      Returns a view of the array with axes transposed.
 |
 |      Refer to `numpy.transpose` for full documentation.
 |
 |      Parameters
 |      ----------
 |      axes : None, tuple of ints, or `n` ints
 |
 |       * None or no argument: reverses the order of the axes.
 |
 |       * tuple of ints: `i` in the `j`-th place in the tuple means that th
e
 |         array's `i`-th axis becomes the transposed array's `j`-th axis.
 |
 |       * `n` ints: same as an n-tuple of the same ints (this form is
 |         intended simply as a "convenience" alternative to the tuple for
m).
 |
 |      Returns
```

Loading [MathJax]/extensions/Safe.js

```
 |      -------
 |      p : ndarray
 |          View of the array with its axes suitably permuted.
 |
 |      See Also
 |      --------
 |      transpose : Equivalent function.
 |      ndarray.T : Array property returning the array transposed.
 |      ndarray.reshape : Give a new shape to an array without changing its
data.
 |
 |      Examples
 |      --------
 |      >>> a = np.array([[1, 2], [3, 4]])
 |      >>> a
 |      array([[1, 2],
 |             [3, 4]])
 |      >>> a.transpose()
 |      array([[1, 3],
 |             [2, 4]])
 |      >>> a.transpose((1, 0))
 |      array([[1, 3],
 |             [2, 4]])
 |      >>> a.transpose(1, 0)
 |      array([[1, 3],
 |             [2, 4]])
 |
 |      >>> a = np.array([1, 2, 3, 4])
 |      >>> a
 |      array([1, 2, 3, 4])
 |      >>> a.transpose()
 |      array([1, 2, 3, 4])
 |
 |  var(...)
 |      a.var(axis=None, dtype=None, out=None, ddof=0, keepdims=False, *, wh
ere=True)
 |
 |      Returns the variance of the array elements, along given axis.
 |
 |      Refer to `numpy.var` for full documentation.
 |
 |      See Also
 |      --------
 |      numpy.var : equivalent function
 |
 |  view(...)
 |      a.view([dtype][, type])
 |
 |      New view of array with the same data.
 |
 |      .. note::
 |          Passing None for ``dtype`` is different from omitting the parame
ter,
 |          since the former invokes ``dtype(None)`` which is an alias for
 |          ``dtype('float_')``.
 |
```

```
 |      Parameters
 |      ----------
 |      dtype : data-type or ndarray sub-class, optional
 |          Data-type descriptor of the returned view, e.g., float32 or int1
6.
 |          Omitting it results in the view having the same data-type as `a
`.
 |          This argument can also be specified as an ndarray sub-class, whi
ch
 |          then specifies the type of the returned object (this is equivale
nt to
 |          setting the ``type`` parameter).
 |      type : Python type, optional
 |          Type of the returned view, e.g., ndarray or matrix.  Again, omis
sion
 |          of the parameter results in type preservation.
 |
 |      Notes
 |      -----
 |      ``a.view()`` is used two different ways:
 |
 |      ``a.view(some_dtype)`` or ``a.view(dtype=some_dtype)`` constructs a
view
 |      of the array's memory with a different data-type.  This can cause a
 |      reinterpretation of the bytes of memory.
 |
 |      ``a.view(ndarray_subclass)`` or ``a.view(type=ndarray_subclass)`` ju
st
 |      returns an instance of `ndarray_subclass` that looks at the same arr
ay
 |      (same shape, dtype, etc.)  This does not cause a reinterpretation of
the
 |      memory.
 |
 |      For ``a.view(some_dtype)``, if ``some_dtype`` has a different number
of
 |      bytes per entry than the previous dtype (for example, converting a r
egular
 |      array to a structured array), then the last axis of ``a`` must be
 |      contiguous. This axis will be resized in the result.
 |
 |      .. versionchanged:: 1.23.0
 |         Only the last axis needs to be contiguous. Previously, the entire
array
 |         had to be C-contiguous.
 |
 |      Examples
 |      --------
 |      >>> x = np.array([(1, 2)], dtype=[('a', np.int8), ('b', np.int8)])
 |
 |      Viewing array data using a different type and dtype:
 |
 |      >>> y = x.view(dtype=np.int16, type=np.matrix)
 |      >>> y
 |      matrix([[513]], dtype=int16)
 |      >>> print(type(y))
```

```
 |      <class 'numpy.matrix'>
 |
 |      Creating a view on a structured array so it can be used in calculati
ons
 |
 |      >>> x = np.array([(1, 2),(3,4)], dtype=[('a', np.int8), ('b', np.int
8)])
 |      >>> xv = x.view(dtype=np.int8).reshape(-1,2)
 |      >>> xv
 |      array([[1, 2],
 |             [3, 4]], dtype=int8)
 |      >>> xv.mean(0)
 |      array([2.,  3.])
 |
 |      Making changes to the view changes the underlying array
 |
 |      >>> xv[0,1] = 20
 |      >>> x
 |      array([(1, 20), (3,  4)], dtype=[('a', 'i1'), ('b', 'i1')])
 |
 |      Using a view to convert an array to a recarray:
 |
 |      >>> z = x.view(np.recarray)
 |      >>> z.a
 |      array([1, 3], dtype=int8)
 |
 |      Views share data:
 |
 |      >>> x[0] = (9, 10)
 |      >>> z[0]
 |      (9, 10)
 |
 |      Views that change the dtype size (bytes per entry) should normally b
e
 |      avoided on arrays defined by slices, transposes, fortran-ordering, e
tc.:
 |
 |      >>> x = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.int16)
 |      >>> y = x[:, ::2]
 |      >>> y
 |      array([[1, 3],
 |             [4, 6]], dtype=int16)
 |      >>> y.view(dtype=[('width', np.int16), ('length', np.int16)])
 |      Traceback (most recent call last):
 |          ...
 |      ValueError: To change to a dtype of a different size, the last axis
must be contiguous
 |      >>> z = y.copy()
 |      >>> z.view(dtype=[('width', np.int16), ('length', np.int16)])
 |      array([[(1, 3)],
 |             [(4, 6)]], dtype=[('width', '<i2'), ('length', '<i2')])
 |
 |      However, views that change dtype are totally fine for arrays with a
 |      contiguous last axis, even if the rest of the axes are not C-contigu
ous:
 |
```

Loading [MathJax]/extensions/Safe.js

```
 |      >>> x = np.arange(2 * 3 * 4, dtype=np.int8).reshape(2, 3, 4)
 |      >>> x.transpose(1, 0, 2).view(np.int16)
 |      array([[[ 256,  770],
 |              [3340, 3854]],
 |      <BLANKLINE>
 |             [[1284, 1798],
 |              [4368, 4882]],
 |      <BLANKLINE>
 |             [[2312, 2826],
 |              [5396, 5910]]], dtype=int16)
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __class_getitem__(...) from builtins.type
 |      a.__class_getitem__(item, /)
 |
 |      Return a parametrized wrapper around the `~numpy.ndarray` type.
 |
 |      .. versionadded:: 1.22
 |
 |      Returns
 |      -------
 |      alias : types.GenericAlias
 |          A parametrized `~numpy.ndarray` type.
 |
 |      Examples
 |      --------
 |      >>> from typing import Any
 |      >>> import numpy as np
 |
 |      >>> np.ndarray[Any, np.dtype[Any]]
 |      numpy.ndarray[typing.Any, numpy.dtype[typing.Any]]
 |
 |      See Also
 |      --------
 |      :pep:`585` : Type hinting generics in standard collections.
 |      numpy.typing.NDArray : An ndarray alias :term:`generic <generic type
>`
 |                          w.r.t. its `dtype.type <numpy.dtype.type>`.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signatu
re.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  T
 |      View of the transposed array.
 |
 |      Same as ``self.transpose()``.
 |
```

```
 |      Examples
 |      --------
 |      >>> a = np.array([[1, 2], [3, 4]])
 |      >>> a
 |      array([[1, 2],
 |             [3, 4]])
 |      >>> a.T
 |      array([[1, 3],
 |             [2, 4]])
 |
 |      >>> a = np.array([1, 2, 3, 4])
 |      >>> a
 |      array([1, 2, 3, 4])
 |      >>> a.T
 |      array([1, 2, 3, 4])
 |
 |      See Also
 |      --------
 |      transpose
 |
 |  __array_interface__
 |      Array protocol: Python side.
 |
 |  __array_priority__
 |      Array priority.
 |
 |  __array_struct__
 |      Array protocol: C-struct side.
 |
 |  base
 |      Base object if memory is from some other object.
 |
 |      Examples
 |      --------
 |      The base of an array that owns its memory is None:
 |
 |      >>> x = np.array([1,2,3,4])
 |      >>> x.base is None
 |      True
 |
 |      Slicing creates a view, whose memory is shared with x:
 |
 |      >>> y = x[2:]
 |      >>> y.base is x
 |      True
 |
 |  ctypes
 |      An object to simplify the interaction of the array with the ctypes
 |      module.
 |
 |      This attribute creates an object that makes it easier to use arrays
 |      when calling shared libraries with the ctypes module. The returned
 |      object has, among others, data, shape, and strides attributes (see
 |      Notes below) which themselves return ctypes objects that can be used
 |      as arguments to a shared library.
 |
```

Loading [MathJax]/extensions/Safe.js

```
 |      Parameters
 |      ----------
 |      None
 |
 |      Returns
 |      -------
 |      c : Python object
 |          Possessing attributes data, shape, strides, etc.
 |
 |      See Also
 |      --------
 |      numpy.ctypeslib
 |
 |      Notes
 |      -----
 |      Below are the public attributes of this object which were documented
 |      in "Guide to NumPy" (we have omitted undocumented public attributes,
 |      as well as documented private attributes):
 |
 |      .. autoattribute:: numpy.core._internal._ctypes.data
 |          :noindex:
 |
 |      .. autoattribute:: numpy.core._internal._ctypes.shape
 |          :noindex:
 |
 |      .. autoattribute:: numpy.core._internal._ctypes.strides
 |          :noindex:
 |
 |      .. automethod:: numpy.core._internal._ctypes.data_as
 |          :noindex:
 |
 |      .. automethod:: numpy.core._internal._ctypes.shape_as
 |          :noindex:
 |
 |      .. automethod:: numpy.core._internal._ctypes.strides_as
 |          :noindex:
 |
 |      If the ctypes module is not available, then the ctypes attribute
 |      of array objects still returns something useful, but ctypes objects
 |      are not returned and errors may be raised instead. In particular,
 |      the object will still have the ``as_parameter`` attribute which will
 |      return an integer equal to the data attribute.
 |
 |      Examples
 |      --------
 |      >>> import ctypes
 |      >>> x = np.array([[0, 1], [2, 3]], dtype=np.int32)
 |      >>> x
 |      array([[0, 1],
 |             [2, 3]], dtype=int32)
 |      >>> x.ctypes.data
 |      31962608 # may vary
 |      >>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint32))
 |      <__main__.LP_c_uint object at 0x7ff2fc1fc200> # may vary
 |      >>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint32)).contents
 |      c_uint(0)
```

```
 |      >>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint64)).contents
 |      c_ulong(4294967296)
 |      >>> x.ctypes.shape
 |      <numpy.core._internal.c_long_Array_2 object at 0x7ff2fc1fce60> # may
vary
 |      >>> x.ctypes.strides
 |      <numpy.core._internal.c_long_Array_2 object at 0x7ff2fc1ff320> # may
vary
 |
 |  data
 |      Python buffer object pointing to the start of the array's data.
 |
 |  dtype
 |      Data-type of the array's elements.
 |
 |      .. warning::
 |
 |          Setting ``arr.dtype`` is discouraged and may be deprecated in th
e
 |          future.  Setting will replace the ``dtype`` without modifying th
e
 |          memory (see also `ndarray.view` and `ndarray.astype`).
 |
 |      Parameters
 |      ----------
 |      None
 |
 |      Returns
 |      -------
 |      d : numpy dtype object
 |
 |      See Also
 |      --------
 |      ndarray.astype : Cast the values contained in the array to a new dat
a-type.
 |      ndarray.view : Create a view of the same data but a different data-t
ype.
 |      numpy.dtype
 |
 |      Examples
 |      --------
 |      >>> x
 |      array([[0, 1],
 |             [2, 3]])
 |      >>> x.dtype
 |      dtype('int32')
 |      >>> type(x.dtype)
 |      <type 'numpy.dtype'>
 |
 |  flags
 |      Information about the memory layout of the array.
 |
 |      Attributes
 |      ----------
 |      C_CONTIGUOUS (C)
 |          The data is in a single, C-style contiguous segment.
```

```
 |      F_CONTIGUOUS (F)
 |          The data is in a single, Fortran-style contiguous segment.
 |      OWNDATA (O)
 |          The array owns the memory it uses or borrows it from another obj
ect.
 |      WRITEABLE (W)
 |          The data area can be written to.  Setting this to False locks
 |          the data, making it read-only.  A view (slice, etc.) inherits WR
ITEABLE
 |          from its base array at creation time, but a view of a writeable
 |          array may be subsequently locked while the base array remains wr
iteable.
 |          (The opposite is not true, in that a view of a locked array may
not
 |          be made writeable.  However, currently, locking a base object do
es not
 |          lock any views that already reference it, so under that circumst
ance it
 |          is possible to alter the contents of a locked array via a previo
usly
 |          created writeable view onto it.)  Attempting to change a non-wri
teable
 |          array raises a RuntimeError exception.
 |      ALIGNED (A)
 |          The data and all elements are aligned appropriately for the hard
ware.
 |      WRITEBACKIFCOPY (X)
 |          This array is a copy of some other array. The C-API function
 |          PyArray_ResolveWritebackIfCopy must be called before deallocatin
g
 |          to the base array will be updated with the contents of this arra
y.
 |      FNC
 |          F_CONTIGUOUS and not C_CONTIGUOUS.
 |      FORC
 |          F_CONTIGUOUS or C_CONTIGUOUS (one-segment test).
 |      BEHAVED (B)
 |          ALIGNED and WRITEABLE.
 |      CARRAY (CA)
 |          BEHAVED and C_CONTIGUOUS.
 |      FARRAY (FA)
 |          BEHAVED and F_CONTIGUOUS and not C_CONTIGUOUS.
 |
 |      Notes
 |      -----
 |      The `flags` object can be accessed dictionary-like (as in ``a.flags
['WRITEABLE']``),
 |      or by using lowercased attribute names (as in ``a.flags.writeable`
`). Short flag
 |      names are only supported in dictionary access.
 |
 |      Only the WRITEBACKIFCOPY, WRITEABLE, and ALIGNED flags can be
 |      changed by the user, via direct assignment to the attribute or dicti
onary
 |      entry, or by calling `ndarray.setflags`.
 |
```

Loading [MathJax]/extensions/Safe.js

```
 |      The array flags cannot be set arbitrarily:
 |
 |      - WRITEBACKIFCOPY can only be set ``False``.
 |      - ALIGNED can only be set ``True`` if the data is truly aligned.
 |      - WRITEABLE can only be set ``True`` if the array owns its own memor
y
 |        or the ultimate owner of the memory exposes a writeable buffer
 |        interface or is a string.
 |
 |      Arrays can be both C-style and Fortran-style contiguous simultaneous
ly.
 |      This is clear for 1-dimensional arrays, but can also be true for hig
her
 |      dimensional arrays.
 |
 |      Even for contiguous arrays a stride for a given dimension
 |      ``arr.strides[dim]`` may be *arbitrary* if ``arr.shape[dim] == 1``
 |      or the array has no elements.
 |      It does *not* generally hold that ``self.strides[-1] == self.itemsiz
e``
 |      for C-style contiguous arrays or ``self.strides[0] == self.itemsize`
` for
 |      Fortran-style contiguous arrays is true.
 |
 |  flat
 |      A 1-D iterator over the array.
 |
 |      This is a `numpy.flatiter` instance, which acts similarly to, but is
not
 |      a subclass of, Python's built-in iterator object.
 |
 |      See Also
 |      --------
 |      flatten : Return a copy of the array collapsed into one dimension.
 |
 |      flatiter
 |
 |      Examples
 |      --------
 |      >>> x = np.arange(1, 7).reshape(2, 3)
 |      >>> x
 |      array([[1, 2, 3],
 |             [4, 5, 6]])
 |      >>> x.flat[3]
 |      4
 |      >>> x.T
 |      array([[1, 4],
 |             [2, 5],
 |             [3, 6]])
 |      >>> x.T.flat[3]
 |      5
 |      >>> type(x.flat)
 |      <class 'numpy.flatiter'>
 |
 |      An assignment example:
 |
```

Loading [MathJax]/extensions/Safe.js

```
 |      >>> x.flat = 3; x
 |      array([[3, 3, 3],
 |             [3, 3, 3]])
 |      >>> x.flat[[1,4]] = 1; x
 |      array([[3, 1, 3],
 |             [3, 1, 3]])
 |
 |  imag
 |      The imaginary part of the array.
 |
 |      Examples
 |      --------
 |      >>> x = np.sqrt([1+0j, 0+1j])
 |      >>> x.imag
 |      array([ 0.        ,  0.70710678])
 |      >>> x.imag.dtype
 |      dtype('float64')
 |
 |  itemsize
 |      Length of one array element in bytes.
 |
 |      Examples
 |      --------
 |      >>> x = np.array([1,2,3], dtype=np.float64)
 |      >>> x.itemsize
 |      8
 |      >>> x = np.array([1,2,3], dtype=np.complex128)
 |      >>> x.itemsize
 |      16
 |
 |  nbytes
 |      Total bytes consumed by the elements of the array.
 |
 |      Notes
 |      -----
 |      Does not include memory consumed by non-element attributes of the
 |      array object.
 |
 |      See Also
 |      --------
 |      sys.getsizeof
 |          Memory consumed by the object itself without parents in case vie
w.
 |          This does include memory consumed by non-element attributes.
 |
 |      Examples
 |      --------
 |      >>> x = np.zeros((3,5,2), dtype=np.complex128)
 |      >>> x.nbytes
 |      480
 |      >>> np.prod(x.shape) * x.itemsize
 |      480
 |
 |  ndim
 |      Number of array dimensions.
 |
```

Loading [MathJax]/extensions/Safe.js

```
 |      Examples
 |      --------
 |      >>> x = np.array([1, 2, 3])
 |      >>> x.ndim
 |      1
 |      >>> y = np.zeros((2, 3, 4))
 |      >>> y.ndim
 |      3
 |
 |  real
 |      The real part of the array.
 |
 |      Examples
 |      --------
 |      >>> x = np.sqrt([1+0j, 0+1j])
 |      >>> x.real
 |      array([ 1.        ,  0.70710678])
 |      >>> x.real.dtype
 |      dtype('float64')
 |
 |      See Also
 |      --------
 |      numpy.real : equivalent function
 |
 |  shape
 |      Tuple of array dimensions.
 |
 |      The shape property is usually used to get the current shape of an ar
ray,
 |      but may also be used to reshape the array in-place by assigning a tu
ple of
 |      array dimensions to it.  As with `numpy.reshape`, one of the new sha
pe
 |      dimensions can be -1, in which case its value is inferred from the s
ize of
 |      the array and the remaining dimensions. Reshaping an array in-place
will
 |      fail if a copy is required.
 |
 |      .. warning::
 |
 |          Setting ``arr.shape`` is discouraged and may be deprecated in th
e
 |          future.  Using `ndarray.reshape` is the preferred approach.
 |
 |      Examples
 |      --------
 |      >>> x = np.array([1, 2, 3, 4])
 |      >>> x.shape
 |      (4,)
 |      >>> y = np.zeros((2, 3, 4))
 |      >>> y.shape
 |      (2, 3, 4)
 |      >>> y.shape = (3, 8)
 |      >>> y
 |      array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
```

```
 |             [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
 |             [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
 |      >>> y.shape = (3, 6)
 |      Traceback (most recent call last):
 |        File "<stdin>", line 1, in <module>
 |      ValueError: total size of new array must be unchanged
 |      >>> np.zeros((4,2))[::2].shape = (-1,)
 |      Traceback (most recent call last):
 |        File "<stdin>", line 1, in <module>
 |      AttributeError: Incompatible shape for in-place modification. Use
 |      `.reshape()` to make a copy with the desired shape.
 |
 |      See Also
 |      --------
 |      numpy.shape : Equivalent getter function.
 |      numpy.reshape : Function similar to setting ``shape``.
 |      ndarray.reshape : Method similar to setting ``shape``.
 |
 |  size
 |      Number of elements in the array.
 |
 |      Equal to ``np.prod(a.shape)``, i.e., the product of the array's
 |      dimensions.
 |
 |      Notes
 |      -----
 |      `a.size` returns a standard arbitrary precision Python integer. This
 |      may not be the case with other methods of obtaining the same value
 |      (like the suggested ``np.prod(a.shape)``, which returns an instance
 |      of ``np.int_``), and may be relevant if the value is used further in
 |      calculations that may overflow a fixed size integer type.
 |
 |      Examples
 |      --------
 |      >>> x = np.zeros((3, 5, 2), dtype=np.complex128)
 |      >>> x.size
 |      30
 |      >>> np.prod(x.shape)
 |      30
 |
 |  strides
 |      Tuple of bytes to step in each dimension when traversing an array.
 |
 |      The byte offset of element ``(i[0], i[1], ..., i[n])`` in an array `
a`
 |      is::
 |
 |          offset = sum(np.array(i) * a.strides)
 |
 |      A more detailed explanation of strides can be found in the
 |      "ndarray.rst" file in the NumPy reference guide.
 |
 |      .. warning::
 |
 |          Setting ``arr.strides`` is discouraged and may be deprecated in
the
```

Loading [MathJax]/extensions/Safe.js

```
 |          future.  `numpy.lib.stride_tricks.as_strided` should be preferre
d
 |          to create a new view of the same data in a safer way.
 |
 |      Notes
 |      -----
 |      Imagine an array of 32-bit integers (each 4 bytes)::
 |
 |        x = np.array([[0, 1, 2, 3, 4],
 |                      [5, 6, 7, 8, 9]], dtype=np.int32)
 |
 |      This array is stored in memory as 40 bytes, one after the other
 |      (known as a contiguous block of memory).  The strides of an array te
ll
 |      us how many bytes we have to skip in memory to move to the next posi
tion
 |      along a certain axis.  For example, we have to skip 4 bytes (1 valu
e) to
 |      move to the next column, but 20 bytes (5 values) to get to the same
 |      position in the next row.  As such, the strides for the array `x` wi
ll be
 |      ``(20, 4)``.
 |
 |      See Also
 |      --------
 |      numpy.lib.stride_tricks.as_strided
 |
 |      Examples
 |      --------
 |      >>> y = np.reshape(np.arange(2*3*4), (2,3,4))
 |      >>> y
 |      array([[[ 0,  1,  2,  3],
 |              [ 4,  5,  6,  7],
 |              [ 8,  9, 10, 11]],
 |             [[12, 13, 14, 15],
 |              [16, 17, 18, 19],
 |              [20, 21, 22, 23]]])
 |      >>> y.strides
 |      (48, 16, 4)
 |      >>> y[1,1,1]
 |      17
 |      >>> offset=sum(y.strides * np.array((1,1,1)))
 |      >>> offset/y.itemsize
 |      17
 |
 |      >>> x = np.reshape(np.arange(5*6*7*8), (5,6,7,8)).transpose(2,3,1,0)
 |      >>> x.strides
 |      (32, 4, 224, 1344)
 |      >>> i = np.array([3,5,2,2])
 |      >>> offset = sum(i * x.strides)
 |      >>> x[3,5,2,2]
 |      813
 |      >>> offset / x.itemsize
 |      813
 |
 |  ----------------------------------------------------------------------
```

Loading [MathJax]/extensions/Safe.js

```
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```

(Of course, you might also prefer to check out the online docs.)

Operator overloading

What's the value of the below expression?

In [20]:
```
[3, 4, 1, 2, 2, 1] + 10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 1
----> 1 [3, 4, 1, 2, 2, 1] + 10

TypeError: can only concatenate list (not "int") to list
```

What a silly question. Of course it's an error.

But what about...

In [21]:
```
rolls + 10
```

Out[21]:
```
array([11, 11, 14, 13, 12, 11, 15, 11, 12, 12])
```

We might think that Python strictly polices how pieces of its core syntax behave such as `+`, `<`, `in`, `==`, or square brackets for indexing and slicing. But in fact, it takes a very hands-off approach. When you define a new type, you can choose how addition works for it, or what it means for an object of that type to be equal to something else.

The designers of lists decided that adding them to numbers wasn't allowed. The designers of `numpy` arrays went a different way (adding the number to each element of the array).

Here are a few more examples of how `numpy` arrays interact unexpectedly with Python operators (or at least differently from lists).

In [22]:
```
# At which indices are the dice less than or equal to 3?
rolls <= 3
```

Out[22]:
```
array([ True,  True, False,  True,  True,  True, False,  True,  True,
        True])
```

In [23]:
```
xlist = [[1,2,3],[2,4,6],]
# Create a 2-dimensional array
x = numpy.asarray(xlist)
print("xlist = {}\nx =\n{}".format(xlist, x))
```

Loading [MathJax]/extensions/Safe.js

```
xlist = [[1, 2, 3], [2, 4, 6]]
x =
[[1 2 3]
 [2 4 6]]
```

In [24]:
```
# Get the last element of the second row of our numpy array
x[1,-1]
```

Out[24]:
```
6
```

In [25]:
```
# Get the last element of the second sublist of our nested list?
xlist[1,-1]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[25], line 2
      1 # Get the last element of the second sublist of our nested list?
----> 2 xlist[1,-1]

TypeError: list indices must be integers or slices, not tuple
```

numpy's `ndarray` type is specialized for working with multi-dimensional data, so it defines its own logic for indexing, allowing us to index by a tuple to specify the index at each dimension.

When does 1 + 1 not equal 2?

Things can get weirder than this. You may have heard of (or even used) tensorflow, a Python library popularly used for deep learning. It makes extensive use of operator overloading.

In [26]:
```
import tensorflow as tf
# Create two constants, each with value 1
a = tf.constant(1)
b = tf.constant(1)
# Add them together to get...
a + b
```

```
2025-03-11 19:48:47.911759: I metal_plugin/src/device/metal_device.cc:1154] Metal device set to: Apple M4 Pro
2025-03-11 19:48:47.911781: I metal_plugin/src/device/metal_device.cc:296] systemMemory: 24.00 GB
2025-03-11 19:48:47.911789: I metal_plugin/src/device/metal_device.cc:313] maxCacheSize: 8.00 GB
2025-03-11 19:48:47.912030: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA support.
2025-03-11 19:48:47.912037: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
```

Loading [MathJax]/extensions/Safe.js

Out[26]: <tf.Tensor: shape=(), dtype=int32, numpy=2>

`a + b` isn't 2, it is (to quote tensorflow's documentation)...

> a symbolic handle to one of the outputs of an `Operation`. It does not hold the values of that operation's output, but instead provides a means of computing those values in a TensorFlow `tf.Session`.

It's important just to be aware of the fact that this sort of thing is possible and that libraries will often use operator overloading in non-obvious or magical-seeming ways.

Understanding how Python's operators work when applied to ints, strings, and lists is no guarantee that you'll be able to immediately understand what they do when applied to a tensorflow `Tensor`, or a numpy `ndarray`, or a pandas `DataFrame`.

Once you've had a little taste of DataFrames, for example, an expression like the one below starts to look appealingly intuitive:

```
# Get the rows with population over 1m in South America
df[(df['population'] > 10**6) & (df['continent'] == 'South America')]
```

But why does it work? The example above features something like **5** different overloaded operators. What's each of those operations doing? It can help to know the answer when things start going wrong.

Curious how it all works?

Have you ever called `help()` or `dir()` on an object and wondered what the heck all those names with the double-underscores were?

In [27]:
```
print(dir(list))
```

```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getstate__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

This turns out to be directly related to operator overloading.

When Python programmers want to define how operators behave on their types, they do so by implementing methods with special names beginning and ending with 2 underscores such as `__lt__`, `__setattr__`, or `__contains__`. Generally, names that follow this double-underscore format have a special meaning to Python.

Loading [MathJax]/extensions/Safe.js

So, for example, the expression `x in [1, 2, 3]` is actually calling the list method `__contains__` behind-the-scenes. It's equivalent to (the much uglier) `[1, 2, 3].__contains__(x)`.

If you're curious to learn more, you can check out Python's official documentation, which describes many, many more of these special "underscores" methods.

We won't be defining our own types in these lessons (if only there was time!), but I hope you'll get to experience the joys of defining your own wonderful, weird types later down the road.

Your turn!

Head over to **the final coding exercise** for one more round of coding questions involving imports, working with unfamiliar objects, and, of course, more gambling.

Loading [MathJax]/extensions/Safe.js