```
"""
The wavio module defines the functions:

read(file)
    Read a WAV file and return a `wavio.Wav` object, with attributes
    `data`, `rate` and `sampwidth`.

write(filename, data, rate, scale=None, sampwidth=None)
    Write a numpy array to a WAV file.


-----
Author: Warren Weckesser
License: BSD 2-Clause:
Copyright (c) 2015, Warren Weckesser
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
"""

from __future__ import division as _division

import wave as _wave
import numpy as _np


__version__ = "0.0.3"


def _wav2array(nchannels, sampwidth, data):
    """data must be the string containing the bytes from the wav file."""
    num_samples, remainder = divmod(len(data), sampwidth * nchannels)
    if remainder > 0:
        raise ValueError('The length of data is not a multiple of '
                         'sampwidth * num_channels.')
    if sampwidth > 4:
        raise ValueError("sampwidth must not be greater than 4.")

    if sampwidth == 3:
        a = _np.empty((num_samples, nchannels, 4), dtype=_np.uint8)
```

```
        raw_bytes = _np.fromstring(data, dtype=_np.uint8)
        a[:, :, :sampwidth] = raw_bytes.reshape(-1, nchannels, sampwidth)
        a[:, :, sampwidth:] = (a[:, :, sampwidth - 1:sampwidth] >> 7) * 255
        result = a.view('<i4').reshape(a.shape[:-1])
    else:
        # 8 bit samples are stored as unsigned ints; others as signed ints.
        dt_char = 'u' if sampwidth == 1 else 'i'
        a = _np.fromstring(data, dtype='<%s%d' % (dt_char, sampwidth))
        result = a.reshape(-1, nchannels)
    return result


def _array2wav(a, sampwidth):
    """
    Convert the input array `a` to a string of WAV data.

    a.dtype must be one of uint8, int16 or int32.  Allowed sampwidth
    values are:
        dtype    sampwidth
        uint8        1
        int16        2
        int32      3 or 4
    When sampwidth is 3, the *low* bytes of `a` are assumed to contain
    the values to include in the string.
    """
    if sampwidth == 3:
        # `a` must have dtype int32
        if a.ndim == 1:
            # Convert to a 2D array with a single column.
            a = a.reshape(-1, 1)
        # By shifting first 0 bits, then 8, then 16, the resulting output
        # is 24 bit little-endian.
        a8 = (a.reshape(a.shape + (1,)) >> _np.array([0, 8, 16])) & 255
        wavdata = a8.astype(_np.uint8).tostring()
    else:
        # Make sure the array is little-endian, and then convert using
        # tostring()
        a = a.astype('<' + a.dtype.str[1:], copy=False)
        wavdata = a.tostring()
    return wavdata


class Wav(object):
    """
    Object returned by `wavio.read`.  Attributes are:

    data : numpy array
        The array of data read from the WAV file.
    rate : float
        The sample rate of the WAV file.
    sampwidth : int
        The sample width (i.e. number of bytes per sample) of the WAV file.
        For example, `sampwidth == 3` is a 24 bit WAV file.

    """

    def __init__(self, data, rate, sampwidth):
        self.data = data
        self.rate = rate
        self.sampwidth = sampwidth
```

```
    def __repr__(self):
        s = ("Wav(data.shape=%s, data.dtype=%s, rate=%r, sampwidth=%r)" %
             (self.data.shape, self.data.dtype, self.rate, self.sampwidth))
        return s


def read(file):
    """
    Read a WAV file.

    Parameters
    ----------
    file : string or file object
        Either the name of a file or an open file pointer.

    Returns
    -------
    wav : wavio.Wav() instance
        The return value is an instance of the class `wavio.Wav`,
        with the following attributes:

            data : numpy array
                The array containing the data.  The shape of the array
                is (num_samples, num_channels).  num_channels is the
                number of audio channels (1 for mono, 2 for stereo).
            rate : float
                The sampling frequency (i.e. frame rate)
            sampwidth : float
                The sample width, in bytes.  E.g. for a 24 bit WAV file,
                sampwidth is 3.

    Notes
    -----
    This function uses the `wave` module of the Python standard libary
    to read the WAV file, so it has the same limitations as that library.
    In particular, the function does not read compressed WAV files, and
    it does not read files with floating point data.

    The array returned by `wavio.read` is alway two-dimensional.  If the
    WAV data is mono, the array will have shape (num_samples, 1).

    """
    wav = _wave.open(file)
    rate = wav.getframerate()
    nchannels = wav.getnchannels()
    sampwidth = wav.getsampwidth()
    nframes = wav.getnframes()
    data = wav.readframes(nframes)
    wav.close()
    array = _wav2array(nchannels, sampwidth, data)
    w = Wav(data=array, rate=rate, sampwidth=sampwidth)
    return w


_sampwidth_dtypes = {1: _np.uint8,
                     2: _np.int16,
                     3: _np.int32,
                     4: _np.int32}
_sampwidth_ranges = {1: (0, 256),
```

```
                    2: (-2**15, 2**15),
                    3: (-2**23, 2**23),
                    4: (-2**31, 2**31)}


def _scale_to_sampwidth(data, sampwidth, vmin, vmax):
    # Scale and translate the values to fit the range of the data type
    # associated with the given sampwidth.

    data = data.clip(vmin, vmax)

    dt = _sampwidth_dtypes[sampwidth]
    if vmax == vmin:
        data = _np.zeros(data.shape, dtyp=dt)
    else:
        outmin, outmax = _sampwidth_ranges[sampwidth]
        if outmin != vmin or outmax != vmax:
            data = ((float(outmax - outmin)) * (data - vmin) /
                    (vmax - vmin)).astype(_np.int64) + outmin
            data[data == outmax] = outmax - 1
        data = data.astype(dt)

    return data


def write(file, data, rate, scale=None, sampwidth=None):
    """
    Write the numpy array `data` to a WAV file.

    The Python standard library "wave" is used to write the data
    to the file, so this function has the same limitations as that
    module.  In particular, the Python library does not support
    floating point data.  When given a floating point array, this
    function converts the values to integers.  See below for the
    conversion rules.

    Parameters
    ----------
    file : string, or file object open for writing in binary mode
        Either the name of a file or an open file pointer.
    data : numpy array, 1- or 2-dimensional, integer or floating point
        If it is 2-d, the rows are the frames (i.e. samples) and the
        columns are the channels.
    rate : float
        The sampling frequency (i.e. frame rate) of the data.
    sampwidth : int, optional
        The sample width, in bytes, of the output file.
        If `sampwidth` is not given, it is inferred (if possible) from
        the data type of `data`, as follows::

            data.dtype     sampwidth
            ----------     ---------
            uint8, int8        1
            uint16, int16      2
            uint32, int32      4

        For any other data types, or to write a 24 bit file, `sampwidth`
        must be given.
    scale : tuple or str, optional
        By default, the data written to the file is scaled up or down to
```

```
    occupy the full range of the output data type.  So, for example,
    the unsigned 8 bit data [0, 1, 2, 15] would be written to the file
    as [0, 17, 30, 255].  More generally, the default behavior is
    (roughly)::

        vmin = data.min()
        vmax = data.max()
        outmin = <minimum integer of the output dtype>
        outmax = <maximum integer of the output dtype>
        outdata = (outmax - outmin)*(data - vmin)/(vmax - vmin) + outmin

    The `scale` argument allows the scaling of the output data to be
    changed.  `scale` can be a tuple of the form `(vmin, vmax)`, in which
    case the given values override the use of `data.min()` and
    `data.max()` for `vmin` and `vmax` shown above.  (If either value
    is `None`, the value shown above is used.)  Data outside the
    range (vmin, vmax) is clipped.

    If `scale` is the string "none", then `vmin` and `vmax` are set to
    `outmin` and `outmax`, respectively. This means the data is written
    to the file with no scaling.  (Note: `scaling="none" is not the same
    as `scaling=None`.  The latter means "use the default behavior",
    which is to scale by the data minimum and maximum.)

    If `scale` is the string "dtype-limits", then `vmin` and `vmax`
    are set to the minimum and maximum integers of `data.dtype`.
    The string "dtype-limits" is not allowed when the `data` is a
    floating point array.

    If using `scale` results in values that exceed the limits of the
    output sample width, the data is clipped.  For example, the
    following code::

        >>> x = np.array([-100, 0, 100, 200, 300, 325])
        >>> wavio.write('foo.wav', x, 8000, scale='none', sampwidth=1)

    will write the values [0, 0, 100, 200, 255, 255] to the file.

Example
-------
Create a 3 second 440 Hz sine wave, and save it in a 24-bit WAV file.

>>> import numpy as np
>>> import wavio
>>> rate = 22050  # samples per second
>>> T = 3         # sample duration (seconds)
>>> f = 440.0     # sound frequency (Hz)
>>> t = np.linspace(0, T, T*rate, endpoint=False)
>>> x = np.sin(2*np.pi * f * t)
>>> wavio.write("sine24.wav", x, rate, sampwidth=3)

Create a file that contains the 16 bit integer values -10000 and 10000
repeated 100 times.  Don't automatically scale the values.  Use a sample
rate 8000.

>>> x = np.empty(200, dtype=np.int16)
>>> x[::2] = -10000
>>> x[1::2] = 10000
>>> wavio.write("foo.wav", x, 8000, scale='none')
```

```
Check that the file contains what we expect.

>>> w = wavio.read("foo.wav")
>>> np.all(w.data[:, 0] == x)
True

In the following, the values -10000 and 10000 (from within the 16 bit
range [-2**15, 2**15-1]) are mapped to the corresponding values 88 and
168 (in the range [0, 2**8-1]).

>>> wavio.write("foo.wav", x, 8000, sampwidth=1, scale='dtype-limits')
>>> w = wavio.read("foo.wav")
>>> w.data[:4, 0]
array([ 88, 168,  88, 168], dtype=uint8)

"""

if sampwidth is None:
    if not _np.issubdtype(data.dtype, _np.integer) or data.itemsize > 4:
        raise ValueError('when data.dtype is not an 8-, 16-, or 32-bit '
                         'integer type, sampwidth must be specified.')
    sampwidth = data.itemsize
else:
    if sampwidth not in [1, 2, 3, 4]:
        raise ValueError('sampwidth must be 1, 2, 3 or 4.')

outdtype = _sampwidth_dtypes[sampwidth]
outmin, outmax = _sampwidth_ranges[sampwidth]

if scale == "none":
    data = data.clip(outmin, outmax-1).astype(outdtype)
elif scale == "dtype-limits":
    if not _np.issubdtype(data.dtype, _np.integer):
        raise ValueError("scale cannot be 'dtype-limits' with "
                         "non-integer data.")
    # Easy transforms that just changed the signedness of the data.
    if sampwidth == 1 and data.dtype == _np.int8:
        data = (data.astype(_np.int16) + 128).astype(_np.uint8)
    elif sampwidth == 2 and data.dtype == _np.uint16:
        data = (data.astype(_np.int32) - 32768).astype(_np.int16)
    elif sampwidth == 4 and data.dtype == _np.uint32:
        data = (data.astype(_np.int64) - 2**31).astype(_np.int32)
    elif data.itemsize != sampwidth:
        # Integer input, but rescaling is needed to adjust the
        # input range to the output sample width.
        ii = _np.iinfo(data.dtype)
        vmin = ii.min
        vmax = ii.max
        data = _scale_to_sampwidth(data, sampwidth, vmin, vmax)
else:
    if scale is None:
        vmin = data.min()
        vmax = data.max()
    else:
        # scale must be a tuple of the form (vmin, vmax)
        vmin, vmax = scale
        if vmin is None:
            vmin = data.min()
        if vmax is None:
            vmax = data.max()
```

```
    data = _scale_to_sampwidth(data, sampwidth, vmin, vmax)

# At this point, `data` has been converted to have one of the following:
#    sampwidth   dtype
#    ---------   -----
#        1       uint8
#        2       int16
#        3       int32
#        4       int32
# The values in `data` are in the form in which they will be saved;
# no more scaling will take place.

if data.ndim == 1:
    data = data.reshape(-1, 1)

wavdata = _array2wav(data, sampwidth)

w = _wave.open(file, 'wb')
w.setnchannels(data.shape[1])
w.setsampwidth(sampwidth)
w.setframerate(rate)
w.writeframes(wavdata)
w.close()
```