Writing compressed data to an already-open file:

```
import lzma
with open("file.xz", "wb") as f:
    f.write(b"This data will not be compressed\n")
    with lzma.open(f, "w") as lzf:
        lzf.write(b"This *will* be compressed\n")
    f.write(b"Not compressed\n")
```

Creating a compressed file using a custom filter chain:

```
import lzma
my_filters = [
    {"id": lzma.FILTER_DELTA, "dist": 5},
    {"id": lzma.FILTER_LZMA2, "preset": 7 | lzma.PRESET_EXTREME},
]
with lzma.open("file.xz", "w", filters=my_filters) as f:
    f.write(b"blah blah blah")
```

13.5 `zipfile` — Work with ZIP archives

Source code: Lib/zipfile.py

The ZIP file format is a common archive and compression standard. This module provides tools to create, read, write, append, and list a ZIP file. Any advanced use of this module will require an understanding of the format, as defined in PKZIP Application Note.

This module does not currently handle multi-disk ZIP files. It can handle ZIP files that use the ZIP64 extensions (that is ZIP files that are more than 4 GiB in size). It supports decryption of encrypted files in ZIP archives, but it currently cannot create an encrypted file. Decryption is extremely slow as it is implemented in native Python rather than C.

The module defines the following items:

exception zipfile.**BadZipFile**

The error raised for bad ZIP files.

New in version 3.2.

exception zipfile.**BadZipfile**

Alias of *BadZipFile*, for compatibility with older Python versions.

Deprecated since version 3.2.

exception zipfile.**LargeZipFile**

The error raised when a ZIP file would require ZIP64 functionality but that has not been enabled.

class zipfile.**ZipFile**

The class for reading and writing ZIP files. See section *ZipFile Objects* for constructor details.

class zipfile.**Path**

Class that implements a subset of the interface provided by *pathlib.Path*, including the full *importlib.resources.abc.Traversable* interface.

New in version 3.8.

class zipfile.**PyZipFile**

Class for creating ZIP archives containing Python libraries.

class zipfile.**ZipInfo**(*filename='NoName'*, *date_time=(1980, 1, 1, 0, 0, 0)*)

Class used to represent information about a member of an archive. Instances of this class are returned by the *getinfo()* and *infolist()* methods of *ZipFile* objects. Most users of the *zipfile* module will not need to create these, but only use those created by this module. *filename* should be the full name of the archive member, and *date_time* should be a tuple containing six fields which describe the time of the last modification to the file; the fields are described in section *ZipInfo Objects*.

zipfile.**is_zipfile**(*filename*)

Returns `True` if *filename* is a valid ZIP file based on its magic number, otherwise returns `False`. *filename* may be a file or file-like object too.

Changed in version 3.1: Support for file and file-like objects.

zipfile.**ZIP_STORED**

The numeric constant for an uncompressed archive member.

zipfile.**ZIP_DEFLATED**

The numeric constant for the usual ZIP compression method. This requires the *zlib* module.

zipfile.**ZIP_BZIP2**

The numeric constant for the BZIP2 compression method. This requires the *bz2* module.

New in version 3.3.

zipfile.**ZIP_LZMA**

The numeric constant for the LZMA compression method. This requires the *lzma* module.

New in version 3.3.

Note: The ZIP file format specification has included support for bzip2 compression since 2001, and for LZMA compression since 2006. However, some tools (including older Python releases) do not support these compression methods, and may either refuse to process the ZIP file altogether, or fail to extract individual files.

See also:

PKZIP Application Note Documentation on the ZIP file format by Phil Katz, the creator of the format and algorithms used.

Info-ZIP Home Page Information about the Info-ZIP project's ZIP archive programs and development libraries.

13.5.1 ZipFile Objects

class zipfile.**ZipFile**(*file*, *mode='r'*, *compression=ZIP_STORED*, *allowZip64=True*, *compresslevel=None*, ***, *strict_timestamps=True*, *metadata_encoding=None*)

Open a ZIP file, where *file* can be a path to a file (a string), a file-like object or a *path-like object*.

The *mode* parameter should be `'r'` to read an existing file, `'w'` to truncate and write a new file, `'a'` to append to an existing file, or `'x'` to exclusively create and write a new file. If *mode* is `'x'` and *file* refers to an existing file, a *FileExistsError* will be raised. If *mode* is `'a'` and *file* refers to an existing ZIP file, then additional files are added to it. If *file* does not refer to a ZIP file, then a new ZIP archive is appended to the file. This is meant for adding a ZIP archive to another file (such as `python.exe`). If *mode* is `'a'` and the file does not exist at all, it is created. If *mode* is `'r'` or `'a'`, the file should be seekable.

compression is the ZIP compression method to use when writing the archive, and should be *ZIP_STORED*, *ZIP_DEFLATED*, *ZIP_BZIP2* or *ZIP_LZMA*; unrecognized values will cause *NotImplementedError* to be raised. If *ZIP_DEFLATED*, *ZIP_BZIP2* or *ZIP_LZMA* is specified but the corresponding module (*zlib*, *bz2* or *lzma*) is not available, *RuntimeError* is raised. The default is *ZIP_STORED*.

If *allowZip64* is `True` (the default) zipfile will create ZIP files that use the ZIP64 extensions when the zipfile is larger than 4 GiB. If it is `false` *zipfile* will raise an exception when the ZIP file would require ZIP64 extensions.

The *compresslevel* parameter controls the compression level to use when writing files to the archive. When using *ZIP_STORED* or *ZIP_LZMA* it has no effect. When using *ZIP_DEFLATED* integers 0 through 9 are accepted (see *zlib* for more information). When using *ZIP_BZIP2* integers 1 through 9 are accepted (see *bz2* for more information).

The *strict_timestamps* argument, when set to False, allows to zip files older than 1980-01-01 at the cost of setting the timestamp to 1980-01-01. Similar behavior occurs with files newer than 2107-12-31, the timestamp is also set to the limit.

When mode is 'r', *metadata_encoding* may be set to the name of a codec, which will be used to decode metadata such as the names of members and ZIP comments.

If the file is created with mode 'w', 'x' or 'a' and then *closed* without adding any files to the archive, the appropriate ZIP structures for an empty archive will be written to the file.

ZipFile is also a context manager and therefore supports the with statement. In the example, *myzip* is closed after the with statement's suite is finished—even if an exception occurs:

```
with ZipFile('spam.zip', 'w') as myzip:
    myzip.write('eggs.txt')
```

Note: *metadata_encoding* is an instance-wide setting for the ZipFile. It is not currently possible to set this on a per-member basis.

This attribute is a workaround for legacy implementations which produce archives with names in the current locale encoding or code page (mostly on Windows). According to the .ZIP standard, the encoding of metadata may be specified to be either IBM code page (default) or UTF-8 by a flag in the archive header. That flag takes precedence over *metadata_encoding*, which is a Python-specific extension.

New in version 3.2: Added the ability to use *ZipFile* as a context manager.

Changed in version 3.3: Added support for *bzip2* and *lzma* compression.

Changed in version 3.4: ZIP64 extensions are enabled by default.

Changed in version 3.5: Added support for writing to unseekable streams. Added support for the 'x' mode.

Changed in version 3.6: Previously, a plain *RuntimeError* was raised for unrecognized compression values.

Changed in version 3.6.2: The *file* parameter accepts a *path-like object*.

Changed in version 3.7: Add the *compresslevel* parameter.

New in version 3.8: The *strict_timestamps* keyword-only argument

Changed in version 3.11: Added support for specifying member name encoding for reading metadata in the zipfile's directory and file headers.

ZipFile.**close**()

Close the archive file. You must call *close()* before exiting your program or essential records will not be written.

ZipFile.**getinfo**(*name*)

Return a *ZipInfo* object with information about the archive member *name*. Calling *getinfo()* for a name not currently contained in the archive will raise a *KeyError*.

ZipFile.**infolist**()

Return a list containing a *ZipInfo* object for each member of the archive. The objects are in the same order as their entries in the actual ZIP file on disk if an existing archive was opened.

ZipFile.**namelist**()

Return a list of archive members by name.

ZipFile.**open**(*name*, *mode='r'*, *pwd=None*, *, *force_zip64=False*)

Access a member of the archive as a binary file-like object. *name* can be either the name of a file within the archive or a *ZipInfo* object. The *mode* parameter, if included, must be `'r'` (the default) or `'w'`. *pwd* is the password used to decrypt encrypted ZIP files as a *bytes* object.

open() is also a context manager and therefore supports the `with` statement:

```
with ZipFile('spam.zip') as myzip:
    with myzip.open('eggs.txt') as myfile:
        print(myfile.read())
```

With *mode* `'r'` the file-like object (`ZipExtFile`) is read-only and provides the following methods: *read()*, *readline()*, *readlines()*, *seek()*, *tell()*, *__iter__()*, *__next__()*. These objects can operate independently of the ZipFile.

With `mode='w'`, a writable file handle is returned, which supports the *write()* method. While a writable file handle is open, attempting to read or write other files in the ZIP file will raise a *ValueError*.

When writing a file, if the file size is not known in advance but may exceed 2 GiB, pass `force_zip64=True` to ensure that the header format is capable of supporting large files. If the file size is known in advance, construct a *ZipInfo* object with *file_size* set, and use that as the *name* parameter.

Note: The *open()*, *read()* and *extract()* methods can take a filename or a *ZipInfo* object. You will appreciate this when trying to read a ZIP file that contains members with duplicate names.

Changed in version 3.6: Removed support of `mode='U'`. Use *io.TextIOWrapper* for reading compressed text files in *universal newlines* mode.

Changed in version 3.6: *ZipFile.open()* can now be used to write files into the archive with the `mode='w'` option.

Changed in version 3.6: Calling *open()* on a closed ZipFile will raise a *ValueError*. Previously, a *RuntimeError* was raised.

ZipFile.**extract**(*member*, *path=None*, *pwd=None*)

Extract a member from the archive to the current working directory; *member* must be its full name or a *ZipInfo* object. Its file information is extracted as accurately as possible. *path* specifies a different directory to extract to. *member* can be a filename or a *ZipInfo* object. *pwd* is the password used for encrypted files as a *bytes* object.

Returns the normalized path created (a directory or new file).

Note: If a member filename is an absolute path, a drive/UNC sharepoint and leading (back)slashes will be stripped, e.g.: `///foo/bar` becomes `foo/bar` on Unix, and `C:\foo\bar` becomes `foo\bar` on Windows. And all `".."` components in a member filename will be removed, e.g.: `../../foo../../ba..r` becomes `foo../ba..r`. On Windows illegal characters (`:`, `<`, `>`, `|`, `"`, `?`, and `*`) replaced by underscore (_).

Changed in version 3.6: Calling *extract()* on a closed ZipFile will raise a *ValueError*. Previously, a *RuntimeError* was raised.

Changed in version 3.6.2: The *path* parameter accepts a *path-like object*.

ZipFile.**extractall**(*path=None*, *members=None*, *pwd=None*)

Extract all members from the archive to the current working directory. *path* specifies a different directory to extract to. *members* is optional and must be a subset of the list returned by *namelist()*. *pwd* is the password used for encrypted files as a *bytes* object.

Warning: Never extract archives from untrusted sources without prior inspection. It is possible that files are created outside of *path*, e.g. members that have absolute filenames starting with `"/"` or filenames with two dots `".."`. This module attempts to prevent that. See `extract()` note.

Changed in version 3.6: Calling `extractall()` on a closed ZipFile will raise a `ValueError`. Previously, a `RuntimeError` was raised.

Changed in version 3.6.2: The *path* parameter accepts a *path-like object*.

`ZipFile.`**`printdir`**`()`

Print a table of contents for the archive to `sys.stdout`.

`ZipFile.`**`setpassword`**`(`*pwd*`)`

Set *pwd* (a `bytes` object) as default password to extract encrypted files.

`ZipFile.`**`read`**`(`*name*, *pwd=None*`)`

Return the bytes of the file *name* in the archive. *name* is the name of the file in the archive, or a `ZipInfo` object. The archive must be open for read or append. *pwd* is the password used for encrypted files as a `bytes` object and, if specified, overrides the default password set with `setpassword()`. Calling `read()` on a ZipFile that uses a compression method other than `ZIP_STORED`, `ZIP_DEFLATED`, `ZIP_BZIP2` or `ZIP_LZMA` will raise a `NotImplementedError`. An error will also be raised if the corresponding compression module is not available.

Changed in version 3.6: Calling `read()` on a closed ZipFile will raise a `ValueError`. Previously, a `RuntimeError` was raised.

`ZipFile.`**`testzip`**`()`

Read all the files in the archive and check their CRC's and file headers. Return the name of the first bad file, or else return `None`.

Changed in version 3.6: Calling `testzip()` on a closed ZipFile will raise a `ValueError`. Previously, a `RuntimeError` was raised.

`ZipFile.`**`write`**`(`*filename*, *arcname=None*, *compress_type=None*, *compresslevel=None*`)`

Write the file named *filename* to the archive, giving it the archive name *arcname* (by default, this will be the same as *filename*, but without a drive letter and with leading path separators removed). If given, *compress_type* overrides the value given for the *compression* parameter to the constructor for the new entry. Similarly, *compresslevel* will override the constructor if given. The archive must be open with mode `'w'`, `'x'` or `'a'`.

Note: The ZIP file standard historically did not specify a metadata encoding, but strongly recommended CP437 (the original IBM PC encoding) for interoperability. Recent versions allow use of UTF-8 (only). In this module, UTF-8 will automatically be used to write the member names if they contain any non-ASCII characters. It is not possible to write member names in any encoding other than ASCII or UTF-8.

Note: Archive names should be relative to the archive root, that is, they should not start with a path separator.

Note: If `arcname` (or `filename`, if `arcname` is not given) contains a null byte, the name of the file in the archive will be truncated at the null byte.

Note: A leading slash in the filename may lead to the archive being impossible to open in some zip programs on Windows systems.

Changed in version 3.6: Calling `write()` on a ZipFile created with mode `'r'` or a closed ZipFile will raise a `ValueError`. Previously, a `RuntimeError` was raised.

ZipFile.**writestr**(*zinfo_or_arcname*, *data*, *compress_type=None*, *compresslevel=None*)

Write a file into the archive. The contents is *data*, which may be either a *str* or a *bytes* instance; if it is a *str*, it is encoded as UTF-8 first. *zinfo_or_arcname* is either the file name it will be given in the archive, or a *ZipInfo* instance. If it's an instance, at least the filename, date, and time must be given. If it's a name, the date and time is set to the current date and time. The archive must be opened with mode `'w'`, `'x'` or `'a'`.

If given, *compress_type* overrides the value given for the *compression* parameter to the constructor for the new entry, or in the *zinfo_or_arcname* (if that is a *ZipInfo* instance). Similarly, *compresslevel* will override the constructor if given.

Note: When passing a *ZipInfo* instance as the *zinfo_or_arcname* parameter, the compression method used will be that specified in the *compress_type* member of the given *ZipInfo* instance. By default, the *ZipInfo* constructor sets this member to *ZIP_STORED*.

Changed in version 3.2: The *compress_type* argument.

Changed in version 3.6: Calling *writestr()* on a ZipFile created with mode `'r'` or a closed ZipFile will raise a *ValueError*. Previously, a *RuntimeError* was raised.

ZipFile.**mkdir**(*zinfo_or_directory*, *mode=511*)

Create a directory inside the archive. If *zinfo_or_directory* is a string, a directory is created inside the archive with the mode that is specified in the *mode* argument. If, however, *zinfo_or_directory* is a *ZipInfo* instance then the *mode* argument is ignored.

The archive must be opened with mode `'w'`, `'x'` or `'a'`.

New in version 3.11.

The following data attributes are also available:

ZipFile.**filename**

Name of the ZIP file.

ZipFile.**debug**

The level of debug output to use. This may be set from `0` (the default, no output) to `3` (the most output). Debugging information is written to `sys.stdout`.

ZipFile.**comment**

The comment associated with the ZIP file as a *bytes* object. If assigning a comment to a *ZipFile* instance created with mode `'w'`, `'x'` or `'a'`, it should be no longer than 65535 bytes. Comments longer than this will be truncated.

13.5.2 Path Objects

class zipfile.**Path**(*root*, *at=''*)

Construct a Path object from a `root` zipfile (which may be a *ZipFile* instance or `file` suitable for passing to the *ZipFile* constructor).

`at` specifies the location of this Path within the zipfile, e.g. 'dir/file.txt', 'dir/', or ''. Defaults to the empty string, indicating the root.

Path objects expose the following features of *pathlib.Path* objects:

Path objects are traversable using the `/` operator or `joinpath`.

Path.**name**

The final path component.

Path.**open**(*mode='r'*, ***, *pwd*, ****)

Invoke *ZipFile.open()* on the current path. Allows opening for read or write, text or binary through supported modes: 'r', 'w', 'rb', 'wb'. Positional and keyword arguments are passed through to *io.TextIOWrapper* when opened as text and ignored otherwise. `pwd` is the `pwd` parameter to *ZipFile.open()*.

Changed in version 3.9: Added support for text and binary modes for open. Default mode is now text.

Changed in version 3.11.2: The `encoding` parameter can be supplied as a positional argument without causing a *TypeError*. As it could in 3.9. Code needing to be compatible with unpatched 3.10 and 3.11 versions must pass all *io.TextIOWrapper* arguments, `encoding` included, as keywords.

Path.**iterdir**()

Enumerate the children of the current directory.

Path.**is_dir**()

Return `True` if the current context references a directory.

Path.**is_file**()

Return `True` if the current context references a file.

Path.**exists**()

Return `True` if the current context references a file or directory in the zip file.

Path.**suffix**

The file extension of the final component.

New in version 3.11: Added *Path.suffix* property.

Path.**stem**

The final path component, without its suffix.

New in version 3.11: Added *Path.stem* property.

Path.**suffixes**

A list of the path's file extensions.

New in version 3.11: Added *Path.suffixes* property.

Path.**read_text**(***, ****)

Read the current file as unicode text. Positional and keyword arguments are passed through to *io.TextIOWrapper* (except `buffer`, which is implied by the context).

Changed in version 3.11.2: The `encoding` parameter can be supplied as a positional argument without causing a *TypeError*. As it could in 3.9. Code needing to be compatible with unpatched 3.10 and 3.11 versions must pass all *io.TextIOWrapper* arguments, `encoding` included, as keywords.

Path.**read_bytes**()

Read the current file as bytes.

Path.**joinpath**(**other*)

Return a new Path object with each of the *other* arguments joined. The following are equivalent:

```
>>> Path(...).joinpath('child').joinpath('grandchild')
>>> Path(...).joinpath('child', 'grandchild')
>>> Path(...) / 'child' / 'grandchild'
```

Changed in version 3.10: Prior to 3.10, `joinpath` was undocumented and accepted exactly one parameter.

The zipp project provides backports of the latest path object functionality to older Pythons. Use `zipp.Path` in place of `zipfile.Path` for early access to changes.

13.5.3 PyZipFile Objects

The *PyZipFile* constructor takes the same parameters as the *ZipFile* constructor, and one additional parameter, *optimize*.

class zipfile.**PyZipFile**(*file*, *mode='r'*, *compression=ZIP_STORED*, *allowZip64=True*, *optimize=-1*)

New in version 3.2: The *optimize* parameter.

Changed in version 3.4: ZIP64 extensions are enabled by default.

Instances have one method in addition to those of *ZipFile* objects:

writepy(*pathname*, *basename=''*, *filterfunc=None*)

Search for files `*.py` and add the corresponding file to the archive.

If the *optimize* parameter to *PyZipFile* was not given or `-1`, the corresponding file is a `*.pyc` file, compiling if necessary.

If the *optimize* parameter to *PyZipFile* was `0`, `1` or `2`, only files with that optimization level (see *compile()*) are added to the archive, compiling if necessary.

If *pathname* is a file, the filename must end with `.py`, and just the (corresponding `*.pyc`) file is added at the top level (no path information). If *pathname* is a file that does not end with `.py`, a *RuntimeError* will be raised. If it is a directory, and the directory is not a package directory, then all the files `*.pyc` are added at the top level. If the directory is a package directory, then all `*.pyc` are added under the package name as a file path, and if any subdirectories are package directories, all of these are added recursively in sorted order.

basename is intended for internal use only.

filterfunc, if given, must be a function taking a single string argument. It will be passed each path (including each individual full file path) before it is added to the archive. If *filterfunc* returns a false value, the path will not be added, and if it is a directory its contents will be ignored. For example, if our test files are all either in `test` directories or start with the string `test_`, we can use a *filterfunc* to exclude them:

```
>>> zf = PyZipFile('myprog.zip')
>>> def notests(s):
...     fn = os.path.basename(s)
...     return (not (fn == 'test' or fn.startswith('test_')))
>>> zf.writepy('myprog', filterfunc=notests)
```

The *writepy()* method makes archives with file names like this:

```
string.pyc                    # Top level name
test/__init__.pyc             # Package directory
test/testall.pyc              # Module test.testall
test/bogus/__init__.pyc       # Subpackage directory
test/bogus/myfile.pyc         # Submodule test.bogus.myfile
```

New in version 3.4: The *filterfunc* parameter.

Changed in version 3.6.2: The *pathname* parameter accepts a *path-like object*.

Changed in version 3.7: Recursion sorts directory entries.

13.5.4 ZipInfo Objects

Instances of the *ZipInfo* class are returned by the *getinfo()* and *infolist()* methods of *ZipFile* objects. Each object stores information about a single member of the ZIP archive.

There is one classmethod to make a *ZipInfo* instance for a filesystem file:

classmethod ZipInfo.**from_file**(*filename*, *arcname=None*, *, *strict_timestamps=True*)

Construct a *ZipInfo* instance for a file on the filesystem, in preparation for adding it to a zip file.

filename should be the path to a file or directory on the filesystem.

If *arcname* is specified, it is used as the name within the archive. If *arcname* is not specified, the name will be the same as *filename*, but with any drive letter and leading path separators removed.

The *strict_timestamps* argument, when set to `False`, allows to zip files older than 1980-01-01 at the cost of setting the timestamp to 1980-01-01. Similar behavior occurs with files newer than 2107-12-31, the timestamp is also set to the limit.

New in version 3.6.

Changed in version 3.6.2: The *filename* parameter accepts a *path-like object*.

New in version 3.8: The *strict_timestamps* keyword-only argument

Instances have the following methods and attributes:

ZipInfo.**is_dir**()

Return `True` if this archive member is a directory.

This uses the entry's name: directories should always end with `/`.

New in version 3.6.

ZipInfo.**filename**

Name of the file in the archive.

ZipInfo.**date_time**

The time and date of the last modification to the archive member. This is a tuple of six values:

Index	Value
0	Year (>= 1980)
1	Month (one-based)
2	Day of month (one-based)
3	Hours (zero-based)
4	Minutes (zero-based)
5	Seconds (zero-based)

Note: The ZIP file format does not support timestamps before 1980.

ZipInfo.**compress_type**

Type of compression for the archive member.

ZipInfo.**comment**

Comment for the individual archive member as a *bytes* object.

ZipInfo.**extra**

Expansion field data. The PKZIP Application Note contains some comments on the internal structure of the data contained in this *bytes* object.

ZipInfo.**create_system**

System which created ZIP archive.

ZipInfo.**create_version**
: PKZIP version which created ZIP archive.

ZipInfo.**extract_version**
: PKZIP version needed to extract archive.

ZipInfo.**reserved**
: Must be zero.

ZipInfo.**flag_bits**
: ZIP flag bits.

ZipInfo.**volume**
: Volume number of file header.

ZipInfo.**internal_attr**
: Internal attributes.

ZipInfo.**external_attr**
: External file attributes.

ZipInfo.**header_offset**
: Byte offset to the file header.

ZipInfo.**CRC**
: CRC-32 of the uncompressed file.

ZipInfo.**compress_size**
: Size of the compressed data.

ZipInfo.**file_size**
: Size of the uncompressed file.

13.5.5 Command-Line Interface

The *zipfile* module provides a simple command-line interface to interact with ZIP archives.

If you want to create a new ZIP archive, specify its name after the *-c* option and then list the filename(s) that should be included:

```
$ python -m zipfile -c monty.zip spam.txt eggs.txt
```

Passing a directory is also acceptable:

```
$ python -m zipfile -c monty.zip life-of-brian_1979/
```

If you want to extract a ZIP archive into the specified directory, use the *-e* option:

```
$ python -m zipfile -e monty.zip target-dir/
```

For a list of the files in a ZIP archive, use the *-l* option:

```
$ python -m zipfile -l monty.zip
```

Command-line options

-l <zipfile>
--list <zipfile>
List files in a zipfile.

-c <zipfile> <source1> ... <sourceN>
--create <zipfile> <source1> ... <sourceN>
Create zipfile from source files.

-e <zipfile> <output_dir>
--extract <zipfile> <output_dir>
Extract zipfile into target directory.

-t <zipfile>
--test <zipfile>
Test whether the zipfile is valid or not.

--metadata-encoding <encoding>
Specify encoding of member names for *-l*, *-e* and *-t*.

New in version 3.11.

13.5.6 Decompression pitfalls

The extraction in zipfile module might fail due to some pitfalls listed below.

From file itself

Decompression may fail due to incorrect password / CRC checksum / ZIP format or unsupported compression method / decryption.

File System limitations

Exceeding limitations on different file systems can cause decompression failed. Such as allowable characters in the directory entries, length of the file name, length of the pathname, size of a single file, and number of files, etc.

Resources limitations

The lack of memory or disk volume would lead to decompression failed. For example, decompression bombs (aka ZIP bomb) apply to zipfile library that can cause disk volume exhaustion.

Interruption

Interruption during the decompression, such as pressing control-C or killing the decompression process may result in incomplete decompression of the archive.

Default behaviors of extraction

Not knowing the default extraction behaviors can cause unexpected decompression results. For example, when extracting the same archive twice, it overwrites files without asking.

13.6 `tarfile` — Read and write tar archive files

Source code: Lib/tarfile.py

The `tarfile` module makes it possible to read and write tar archives, including those using gzip, bz2 and lzma compression. Use the `zipfile` module to read or write `.zip` files, or the higher-level functions in *shutil*.

Some facts and figures:

- reads and writes `gzip`, `bz2` and `lzma` compressed archives if the respective modules are available.
- read/write support for the POSIX.1-1988 (ustar) format.
- read/write support for the GNU tar format including *longname* and *longlink* extensions, read-only support for all variants of the *sparse* extension including restoration of sparse files.
- read/write support for the POSIX.1-2001 (pax) format.
- handles directories, regular files, hardlinks, symbolic links, fifos, character devices and block devices and is able to acquire and restore file information like timestamp, access permissions and owner.

Changed in version 3.3: Added support for `lzma` compression.

`tarfile.`**`open`**(*name=None*, *mode='r'*, *fileobj=None*, *bufsize=10240*, ***kwargs*)

Return a `TarFile` object for the pathname *name*. For detailed information on `TarFile` objects and the keyword arguments that are allowed, see *TarFile Objects*.

mode has to be a string of the form `'filemode[:compression]'`, it defaults to `'r'`. Here is a full list of mode combinations:

mode	action
`'r'` or `'r:*'`	Open for reading with transparent compression (recommended).
`'r:'`	Open for reading exclusively without compression.
`'r:gz'`	Open for reading with gzip compression.
`'r:bz2'`	Open for reading with bzip2 compression.
`'r:xz'`	Open for reading with lzma compression.
`'x'` or `'x:'`	Create a tarfile exclusively without compression. Raise a `FileExistsError` exception if it already exists.
`'x:gz'`	Create a tarfile with gzip compression. Raise a `FileExistsError` exception if it already exists.
`'x:bz2'`	Create a tarfile with bzip2 compression. Raise a `FileExistsError` exception if it already exists.
`'x:xz'`	Create a tarfile with lzma compression. Raise a `FileExistsError` exception if it already exists.
`'a'` or `'a:'`	Open for appending with no compression. The file is created if it does not exist.
`'w'` or `'w:'`	Open for uncompressed writing.
`'w:gz'`	Open for gzip compressed writing.
`'w:bz2'`	Open for bzip2 compressed writing.
`'w:xz'`	Open for lzma compressed writing.