MemoryView objects

A `memoryview` object exposes the C level buffer interface as a Python object which can then be passed around like any other object.

PyObject *__PyMemoryView_FromObject__(PyObject *obj)

Create a memoryview object from an object that provides the buffer interface. If *obj* supports writable buffer exports, the memoryview object will be read/write, otherwise it may be either read-only or read/write at the discretion of the exporter.

PyObject *__PyMemoryView_FromMemory__(char *mem, Py_ssize_t size, int flags)

Create a memoryview object using *mem* as the underlying buffer. *flags* can be one of `PyBUF_READ` or `PyBUF_WRITE`.

New in version 3.3.

PyObject *__PyMemoryView_FromBuffer__(Py_buffer *view)

Create a memoryview object wrapping the given buffer structure *view*. For simple byte buffers, `PyMemoryView_FromMemory()` is the preferred function.

PyObject *__PyMemoryView_GetContiguous__(PyObject *obj, int buffertype, char order)

Create a memoryview object to a contiguous chunk of memory (in either 'C' or 'F'ortran *order*) from an object that defines the buffer interface. If memory is contiguous, the memoryview object points to the original memory. Otherwise, a copy is made and the memoryview points to a new bytes object.

int __PyMemoryView_Check__(PyObject *obj)

Return true if the object *obj* is a memoryview object. It is not currently allowed to create subclasses of `memoryview`.

Py_buffer *__PyMemoryView_GET_BUFFER__(PyObject *mview)

Return a pointer to the memoryview's private copy of the exporter's buffer. *mview* **must** be a memoryview instance; this macro doesn't check its type, you must do it yourself or you will risk crashes.

Py_buffer *__PyMemoryView_GET_BASE__(PyObject *mview)

Return either a pointer to the exporting object that the memoryview is based on or *NULL* if the memoryview has been created by one of the functions `PyMemoryView_FromMemory()` or `PyMemoryView_FromBuffer()`. *mview* **must** be a memoryview instance.