# Sequence Protocol

int **PySequence_Check**(PyObject *o)

Return `1` if the object provides sequence protocol, and `0` otherwise. Note that it returns `1` for Python classes with a `__getitem__()` method unless they are `dict` subclasses since in general case it is impossible to determine what the type of keys it supports. This function always succeeds.

Py_ssize_t **PySequence_Size**(PyObject *o)
Py_ssize_t **PySequence_Length**(PyObject *o)

Returns the number of objects in sequence *o* on success, and `-1` on failure. This is equivalent to the Python expression `len(o)`.

PyObject* **PySequence_Concat**(PyObject *o1, PyObject *o2)

*Return value: New reference.*

Return the concatenation of *o1* and *o2* on success, and *NULL* on failure. This is the equivalent of the Python expression `o1 + o2`.

PyObject* **PySequence_Repeat**(PyObject *o, Py_ssize_t *count*)

*Return value: New reference.*

Return the result of repeating sequence object *o* *count* times, or *NULL* on failure. This is the equivalent of the Python expression `o * count`.

PyObject* **PySequence_InPlaceConcat**(PyObject *o1, PyObject *o2)

*Return value: New reference.*

Return the concatenation of *o1* and *o2* on success, and *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python expression `o1 += o2`.

PyObject* **PySequence_InPlaceRepeat**(PyObject *o, Py_ssize_t *count*)

*Return value: New reference.*

Return the result of repeating sequence object *o* *count* times, or *NULL* on failure. The operation is done *in-place* when *o* supports it. This is the equivalent of the Python expression `o *= count`.

PyObject* **PySequence_GetItem**(PyObject *o, Py_ssize_t *i*)

*Return value: New reference.*

Return the *i*th element of *o*, or *NULL* on failure. This is the equivalent of the Python expression `o[i]`.

PyObject* **PySequence_GetSlice**(PyObject *o, Py_ssize_t *i1*, Py_ssize_t *i2*)

*Return value: New reference.*

Return the slice of sequence object *o* between *i1* and *i2*, or *NULL* on failure. This is the equivalent of the Python expression `o[i1:i2]`.

int **PySequence_SetItem**(PyObject **o*, Py_ssize_t *i*, PyObject **v*)

Assign object *v* to the *i*th element of *o*. Raise an exception and return `-1` on failure; return `0` on success. This is the equivalent of the Python statement `o[i] = v`. This function *does not* steal a reference to *v*.

If *v* is *NULL*, the element is deleted, however this feature is deprecated in favour of using `PySequence_DelItem()`.

int **PySequence_DelItem**(PyObject **o*, Py_ssize_t *i*)

Delete the *i*th element of object *o*. Returns `-1` on failure. This is the equivalent of the Python statement `del o[i]`.

int **PySequence_SetSlice**(PyObject **o*, Py_ssize_t *i1*, Py_ssize_t *i2*, PyObject **v*)

Assign the sequence object *v* to the slice in sequence object *o* from *i1* to *i2*. This is the equivalent of the Python statement `o[i1:i2] = v`.

int **PySequence_DelSlice**(PyObject **o*, Py_ssize_t *i1*, Py_ssize_t *i2*)

Delete the slice in sequence object *o* from *i1* to *i2*. Returns `-1` on failure. This is the equivalent of the Python statement `del o[i1:i2]`.

Py_ssize_t **PySequence_Count**(PyObject **o*, PyObject **value*)

Return the number of occurrences of *value* in *o*, that is, return the number of keys for which `o[key] == value`. On failure, return `-1`. This is equivalent to the Python expression `o.count(value)`.

int **PySequence_Contains**(PyObject **o*, PyObject **value*)

Determine if *o* contains *value*. If an item in *o* is equal to *value*, return `1`, otherwise return `0`. On error, return `-1`. This is equivalent to the Python expression `value in o`.

Py_ssize_t **PySequence_Index**(PyObject **o*, PyObject **value*)

Return the first index *i* for which `o[i] == value`. On error, return `-1`. This is equivalent to the Python expression `o.index(value)`.

PyObject* **PySequence_List**(PyObject **o*)

*Return value: New reference.*

Return a list object with the same contents as the sequence or iterable *o*, or *NULL* on failure. The returned list is guaranteed to be new. This is equivalent to the Python expression `list(o)`.

PyObject* **PySequence_Tuple**(PyObject *o)

*Return value: New reference.*

Return a tuple object with the same contents as the sequence or iterable *o*, or *NULL* on failure. If *o* is a tuple, a new reference will be returned, otherwise a tuple will be constructed with the appropriate contents. This is equivalent to the Python expression `tuple(o)`.

PyObject* **PySequence_Fast**(PyObject *o, const char *m)

*Return value: New reference.*

Return the sequence or iterable *o* as a list, unless it is already a tuple or list, in which case *o* is returned. Use `PySequence_Fast_GET_ITEM()` to access the members of the result. Returns *NULL* on failure. If the object is not a sequence or iterable, raises `TypeError` with *m* as the message text.

Py_ssize_t **PySequence_Fast_GET_SIZE**(PyObject *o)

Returns the length of *o*, assuming that *o* was returned by `PySequence_Fast()` and that *o* is not *NULL*. The size can also be gotten by calling `PySequence_Size()` on *o*, but `PySequence_Fast_GET_SIZE()` is faster because it can assume *o* is a list or tuple.

PyObject* **PySequence_Fast_GET_ITEM**(PyObject *o, Py_ssize_t *i*)

*Return value: Borrowed reference.*

Return the *i*th element of *o*, assuming that *o* was returned by `PySequence_Fast()`, *o* is not *NULL*, and that *i* is within bounds.

PyObject** **PySequence_Fast_ITEMS**(PyObject *o)

Return the underlying array of PyObject pointers. Assumes that *o* was returned by `PySequence_Fast()` and *o* is not *NULL*.

Note, if a list gets resized, the reallocation may relocate the items array. So, only use the underlying array pointer in contexts where the sequence cannot change.

PyObject* **PySequence_ITEM**(PyObject *o, Py_ssize_t *i*)

*Return value: New reference.*

Return the *i*th element of *o* or *NULL* on failure. Macro form of `PySequence_GetItem()` but without checking that `PySequence_Check()` on *o* is true and without adjustment for negative indices.