# List Objects

**`PyListObject`**

This subtype of `PyObject` represents a Python list object.

PyTypeObject **`PyList_Type`**

This instance of `PyTypeObject` represents the Python list type. This is the same object as `list` in the Python layer.

int **`PyList_Check`**(PyObject **p*)

Return true if *p* is a list object or an instance of a subtype of the list type.

int **`PyList_CheckExact`**(PyObject **p*)

Return true if *p* is a list object, but not an instance of a subtype of the list type.

PyObject* **`PyList_New`**(Py_ssize_t *len*)

*Return value: New reference.*

Return a new list of length *len* on success, or *NULL* on failure.

> **Note:** If *len* is greater than zero, the returned list object's items are set to `NULL`. Thus you cannot use abstract API functions such as `PySequence_SetItem()` or expose the object to Python code before setting all items to a real object with `PyList_SetItem()`.

Py_ssize_t **`PyList_Size`**(PyObject **list*)

Return the length of the list object in *list*; this is equivalent to `len(list)` on a list object.

Py_ssize_t **`PyList_GET_SIZE`**(PyObject **list*)

Macro form of `PyList_Size()` without error checking.

PyObject* **`PyList_GetItem`**(PyObject **list*, Py_ssize_t *index*)

*Return value: Borrowed reference.*

Return the object at position *index* in the list pointed to by *list*. The position must be positive, indexing from the end of the list is not supported. If *index* is out of bounds, return *NULL* and set an `IndexError` exception.

PyObject* **`PyList_GET_ITEM`**(PyObject **list*, Py_ssize_t *i*)

*Return value: Borrowed reference.*

Macro form of `PyList_GetItem()` without error checking.

int **`PyList_SetItem`**(PyObject **list*, Py_ssize_t *index*, PyObject **item*)

Set the item at index *index* in list to *item*. Return `0` on success or `-1` on failure.

> **Note:** This function "steals" a reference to *item* and discards a reference to an item already in the list at the affected position.

void **`PyList_SET_ITEM`**(PyObject **list*, Py_ssize_t *i*, PyObject **o*)

Macro form of `PyList_SetItem()` without error checking. This is normally only used to fill in new lists where there is no previous content.

> **Note:** This macro "steals" a reference to *item*, and, unlike `PyList_SetItem()`, does *not* discard a reference to any item that is being replaced; any reference in *list* at position *i* will be leaked.

int **`PyList_Insert`**(PyObject **list*, Py_ssize_t *index*, PyObject **item*)

Insert the item *item* into list *list* in front of index *index*. Return `0` if successful; return `-1` and set an exception if unsuccessful. Analogous to `list.insert(index, item)`.

int **`PyList_Append`**(PyObject **list*, PyObject **item*)

Append the object *item* at the end of list *list*. Return `0` if successful; return `-1` and set an exception if unsuccessful. Analogous to `list.append(item)`.

PyObject* **`PyList_GetSlice`**(PyObject **list*, Py_ssize_t *low*, Py_ssize_t *high*)

*Return value: New reference.*

Return a list of the objects in *list* containing the objects *between low* and *high*. Return *NULL* and set an exception if unsuccessful. Analogous to `list[low:high]`. Negative indices, as when slicing from Python, are not supported.

int **`PyList_SetSlice`**(PyObject **list*, Py_ssize_t *low*, Py_ssize_t *high*, PyObject **itemlist*)

Set the slice of *list* between *low* and *high* to the contents of *itemlist*. Analogous to `list[low:high] = itemlist`. The *itemlist* may be *NULL*, indicating the assignment of an empty list (slice deletion). Return `0` on success, `-1` on failure. Negative indices, as when slicing from Python, are not supported.

int **`PyList_Sort`**(PyObject **list*)

Sort the items of *list* in place. Return `0` on success, `-1` on failure. This is equivalent to `list.sort()`.

int **`PyList_Reverse`**(PyObject **list*)

Reverse the items of *list* in place. Return `0` on success, `-1` on failure. This is the equivalent of `list.reverse()`.

PyObject* **`PyList_AsTuple`**(PyObject **list*)

*Return value: New reference.*

Return a new tuple object containing the contents of *list*; equivalent to `tuple(list)`.

int **`PyList_ClearFreeList`**()

Clear the free list. Return the total number of freed items.

*New in version 3.3.*