Integer Objects

All integers are implemented as “long” integer objects of arbitrary size.

On error, most `PyLong_As*` APIs return `(return type)-1` which cannot be distinguished from a number. Use `PyErr_Occurred()` to disambiguate.

`PyLongObject`

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

PyTypeObject **`PyLong_Type`**

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

int **`PyLong_Check`**(PyObject **p*)

Return true if its argument is a `PyLongObject` or a subtype of `PyLongObject`.

int **`PyLong_CheckExact`**(PyObject **p*)

Return true if its argument is a `PyLongObject`, but not a subtype of `PyLongObject`.

PyObject* **`PyLong_FromLong`**(long *v*)

Return value: New reference.

Return a new `PyLongObject` object from *v*, or *NULL* on failure.

The current implementation keeps an array of integer objects for all integers between `-5` and `256`, when you create an int in that range you actually just get back a reference to the existing object. So it should be possible to change the value of `1`. I suspect the behaviour of Python in this case is undefined. :-)

PyObject* **`PyLong_FromUnsignedLong`**(unsigned long *v*)

Return value: New reference.

Return a new `PyLongObject` object from a C `unsigned long`, or *NULL* on failure.

PyObject* **`PyLong_FromSsize_t`**(Py_ssize_t *v*)

Return a new `PyLongObject` object from a C `Py_ssize_t`, or *NULL* on failure.

PyObject* **`PyLong_FromSize_t`**(size_t *v*)

Return a new `PyLongObject` object from a C `size_t`, or *NULL* on failure.

PyObject* **`PyLong_FromLongLong`**(long long *v*)

Return value: New reference.

Return a new `PyLongObject` object from a C `long long`, or *NULL* on failure.

PyObject* **`PyLong_FromUnsignedLongLong`**(unsigned long long *v*)

Return value: New reference.

Return a new `PyLongObject` object from a C `unsigned long long`, or *NULL* on failure.

PyObject* **`PyLong_FromDouble`**(double *v*)

Return value: New reference.

Return a new `PyLongObject` object from the integer part of *v*, or *NULL* on failure.

PyObject* **`PyLong_FromString`**(const char **str*, char ***pend*, int *base*)

Return value: New reference.

Return a new `PyLongObject` based on the string value in *str*, which is interpreted according to the radix in *base*. If *pend* is non-*NULL*, **pend* will point to the first character in *str* which follows the representation of the number. If *base* is `0`, *str* is interpreted using the Integer literals definition; in this case, leading zeros in a non-zero decimal number raises a `ValueError`. If *base* is not `0`, it must be between `2` and `36`, inclusive. Leading spaces and single underscores after a base specifier and between digits are ignored. If there are no digits, `ValueError` will be raised.

PyObject* **`PyLong_FromUnicode`**(Py_UNICODE **u*, Py_ssize_t *length*, int *base*)

Return value: New reference.

Convert a sequence of Unicode digits to a Python integer value. The Unicode string is first encoded to a byte string using `PyUnicode_EncodeDecimal()` and then converted using `PyLong_FromString()`.

Deprecated since version 3.3, will be removed in version 4.0: Part of the old-style `Py_UNICODE` API; please migrate to using `PyLong_FromUnicodeObject()`.

PyObject* **`PyLong_FromUnicodeObject`**(PyObject **u*, int *base*)

Convert a sequence of Unicode digits in the string *u* to a Python integer value. The Unicode string is first encoded to a byte string using `PyUnicode_EncodeDecimal()` and then converted using `PyLong_FromString()`.

New in version 3.3.

PyObject* **`PyLong_FromVoidPtr`**(void **p*)

Return value: New reference.

Create a Python integer from the pointer *p*. The pointer value can be retrieved from the resulting value using `PyLong_AsVoidPtr()`.

long **`PyLong_AsLong`**(PyObject **obj*)

Return a C `long` representation of *obj*. If *obj* is not an instance of `PyLongObject`, first call its `__int__()` method (if present) to convert it to a `PyLongObject`.

Raise `OverflowError` if the value of *obj* is out of range for a `long`.

Returns `-1` on error. Use `PyErr_Occurred()` to disambiguate.

long **`PyLong_AsLongAndOverflow`**(PyObject **obj*, int **overflow*)

Return a C `long` representation of *obj*. If *obj* is not an instance of `PyLongObject`, first call its `__int__()` method (if present) to convert it to a `PyLongObject`.

If the value of *obj* is greater than `LONG_MAX` or less than `LONG_MIN`, set **overflow* to `1` or `-1`, respectively, and return `-1`; otherwise, set **overflow* to `0`. If any other exception occurs set **overflow* to `0` and return `-1` as usual.

Returns `-1` on error. Use `PyErr_Occurred()` to disambiguate.

long long **`PyLong_AsLongLong`**(PyObject **obj*)

Return a C `long long` representation of *obj*. If *obj* is not an instance of `PyLongObject`, first call its `__int__()` method (if present) to convert it to a `PyLongObject`.

Raise `OverflowError` if the value of *obj* is out of range for a `long`.

Returns `-1` on error. Use `PyErr_Occurred()` to disambiguate.

long long **`PyLong_AsLongLongAndOverflow`**(PyObject **obj*, int **overflow*)

Return a C `long long` representation of *obj*. If *obj* is not an instance of `PyLongObject`, first call its `__int__()` method (if present) to convert it to a `PyLongObject`.

If the value of *obj* is greater than `PY_LLONG_MAX` or less than `PY_LLONG_MIN`, set **overflow* to `1` or `-1`, respectively, and return `-1`; otherwise, set **overflow* to `0`. If any other exception occurs set **overflow* to `0` and return `-1` as usual.

Returns `-1` on error. Use `PyErr_Occurred()` to disambiguate.

New in version 3.2.

Py_ssize_t **`PyLong_AsSsize_t`**(PyObject **pylong*)

Return a C `Py_ssize_t` representation of *pylong*. *pylong* must be an instance of `PyLongObject`.

Raise `OverflowError` if the value of *pylong* is out of range for a `Py_ssize_t`.

Returns `-1` on error. Use `PyErr_Occurred()` to disambiguate.

unsigned long **`PyLong_AsUnsignedLong`**(PyObject **pylong*)

Return a C `unsigned long` representation of *pylong*. *pylong* must be an instance of `PyLongObject`.

Raise `OverflowError` if the value of *pylong* is out of range for a `unsigned long`.

Returns `(unsigned long)-1` on error. Use `PyErr_Occurred()` to disambiguate.

size_t **`PyLong_AsSize_t`**(PyObject **pylong*)

Return a C `size_t` representation of *pylong*. *pylong* must be an instance of `PyLongObject`.

Raise `OverflowError` if the value of *pylong* is out of range for a `size_t`.

Returns `(size_t)-1` on error. Use `PyErr_Occurred()` to disambiguate.

unsigned long long **`PyLong_AsUnsignedLongLong`**(PyObject **pylong*)

Return a C `unsigned long long` representation of *pylong*. *pylong* must be an instance of `PyLongObject`.

Raise `OverflowError` if the value of *pylong* is out of range for an `unsigned long long`.

Returns `(unsigned long long)-1` on error. Use `PyErr_Occurred()` to disambiguate.

Changed in version 3.1: A negative *pylong* now raises `OverflowError`, not `TypeError`.

unsigned long **`PyLong_AsUnsignedLongMask`**(PyObject **obj*)

Return a C `unsigned long` representation of *obj*. If *obj* is not an instance of `PyLongObject`, first call its `__int__()` method (if present) to convert it to a `PyLongObject`.

If the value of *obj* is out of range for an `unsigned long`, return the reduction of that value modulo `ULONG_MAX + 1`.

Returns `-1` on error. Use `PyErr_Occurred()` to disambiguate.

unsigned long long **`PyLong_AsUnsignedLongLongMask`**(PyObject **obj*)

Return a C `unsigned long long` representation of *obj*. If *obj* is not an instance of `PyLongObject`, first call its `__int__()` method (if present) to convert it to a `PyLongObject`.

If the value of *obj* is out of range for an `unsigned long long`, return the reduction of that value modulo `PY_ULLONG_MAX + 1`.

Returns `-1` on error. Use `PyErr_Occurred()` to disambiguate.

double **`PyLong_AsDouble`**(PyObject **pylong*)

Return a C `double` representation of *pylong*. *pylong* must be an instance of `PyLongObject`.

Raise `OverflowError` if the value of *pylong* is out of range for a `double`.

Returns `-1.0` on error. Use `PyErr_Occurred()` to disambiguate.

void* **`PyLong_AsVoidPtr`**(PyObject **pylong*)

Convert a Python integer *pylong* to a C `void` pointer. If *pylong* cannot be converted, an `OverflowError` will be raised. This is only assured to produce a usable `void` pointer for values created with `PyLong_FromVoidPtr()`.

Returns *NULL* on error. Use `PyErr_Occurred()` to disambiguate.