Number Protocol

int **`PyNumber_Check`**(PyObject **o*)

Returns `1` if the object *o* provides numeric protocols, and false otherwise. This function always succeeds.

PyObject* **`PyNumber_Add`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the result of adding *o1* and *o2*, or *NULL* on failure. This is the equivalent of the Python expression `o1 + o2`.

PyObject* **`PyNumber_Subtract`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the result of subtracting *o2* from *o1*, or *NULL* on failure. This is the equivalent of the Python expression `o1 - o2`.

PyObject* **`PyNumber_Multiply`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the result of multiplying *o1* and *o2*, or *NULL* on failure. This is the equivalent of the Python expression `o1 * o2`.

PyObject* **`PyNumber_MatrixMultiply`**(PyObject **o1*, PyObject **o2*)

Returns the result of matrix multiplication on *o1* and *o2*, or *NULL* on failure. This is the equivalent of the Python expression `o1 @ o2`.

New in version 3.5.

PyObject* **`PyNumber_FloorDivide`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Return the floor of *o1* divided by *o2*, or *NULL* on failure. This is equivalent to the "classic" division of integers.

PyObject* **`PyNumber_TrueDivide`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Return a reasonable approximation for the mathematical value of *o1* divided by *o2*, or *NULL* on failure. The return value is "approximate" because binary floating point numbers are approximate; it is not possible to represent all real numbers in base two. This function can return a floating point value when passed two integers.

PyObject* **`PyNumber_Remainder`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the remainder of dividing *o1* by *o2*, or *NULL* on failure. This is the equivalent of the Python expression `o1 % o2`.

PyObject* **`PyNumber_Divmod`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

See the built-in function `divmod()`. Returns *NULL* on failure. This is the equivalent of the Python expression `divmod(o1, o2)`.

PyObject* **`PyNumber_Power`**(PyObject **o1*, PyObject **o2*, PyObject **o3*)

Return value: New reference.

See the built-in function `pow()`. Returns *NULL* on failure. This is the equivalent of the Python expression `pow(o1, o2, o3)`, where *o3* is optional. If *o3* is to be ignored, pass `Py_None` in its place (passing *NULL* for *o3* would cause an illegal memory access).

PyObject* **`PyNumber_Negative`**(PyObject **o*)

Return value: New reference.

Returns the negation of *o* on success, or *NULL* on failure. This is the equivalent of the Python expression `-o`.

PyObject* **`PyNumber_Positive`**(PyObject **o*)

Return value: New reference.

Returns *o* on success, or *NULL* on failure. This is the equivalent of the Python expression `+o`.

PyObject* **`PyNumber_Absolute`**(PyObject **o*)

Return value: New reference.

Returns the absolute value of *o*, or *NULL* on failure. This is the equivalent of the Python expression `abs(o)`.

PyObject* **`PyNumber_Invert`**(PyObject **o*)

Return value: New reference.

Returns the bitwise negation of *o* on success, or *NULL* on failure. This is the equivalent of the Python expression `~o`.

PyObject* **`PyNumber_Lshift`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the result of left shifting *o1* by *o2* on success, or *NULL* on failure. This is the equivalent of the Python expression `o1 << o2`.

PyObject* **`PyNumber_Rshift`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the result of right shifting *o1* by *o2* on success, or *NULL* on failure. This is the equivalent of the Python expression `o1 >> o2`.

PyObject* **PyNumber_And**(PyObject *o1*, PyObject *o2*)

Return value: New reference.

Returns the "bitwise and" of *o1* and *o2* on success and *NULL* on failure. This is the equivalent of the Python expression `o1 & o2`.

PyObject* **PyNumber_Xor**(PyObject *o1*, PyObject *o2*)

Return value: New reference.

Returns the "bitwise exclusive or" of *o1* by *o2* on success, or *NULL* on failure. This is the equivalent of the Python expression `o1 ^ o2`.

PyObject* **PyNumber_Or**(PyObject *o1*, PyObject *o2*)

Return value: New reference.

Returns the "bitwise or" of *o1* and *o2* on success, or *NULL* on failure. This is the equivalent of the Python expression `o1 | o2`.

PyObject* **PyNumber_InPlaceAdd**(PyObject *o1*, PyObject *o2*)

Return value: New reference.

Returns the result of adding *o1* and *o2*, or *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 += o2`.

PyObject* **PyNumber_InPlaceSubtract**(PyObject *o1*, PyObject *o2*)

Return value: New reference.

Returns the result of subtracting *o2* from *o1*, or *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 -= o2`.

PyObject* **PyNumber_InPlaceMultiply**(PyObject *o1*, PyObject *o2*)

Return value: New reference.

Returns the result of multiplying *o1* and *o2*, or *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 *= o2`.

PyObject* **PyNumber_InPlaceMatrixMultiply**(PyObject *o1*, PyObject *o2*)

Returns the result of matrix multiplication on *o1* and *o2*, or *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 @= o2`.

New in version 3.5.

PyObject* **PyNumber_InPlaceFloorDivide**(PyObject *o1*, PyObject *o2*)

Return value: New reference.

Returns the mathematical floor of dividing *o1* by *o2*, or *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 //= o2`.

PyObject* **`PyNumber_InPlaceTrueDivide`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Return a reasonable approximation for the mathematical value of *o1* divided by *o2*, or *NULL* on failure. The return value is "approximate" because binary floating point numbers are approximate; it is not possible to represent all real numbers in base two. This function can return a floating point value when passed two integers. The operation is done *in-place* when *o1* supports it.

PyObject* **`PyNumber_InPlaceRemainder`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the remainder of dividing *o1* by *o2*, or *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 %= o2`.

PyObject* **`PyNumber_InPlacePower`**(PyObject **o1*, PyObject **o2*, PyObject **o3*)

Return value: New reference.

See the built-in function `pow()`. Returns *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 **= o2` when o3 is `Py_None`, or an in-place variant of `pow(o1, o2, o3)` otherwise. If *o3* is to be ignored, pass `Py_None` in its place (passing *NULL* for *o3* would cause an illegal memory access).

PyObject* **`PyNumber_InPlaceLshift`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the result of left shifting *o1* by *o2* on success, or *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 <<= o2`.

PyObject* **`PyNumber_InPlaceRshift`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the result of right shifting *o1* by *o2* on success, or *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 >>= o2`.

PyObject* **`PyNumber_InPlaceAnd`**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the "bitwise and" 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 statement `o1 &= o2`.

PyObject* **PyNumber_InPlaceXor**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the “bitwise exclusive or” of *o1* by *o2* on success, or *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 ^= o2`.

PyObject* **PyNumber_InPlaceOr**(PyObject **o1*, PyObject **o2*)

Return value: New reference.

Returns the “bitwise or” of *o1* and *o2* on success, or *NULL* on failure. The operation is done *in-place* when *o1* supports it. This is the equivalent of the Python statement `o1 |= o2`.

PyObject* **PyNumber_Long**(PyObject **o*)

Return value: New reference.

Returns the *o* converted to an integer object on success, or *NULL* on failure. This is the equivalent of the Python expression `int(o)`.

PyObject* **PyNumber_Float**(PyObject **o*)

Return value: New reference.

Returns the *o* converted to a float object on success, or *NULL* on failure. This is the equivalent of the Python expression `float(o)`.

PyObject* **PyNumber_Index**(PyObject **o*)

Returns the *o* converted to a Python int on success or *NULL* with a `TypeError` exception raised on failure.

PyObject* **PyNumber_ToBase**(PyObject **n*, int *base*)

Returns the integer *n* converted to base *base* as a string. The *base* argument must be one of 2, 8, 10, or 16. For base 2, 8, or 16, the returned string is prefixed with a base marker of `'0b'`, `'0o'`, or `'0x'`, respectively. If *n* is not a Python int, it is converted with `PyNumber_Index()` first.

Py_ssize_t **PyNumber_AsSsize_t**(PyObject **o*, PyObject **exc*)

Returns *o* converted to a Py_ssize_t value if *o* can be interpreted as an integer. If the call fails, an exception is raised and `-1` is returned.

If *o* can be converted to a Python int but the attempt to convert to a Py_ssize_t value would raise an `OverflowError`, then the *exc* argument is the type of exception that will be raised (usually `IndexError` or `OverflowError`). If *exc* is *NULL*, then the exception is cleared and the value is clipped to *PY_SSIZE_T_MIN* for a negative integer or *PY_SSIZE_T_MAX* for a positive integer.

int **PyIndex_Check**(PyObject **o*)

Returns `1` if *o* is an index integer (has the nb_index slot of the tp_as_number structure filled in), and `0` otherwise.