Floating-Point Objects

- Pack and Unpack functions
  - Pack functions
  - Unpack functions

pair: object; floating-point

> This subtype of `PyObject` represents a Python floating-point object.

> This instance of `PyTypeObject` represents the Python floating-point type. This is the same object as `float` in the Python layer.

> Return true if its argument is a `PyFloatObject` or a subtype of `PyFloatObject`. This function always succeeds.

> Return true if its argument is a `PyFloatObject`, but not a subtype of `PyFloatObject`. This function always succeeds.

> Create a `PyFloatObject` object based on the string value in *str*, or `NULL` on failure.

> Create a `PyFloatObject` object from *v*, or `NULL` on failure.

Return a C `double` representation of the contents of *pyfloat*. If *pyfloat* is not a Python floating-point object but has a `~object.__float__` method, this method will first be called to convert *pyfloat* into a float. If `!__float__` is not defined then it falls back to `~object.__index__`. This method returns `-1.0` upon failure, so one should call `PyErr_Occurred` to check for errors.

3.8 Use `~object.__index__` if available.

> Return a C `double` representation of the contents of *pyfloat*, but without error checking.

> Return a structseq instance which contains information about the precision, minimum and maximum values of a float. It's a thin wrapper around the header file `float.h`.

> Return the maximum representable finite float *DBL_MAX* as C `double`.

> Return the minimum normalized positive float *DBL_MIN* as C `double`.

> This macro expands to a constant expression of type `double`, that represents the positive infinity.
>
> It is equivalent to the `!INFINITY` macro from the C11 standard `<math.h>` header.
>
> 3.15

> This macro expands to a constant expression of type `double`, that represents a quiet not-a-number (qNaN) value.
>
> On most platforms, this is equivalent to the `!NAN` macro from the C11 standard `<math.h>` header.

> Equivalent to `!INFINITY`.
>
> 3.14

> The definition (accurate for a `double` type) of the `math.e` constant.

> High precision (long double) definition of `~math.e` constant.
>
> 3.15 3.20

> The definition (accurate for a `double` type) of the `math.pi` constant.

> High precision (long double) definition of `~math.pi` constant.
>
> 3.15 3.20

> The definition (accurate for a `double` type) of the `math.tau` constant.
>
> 3.6

> Return `math.nan` from a function.
>
> On most platforms, this is equivalent to `return PyFloat_FromDouble(NAN)`.

> Return `math.inf` or `-math.inf <math.inf>` from a function, depending on the sign of *sign*.
>
> On most platforms, this is equivalent to the following:

```
return PyFloat_FromDouble(copysign(INFINITY, sign));
```

> Return `1` if the given floating-point number *X* is finite, that is, it is normal, subnormal or zero, but not infinite or NaN. Return `0` otherwise.
>
> 3.14 Use `!isfinite` instead.

> Return `1` if the given floating-point number *X* is positive or negative infinity. Return `0` otherwise.
>
> 3.14 Use `!isinf` instead.

> Return `1` if the given floating-point number *X* is a not-a-number (NaN) value. Return `0` otherwise.
>
> 3.14 Use `!isnan` instead.

Pack and Unpack functions

The pack and unpack functions provide an efficient platform-independent way to store floating-point values as byte strings. The Pack routines produce a bytes string from a C `double`, and the Unpack routines produce a C `double` from such a bytes string. The suffix (2, 4 or 8) specifies the number of bytes in the bytes string:

- The 2-byte format is the IEEE 754 binary16 half-precision format.
- The 4-byte format is the IEEE 754 binary32 single-precision format.
- The 8-byte format is the IEEE 754 binary64 double-precision format.

The NaN type may not be preserved on some platforms while unpacking (signaling NaNs become quiet NaNs), for example on x86 systems in 32-bit mode.

It's assumed that the `double` type has the IEEE 754 binary64 double precision format. What happens if it's not true is partly accidental (alas). On non-IEEE platforms with more precision, or larger dynamic range, than IEEE 754 supports, not all values can be packed; on non-IEEE platforms with less precision, or smaller dynamic range, not all values can be unpacked. The packing of

special numbers like INFs and NaNs (if such things exist on the platform) may not be handled correctly, and attempting to unpack a bytes string containing an IEEE INF or NaN may raise an exception.

3.11

Pack functions

The pack routines write 2, 4 or 8 bytes, starting at *p*. *le* is an `int` argument, non-zero if you want the bytes string in little-endian format (exponent last, at `p+1`, `p+3`, or `p+6` and `p+7`), zero if you want big-endian format (exponent first, at *p*). Use the `PY_LITTLE_ENDIAN` constant to select the native endian: it is equal to `0` on big endian processor, or `1` on little endian processor.

Return value: `0` if all is OK, `-1` if error (and an exception is set, most likely `OverflowError`).

> Pack a C double as the IEEE 754 binary16 half-precision format.

> Pack a C double as the IEEE 754 binary32 single precision format.

> Pack a C double as the IEEE 754 binary64 double precision format.

Unpack functions

The unpack routines read 2, 4 or 8 bytes, starting at *p*. *le* is an `int` argument, non-zero if the bytes string is in little-endian format (exponent last, at `p+1`, `p+3` or `p+6` and `p+7`), zero if big-endian (exponent first, at *p*). Use the `PY_LITTLE_ENDIAN` constant to select the native endian: it is equal to `0` on big endian processor, or `1` on little endian processor.

Return value: The unpacked double. On error, this is `-1.0` and `PyErr_Occurred` is true (and an exception is set, most likely `OverflowError`).

> Unpack the IEEE 754 binary16 half-precision format as a C double.

>

> Unpack the IEEE 754 binary32 single precision format as a C double.

> “ Unpack the IEEE 754 binary64 double precision format as a C double.

□□□ #2
□ □□□ □□□ 2026-10-31 16:51:12 UTC
□ □□□ □□□ 2026-10-31 16:51:14 UTC