Function Objects

pair: object; function

There are a few functions specific to Python functions.

> The C structure used for functions.
>
> In the free-threaded build, direct access to mutable fields requires synchronization if another thread may modify the function. See `freethreading-extensions-howto`.

> single: MethodType (in module types)
>
> This is an instance of `PyTypeObject` and represents the Python function type. It is exposed to Python programmers as `types.FunctionType`.

> Return true if *o* is a function object (has type `PyFunction_Type`). The parameter must not be `NULL`. This function always succeeds.

> Return a new function object associated with the code object *code*. *globals* must be a dictionary with the global variables accessible to the function.
>
> The function's docstring and name are retrieved from the code object. `~function.__module__` is retrieved from *globals*. The argument defaults, annotations and closure are set to `NULL`. `~function.__qualname__` is set to the same value as the code object's `~codeobject.co_qualname` field.

> As `PyFunction_New`, but also allows setting the function object's `~function.__qualname__` attribute. *qualname* should be a unicode object or `NULL`; if `NULL`, the `!__qualname__` attribute is set to the same value as the code object's `~codeobject.co_qualname` field.

3.3

Return the code object associated with the function object *op*.

Return the globals dictionary associated with the function object *op*.

Return a `borrowed reference` to the `~function.__module__` attribute of the `function object <user-defined-funcs>` *op*. It can be *NULL*.

This is normally a `string <str>` containing the module name, but can be set to any other object by Python code.

In the free-threaded build, if the attribute may be modified concurrently, convert the borrowed reference to a strong reference with `Py_XNewRef` immediately, before calling any other Python API. The same applies to `PyFunction_GET_MODULE`.

Return the argument default values of the function object *op*. This can be a tuple of arguments or `NULL`.

Set the argument default values for the function object *op*. *defaults* must be `Py_None` or a tuple.

Raises `SystemError` and returns `-1` on failure.

Set the vectorcall field of a given function object *func*.

Warning: extensions using this API must preserve the behavior of the unaltered (default) vectorcall function!

3.12

Return the keyword-only argument default values of the function object *op*. This can be a dictionary of arguments or `NULL`.

Set the keyword-only argument default values of the function object *op*. *defaults* must be a dictionary of keyword-only arguments or `Py_None`.

This function returns `0` on success, and returns `-1` with an exception set on failure.

Return the closure associated with the function object *op*. This can be `NULL` or a tuple of cell objects.

Set the closure associated with the function object *op*. *closure* must be `Py_None` or a tuple of cell objects.

Raises `SystemError` and returns `-1` on failure.

Return the annotations of the function object *op*. This can be a mutable dictionary or `NULL`.

Set the annotations for the function object *op*. *annotations* must be a dictionary or `Py_None`.

Raises `SystemError` and returns `-1` on failure.

Register *callback* as a function watcher for the current interpreter. Return an ID which may be passed to `PyFunction_ClearWatcher`. In case of error (e.g. no more watcher IDs available), return `-1` and set an exception.

3.12

Clear watcher identified by *watcher_id* previously returned from `PyFunction_AddWatcher` for the current interpreter. Return `0` on success, or `-1` and set an exception on error (e.g. if the given *watcher_id* was never registered.)

3.12

Enumeration of possible function watcher events:

- `PyFunction_EVENT_CREATE`
- `PyFunction_EVENT_DESTROY`
- `PyFunction_EVENT_MODIFY_CODE`
- `PyFunction_EVENT_MODIFY_DEFAULTS`
- `PyFunction_EVENT_MODIFY_KWDEFAULTS`

3.12

- `PyFunction_EVENT_MODIFY_QUALNAME`

3.15

Apply the macro *V* to each suffix of `PyFunction_WatchEvent` constants. Currently expands to `V(CREATE) V(DESTROY) V(MODIFY_CODE) V(MODIFY_DEFAULTS) V(MODIFY_KWDEFAULTS) V(MODIFY_QUALNAME)`. The macro *V* must accept one event name and supply any punctuation needed between expansions.

Type of a function watcher callback function.

If *event* is `PyFunction_EVENT_CREATE` or `PyFunction_EVENT_DESTROY` then *new_value* will be `NULL`. Otherwise, *new_value* will hold a `borrowed reference` to the new value that is about to be stored in *func* for the attribute that is being modified.

The callback may inspect but must not modify *func*; doing so could have unpredictable effects, including infinite recursion.

If *event* is `PyFunction_EVENT_CREATE`, then the callback is invoked after *func* has been fully initialized. Otherwise, the callback is invoked before the modification to *func* takes place, so the prior state of *func* can be inspected. The runtime is permitted to optimize away the creation of function objects when possible. In

such cases no event will be emitted. Although this creates the possibility of an observable difference of runtime behavior depending on optimization decisions, it does not change the semantics of the Python code being executed.

If *event* is `PyFunction_EVENT_DESTROY`, taking a reference in the callback to the about-to-be-destroyed function will resurrect it, preventing it from being freed at this time. When the resurrected object is destroyed later, any watcher callbacks active at that time will be called again.

If the callback sets an exception, it must return `-1`; this exception will be printed as an unraisable exception using `PyErr_WriteUnraisable`. Otherwise it should return `0`.

There may already be a pending exception set on entry to the callback. In this case, the callback should return `0` with the same exception still set. This means the callback may not call any other API that can set an exception unless it saves and clears the exception state first, and restores it before returning.

3.12

□□□ #2
□ □□□ □□□ 2026-10-31 16:51:19 UTC
□ □□□ □□□ 2026-10-31 16:51:21 UTC