Allocating Objects on the Heap

PyObject* **_PyObject_New**(PyTypeObject **type*)

Return value: New reference.

PyVarObject* **_PyObject_NewVar**(PyTypeObject **type*, Py_ssize_t *size*)

Return value: New reference.

PyObject* **PyObject_Init**(PyObject **op*, PyTypeObject **type*)

Return value: Borrowed reference.

Initialize a newly-allocated object *op* with its type and initial reference. Returns the initialized object. If *type* indicates that the object participates in the cyclic garbage detector, it is added to the detector's set of observed objects. Other fields of the object are not affected.

PyVarObject* **PyObject_InitVar**(PyVarObject **op*, PyTypeObject **type*, Py_ssize_t *size*)

Return value: Borrowed reference.

This does everything `PyObject_Init()` does, and also initializes the length information for a variable-size object.

TYPE* **PyObject_New**(TYPE, PyTypeObject **type*)

Return value: New reference.

Allocate a new Python object using the C structure type *TYPE* and the Python type object *type*. Fields not defined by the Python object header are not initialized; the object's reference count will be one. The size of the memory allocation is determined from the `tp_basicsize` field of the type object.

TYPE* **PyObject_NewVar**(TYPE, PyTypeObject **type*, Py_ssize_t *size*)

Return value: New reference.

Allocate a new Python object using the C structure type *TYPE* and the Python type object *type*. Fields not defined by the Python object header are not initialized. The allocated memory allows for the *TYPE* structure plus *size* fields of the size given by the `tp_itemsize` field of *type*. This is useful for implementing objects like tuples, which are able to determine their size at construction time. Embedding the array of fields into the same allocation decreases the number of allocations, improving the memory management efficiency.

void **PyObject_Del**(PyObject **op*)

Releases memory allocated to an object using `PyObject_New()` or `PyObject_NewVar()`. This is normally called from the `tp_dealloc` handler specified in the object's type. The fields of the object should not be accessed after this call as the memory is no longer a valid Python object.

PyObject **_Py_NoneStruct**

Object which is visible in Python as `None`. This should only be accessed using the `Py_None` macro, which evaluates to a pointer to this object.

See also:

`PyModule_Create()`

To allocate and create extension modules.