

Defining Extension Types: Tutorial

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 1)

Unknown directive type "highlight".

```
.. highlight:: c
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 9)

Unknown directive type "sectionauthor".

```
.. sectionauthor:: Michael Hudson <mwh@python.net>
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 10)

Unknown directive type "sectionauthor".

```
.. sectionauthor:: Dave Kuhlman <dkuhlman@rexx.com>
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 11)

Unknown directive type "sectionauthor".

```
.. sectionauthor:: Jim Fulton <jim@zope.com>
```

Python allows the writer of a C extension module to define new types that can be manipulated from Python code, much like the built-in `:class: str` and `:class: list` types. The code for all extension types follows a pattern, but there are some details that you need to understand before you can get started. This document is a gentle introduction to the topic.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 14); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 14); [backlink](#)

Unknown interpreted text role "class".

The Basics

The `:term: CPython` runtime sees all Python objects as variables of type `:c:type: PyObject*`, which serves as a "base type" for all Python objects. The `:c:type: PyObject` structure itself only contains the object's `:term: reference count` and a pointer to the object's "type object". This is where the action is; the type object determines which (C) functions get called by the interpreter when, for instance, an attribute gets looked up on an object, a method called, or it is multiplied by another object. These C functions are called "type methods".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 26); [backlink](#)

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 26); [backlink](#)

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 26); [backlink](#)

Unknown interpreted text role "ctype".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 26); [backlink](#)

Unknown interpreted text role "term".

So, if you want to define a new extension type, you need to create a new type object.

This sort of thing can only be explained by example, so here's a minimal, but complete, module that defines a new type named `class:Custom` inside a C extension module `mod:custom`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 38); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 38); [backlink](#)

Unknown interpreted text role "mod".

Note

What we're showing here is the traditional way of defining *static* extension types. It should be adequate for most uses. The C API also allows defining heap-allocated extension types using the `c:func:PyType_FromSpec` function, which isn't covered in this tutorial.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 43); [backlink](#)

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 48)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/custom.c
```

Now that's quite a bit to take in at once, but hopefully bits will seem familiar from the previous chapter. This file defines three things:

1. What a `class:Custom` **object** contains: this is the `CustomObject` struct, which is allocated once for each `class:Custom` instance.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 53); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 53); [backlink](#)

Unknown interpreted text role "class".

2. How the `class:Custom` **type** behaves: this is the `CustomType` struct, which defines a set of flags and function pointers that the interpreter inspects when specific operations are requested.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-

resources\cpython-main\Doc\extending\[cpython-main] [Doc]
[extending]newtypes_tutorial.rst, line 55); [backlink](#)

Unknown interpreted text role "class".

- How to initialize the `mod:custom` module: this is the `PyInit_custom` function and the associated `custommodule` struct.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc]
[extending]newtypes_tutorial.rst, line 58); [backlink](#)

Unknown interpreted text role "mod".

The first bit is:

```
typedef struct {  
    PyObject_HEAD  
} CustomObject;
```

This is what a Custom object will contain. `PyObject_HEAD` is mandatory at the start of each object struct and defines a field called `ob_base` of type `:c:type:'PyObject'`, containing a pointer to a type object and a reference count (these can be accessed using the macros `:c:macro:'Py_TYPE'` and `:c:macro:'Py_REFCNT'` respectively). The reason for the macro is to abstract away the layout and to enable additional fields in `ref: debug builds <debug-build>`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc]
[extending]newtypes_tutorial.rst, line 67); [backlink](#)

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc]
[extending]newtypes_tutorial.rst, line 67); [backlink](#)

Unknown interpreted text role "c:macro".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc]
[extending]newtypes_tutorial.rst, line 67); [backlink](#)

Unknown interpreted text role "c:macro".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc]
[extending]newtypes_tutorial.rst, line 67); [backlink](#)

Unknown interpreted text role "ref".

Note

There is no semicolon above after the `:c:macro:'PyObject_HEAD'` macro. Be wary of adding one by accident: some compilers will complain.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc]
[extending]newtypes_tutorial.rst, line 76); [backlink](#)

Unknown interpreted text role "c:macro".

Of course, objects generally store additional data besides the standard `PyObject_HEAD` boilerplate; for example, here is the definition for standard Python floats:

```
typedef struct {  
    PyObject_HEAD  
    double ob_fval;  
} PyFloatObject;
```

The second bit is the definition of the type object.

```
static PyTypeObject CustomType = {  
    PyVarObject_HEAD_INIT(NULL, 0)  
    .tp_name = "custom.Custom",
```

```
.tp_doc = "Custom objects",
.tp_basicsize = sizeof(CustomObject),
.tp_itemsize = 0,
.tp_flags = Py_TPFLAGS_DEFAULT,
.tp_new = PyType_GenericNew,
};
```

Note

We recommend using C99-style designated initializers as above, to avoid listing all the `:c:type:'PyTypeObject'` fields that you don't care about and also to avoid caring about the fields' declaration order.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 101); [backlink](#)

Unknown interpreted text role "c.type".

The actual definition of `:c:type:'PyTypeObject'` in `:file:'object.h'` has many more `:ref:'fields <type-structs>'` than the definition above. The remaining fields will be filled with zeros by the C compiler, and it's common practice to not specify them explicitly unless you need them.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 105); [backlink](#)

Unknown interpreted text role "c.type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 105); [backlink](#)

Unknown interpreted text role "file".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 105); [backlink](#)

Unknown interpreted text role "ref".

We're going to pick it apart, one field at a time:

```
PyVarObject_HEAD_INIT(NULL, 0)
```

This line is mandatory boilerplate to initialize the `ob_base` field mentioned above.

```
.tp_name = "custom.Custom",
```

The name of our type. This will appear in the default textual representation of our objects and in some error messages, for example:

```
>>> "" + custom.Custom()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "custom.Custom") to str
```

Note that the name is a dotted name that includes both the module name and the name of the type within the module. The module in this case is `:mod:'custom'` and the type is `:class:'Custom'`, so we set the type name to `:class:'custom.Custom'`. Using the real dotted import path is important to make your type compatible with the `:mod:'pydoc'` and `:mod:'pickle'` modules.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 129); [backlink](#)

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 129); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 129); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 129); [backlink](#)

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 129); [backlink](#)

Unknown interpreted text role "mod".

```
.tp_basicsize = sizeof(CustomObject),  
.tp_itemsize = 0,
```

This is so that Python knows how much memory to allocate when creating new `:class:`Custom`` instances. `:c:member:`~PyTypeObject.tp_itemsize`` is only used for variable-sized objects and should otherwise be zero.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 138); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 138); [backlink](#)

Unknown interpreted text role "c:member".

Note

If you want your type to be subclassable from Python, and your type has the same `:c:member:`~PyTypeObject.tp_basicsize`` as its base type, you may have problems with multiple inheritance. A Python subclass of your type will have to list your type first in its `:attr:`~class.__bases__``, or else it will not be able to call your type's `:meth:`~__new__`` method without getting an error. You can avoid this problem by ensuring that your type has a larger value for `:c:member:`~PyTypeObject.tp_basicsize`` than its base type does. Most of the time, this will be true anyway, because either your base type will be `:class:`object``, or else you will be adding data members to your base type, and therefore increasing its size.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 144); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 144); [backlink](#)

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 144); [backlink](#)

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 144); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 144); [backlink](#)

Unknown interpreted text role "class".

We set the class flags to `:const:Py_TPFLAGS_DEFAULT`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 154); [backlink](#)

Unknown interpreted text role "const".

```
.tp_flags = Py_TPFLAGS_DEFAULT,
```

All types should include this constant in their flags. It enables all of the members defined until at least Python 3.3. If you need further members, you will need to OR the corresponding flags.

We provide a doc string for the type in `:c:member:~PyTypeObject.tp_doc`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 162); [backlink](#)

Unknown interpreted text role "c:member".

```
.tp_doc = "Custom objects",
```

To enable object creation, we have to provide a `:c:member:~PyTypeObject.tp_new` handler. This is the equivalent of the Python method `:meth:~__new__`, but has to be specified explicitly. In this case, we can just use the default implementation provided by the API function `:c:func:PyType_GenericNew`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 166); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 166); [backlink](#)

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 166); [backlink](#)

Unknown interpreted text role "c:func".

```
.tp_new = PyType_GenericNew,
```

Everything else in the file should be familiar, except for some code in `:c:func:PyInit_custom`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 173); [backlink](#)

Unknown interpreted text role "c:func".

```
if (PyType_Ready(&CustomType) < 0)
    return;
```

This initializes the `:class:Custom` type, filling in a number of members to the appropriate default values, including `:attr:ob_type` that we initially set to NULL.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 179); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 179); [backlink](#)

Unknown interpreted text role "attr".

```
Py_INCREF(&CustomType);
if (PyModule_AddObject(m, "Custom", (PyObject *) &CustomType) < 0) {
    Py_DECREF(&CustomType);
    Py_DECREF(m);
    return NULL;
}
```

This adds the type to the module dictionary. This allows us to create `:class:`Custom`` instances by calling the `:class:`Custom`` class:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 190); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 190); [backlink](#)

Unknown interpreted text role "class".

```
>>> import custom
>>> mycustom = custom.Custom()
```

That's it! All that remains is to build it; put the above code in a file called `:file:`custom.c`` and:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 198); [backlink](#)

Unknown interpreted text role "file".

```
from distutils.core import setup, Extension
setup(name="custom", version="1.0",
      ext_modules=[Extension("custom", ["custom.c"])])
```

in a file called `:file:`setup.py``; then typing

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 207); [backlink](#)

Unknown interpreted text role "file".

```
$ python setup.py build
```

at a shell should produce a file `:file:`custom.so`` in a subdirectory; move to that directory and fire up Python --- you should be able to `import custom` and play around with `Custom` objects.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 213); [backlink](#)

Unknown interpreted text role "file".

That wasn't so hard, was it?

Of course, the current `Custom` type is pretty uninteresting. It has no data and doesn't do anything. It can't even be subclassed.

Note

While this documentation showcases the standard `:mod:`distutils`` module for building C extensions, it is recommended in real-world use cases to use the newer and better-maintained `setuptools` library. Documentation on how to do this is out of scope for this document and can be found in the [Python Packaging User's Guide](#).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 223); [backlink](#)

Unknown interpreted text role "mod".

Adding data and methods to the Basic example

Let's extend the basic example to add some data and methods. Let's also make the type usable as a base class. We'll create a new module, `mod:custom2` that adds these capabilities:

```
System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 233); backlink
Unknown interpreted text role "mod".
```

```
System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 237)
Unknown directive type "literalinclude".

.. literalinclude:: ../includes/custom2.c
```

This version of the module has a number of changes.

We've added an extra include:

```
#include <structmember.h>
```

This include provides declarations that we use to handle attributes, as described a bit later.

The `:class:Custom` type now has three data attributes in its C struct, *first*, *last*, and *number*. The *first* and *last* variables are Python strings containing first and last names. The *number* attribute is a C integer.

```
System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 249); backlink
Unknown interpreted text role "class".
```

The object structure is updated accordingly:

```
typedef struct {
    PyObject_HEAD
    PyObject *first; /* first name */
    PyObject *last;  /* last name */
    int number;
} CustomObject;
```

Because we now have data to manage, we have to be more careful about object allocation and deallocation. At a minimum, we need a deallocation method:

```
static void
Custom_dealloc(CustomObject *self)
{
    Py_XDECREF(self->first);
    Py_XDECREF(self->last);
    Py_TYPE(self)->tp_free((PyObject *) self);
}
```

which is assigned to the `:c:member:~PyTypeObject.tp_dealloc` member:

```
System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 273); backlink
Unknown interpreted text role "c:member".
```

```
.tp_dealloc = (destructor) Custom_dealloc,
```

This method first clears the reference counts of the two Python attributes. `:c:func:Py_XDECREF` correctly handles the case where its argument is NULL (which might happen here if `tp_new` failed midway). It then calls the `:c:member:~PyTypeObject.tp_free` member of the object's type (computed by `Py_TYPE(self)`) to free the object's memory. Note that the object's type might not be `:class:CustomType`, because the object may be an instance of a subclass.

```
System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 277); backlink
Unknown interpreted text role "c:func".
```

```
System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-
```


main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 277); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 277); [backlink](#)

Unknown interpreted text role "class".

Note

The explicit cast to destructor above is needed because we defined `Custom_dealloc` to take a `CustomObject *` argument, but the `tp_dealloc` function pointer expects to receive a `PyObject *` argument. Otherwise, the compiler will emit a warning. This is object-oriented polymorphism, in C!

We want to make sure that the first and last names are initialized to empty strings, so we provide a `tp_new` implementation:

```
static PyObject *
Custom_new(PyTypeObject *type, PyObject *args, PyObject *kwargs)
{
    CustomObject *self;
    self = (CustomObject *) type->tp_alloc(type, 0);
    if (self != NULL) {
        self->first = PyUnicode_FromString("");
        if (self->first == NULL) {
            Py_DECREF(self);
            return NULL;
        }
        self->last = PyUnicode_FromString("");
        if (self->last == NULL) {
            Py_DECREF(self);
            return NULL;
        }
        self->number = 0;
    }
    return (PyObject *) self;
}
```

and install it in the `c:member:~PyTypeObject.tp_new` member:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 316); [backlink](#)

Unknown interpreted text role "c:member".

```
.tp_new = Custom_new,
```

The `tp_new` handler is responsible for creating (as opposed to initializing) objects of the type. It is exposed in Python as the `meth: '__new__'` method. It is not required to define a `tp_new` member, and indeed many extension types will simply reuse `c:func:PyType_GenericNew` as done in the first version of the `Custom` type above. In this case, we use the `tp_new` handler to initialize the `first` and `last` attributes to non-NULL default values.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 320); [backlink](#)

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 320); [backlink](#)

Unknown interpreted text role "c:func".

`tp_new` is passed the type being instantiated (not necessarily `CustomType`, if a subclass is instantiated) and any arguments passed when the type was called, and is expected to return the instance created. `tp_new` handlers always accept positional and keyword arguments, but they often ignore the arguments, leaving the argument handling to initializer (a.k.a. `tp_init` in C or `__init__` in Python) methods.

Note

`tp_new` shouldn't call `tp_init` explicitly, as the interpreter will do it itself.

The `tp_new` implementation calls the `:c:member:~PyObject.tp_alloc` slot to allocate memory:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 339); [backlink](#)
Unknown interpreted text role "c:member".

```
self = (CustomObject *) type->tp_alloc(type, 0);
```

Since memory allocation may fail, we must check the `:c:member:~PyObject.tp_alloc` result against `NULL` before proceeding.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 344); [backlink](#)
Unknown interpreted text role "c:member".

Note

We didn't fill the `:c:member:~PyObject.tp_alloc` slot ourselves. Rather `:c:func:PyObjectReady` fills it for us by inheriting it from our base class, which is `:class:object` by default. Most types use the default allocation strategy.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 348); [backlink](#)
Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 348); [backlink](#)
Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 348); [backlink](#)
Unknown interpreted text role "class".

Note

If you are creating a co-operative `:c:member:~PyObject.tp_new` (one that calls a base type's `:c:member:~PyObject.tp_new` or `:meth:__new__`), you must *not* try to determine what method to call using method resolution order at runtime. Always statically determine what type you are going to call, and call its `:c:member:~PyObject.tp_new` directly, or via `type->tp_base->tp_new`. If you do not do this, Python subclasses of your type that also inherit from other Python-defined classes may not work correctly. (Specifically, you may not be able to create instances of such subclasses without getting a `:exc:TypeError`.)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 354); [backlink](#)
Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 354); [backlink](#)
Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 354); [backlink](#)
Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 354); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 354); [backlink](#)

Unknown interpreted text role "exc".

We also define an initialization function which accepts arguments to provide initial values for our instance:

```
static int
Custom_init(CustomObject *self, PyObject *args, PyObject *kwds)
{
    static char *kwlist[] = {"first", "last", "number", NULL};
    PyObject *first = NULL, *last = NULL, *tmp;

    if (!PyArg_ParseTupleAndKeywords(args, kwds, "|OOi", kwlist,
                                     &first, &last,
                                     &self->number))
        return -1;

    if (first) {
        tmp = self->first;
        Py_INCREF(first);
        self->first = first;
        Py_XDECREF(tmp);
    }
    if (last) {
        tmp = self->last;
        Py_INCREF(last);
        self->last = last;
        Py_XDECREF(tmp);
    }
    return 0;
}
```

by filling the `c:member:~PyObject.tp_init` slot.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 393); [backlink](#)

Unknown interpreted text role "c:member".

```
.tp_init = (initproc) Custom_init,
```

The `c:member:~PyObject.tp_init` slot is exposed in Python as the `meth:~__init__` method. It is used to initialize an object after it's created. Initializers always accept positional and keyword arguments, and they should return either 0 on success or -1 on error.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 397); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 397); [backlink](#)

Unknown interpreted text role "meth".

Unlike the `tp_new` handler, there is no guarantee that `tp_init` is called at all (for example, the `mod:pickle` module by default doesn't call `meth:~__init__` on unpickled instances). It can also be called multiple times. Anyone can call the `meth:~__init__` method on our objects. For this reason, we have to be extra careful when assigning the new attribute values. We might be tempted, for example to assign the `first` member like this:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 402); [backlink](#)

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 402); [backlink](#)

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 402); [backlink](#)

Unknown interpreted text role "meth".

```
if (first) {
    Py_XDECREF(self->first);
    Py_INCREF(first);
    self->first = first;
}
```

But this would be risky. Our type doesn't restrict the type of the `first` member, so it could be any kind of object. It could have a destructor that causes code to be executed that tries to access the `first` member; or that destructor could release the `:term:`Global interpreter Lock <GIL>`` and let arbitrary code run in other threads that accesses and modifies our object.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 416); [backlink](#)

Unknown interpreted text role "term".

To be paranoid and protect ourselves against this possibility, we almost always reassign members before decrementing their reference counts. When don't we have to do this?

- when we absolutely know that the reference count is greater than 1;
- when we know that deallocation of the object [1] will neither release the `:term:`GIL`` nor cause any calls back into our type's code;

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 429); [backlink](#)

Unknown interpreted text role "term".

- when decrementing a reference count in a `:c:member:`~PyObject.tp_dealloc`` handler on a type which doesn't support cyclic garbage collection [2].

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 432); [backlink](#)

Unknown interpreted text role "c:member".

We want to expose our instance variables as attributes. There are a number of ways to do that. The simplest way is to define member definitions:

```
static PyMemberDef Custom_members[] = {
    {"first", T_OBJECT_EX, offsetof(CustomObject, first), 0,
     "first name"},
    {"last", T_OBJECT_EX, offsetof(CustomObject, last), 0,
     "last name"},
    {"number", T_INT, offsetof(CustomObject, number), 0,
     "custom number"},
    {NULL} /* Sentinel */
};
```

and put the definitions in the `:c:member:`~PyObject.tp_members`` slot:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 448); [backlink](#)

Unknown interpreted text role "c:member".

```
.tp_members = Custom_members,
```

Each member definition has a member name, type, offset, access flags and documentation string. See the [:ref: Generic-Attribute-Management](#) section below for details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 452); [backlink](#)
Unknown interpreted text role "ref".

A disadvantage of this approach is that it doesn't provide a way to restrict the types of objects that can be assigned to the Python attributes. We expect the first and last names to be strings, but any Python objects can be assigned. Further, the attributes can be deleted, setting the C pointers to `NULL`. Even though we can make sure the members are initialized to non-`NULL` values, the members can be set to `NULL` if the attributes are deleted.

We define a single method, `meth:'Custom.name()'`, that outputs the objects name as the concatenation of the first and last names.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 463); [backlink](#)
Unknown interpreted text role "meth".

```
static PyObject *
Custom_name(CustomObject *self, PyObject *Py_UNUSED(ignored))
{
    if (self->first == NULL) {
        PyErr_SetString(PyExc_AttributeError, "first");
        return NULL;
    }
    if (self->last == NULL) {
        PyErr_SetString(PyExc_AttributeError, "last");
        return NULL;
    }
    return PyUnicode_FromFormat("%S %S", self->first, self->last);
}
```

The method is implemented as a C function that takes a `:class:'Custom'` (or `:class:'Custom'` subclass) instance as the first argument. Methods always take an instance as the first argument. Methods often take positional and keyword arguments as well, but in this case we don't take any and don't need to accept a positional argument tuple or keyword argument dictionary. This method is equivalent to the Python method:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 480); [backlink](#)
Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 480); [backlink](#)
Unknown interpreted text role "class".

```
def name(self):
    return "%s %s" % (self.first, self.last)
```

Note that we have to check for the possibility that our `:attr:'first'` and `:attr:'last'` members are `NULL`. This is because they can be deleted, in which case they are set to `NULL`. It would be better to prevent deletion of these attributes and to restrict the attribute values to be strings. We'll see how to do that in the next section.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 492); [backlink](#)
Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 492); [backlink](#)
Unknown interpreted text role "attr".

Now that we've defined the method, we need to create an array of method definitions:

```
static PyMethodDef Custom_methods[] = {
    {"name", (PyCFunction) Custom_name, METH_NOARGS,
     "Return the name, combining the first and last name"
    },
    {NULL} /* Sentinel */
};
```

(note that we used the `:const:'METH_NOARGS'` flag to indicate that the method is expecting no arguments other than *self*)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending] newtypes_tutorial.rst, line 508); [backlink](#)
Unknown interpreted text role "const".

and assign it to the `:c:member:'~PyTypeObject.tp_methods'` slot:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending] newtypes_tutorial.rst, line 511); [backlink](#)
Unknown interpreted text role "c:member".

```
.tp_methods = Custom_methods,
```

Finally, we'll make our type usable as a base class for subclassing. We've written our methods carefully so far so that they don't make any assumptions about the type of the object being created or used, so all we need to do is to add the `:const:'Py_TPFLAGS_BASETYPE'` to our class flag definition:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending] newtypes_tutorial.rst, line 515); [backlink](#)
Unknown interpreted text role "const".

```
.tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE,
```

We rename `:c:func:'PyInit_custom'` to `:c:func:'PyInit_custom2'`, update the module name in the `:c:type:'PyModuleDef'` struct, and update the full class name in the `:c:type:'PyTypeObject'` struct.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending] newtypes_tutorial.rst, line 522); [backlink](#)
Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending] newtypes_tutorial.rst, line 522); [backlink](#)
Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending] newtypes_tutorial.rst, line 522); [backlink](#)
Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending] newtypes_tutorial.rst, line 522); [backlink](#)
Unknown interpreted text role "c:type".

Finally, we update our `:file:'setup.py'` file to build the new module:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending] newtypes_tutorial.rst, line 526); [backlink](#)
Unknown interpreted text role "file".

```
from distutils.core import setup, Extension
setup(name="custom", version="1.0",
      ext_modules=[
          Extension("custom", ["custom.c"]),
          Extension("custom2", ["custom2.c"]),
```

Providing finer control over data attributes

In this section, we'll provide finer control over how the `:attr:'first'` and `:attr:'last'` attributes are set in the `:class:'Custom'` example. In the previous version of our module, the instance variables `:attr:'first'` and `:attr:'last'` could be set to non-string values or even deleted. We want to make sure that these attributes always contain strings.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 541); [backlink](#)

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 541); [backlink](#)

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 541); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 541); [backlink](#)

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 541); [backlink](#)

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 547)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/custom3.c
```

To provide greater control, over the `:attr:'first'` and `:attr:'last'` attributes, we'll use custom getter and setter functions. Here are the functions for getting and setting the `:attr:'first'` attribute:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 550); [backlink](#)

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 550); [backlink](#)

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 550); [backlink](#)

Unknown interpreted text role "attr".

```
static PyObject *
Custom_getfirst(CustomObject *self, void *closure)
{
    Py_INCREF(self->first);
    return self->first;
}
```



```
static int
Custom_setfirst(CustomObject *self, PyObject *value, void *closure)
{
    PyObject *tmp;
    if (value == NULL) {
        PyErr_SetString(PyExc_TypeError, "Cannot delete the first attribute");
        return -1;
    }
    if (!PyUnicode_Check(value)) {
        PyErr_SetString(PyExc_TypeError,
                        "The first attribute value must be a string");
        return -1;
    }
    tmp = self->first;
    Py_INCREF(value);
    self->first = value;
    Py_DECREF(tmp);
    return 0;
}
```

The getter function is passed a `:class:'Custom'` object and a "closure", which is a void pointer. In this case, the closure is ignored. (The closure supports an advanced usage in which definition data is passed to the getter and setter. This could, for example, be used to allow a single set of getter and setter functions that decide the attribute to get or set based on data in the closure.)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 581); [backlink](#)
Unknown interpreted text role "class".

The setter function is passed the `:class:'Custom'` object, the new value, and the closure. The new value may be `NULL`, in which case the attribute is being deleted. In our setter, we raise an error if the attribute is deleted or if its new value is not a string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 587); [backlink](#)
Unknown interpreted text role "class".

We create an array of `:c:type:'PyGetSetDef'` structures:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 592); [backlink](#)
Unknown interpreted text role "c:type".

```
static PyGetSetDef Custom_getsetters[] = {
    {"first", (getter) Custom_getfirst, (setter) Custom_setfirst,
     "first name", NULL},
    {"last", (getter) Custom_getlast, (setter) Custom_setlast,
     "last name", NULL},
    {NULL} /* Sentinel */
};
```

and register it in the `:c:member:'~PyTypeObject.tp_getset'` slot:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 602); [backlink](#)
Unknown interpreted text role "c:member".

```
.tp_getset = Custom_getsetters,
```

The last item in a `:c:type:'PyGetSetDef'` structure is the "closure" mentioned above. In this case, we aren't using a closure, so we just pass `NULL`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 606); [backlink](#)
Unknown interpreted text role "c:type".

We also remove the member definitions for these attributes:

```
static PyMemberDef Custom_members[] = {
    {"number", T_INT, offsetof(CustomObject, number), 0,
```

```

        "custom number"},
        {NULL} /* Sentinel */
    };

```

We also need to update the `:c:member:'~PyObject.tp_init'` handler to only allow strings [3] to be passed:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 617); [backlink](#)
Unknown interpreted text role "c:member".

```

static int
Custom_init(CustomObject *self, PyObject *args, PyObject *kwds)
{
    static char *kwlist[] = {"first", "last", "number", NULL};
    PyObject *first = NULL, *last = NULL, *tmp;

    if (!PyArg_ParseTupleAndKeywords(args, kwds, "|UUi", kwlist,
                                     &first, &last,
                                     &self->number))
        return -1;

    if (first) {
        tmp = self->first;
        Py_INCREF(first);
        self->first = first;
        Py_DECREF(tmp);
    }
    if (last) {
        tmp = self->last;
        Py_INCREF(last);
        self->last = last;
        Py_DECREF(tmp);
    }
    return 0;
}

```

With these changes, we can assure that the `first` and `last` members are never `NULL` so we can remove checks for `NULL` values in almost all cases. This means that most of the `:c:func:'Py_XDECREF'` calls can be converted to `:c:func:'Py_DECREF'` calls. The only place we can't change these calls is in the `tp_dealloc` implementation, where there is the possibility that the initialization of these members failed in `tp_new`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 646); [backlink](#)
Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 646); [backlink](#)
Unknown interpreted text role "c:func".

We also rename the module initialization function and module name in the initialization function, as we did before, and we add an extra definition to the `:file:'setup.py'` file.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 653); [backlink](#)
Unknown interpreted text role "file".

Supporting cyclic garbage collection

Python has a `term'`cyclic garbage collector (GC) `<garbage collection>` that can identify unneeded objects even when their reference counts are not zero. This can happen when objects are involved in cycles. For example, consider:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 661); [backlink](#)
Unknown interpreted text role "term".

```

>>> l = []
>>> l.append(l)

```

```
>>> del l
```

In this example, we create a list that contains itself. When we delete it, it still has a reference from itself. Its reference count doesn't drop to zero. Fortunately, Python's cyclic garbage collector will eventually figure out that the list is garbage and free it.

In the second version of the `:class: Custom` example, we allowed any kind of object to be stored in the `:attr: first` or `:attr: last` attributes [4]. Besides, in the second and third versions, we allowed subclassing `:class: Custom`, and subclasses may add arbitrary attributes. For any of those two reasons, `:class: Custom` objects can participate in cycles:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 676); [backlink](#)
Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 676); [backlink](#)
Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 676); [backlink](#)
Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 676); [backlink](#)
Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 676); [backlink](#)
Unknown interpreted text role "class".

```
>>> import custom3
>>> class Derived(custom3.Custom): pass
...
>>> n = Derived()
>>> n.some_attribute = n
```

To allow a `:class: Custom` instance participating in a reference cycle to be properly detected and collected by the cyclic GC, our `:class: Custom` type needs to fill two additional slots and to enable a flag that enables these slots:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 690); [backlink](#)
Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 690); [backlink](#)
Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 694)
Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/custom4.c
```

First, the traversal method lets the cyclic GC know about subobjects that could participate in cycles:

```
static int
Custom_traverse(CustomObject *self, visitproc visit, void *arg)
{
    int vret;
    if (self->first) {
```

```

    vret = visit(self->first, arg);
    if (vret != 0)
        return vret;
}
if (self->last) {
    vret = visit(self->last, arg);
    if (vret != 0)
        return vret;
}
return 0;
}

```

For each subobject that can participate in cycles, we need to call the `:c:func:'visit'` function, which is passed to the traversal method. The `:c:func:'visit'` function takes as arguments the subobject and the extra argument *arg* passed to the traversal method. It returns an integer value that must be returned if it is non-zero.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 717); [backlink](#)

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 717); [backlink](#)

Unknown interpreted text role "c:func".

Python provides a `:c:func:'Py_VISIT'` macro that automates calling visit functions. With `:c:func:'Py_VISIT'`, we can minimize the amount of boilerplate in `Custom_traverse`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 723); [backlink](#)

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 723); [backlink](#)

Unknown interpreted text role "c:func".

```

static int
Custom_traverse(CustomObject *self, visitproc visit, void *arg)
{
    Py_VISIT(self->first);
    Py_VISIT(self->last);
    return 0;
}

```

Note

The `:c:member:'~PyTypeObject.tp_traverse'` implementation must name its arguments exactly *visit* and *arg* in order to use `:c:func:'Py_VISIT'`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 736); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 736); [backlink](#)

Unknown interpreted text role "c:func".

Second, we need to provide a method for clearing any subobjects that can participate in cycles:

```

static int
Custom_clear(CustomObject *self)
{

```

```

Py_CLEAR(self->first);
Py_CLEAR(self->last);
return 0;
}

```

Notice the use of the `:func:Py_CLEAR` macro. It is the recommended and safe way to clear data attributes of arbitrary types while decrementing their reference counts. If you were to call `:func:Py_XDECREF` instead on the attribute before setting it to `NULL`, there is a possibility that the attribute's destructor would call back into code that reads the attribute again (*especially* if there is a reference cycle).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 750); [backlink](#)

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 750); [backlink](#)

Unknown interpreted text role "c:func".

Note

You could emulate `:func:Py_CLEAR` by writing:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 758); [backlink](#)

Unknown interpreted text role "c:func".

```

PyObject *tmp;
tmp = self->first;
self->first = NULL;
Py_XDECREF(tmp);

```

Nevertheless, it is much easier and less error-prone to always use `:func:Py_CLEAR` when deleting an attribute. Don't try to micro-optimize at the expense of robustness!

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 765); [backlink](#)

Unknown interpreted text role "c:func".

The deallocator `Custom_dealloc` may call arbitrary code when clearing attributes. It means the circular GC can be triggered inside the function. Since the GC assumes reference count is not zero, we need to untrack the object from the GC by calling `:func:PyObject_GC_UnTrack` before clearing members. Here is our reimplemented deallocator using `:func:PyObject_GC_UnTrack` and `Custom_clear`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 769); [backlink](#)

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\ [cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 769); [backlink](#)

Unknown interpreted text role "c:func".

```

static void
Custom_dealloc(CustomObject *self)
{
    PyObject_GC_UnTrack(self);
    Custom_clear(self);
    Py_TYPE(self)->tp_free((PyObject *) self);
}

```

Finally, we add the `:const:Py_TPFLAGS_HAVE_GC` flag to the class flags:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 784); [backlink](#)

Unknown interpreted text role "const".

```
.tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE | Py_TPFLAGS_HAVE_GC,
```

That's pretty much it. If we had written custom `:c:member:~PyObject.tp_alloc` or `:c:member:~PyObject.tp_free` handlers, we'd need to modify them for cyclic garbage collection. Most extensions will use the versions automatically provided.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 788); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 788); [backlink](#)

Unknown interpreted text role "c:member".

Subclassing other types

It is possible to create new extension types that are derived from existing types. It is easiest to inherit from the built in types, since an extension can easily use the `:c:type:PyObject` it needs. It can be difficult to share these `:c:type:PyObject` structures between extension modules.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 796); [backlink](#)

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 796); [backlink](#)

Unknown interpreted text role "c:type".

In this example we will create a `:class:SubList` type that inherits from the built-in `:class:list` type. The new type will be completely compatible with regular lists, but will have an additional `:meth:increment` method that increases an internal counter:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 801); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 801); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 801); [backlink](#)

Unknown interpreted text role "meth".

```
>>> import sublist
>>> s = sublist.SubList(range(3))
>>> s.extend(s)
>>> print(len(s))
6
>>> print(s.increment())
1
>>> print(s.increment())
2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 818)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/sublist.c
```

As you can see, the source code closely resembles the `:class:`Custom`` examples in previous sections. We will break down the main differences between them.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 821); [backlink](#)

Unknown interpreted text role "class".

```
typedef struct {
    PyListObject list;
    int state;
} SubListObject;
```

The primary difference for derived type objects is that the base type's object structure must be the first value. The base type will already include the `:cfunc:`PyObject_HEAD`` at the beginning of its structure.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 829); [backlink](#)

Unknown interpreted text role "cfunc".

When a Python object is a `:class:`SubList`` instance, its `PyObject *` pointer can be safely cast to both `PyListObject *` and `SubListObject *`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 833); [backlink](#)

Unknown interpreted text role "class".

```
static int
SubList_init(SubListObject *self, PyObject *args, PyObject *kwds)
{
    if (PyList_Type.tp_init((PyObject *) self, args, kwds) < 0)
        return -1;
    self->state = 0;
    return 0;
}
```

We see above how to call through to the `:attr:`__init__`` method of the base type.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 845); [backlink](#)

Unknown interpreted text role "attr".

This pattern is important when writing a type with custom `:cmember:`~PyTypeObject.tp_new`` and `:cmember:`~PyTypeObject.tp_dealloc`` members. The `:cmember:`~PyTypeObject.tp_new`` handler should not actually create the memory for the object with its `:cmember:`~PyTypeObject.tp_alloc``, but let the base class handle it by calling its own `:cmember:`~PyTypeObject.tp_new``.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 848); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 848); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 848); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 848); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 848); [backlink](#)

Unknown interpreted text role "c:member".

The `c:type:PyTypeObject` struct supports a `c:member:~PyTypeObject.tp_base` specifying the type's concrete base class. Due to cross-platform compiler issues, you can't fill that field directly with a reference to `c:type:PyList_Type`; it should be done later in the module initialization function:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 854); [backlink](#)

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 854); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 854); [backlink](#)

Unknown interpreted text role "c:type".

```
PyMODINIT_FUNC
PyInit_sublist(void)
{
    PyObject* m;
    SubListType.tp_base = &PyList_Type;
    if (PyType_Ready(&SubListType) < 0)
        return NULL;

    m = PyModule_Create(&sublistmodule);
    if (m == NULL)
        return NULL;

    Py_INCREF(&SubListType);
    if (PyModule_AddObject(m, "SubList", (PyObject *) &SubListType) < 0) {
        Py_DECREF(&SubListType);
        Py_DECREF(m);
        return NULL;
    }

    return m;
}
```

Before calling `c:func:PyType_Ready`, the type structure must have the `c:member:~PyTypeObject.tp_base` slot filled in. When we are deriving an existing type, it is not necessary to fill out the `c:member:~PyTypeObject.tp_alloc` slot with `c:func:PyType_GenericNew` -- the allocation function from the base type will be inherited.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 882); [backlink](#)

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 882); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 882); [backlink](#)

Unknown interpreted text role "c:member".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 882); [backlink](#)

Unknown interpreted text role "c:func".

After that, calling `:c:func:PyType_Ready` and adding the type object to the module is the same as with the basic `:class:Custom` examples.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 888); [backlink](#)

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 888); [backlink](#)

Unknown interpreted text role "class".

Footnotes

- [1] This is true when we know that the object is a basic type, like a string or a float.
- [2] We relied on this in the `:c:member:~PyTypeObject.tp_dealloc` handler in this example, because our type doesn't support garbage collection.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 897); [backlink](#)

Unknown interpreted text role "c:member".

- [3] We now know that the first and last members are strings, so perhaps we could be less careful about decrementing their reference counts, however, we accept instances of string subclasses. Even though deallocating normal strings won't call back into our objects, we can't guarantee that deallocating an instance of a string subclass won't call back into our objects.
- [4] Also, even with our attributes restricted to strings instances, the user could pass arbitrary `:class:str` subclasses and therefore still create reference cycles.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\extending\[cpython-main] [Doc] [extending]newtypes_tutorial.rst, line 906); [backlink](#)

Unknown interpreted text role "class".