3.3.1 Basic customization

__init__ (*self*[, *args...*])
Called when the instance is created. The arguments are those passed to the class constructor expression. If a base class has an `__init__()` method the derived class's `__init__()` method must explicitly call it to ensure proper initialization of the base class part of the instance, e.g., "`BaseClass.__init__(`*self*`, [`*args*`...])`".

__del__ (*self*)
Called when the instance is about to be destroyed. This is also called a destructor. If a base class has a `__del__()` method, the derived class's `__del__()` method must explicitly call it to ensure proper deletion of the base class part of the instance. Note that it is possible (though not recommended!) for the `__del__()` method to postpone destruction of the instance by creating a new reference to it. It may then be called at a later time when this new reference is deleted. It is not guaranteed that `__del__()` methods are called for objects that still exist when the interpreter exits.

Programmer's note: "`del x`" doesn't directly call `x.__del__()` -- the former decrements the reference count for `x` by one, and the latter is only called when its reference count reaches zero. Some common situations that may prevent the reference count of an object to go to zero include: circular references between objects (e.g., a doubly-linked list or a tree data structure with parent and child pointers); a reference to the object on the stack frame of a function that caught an exception (the traceback stored in `sys.exc_traceback` keeps the stack frame alive); or a reference to the object on the stack frame that raised an unhandled exception in interactive mode (the traceback stored in `sys.last_traceback` keeps the stack frame alive). The first situation can only be remedied by explicitly breaking the cycles; the latter two situations can be resolved by storing None in `sys.exc_traceback` or `sys.last_traceback`.

Warning: due to the precarious circumstances under which `__del__()` methods are invoked, exceptions that occur during their execution are ignored, and a warning is printed to `sys.stderr` instead. Also, when `__del__()` is invoked is response to a module being deleted (e.g., when execution of the program is done), other globals referenced by the `__del__()` method may already have been deleted. For this reason, `__del__()` methods should do the absolute minimum needed to maintain external invariants. Python 1.5 guarantees that globals whose name begins with a single underscore are deleted from their module before other globals are deleted; if no other references to such globals exist, this may help in assuring that imported modules are still available at the time when the `__del__()` method is called.

__repr__ (*self*)
Called by the `repr()` built-in function and by string conversions (reverse quotes) to compute the ``official'' string representation of an object. This should normally look like a valid Python expression that can be used to recreate an object with the same value. By convention, objects which cannot be trivially converted to strings which can be used to create a similar object produce a string of the form "<*...some useful description...*>".

__str__ (*self*)
Called by the `str()` built-in function and by the `print` statement to compute the ``informal'' string representation of an object. This differs from `__repr__()` in that it does not have to be a valid Python expression: a more convenient or concise representation may be used instead.

__cmp__ (*self, other*)
Called by all comparison operations. Should return a negative integer if `self < other`, zero if `self == other`, a positive integer if `self > other`. If no `__cmp__()` operation is defined, class instances are compared by object identity (``address''). (Note: the restriction that exceptions are not propagated by `__cmp__()` has been removed in Python 1.5.)

__rcmp__ (*self, other*)
Called by all comparison operations. Should return a negative integer if `self < other`, zero if `self == other`, a positive integer if `self > other`. If no `__cmp__()` operation is defined, class instances are compared by object identity (``address''). (Note: the restriction that exceptions are not propagated by `__cmp__()` has been removed in Python 1.5.)

__hash__ (*self*)
Called for the key object for dictionary operations, and by the built-in function `hash()`. Should return a 32-bit integer usable as a hash value for dictionary operations. The only required property is that objects which compare equal have the same hash value; it is advised to somehow mix together (e.g., using exclusive or) the hash values for the components of the object that also play a part in comparison of objects. If a class does not define a `__cmp__()` method it should not define a `__hash__()` operation either; if it defines `__cmp__()` but not `__hash__()` its instances will not be usable as dictionary keys. If a class defines mutable objects and implements a `__cmp__()` method it should not implement `__hash__()`, since the dictionary implementation requires that a key's hash value is immutable (if the object's hash value changes, it will be in the wrong hash bucket).

__nonzero__ (*self*)
Called to implement truth value testing; should return `0` or `1`. When this method is not defined, `__len__()` is called, if it is defined (see below). If a class defines neither `__len__()` nor `__nonzero__()`, all its instances are considered true.


Python Reference Manual


Previous: 3.3 Special method names **Up:** 3.3 Special method names **Next:** 3.3.2 Customizing attribute access

---

See About this document... for information on suggesting changes.