

## Suggested updates to Python Library Reference, Release 2.3.3, Section 2.1 – Built-in Functions

`reload(module)`

Re-compile and reload an already imported *module*. The argument must be a module object, so it must have been successfully imported before. This is useful if you have edited the module source file using an external editor and want to try out the new version without leaving the Python interpreter. The return value is the module object (the same as the *module* argument).

When `reload(module)` is executed:

The objects defined in *module* are compiled and loaded into memory as new objects.

The old objects remain in memory until all references to them are gone, and they are removed by the normal garbage-collection process.

The names in the *module* namespace are updated to point to any new or changed objects. Names of unchanged objects, or of objects no longer present in the new module, remain pointing at the old objects.

Names in other modules that refer directly to the old objects (without the module-name qualifier) remain unchanged and must be updated in each namespace where they occur.

There are a number of caveats:

If a module is syntactically correct but its initialization fails, the first `import` statement for it does not bind its name locally, but does store a (partially initialized) module object in `sys.modules`. To reload the module you must first `import` it again (this will bind the name to the partially initialized module object) before you can `reload()` it.

When a module is reloaded, its dictionary (containing the module's global variables) is retained. Redefinitions of names will override the old definitions, so this is generally not a problem. If the new version of a module does not define a name that was defined by the old version, the old definition remains. This feature can be used to the module's advantage if it maintains a global table or cache of objects -- with a `try` statement it can test for the table's presence and skip its initialization if desired.

It is legal though generally not very useful to reload built-in or dynamically loaded modules, except for `sys`, `__main__` and `__builtin__`. In many cases, however, extension modules are not designed to be initialized more than once, and may fail in arbitrary ways when reloaded.

If a module imports objects from another module using `from ... import ...`, calling `reload()` for the other module does not redefine the objects imported from it -- one way around this is to re-execute the `from` statement, another is to use `import` and qualified names (*module.name*) instead.

If a module instantiates instances of a class, reloading the module that defines the class does not affect the method definitions of the instances -- they continue to use the old class definition. The same is true for derived classes.