The gc module

(Optional, 2.0 and later) This module provides an interface to the built-in cyclic garbage collector.

Python uses reference counting to keep track of when to get rid of objects; as soon as the last reference to an object goes away, the object is destroyed.

Starting with version 2.0, Python also provides a cyclic garbage collector, which runs at regular intervals. This collector looks for data structures that point to themselves, and does what it can to break the cycles.

You can use the **gc.collect** function to force full collection. This function returns the number of objects destroyed by the collector.

Example: Using the gc module to collect cyclic garbage

```
# File: gc-example-1.py

import gc

# create a simple object that links to itself
class Node:

    def __init__(self, name):
        self.name = name
        self.parent = None
        self.children = []

    def addchild(self, node):
        node.parent = self
        self.children.append(node)

    def __repr__(self):
        return "<Node %s at %x>" % (repr(self.name), id(self))

# set up a self-referencing structure
root = Node("monty")

root.addchild(Node("eric"))
root.addchild(Node("john"))
root.addchild(Node("michael"))

# remove our only reference
del root

print gc.collect(), "unreachable objects"
print gc.collect(), "unreachable objects"


12 unreachable objects
0 unreachable objects
```

If you're sure that your program doesn't create any self-referencing data structures, you can use the **gc.disable** function to disable collection. After calling this function, Python works exactly like 1.5.2 and earlier.