GeeksforGeeks
A computer science portal for geeks

Garbage Collection in Python

Python’s memory allocation and deallocation method is automatic. The user does not have to preallocate or deallocate memory similar to using dynamic memory allocation in languages such as C or C++.
Python uses two strategies for memory allocation:

- Reference counting
- Garbage collection

Prior to Python version 2.0, the Python interpreter only used reference counting for memory management. Reference counting works by counting the number of times an object is referenced by other objects in the system. When references to an object are removed, the reference count for an object is decremented. When the reference count becomes zero, the object is deallocated. Ex-

```
# Literal 9 is an object
b = 9

# Reference count of object 9
# becomes 0.
b = 4
```

The literal value 9 is an object. The reference count of object 9 is incremented to 1 in line 1. In line 2 its reference count becomes zero as it is dereferenced. So garbage collector deallocates the object.

A reference cycle is created when there is no way the reference count of the object can reach. Reference cycles involving lists, tuples, instances, classes, dictionaries, and functions are common. The easiest way to create a reference cycle is to create an object which refers to itself as in the example below:

```
def create_cycle():

    # create a list x
    x = [ ]

    # A reference cycle is created
    # here as x contains reference to
    # to self.
    x.append(x)

create_cycle()
```

Because create_cycle() creates an object x which refers to itself, the object x will not automatically be freed when the function returns. This will cause the memory that x is using to be held onto until the Python garbage collector is invoked.

Ways to make an object eligible for garbage collection

```
x = []
x.append(l)
x.append(2)

# delete the list from memory or
# assigning object x to None(Null)
del x
# x = None
```

The reference count for the list created is now two. However, since it cannot be reached from inside Python and cannot possibly be used again, it is considered garbage. In the current version of Python, this list is never freed.

Automatic Garbage Collection of Cycles

Because reference cycles take computational work to discover, garbage collection must be a scheduled activity. Python schedules garbage collection based upon a threshold of object allocations and object deallocations. When the number of allocations minus the number of deallocations is greater than the threshold number, the garbage collector is run. One can inspect the threshold for new objects (objects in Python known as generation 0 objects) by importing the gc module and asking for garbage collection thresholds:

```
# loading gc
import gc

# get the current collection
# thresholds as a tuple
print("Garbage collection thresholds:",
                    gc.get_threshold())
```

Output:

```
Garbage collection thresholds: (700, 10, 10)
```

Here, the default threshold on the above system is 700. This means when the number of allocations vs. the number of deallocations is greater than 700 the automatic garbage collector will run. Thus any portion of your code which frees up large blocks of memory is a good candidate for running manual garbage collection.

Manual Garbage Collection

Invoking the garbage collector manually during the execution of a program can be a good idea on how to handle memory being consumed by reference cycles.
The garbage collection can be invoked manually in the following way:

```
# Importing gc module
import gc

# Returns the number of
# objects it has collected
# and deallocated
collected = gc.collect()

# Prints Garbage collector
# as 0 object
print("Garbage collector: collected",
          "%d objects." % collected)
```

If few cycles are created, then how manual collection works:

Example:

```
import gc
i = 0

# create a cycle and on each iteration x as a dictionary
# assigned to 1
def create_cycle():
    x = { }
    x[i+1] = x
    print x

# lists are cleared whenever a full collection or
# collection of the highest generation (2) is run
collected = gc.collect() # or gc.collect(2)
print "Garbage collector: collected %d objects." % (collected)

print "Creating cycles..."
for i in range(10):
    create_cycle()

collected = gc.collect()

print "Garbage collector: collected %d objects." % (collected)
```

Output:

```
Garbage collector: collected 0 objects.
Creating cycles...
{1: {...}}
{2: {...}}
{3: {...}}
{4: {...}}
{5: {...}}
{6: {...}}
{7: {...}}
{8: {...}}
{9: {...}}
{10: {...}}
Garbage collector: collected 10 objects.
```

There are two ways for performing manual garbage collection: time-based and event-based garbage collection.
Time-based garbage collection is simple: the garbage collector is called after a fixed time interval.
Event-based garbage collection calls the garbage collector on event occurrence. For example, when a user exits the application or when the application enters into idle state.

Reference

- Python Docs

Recommended Posts:

MongoDB python | Delete Data and Drop Collection

Reading Python File-Like Objects from C | Python

Important differences between Python 2.x and Python 3.x with examples

Python | Set 4 (Dictionary, Keywords in Python)

Python | Sort Python Dictionaries by Key or Value

try and except in Python

zip() in Python

set add() in python

SHA in Python

SQL using Python | Set 1

abs() in Python

bin() in Python

Use of min() and max() in Python

Any & All in Python

gcd() in Python

AFZAL ANSARI
Check out this Author's contributed articles.

If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please Improve this article if you find anything incorrect by clicking on the "Improve Article" button below.

Improved By : Akanksha_Rai

Article Tags : Python python-basics

4

□ To-do □ Done

2

Based on **2** vote(s)

Feedback/ Suggest Improvement Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Previous

Python List Comprehension | Three way partitioning of an array around a given range

Next

Stack and Queue in Python using queue Module

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!

Most popular in Python

- Adding new column to existing DataFrame in Pandas
- Dealing with Rows and Columns in Pandas DataFrame
- Iterating over rows and columns in Pandas DataFrame
- Car driving using hand detection in Python
- Python | Real time currency convertor using Tkinter

More related articles in Python

- Take Matrix input from user in Python
- Currying Function in Python
- Python | Split string into list of characters
- Creating a Pandas DataFrame
- Indexing and Selecting Data with Pandas

GeeksforGeeks
A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

About Us
Careers
Privacy Policy
Contact Us

LEARN

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE

Company-wise
Topic-wise
Contests
Subjective Questions

CONTRIBUTE

Write an Article
Write Interview Experience
Internships
Videos