

# Basics of Memory Management in Python

By  Guest Contributor (<https://twitter.com/stackabuse>) •

3 Comments ([/basics-of-memory-management-in-python/#disqus\\_thread](/basics-of-memory-management-in-python/#disqus_thread))

## Introduction

Memory management is the process of efficiently allocating, de-allocating, and coordinating memory so that all the different processes run smoothly and can optimally access different system resources. Memory management also involves cleaning memory of objects that are no longer being accessed.

In Python, the memory manager is responsible for these kinds of tasks by periodically running to clean up, allocate, and manage the memory. Unlike C, Java, and other programming languages, Python manages objects by using reference counting. This means that the memory manager keeps track of the number of references to each object in the program. When an object's reference count drops to zero, which means the object is no longer being used, the garbage collector (part of the memory manager) automatically frees the memory from that particular object.

The user need not to worry about memory management as the process of allocation and de-allocation of memory is fully automatic. The reclaimed memory can be used by other objects.

## Python Garbage Collection

As explained earlier, Python deletes objects that are no longer referenced in the program to free up memory space. This process in which Python frees blocks of memory that are no longer used is called Garbage Collection. The Python Garbage Collector (<https://docs.python.org/3/library/gc.html>) (GC) runs during the program execution and is triggered if the reference count reduces to zero. The reference count increases if an object is assigned a new name or is placed in a container, like tuple or dictionary. Similarly, the reference count decreases when the reference to an object is reassigned, when the object's reference goes out of scope (/local-and-global-variables-in-python/), or when an object is deleted.

The memory is a heap that contains objects and other data structures used in the program. The allocation and de-allocation of this heap space is controlled by the Python Memory manager through the use of API functions.

## Python Objects in Memory

Each variable in Python acts as an object. Objects can either be simple (containing numbers, strings, etc.) or containers (dictionaries, lists, or user defined classes). Furthermore, Python is a dynamically typed language which means that we do not need to declare the variables or their types before using them in a program.

For example:

```
>>> x = 5
>>> print(x)
5
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<mem_manage>", line 1, in <module>
    print(x)
NameError : name 'x' is not defined
```

If you look at the first 2 lines of the above program, object `x` is known. When we delete the object `x` and try to use it, we get an error stating that the variable `x` is not defined.

You can see that the garbage collection in Python is fully automated and the programmer does not need worry about it, unlike languages like C.

## Modifying the Garbage Collector

The Python garbage collector has three generations in which objects are classified. A new object at the starting point of it's life cycle is the first generation of the garbage collector. As the object survives garbage collection, it will be moved up to the next generations. Each of the 3 generations of the garbage collector has a threshold. Specifically, when the threshold of number of allocations minus the number of deAllocations is exceeded, that generation will run garbage collection.

Earlier generations are also garbage collected more often than the higher generations. This is because newer objects are more likely to be discarded than old objects.

The `gc` module includes functions to change the threshold value, trigger a garbage collection process manually, disable the garbage collection process, etc. We can check the threshold values of different generations of the garbage collector using the `get_threshold()` method:

```
import gc
print(gc.get_threshold())
```

### Sample Output:

```
(700, 10, 10)
```

As you see, here we have a threshold of 700 for the first generation, and 10 for each of the other two generations.

We can alter the threshold value for triggering the garbage collection process using the `set_threshold()` method of the `gc` module:

```
gc.set_threshold(900, 15, 15)
```

## Subscribe to our Newsletter

Get occasional tutorials, guides, and jobs in your inbox. No spam ever.

Unsubscribe at any time.

In the above example, we have increased the threshold value for all the 3 generations. Increasing the threshold value will decrease the frequency of running the garbage collector. Normally, we need not think too much about Python's garbage collection as a developer, but this may be useful when optimizing the Python runtime for your target system. One of the key benefits is that Python's garbage collection mechanism handles a lot of low-level details for the developer automatically.

## Why Perform Manual Garbage Collection?

We know that the Python interpreter keeps a track of references to objects used in a program. In earlier versions of Python (until version 1.6), the Python interpreter used only the reference counting mechanism to handle memory. When the reference count drops to zero, the Python interpreter automatically frees the memory. This classical reference counting mechanism is very effective, except that

it fails to work when the program has *reference cycles*. A reference cycle happens if one or more objects are referenced each other, and hence the reference count never reaches zero.

Let's consider an example.

```
>>> def create_cycle():  
...     list = [8, 9, 10]  
...     list.append(list)  
...     return list  
...  
>>> create_cycle()  
[8, 9, 10, [...]]
```

The above code creates a reference cycle, where the object `list` refers to itself. Hence, the memory for the object `list` will not be freed automatically when the function returns. The reference cycle problem can't be solved by reference counting. However, this reference cycle problem can be solved by change the behavior of the garbage collector in your Python application.

To do so, we can use the `gc.collect()` function of the `gc` module.

```
import gc  
n = gc.collect()  
print("Number of unreachable objects collected by GC:", n)
```

The `gc.collect()` returns the number of objects it has collected and de-allocated.

There are two ways to perform manual garbage collection: time-based or event-based garbage collection.

Time-based garbage collection is pretty simple: the `gc.collect()` function is called after a fixed time interval.

Event-based garbage collection calls the `gc.collect()` function after an event occurs (i.e. when the application is exited or the application remains idle for a specific time period).

Let's understand the manual garbage collection work by creating a few reference cycles.

```
import sys, gc

def create_cycle():
    list = [8, 9, 10]
    list.append(list)

def main():
    print("Creating garbage...")
    for i in range(8):
        create_cycle()

    print("Collecting...")
    n = gc.collect()
    print("Number of unreachable objects collected by GC:", n)
    print("Uncollectable garbage:", gc.garbage)

if __name__ == "__main__":
    main()
    sys.exit()
```

The output is as below:

```
Creating garbage...
Collecting...
Number of unreachable objects collected by GC: 8
Uncollectable garbage: []
```

The script above creates a list object that is referred by a variable, creatively named `list`. The first element of the list object refers to itself. The reference count of the list object is always greater than zero even if it is deleted or out of

Privacy


scope in the program. Hence, the `list` object is not garbage collected due to the circular reference. The garbage collector mechanism in Python will automatically check for, and collect, circular references periodically.

In the above code, as the reference count is at least 1 and can never reach 0, we have forcefully garbage collected the objects by calling `gc.collect()`. However, remember not to force garbage collection frequently. The reason is that even after freeing the memory, the GC takes time to evaluate the object's eligibility to be garbage collected, taking up processor time and resources. Also, remember to manually manage the garbage collector only after your app has started completely.

## Conclusion

In this article, we discussed how memory management in Python is handled automatically by using reference counting and garbage collection strategies. Without garbage collection, implementing a successful memory management mechanism in Python is impossible. Also, programmers need not worry about deleting allocated memory, as it is taken care by Python memory manager. This leads to fewer memory leaks and better performance.

---

 [python \(/tag/python/](#)

[com/share?](#)

[%20Memory%20Management%20in%20Python&url=https://stackabuse.com/basics-of-memory-management-in-python/\)](#)

[facebook.com/sharer/sharer.php?u=https://stackabuse.com/basics-of-memory-management-in-python/\)](#)

[linkedin.com/shareArticle?mini=true%26url=https://stackabuse.com/basics-of-memory-management-in-python/%26source=https://stackabuse.com\)](#)

[\(/author/guest/\)](#)

About Guest Contributor [\(/author/guest/\)](#)

Privacy



Twitter (<https://twitter.com/stackabuse>)

## Subscribe to our Newsletter

Get occasional tutorials, guides, and jobs in your inbox. No spam ever. Unsubscribe at any time.

Subscribe

[< Previous Post \(/stripe-integration-with-java-spring-for-payment-processing/\)](/stripe-integration-with-java-spring-for-payment-processing/)

[Next Post > \(/debugging-python-applications-with-the-pdb-module/\)](/debugging-python-applications-with-the-pdb-module/)

### Ad

---

Privacy



## Follow Us

---



(<https://twitter.com/StackAbuse>)



(<https://www.facebook.com/stackabuse/>)

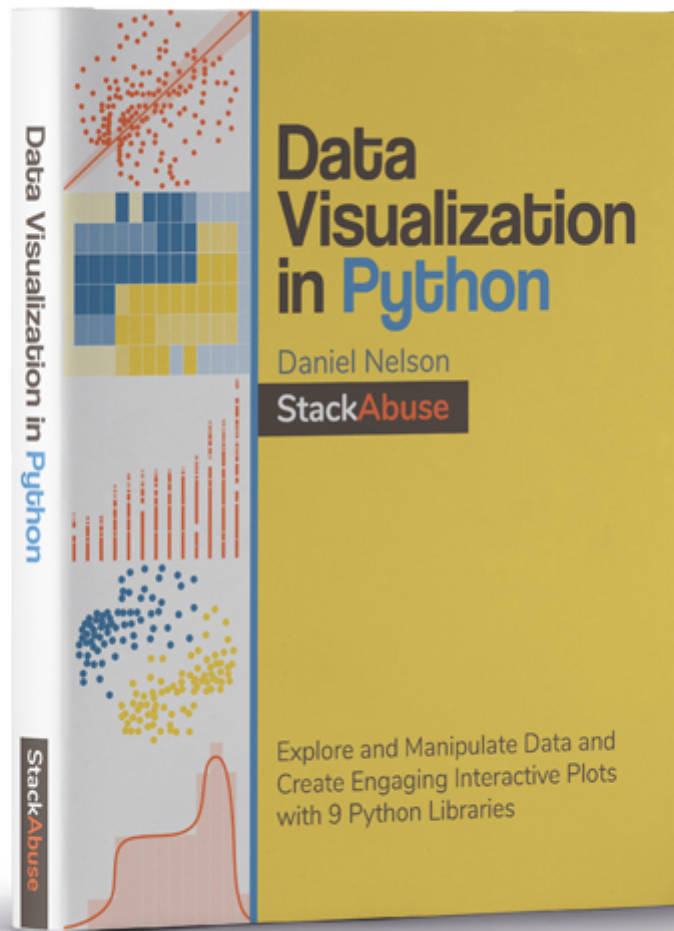


(<https://stackabuse.com/rss/>)

## Data Visualization in Python

---

(<https://gum.co/data-visualization-in-python/>)



(<https://gum.co/data-visualization-in-python>)

**Understand your data better with visualizations!** With over 275+ pages, you'll learn the ins and outs of visualizing data in Python with popular libraries like Matplotlib, Seaborn, Bokeh, and more.

Learn more (<https://gum.co/data-visualization-in-python>)

## Newsletter

Subscribe to our newsletter! Get occasional tutorials, guides, and reviews in your inbox.

Subscribe

No spam ever. Unsubscribe at any time.

Privacy

## Ad

---

## Want a remote job?

---

Backend/Fullstack Engineer

**Bravado** 1 month ago (<https://hireremote.io/remote-job/3007-backend-fullstack-engineer-at-bravado>)

[ruby](https://hireremote.io/remote-ruby-jobs) (<https://hireremote.io/remote-ruby-jobs>) [ruby-on-rails](https://hireremote.io/remote-ruby-on-rails-jobs) (<https://hireremote.io/remote-ruby-on-rails-jobs>) [full-stack](https://hireremote.io/remote-full-stack-jobs) (<https://hireremote.io/remote-full-stack-jobs>) [python](https://hireremote.io/remote-python-jobs) (<https://hireremote.io/remote-python-jobs>)

Junior OTC Trade Desk Developer

**Kraken Digital Asset Exchange** 13 hours ago (<https://hireremote.io/remote-job/3562-junior-otc-trade-desk-developer-at-kraken-digital-asset-exchange>)

[junior](https://hireremote.io/remote-junior-jobs) (<https://hireremote.io/remote-junior-jobs>) [python](https://hireremote.io/remote-python-jobs) (<https://hireremote.io/remote-python-jobs>) [docker](https://hireremote.io/remote-docker-jobs) (<https://hireremote.io/remote-docker-jobs>) [aws](https://hireremote.io/remote-aws-jobs) (<https://hireremote.io/remote-aws-jobs>)

Principal Data Scientist, Attribution

15 hours ago (<https://hireremote.io/remote-job/3567-principal-data-scientist-attribution>)

[python](https://hireremote.io/remote-python-jobs) (<https://hireremote.io/remote-python-jobs>) [data science](https://hireremote.io/remote-data-science-jobs) (<https://hireremote.io/remote-data-science-jobs>)

➔ [More jobs \(https://hireremote.io\)](https://hireremote.io)

Jobs via HireRemote.io (<https://hireremote.io>)

Privacy

## Prepping for an interview?

---

(<https://stackabu.se/daily-coding-problem>)

- Improve your skills by solving one coding problem every day
- Get the solutions the next morning via email
- Practice on **actual problems** asked by top companies, like:

`</>` Daily Coding Problem (<https://stackabu.se/daily-coding-problem>)

## Ad

---



## Recent Posts

---

Flask Form Validation with Flask-WTF (/flask-form-validation-with-flask-wtf/)

---

Java: Check if Array Contains Value or Element (/java-check-if-array-contains-value-or-element/)

---

Guide to Parsing HTML with BeautifulSoup in Python (/guide-to-parsing-html-with-beautifulsoup-in-python/)

---

## Tags

---

[ai \(/tag/ai/\)](/tag/ai/)[algorithms \(/tag/algorithms/\)](/tag/algorithms/)[amqp \(/tag/amqp/\)](/tag/amqp/)[angular \(/tag/angular/\)](/tag/angular/)[announcements \(/tag/announcements/\)](/tag/announcements/)[apache \(/tag/apache/\)](/tag/apache/)[apache commons \(/tag/apache-commons/\)](/tag/apache-commons/)[api \(/tag/api/\)](/tag/api/)[arduino \(/tag/arduino/\)](/tag/arduino/)[artificial intelligence \(/tag/artificial-intelligence/\)](/tag/artificial-intelligence/)

## Follow Us

---

<https://twitter.com/StackAbuse><https://www.facebook.com/stackabuse/><https://stackabuse.com/rss/>

---

Copyright © 2020, Stack Abuse (<https://stackabuse.com>). All Rights Reserved.

[Disclosure \(/disclosure\)](/disclosure) • [Privacy Policy \(/privacy-policy\)](/privacy-policy) • [Terms of Service \(/terms-of-service\)](/terms-of-service)