What is garbage collection (GC)?

Garbage collection (GC) is a memory recovery feature built into programming languages such as C# and Java. A GC-enabled programming language includes one or more garbage collectors (GC engines) that automatically free up memory space that has been allocated to objects no longer needed by the program. The reclaimed memory space can then be used for future object allocations within that program.

Garbage collection ensures that a program does not exceed its memory quota or reach a point that it can no longer function. It also frees up developers from having to manually manage a program's memory, which, in turn, reduces the potential for memory-related bugs.

In older programming languages, such as C and C++, the developer must manually delete objects and free up memory. Relying on manual processes made it easy to introduce bugs into the code, some of which can have serious consequences.

For example, a developer might forget to free up memory after the program no longer needs it, leading to a memory leak that quickly consumes all the available RAM. Or the developer might free up an object's memory space without modifying a corresponding pointer, resulting in a dangling pointer that causes the application to be buggy or even to crash.

Programming languages that include garbage collection try to eliminate these types of bugs by using carefully designed GC algorithms to control memory deallocation. The garbage collector automatically detects when an object is no longer needed and removes it, freeing up the memory space allocated to that object without affecting objects that are still being used.

How does garbage collection work?

Each programming language implements GC differently, but they are similar in that the process occurs automatically behind the scenes. Some GC-enabled languages provide controls for launching the GC process or for optimizing its behavior, but the implementation of controls varies from one language to the next. In some cases, GC capabilities can be added to a language through a library or module.

Garbage collection is a fairly common component in most modern programming languages. Despite its benefits, however, garbage collection can have a negative impact on performance. Garbage collection is an ongoing process that requires central processing unit resources, which can affect an application's general performance or even disrupt its operations. For this reason, some developers still debate GC's benefits, believing that they can better control memory deallocation than an automated process.

Overall, however, GC is accepted as an important and necessary asset. Given that it's built into most modern programming languages, it's the only approach to memory management that many developers know. At the same time, the languages themselves continue to improve their GC capabilities, using different techniques to mitigate the impact on performance.

One common technique is to divide the heap's memory space into three generations based on an object's longevity:

- All newly created objects start in the first generation. Short-lived objects, such as temporary variables, never make it out of this generation. The garbage collector identifies them and removes them, freeing up their allocated memory space.
- Objects that survive the GC process in the first generation move to the second generation. When the GC process runs at this level, the garbage collector again identifies and removes any objects in this generation that are no longer needed. Garbage collection at this level is usually performed less often than for first-generation objects.
- The third generation contains the long-lived and permanent objects that survive the second-generation GC process. Objects remain at this level until they are no longer needed. Garbage collection at this level tends to be performed the least often.