**What languages use garbage collection?**

Lisp has used garbage collection since John McCarthy invented it in 1958. Java, Scala, Python, and .NET/C# are all popular GC languages. Additional garbage collection languages include the relatively young Go, Ruby, D, OCaml, and Swift, as well the older languages Eiffel, Haskell, ML, Modula-3, Perl, Prolog, Scheme, and Smalltalk.

Java, Python, and .NET/C# are some of the more popular programming languages that implement garbage collection. The [Java virtual machine (JVM)](https://www.infoworld.com/article/3272244/what-is-the-jvm-introducing-the-java-virtual-machine.html) actually provides four different garbage collectors: serial, parallel, concurrent mark and sweep, and [G1GC](https://www.oracle.com/technical-resources/articles/java/g1gc.html), the garbage first garbage collector. G1GC is now the default in Java; it is a regionalized and generational parallel compacting collector that achieves soft real-time goals.

Python, specifically the standard CPython implementation, [combines reference counting with three-level generational collection](https://devguide.python.org/internals/garbage-collector/) that only focuses on cleaning container objects. The .NET CLR (common language runtime) uses a [three-level generational mark and compact collection algorithm](https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals). The CLR also segregates memory objects into two heaps, one for large objects (85,000 bytes or higher) and one for small objects; the large object heap usually isn’t compacted, just marked and swept, but can be compacted if necessary.

**What languages do not use garbage collection?**

C and C++,

**How to make operating system control garbage collection?**

For a majority of the objects that your app creates, you can rely on the [.NET garbage collector](https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/) to handle memory management. However, when you create objects that include unmanaged resources, you **must** explicitly release those resources when you finish using them. The most common types of unmanaged resources are objects that wrap operating system resources, such as files, windows, network connections, or database connections. Although the garbage collector is able to track the lifetime of an object that encapsulates an unmanaged resource, it doesn't know how to release and clean up the unmanaged resource.

If your types use unmanaged resources, you should do the following:

* Implement the [dispose pattern](https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/implementing-dispose). This requires that you provide an [IDisposable.Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) implementation to enable the deterministic release of unmanaged resources. A consumer of your type calls [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) when the object (and the resources it uses) are no longer needed. The [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) method immediately releases the unmanaged resources.
* In the event that a consumer of your type forgets to call [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose), provide a way for your unmanaged resources to be released. There are two ways to do this:
  + Use a safe handle to wrap your unmanaged resource. This is the recommended technique. Safe handles are derived from the [System.Runtime.InteropServices.SafeHandle](https://learn.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.safehandle) abstract class and include a robust [Finalize](https://learn.microsoft.com/en-us/dotnet/api/system.object.finalize) method. When you use a safe handle, you simply implement the [IDisposable](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable) interface and call your safe handle's [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.safehandle.dispose) method in your [IDisposable.Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) implementation. The safe handle's finalizer is called automatically by the garbage collector if its [Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) method is not called.

—**or**—

* + Define a [finalizer](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/finalizers). Finalization enables the non-deterministic release of unmanaged resources when the consumer of a type fails to call [IDisposable.Dispose](https://learn.microsoft.com/en-us/dotnet/api/system.idisposable.dispose) to dispose of them deterministically.