ARC

ARC is an acronym for automatic reference counter. ARC only works for reference types, not value types at all. It also specifically works for class instances. It is used to track and manage the apps memory. In other languages, memory has to be manually allocated and deleted, however with ARC, it is done automatically. When an instance of a class is created, a block of memory is allocated for the instance. The values of any stored properties are stored in the allocated memory block. ARC remembers how many properties, constants and variables are referring to the class instance. If this does not happen, it is probable that the app will crash. When the instance is created, the properties, constants and variables are created with strong references so the memory is not deallocated when it could potentially be needed still. Every instance of a class created adds to a reference counter to keep track of when to deallocate the instance. When the reference counter hits 0, the memory is freed. When creating a weak instance, it does not add or decrement the reference counter. So, when the counter hits zero, even if the weak instance is still in use, the memory will be deallocated and the weak instance will reference nothing. There is also an unowned reference type, which is similar to the weak reference such that they both dont increment the reference counter. The difference between the 2 is, a weak reference is an optional, either points to the allocated memory, or nil. The unowned always has to have a value, so will never point to nil. This is created for the purpose of having the same lifetime or shorter lifetime of the strong reference type.

Some pros of ARC are it does real time, deterministic destruction of objects as they go from used to unused. It is more efficient for use in low power systems, like cellular devices, because it does not do any background processing. One con to ARC is that it does not do well with retain cycles. When multiple objects reference each other, ARC does not do well at deallocating the objects when the references go out of scope.

Garbage collection works well with retain cycles because garbage collection looks at the “reachable” objects. So, when a reference goes out of scope, and no outside reference exists, it can detect that it is no longer reachable and collect that garbage. Garbage collection uses more memory, and can lead to poor performance if there is a tight limit on the amount of memory available. ARC determines the memory allocation at compile time, which causes the developers to think a bit about the types of reference modifier needed. However, this allows for there to be no overhead during run time.