# Closures and Weak/Strong References - Simeon Savov

In Swift, a closure is a self-contained block of functionality that can be passed around and used in your code. They can store references to any constants and variables from the context in which they are defined, known as “*closing over”* said constants and variables. Closures are similar to functions but you don’t need to name them and can pass them around as arguments to other functions. For example, a closure typically looks like this:

{ (parameters) -> return type in

statements

}

Closures are useful for instances when you want to pass a block of code which gets executed as part of another function’s execution. Such as animations, sorting, API designs, asynch calls to make sure UI’s load properly, etc.

Strong references are essentially normal references which protect any referred objects from being deallocated by Apple’s ARC (Automatic Reference Counting) memory management system. Strong references are used often in Swift, for example, the declaration of a property with “let” is strong by default. It is generally safe to use strong references when the object relationship hierarchies are linear, such as from parent to child.

Weak references, however, are references that do not protect objects from deallocation by the ARC functionality. Additionally, a weak reference zeroes out the pointer to the object when it successfully deallocates, ensuring that the weak reference will either be a valid object or nil. In Swift, all weak references are mutable non-constants like “var” because the reference can and will be changed to nil when there is no longer anything holding a strong reference to it.