# Reference & Value Types

Within the Swift programming language, there are various reference and value types that may allocate memory differently and possess different functionalities.

**Reference types** in swift consist of functions, closures, and classes. This type is typically defined in classes and are able to have instances share a single copy of data. Copying references implicitly creates shared instances which lets the data copied alter the original data. In programs, reference types are usually used when creating a shared mutable state or comparing instance identity. In Objective C, reference types can be represented by class instances that are handled by utilizing reference semantics. The problem with reference types is that it enables implicit data sharing. To dive even deeper in reference types, when reference type variables are declared, the system then allocates a piece of memory space in RAM that then stores the memory address of the object into the variable. Values that are reference types are stored in the system's heap.

**Value types** in swift consist of int, double, string, array, dictionaries, enums, structs, tuples, and sets. In contrast to reference types, value types create unique copies of data in enums, structs, & tuples. This is what we'd call an independent instance, unreliant on the original data and storing the data in its own memory allocation. Swift seems to much rather use value types as much of its components are already value types. Value types are especially useful when it comes to customizations because they let you choose which variables you wish to modify, but an issue that arises using value types is that copies of values are cheap and run in constant time. You can store a value into a variable, set it equal to another variable, and still be able to modify the first variable without affecting the copy you made. Both variable values are independently stored in their respective memory spaces. Value types themselves are stored in the stack rather than the heap like reference types.