**What each is (reference and value type)**

A Value Type stores its contents in memory allocated on the stack.

When I created a Value Type, a single space in memory is allocated to store the value and that variable directly holds a value. If I assign it to another variable, the value is copied directly and both variables work independently.

If value got changed in the one method, it wouldn't affect the variable in another method. For example Predefined datatypes, structures, int, float, double, struct, enum. It can be created at compile time and Stored in stack memory, because of this, Garbage collector can't access the stack.

Reference Type contains a pointer to another memory location that holds the data, it is used by a reference which holds a reference (address) to the object but not the object itself. Because reference types represent the address of the variable rather than the data itself, assigning a reference variable to another doesn't copy the data. Instead it creates a second copy of the reference, which refers to the same location of the heap as the original value. Reference Type variables are stored in a different area of memory called the heap. This means that when a reference type variable is no longer used, it can be marked for garbage collection. For example Classes, Objects, Arrays, Indexers, Interfaces etc.

Reference type copies memory address and modify the data in that address

**How are they different**

* A value type holds a data value within its own memory space.
* A reference type holds a pointer to another memory location that holds the data.
* In value type, the value is copied to the new location, so there are two identical copies of the same value in the memory.
* In reference type, the reference is copied while the actual value remains the same.
* A value type is stored in the stack
* A reference type is stored on the heap.

**How they are utilized in iOS, swift and Objective-C**

In swift “value types”, where each instance keeps a unique copy of its data, usually defined as a **struct, enum, or tuple.** The second, “reference types”, where instances share a single copy of the data, and the type is usually defined as a **class**.

In Objective-C, compound data types are represented using ***@class*** instances, which are handled using reference semantics. This means that when an instance is assigned or passed as a parameter to a function, the memory allocation represented by the object instance is not copied or duplicated, and can be changed, or mutated, by operations performed on any reference to the instance.

In Objective-C, we can use **NSLog(@"%p", variable)** to print out the memory address. Reference type

Use a ***value type*** when you want copies to have an independent state, and the data will be used in code across multiple threads.

Use a ***reference type*** when comparing instance identity with === makes sense. === checks if two objects share the same memory address.

**Pros and Cons to either**

Pros

* independent state in value type
* share the same memory address reference type
* too many points with the same reference type

Cons

* memory space in value type
* the value change in value type when the same variable is used



