Differences of both structs and classes:

Property	Struct	Class
Type	Value type	Reference type
Copy behavior	Copied when passed or assigned	Passed around via pointers
Memory management	Copy when write	Automatically reference counted
Programming paradigm	Functional programming	Object-oriented programming
Inheritance	Not supported	Supported
Free init	Initializes all variables	Initializes no variables
Mutability	Explicit (var vs let)	Always mutable
Go-to data structure	Yes	Used in specific circumstances

Value type vs reference type

Structs are value types, which means that when they are copied or passed to a function, a new copy of the struct is created. Classes are reference types, which means that when they are copied or passed to a function, a reference to the class is copied instead of the class

itself.

Copied when passed or assigned vs passed around via pointers

When a struct is passed to a function or assigned to a variable, a new copy of the struct is created. This means that the original struct and the new copy are two separate objects. When a class is passed to a function or assigned to a variable, a reference to the class is copied instead of the class itself. This means that the original class and the new reference refer to the same object.

Copy when write vs automatically reference counted

Swift uses a memory management technique called automatic reference counting (ARC) to manage the memory of class instances. ARC keeps track of how many references there are to a class instance and deallocates the instance when there are no more references to it. Structs use a memory management technique called copy when write. This means that when a struct is modified, a new copy of the struct is created and the old struct is left untouched.

Functional programming vs object-oriented programming

Structs are well-suited for functional programming because they are immutable and can be copied easily. Classes are well-suited for object-oriented programming because they can inherit from other classes and their properties and methods can be modified.

No inheritance vs inheritance

Structs cannot inherit from other structs. Classes can inherit from other classes and inherit their properties and methods.

"Free" init initializes all variables vs "free" init initializes no variables

The free init (required init?()) initializer for a struct initializes all of the struct's properties. The free init initializer for a class does not initialize any of the class's properties.

Mutability is explicit (var vs let) vs always mutable

The mutability of a struct's property is explicit and is controlled by the var or let keyword. The properties of a class are always mutable.

Your "go to" data structure vs used in specific circumstances

Structs are a good go-to data structure because they are lightweight, immutable, and can be

copied easily. Classes are used in specific circumstances, such as when you need to inherit from another class or when you need to store a mutable object.

In general, structs are a good choice for representing small, immutable pieces of data. Classes are a good choice for representing complex data structures or mutable objects.