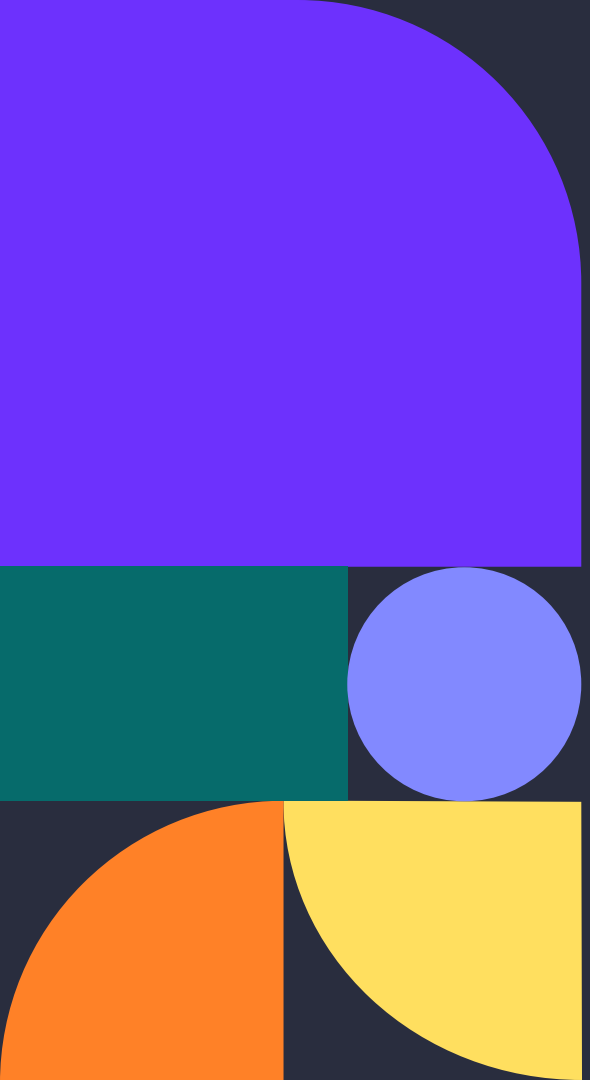

Classes vs Structs

Structs

What is a Struct?

1. **A Struct is a type, just like “String”, “Color”, or “Double” (Which are just Structs pre-made by Apple).**

2. **A Struct is like a blueprint. Instructions and guidelines to make an object, not an object itself.**

3. **A Struct consists of both type properties and type methods, which are a mix of Variables, Constants, and Functions used as the building blocks for your object.**

Classes

What is a Class?

1. **A Class is the same as a Struct in that it is also a type, or a blueprint for an object.**

2. **A class will also contain type properties and methods.**

3. **You can make a new instance of class anytime using the building blocks given to you in the class, just like a struct.**

So why have both?

There are a few ways that Structs and classes differ, the main two being:

1. **Inheritance**

2. **Reference vs Value**

Inheritance

The first way that classes differ from structs is that classes have what is called inheritance. This means that you can create a subclass from your main class (superclass) which will inherit all of the properties, methods, or initializers from your superclass.

For example, in a video game, you might want to let your player choose what type of character they want to play as (Wizard, Rouge, Archer etc.). But each of these characters will still be a human with human traits (name, age, height etc.). So you could make a class called "Human" with all the traits you want all humans to have, and then you could make a subclass of human call "Wizard". This new "Wizard" subclass would still have everything that "human" has, but you can add any properties or methods that you want the "Wizard" class to have.

Reference vs Value

The other key difference between Structs and Classes is the kind of type that they are.

Value: Structs are what's known as a value type. This means that every time that an instance of this type is being used somewhere else, it is being copied and a new version of that instance is being made. So if you were to change one of the values on one of those instances, it would not affect the other one in any way.

Reference: Classes are the opposite and what's known as a reference type. This means that every new instance is just pointing to an object in memory, not making a new object. So if one of these references where to change a value on that object, It would change that same value for every other reference to that object.

Other Differences and Notes

1. **Initializers. Structs have built in initializers for when you don't specify a default value. These are known as memberwise initializers and are not included with classes. All initializers on classes must be declared or you will get a compiler error.**

2. **The "mutating" keyword. Structs require that "mutating" must be included on any type method that will be changing any properties on that struct.**

3. **Using a value type is safer in general, so Swift recommends using a value type first and changing to a reference type if needed.**