STRUCT

Struct is a custom data type. It combines one or more variables into a single type

Initializers

All Swift types have initializers that similar to a function that returns a new instance of the type

Example:

```
var string = String.init() // ""
var integer = Int.init() // 0

OR

var string = String() // ""
var integer = Int() // 0
```

Types of Initializers

- Default Initializer — Default is `init()`
- memberwise initializers — memberwise initializer are created by Swift that includes all properties of a structure. It is useful for creating new instance with a non-default value

Example:

```
struct Car {
  var make: String
  var year: Int
  var color: Color
  var topSpeed: Int
}

let firstCar = Car(make: "Honda", year: 2010,
color: .blue, topSpeed: 120)
// Memberwise initializer
```

- custom initializers — Defining your own initializer. All properties must be set to initial values before completing initialization.

Example:

```
struct Temperature {
  var celsius: Double
  var kelvin: Double

  init(celsius: Double) {
    self.celsius = celsius
    kelvin = self.celsius + 273.15
  }

  init(fahrenheit: Double) {
    celsius = (fahrenheit - 32) / 1.8
    kelvin = celsius + 273.15
  }
}

let currentTemp = Temperature(celsius: 18.5)
let boiling = Temperature(fahrenheit: 212.0)
```

Computed properties

Swift allows a property to perform logic that returns a calculated value

Example:

```
struct Temperature {
  var celsius: Double

  var fahrenheit: Double {
    return celsius * 1.8 + 32
  }

  var kelvin: Double {
    return celsius + 273.15
  }
}

let currentTemperature = Temperature(celsius: 0.0)
print(currentTemperature.fahrenheit)
print(currentTemperature.kelvin)
```

Mutating methods

update the property values of a structure within an instance method

Example:

```
struct Odometer {
  var count: Int = 0 // Assigns a default value to the
`count` property.

  mutating func increment() { //complier error without
mutating word
    count += 1
  }

  mutating func increment(by amount: Int) {
    count += amount
  }

  mutating func reset() {
    count = 0
  }
}
```

Type property & method

Type property or method can be accessed with an instance of that type

Use prefix `static` before property or method for declaring that as Type property or method

Example:

```
struct Temperature {
  static var boilingPoint = 100
}

print("Boiling temp = " + Temperature.boilingPoint))
```

See there is no instance of type Temperature created.

Definition syntax

`name` is variable

`sayHello()` Is method

Example:

```
struct Person {
  var name: String
  func sayHello() {
    print("Hello, there! My name is \(name)!")
  }
}
```

Instances

Declare a variable of type `Person`
Use . (dot) convention to access its variables and methods

Example:

```
let person = Person(name: "Jasmine")
print(person.name)
person.sayHello()
```

Property Observer

Swift allows to observe any property and respond to the changes in the property's value. These property observers are called every time a property's value is set

`willSet` will be called just before value is assigned

`didSet` will be called after value is assigned

Example:

```
struct StepCounter {
    var totalSteps: Int = 0 {
        willSet {
            print("About to set totalSteps to \(newValue)")
        }
        didSet {
            if totalSteps > oldValue  {
                print("Added \(totalSteps - oldValue) steps")
            }
        }
    }
}
```