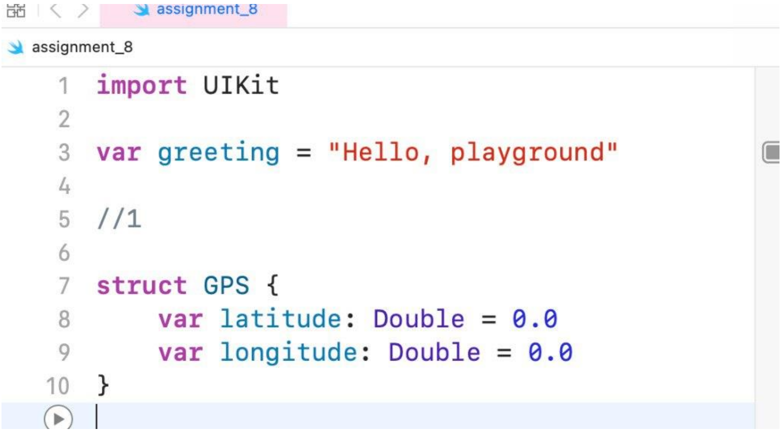
**Name- Khushi Nitinkumar Patel**

**PRN- 2020BTECS00037**

**Batch- T2**

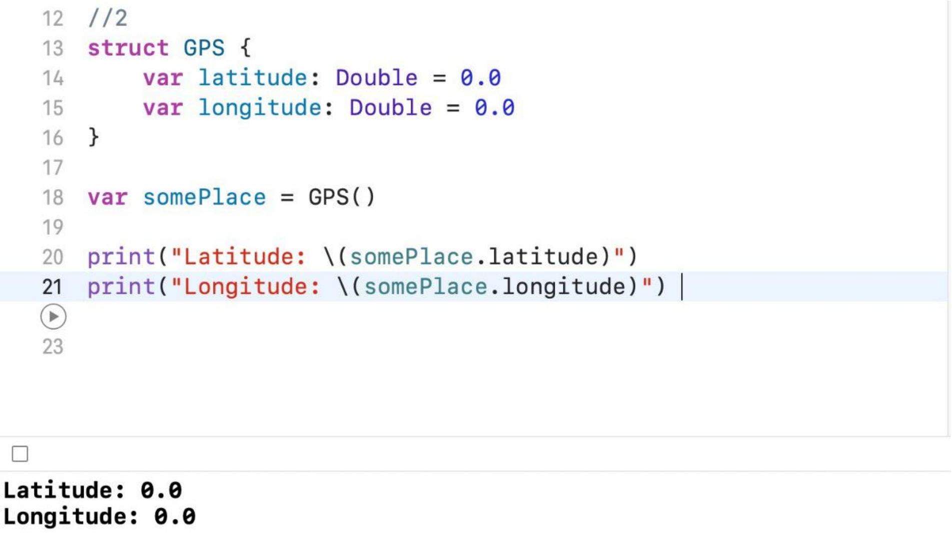
# Exercise - Structs, Instances, and Default Values

1. Imagine you are creating an app that will monitor location. Create a GPS struct with two variable properties, latitude and longitude, both with default values of 0.0.



```swift
import UIKit

var greeting = "Hello, playground"

//1

struct GPS {
    var latitude: Double = 0.0
    var longitude: Double = 0.0
}
```

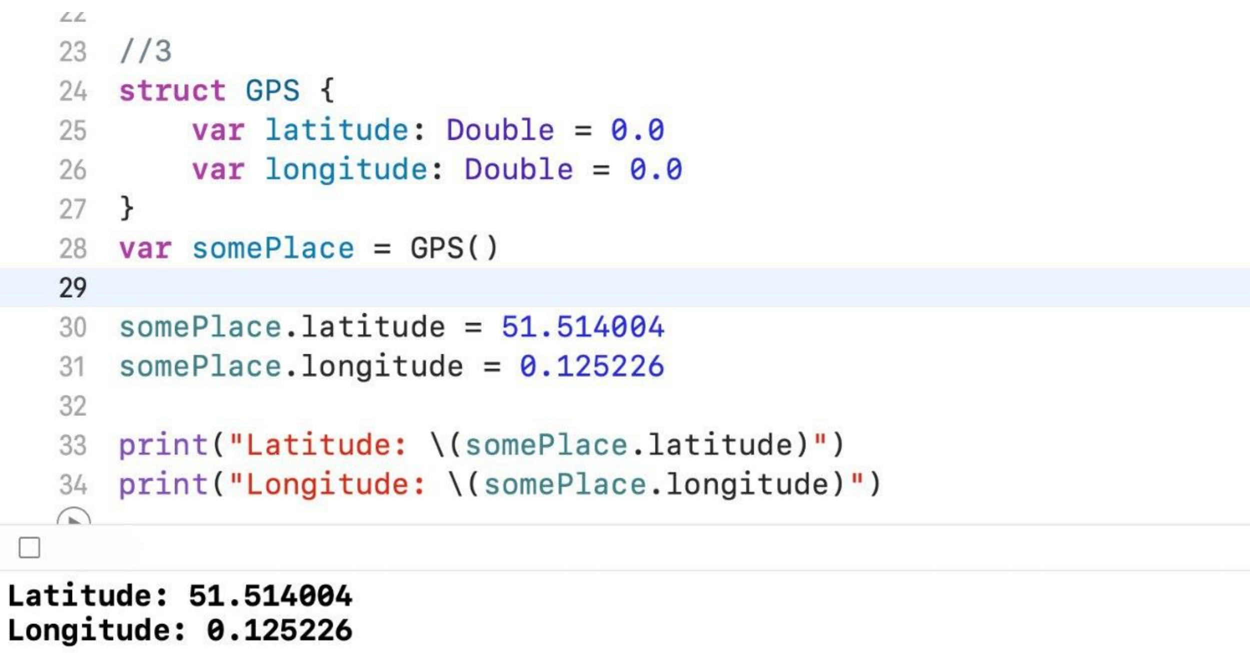
2. Create a variable instance of GPS called somePlace. It should be initialized without supplying any arguments. Print out the latitude and longitude of somePlace, which should be 0.0 for both.



```swift
//2
struct GPS {
    var latitude: Double = 0.0
    var longitude: Double = 0.0
}

var somePlace = GPS()

print("Latitude: \(somePlace.latitude)")
print("Longitude: \(somePlace.longitude)")
```

```
Latitude: 0.0
Longitude: 0.0
```

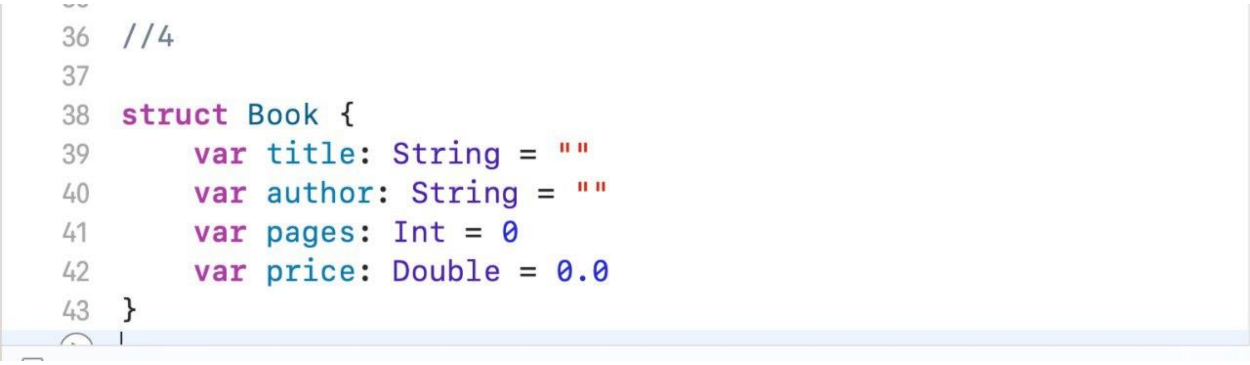
3. Change somePlace's latitude to 51.514004, and the longitude to 0.125226, then print the updated values.



```
//3
struct GPS {
    var latitude: Double = 0.0
    var longitude: Double = 0.0
}
var somePlace = GPS()

somePlace.latitude = 51.514004
somePlace.longitude = 0.125226

print("Latitude: \(somePlace.latitude)")
print("Longitude: \(somePlace.longitude)")
```

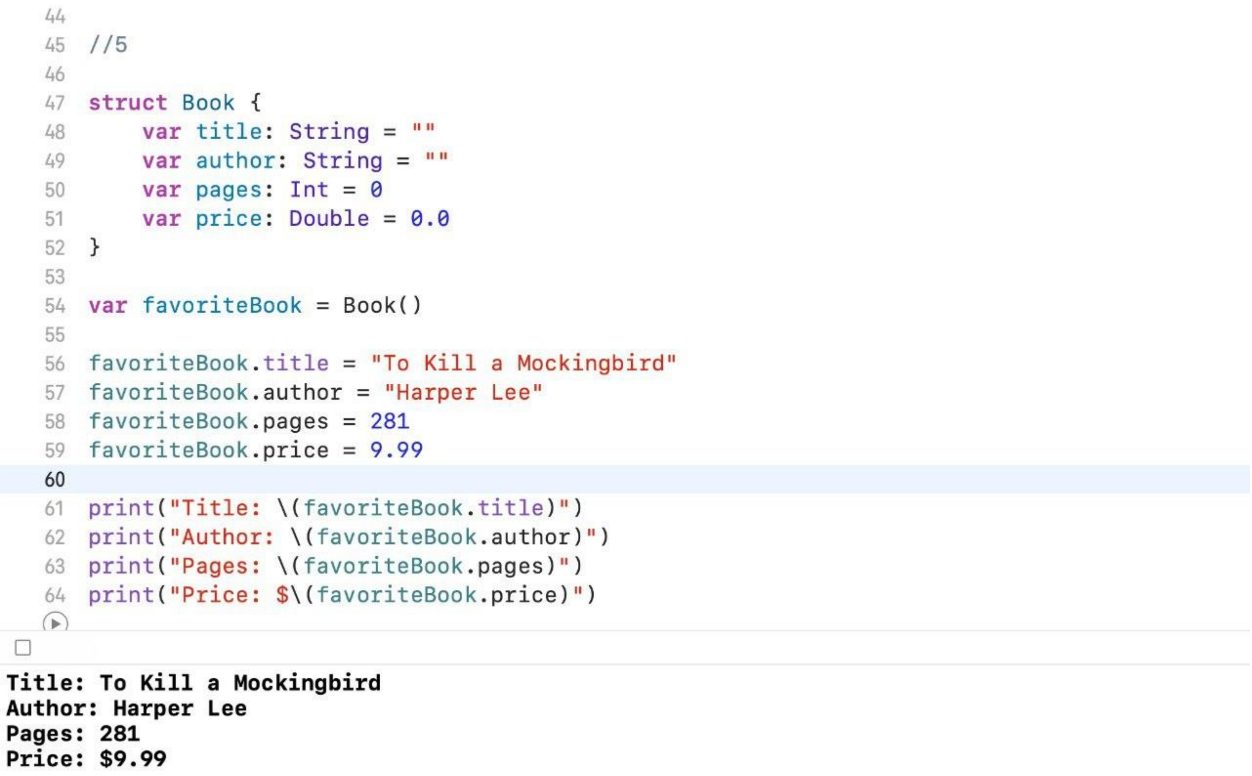
```
Latitude: 51.514004
Longitude: 0.125226
```

4. Now imagine you are making a social app for sharing your favorite books. Create a Book struct with four variable properties: title, author, pages, and price. The default values for both title and author should be an empty string. pages should default to 0, and price should default to 0.0.



```
//4

struct Book {
    var title: String = ""
    var author: String = ""
    var pages: Int = 0
    var price: Double = 0.0
}
```

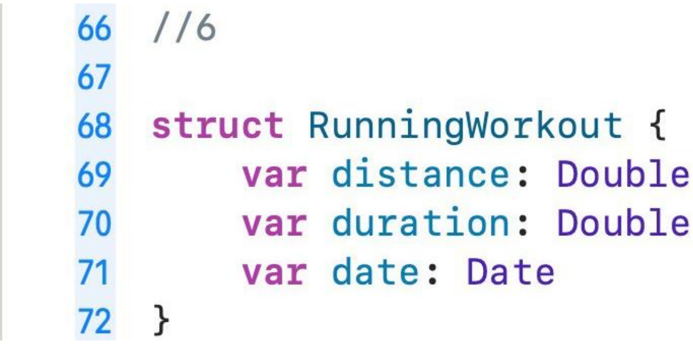
5. Create a variable instance of Book called favoriteBook without supplying any arguments. Print out the title of favoriteBook. Does it currently reflect the title of your favorite book? Probably not. Change all four properties of your favoriteBook to reflect your favorite book. Then, using the properties of favoriteBook, print out facts about the book.



```

//5

struct Book {
    var title: String = ""
    var author: String = ""
    var pages: Int = 0
    var price: Double = 0.0
}

var favoriteBook = Book()

favoriteBook.title = "To Kill a Mockingbird"
favoriteBook.author = "Harper Lee"
favoriteBook.pages = 281
favoriteBook.price = 9.99

print("Title: \(favoriteBook.title)")
print("Author: \(favoriteBook.author)")
print("Pages: \(favoriteBook.pages)")
print("Price: $\(favoriteBook.price)")
```

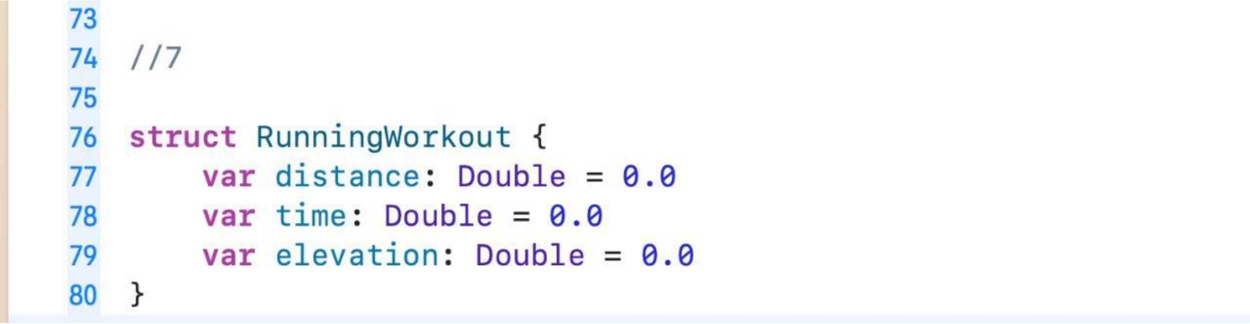
```
Title: To Kill a Mockingbird
Author: Harper Lee
Pages: 281
Price: $9.99
```

6. Your fitness tracking app wouldn't be much of a fitness tracker if it couldn't help users track their workouts. In order to track a user's run, you'll need to have some kind of data structure that can hold information about the workout. For the sake of simplicity, you'll focus specifically on running workouts.



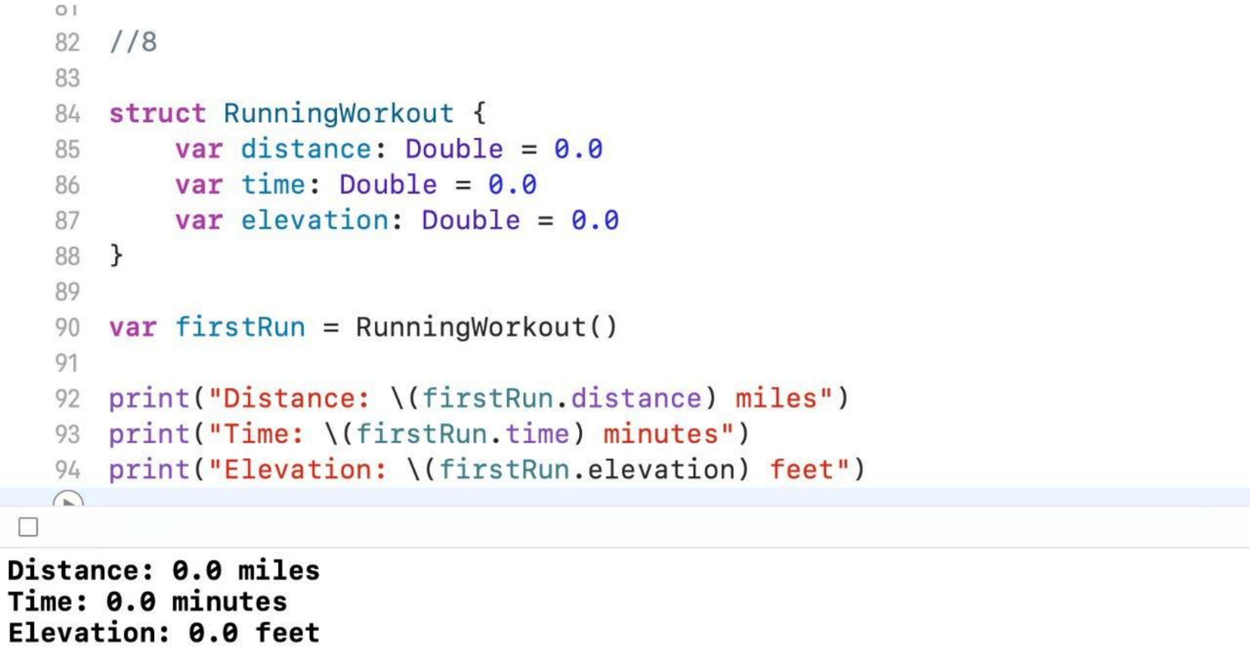
```
//6

struct RunningWorkout {
    var distance: Double
    var duration: Double
    var date: Date
}
```

7. Create a RunningWorkout struct. It should have variables properties for distance, time, and elevation. All three properties should have default values of 0.0.



```

//7

struct RunningWorkout {
    var distance: Double = 0.0
    var time: Double = 0.0
    var elevation: Double = 0.0
}
```

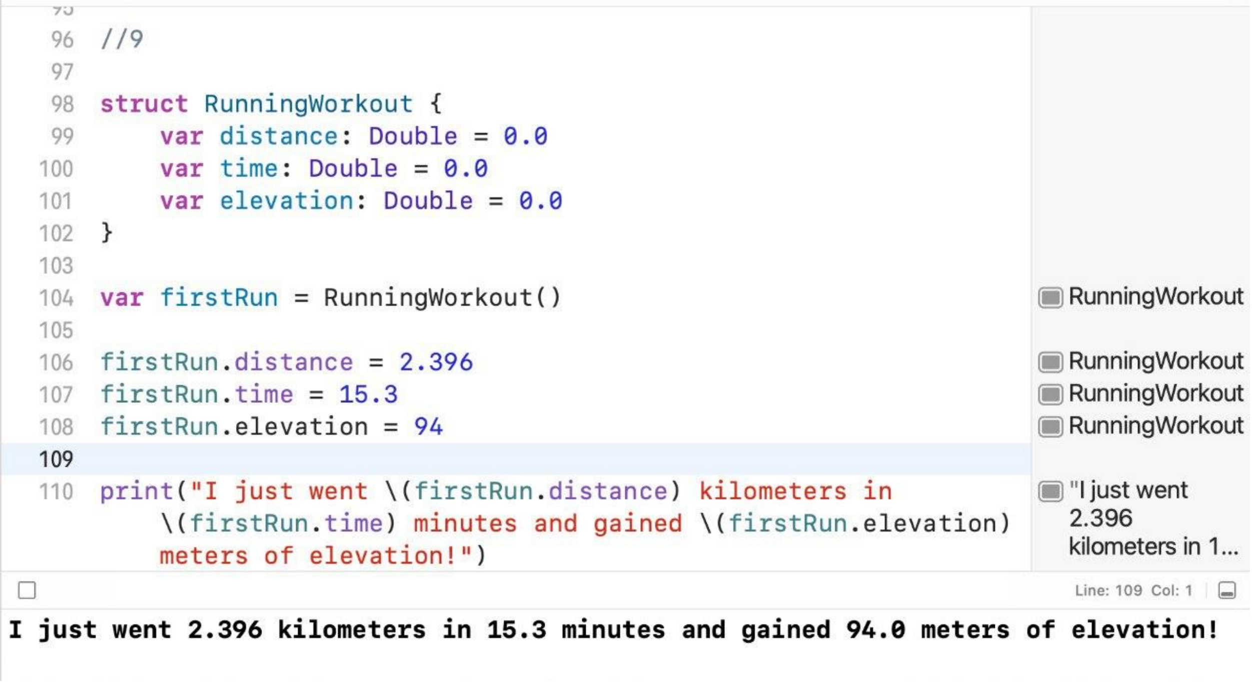
8. Create a variable instance of RunningWorkout called firstRun without supplying any arguments. Print out all three properties of firstRun. This is a good example of when using default values is appropriate, seeing as all running workouts start with a distance, time, and elevation change of 0.



```
//8

struct RunningWorkout {
    var distance: Double = 0.0
    var time: Double = 0.0
    var elevation: Double = 0.0
}

var firstRun = RunningWorkout()

print("Distance: \(firstRun.distance) miles")
print("Time: \(firstRun.time) minutes")
print("Elevation: \(firstRun.elevation) feet")
```

```
Distance: 0.0 miles
Time: 0.0 minutes
Elevation: 0.0 feet
```

9. Now imagine that throughout the course of the run, you go a distance of 2,396 meters in 15.3 minutes, and gain 94 meters of elevation. Update the values of firstRun's properties accordingly. Print a statement about your run using the values of each property.

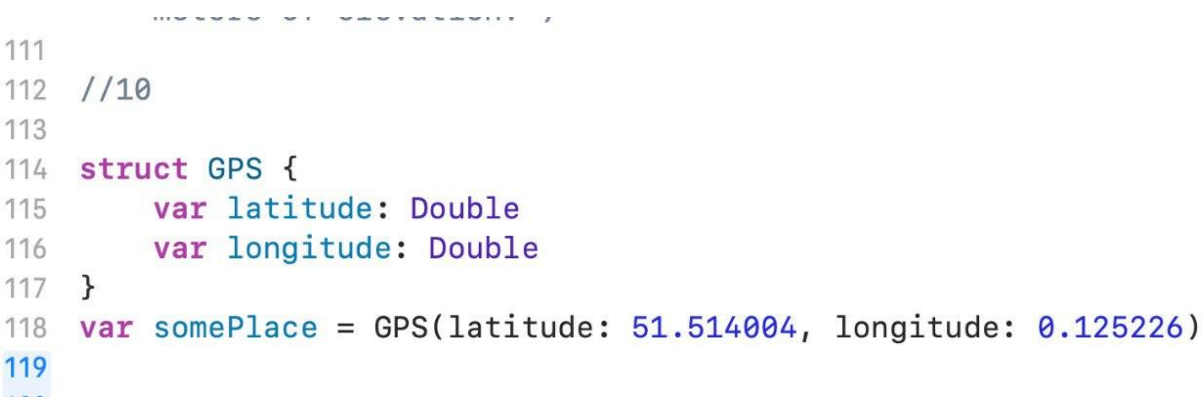


```swift
//9

struct RunningWorkout {
    var distance: Double = 0.0
    var time: Double = 0.0
    var elevation: Double = 0.0
}

var firstRun = RunningWorkout()

firstRun.distance = 2.396
firstRun.time = 15.3
firstRun.elevation = 94

print("I just went \(firstRun.distance) kilometers in
    \(firstRun.time) minutes and gained \(firstRun.elevation)
    meters of elevation!")
```

```
I just went 2.396 kilometers in 15.3 minutes and gained 94.0 meters of elevation!
```

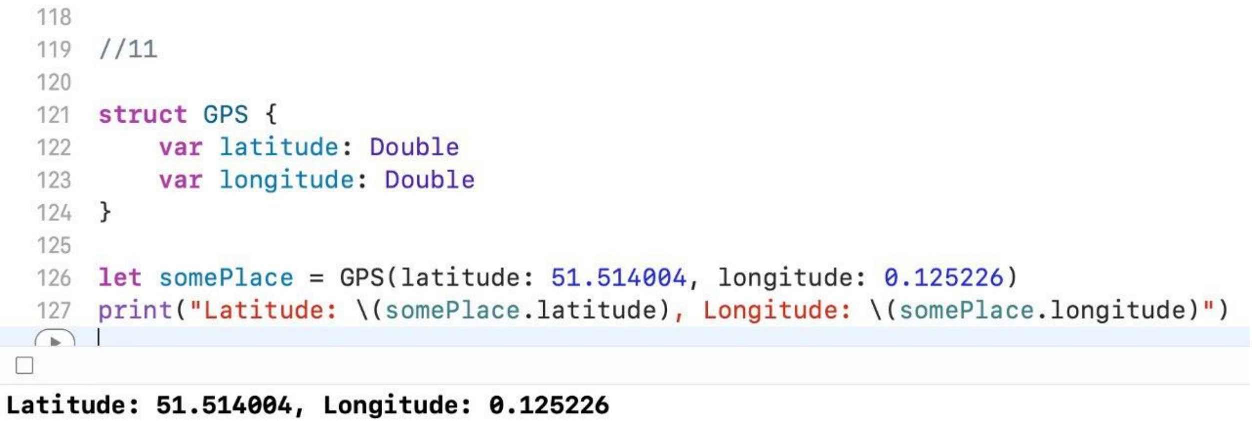
## Exercise - Memberwise and Custom Initializers

10. If you completed the exercise Structs, Instances, and Default Values, you created a GPS struct with default values for properties of latitude and longitude. Create your GPS struct again, but this time do not provide default values. Both properties should be of type Double.



```swift
    meters of elevation!")

//10

struct GPS {
    var latitude: Double
    var longitude: Double
}
var somePlace = GPS(latitude: 51.514004, longitude: 0.125226)
```

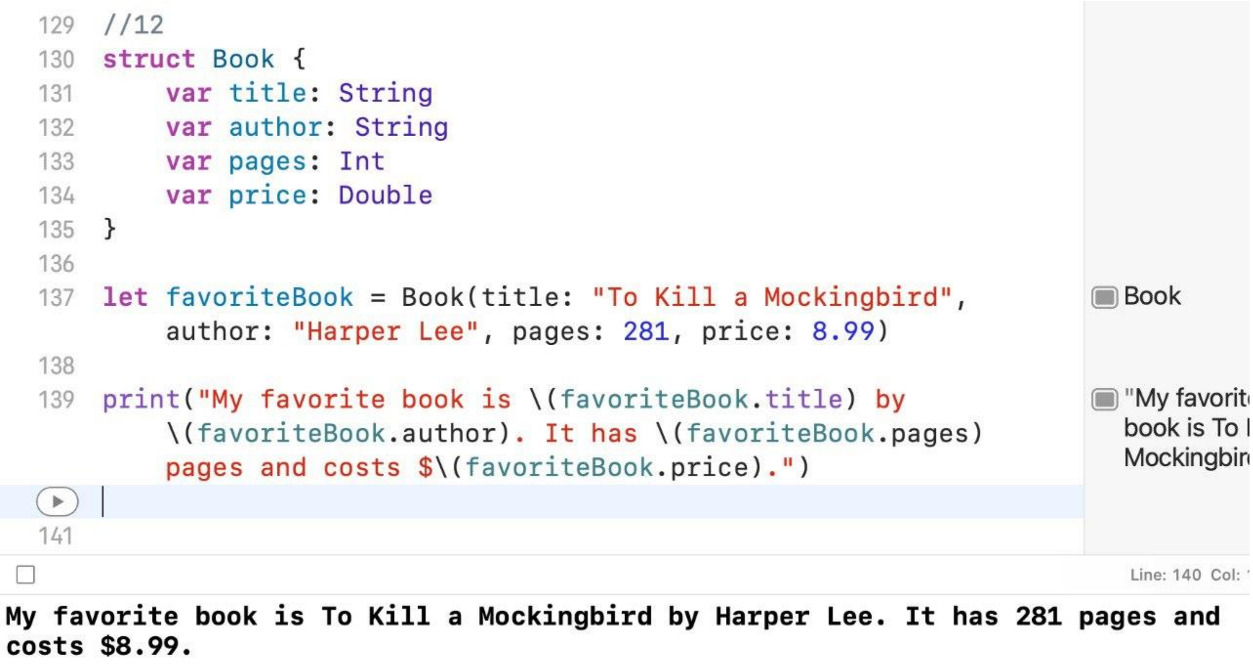
11. Now create a constant instance of GPS called somePlace, and use the memberwise initializer to set latitude to 51.514004, and longitude to 0.125226. Print the values of somePlace's properties.



```
//11

struct GPS {
    var latitude: Double
    var longitude: Double
}

let somePlace = GPS(latitude: 51.514004, longitude: 0.125226)
print("Latitude: \(somePlace.latitude), Longitude: \(somePlace.longitude)")
```

```
Latitude: 51.514004, Longitude: 0.125226
```

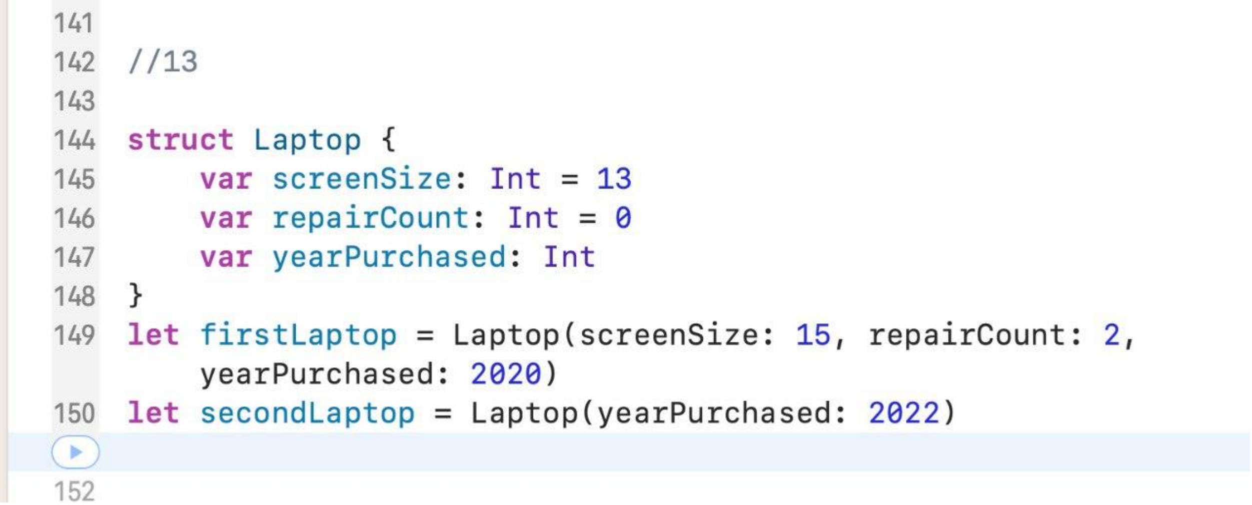
12. In Structs, Instance, and Default Values, you also created a Book struct with properties title, author, pages, and price. Create this struct again without default values. Give each property the appropriate type. Declare your favoriteBook instance and pass in the values of your favorite book using the memberwise initializer. Print a statement about your favorite book using favoriteBook's properties.



```
//12
struct Book {
    var title: String
    var author: String
    var pages: Int
    var price: Double
}

let favoriteBook = Book(title: "To Kill a Mockingbird",
    author: "Harper Lee", pages: 281, price: 8.99)

print("My favorite book is \(favoriteBook.title) by
    \(favoriteBook.author). It has \(favoriteBook.pages)
    pages and costs $\(favoriteBook.price).")
```

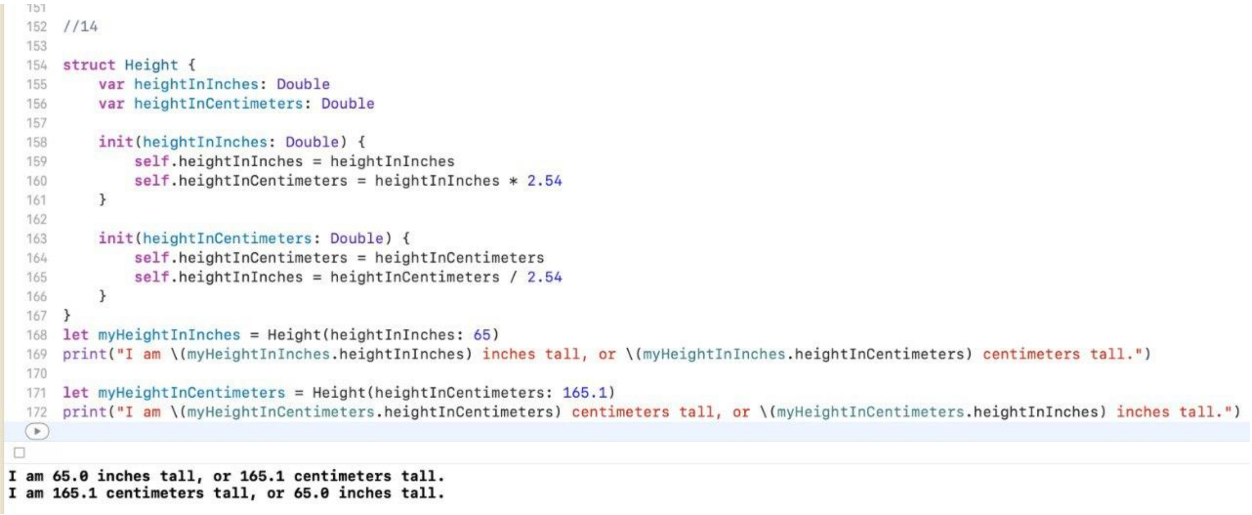
```
My favorite book is To Kill a Mockingbird by Harper Lee. It has 281 pages and
costs $8.99.
```

13. Make a Laptop struct with three variable properties, screenSize of type Int, repairCount of type Int, and yearPurchased of type Int. Give screenSize a default value of 13 and repairCount a default value of 0, and leave yearPurchased without a default value. Declare two instances of Laptop, using the two provided memberwise initializers.



```swift

//13

struct Laptop {
    var screenSize: Int = 13
    var repairCount: Int = 0
    var yearPurchased: Int
}
let firstLaptop = Laptop(screenSize: 15, repairCount: 2,
    yearPurchased: 2020)
let secondLaptop = Laptop(yearPurchased: 2022)


```

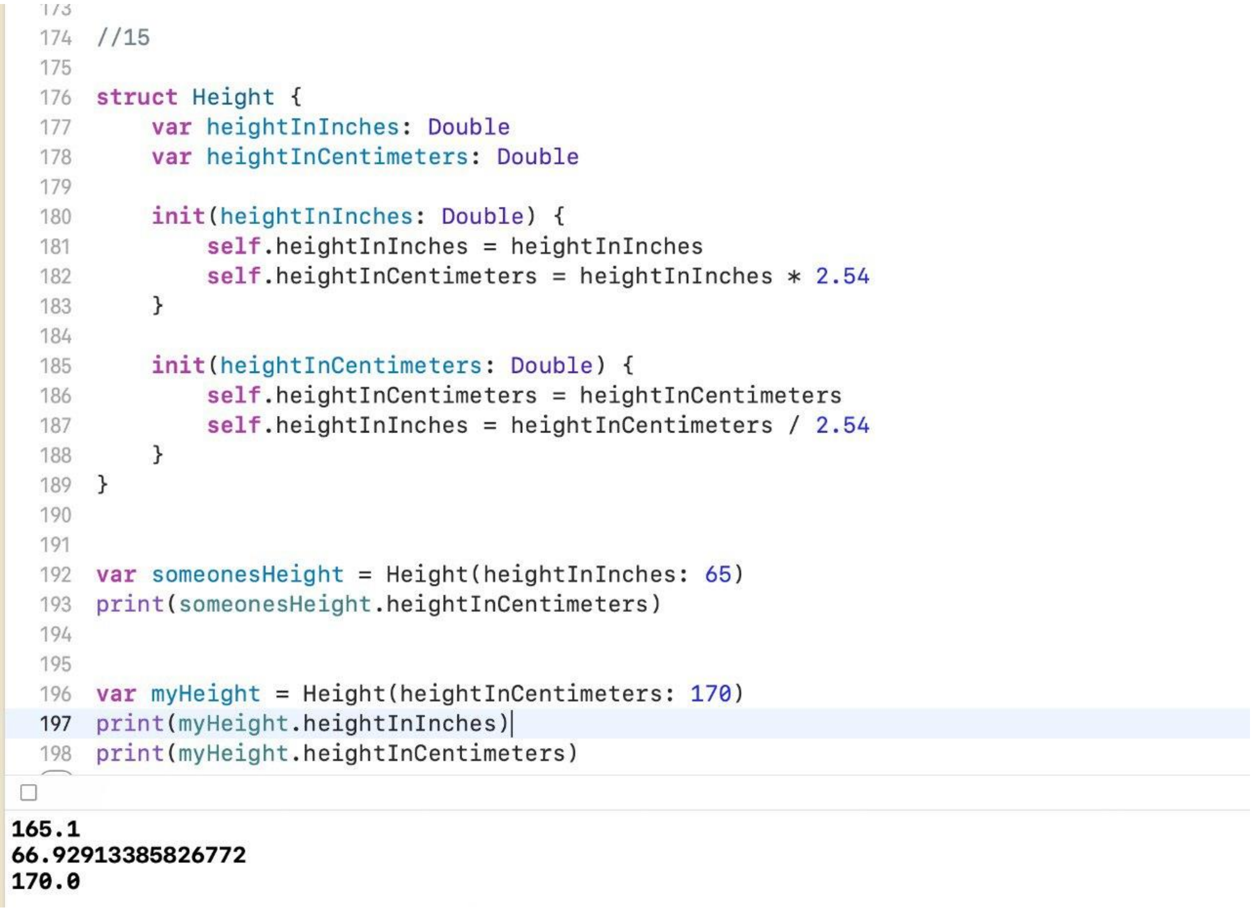
14. Make a Height struct with two variable properties, heightInInches and heightInCentimeters. Both should be of type Double.Create two custom initializers. One initializer will take a Double argument that represents height in inches. The other initializer will take a Double argument that represents height in centimeters. Each initializer should take the passed in value and use it to set the property that corresponds to the unit of measurement passed in. It should then set the other property by calculating the right value from the passed in value. Hint: *1 inch = 2.54 centimeters*. Example:If you use the initializer for inches to pass in a height of 65, the initializer should set heightInInches to 65 and heightInCentimeters to 165.1.



```swift

//14

struct Height {
    var heightInInches: Double
    var heightInCentimeters: Double

    init(heightInInches: Double) {
        self.heightInInches = heightInInches
        self.heightInCentimeters = heightInInches * 2.54
    }

    init(heightInCentimeters: Double) {
        self.heightInCentimeters = heightInCentimeters
        self.heightInInches = heightInCentimeters / 2.54
    }
}
let myHeightInInches = Height(heightInInches: 65)
print("I am \(myHeightInInches.heightInInches) inches tall, or \(myHeightInInches.heightInCentimeters) centimeters tall.")

let myHeightInCentimeters = Height(heightInCentimeters: 165.1)
print("I am \(myHeightInCentimeters.heightInCentimeters) centimeters tall, or \(myHeightInCentimeters.heightInInches) inches tall.")
```

```
I am 65.0 inches tall, or 165.1 centimeters tall.
I am 165.1 centimeters tall, or 65.0 inches tall.
```

15. Now create a variable instance of Height called someonesHeight. Use the initializer for inches to set the height to 65. Print out the property for height in centimeters and verify that it is equal to 165.1. Now create a variable instance of Height called myHeight and initialize it with your own height. Verify that both heightInInches and heightInCentimeters are accurate.



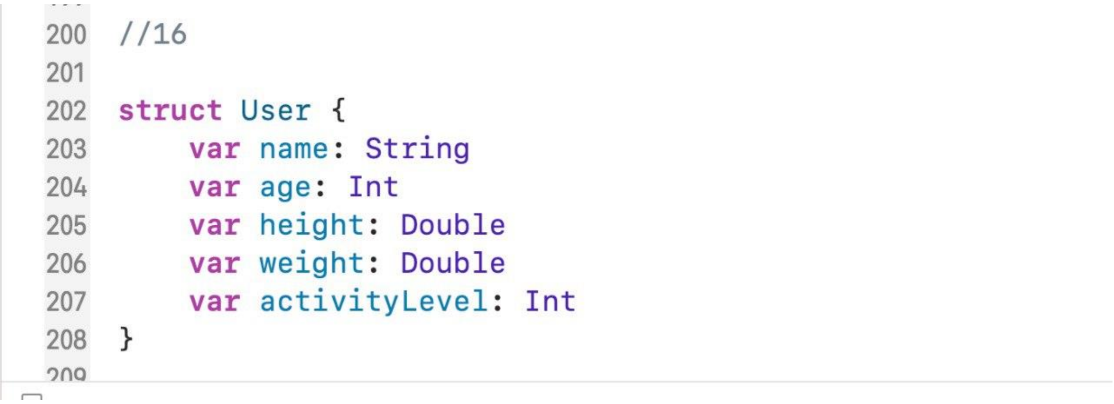
```swift
//15

struct Height {
    var heightInInches: Double
    var heightInCentimeters: Double

    init(heightInInches: Double) {
        self.heightInInches = heightInInches
        self.heightInCentimeters = heightInInches * 2.54
    }

    init(heightInCentimeters: Double) {
        self.heightInCentimeters = heightInCentimeters
        self.heightInInches = heightInCentimeters / 2.54
    }
}


var someonesHeight = Height(heightInInches: 65)
print(someonesHeight.heightInCentimeters)


var myHeight = Height(heightInCentimeters: 170)
print(myHeight.heightInInches)
print(myHeight.heightInCentimeters)
```

□

```
165.1
66.92913385826772
170.0
```

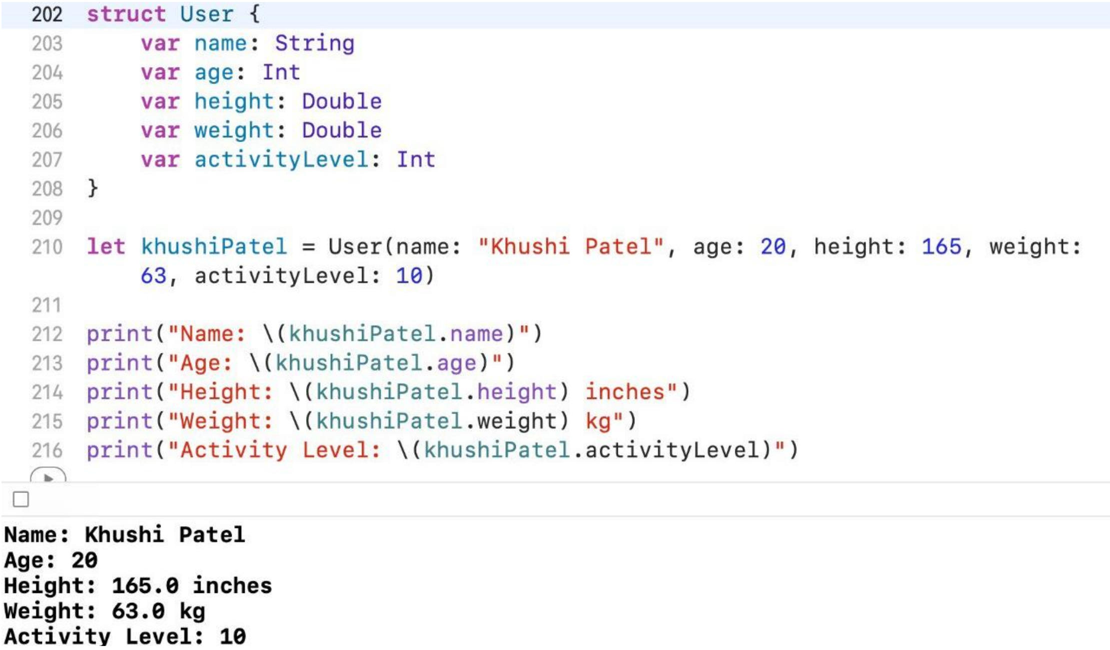
## Users and Distance

16. For most apps you'll need to have a data structure to hold information about a user. Create a User struct that has properties for basic information about a user. At a minimum, it should have properties to represent a user's name, age, height, weight, and activity level. You could do this by having name be a String, age be an Int, height and weight be of type Double, and activityLevel be an Int that will represent a scoring 1-10 of how active they are. Implement this now.



```swift
//16

struct User {
    var name: String
    var age: Int
    var height: Double
    var weight: Double
    var activityLevel: Int
}
```

17. Create a variable instance of User and call it your name. Use the memberwise initializer to pass in information about yourself. Then print out a description of your User instance using the instance's properties.



```swift
struct User {
    var name: String
    var age: Int
    var height: Double
    var weight: Double
    var activityLevel: Int
}

let khushiPatel = User(name: "Khushi Patel", age: 20, height: 165, weight:
    63, activityLevel: 10)

print("Name: \(khushiPatel.name)")
print("Age: \(khushiPatel.age)")
print("Height: \(khushiPatel.height) inches")
print("Weight: \(khushiPatel.weight) kg")
print("Activity Level: \(khushiPatel.activityLevel)")
```

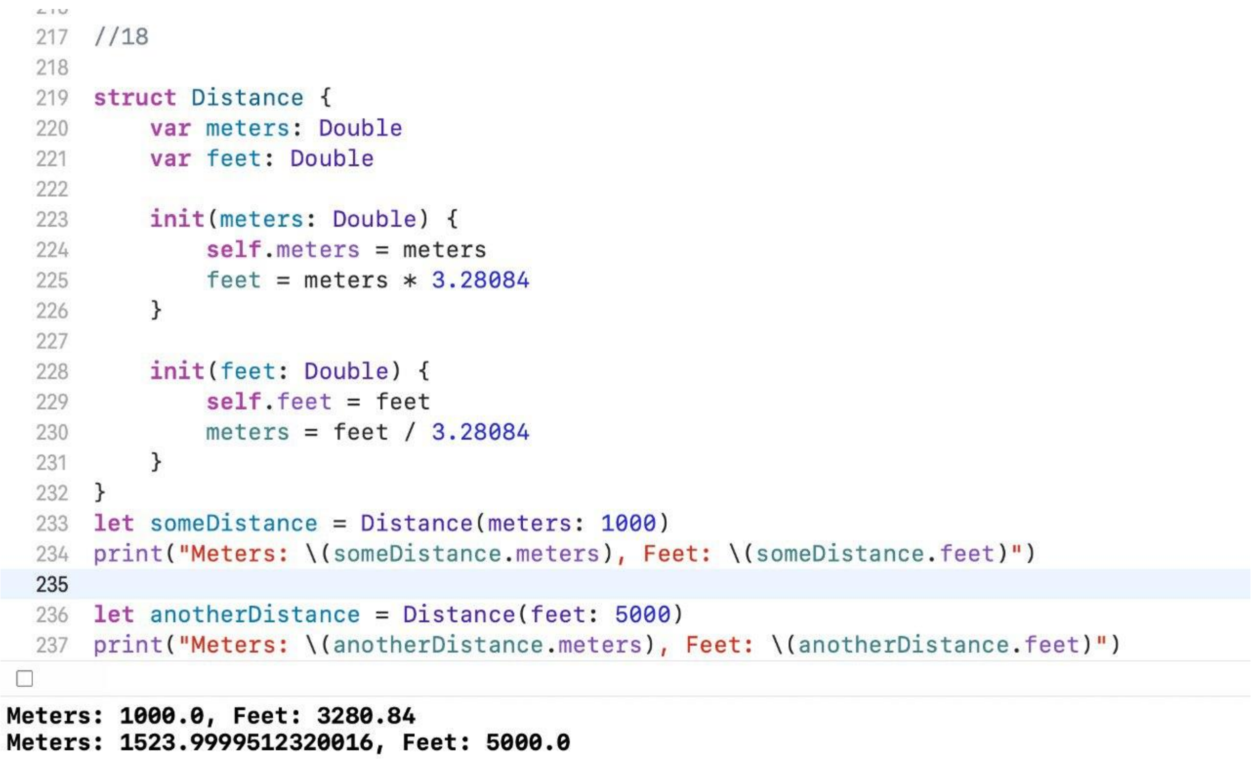
```
Name: Khushi Patel
Age: 20
Height: 165.0 inches
Weight: 63.0 kg
Activity Level: 10
```

18. In previous app exercises, you've worked with distance in the fitness tracking app example as a simple number. However, distance can be represented using a variety of units of measurement. Create a Distance struct that will represent distance in various units of measurement. At a minimum, it should have a meters property and a feet property. Create a custom initializer corresponding to each property (i.e. if you only have the two properties for meters and feet you will then have two initializers) that will take in a distance in one unit of measurement and assign the correct value to both units of measurements. Hint: *1 meter = 3.28084 feet*

    Example:
    If you use the initializer for meters and pass in a distance of 1600, the initializer should set meters to 1600 and feet to 5249.344.

    Now create an instance of Distance called mile. Use the initializer for meters to set the distance to 1600. Print out the property for feet and verify that it is equal to 5249.344.

    Now create another instance of Distance and give it some other distance. Ensure that both properties are set correctly.



```swift
//18

struct Distance {
    var meters: Double
    var feet: Double

    init(meters: Double) {
        self.meters = meters
        feet = meters * 3.28084
    }

    init(feet: Double) {
        self.feet = feet
        meters = feet / 3.28084
    }
}
let someDistance = Distance(meters: 1000)
print("Meters: \(someDistance.meters), Feet: \(someDistance.feet)")

let anotherDistance = Distance(feet: 5000)
print("Meters: \(anotherDistance.meters), Feet: \(anotherDistance.feet)")
```

```
Meters: 1000.0, Feet: 3280.84
Meters: 1523.9999512320016, Feet: 5000.0
```

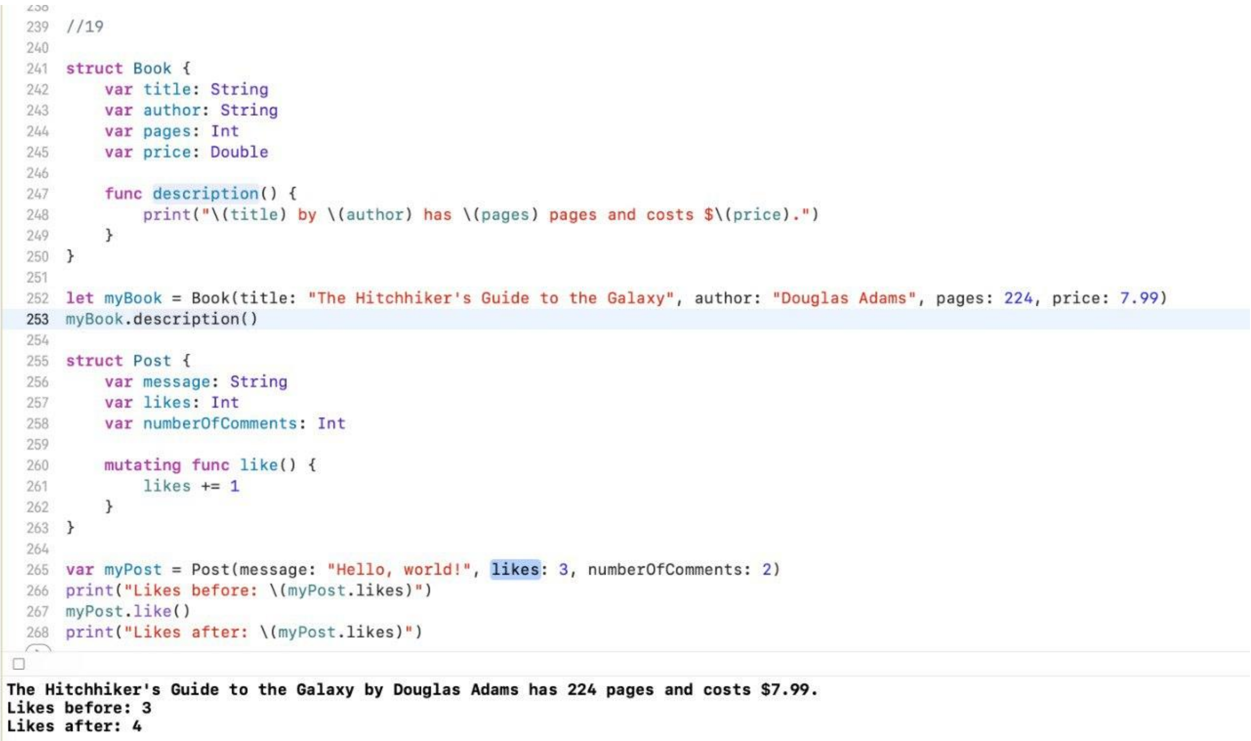
## Methods

19. A Book struct has been created for you below. Add an instance method on Book called description that will print out facts about the book. Then create an instance of Book and call this method on that instance.

```
 struct  Book
{
   var   title:   String
var   author:   String
var pages: Int     var
price: Double


}
```

A Post struct has been created for you below, representing a generic social media post. Add a mutating method on Post called like that will increment likes by one. Then create an instance of Post and call like() on it. Print out the likes property before and after calling the method to see whether or not the value was incremented.

```
 struct  Post  {        var
message: String     var
likes: Int
   var numberOfComments: Int
}
```



```
//19

struct Book {
    var title: String
    var author: String
    var pages: Int
    var price: Double

    func description() {
        print("\(title) by \(author) has \(pages) pages and costs $\(price).")
    }
}

let myBook = Book(title: "The Hitchhiker's Guide to the Galaxy", author: "Douglas Adams", pages: 224, price: 7.99)
myBook.description()

struct Post {
    var message: String
    var likes: Int
    var numberOfComments: Int

    mutating func like() {
        likes += 1
    }
}

var myPost = Post(message: "Hello, world!", likes: 3, numberOfComments: 2)
print("Likes before: \(myPost.likes)")
myPost.like()
print("Likes after: \(myPost.likes)")
```

```
The Hitchhiker's Guide to the Galaxy by Douglas Adams has 224 pages and costs $7.99.
Likes before: 3
Likes after: 4
```

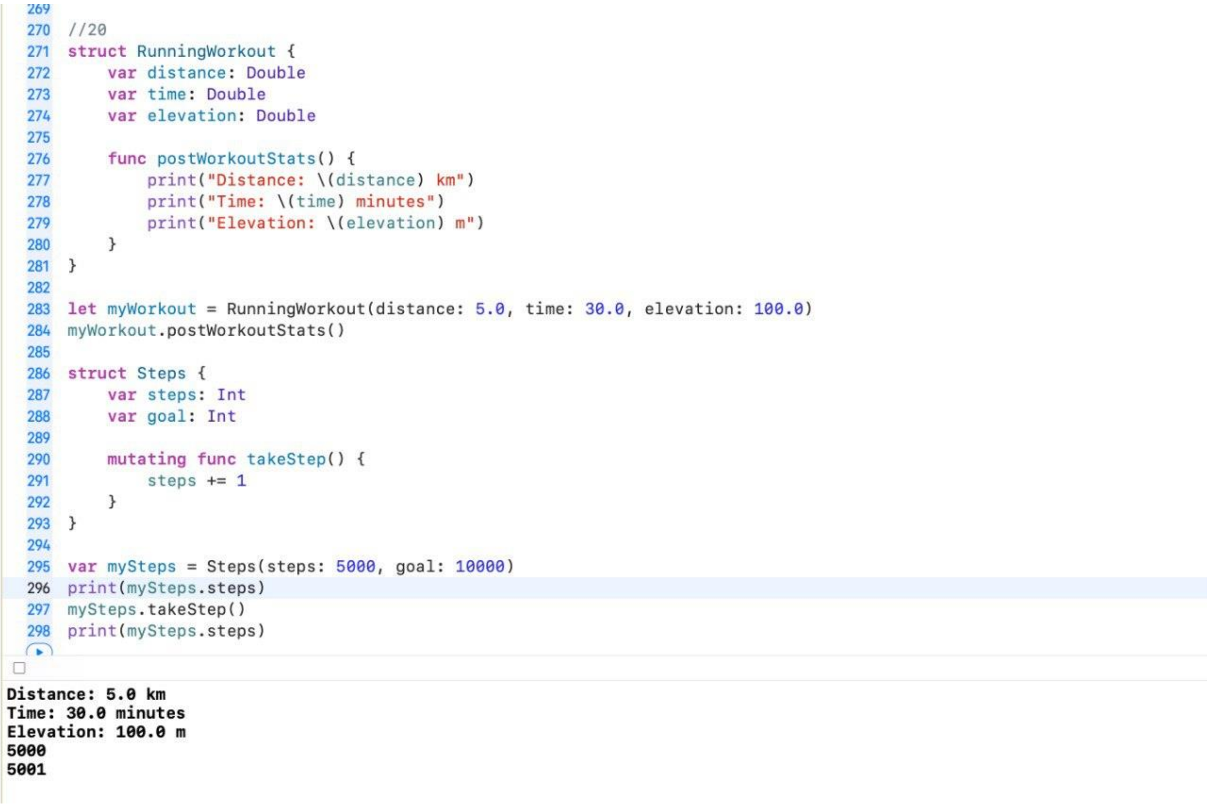
# Workout Functions

20. A RunningWorkout struct has been created for you below. Add a method on RunningWorkout called postWorkoutStats that prints out the details of the run. Then create an instance of RunningWorkout and call postWorkoutStats().

```
struct RunningWorkout {
var    distance:    Double
var time: Double       var
elevation: Double

}
```

A Steps struct has been created for you below, representing the day's step-tracking data. It has the goal number of steps for the day and the number of steps taken so far. Create a method on Steps called takeStep that increments the value of steps by one. Then create an instance of Steps and call takeStep(). Print the value of the instance's steps property before and after the method call.

```
struct  Steps  {
var   steps:   Int
var goal: Int

}
```



```
//20
struct RunningWorkout {
    var distance: Double
    var time: Double
    var elevation: Double

    func postWorkoutStats() {
        print("Distance: \(distance) km")
        print("Time: \(time) minutes")
        print("Elevation: \(elevation) m")
    }
}

let myWorkout = RunningWorkout(distance: 5.0, time: 30.0, elevation: 100.0)
myWorkout.postWorkoutStats()

struct Steps {
    var steps: Int
    var goal: Int

    mutating func takeStep() {
        steps += 1
    }
}

var mySteps = Steps(steps: 5000, goal: 10000)
print(mySteps.steps)
mySteps.takeStep()
print(mySteps.steps)
```

```
Distance: 5.0 km
Time: 30.0 minutes
Elevation: 100.0 m
5000
5001
```

## Computed Properties and Property Observers

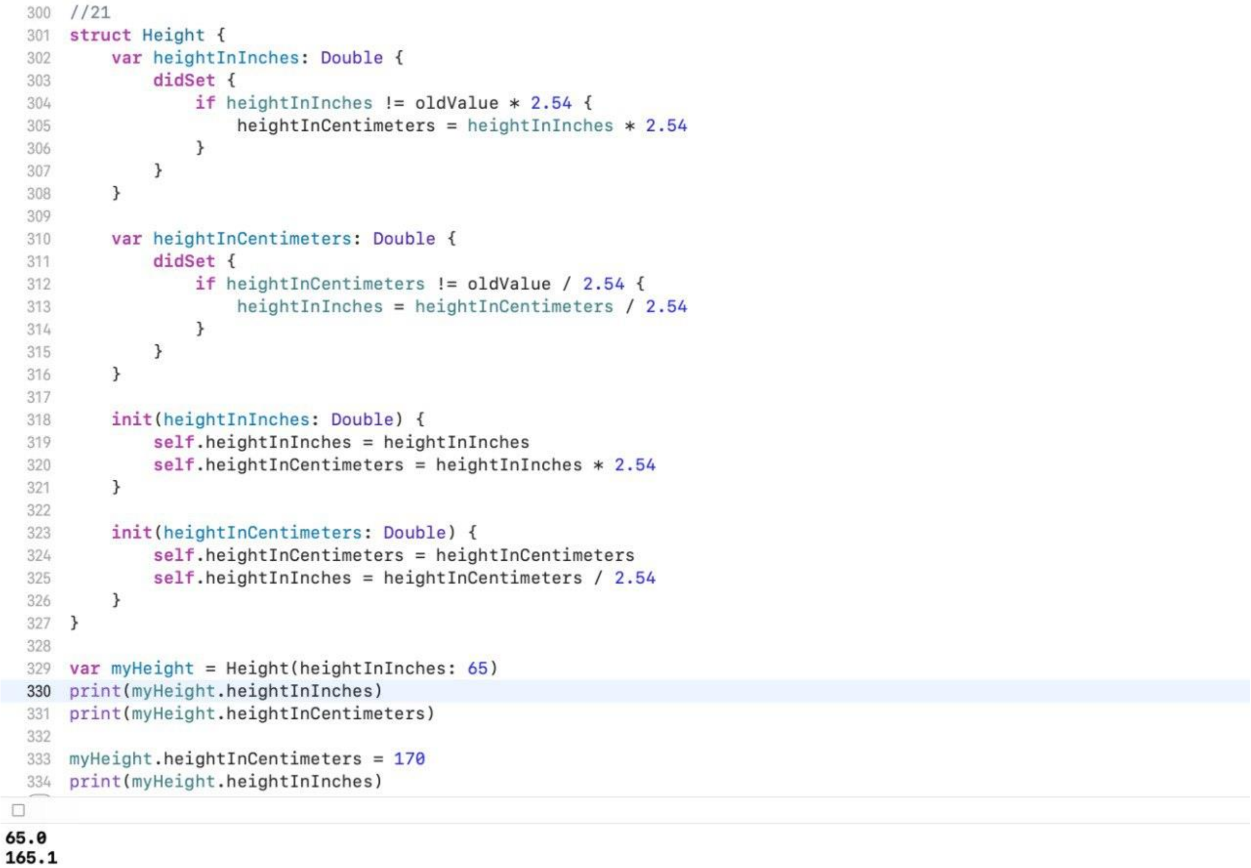
21. The Rectangle struct below has two properties, one for width and one for height. Add a computed property that computes the area of the rectangle (i.e. width * height). Create an instance of Rectangle and print the area property.

```
struct Rectangle {
    var width: Int
   var height: Int

}
```

In the Height struct below, height is represented in both inches and centimeters. However, if heightInInches is changed, heightInCentimeters should also adjust to match it. Add a didSet to each property that will check if the other property is what it should be, and if not, sets the proper value. If you set the value of the other property even though it already has the right value, you will end up with an infinite loop of each property setting the other.

Create an instance of Height and then change one of its properties. Print out the other property to ensure that it was adjusted accordingly.

```
struct Height {
   var heightInInches: Double

   var heightInCentimeters: Double

   init(heightInInches:            Double)          {
self.heightInInches = heightInInches
       self.heightInCentimeters = heightInInches*2.54
   }

   init(heightInCentimeters:          Double)         {
self.heightInCentimeters    =    heightInCentimeters
self.heightInInches = heightInCentimeters/2.54
   }
}
```



```swift
//21
struct Height {
    var heightInInches: Double {
        didSet {
            if heightInInches != oldValue * 2.54 {
                heightInCentimeters = heightInInches * 2.54
            }
        }
    }

    var heightInCentimeters: Double {
        didSet {
            if heightInCentimeters != oldValue / 2.54 {
                heightInInches = heightInCentimeters / 2.54
            }
        }
    }

    init(heightInInches: Double) {
        self.heightInInches = heightInInches
        self.heightInCentimeters = heightInInches * 2.54
    }

    init(heightInCentimeters: Double) {
        self.heightInCentimeters = heightInCentimeters
        self.heightInInches = heightInCentimeters / 2.54
    }
}

var myHeight = Height(heightInInches: 65)
print(myHeight.heightInInches)
print(myHeight.heightInCentimeters)

myHeight.heightInCentimeters = 170
print(myHeight.heightInInches)
```

```
65.0
165.1
```

## Mile Times and Congratulations

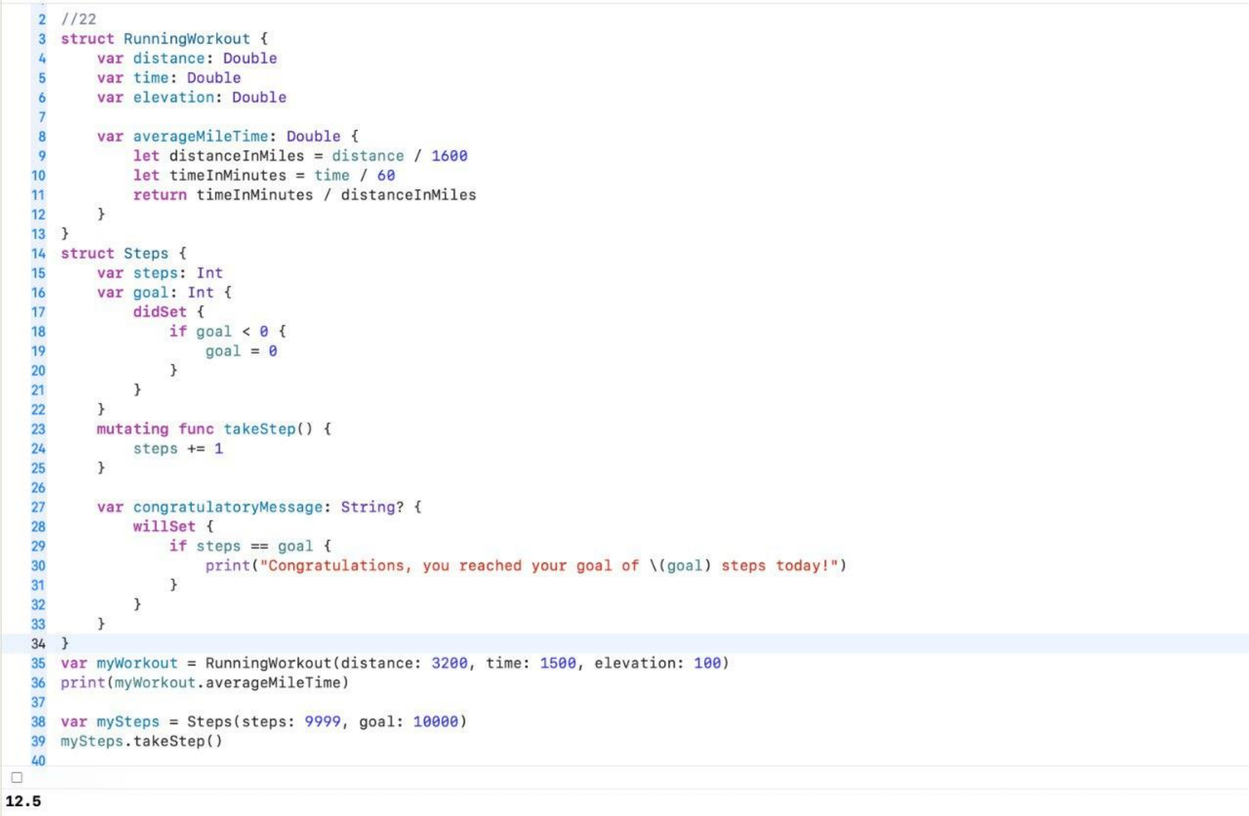
22. The RunningWorkout struct below holds information about your users' running workouts. However, you decide to add information about average mile time. Add a computed property called averageMileTime that uses distance and time to compute
the user's average mile time. Assume that distance is in meters and 1600 meters is a
mile.
Create an instance of RunningWorkout and print the averageMileTime property. Check that it works properly.

```
struct RunningWorkout {
var    distance:    Double
var time: Double
   var elevation: Double


}
```

In other app exercises, you've provided encouraging messages to the user based on how many steps they've completed. A great place to check whether or not you should display something to the user is in a property observer.

In the Steps struct below, add a willSet to the steps property that will check if the new value is equal to goal, and if it is, prints a congratulatory message. Create an instance of Steps where steps is 9999 and goal is 10000, then call takeStep() and see if your message is printed to the console.

```
struct   Steps   {
var steps: Int
   var goal: Int

   mutating func takeStep() {
       steps += 1
   }
}
```



```swift
//22
struct RunningWorkout {
    var distance: Double
    var time: Double
    var elevation: Double

    var averageMileTime: Double {
        let distanceInMiles = distance / 1600
        let timeInMinutes = time / 60
        return timeInMinutes / distanceInMiles
    }
}
struct Steps {
    var steps: Int
    var goal: Int {
        didSet {
            if goal < 0 {
                goal = 0
            }
        }
    }
    mutating func takeStep() {
        steps += 1
    }

    var congratulatoryMessage: String? {
        willSet {
            if steps == goal {
                print("Congratulations, you reached your goal of \(goal) steps today!")
            }
        }
    }
}
var myWorkout = RunningWorkout(distance: 3200, time: 1500, elevation: 100)
print(myWorkout.averageMileTime)

var mySteps = Steps(steps: 9999, goal: 10000)
mySteps.takeStep()

```

```
12.5
```

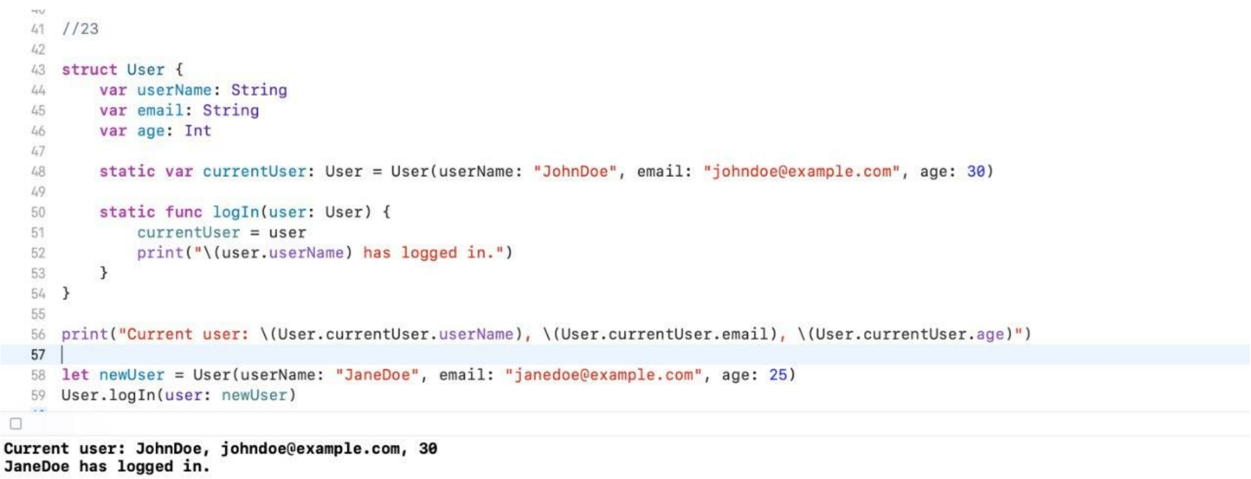
## Type Properties and Methods

23. Imagine you have an app that requires the user to log in. You may have a User struct similar to that shown below. However, in addition to keeping track of specific user information, you might want to have a way of knowing who the current logged in user is. Create a currentUser type property on the User struct below and assign it to a user object representing you. Now you can access the current user through the User struct. Print out the properties of currentUser.

```
struct User {

   var    userName:    String
var email: String
   var age: Int
}
```

There are other properties and actions associated with a User struct that might be good candidates for a type property or method. One might be a method for logging in. Go back and create a type method called logIn(user:) where user is of type User. In the body of the method, assign the passed in user to the currentUser property, and print out a statement using the user's userName saying that the user has logged in.

Below, call the logIn(user:) method and pass in a different User instance than what you assigned to currentUser above. Observe the printout in the console.



```
//23

struct User {
    var userName: String
    var email: String
    var age: Int

    static var currentUser: User = User(userName: "JohnDoe", email: "johndoe@example.com", age: 30)

    static func logIn(user: User) {
        currentUser = user
        print("\(user.userName) has logged in.")
    }
}

print("Current user: \(User.currentUser.userName), \(User.currentUser.email), \(User.currentUser.age)")

let newUser = User(userName: "JaneDoe", email: "janedoe@example.com", age: 25)
User.logIn(user: newUser)
```

```
Current user: JohnDoe, johndoe@example.com, 30
JaneDoe has logged in.
```

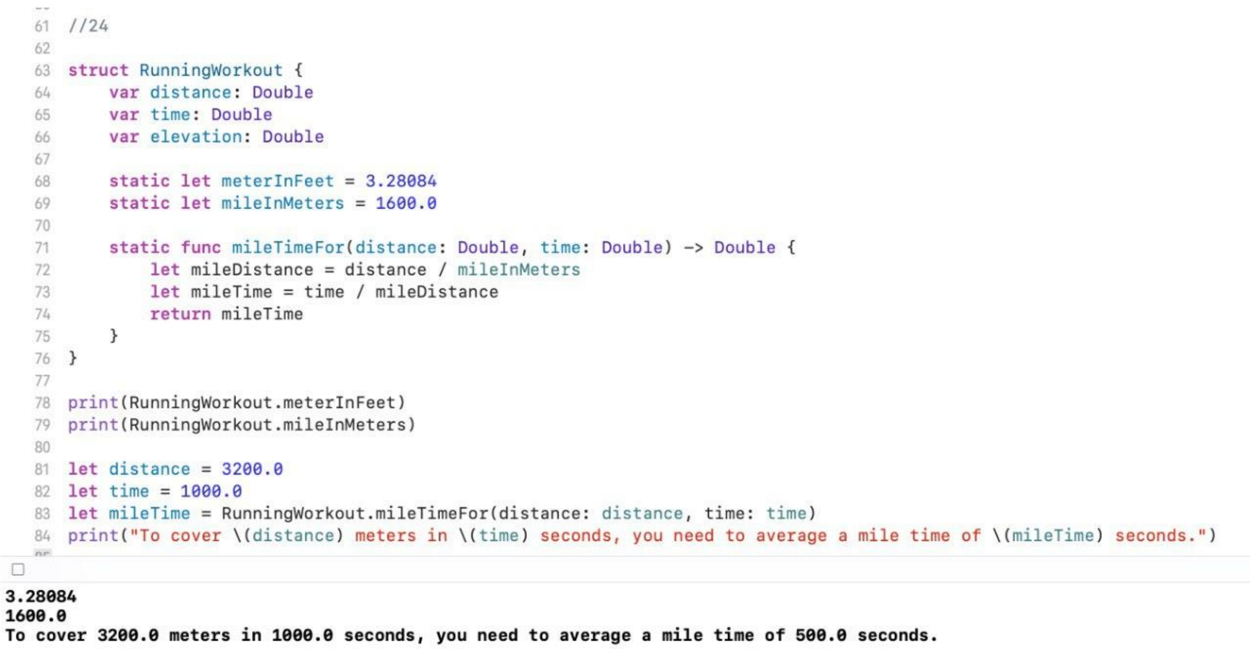
## Type Properties and Methods

24. In another exercise, you added a computed property representing the average mile time from a run. However, you may want to have a calculator of sorts that users can use before their run to find out what mile time they need to average in order to run a given distance in a given time. In this case it might be helpful to have a type method on RunningWorkout that can be accessed without having an instance of RunningWorkout.

    Add to RunningWorkout a type method mileTimeFor(distance:time:) where distance and time are both of type Double. This method should have a return value of type Double. The body of the method should calculate the average mile time needed to cover the passed in distance in the passed in time. Assume that distance is in meters and that one mile is 1600 meters. Call the method from outside of the struct and print the result to ensure that it works properly.

```
struct RunningWorkout {
var    distance:    Double
var time: Double
   var elevation: Double
}
```

    It may be helpful to have a few type properties on RunningWorkout representing unit conversions (i.e. meters to mile, feet to meters, etc.). Go back and add a type property for meterInFeet and assign it 3.28084. Then add a type property for mileInMeters and assign it 1600.0. Print both of these values below.



```
//24

struct RunningWorkout {
    var distance: Double
    var time: Double
    var elevation: Double

    static let meterInFeet = 3.28084
    static let mileInMeters = 1600.0

    static func mileTimeFor(distance: Double, time: Double) -> Double {
        let mileDistance = distance / mileInMeters
        let mileTime = time / mileDistance
        return mileTime
    }
}

print(RunningWorkout.meterInFeet)
print(RunningWorkout.mileInMeters)

let distance = 3200.0
let time = 1000.0
let mileTime = RunningWorkout.mileTimeFor(distance: distance, time: time)
print("To cover \(distance) meters in \(time) seconds, you need to average a mile time of \(mileTime) seconds.")
```

```
3.28084
1600.0
To cover 3200.0 meters in 1000.0 seconds, you need to average a mile time of 500.0 seconds.
```