Department of Computer Science and Engineering

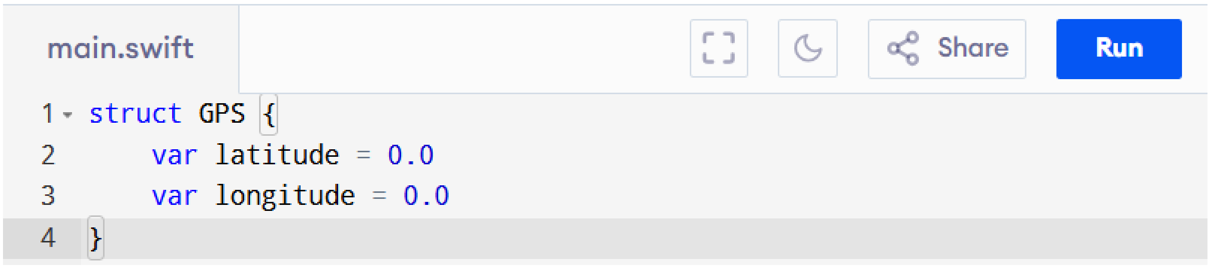
Sub: iOS Lab

Assignment No 2 : Structures

Name : Suraj Shantinath Upadhye. **PRN :** 245200001 **Class :** SY CSE

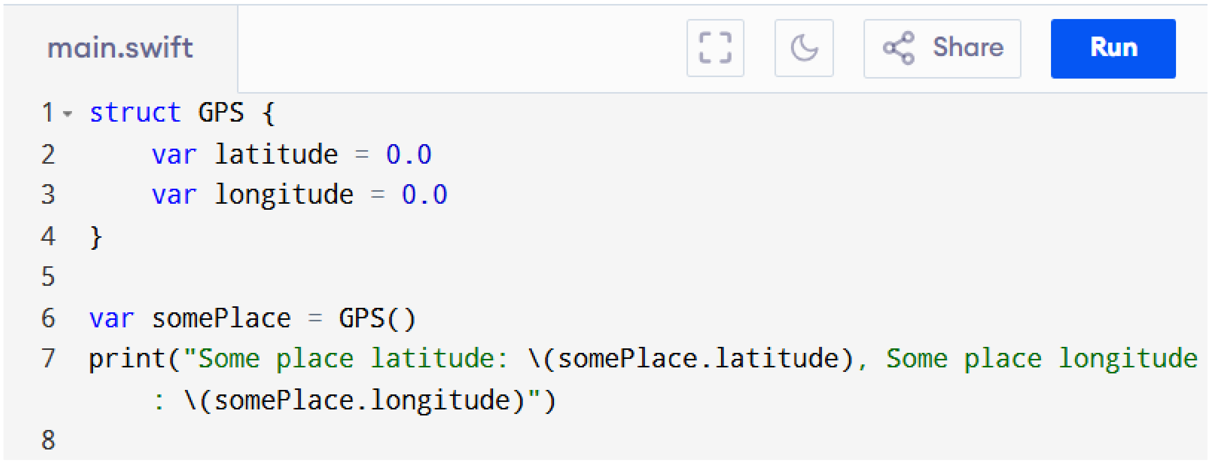
Exercise - Structs, Instances, and Default Values

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.

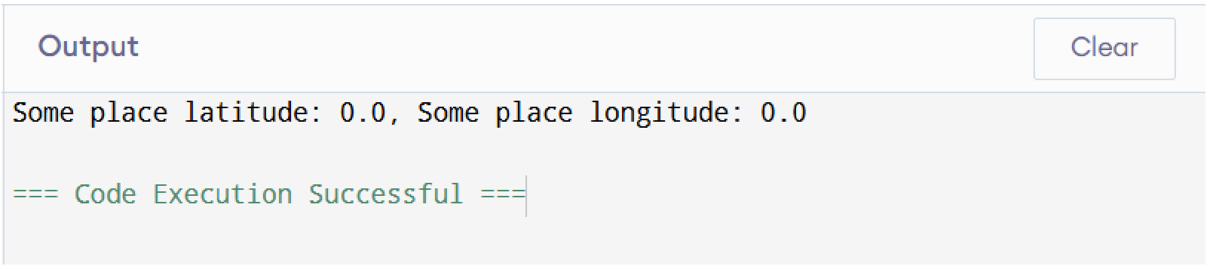


```swift
struct GPS {
    var latitude = 0.0
    var longitude = 0.0
}
```

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.

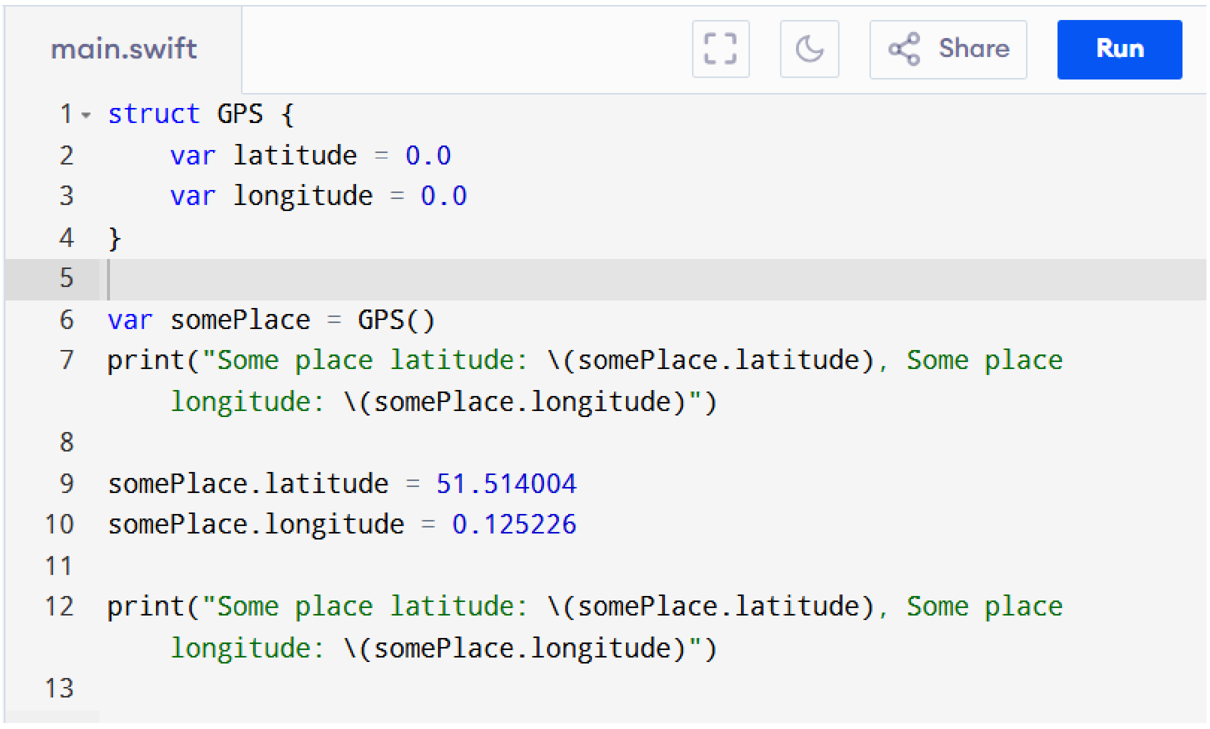


```swift
struct GPS {
    var latitude = 0.0
    var longitude = 0.0
}

var somePlace = GPS()
print("Some place latitude: \(somePlace.latitude), Some place longitude
    : \(somePlace.longitude)")

```

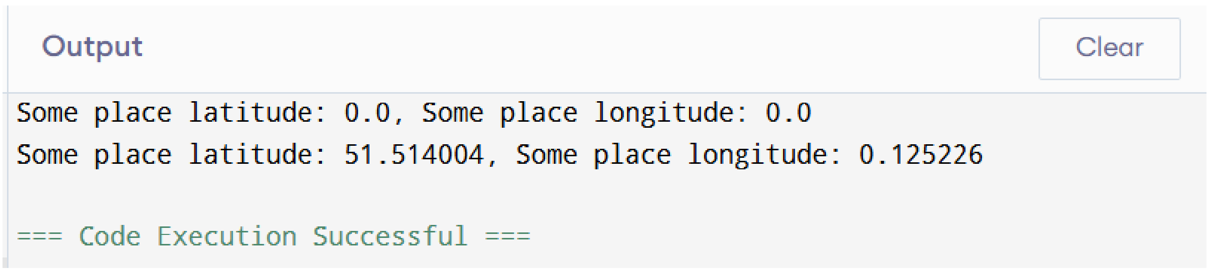


```
Some place latitude: 0.0, Some place longitude: 0.0

=== Code Execution Successful ===
```

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

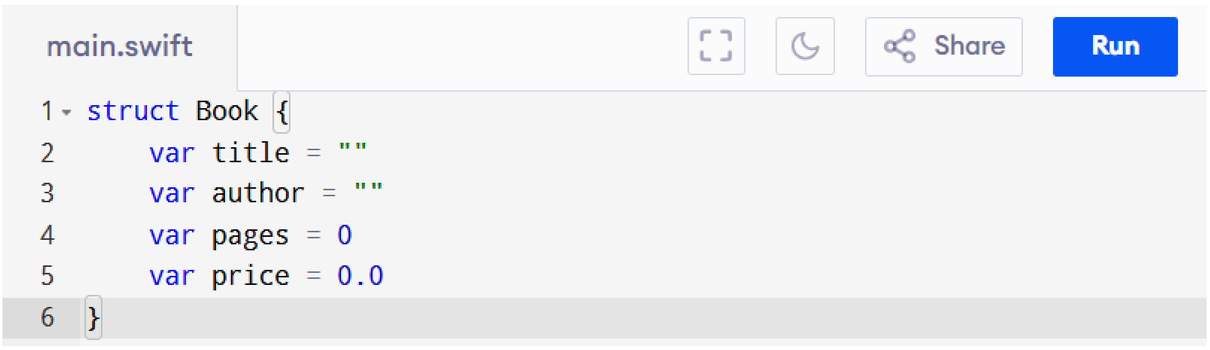


```swift
struct GPS {
    var latitude = 0.0
    var longitude = 0.0
}

var somePlace = GPS()
print("Some place latitude: \(somePlace.latitude), Some place
    longitude: \(somePlace.longitude)")

somePlace.latitude = 51.514004
somePlace.longitude = 0.125226

print("Some place latitude: \(somePlace.latitude), Some place
    longitude: \(somePlace.longitude)")

```



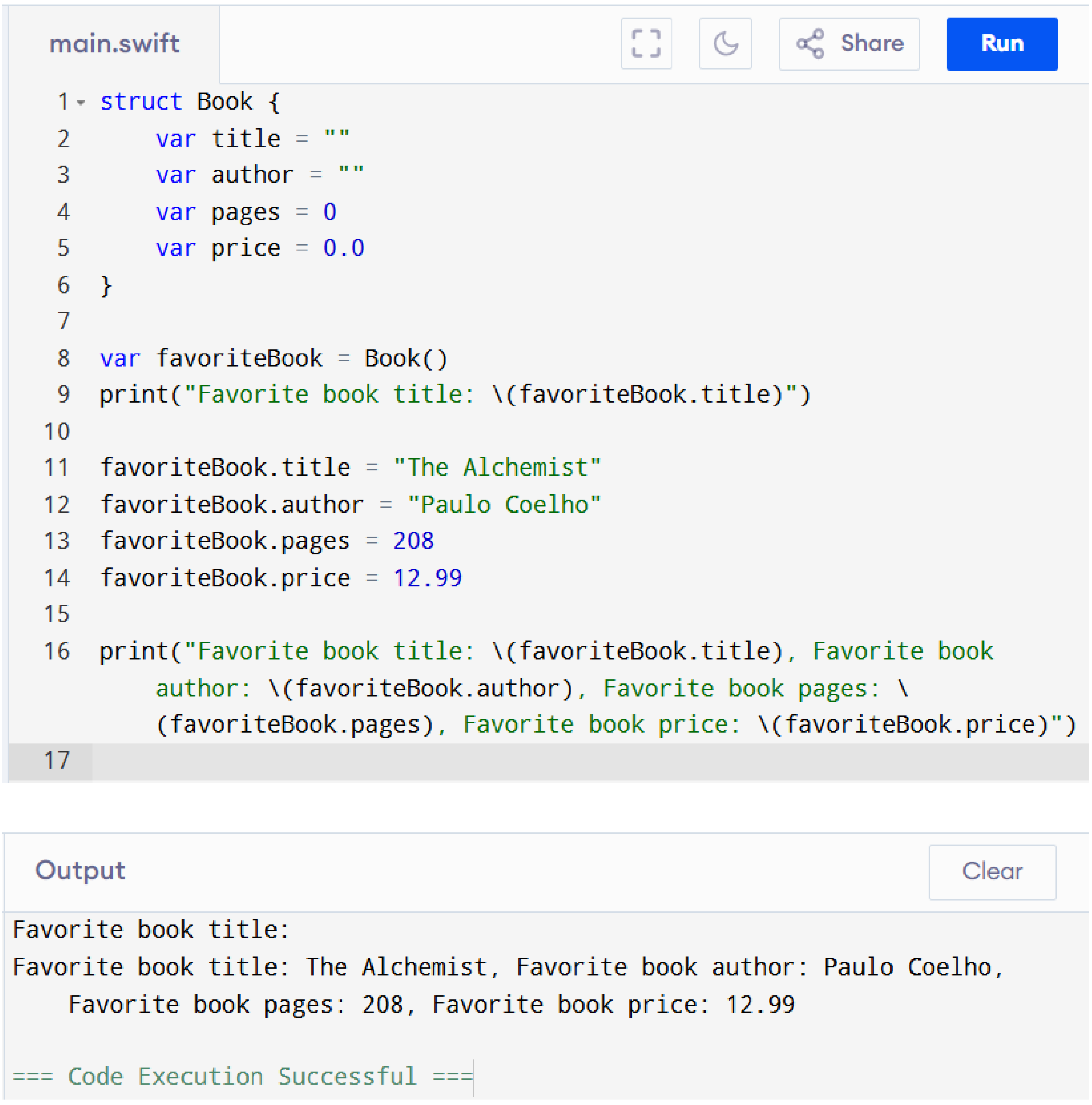
```
Some place latitude: 0.0, Some place longitude: 0.0
Some place latitude: 51.514004, Some place longitude: 0.125226

=== Code Execution Successful ===
```

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.



```swift
struct Book {
    var title = ""
    var author = ""
    var pages = 0
    var price = 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.

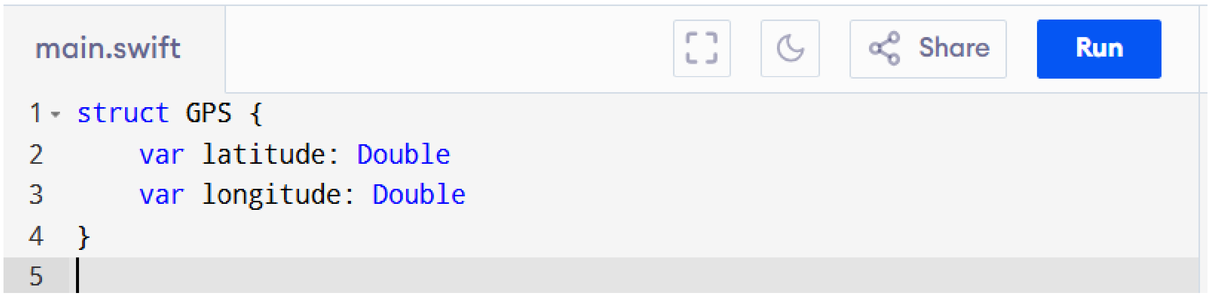


```swift
struct Book {
    var title = ""
    var author = ""
    var pages = 0
    var price = 0.0
}

var favoriteBook = Book()
print("Favorite book title: \(favoriteBook.title)")

favoriteBook.title = "The Alchemist"
favoriteBook.author = "Paulo Coelho"
favoriteBook.pages = 208
favoriteBook.price = 12.99

print("Favorite book title: \(favoriteBook.title), Favorite book
    author: \(favoriteBook.author), Favorite book pages: \
    (favoriteBook.pages), Favorite book price: \(favoriteBook.price)")

```

```
Favorite book title:
Favorite book title: The Alchemist, Favorite book author: Paulo Coelho,
    Favorite book pages: 208, Favorite book price: 12.99

=== Code Execution Successful ===
```

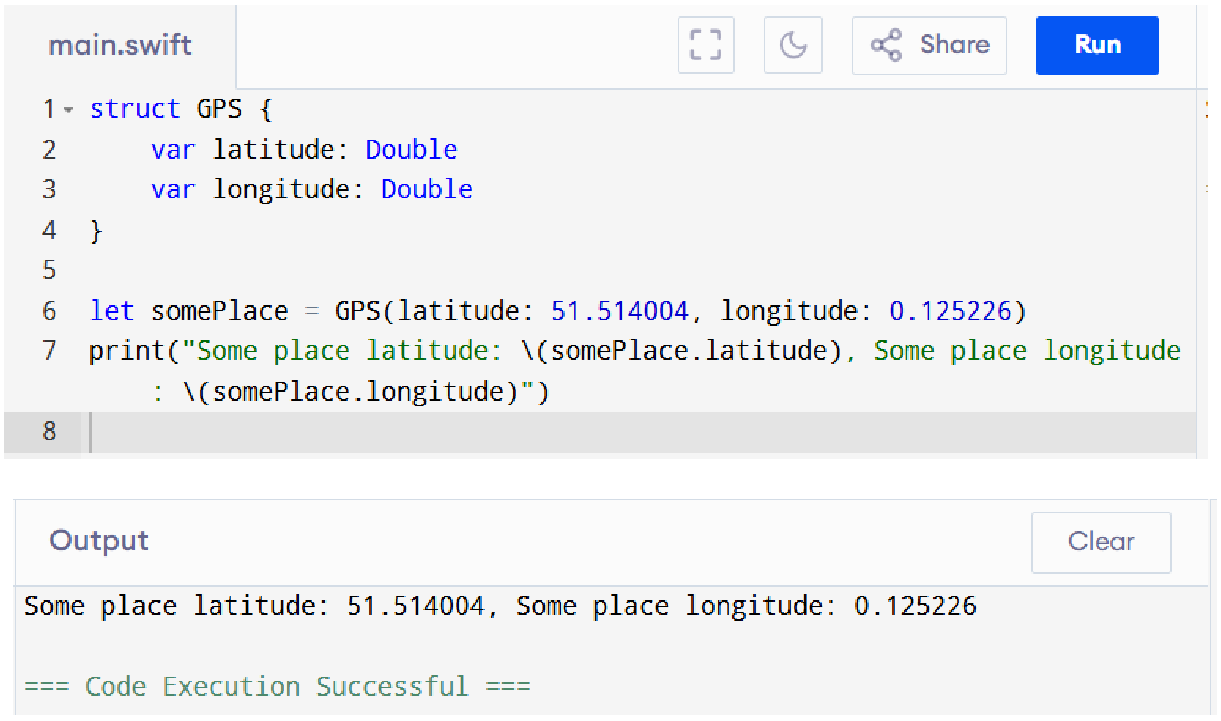
Exercise - Memberwise and Custom Initializers

6. 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
struct GPS {
    var latitude: Double
    var longitude: Double
}
```

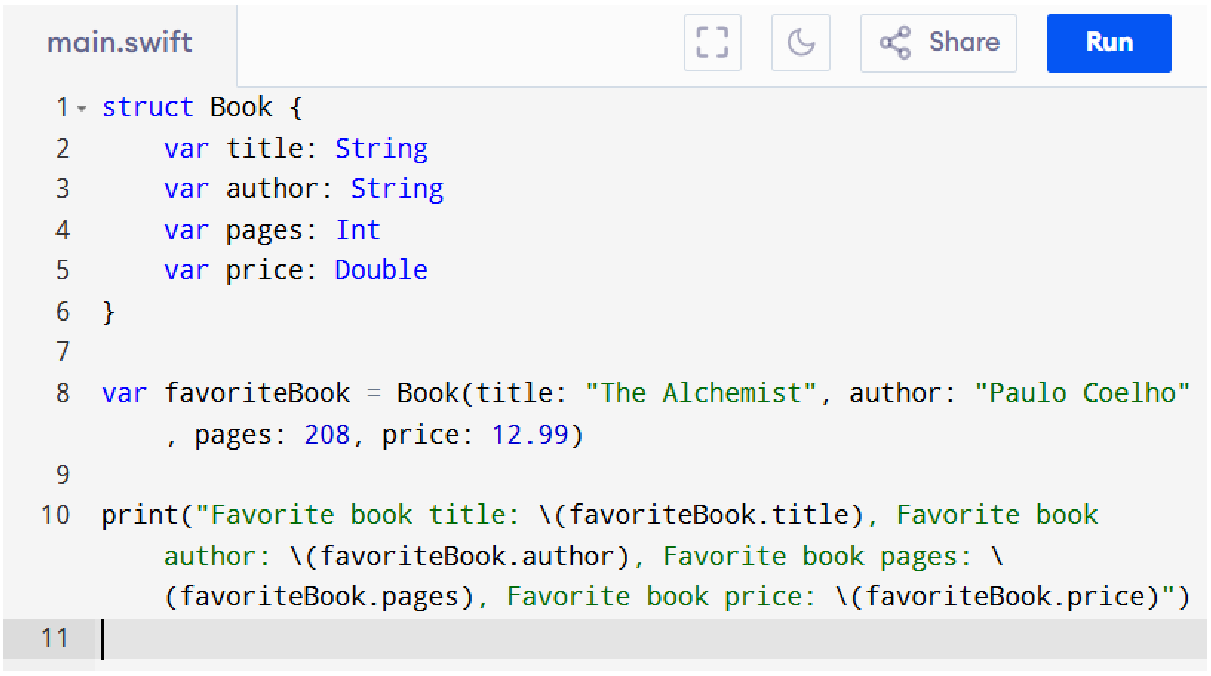
7. Now create a constant instance of GPS called somePlace, and use the memberwise initializer to set latitude to 51.514004, and longitude to 0.125226. Print the values of somePlace's properties.



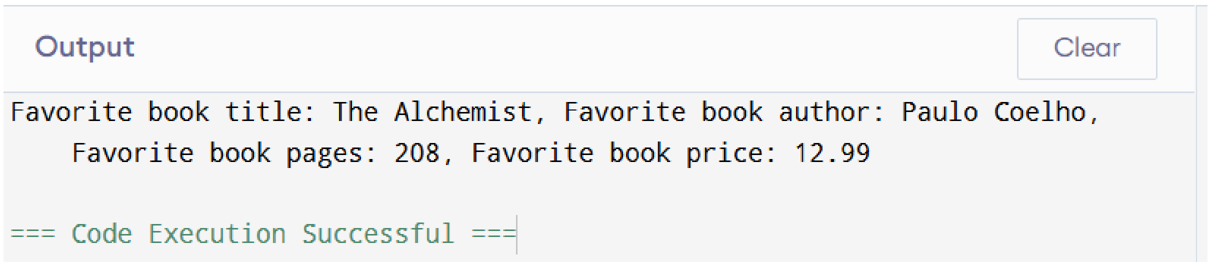
```swift
struct GPS {
    var latitude: Double
    var longitude: Double
}

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

```
Some place latitude: 51.514004, Some place longitude: 0.125226

=== Code Execution Successful ===
```

8. 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.



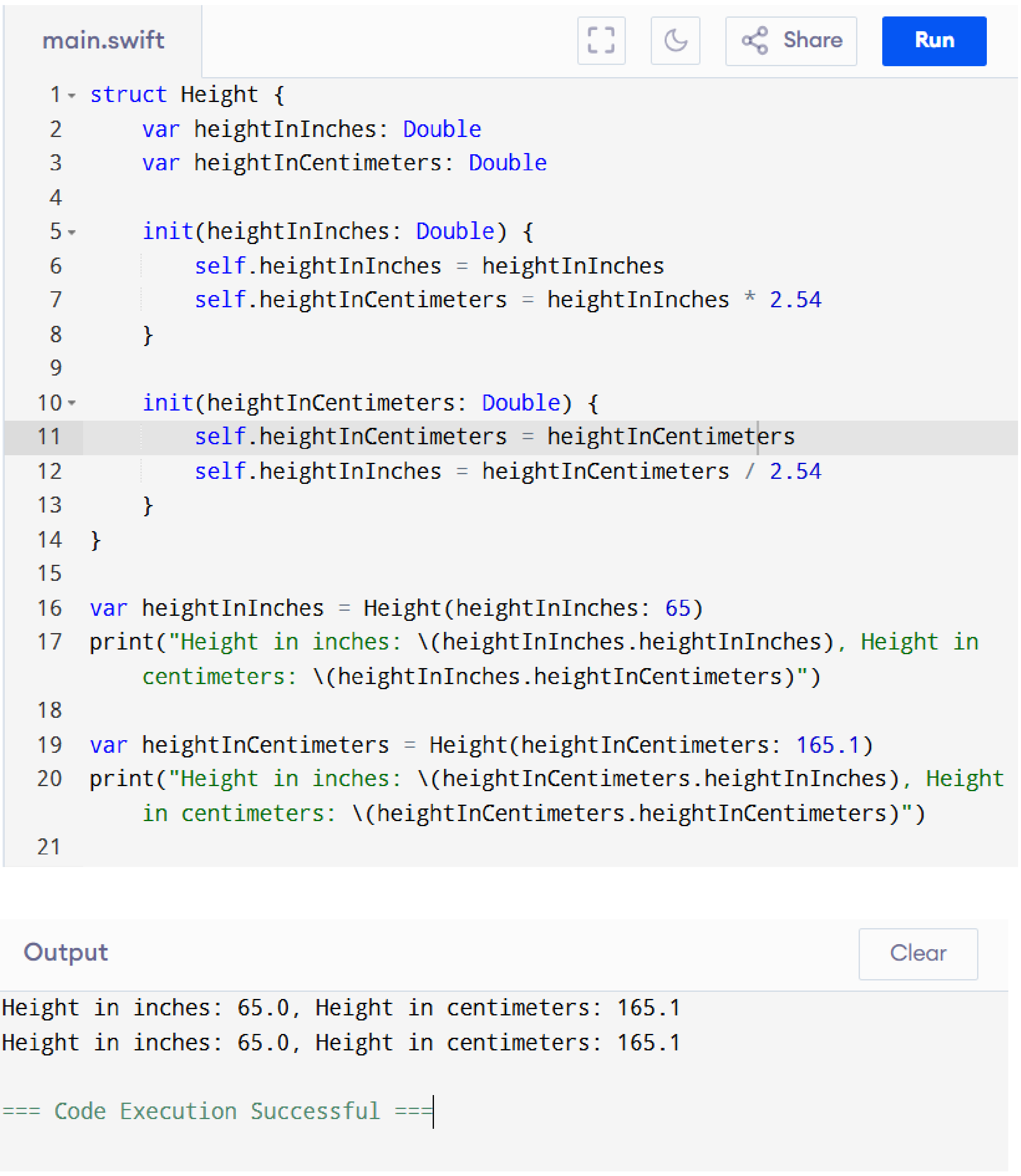
```swift
struct Book {
    var title: String
    var author: String
    var pages: Int
    var price: Double
}

var favoriteBook = Book(title: "The Alchemist", author: "Paulo Coelho"
    , pages: 208, price: 12.99)

print("Favorite book title: \(favoriteBook.title), Favorite book
    author: \(favoriteBook.author), Favorite book pages: \
    (favoriteBook.pages), Favorite book price: \(favoriteBook.price)")
```



```
Favorite book title: The Alchemist, Favorite book author: Paulo Coelho,
    Favorite book pages: 208, Favorite book price: 12.99

=== Code Execution Successful ===
```

9. 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 centimetres.

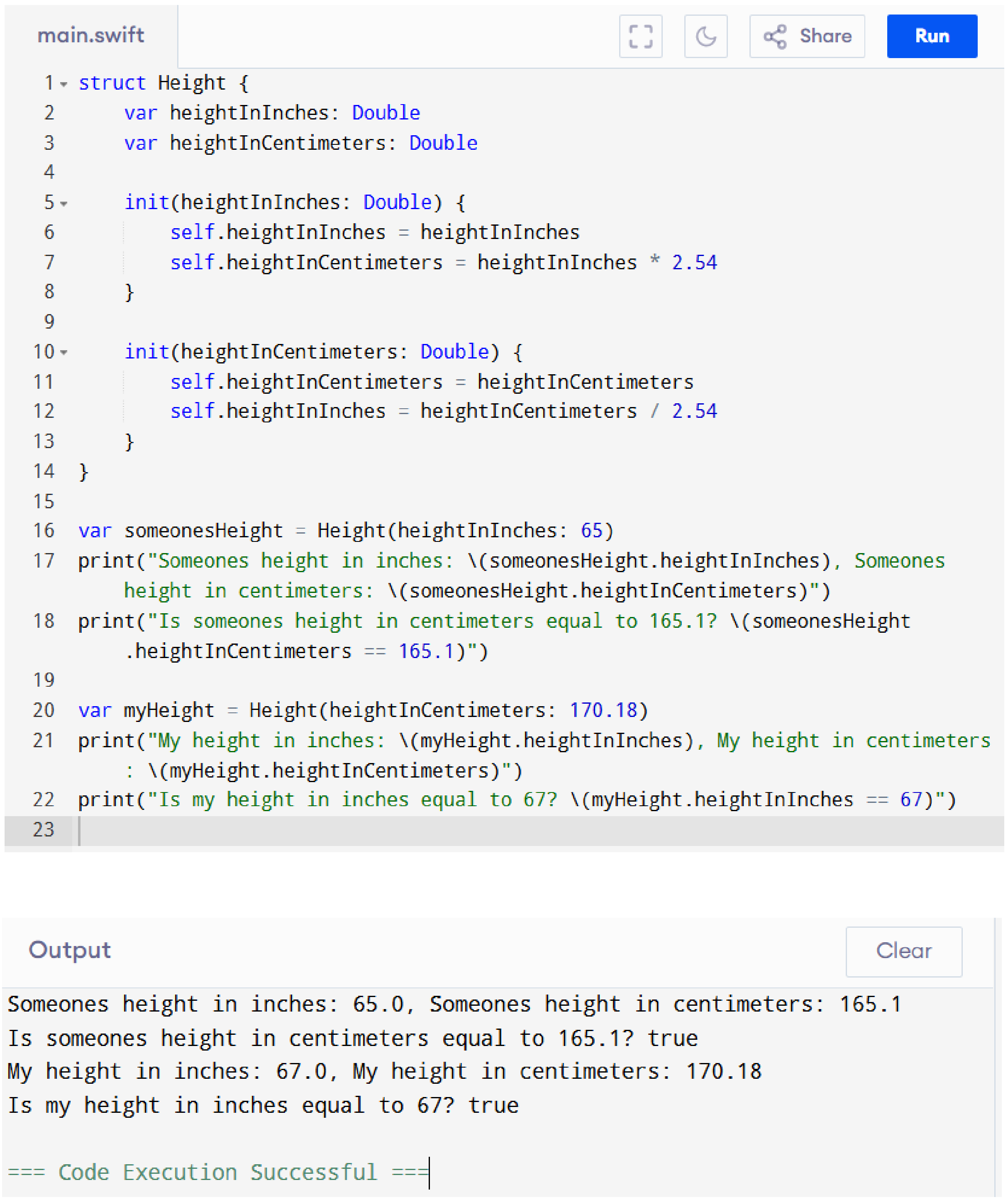
   Example: If you use the initializer for inches to pass in a height of 65, the initializer should set heightInInches to 65 and heightInCentimeters to 165.1.



```swift
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 heightInInches = Height(heightInInches: 65)
print("Height in inches: \(heightInInches.heightInInches), Height in
    centimeters: \(heightInInches.heightInCentimeters)")

var heightInCentimeters = Height(heightInCentimeters: 165.1)
print("Height in inches: \(heightInCentimeters.heightInInches), Height
    in centimeters: \(heightInCentimeters.heightInCentimeters)")
```

```
Height in inches: 65.0, Height in centimeters: 165.1
Height in inches: 65.0, Height in centimeters: 165.1

=== Code Execution Successful ===
```

10. 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.

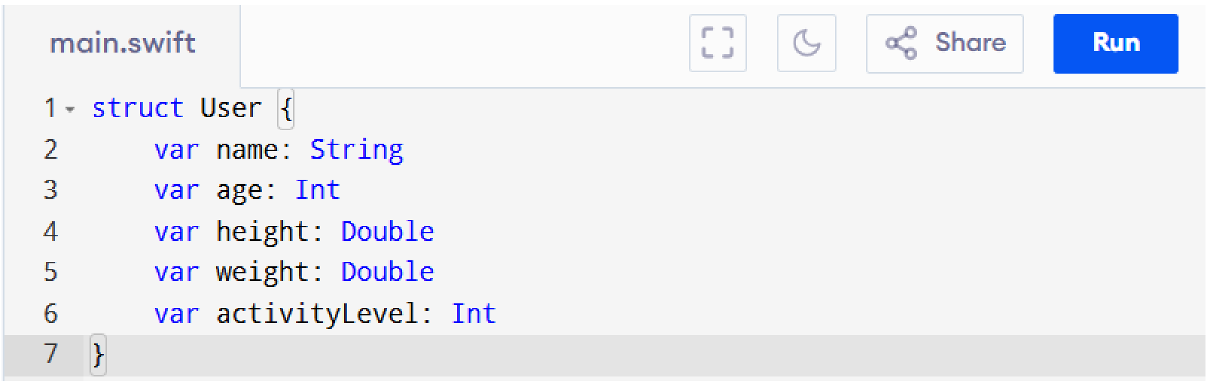


```swift
struct Height {
    var heightInInches: Double
    var heightInCentimeters: Double

    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("Someones height in inches: \(someonesHeight.heightInInches), Someones
    height in centimeters: \(someonesHeight.heightInCentimeters)")
print("Is someones height in centimeters equal to 165.1? \(someonesHeight
    .heightInCentimeters == 165.1)")

var myHeight = Height(heightInCentimeters: 170.18)
print("My height in inches: \(myHeight.heightInInches), My height in centimeters
    : \(myHeight.heightInCentimeters)")
print("Is my height in inches equal to 67? \(myHeight.heightInInches == 67)")

```

```
Someones height in inches: 65.0, Someones height in centimeters: 165.1
Is someones height in centimeters equal to 165.1? true
My height in inches: 67.0, My height in centimeters: 170.18
Is my height in inches equal to 67? true

=== Code Execution Successful ===
```

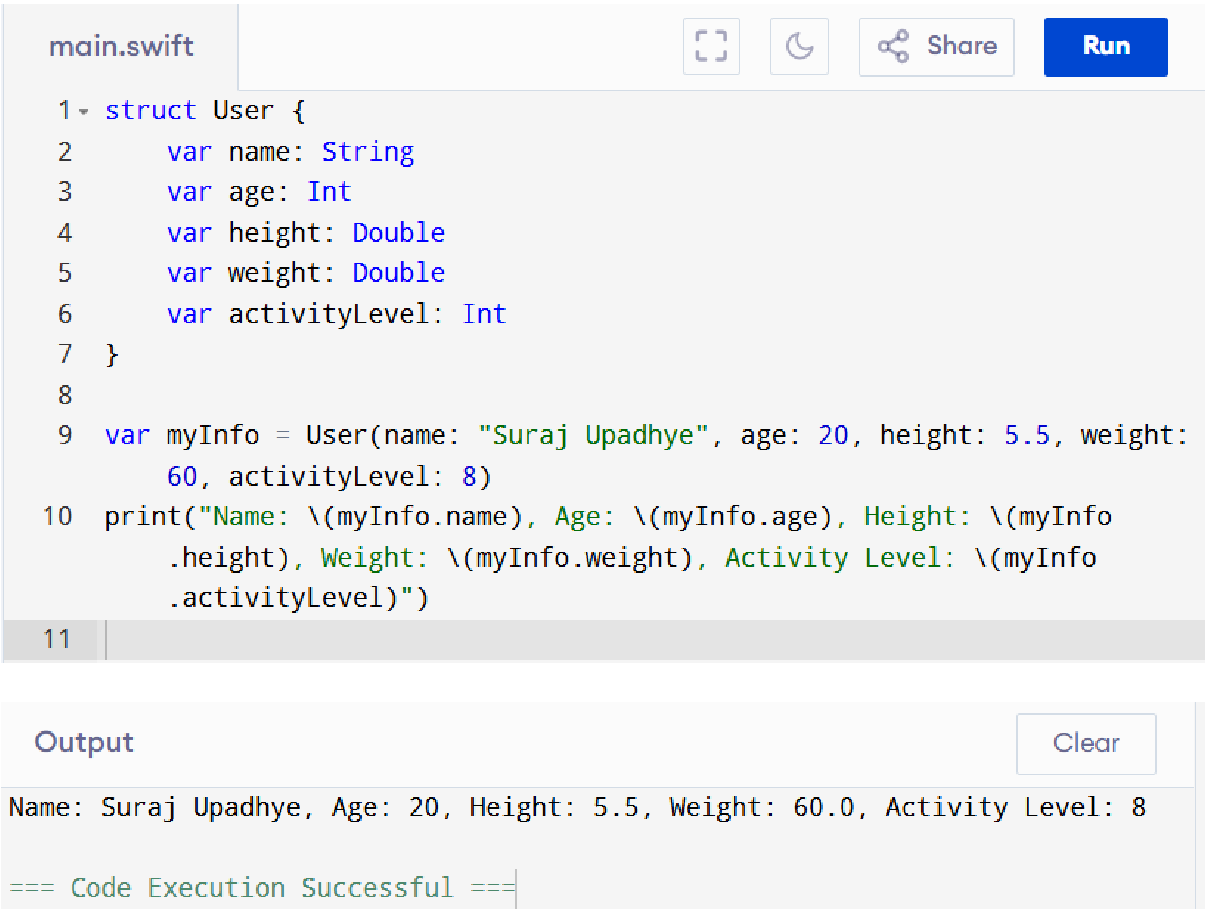
Users and Distance

11. 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
struct User {
    var name: String
    var age: Int
    var height: Double
    var weight: Double
    var activityLevel: Int
}
```

12. 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.



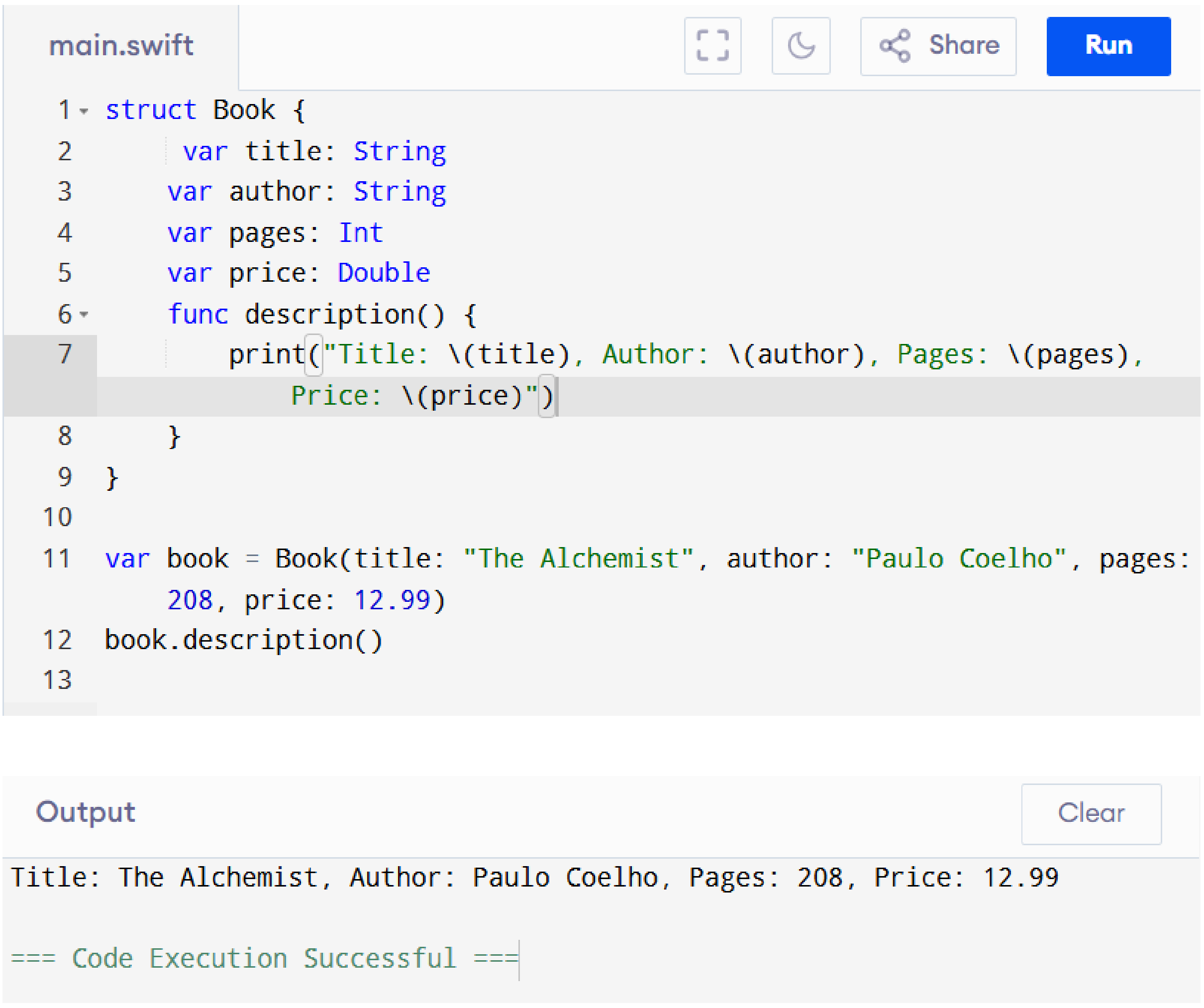
```swift
struct User {
    var name: String
    var age: Int
    var height: Double
    var weight: Double
    var activityLevel: Int
}

var myInfo = User(name: "Suraj Upadhye", age: 20, height: 5.5, weight:
    60, activityLevel: 8)
print("Name: \(myInfo.name), Age: \(myInfo.age), Height: \(myInfo
    .height), Weight: \(myInfo.weight), Activity Level: \(myInfo
    .activityLevel)")
```

```
Name: Suraj Upadhye, Age: 20, Height: 5.5, Weight: 60.0, Activity Level: 8

=== Code Execution Successful ===
```

Methods

13. 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
}
```

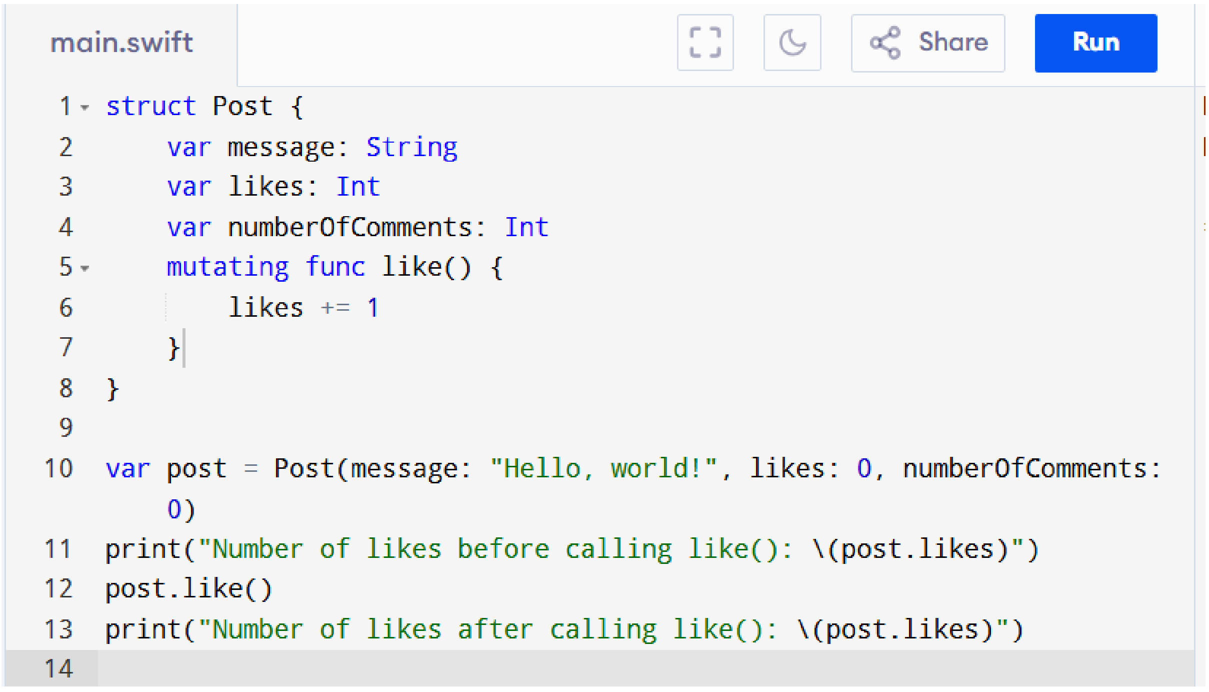


```
struct Book {
     var title: String
    var author: String
    var pages: Int
    var price: Double
    func description() {
        print("Title: \(title), Author: \(author), Pages: \(pages),
            Price: \(price)")
    }
}

var book = Book(title: "The Alchemist", author: "Paulo Coelho", pages:
    208, price: 12.99)
book.description()

```

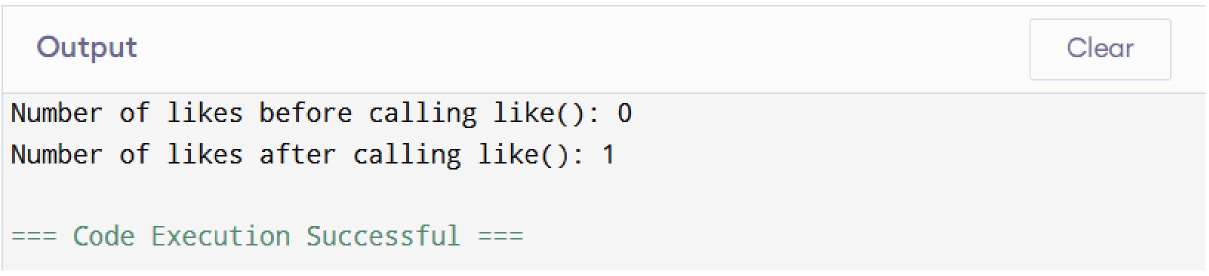
```
Title: The Alchemist, Author: Paulo Coelho, Pages: 208, Price: 12.99

=== Code Execution Successful ===
```

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
}
```



```
struct Post {
    var message: String
    var likes: Int
    var numberOfComments: Int
    mutating func like() {
        likes += 1
    }
}

var post = Post(message: "Hello, world!", likes: 0, numberOfComments:
    0)
print("Number of likes before calling like(): \(post.likes)")
post.like()
print("Number of likes after calling like(): \(post.likes)")
```

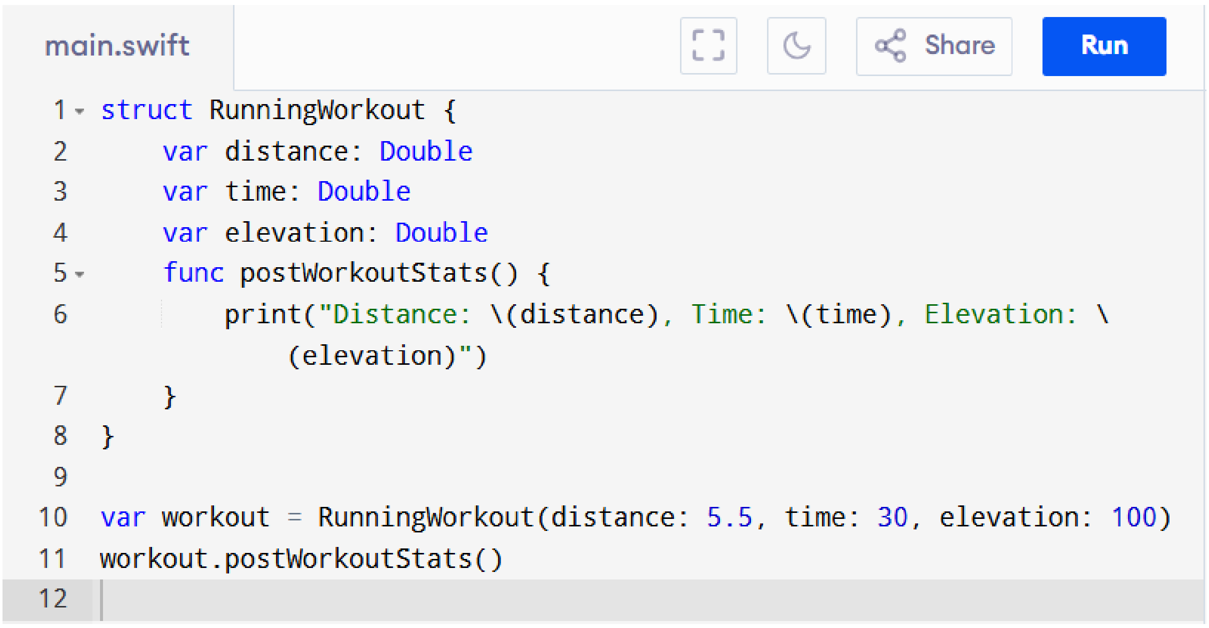


```
Number of likes before calling like(): 0
Number of likes after calling like(): 1

=== Code Execution Successful ===
```

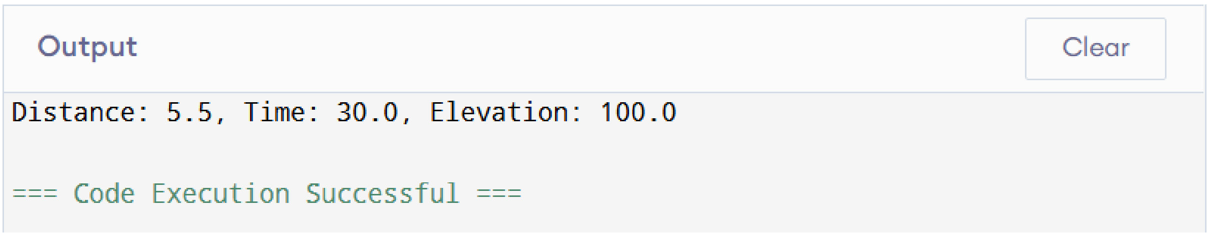
Workout Functions

14. 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

}
```



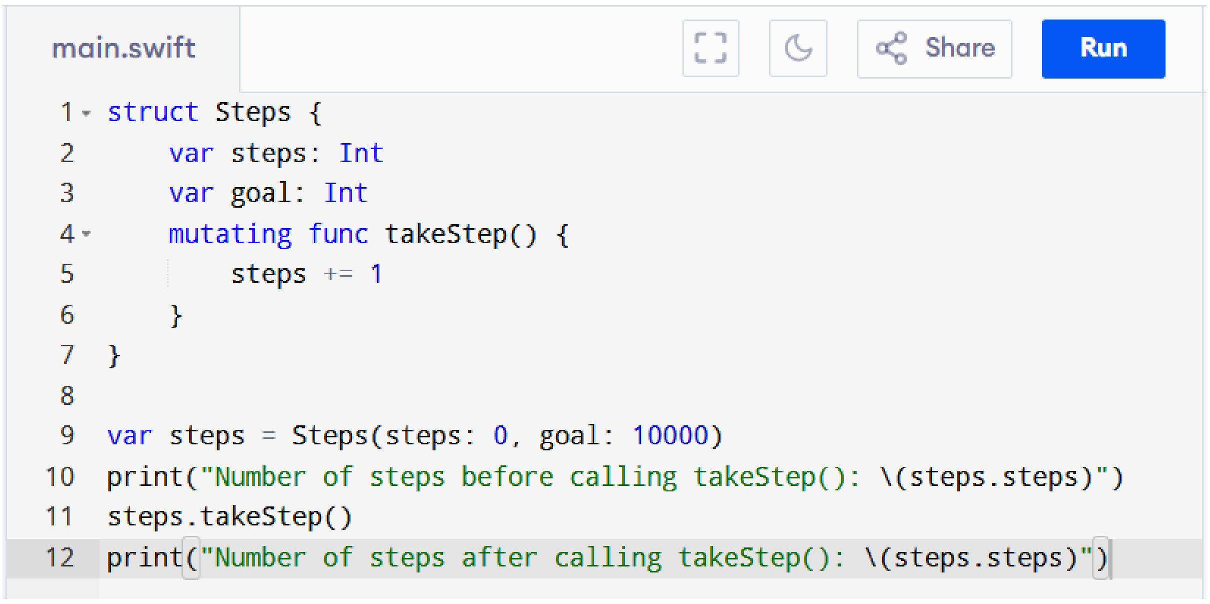
```swift
struct RunningWorkout {
    var distance: Double
    var time: Double
    var elevation: Double
    func postWorkoutStats() {
        print("Distance: \(distance), Time: \(time), Elevation: \
            (elevation)")
    }
}

var workout = RunningWorkout(distance: 5.5, time: 30, elevation: 100)
workout.postWorkoutStats()
```



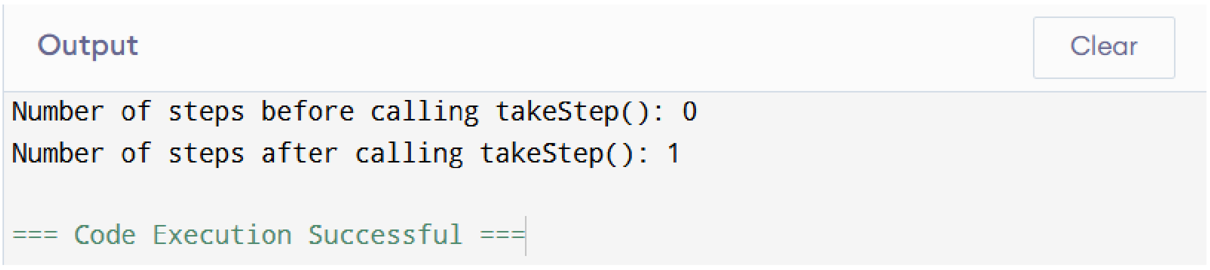
```
Distance: 5.5, Time: 30.0, Elevation: 100.0

=== Code Execution Successful ===
```

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
}
```



```
struct Steps {
    var steps: Int
    var goal: Int
    mutating func takeStep() {
        steps += 1
    }
}

var steps = Steps(steps: 0, goal: 10000)
print("Number of steps before calling takeStep(): \(steps.steps)")
steps.takeStep()
print("Number of steps after calling takeStep(): \(steps.steps)")
```

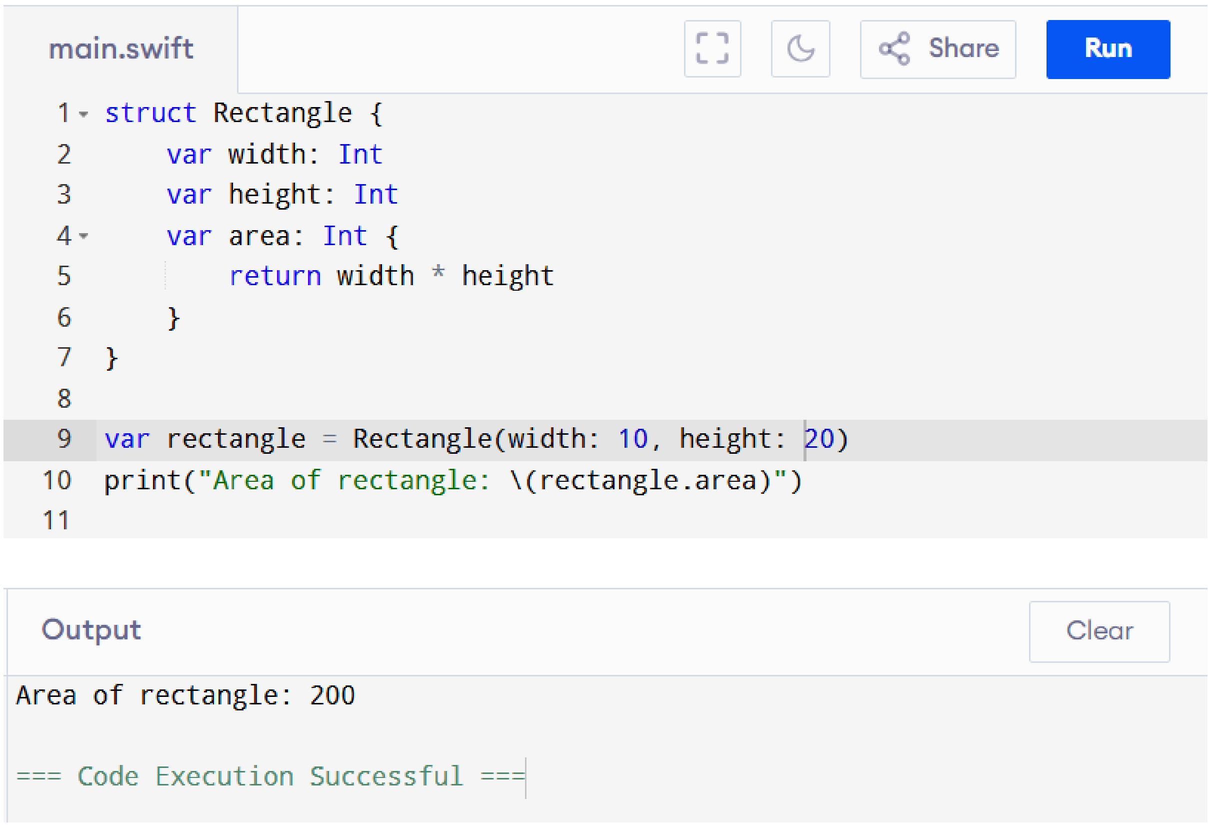


```
Number of steps before calling takeStep(): 0
Number of steps after calling takeStep(): 1

=== Code Execution Successful ===
```

Computed Properties and Property Observers

15. 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
}
```

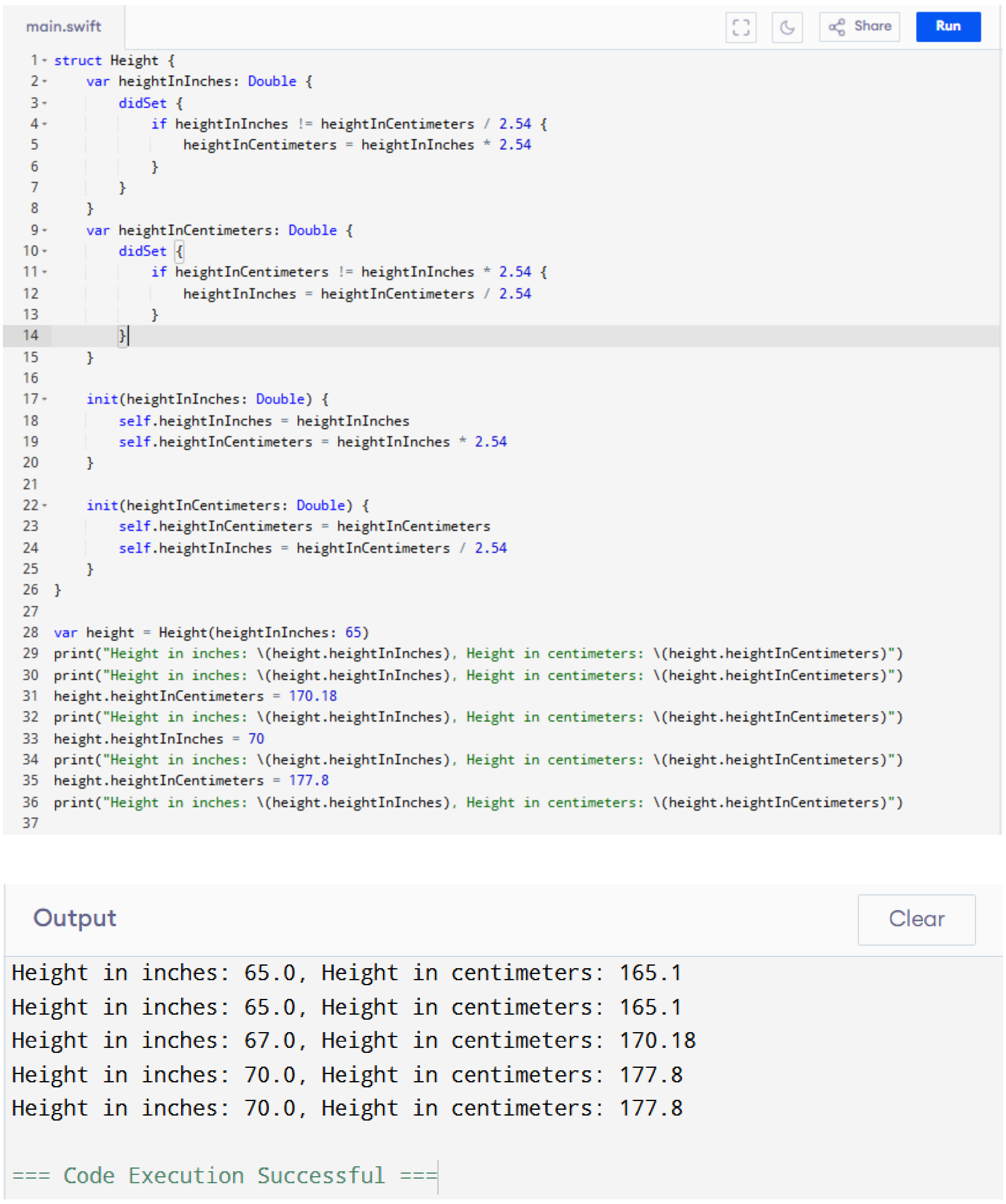


```
struct Rectangle {
    var width: Int
    var height: Int
    var area: Int {
        return width * height
    }
}

var rectangle = Rectangle(width: 10, height: 20)
print("Area of rectangle: \(rectangle.area)")

```

```
Area of rectangle: 200

=== Code Execution Successful ===
```

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
struct Height {
    var heightInInches: Double {
        didSet {
            if heightInInches != heightInCentimeters / 2.54 {
                heightInCentimeters = heightInInches * 2.54
            }
        }
    }
    var heightInCentimeters: Double {
        didSet {
            if heightInCentimeters != heightInInches * 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 height = Height(heightInInches: 65)
print("Height in inches: \(height.heightInInches), Height in centimeters: \(height.heightInCentimeters)")
print("Height in inches: \(height.heightInInches), Height in centimeters: \(height.heightInCentimeters)")
height.heightInCentimeters = 170.18
print("Height in inches: \(height.heightInInches), Height in centimeters: \(height.heightInCentimeters)")
height.heightInInches = 70
print("Height in inches: \(height.heightInInches), Height in centimeters: \(height.heightInCentimeters)")
height.heightInCentimeters = 177.8
print("Height in inches: \(height.heightInInches), Height in centimeters: \(height.heightInCentimeters)")
```

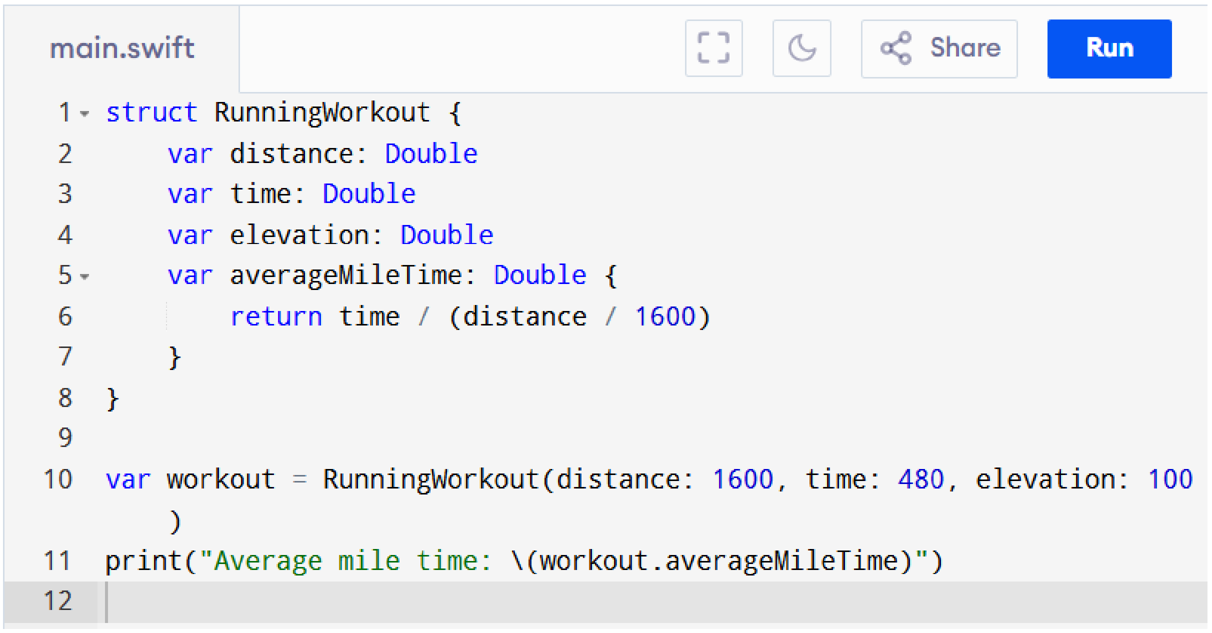
```
Height in inches: 65.0, Height in centimeters: 165.1
Height in inches: 65.0, Height in centimeters: 165.1
Height in inches: 67.0, Height in centimeters: 170.18
Height in inches: 70.0, Height in centimeters: 177.8
Height in inches: 70.0, Height in centimeters: 177.8

=== Code Execution Successful ===
```

Mile Times and Congratulations

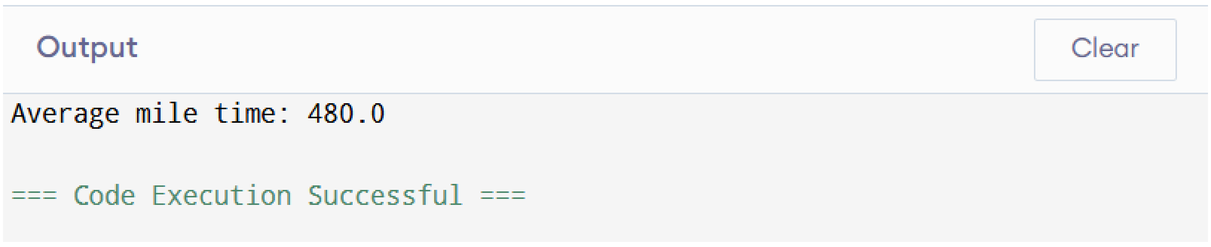
16. The RunningWorkout struct below holds information about your users' running workouts. However, you decide to add information about average mile time. Add a computed property called averageMileTime that uses distance and time to compute the user's average mile time. Assume that distance is in meters 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


}
```



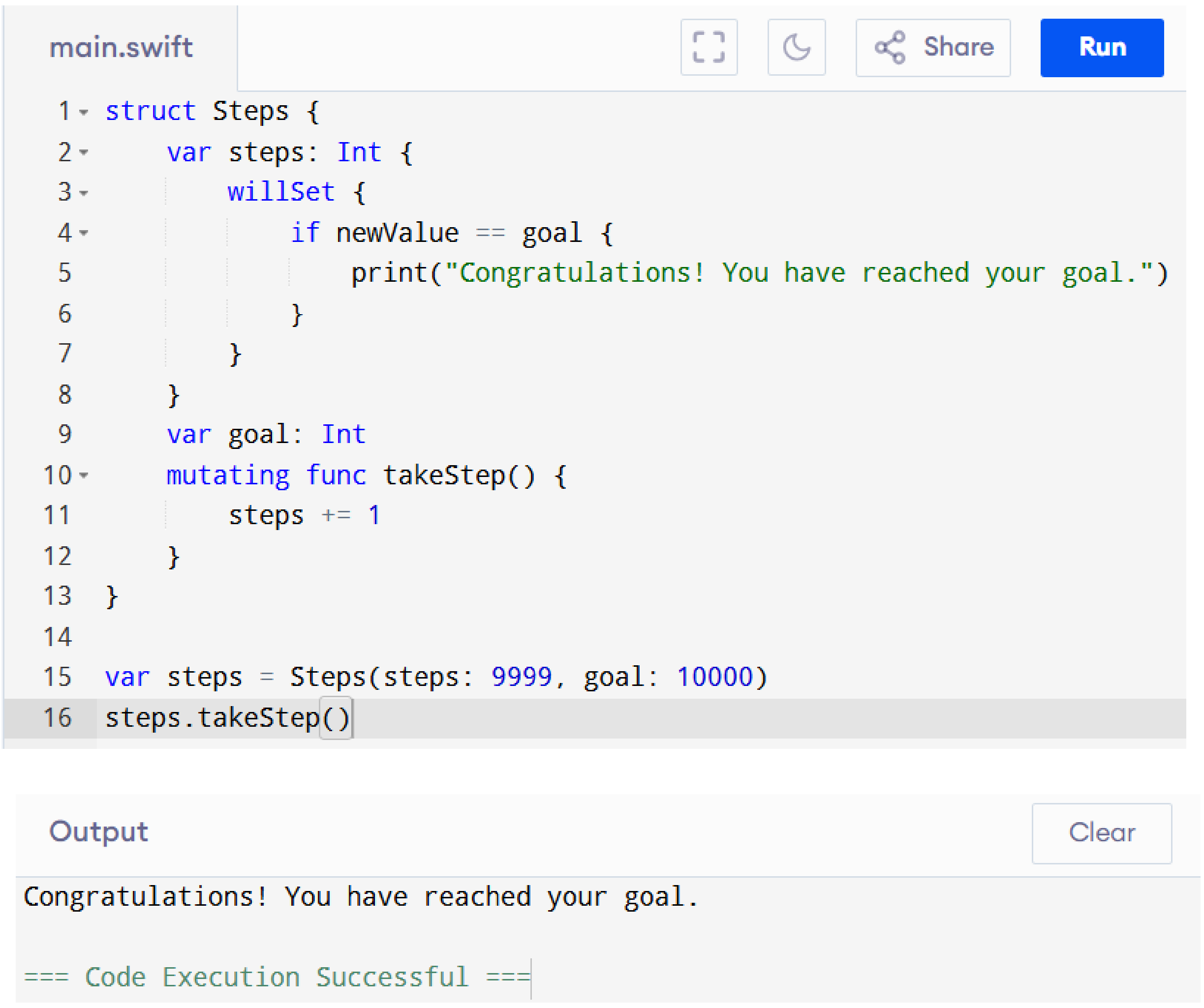
```
struct RunningWorkout {
    var distance: Double
    var time: Double
    var elevation: Double
    var averageMileTime: Double {
        return time / (distance / 1600)
    }
}

var workout = RunningWorkout(distance: 1600, time: 480, elevation: 100
    )
print("Average mile time: \(workout.averageMileTime)")
```



```
Average mile time: 480.0

=== Code Execution Successful ===
```

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 are 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
    }
}
```



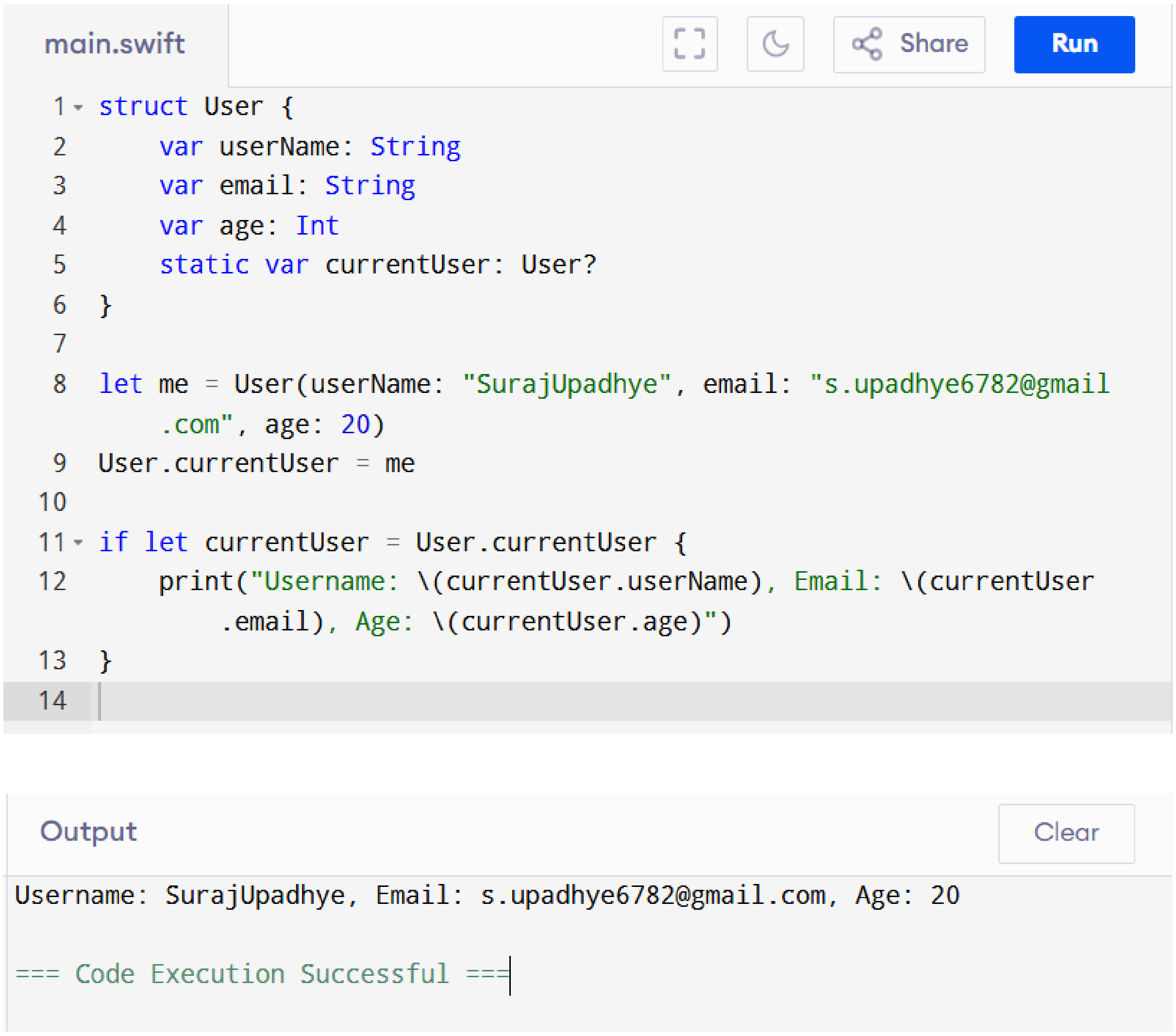
```
struct Steps {
    var steps: Int {
        willSet {
            if newValue == goal {
                print("Congratulations! You have reached your goal.")
            }
        }
    }
    var goal: Int
    mutating func takeStep() {
        steps += 1
    }
}

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

```
Congratulations! You have reached your goal.

=== Code Execution Successful ===
```

Type Properties and Methods

17. 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.

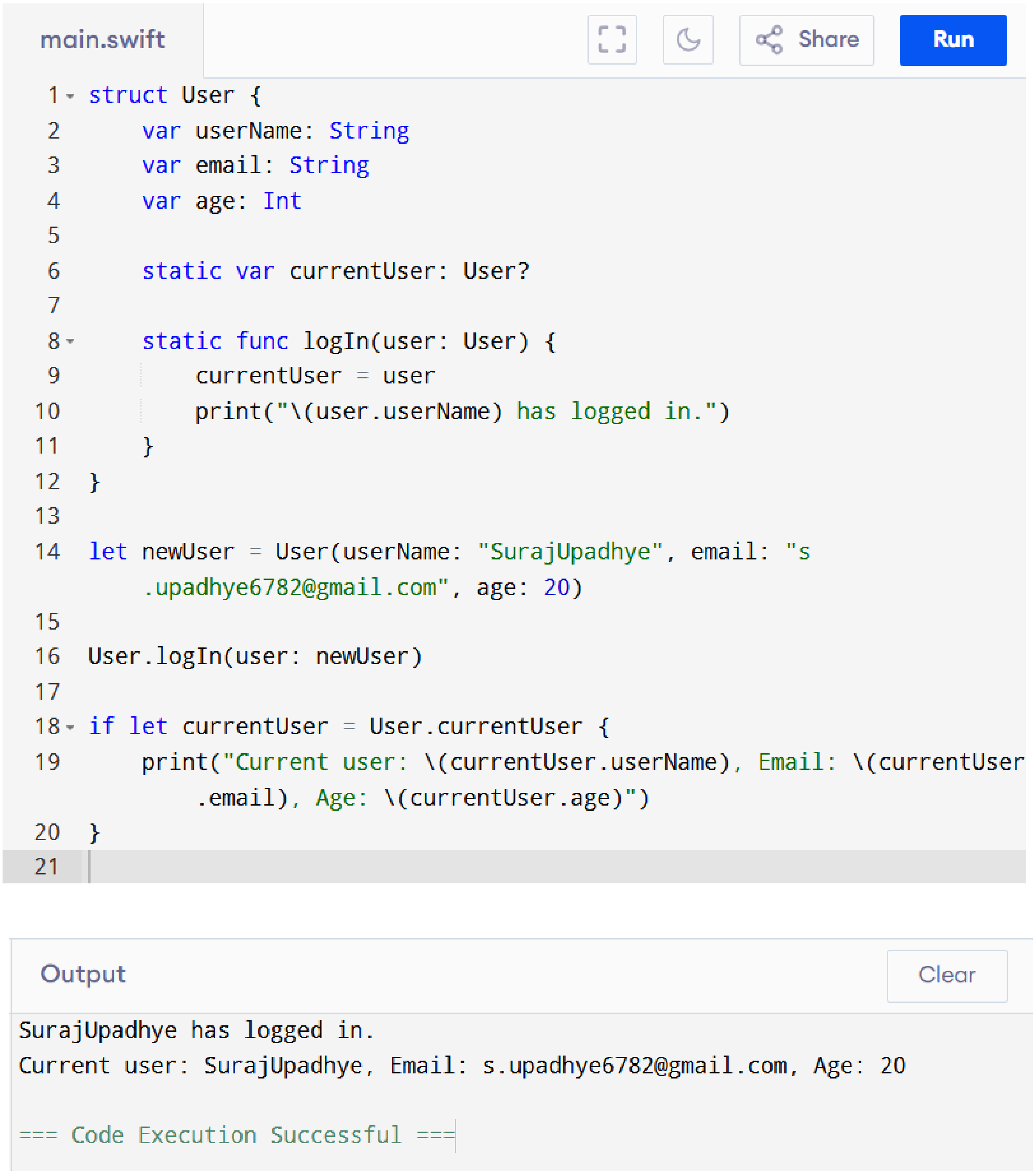
```
struct User {
    var userName: String
    var email: String
    var age: Int
}
```



```swift
struct User {
    var userName: String
    var email: String
    var age: Int
    static var currentUser: User?
}

let me = User(userName: "SurajUpadhye", email: "s.upadhye6782@gmail
    .com", age: 20)
User.currentUser = me

if let currentUser = User.currentUser {
    print("Username: \(currentUser.userName), Email: \(currentUser
        .email), Age: \(currentUser.age)")
}
```

```
Username: SurajUpadhye, Email: s.upadhye6782@gmail.com, Age: 20

=== Code Execution Successful ===
```

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.



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

    static var currentUser: User?

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

let newUser = User(userName: "SurajUpadhye", email: "s
    .upadhye6782@gmail.com", age: 20)

User.logIn(user: newUser)

if let currentUser = User.currentUser {
    print("Current user: \(currentUser.userName), Email: \(currentUser
        .email), Age: \(currentUser.age)")
}
```

```
SurajUpadhye has logged in.
Current user: SurajUpadhye, Email: s.upadhye6782@gmail.com, Age: 20

=== Code Execution Successful ===
```

Type Properties and Methods

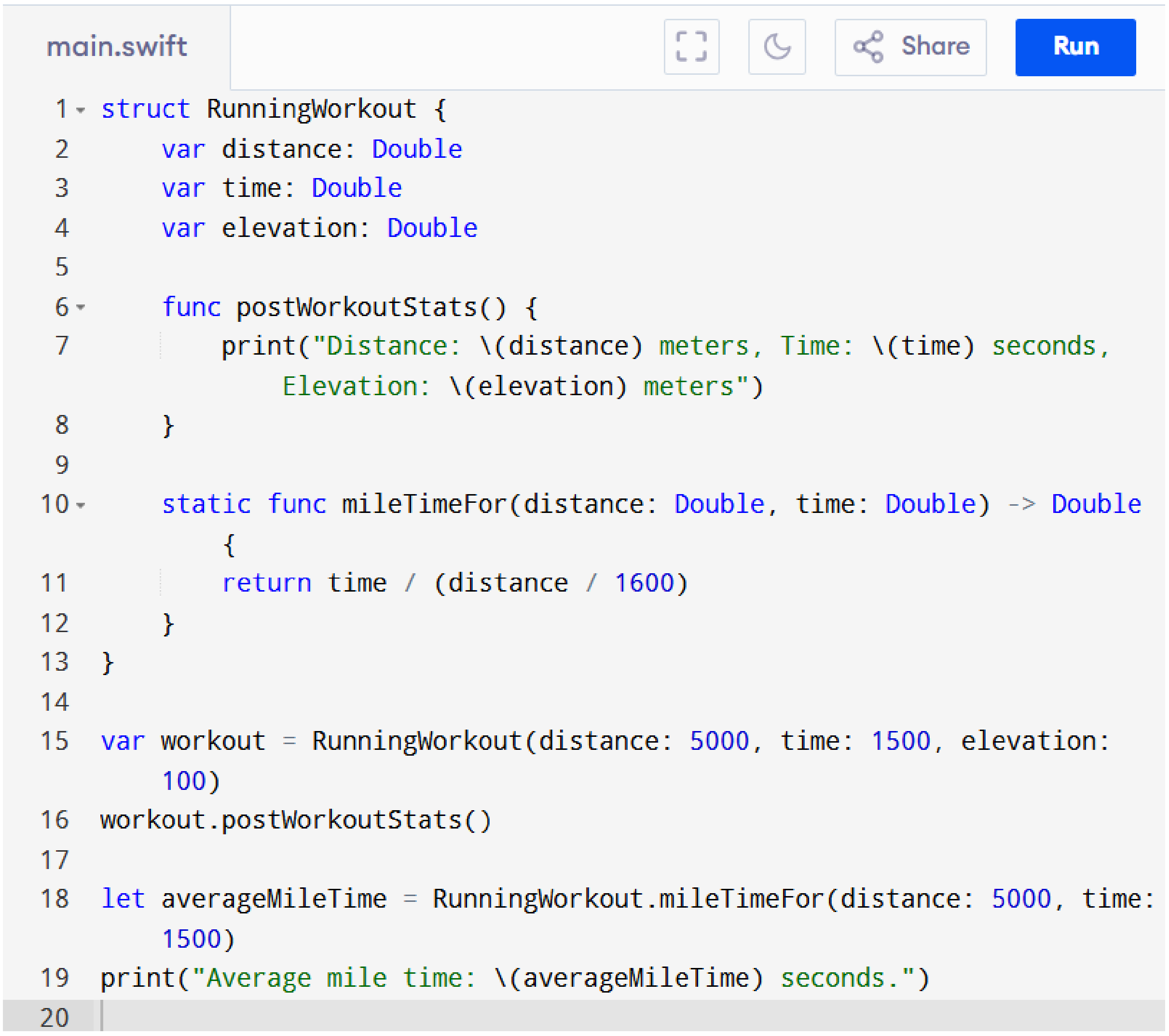
18. In another exercise, you added a computed property representing the average mile time from a run. However, you may want to have a calculator of sorts that users can use before their run to find out what mile time they need to average in order to run a given distance in a given time. In this case 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 {

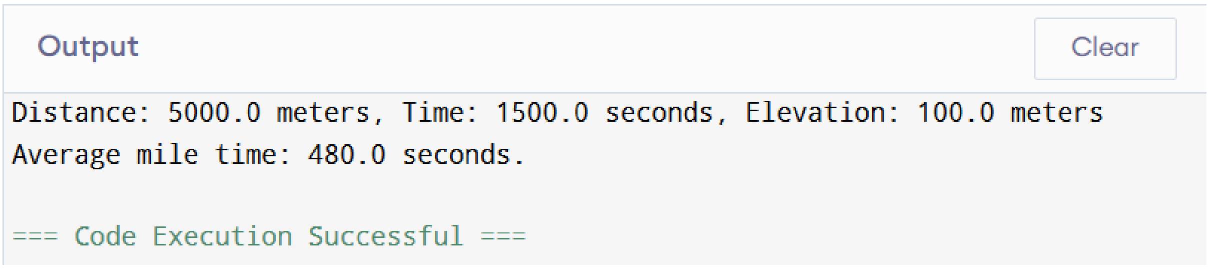
var distance: Double

var time: Double

var elevation: Double }

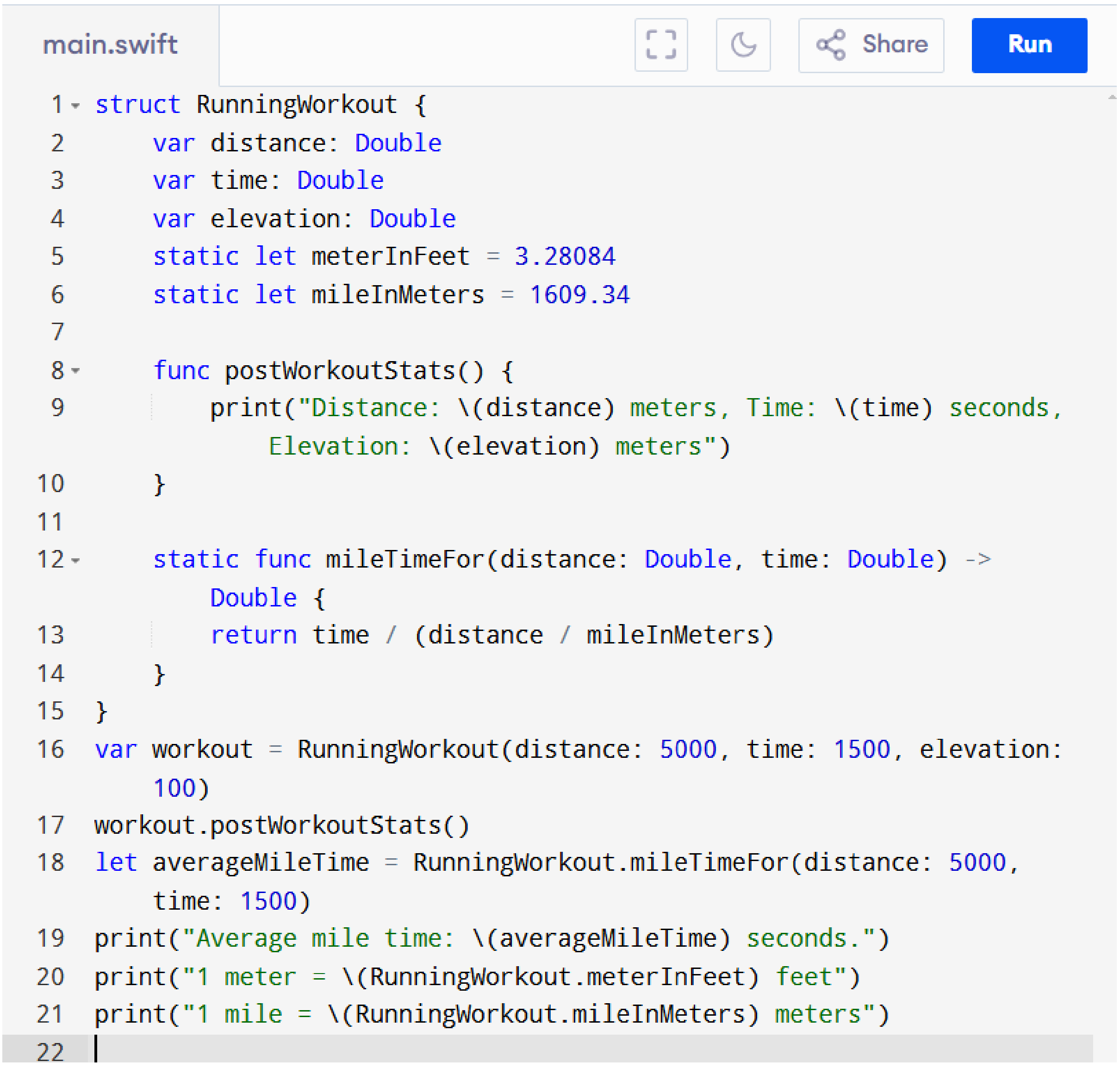


```swift
struct RunningWorkout {
    var distance: Double
    var time: Double
    var elevation: Double

    func postWorkoutStats() {
        print("Distance: \(distance) meters, Time: \(time) seconds,
            Elevation: \(elevation) meters")
    }

    static func mileTimeFor(distance: Double, time: Double) -> Double
        {
        return time / (distance / 1600)
    }
}

var workout = RunningWorkout(distance: 5000, time: 1500, elevation:
    100)
workout.postWorkoutStats()

let averageMileTime = RunningWorkout.mileTimeFor(distance: 5000, time:
    1500)
print("Average mile time: \(averageMileTime) seconds.")
```

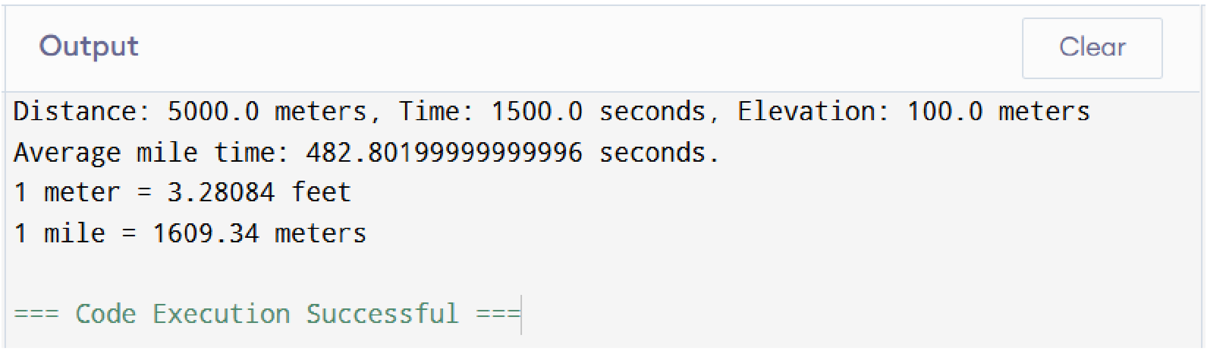


```
Distance: 5000.0 meters, Time: 1500.0 seconds, Elevation: 100.0 meters
Average mile time: 480.0 seconds.

=== Code Execution Successful ===
```

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.



```swift
struct RunningWorkout {
    var distance: Double
    var time: Double
    var elevation: Double
    static let meterInFeet = 3.28084
    static let mileInMeters = 1609.34

    func postWorkoutStats() {
        print("Distance: \(distance) meters, Time: \(time) seconds,
            Elevation: \(elevation) meters")
    }

    static func mileTimeFor(distance: Double, time: Double) ->
        Double {
        return time / (distance / mileInMeters)
    }
}
var workout = RunningWorkout(distance: 5000, time: 1500, elevation:
    100)
workout.postWorkoutStats()
let averageMileTime = RunningWorkout.mileTimeFor(distance: 5000,
    time: 1500)
print("Average mile time: \(averageMileTime) seconds.")
print("1 meter = \(RunningWorkout.meterInFeet) feet")
print("1 mile = \(RunningWorkout.mileInMeters) meters")

```



```
Distance: 5000.0 meters, Time: 1500.0 seconds, Elevation: 100.0 meters
Average mile time: 482.80199999999996 seconds.
1 meter = 3.28084 feet
1 mile = 1609.34 meters

=== Code Execution Successful ===
```

****