Department of Computer Science and Engineering

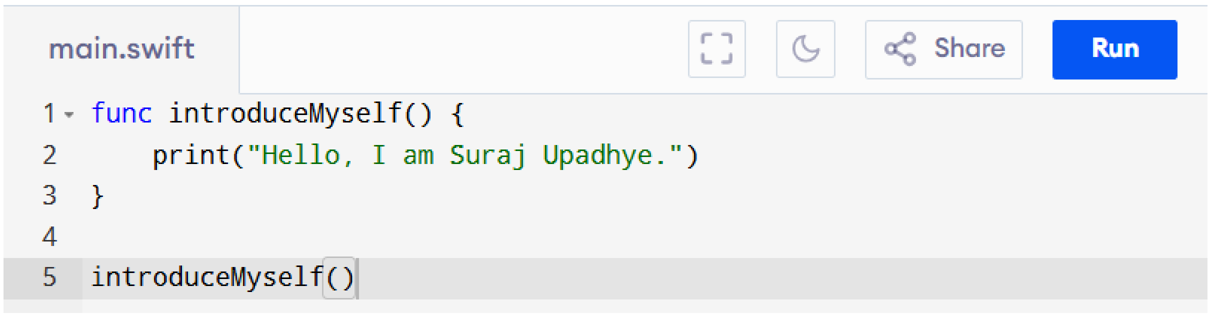
Sub: iOS Lab

Assignment No 3 : Functions

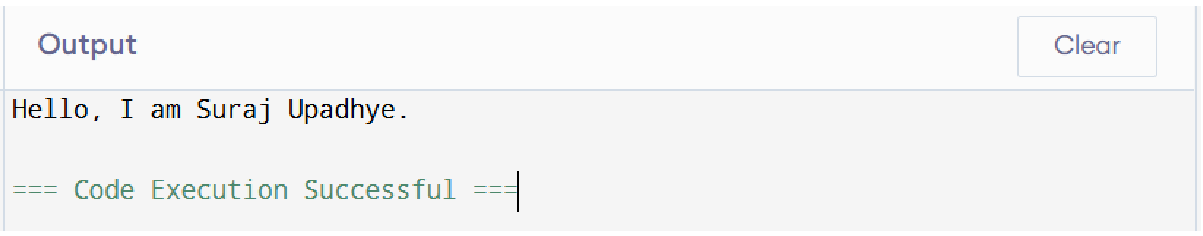
Name : Suraj Shantinath Upadhye. **PRN :** 245200001 **Class :** SY CSE

## Exercise - Create Functions

1. Write a function called `introduceMyself` that prints a brief introduction of yourself. Call the function and observe the printout.



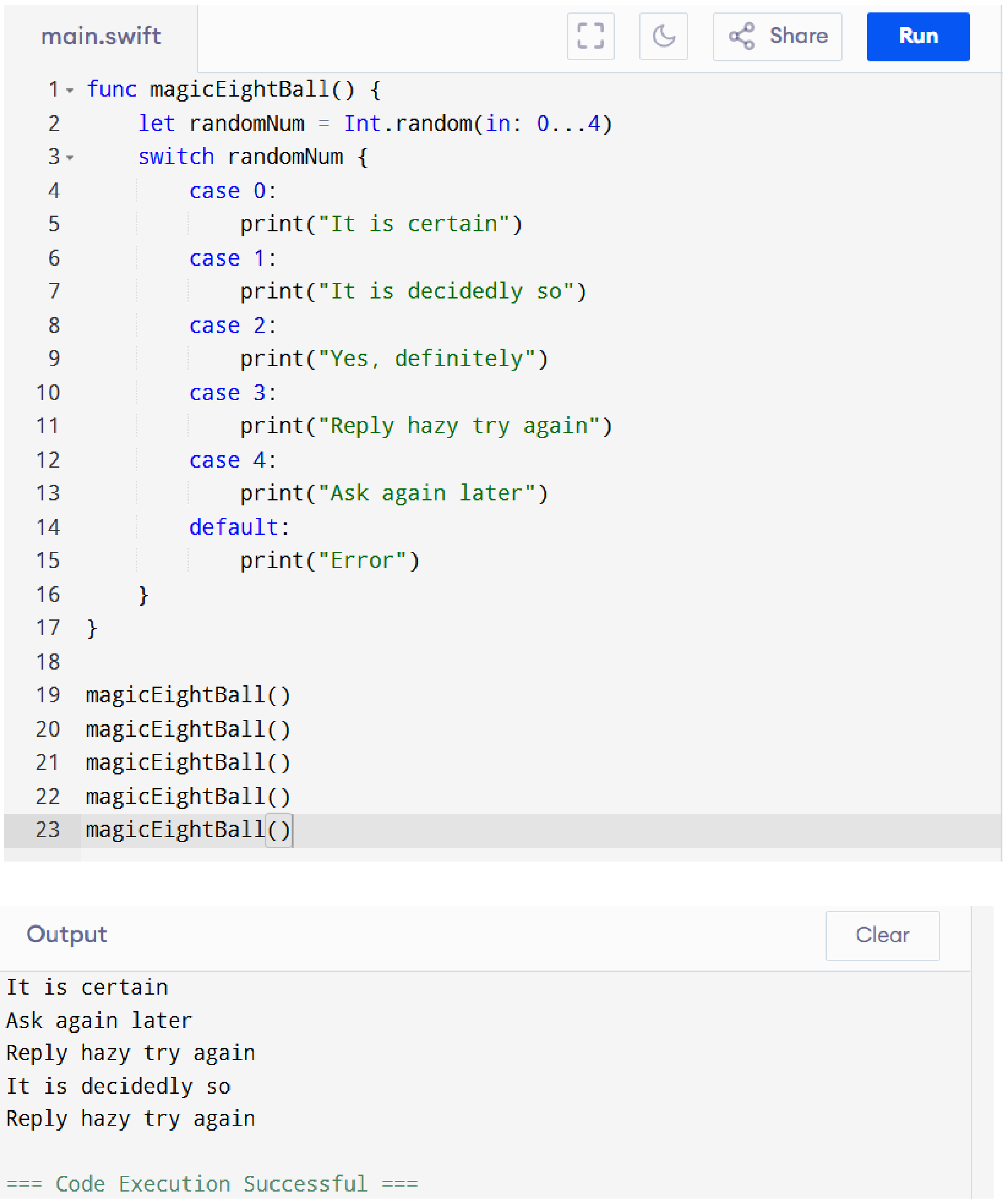
```swift
func introduceMyself() {
    print("Hello, I am Suraj Upadhye.")
}

introduceMyself()
```



```
Hello, I am Suraj Upadhye.

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

2. Write a function called `magicEightBall` that generates a random number and then uses either a switch statement or if-else-if statements to print different responses based on the random number generated. `let randomNum = Int.random(in: 0...4)` will generate a random number from 0 to 4, after which you can print different phrases corresponding to the number generated. Call the function multiple times and observe the different printouts.

import Foundation



```swift
func magicEightBall() {
    let randomNum = Int.random(in: 0...4)
    switch randomNum {
        case 0:
            print("It is certain")
        case 1:
            print("It is decidedly so")
        case 2:
            print("Yes, definitely")
        case 3:
            print("Reply hazy try again")
        case 4:
            print("Ask again later")
        default:
            print("Error")
    }
}

magicEightBall()
magicEightBall()
magicEightBall()
magicEightBall()
magicEightBall()
```

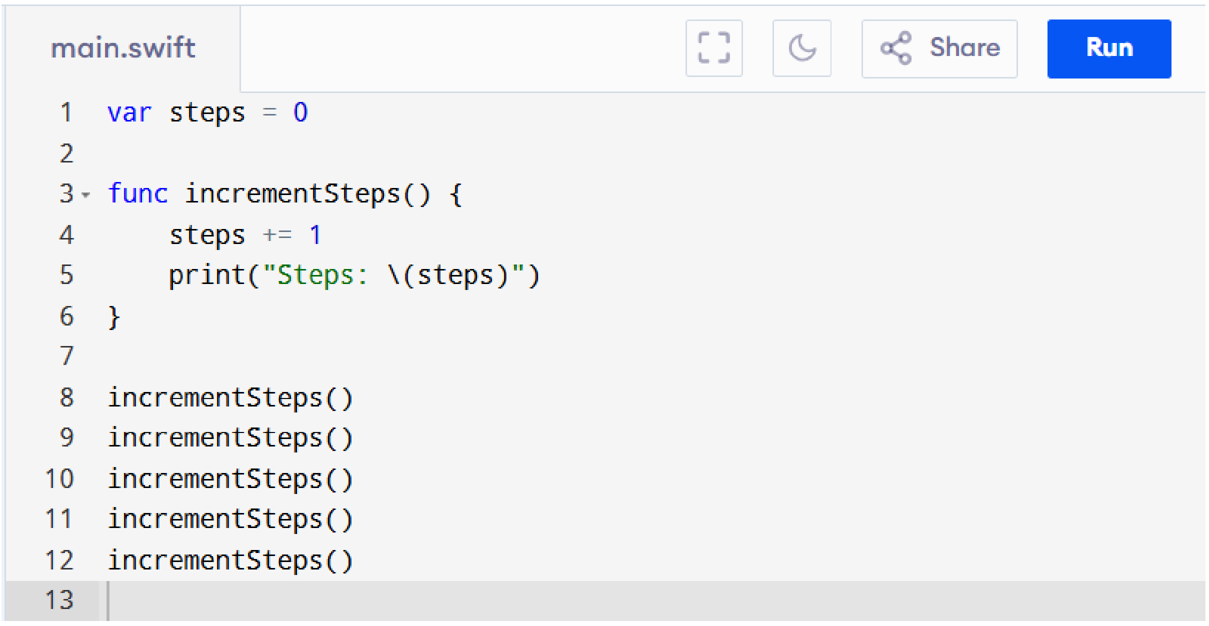
```
It is certain
Ask again later
Reply hazy try again
It is decidedly so
Reply hazy try again

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

## App Exercise - A Functioning App

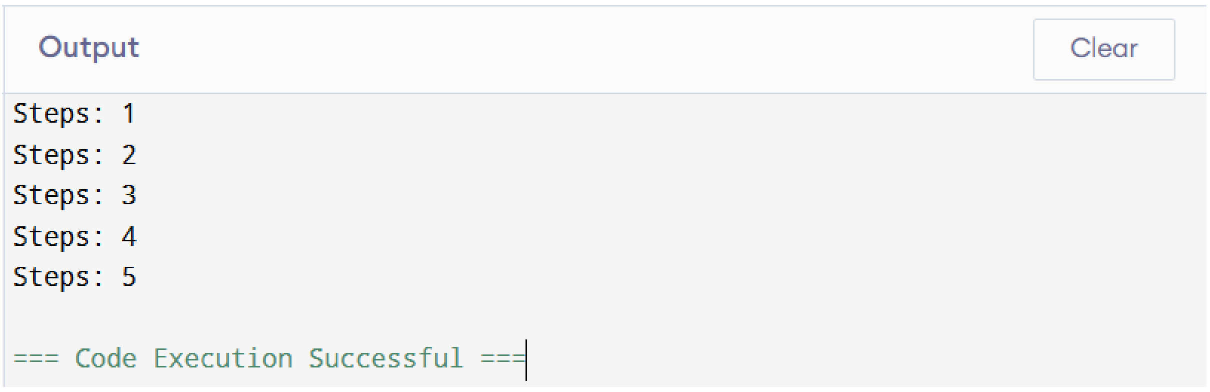
3. As you may have guessed, functions are key to making your app work. For example, in every exercise dealing with step count until now, you have simply assigned a number of steps to a `steps` variable. This isn't very realistic seeing as the number of steps you take increments one at a time and continues changing throughout the day.

A reoccurring process like this is a perfect candidate for a function. Write a function called `incrementSteps` after the declaration of `steps` below that will increment `steps` by one and then print its value. Call the function multiple times and observe the printouts.

var steps = 0



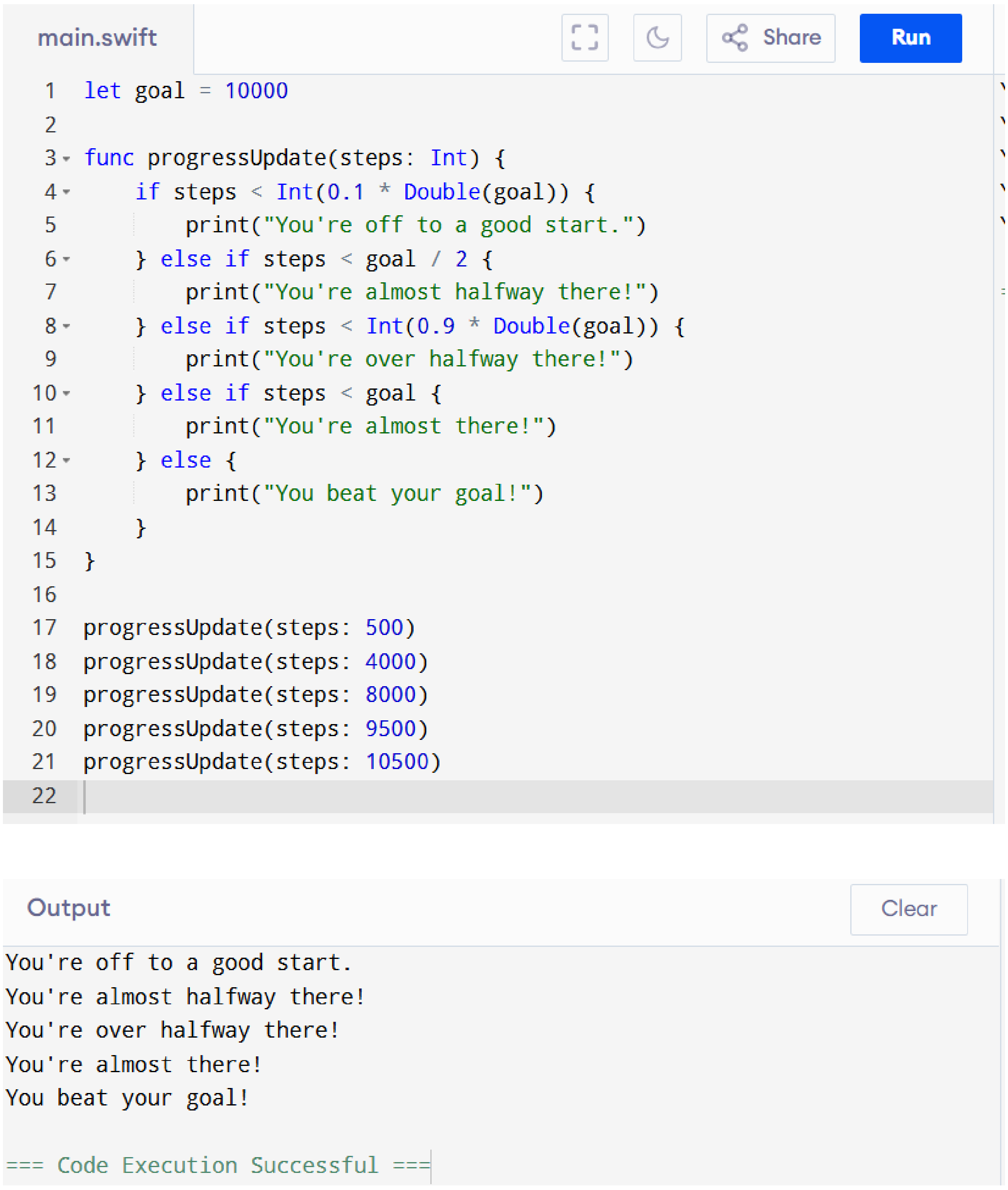
```swift
var steps = 0

func incrementSteps() {
    steps += 1
    print("Steps: \(steps)")
}

incrementSteps()
incrementSteps()
incrementSteps()
incrementSteps()
incrementSteps()
```



```
Steps: 1
Steps: 2
Steps: 3
Steps: 4
Steps: 5

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

4. Similarly, if you want to regularly provide progress updates to your user, you can put your control flow statements that check on progress into a function. Write a function called `progressUpdate` after the declaration of `goal` below. The function should print "You're off to a good start." if `steps` is less than 10% of `goal`, "You're almost halfway there!" if `steps` is less than half of `goal`, "You're over halfway there!" if `steps` is less than 90% of `goal`, "You're almost there!" if `steps` is less than `goal`, and "You beat your goal!" otherwise. Call the function and observe the printout.

let goal = 10000

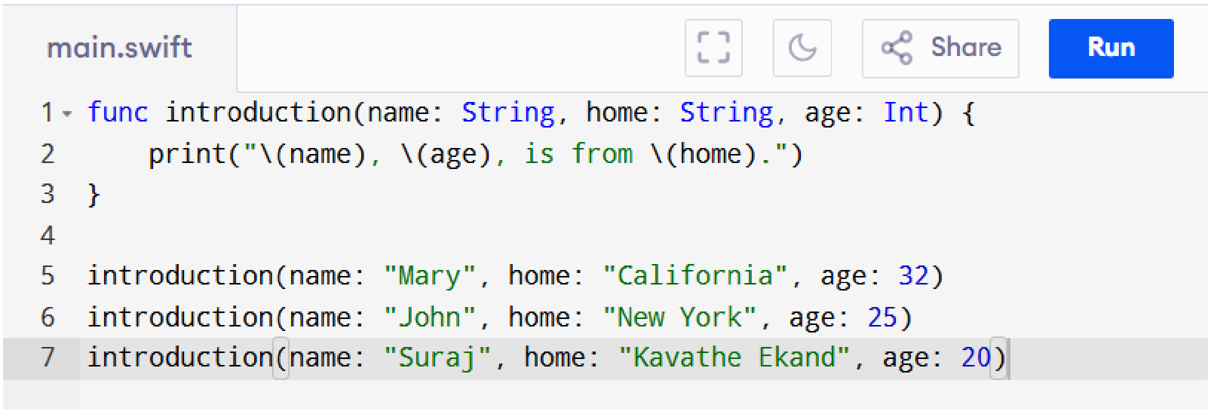


```swift
let goal = 10000

func progressUpdate(steps: Int) {
    if steps < Int(0.1 * Double(goal)) {
        print("You're off to a good start.")
    } else if steps < goal / 2 {
        print("You're almost halfway there!")
    } else if steps < Int(0.9 * Double(goal)) {
        print("You're over halfway there!")
    } else if steps < goal {
        print("You're almost there!")
    } else {
        print("You beat your goal!")
    }
}

progressUpdate(steps: 500)
progressUpdate(steps: 4000)
progressUpdate(steps: 8000)
progressUpdate(steps: 9500)
progressUpdate(steps: 10500)
```

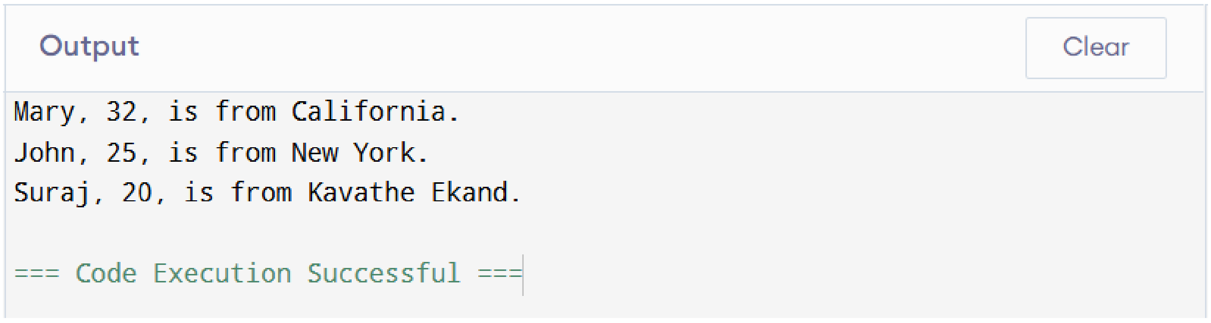
```
You're off to a good start.
You're almost halfway there!
You're over halfway there!
You're almost there!
You beat your goal!

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

## Exercise - Parameters and Argument Labels

5. Write a new introduction function called `introduction`. It should take two `String` parameters,
`name` and `home`, and one `Int` parameter, `age`. The function should print a brief introduction. I.e. if "Mary," "California," and 32 were passed into the function, it might print "Mary, 32, is from California." Call the function and observe the printout.

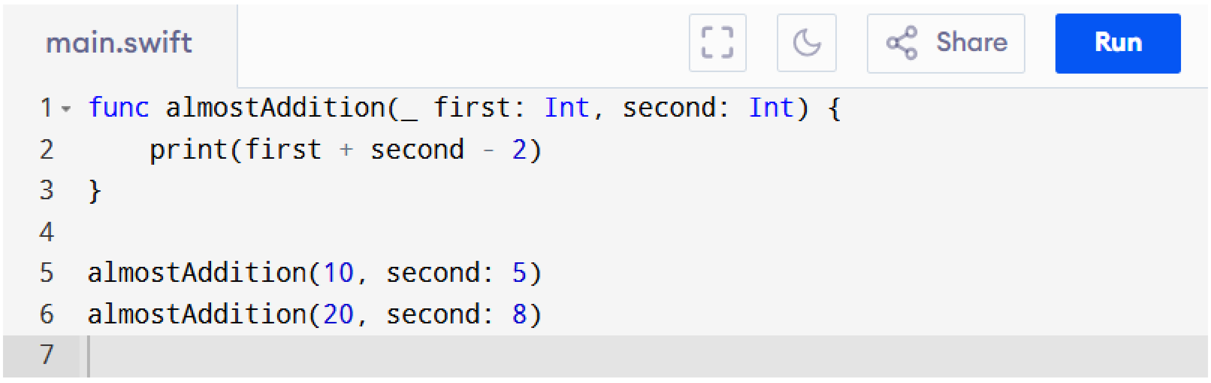


```swift
func introduction(name: String, home: String, age: Int) {
    print("\(name), \(age), is from \(home).")
}

introduction(name: "Mary", home: "California", age: 32)
introduction(name: "John", home: "New York", age: 25)
introduction(name: "Suraj", home: "Kavathe Ekand", age: 20)
```

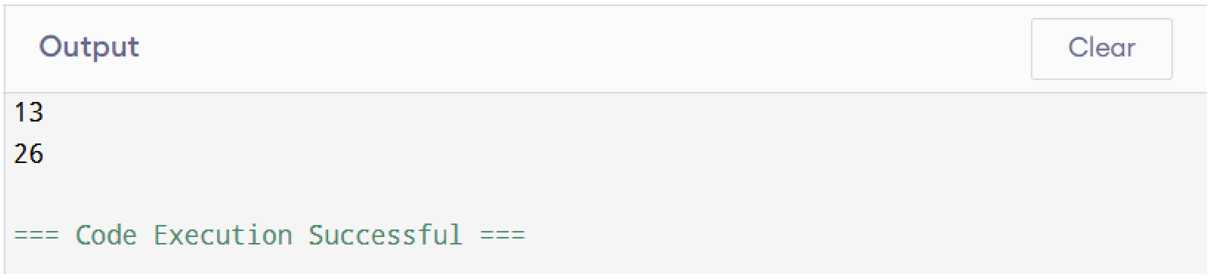


```
Mary, 32, is from California.
John, 25, is from New York.
Suraj, 20, is from Kavathe Ekand.

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

6. Write a function called `almostAddition` that takes two `Int` arguments. The first argument should not require an argument label. The function should add the two arguments together, subtract 2, then print the result. Call the function and observe the printout.

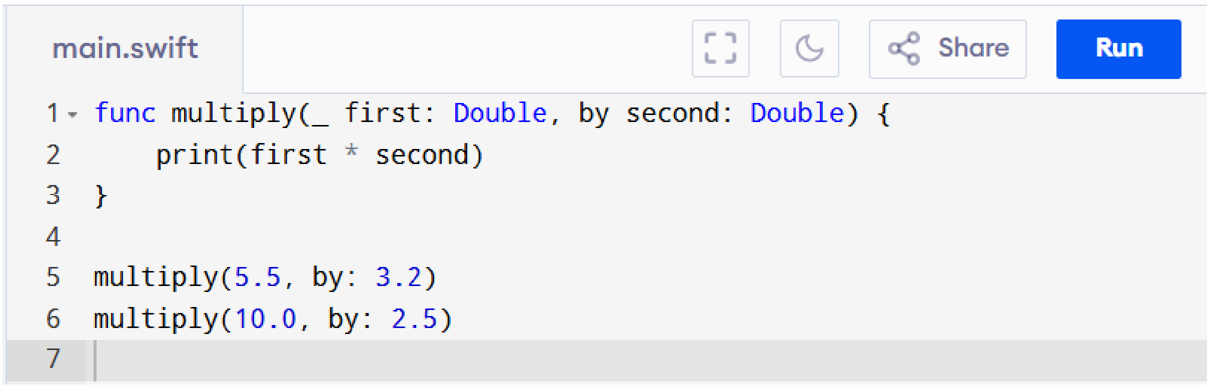


```swift
func almostAddition(_ first: Int, second: Int) {
    print(first + second - 2)
}

almostAddition(10, second: 5)
almostAddition(20, second: 8)
```

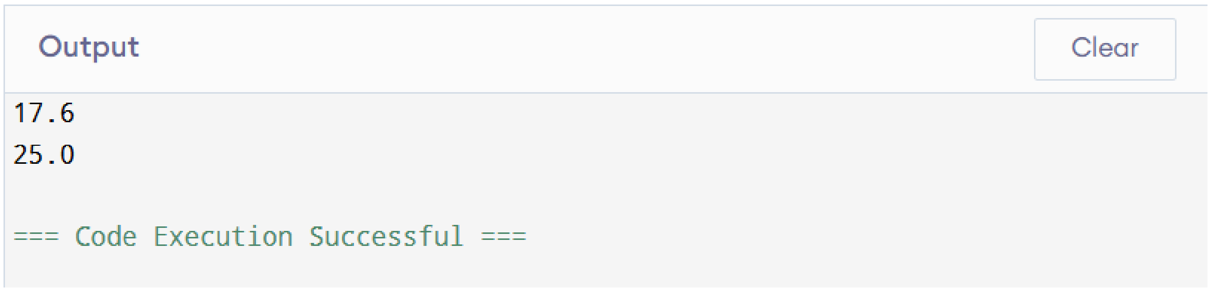


```
13
26

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

7. Write a function called `multiply` that takes two `Double` arguments. The function should multiply the two arguments and print the result. The first argument should not require a label, and the second argument should have an external label, `by`, that differs from the internal label. Call the function and observe the printout.



```swift
func multiply(_ first: Double, by second: Double) {
    print(first * second)
}

multiply(5.5, by: 3.2)
multiply(10.0, by: 2.5)
```



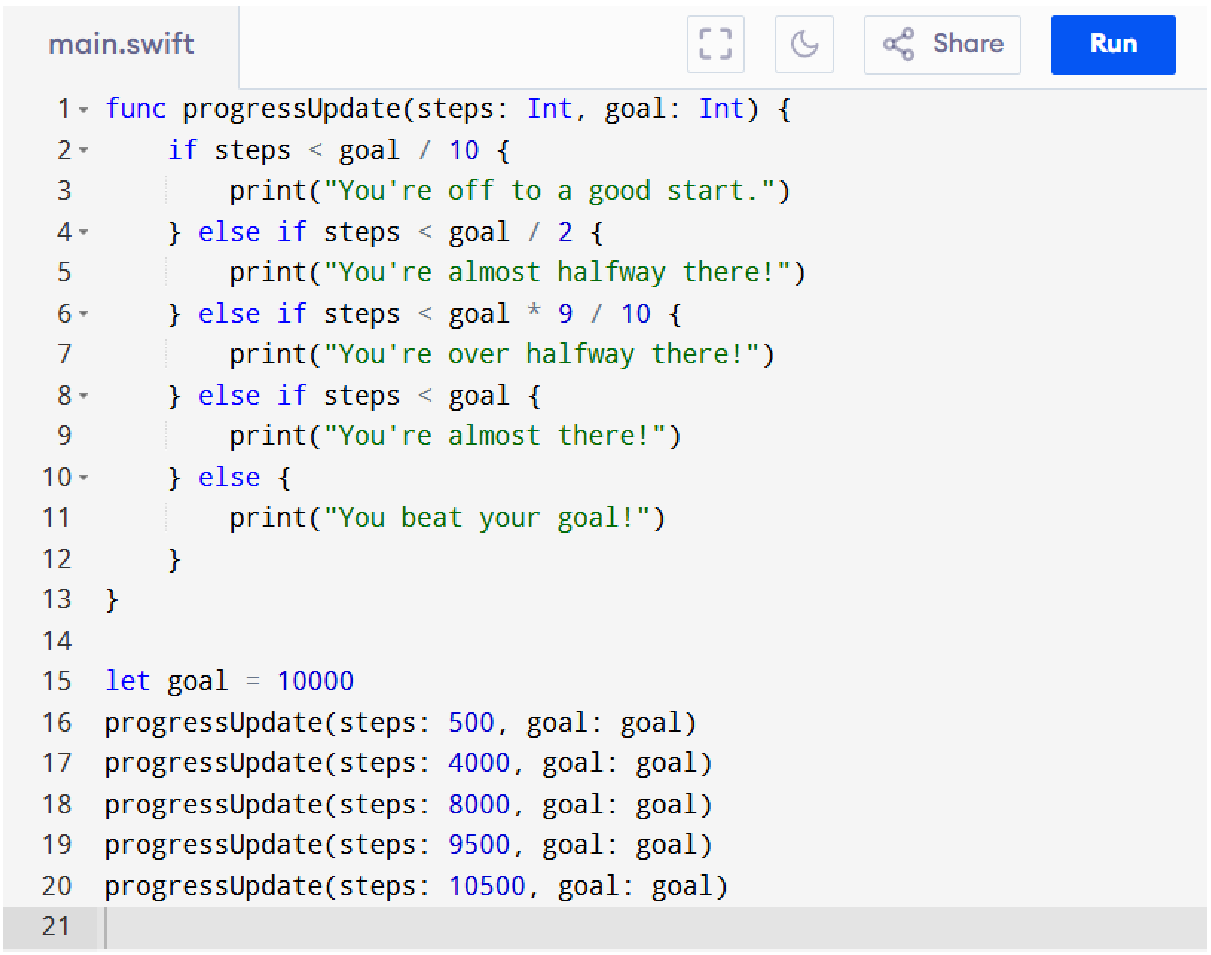
```
17.6
25.0

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

## App Exercise - Progress Updates

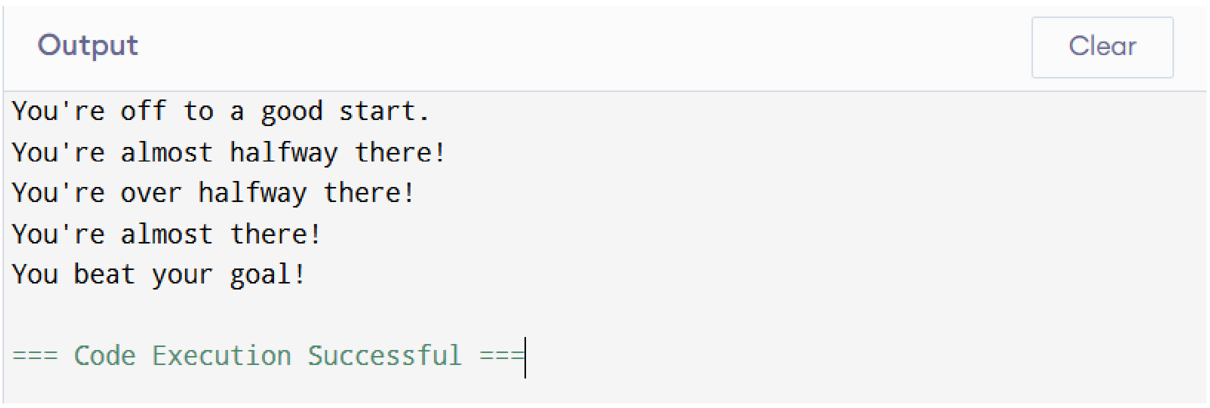
8. In many cases you want to provide input to a function. For example, the progress function you wrote in the Functioning App exercise might be located in an area of your project that doesn't have access to the value of `steps` and `goal`. In that case, whenever you called the function, you would need to provide it with the number of steps that have been taken and the goal for the day so it can print the correct progress statement.

Rewrite the function `progressUpdate`, only this time give it two parameters of type `Int` called `steps` and `goal`, respectively. Like before, it should print "You're off to a good start." if steps is less than 10% of goal, "You're almost halfway there!" if steps is less than half of goal, "You're over halfway there!" if steps is less than 90% of goal, "You're almost there!" if steps is less than goal, and "You beat your goal!" otherwise. Call the function and observe the printout.

Call the function a number of times, passing in different values of `steps` and `goal`. Observe the printouts and make sure what is printed to the console is what you would expect for the parameters passsed in.

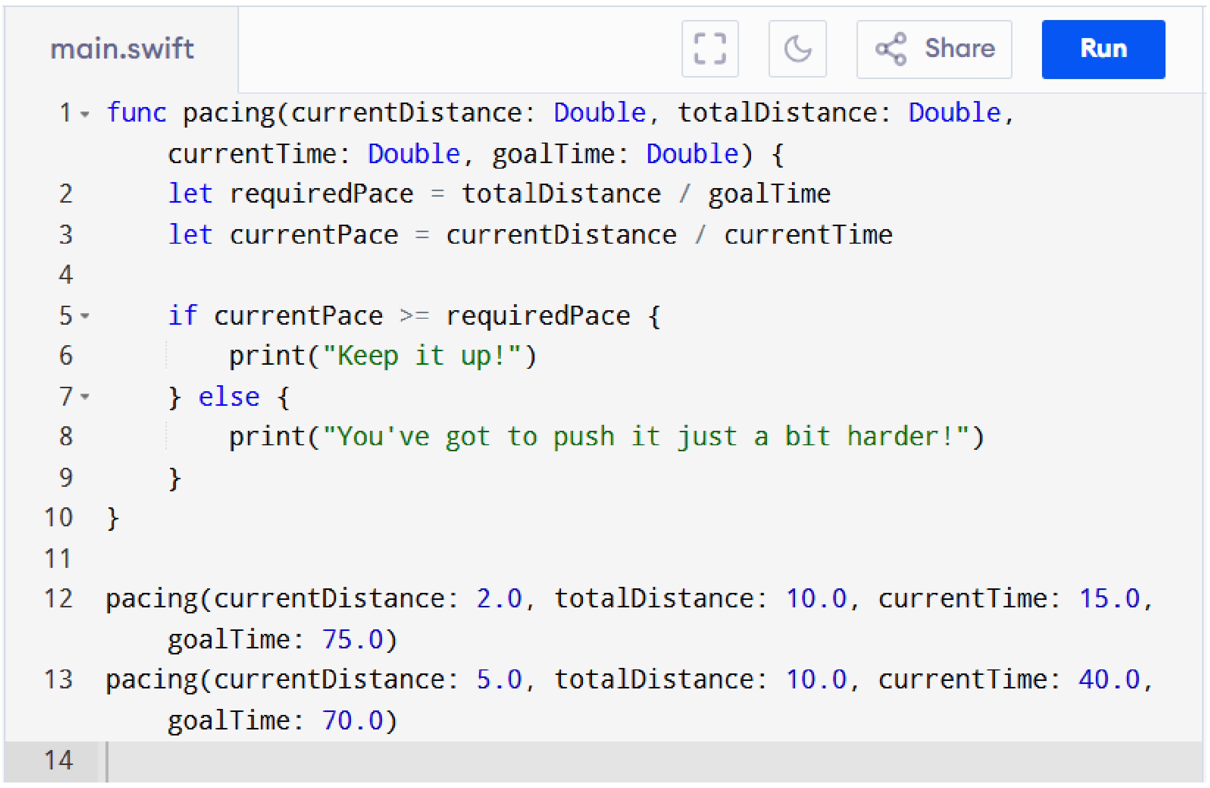


```swift
func progressUpdate(steps: Int, goal: Int) {
    if steps < goal / 10 {
        print("You're off to a good start.")
    } else if steps < goal / 2 {
        print("You're almost halfway there!")
    } else if steps < goal * 9 / 10 {
        print("You're over halfway there!")
    } else if steps < goal {
        print("You're almost there!")
    } else {
        print("You beat your goal!")
    }
}

let goal = 10000
progressUpdate(steps: 500, goal: goal)
progressUpdate(steps: 4000, goal: goal)
progressUpdate(steps: 8000, goal: goal)
progressUpdate(steps: 9500, goal: goal)
progressUpdate(steps: 10500, goal: goal)
```

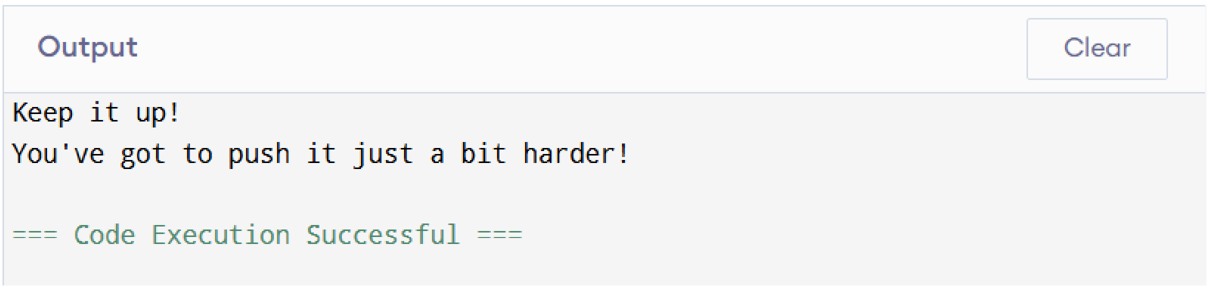


```
You're off to a good start.
You're almost halfway there!
You're over halfway there!
You're almost there!
You beat your goal!

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

9. Your fitness tracking app is going to help runners stay on pace to reach their goals. Write a function called pacing that takes four `Double` parameters called `currentDistance`, `totalDistance`, `currentTime`, and `goalTime`. Your function should calculate whether or not the user is on pace to hit or beat `goalTime`. If yes, print "Keep it up!", otherwise print "You've got to push it just a bit harder!"



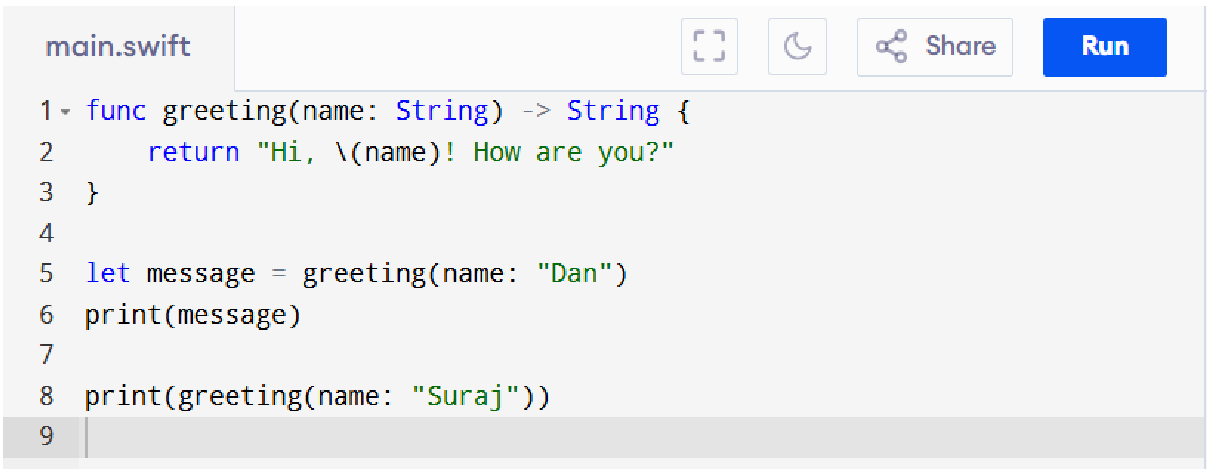
```swift
func pacing(currentDistance: Double, totalDistance: Double,
    currentTime: Double, goalTime: Double) {
    let requiredPace = totalDistance / goalTime
    let currentPace = currentDistance / currentTime

    if currentPace >= requiredPace {
        print("Keep it up!")
    } else {
        print("You've got to push it just a bit harder!")
    }
}

pacing(currentDistance: 2.0, totalDistance: 10.0, currentTime: 15.0,
    goalTime: 75.0)
pacing(currentDistance: 5.0, totalDistance: 10.0, currentTime: 40.0,
    goalTime: 70.0)
```



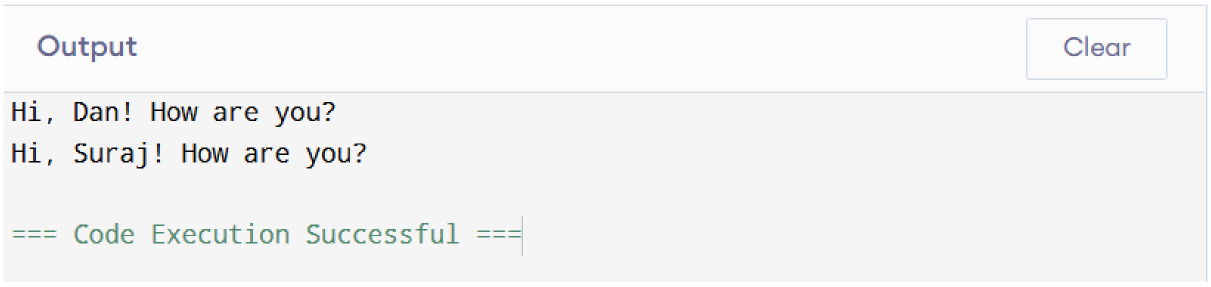
```
Keep it up!
You've got to push it just a bit harder!

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

## Exercise - Return Values

10. Write a function called `greeting` that takes a `String` argument called name, and returns a `String` that greets the name that was passed into the function. I.e. if you pass in "Dan" the return value might be "Hi, Dan! How are you?" Use the function and print the result.

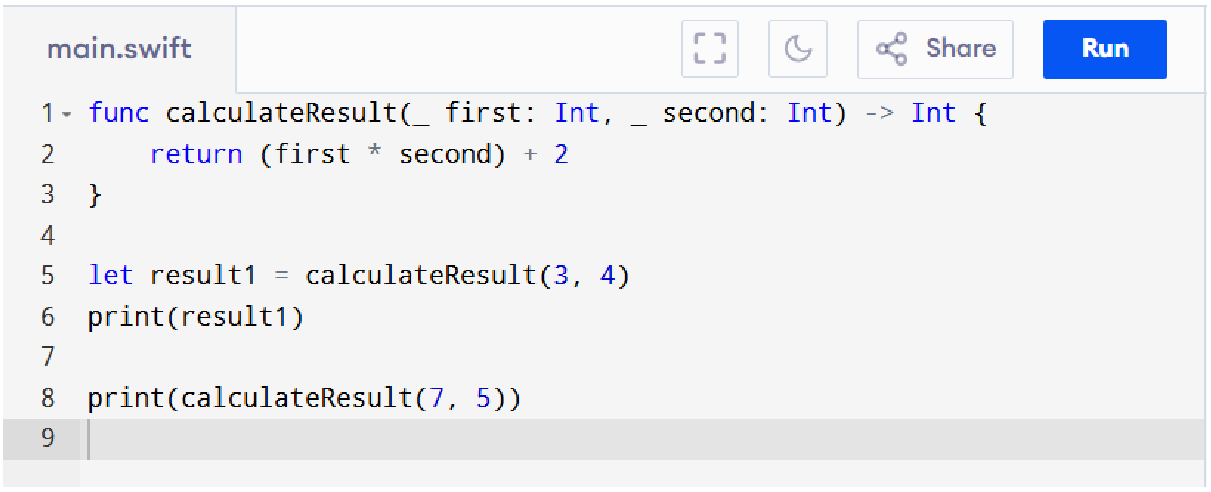


```swift
func greeting(name: String) -> String {
    return "Hi, \(name)! How are you?"
}

let message = greeting(name: "Dan")
print(message)

print(greeting(name: "Suraj"))

```

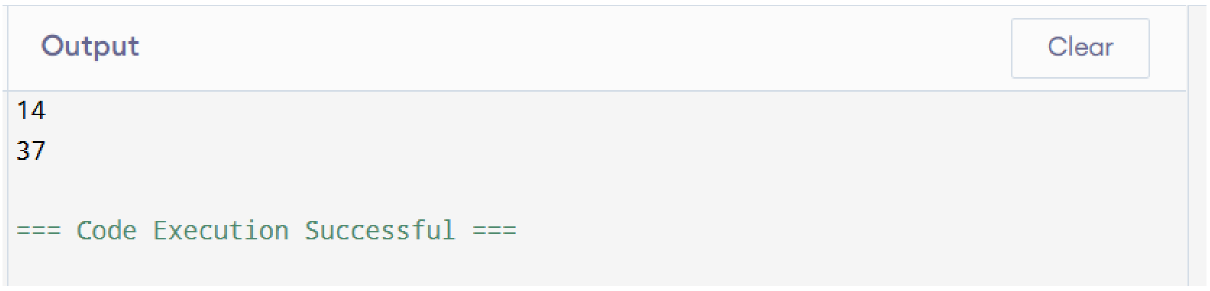


```
Hi, Dan! How are you?
Hi, Suraj! How are you?

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

11. Write a function that takes two `Int` arguments, and returns an `Int`. The function should multiply the two arguments, add 2, then return the result. Use the function and print the result.



```swift
func calculateResult(_ first: Int, _ second: Int) -> Int {
    return (first * second) + 2
}

let result1 = calculateResult(3, 4)
print(result1)

print(calculateResult(7, 5))

```

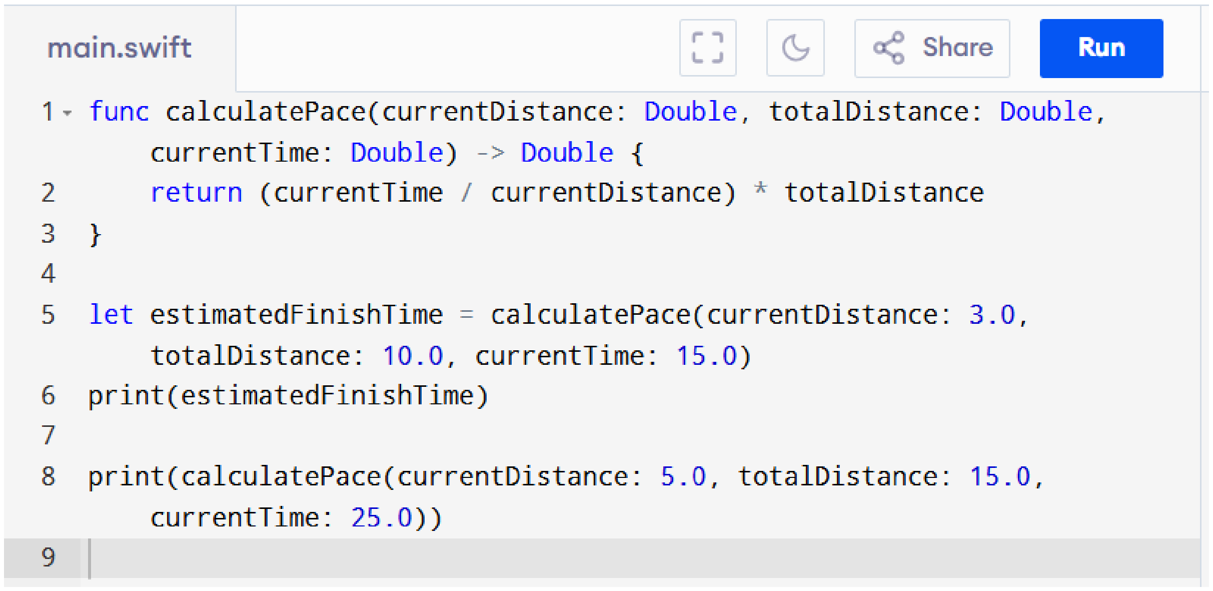


```
14
37

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

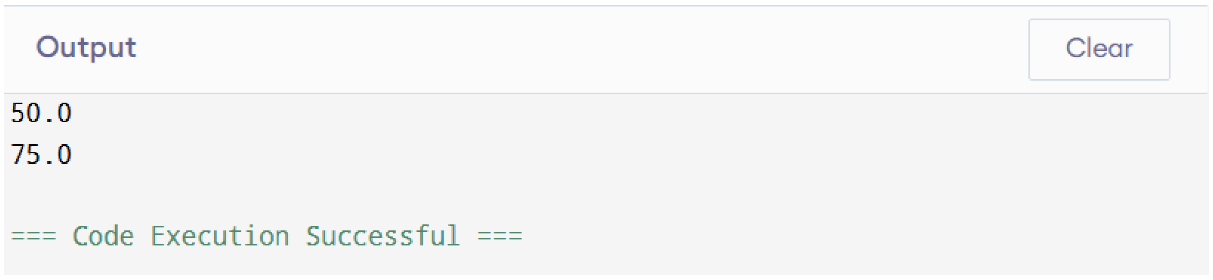
## App Exercise - Separating Functions

12. One principle that can help in debugging and maintaining code is abstraction. For example, in your fitness tracking app some of your existing functions have been written to both perform a calculation and print a message. But it's very possible that you'll decide to change either the calculation or the message in the future. It will be easier to go back and change this if you separate the calculation from the message.

As an example, write a function that only does a portion of what your previous `pacing` function did. This function will be called `calculatePace`. It should take three `Double` arguments called
`currentDistance`, `totalDistance`, and `currentTime`, and should return a `Double` that will represent the time at which the user will finish the run based on the user's current distance and time. call the function and print the return value.

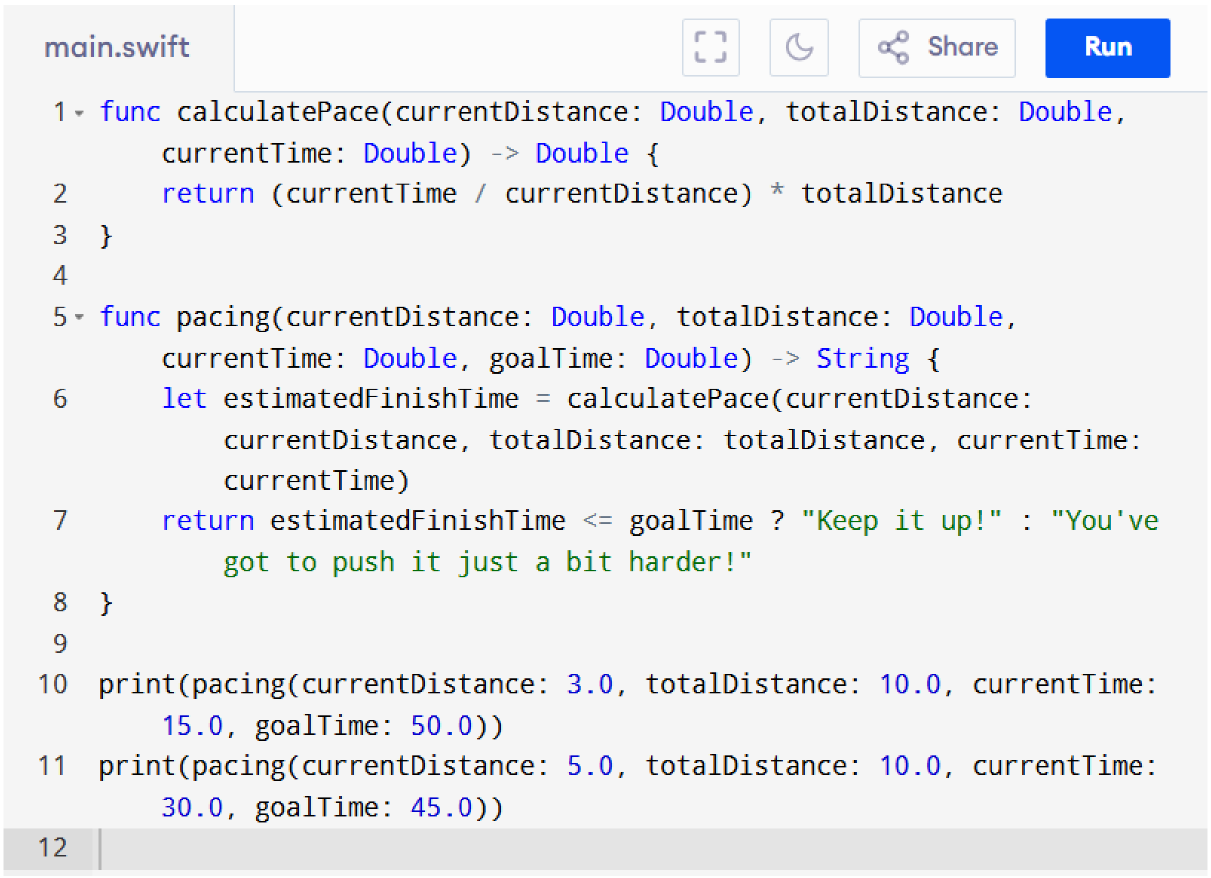


```swift
func calculatePace(currentDistance: Double, totalDistance: Double,
    currentTime: Double) -> Double {
    return (currentTime / currentDistance) * totalDistance
}

let estimatedFinishTime = calculatePace(currentDistance: 3.0,
    totalDistance: 10.0, currentTime: 15.0)
print(estimatedFinishTime)

print(calculatePace(currentDistance: 5.0, totalDistance: 15.0,
    currentTime: 25.0))

```

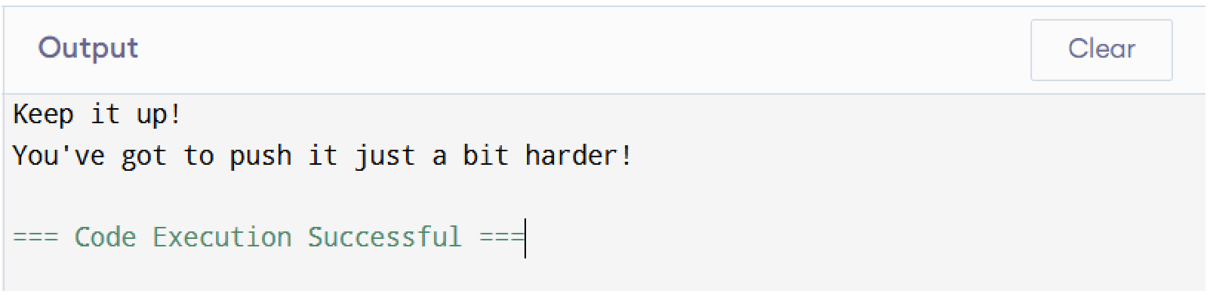


```
50.0
75.0

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

13. Now write a function called `pacing` that takes four `Double` arguments called `currentDistance`,
`totalDistance`, `currentTime`, and `goalTime`. The function should also return a `String`, which will be the message to show the user. The function should call `calculatePace`, passing in the appropriate values, and capture the return value. The function should then compare the returned value to
`goalTime` and if the user is on pace return "Keep it up!", and return "You've got to push it just a bit harder!" otherwise. Call the function and print the return value.



```swift
func calculatePace(currentDistance: Double, totalDistance: Double,
    currentTime: Double) -> Double {
    return (currentTime / currentDistance) * totalDistance
}

func pacing(currentDistance: Double, totalDistance: Double,
    currentTime: Double, goalTime: Double) -> String {
    let estimatedFinishTime = calculatePace(currentDistance:
        currentDistance, totalDistance: totalDistance, currentTime:
        currentTime)
    return estimatedFinishTime <= goalTime ? "Keep it up!" : "You've
        got to push it just a bit harder!"
}

print(pacing(currentDistance: 3.0, totalDistance: 10.0, currentTime:
    15.0, goalTime: 50.0))
print(pacing(currentDistance: 5.0, totalDistance: 10.0, currentTime:
    30.0, goalTime: 45.0))
```



```
Keep it up!
You've got to push it just a bit harder!

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

****