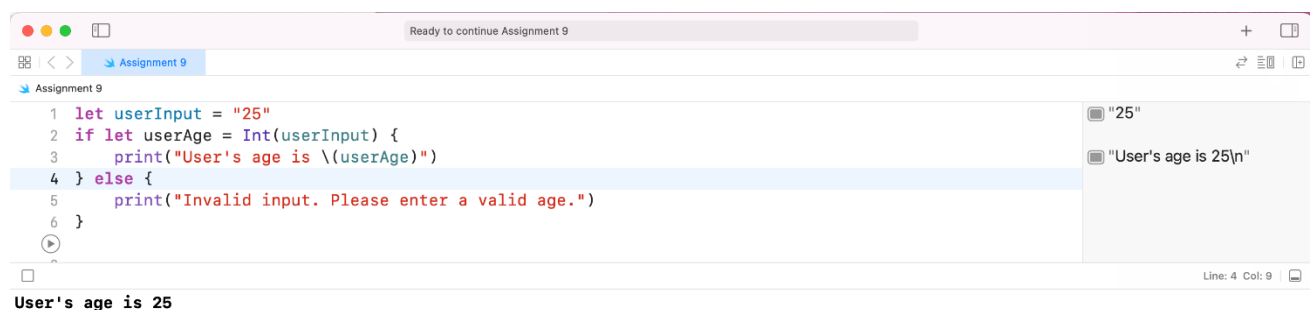


**Name: Khushi Nitinkumar Patel**  
**PRN: 2020BTECS00037**  
**Batch: T2**

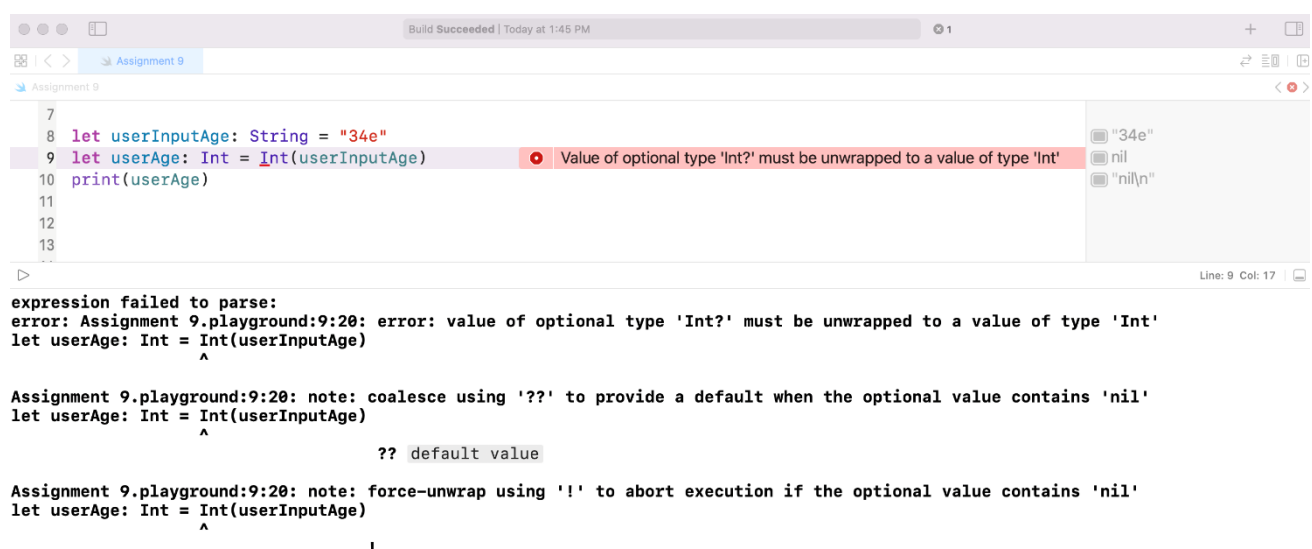
## Assignment No 9

1. Throughout the exercises in this playground, you will be printing optional values. The Swift compiler will display a warning: "Expression implicitly coerced from `Int?` to Any." For the purposes of these exercises, you can ignore this warning.

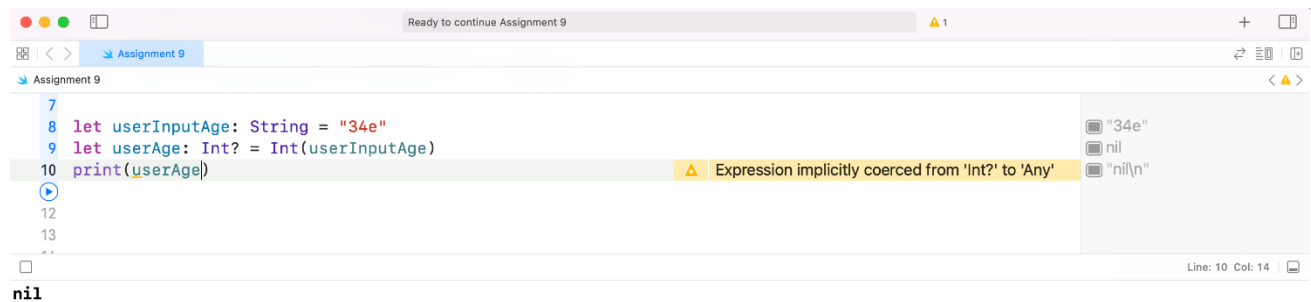
Imagine you have an app that asks the user to enter his/her age using the keyboard. When your app allows a user to input text, what is captured for you is given as a `String`. However, you want to store this information as an `Int`. Is it possible for the user to make a mistake and for the input to not match the type you want to store?



2. Declare a constant `userInputAge` of type `String` and assign it "34e" to simulate a typo while typing age. Then declare a constant `userAge` of type `Int` and set its value using the `Int` initializer that takes an instance of `String` as input. Pass in `userInputAge` as the argument for the initializer. What error do you get?



3. `//`: Go back and change the type of `userAge` to `Int?`, and print the value of `userAge`. Why is `userAge`'s value `nil`? Provide your answer in a comment or print statement below.



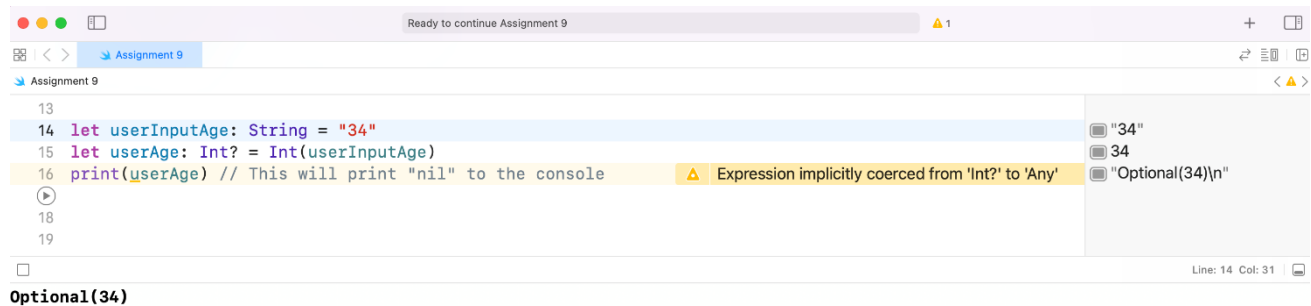
```
7
8 let userInputAge: String = "34e"
9 let userAge: Int? = Int(userInputAge)
10 print(userAge)
```

Expression implicitly coerced from 'Int?' to 'Any'

Line: 10 Col: 14

nil

4. Now go back and fix the typo on the value of `userInputAge`. Is there anything about the value printed that seems off?



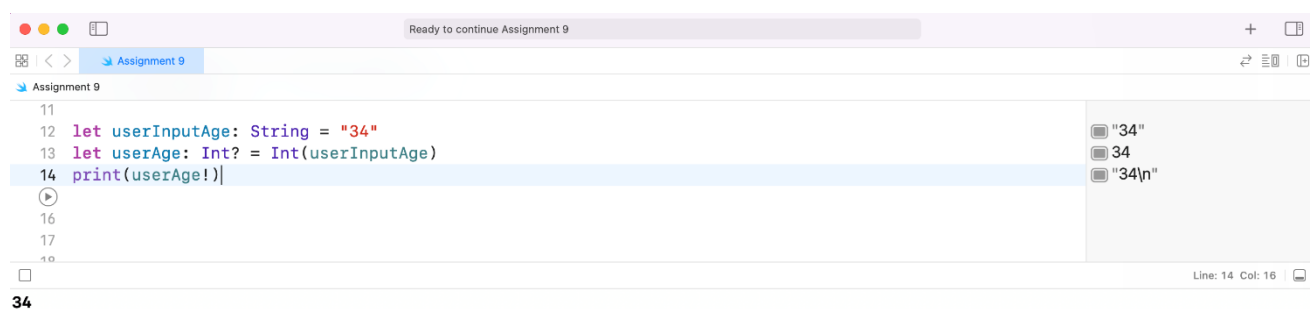
```
13
14 let userInputAge: String = "34"
15 let userAge: Int? = Int(userInputAge)
16 print(userAge) // This will print "nil" to the console
```

Expression implicitly coerced from 'Int?' to 'Any'

Line: 14 Col: 31

Optional(34)

5. Print `userAge` again, but this time unwrap `userAge` using the force unwrap operator.

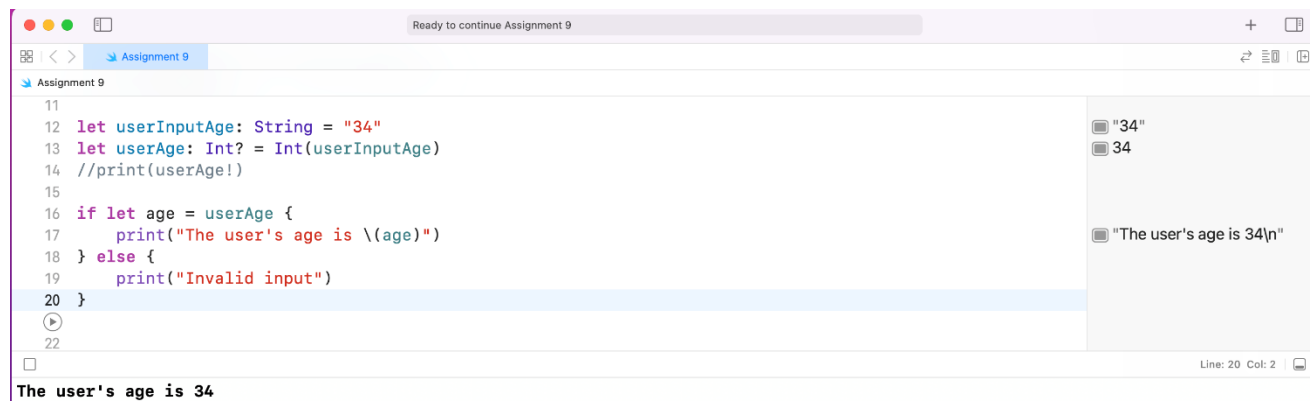


```
11
12 let userInputAge: String = "34"
13 let userAge: Int? = Int(userInputAge)
14 print(userAge!)
```

Line: 14 Col: 16

34

6. `//`: Now use optional binding to unwrap `userAge`. If `userAge` has a value, print it to the console.



```
11
12 let userInputAge: String = "34"
13 let userAge: Int? = Int(userInputAge)
14 //print(userAge!)
15
16 if let age = userAge {
17     print("The user's age is \(age)")
18 } else {
19     print("Invalid input")
20 }
21
22
```

The user's age is 34

7. Many APIs that give you information gathered by the hardware return optionals. For example, an API for working with a heart rate monitor may give you `nil` if the heart rate monitor is adjusted poorly and cannot properly read the user's heart rate. Declare a variable `heartRate` of type `Int?` and set it to `nil`. Print the value.

8. `//`: In this example, if the user fixes the positioning of the heart rate monitor, the app may get a proper heart rate reading. Below, update the value of `heartRate` to 74. Print the value.

9. `//`: As you've done in other app exercises, create a variable `hrAverage` of type `Int` and use the values stored below and the value of `heartRate` to calculate an average heart rate.

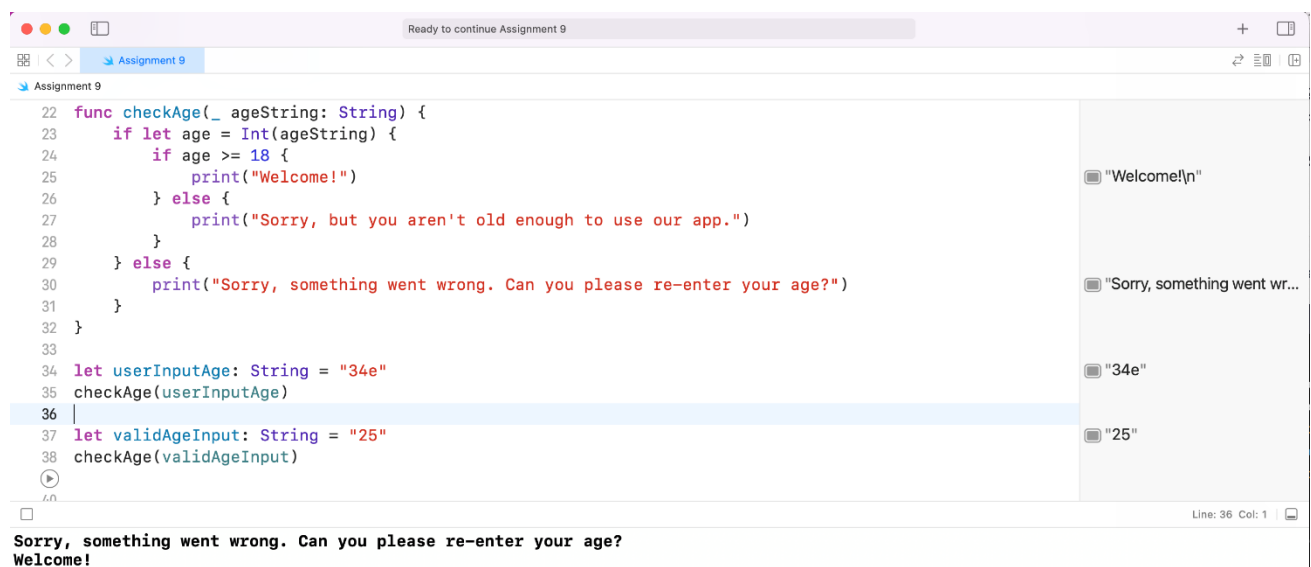
```
let oldHR1 = 80
let oldHR2 = 76
let oldHR3 = 79
let oldHR4 = 70
```

10. If you didn't unwrap the value of `heartRate`, you've probably noticed that you cannot perform mathematical operations on an optional value. You will first need to unwrap `heartRate`.

11. Safely unwrap the value of `heartRate` using optional binding. If it has a value, calculate the average heart rate using that value and the older heart rates stored above. If it doesn't have a value, calculate the average heart rate using only the older heart rates. In each case, print the value of `hrAverage`.

12. If an app asks for a user's age, it may be because the app requires a user to be over a certain age to use some of the services it provides. Write a function called `checkAge` that takes one parameter of type `String`. The function should try to convert this parameter into an `Int` value and then check if the user is over 18 years old. If he/she is old enough, print "Welcome!", otherwise print "Sorry, but you aren't old enough to use our app." If the `String` parameter cannot be converted into an `Int` value, print "Sorry, something went wrong. Can you please re-enter your age?" Call the function and pass in `userInputAge` below as the single parameter. Then call the function and pass in a string that can be converted to an integer.

let userInputAge: String = "34e"



```
22 func checkAge(_ ageString: String) {
23     if let age = Int(ageString) {
24         if age >= 18 {
25             print("Welcome!")
26         } else {
27             print("Sorry, but you aren't old enough to use our app.")
28         }
29     } else {
30         print("Sorry, something went wrong. Can you please re-enter your age?")
31     }
32 }
33
34 let userInputAge: String = "34e"
35 checkAge(userInputAge)
36
37 let validAgeInput: String = "25"
38 checkAge(validAgeInput)
```

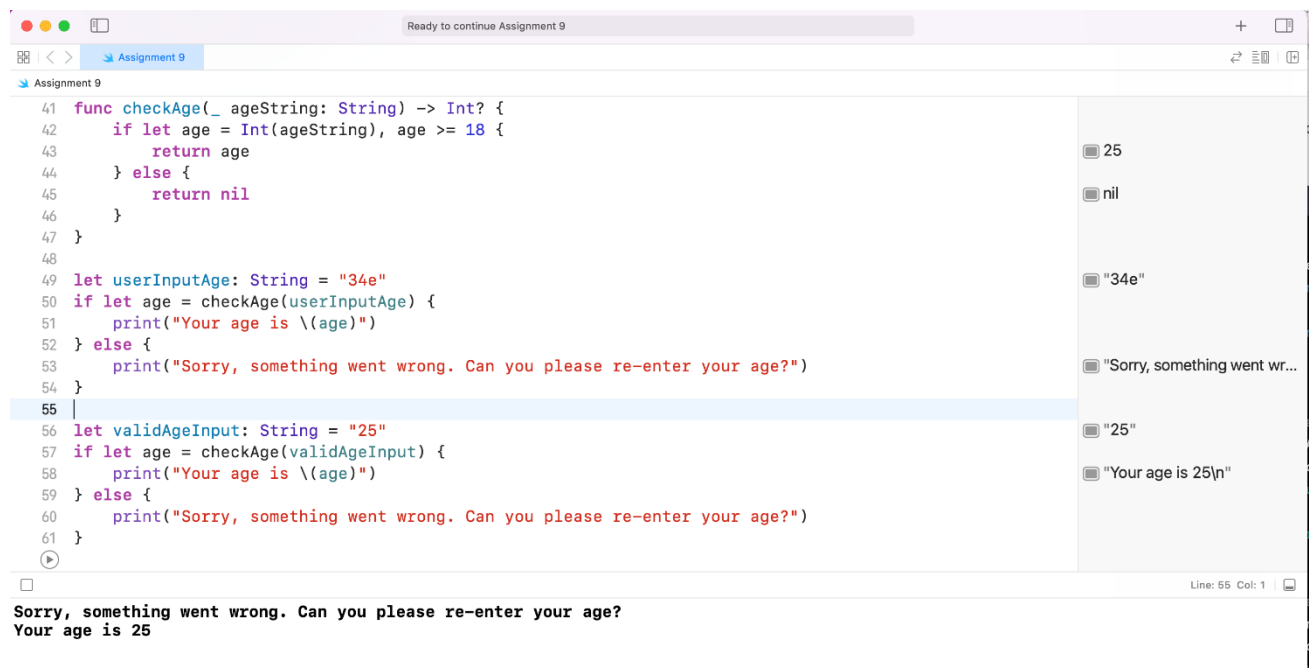
Output:

- "Welcome!\n"
- "Sorry, something went wr..."
- "34e"
- "25"

Line: 36 Col: 1

Sorry, something went wrong. Can you please re-enter your age?  
Welcome!

13. //: Go back and update your function to return the age as an integer. Will your function always return a value? Make sure your return type accurately reflects this. Call the function and print the return value.



```
41 func checkAge(_ ageString: String) -> Int? {
42     if let age = Int(ageString), age >= 18 {
43         return age
44     } else {
45         return nil
46     }
47 }
48
49 let userInputAge: String = "34e"
50 if let age = checkAge(userInputAge) {
51     print("Your age is \(age)")
52 } else {
53     print("Sorry, something went wrong. Can you please re-enter your age?")
54 }
55
56 let validAgeInput: String = "25"
57 if let age = checkAge(validAgeInput) {
58     print("Your age is \(age)")
59 } else {
60     print("Sorry, something went wrong. Can you please re-enter your age?")
61 }
```

25

nil

"34e"

"Sorry, something went wr..."

"25"

"Your age is 25\n"

Line: 55 Col: 1

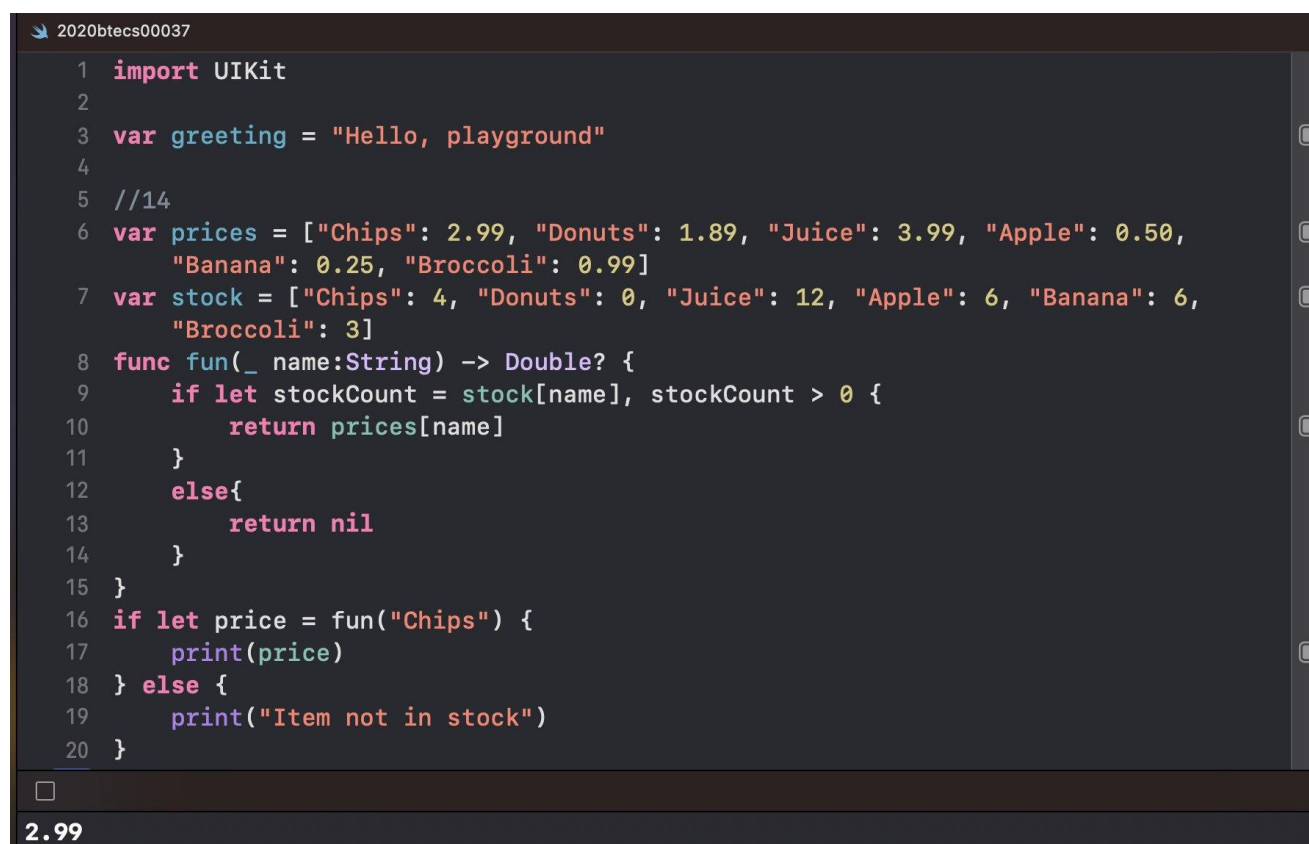
Sorry, something went wrong. Can you please re-enter your age?

Your age is 25

14. //: Imagine you are creating an app for making purchases. Write a function that will take in the name of an item for purchase as a `String` and will return the cost of that item as an optional `Double`. In the body of the function, check to see if the item is in stock by accessing it in the dictionary `stock`. If it is, return the price of the item by accessing it in the dictionary `prices`. If the item is out of stock, return `nil`. Call the function and pass in a `String` that exists in the dictionaries below. Print the return value.

```
var prices = ["Chips": 2.99, "Donuts": 1.89, "Juice": 3.99, "Apple": 0.50, "Banana": 0.25, "Broccoli": 0.99]
```

```
var stock = ["Chips": 4, "Donuts": 0, "Juice": 12, "Apple": 6, "Banana": 6, "Broccoli": 3]
```



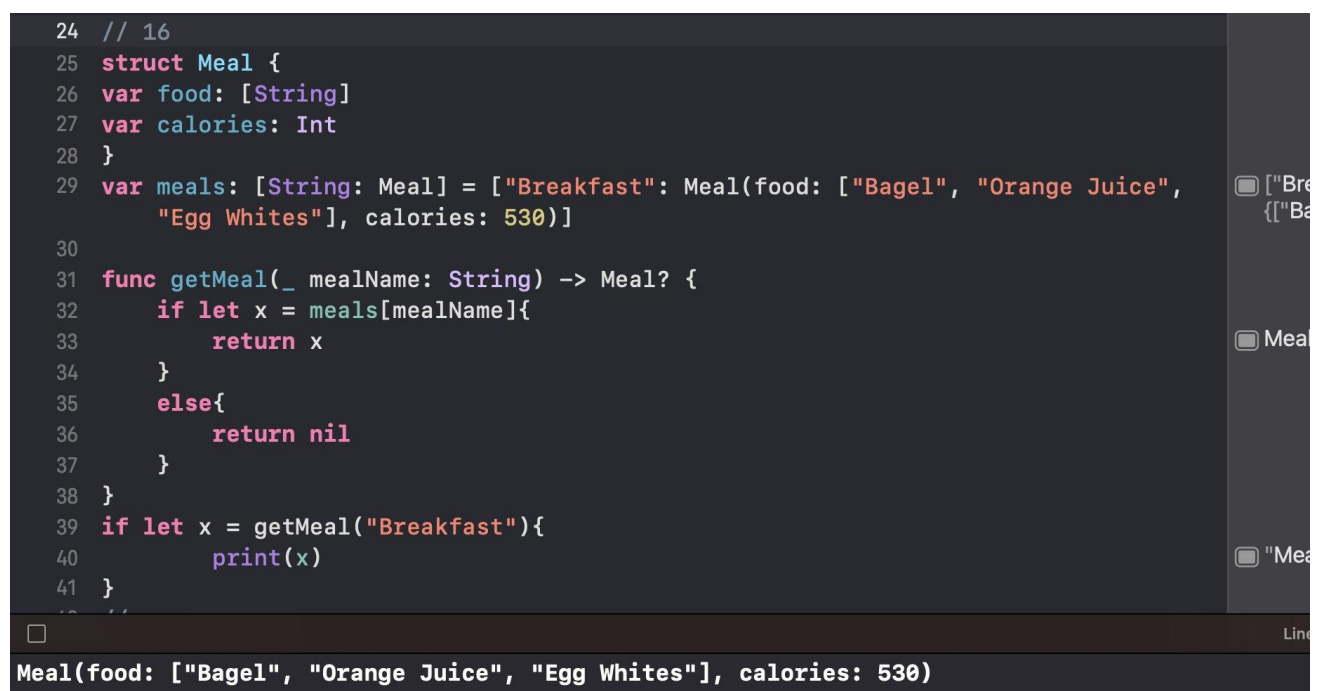
```
2020btcs00037
1 import UIKit
2
3 var greeting = "Hello, playground"
4
5 //14
6 var prices = ["Chips": 2.99, "Donuts": 1.89, "Juice": 3.99, "Apple": 0.50,
7               "Banana": 0.25, "Broccoli": 0.99]
8 var stock = ["Chips": 4, "Donuts": 0, "Juice": 12, "Apple": 6, "Banana": 6,
9               "Broccoli": 3]
10 func fun(_ name:String) -> Double? {
11     if let stockCount = stock[name], stockCount > 0 {
12         return prices[name]
13     }
14     else{
15         return nil
16     }
17 }
18 if let price = fun("Chips") {
19     print(price)
20 } else {
21     print("Item not in stock")
22 }
```

2.99

15. These exercises reinforce Swift concepts in the context of a fitness tracking app. Suppose you want your fitness tracking app to give users the ability to log food. Once food has been logged, users should be able to go back and see what they ate at a specific meal.

16. Write a function that takes a `String` parameter where you will pass in either "Breakfast," "Lunch," or "Dinner." The function should then return the `Meal` object associated with that meal, or return `nil` if the user hasn't logged that meal yet. Note that a `Meal` object and a dictionary, `meals`, representing the meal log have been created for you below. Call the function one or twice and print the return value.

```
struct Meal {  
  var food: [String]  
  var calories: Int  
}  
  
var meals: [String: Meal] = ["Breakfast": Meal(food: ["Bagel", "Orange Juice", "Egg Whites"], calories: 530)]
```



```
24 // 16  
25 struct Meal {  
26   var food: [String]  
27   var calories: Int  
28 }  
29 var meals: [String: Meal] = ["Breakfast": Meal(food: ["Bagel", "Orange Juice",  
   "Egg Whites"], calories: 530)]  
30  
31 func getMeal(_ mealName: String) -> Meal? {  
32   if let x = meals[mealName]{  
33     return x  
34   }  
35   else{  
36     return nil  
37   }  
38 }  
39 if let x = getMeal("Breakfast"){  
40   print(x)  
41 }
```

Meal(food: ["Bagel", "Orange Juice", "Egg Whites"], calories: 530)

17. iOS comes with a few different APIs for persistence, or saving data. You'll learn more about persistence in another lesson, but for now imagine what an app experience would be like if every time you opened the app all of your data was gone. That would be frustrating, right?

18. Write a function that will check to see if your meal log (a dictionary like that in the previous exercise) is saved to the device. If it is, return the meal log. If it isn't, return an empty dictionary of type `[String: Any]`. The code you should use in this exercise for retrieving something saved to the device is `UserDefaults.standard.dictionary(forKey: "mealLog")`. This code will return an optional `[String: Any]`. If it returns a value, that is your meal log. If it returns `nil`, then no meal log has been saved. Call the function and print the return value.

19. Create a `Computer` struct with two properties, `ram` and `yearManufactured`, where both parameters are of type `Int`. Create a failable initializer that will only create an instance of `Computer` if `ram` is greater than 0, and if `yearManufactured` is greater than 1970, and less than 2020.

```
46 ///19
47 struct Computer
48 {
49     var ram:Int
50     var yearManufactured:Int
51     init?(ram: Int, yearManufactured: Int)
52     {
53         if ram > 0 && yearManufactured > 1970 && yearManufactured < 2020
54         {
55             self.ram = ram
56             self.yearManufactured = yearManufactured
57         }
58         else
59         {
60             return nil
61         }
62     }
63 }
```



20. `//`: Create two instances of `Computer?` using the failable initializer. One instance should use values that will have a value within the optional, and the other should result in `nil`. Use `if-let` syntax to unwrap each of the `Computer?` objects and print the `ram` and `yearManufactured` if the optional contains a value.

```
65 // 20
66 struct Computer
67 {
68     var ram: Int
69     var yearManufactured: Int
70     init?(_ ram: Int, _ yearManufactured: Int)
71     {
72         if ram > 0 && yearManufactured > 1970 && yearManufactured < 2020
73         {
74             self.ram = ram
75             self.yearManufactured = yearManufactured
76         }
77         else
78         {
79             return nil
80         }
81     }
82 }
83
84 let firstComputer = Computer(8, 2015)
85 let secondComputer = Computer(0, 1995)
86
87 if let computer = firstComputer {
88     print("RAM: \(computer.ram), Year Manufactured: \(computer.yearManufactured)")
89 }
90
91 if let computer = secondComputer {
92     print("RAM: \(computer.ram), Year Manufactured: \(computer.yearManufactured)")
93 } else {
94     print("Invalid computer")
95 }
96 //
```

RAM: 8, Year Manufactured: 2015  
Invalid computer

21. These exercises reinforce Swift concepts in the context of a fitness tracking app. Have you ever accidentally tapped a button in an app? It's fairly common. In your fitness tracking app, you decide that if a user accidentally taps a button to start a workout and then ends the workout within 10 seconds, you simply don't want to create and log the workout. Otherwise the user would have to go delete the workout from the log.

22. Create a `Workout` struct that has properties `startTime` and `endTime` of type `Double`. Dates are difficult to work with, so you'll be using doubles to represent the number of seconds since midnight, i.e. 28800 would represent 28,800 seconds, which is exactly 8 hours, so the start time is 8am.

```
99  /// 22
100 struct Workout {
101     var startTime: Double
102     var endTime: Double
103 }
104
```

23. Write a failable initializer that takes parameters for your start and end times, and then checks to see if they are fewer than 10 seconds apart. If they are, your initializer should fail. Otherwise, they should set the properties accordingly.

```
105 /// 23
106 struct Workout {
107     var startTime: Double
108     var endTime: Double
109
110     init?(startTime: Double, endTime: Double)
111     {
112         if endTime - startTime < 10
113         {
114             return nil
115         }
116         self.startTime = startTime
117         self.endTime = endTime
118     }
119 }
```

24. //: Try to initialize two instances of a `Workout` object. Unwrap each of them and print its properties. One of them should not be initialized because the start and end times are too close together. The other should successfully initialize a `Workout` object.

```
121 ///24
122 struct Workout {
123     var startTime: Double
124     var endTime: Double
125
126     init?(startTime: Double, endTime: Double)
127     {
128         if endTime - startTime < 10
129         {
130             return nil
131         }
132         self.startTime = startTime
133         self.endTime = endTime
134     }
135 }
136 if let workout1 = Workout(startTime: 3600, endTime: 3605) {
137     print("Workout 1 start time: \(workout1.startTime), end time: \(workout1.endTime)")
138 } else {
139     print("Workout 1 failed to initialize")
140 }
141
142 if let workout2 = Workout(startTime: 3600, endTime: 3660) {
143     print("Workout 2 start time: \(workout2.startTime), end time: \(workout2.endTime)")
144 } else {
145     print("Workout 2 failed to initialize")
146 }
```

```
□
Workout 1 failed to initialize
Workout 2 start time: 3600.0, end time: 3660.0
```