TY CSE AY-2023-24 Sem-II

iOS Lab (6CS381)

Assignment No 10

Due date- 02/04/2024

PART A (Applicable for all batches)

Exercise - Optionals

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.

1. 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.

4. Now go back and fix the typo on the value of `userInputAge`. Is there anything about the value printed that seems off? Print `userAge` again, but this time unwrap `userAge` using the force unwrap operator.

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

   ## App Exercise - Finding a Heart Rate

6. 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.

7. 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.

8. 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
```

9. 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`. 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`.

Type Casting and Inspection

10. Create a collection of type [Any], including a few doubles, integers, strings, and booleans within the collection. Print the contents of the collection.

11. Loop through the collection. For each integer, print "The integer has a value of ", followed by the integer value. Repeat the steps for doubles, strings and booleans.

12. Create a [String : Any] dictionary, where the values are a mixture of doubles, integers, strings, and booleans. Print the key/value pairs within the collection

13. Create a variable `total` of type `Double` set to 0. Then loop through the dictionary, and add the value of each integer and double to your variable's value. For each string value, add 1 to the total. For each boolean, add 2 to the total if the boolean is `true`, or subtract 3 if it's `false`. Print the value of `total`.

14. Create a variable `total2` of type `Double` set to 0. Loop through the collection again, adding up all the integers and doubles. For each string that you come across during the loop, attempt to convert the string into a number, and add that value to the total. Ignore booleans. Print the total.

App Exercise - Workout Types

15. You fitness tracking app may allow users to track different kinds of workouts. When architecting the app, you may decide to have a `Workout` base class from which other types of workout classes inherit. Below are three classes. `Workout` is the base class with `time` and `distance` properties, and `Run` and `Swim` are subclasses that add more specific properties to the `Workout` class. Also provided is a `workouts` array that represents a log of past workouts. You'll use these classes and the array for the exercises below.

```
class Workout {
let time: Double
let distance: Double
init(time: Double, distance: Double) {
self.time = time
```

```
    self.distance = distance
    }
}

class Run: Workout {
   let cadence: Double

   init(cadence: Double, time: Double, distance: Double) {
      self.cadence = cadence
      super.init(time: time, distance: distance)
   }
}

class Swim: Workout {
   let stroke: String

   init(stroke: String, time: Double, distance: Double) {
      self.stroke = stroke
      super.init(time: time, distance: distance)
   }
}

var workouts: [Workout] = [
   Run(cadence: 80, time: 1200, distance: 4000),
   Swim(stroke: "Freestyle", time: 32.1, distance: 50),
   Swim(stroke: "Butterfly", time: 36.8, distance: 50),
   Swim(stroke: "Freestyle", time: 523.6, distance: 500),
   Run(cadence: 90, time: 358.9, distance: 1600)
    ]
```

16. Write simple functions called `describeRun(runningWorkout:)` and `describeSwim(swimmingWorkout:)` that take a `Run` object and a `Swim` object, respectively. Neither should return values. Each function should print a description of the workout, including the run's cadence or the swim's stroke. Time is represented in seconds, distance is represented in meters, and cadence is represented in steps per minute.

17. Now loop through each workout in `workouts` and, using type casting, call either `describeRun(runningWorkout:)` or `describeSwim(swimmingWorkout:)` on each. Observe what is printed to the console.

PART B (Applicable for T1, T2 and T3 batches)

1. Study of Auto layout and stack views.

   Create a calculator using auto layout and stack view. Calculator should perform below tasks.
   a. Addition
   b. Subtraction
   c. Multiplication
   d. Division
   e. Modulus
   f. Square, Cube
   g. Square root

   Note: Refer book page no. 344 for help.

PART C (Applicable for T4, T5 and T6 batches)

1. Study of Auto layout and stack views.
   Create an Apple Pie- word-guessing game using auto layout and stack view.

   Note: Refer book page no. 356 for help.

****