

## Department of Computer Science and Engineering

### Sub: iOS application development using swift

### Assignment No 4 : Optionals and Enumerations

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

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?

Ans., Yes, it's possible for the user to make a mistake when entering their age, resulting in input that cannot be directly converted to an Int. For example, they might:

1. Enter a non-numeric value (e.g., "twenty" or "abc").
2. Enter a floating-point number (e.g., "18.5").
3. Leave the input empty ("").

To handle such cases safely, you can use **optional binding** with `Int(inputText)`, which attempts to convert the String to an Int. If the conversion fails, it returns nil, preventing runtime crashes.

```
main.swift
1 let userInput = "25"
2 if let age = Int(userInput) {
3     print("User's age is \(age).")
4 } else {
5     print("Invalid input. Please enter a valid integer.")
6 }
```

Output

User's age is 25.

=== Code Execution Successful ===

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?

```
main.swift
1 let userInputAge: String = "34e"
2 let userAge: Int = Int(userInputAge)
3 print(userAge)
```

```
Output
ERROR!
/tmp/6KsVNaq1Cw/main.swift:2:20: error: value of optional type 'Int?' must be
  unwrapped to a value of type 'Int'
let userAge: Int = Int(userInputAge)
                    ^

/tmp/6KsVNaq1Cw/main.swift:2:20: note: coerce using '!' to provide a default when
  the optional value contains 'nil'
let userAge: Int = Int(userInputAge)
                    ^
                        ?? <#default value#>

/tmp/6KsVNaq1Cw/main.swift:2:20: note: force-unwrap using '!' to abort execution if
  the optional value contains 'nil'
let userAge: Int = Int(userInputAge)
                    ^
                        !

=== Code Exited With Errors ===
```

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.

```
main.swift
1 let userInputAge: String = "34e"
2 let userAge: Int? = Int(userInputAge)
3
4 print(userAge)
5
```

Output

Clear

```
/tmp/0uVwX7LGf6/main.swift:4:7: warning: expression implicitly coerced from 'Int?' to 'Any'
print(userAge)
  ^~~~~~

/tmp/0uVwX7LGf6/main.swift:4:7: note: provide a default value to avoid this warning
print(userAge)
  ^~~~~~
      ?? <#default value#>
/tmp/0uVwX7LGf6/main.swift:4:7: note: force-unwrap the value to avoid this warning
print(userAge)
  ^~~~~~
      !
/tmp/0uVwX7LGf6/main.swift:4:7: note: explicitly cast to 'Any' with 'as Any' to silence this warning
print(userAge)
  ^~~~~~
      as Any
nil

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

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.

main.swift

Share

Run

```
1 let userInputAge: String = "34"
2 let userAge: Int? = Int(userInputAge)
3 print("The value of userAge is \(userAge!)")
4
```

Output




Clear

```
The value of userAge is 34

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

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

main.swift

 Share 

```
1 let userInputAge: String = "34"
2 let userAge: Int? = Int(userInputAge)
3
4 if let age = userAge {
5     print("The value of userAge is \(age)")
6 } else {
7     print("userAge is nil")
8 }
```

Output 





The value of userAge is 34

=== Code Execution Successful ===

## ## 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.

main.swift

 Share 

```
1 var heartRate: Int? = nil
2 print(heartRate)
```

nil

=== Code Execution Successful ===

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.

```
main.swift
1 var heartRate: Int? = nil
2 print(heartRate)
3 heartRate = 74
4 print(heartRate)
```

```
nil
Optional(74)
```

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

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
```

```
main.swift
1 var heartRate: Int? = 74
2 let oldHR1 = 80
3 let oldHR2 = 76
4 let oldHR3 = 79
5 let oldHR4 = 70
6 if let currentHeartRate = heartRate {
7     let hrAverage: Int = (currentHeartRate + oldHR1 + oldHR2 + oldHR3 + oldHR4) /
8         5
9     print("Average heart rate: \(hrAverage)")
10 } else {
11     print("Heart rate data is not available.")
12 }
```

Output




Clear

```
Average heart rate: 75
```

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

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`.

main.swift

 Share 

```
1 var heartRate: Int? = nil
2 let oldHR1 = 80
3 let oldHR2 = 76
4 let oldHR3 = 79
5 let oldHR4 = 70
6 if let currentHeartRate = heartRate {
7     let hrAverage: Int = (currentHeartRate + oldHR1 + oldHR2 + oldHR3 + oldHR4) /
        5
8     print("Average heart rate: \(hrAverage)")
9 } else {
10     let hrAverage: Int = (oldHR1 + oldHR2 + oldHR3 + oldHR4) / 4
11     print("Average heart rate: \(hrAverage)")
12 }
```

Output





Clear

Average heart rate: 76

=== Code Execution Successful ===

10. Define a `Suit` enum with four possible cases: `clubs`, `spades`, `diamonds`, and `hearts`.

main.swift

 Share 

```
1 enum Suit {
2     case clubs
3     case spades
4     case diamonds
5     case hearts
6 }
7
```

11. Imagine you are being shown a card trick and have to draw a card and remember the suit. Create a variable instance of `Suit` called `cardInHand` and assign it to the `hearts` case. Print out the instance.

main.swift

Share

Run

```
1 enum Suit {
2     case clubs
3     case spades
4     case diamonds
5     case hearts
6 }
7
8 var cardInHand = Suit.hearts
9 print(cardInHand)
```

Output

Clear

hearts

=== Code Execution Successful ===

12. Now imagine you have to put back the card you drew and draw a different card. Update the variable to be a spade instead of a heart.

main.swift

Share

Run

```
1 enum Suit {
2     case clubs
3     case spades
4     case diamonds
5     case hearts
6 }
7
8 var cardInHand = Suit.hearts
9 print(cardInHand)
10 cardInHand = Suit.spades
11 print(cardInHand)
```

Output

Clear

hearts
spades

=== Code Execution Successful ===

13. Imagine you are writing an app that will display a fun fortune (i.e. something like "You will soon find what you seek.") based on cards drawn. Write a function called `getFortune(cardSuit:)` that takes a parameter of type `Suit`. Inside the body of the function, write a switch statement based on the value of `cardSuit`. Print a different fortune for each `Suit` value. Call the function a few times, passing in different values for `cardSuit` each time.

```
main.swift
1- enum Suit {
2     case clubs
3     case spades
4     case diamonds
5     case hearts
6 }
7
8- func getFortune(cardSuit: Suit) {
9     switch cardSuit {
10    case .hearts:
11        print("You will soon find what you seek.")
12    case .diamonds:
13        print("Wealth is on the way to you!")
14    case .clubs:
15        print("An unexpected opportunity will come your way.")
16    case .spades:
17        print("Be prepared for a challenge ahead.")
18    }
19 }
20
21
22 getFortune(cardSuit: .clubs)
23 getFortune(cardSuit: .spades)
24 getFortune(cardSuit: .diamonds)
25 getFortune(cardSuit: .hearts)
```

#### Output

Clear

```
An unexpected opportunity will come your way.
Be prepared for a challenge ahead.
Wealth is on the way to you!
You will soon find what you seek.
```

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



14. Create a `Card` struct below. It should have two properties, one for `suit` of type `Suit` and another for `value` of type `Int`.

```
main.swift
1 enum Suit {
2     case clubs
3     case spades
4     case diamonds
5     case hearts
6 }
7
8 struct Card {
9     var suit: Suit
10    var value: Int
11 }
```

15. How many values can playing cards have? How many values can `Int` be? It would be safer to have an enum for the card's value as well. Inside the struct above, create an enum for `Value`. It should have cases for `ace`, `two`, `three`, `four`, `five`, `six`, `seven`, `eight`, `nine`, `ten`, `jack`, `queen`, `king`. Change the type of `value` from `Int` to `Value`. Initialize two `Card` objects and print a statement for each that details the card's value and suit.

```
main.swift
1 enum Suit {
2     case clubs
3     case spades
4     case diamonds
5     case hearts
6 }
7
8 enum Value: Int {
9     case two, three, four, five, six, seven, eight, nine, ten, ace, jack, queen,
        king
10 }
11
12 struct Card {
13     var suit: Suit
14     var value: Value
15 }
16
17 let card1 = Card(suit: .hearts, value: .ace)
18 let card2 = Card(suit: .clubs, value: .king)
19 print("Card 1 is a \(card1.value) of \(card1.suit)")
20 print("Card 2 is a \(card2.value) of \(card2.suit)")
21
```

Output

Clear

Card 1 is a ace of hearts  
Card 2 is a king of clubs

=== Code Execution Successful ===

\*\*\*\*\*