**Mobile Application Development-II**

**DEVELOPED BY - DHAIRYA JAIN**

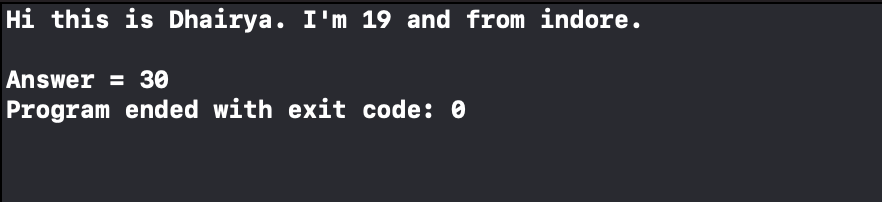
**INDEX**

|  |  |  |
| --- | --- | --- |
| **NUMBER** | **DATE** | **TITLE** |
| **1** | 19-04-2021 | Programs to demonstrate function with and without return type and parameters |
| **2** | 26-04-2021 | Program to demonstrate function returning multiple values. |
| **3** | 28-04-2021 | Program to demonstrate function returning optional tuple. |
| **4** | 03-05-2021 | Programs to demonstrate function with and without arugument label. |
| **5** | 05-05-2021 | Program to demonstrate Closuers. |
| **6** | 10-05-2021 | Program to demonstrate Single-Expression Closures. |
| **7** | 12-05-2021 | Program to demonstrate Shorthand Argument Names. |
| **8** | 17-05-2021 | Program to demonstrate Trailing Closures. |
| **9** | 19-05-2021 | Program to demonstrate ENUMERATION. |
| **10** | 24-05-2021 | Program to demonstrate with Switch case. |
| **11** | 02-06-2021 | Program to demonstrate Enumeration Associated values,Raw Values. |
| **12** | 09-06-2021 | Program to demonstrate Structure . |
| **13** | 14-06-2021 | Program to demonstrate Properties, Memberwise and Initializers for Structure Types. |
| **14** | 16-06-2021 | Programs to demonstrate Stored Properties, Lazy Stored Properties, Computed Properties, and Property Observers. |
| **15** | 21-06-2021 | Programs to demonstrate different types of Inheritance in Swift. |
| **16** | 23-06-2021 | Programs to demonstrate Methods, Instance Methods, self Property and Mutating Method |
| **17** | 28-06-2021 | Programs to demonstrate Accessing Superclass Methods, Properties, Overriding  Methods and Overriding Properties. |
| **18** | 30-06-2021 | Programs to demonstrate Initializers, Default Property Values and Custom Initializers. |
| **19** | 05-07-2021 | Programs to demonstrate Initializer Inheritance , Overriding and Automatic Initializer Inheritance. |
| **20** | 07-07-2021 | Programs to demonstrate Failable Initializers, Failable Initializers for Enumerations and  Overriding a Failable Initializer. |

**1.Programs to demonstrate function with and without return type and parameters.**

1. ////func without retrurn & parameter
2. import foundation
3. func introduction(name: String, home: String, age: Int)
4. {
5. print("Hi this is \(name). I'm \(age) and from \(home).")
6. }
7. introduction(name: "Dhairya", home: "indore", age: 19)
8. //func with return & parameter
9. func addTwoInts(\_ a: Int,\_ b: Int) -> Int
10. {
11. return a+b
12. }
13. print("\nAnswer = \(addTwoInts(20, 10))")

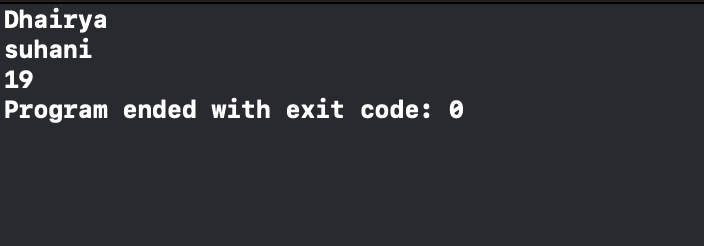
**OUTPUT**

****

**2.Program to demonstrate function returning multiple values.**

1. import Foundation
2. func dosomeThing()->(String , String,Int)
3. {
4. return ("Dhairya" , "suhani", 19)
5. }
6. let a = dosomeThing()
7. print(a.0); print(a.1);print(a.2)

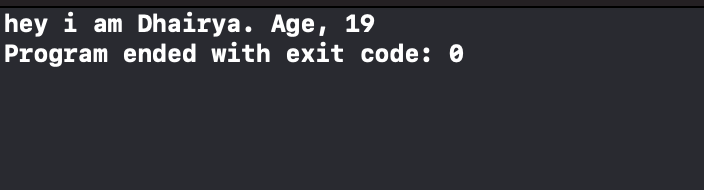
**OUTPUT**



**3.Program to demonstrate function returning optional tuple.**

1. import Foundation
2. let a = 0
3. func returnMultipleValues() -> (name: String, count: Int)?
4. {
5. if a==0 {
6. return ("hey i am Dhairya. Age", 19)
7. }
8. else{
9. return nil
10. }
11. }
12. let result = returnMultipleValues()!
13. print("\(result.name), \(result.count) ")

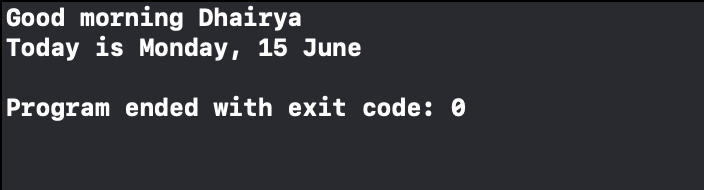
**OUTPUT**

****

**4.Program to demonstrate function with and without arugument label.**

1. import Foundation
2. //Function with Argument label
3. func greet(name: String) -> String{
4. let txt = "Good morning " + name
5. return txt
6. }
7. //Function without Argument label
8. func today(\_ day: String, \_ date: String) -> String
9. {
10. let txt\_d = "Today is " + day + ", " + date
11. return txt\_d
12. }
13. print(greet(name: "Dhairya"))
14. print(today("Monday","15 June \n"))

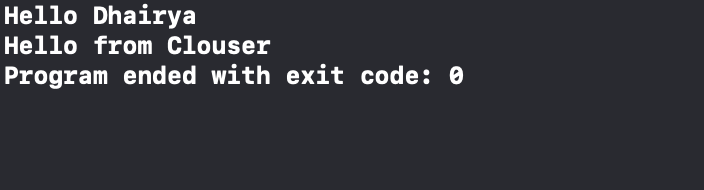
**OUTPUT**



**5.Program to demonstrate Closuers.**

1. import Foundation
2. func greeting (name : String)
3. {
4. print("Hello \(name)")
5. }
6. greeting(name: "Dhairya")
7. var a =
8. {
9. print("Hello from Clouser")
10. }
11. a()

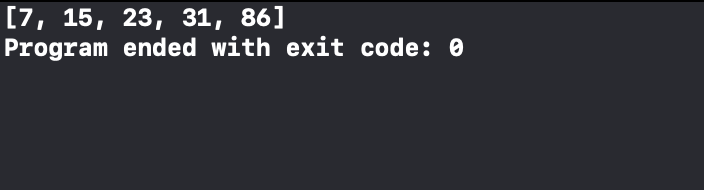
**OUTPUT**



**6. Program to demonstrate Single-Expression Closures.**

1. import Foundation
2. //single expression closure
3. var list = [15,31,7,23,86]
4. var assendinglist = list.sorted(by: {x, y in x < y} )
5. print(assendinglist)

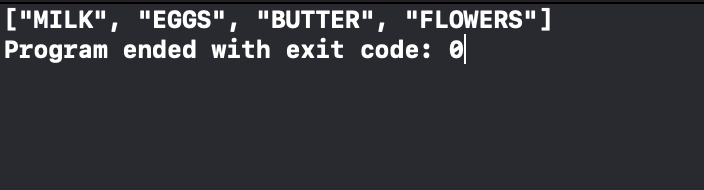
**OUTPUT**



**7.** **Program to demonstrate Shorthand Argument Names.**

1. import Foundation
2. //Shorthand Argument Names
3. var groceries = ["milk", "eggs", "butter", "flowers"]
4. var uppercasedGroceries = groceries.map
5. {
6. $0.uppercased()
7. }
8. print(uppercasedGroceries)

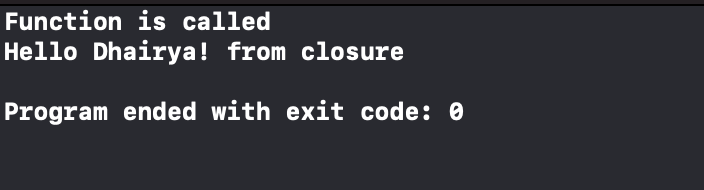
**OUTPUT**



**8.** **Program to demonstrate Trailing Closures.**

1. import Foundation
2. func someSimpleFunction(msg:String ,someClosure:()->())
3. {
4. print(msg)
5. someClosure()
6. }
7. someSimpleFunction(msg:"Function is called ")
8. {
9. print("Hello Dhairya! from closure\n")
10. }

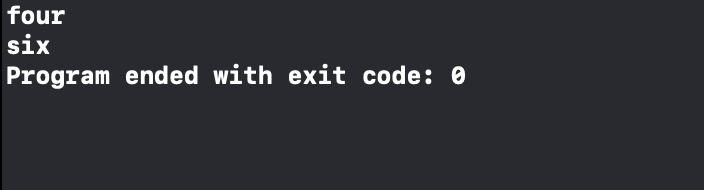
**OUTPUT**



**9.Program Program to demonstrate Enumeration.**

1. import Foundation
2. enum Dice
3. {
4. case one,two,three,four,five ,six
5. }
6. var diceScore = Dice.four
7. print(diceScore)
8. var diceScore1 :Dice = .six
9. print(diceScore1)

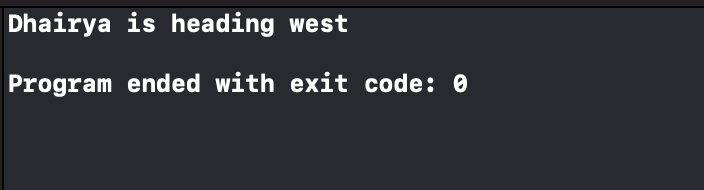
**OUTPUT**

****

**10.Program Program to demonstrate Enumeration with Switch case.**

1. import Foundation
2. enum CompassPoint
3. {
4. case north, east, south, west
5. }
6. let compassHeading: CompassPoint = .west
7. switch compassHeading
8. {
9. case .north:
10. print("Dhairya is heading north\n")
11. case .east:
12. print("Dhairya is heading east.\n")
13. case .south:
14. print("Dhairya is heading south\n")
15. case .west:
16. print("Dhairya is heading west\n")
17. }

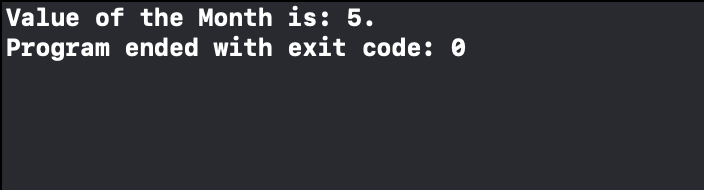
**OUTPUT**



**11.Program to demonstrate Enumeration Associated values, Raw Values.**

1. import Foundation
2. enum Month: Int
3. {
4. case January = 1, February, March, April, May, June, July, August,
5. September, October, November, December
6. }
7. let yearMonth = Month.May.rawValue
8. print("Value of the Month is: \(yearMonth).")

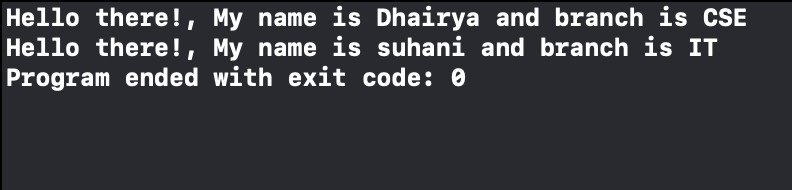
**OUTPUT**

****

**12.Program to demonstrate STRUCTURE.**

1. import Foundation
2. struct Student
3. {
4. var name:String
5. var age : Double
6. var rollno:Int
7. var branch:String
8. func sayHello()
9. {
10. print("Hello there!, My name is \(name) and branch is \(branch)")
11. }
12. }
13. var s1 = Student(name:"Dhairya",age:19,rollno:101,branch:"CSE")
14. s1.sayHello()
15. var s2 = Student(name:"suhani",age:16,rollno:102,branch:"IT")
16. s2.sayHello()

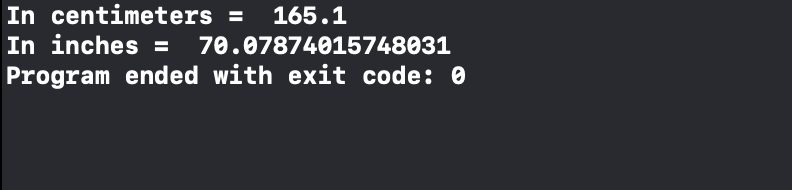
**OUTPUT**

****

**13.Program to demonstrate Properties, Member wise and Initializers for Structure Types.**

1. import Foundation
2. struct Height
3. {
4. var heightInInches: Double
5. var heightInCentimeters: Double
6. init(heightInInches: Double)
7. {
8. self.heightInInches = heightInInches
9. self.heightInCentimeters = heightInInches \* 2.54
10. }
11. init(heightInCentimeters: Double)
12. {
13. self.heightInCentimeters = heightInCentimeters; self.heightInInches = heightInCentimeters / 2.54
14. }
15. }
16. var someonesHeight = Height(heightInInches: 65)
17. print("In centimeters = ",someonesHeight.heightInCentimeters)
18. var myHeight = Height(heightInCentimeters: 178)
19. print("In inches = ",myHeight.heightInInches)

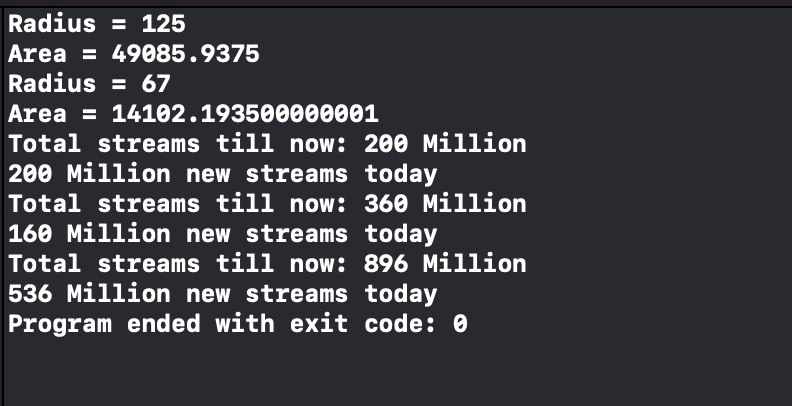
**OUTPUT**



**14.Programs to demonstrate Stored Properties, Lazy Stored Properties, Computed Properties, and Property Observers.**

1. import Foundation
2. class StreamsCounter
3. {
4. var totalStreams: Int = 0
5. {
6. willSet(updatedStreams)
7. {
8. print("Total streams till now: \(updatedStreams) Million")
9. }
10. didSet
11. {
12. if totalStreams > oldValue
13. {
14. print("\(totalStreams - oldValue) Million new streams today")
15. }
16. }
17. }
18. }
19. struct Circle\_Area
20. {
21. var radius: Int; let pi = 3.1415; var area: Double
22. {
23. get
24. {
25. return pi \* Double(radius) \* Double(radius)
26. }
27. }
28. }
29. var n = Circle\_Area(radius: 125)
30. print("""
31. Radius = \(n.radius)
32. Area = \(n.area)
33. """)
34. n.radius = 67
35. print("""
36. Radius = \(n.radius)
37. Area = \(n.area)
38. """)
39. let Heartless\_streams = StreamsCounter()
40. Heartless\_streams.totalStreams = 200
41. Heartless\_streams.totalStreams = 360
42. Heartless\_streams.totalStreams = 896

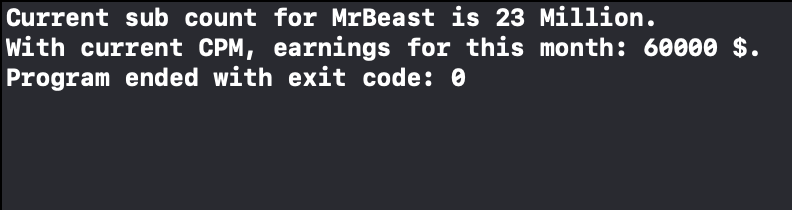
**OUTPUT**

****

**15.Programs to demonstrate different types of Inheritance in Swift.**

1. import Foundation
2. class YT\_Channel
3. {
4. //base class
5. var ch\_name = ""
6. var subscribers = 0
7. var views = 0
8. }
9. class YT\_Ch\_India: YT\_Channel
10. {
11. //derived class
12. let CPM\_India = 5
13. var currentSubs: String
14. {
15. return "Current sub count for \(ch\_name) is \(subscribers) Million."
16. }
17. var earnings: String
18. {
19. return "With current CPM, earnings for this month: \(CPM\_India \* (views/1000)) $."
20. }
21. }
22. var MrBeast = YT\_Ch\_India()
23. MrBeast.ch\_name = "MrBeast"
24. MrBeast.subscribers = 23
25. MrBeast.views = 12000000
26. print(MrBeast.currentSubs)
27. print(MrBeast.earnings)

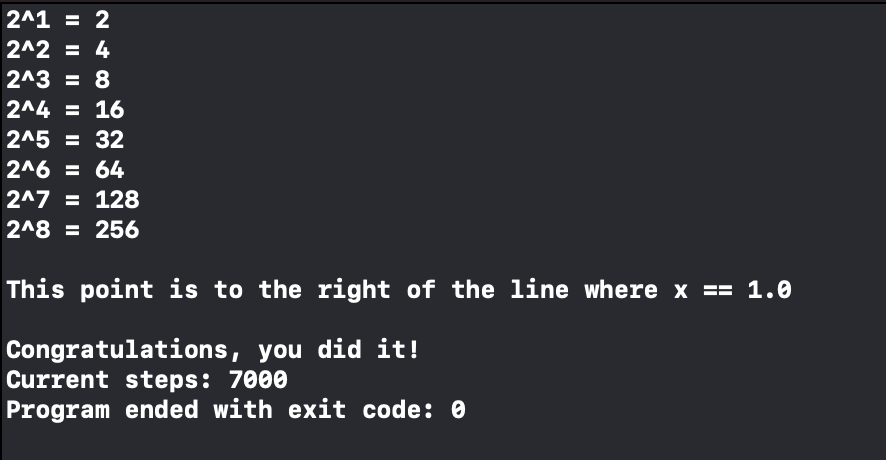
**OUTPUT**



**16.Programs to demonstrate Methods, Instance Methods, self Property and Mutating Method.**

1. import Foundation
2. class returnExpo
3. {
4. var num = 0
5. var upto = 0
6. func ExpoSeries()
7. {
8. for expo in 0...upto
9. {
10. print("\(num)^\(expo+1) = \(num << expo)")
11. }
12. }
13. }
14. var series\_2 = returnExpo()
15. series\_2.num = 2
16. series\_2.upto = 7
17. series\_2.ExpoSeries()
18. struct Point
19. {
20. var x = 0.0, y = 0.0
21. func isToTheRightOf(x: Double) -> Bool
22. {
23. return self.x > x
24. }
25. }
26. let somePoint = Point(x: 4.0, y: 5.0)
27. if somePoint.isToTheRightOf(x: 1.0)
28. {
29. print("\nThis point is to the right of the line where x == 1.0")
30. }
31. struct Steps
32. {
33. var steps: Int
34. {
35. willSet
36. {
37. if newValue == goal
38. {
39. print("\nCongratulations, you did it!")
40. }
41. }
42. }
43. var goal: Int
44. mutating func takeStep()
45. {
46. steps += 1
47. print("Current steps: \(steps)")
48. }
49. }
50. var myStep = Steps(steps: 6999, goal: 7000)
51. myStep.takeStep()

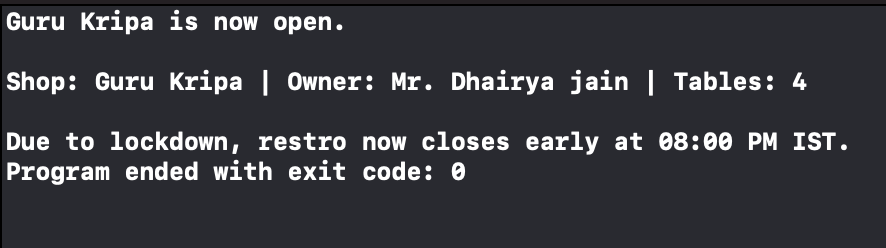
**OUTPUT**

****

**17.Programs to demonstrate Accessing Superclass Methods, Properties, Overriding Methods and Overriding Properties.**

1. import Foundation
2. class Shop
3. {
4. var shop\_name = ""
5. var owner\_name = ""
6. var days\_per\_week = 0
7. var description: String
8. {
9. return "\nShop: \(shop\_name) | Owner: \(owner\_name)"
10. }
11. func open()
12. {
13. print("\(shop\_name) is now open.")
14. }
15. func close()
16. {
17. print("\(shop\_name) is now closed.")
18. }
19. }
20. class Restaurant: Shop
21. {
22. var tables = 0
23. override var description: String
24. {
25. return super.description + " | Tables: \(tables)"
26. //overriding properties and accessing superclass items
27. }
28. override func close()
29. {
30. print("\nDue to lockdown, restro now closes early at 08:00 PM IST.")
31. //overriding methods
32. }
33. }
34. var Restro\_1 = Restaurant()
35. Restro\_1.shop\_name = "Guru Kripa"
36. Restro\_1.owner\_name = "Mr. Dhairya jain"
37. Restro\_1.days\_per\_week = 7
38. Restro\_1.tables = 4
39. Restro\_1.open()
40. print("\(Restro\_1.description)")
41. Restro\_1.close()

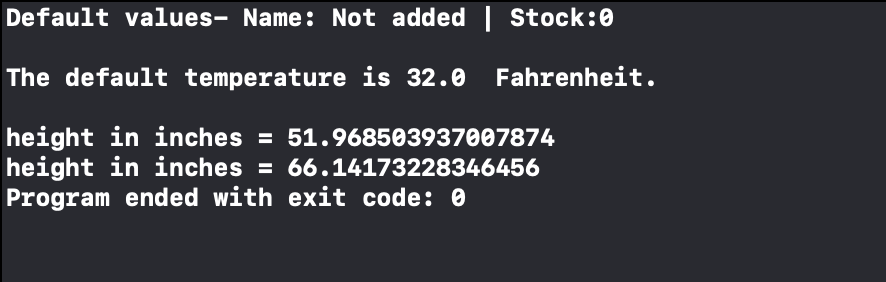
**OUTPUT**

****

**18. Programs to demonstrate Initializers, Default Property Values and Custom Initializers.**

1. import Foundation
2. struct Inventory
3. {
4. var item\_name = "Not added"
5. var item\_stock = 00
6. **//default property values**
7. }
8. var Example\_Inv = Inventory()
9. print("Default values- Name: \(Example\_Inv.item\_name) | Stock:\(Example\_Inv.item\_stock)\n")
10. struct Fahrenheit
11. {
12. var temperature: Double
13. init()
14. {
15. temperature = 32.0
16. **//initializer**
17. }
18. }
19. var f = Fahrenheit()
20. print("The default temperature is \(f.temperature)  Fahrenheit.\n")
21. struct Height
22. {
23. var heightInInches: Double
24. {
25. didSet
26. {
27. if (heightInCentimeters != heightInInches \* 2.54)
28. {
29. heightInCentimeters = heightInInches \* 2.54
30. }
31. }
32. }
33. var heightInCentimeters: Double
34. {
35. didSet
36. {
37. if (heightInInches != heightInCentimeters / 2.54)
38. {
39. heightInInches = heightInCentimeters / 2.54
40. }
41. }
42. }
43. init(heightInInches: Double)
44. {
45. self.heightInInches = heightInInches
46. self.heightInCentimeters = heightInInches\*2.54
47. **//a custom initializer**
48. }
49. init(heightInCentimeters: Double)
50. {
51. self.heightInCentimeters = heightInCentimeters
52. self.heightInInches = heightInCentimeters/2.54
53. **//a custom initializer**
54. }
55. }
56. var Height1 = Height(heightInCentimeters: 132)
57. print("height in inches =",Height1.heightInInches)
58. Height1.heightInCentimeters = 168
59. print("height in inches =",Height1.heightInInches)

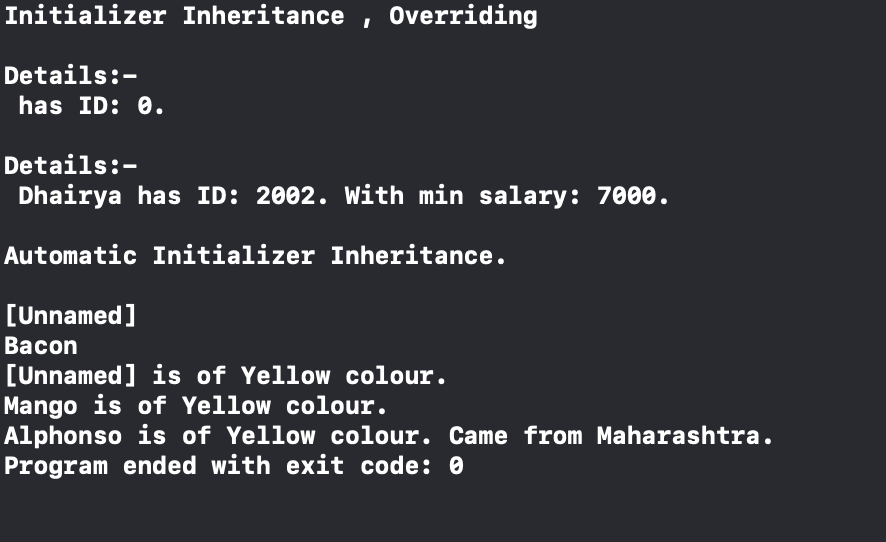
**OUTPUT**

****

**19. Programs to demonstrate Initializer INHERITANCE, Overriding and Automatic Initializer Inheritance.**

1. import Foundation
2. **//Initializer Inheritance , Overriding**
3. print("Initializer Inheritance , Overriding \n")
4. class Employee
5. {
6. var Name = ""
7. var ID = 0
8. var Salary = 0
9. var description: String
10. {
11. return "\(Name) has ID: \(ID)."
12. }
13. }
14. let Emp\_1 = Employee()
15. print("Details:-\n\(Emp\_1.description)")
16. class Accounts: Employee
17. {
18. override init()
19. {
20. super.init()
21. Salary = 7000
22. }
23. override var description: String
24. {
25. return super.description + " With min salary: \(Salary)."
26. }
27. }
28. let Emp\_2 = Accounts()
29. Emp\_2.Name = "Dhairya"
30. Emp\_2.ID = 2002
31. print("\nDetails:-\n",(Emp\_2.description),"\n")
32. **// Automatic Initializer Inheritance**.
33. print("Automatic Initializer Inheritance. \n")
34. class Food
35. {
36. var name: String
37. init(name: String)
38. {
39. self.name = name
40. }
41. convenience init()
42. {
43. self.init(name: "[Unnamed]")
44. }
45. }
46. let mysteryMeat = Food()
47. print(mysteryMeat.name)
48. let namedMeat = Food(name: "Bacon")
49. print(namedMeat.name)
50. class Fruit: Food
51. {
52. var colour: String
53. init(name: String, colour: String)
54. {
55. self.colour = colour
56. super.init(name: name)
57. }
58. override convenience init(name: String)
59. {
60. self.init(name: name, colour: "Yellow")
61. }
62. //automatically inherits Food's initializers
63. }
64. let Mango = Fruit()
65. Mango.colour = "Yellow"
66. print("\(Mango.name) is of \(Mango.colour) colour.")
67. Mango.name = "Mango"
68. print("\(Mango.name) is of \(Mango.colour) colour.")
69. class Alphonso: Fruit
70. {
71. var origin: String = ""
72. }
73. let newMango = Alphonso()
74. newMango.name = "Alphonso"
75. newMango.origin = "Maharashtra"
76. print("\(newMango.name) is of \(newMango.colour) colour. Came from \(newMango.origin).")

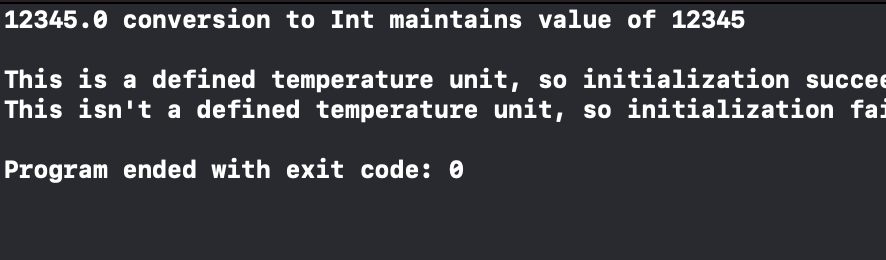
**OUTPUT**



**20. Programs to demonstrate Failable Initializers, Failable Initializers for Enumerations and Overriding a Failable Initializer.**

1. import Foundation
2. **//failable initializer**
3. let wholeNumber: Double = 12345.0
4. if let valueMaintained = Int(exactly: wholeNumber)
5. {
6. print("\(wholeNumber) conversion to Int maintains value of \(valueMaintained)\n")
7. }
8. **//Failable Initializers for Enumerations**
9. enum TemperatureUnit
10. {
11. case kelvin, celsius, fahrenheit
12. init?(symbol: Character)
13. {
14. switch symbol
15. {
16. case "K":
17. self = .kelvin
18. case "C":
19. self = .celsius
20. case "F":
21. self = .fahrenheit
22. default:
23. return nil
24. }
25. }
26. }
27. let fahrenheitUnit = TemperatureUnit(symbol: "F")
28. if fahrenheitUnit != nil
29. {
30. print("This is a defined temperature unit, so initialization succeeded.")
31. }
32. // Prints "This is a defined temperature unit, so initialization succeeded."
33. let unknownUnit = TemperatureUnit(symbol: "X")
34. if unknownUnit == nil
35. {
36. print("This isn't a defined temperature unit, so initialization failed.\n")
37. }

**OUTPUT**

****

**//You can override Failable Initializer by Non-Failable in sub-class.**

1. import Foundation
2. class Account: CustomStringConvertible
3. {
4. let name: String
5. let pass: String
6. init()
7. {
8. name = "Garima"; pass = "Jain"
9. }
10. init?(name:String, pass:String)
11. {
12. self.name = name
13. self.pass = pass
14. if name.isEmpty || pass.isEmpty
15. {
16. return nil
17. }
18. }
19. var description: String
20. {
21. return "name:\(name) pass:\(pass)"
22. }
23. }
24. class Guest: Account
25. {
26. override init(name: String, pass: String)
27. {
28. if name.isEmpty || pass.isEmpty
29. {
30. super.init()
31. } else
32. {
33. // Can't fail certainly, unwrap implicity.
34. super.init(name: name, pass: pass)!
35. }
36. }
37. }
38. // Failable Initializer
39. let account = Account(name: "", pass: "")
40. print(account?.description) // nil
41. // Non-Failable Initializer
42. let guest = Guest(name: "Dhairya", pass: "Jain")
43. print(guest.description)

**OUTPUT**

****