Background pattern

Description automatically generated

Swift Programs Advanced

By Prateek Panwar

Functions, Return types, Tuples, Trailing Closures,

Enumeration, Inheritance

Index

|  |  |  |  |
| --- | --- | --- | --- |
| Practical | Program | Date | Page No. |
| 1 | Programs to demonstrate function with and without return type and parameters | 15 May 2021 | 4 |
| 2 | Program to demonstrate function returning multiple values | 15 May 2021 | 5 |
| 3 | Program to demonstrate function returning optional tuple | 15 May 2021 | 6 |
| 4 | Programs to demonstrate function with and without argument label | 15 May 2021 | 7 |
| 5 | Program to demonstrate Closures | 29 June 2021 | 8 |
| 6 | Program to demonstrate Single-Expression Closures | 29 June 2021 | 9 |
| 7 | Program to demonstrate Shorthand Argument Names | 29 June 2021 | 10 |
| 8 | Program to demonstrate Trailing Closures | 29 June 2021 | 11 |
| 9 | Program to demonstrate Enumeration | 30 June 2021 | 12 |
| 10 | Program to demonstrate with Switch case | 30 June 2021 | 13 |
| 11 | Program to demonstrate Enumeration Associated values, Raw Values | 10 July 2021 | 14 |
| 12 | Program to demonstrate Structure | 10 July 2021 | 16 |
| 13 | Program to demonstrate Properties, Memberwise and Initializers for Structure Types. | 10 July 2021 | 17 |
| 14 | Programs to demonstrate Stored Properties, Lazy Stored Properties, Computed  Properties, and Property Observers. | 10 July 2021 | 18 |
| 15 | Programs to demonstrate different types of Inheritance in Swift. | 13 July 2021 | 20 |
| 16 | Programs to demonstrate Methods, Instance Methods, self-Property and Mutating Method. | 13 July 2021 | 22 |
| 17 | Programs to demonstrate Accessing Superclass Methods, Properties, Overriding  Methods and Overriding Properties. | 13 July 2021 | 24 |
| 18 | Programs to demonstrate Initializers, Default Property Values and Custom Initializers. | 13 July 2021 | 25 |
| 19 | Programs to demonstrate Initializer Inheritance, Overriding and Automatic Initializer Inheritance. | 14 July 2021 | 27 |
| 20 | Programs to demonstrate Failable Initializers, Failable Initializers for Enumerations and  Overriding a Failable Initializer. | 14 July 2021 | 29 |

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

Ans. Program:

import Foundation

//Function with return and parameter

func add(a: Int, b: Int) -> Int {

var c = a + b

return c

}

//Function without retrurn and parameter

func val()

{

print("Value of 10+20 = ")

}

val()

print(add(a: 10, b: 20))

Output:

Text

Description automatically generated with low confidence

1. Program to demonstrate function returning multiple values.

Ans. Program:

import Foundation

//func with multiple return

func rectangle() -> (Int, Int) {

var l = 10, b = 20

var area = l \* b, peri = 2 \* (l + b)

return (area,peri)

}

let r = rectangle()

print("Area: \(r.0), Perimeter: \(r.1)")

Output:

Text

Description automatically generated with low confidence

1. Program to demonstrate function returning optional tuple.

Ans. Program:

import Foundation

//Function returning optional tuple

func student(id: Int) -> (name: String, age: Int)? {

if id == 1 {

return ("Prateek",18) }

else {

print("Enter correct id")

return nil

}

}

let s = student(id: 1)

print("Student Deatils: \n\(s)")

Output:

Text

Description automatically generated

1. Programs to demonstrate function with and without argument label.

Ans. Program:

import Foundation

//Function with Argument label

func greet(name: String) -> String{

var txt = "Good morning " + name

return txt

}

//Function without Argument label

func today(\_ day: String, \_ date: String) -> String{

var txt\_d = "Today is " + day + ", " + date

return txt\_d

}

print(greet(name: "Prateek"))

print(today("Monday","12 July"))

Output:

A picture containing rectangle

Description automatically generated

1. Program to demonstrate Closures.

Ans. Program:

import Foundation

//Closure Example

let difference = { (a: Int, b: Int) -> Bool in

return a>b

}

print(difference(10, 5))

Output:

Text

Description automatically generated with medium confidence

1. Program to demonstrate Single-Expression Closures.

Ans. Program:

import Foundation

//Single Expression Closure

let values = [10,20,2,5,6]

var ascending = values.sorted(by: {a, b in a < b} )

print(ascending)

Output:

A picture containing rectangle

Description automatically generated

1. Program to demonstrate Shorthand Argument Names.

Ans. Program:

import Foundation

//Shorthand Argument Names

let values = [8,4,2,12,5]

var ascending = values.sorted(by: { $0 > $1 } )

print(ascending)

Output:

Text

Description automatically generated with medium confidence

1. Program to demonstrate Trailing Closures.

Ans. Program:

//Program to calculate Factorial of a number using Closure

var num = 4

var ans = 1

func Display(num: Int, fact: ()->(Int)) { //Display function with "fact" closure

print("Factorial of \(num) is \(fact())")

}

Display(num: 3){ //"fact" closure declared here

for i in 1...num{

ans = ans \* i

}

return ans

}

Output:

Text

Description automatically generated with medium confidence

1. Program to demonstrate Enumeration.

Ans. Program:

import Foundation

enum Vehicles { //enumeration contains similar type of data

case bicycle

case motorcycle

case car

}

var myvehicle1 = Vehicles.bicycle //Enter your vehicles here

var myvehicle2 = Vehicles.motorcycle

print("My vehicle 1 is: \(myvehicle1)")

print("My vehicle 2 is: \(myvehicle2)")

Output:

Text

Description automatically generated with medium confidence

1. Program to demonstrate with Switch case.

Ans. Program:

import Foundation

//Switch Case

func menu (choice: Int, qty: Int) {

var cost = 0

switch choice {

case 1: cost = 79

case 2: cost = 99

case 3: cost = 149

default: print("Error: Enter correct value")

}

print("Price = \(cost) Rs, Qty = \(qty) Rs;\n==Est. total Cost = \(cost\*qty) Rs==")

}

print("""

Welcome to Pizza King <)

Select your order:

1. Tomato Cheese Pizza

2. Onion Cheese Pizza

3. Golden Corn Cheeze Pizza

""")

menu(choice: 3, qty: 2)

Output:

Text

Description automatically generated with medium confidence

1. Program to demonstrate Enumeration associated values, Raw values.

Ans. Program:

a. Enumeration associated values

import Foundation

//Enumeration with Associated values

enum Class {

case Student(Int, String) //Student with associated values of Int, String

}

var stud1 = Class.Student(101, "Prateek Panwar") // Passed argument values

switch stud1{ //Switch statement to write separate values

case .Student(let rollnm, let name):

print("Student Details:")

print("Roll number = \(rollnm) & Name = \(name)")

}

a. Output:

Text

Description automatically generated

11. b. Enumeration Raw values

import Foundation

//Enumeration with Raw values

enum Vehicles: Int { //We must specify Return Type when we use RawValues

case bike = 60, car = 100

}

var bike = Vehicles.bike //Assigning to variables

var car = Vehicles.car

print("Speed of bike is \(bike.rawValue) kmph")

print("Speed of car is \(car.rawValue) kmph")

b. Output:

Text

Description automatically generated with medium confidence

1. Program to demonstrate Structure.

Ans. Program:

import Foundation

struct Player { //Structure stores data members and methods

var ammo = 200

var health = 60

}

print("Player ammo = \(Player().ammo) \nPlayer health = \(Player().health)")

Output:

Text

Description automatically generated with medium confidence

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

Ans. Program:

import Foundation

struct Student{

var name = "Rohit"

var roll\_num = 101

}

//Accessing property value

let student1 = Student(name: "Prateek", roll\_num: 105)

print(student1.name,student1.roll\_num)

//Using memberwise initializer

//When we call memberwise init, We omit any property with default values

let student2 = Student()

print(student2.name,student2.roll\_num)

//Using initializer

struct Details{

var vehicle: String

init(vehicle: String){

self.vehicle = "Scooter"

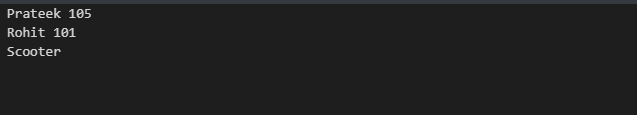
}

}

let data1 = Details(vehicle: "Bicycle")

print(data1.vehicle)

Output:



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

Ans. Program:

Stored, Lazy and Computed Properties:

import Foundation

struct Dimensions {

var l = 0, b = 0 //Stored property "l" and "b"

}

var rect1 = Dimensions(l: 12, b: 5)

class Rectangle {

lazy var perimeter = rect1.l + rect1.b //Lazy property "perimeter"

var dimensions = Dimensions()

var area: Int { //Computed property "area"

get {

let ar = rect1.l \* rect1.b

return ar

}

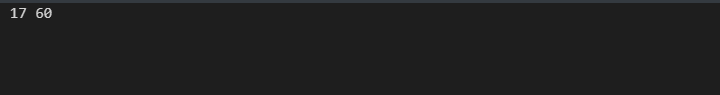
}

}

var shape = Rectangle()

print(shape.perimeter, shape.area)

Output:



Property Observers:

import Foundation

// Property observers

class Scores {

var highscore: Int = 0 {

willSet(newhighscore) { //Property observer "willSet"

print("New high score near way \(newhighscore)")

}

didSet{ //Property observer "didSet"

if highscore > oldValue {

print("Your new highscore is \(highscore)")

}

}

}

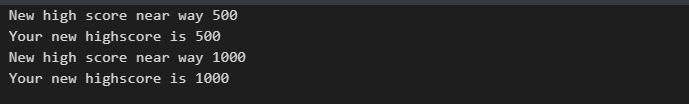
}

let score1 = Scores()

score1.highscore = 500

score1.highscore = 1000

Output:



1. Programs to demonstrate different types of Inheritance in Swift.

Ans. Program:

Multiple Inheritance:

import Foundation

//Multiple inheritance

class Vehicle { //Base class

var speed = 0

}

class Bike: Vehicle { //Derived class 1 (Vehicle -> Bike)

var tires = 2

func description() {

speed = 20

print("Speed of bike is \(speed) & Tires are \(tires)")

}

}

class Car: Vehicle { //Derived class 2 (Vehicle -> Car)

var tires = 4

func description() {

speed = 60

print("Speed of car is \(speed) & Tires are \(tires)")

}

}

let bike = Bike()

bike.tires = 2

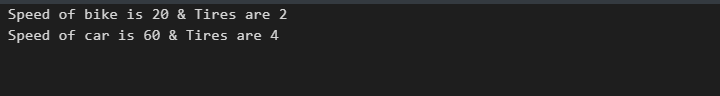
bike.description()

let car = Car()

car.tires = 4

car.description()

Output:



Multilevel inheritance:

import Foundation

//Multiple inheritance

class Vehicle { //Base class

var type = "Car"

}

class Manufacturer: Vehicle { //Derived class 1 (Vehicle -> Bike)

var brand = "Hyundai"

}

class Model: Manufacturer {

var model = "Verna"

func display() {

print("Type: \(type) \nBrand: \(brand) \nModel: \(model)")

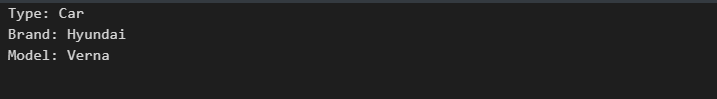
}

}

let carmodel = Model()

carmodel.display()

Output:



1. Programs to demonstrate Methods, Instance Methods, self-Property and Mutating Method

Ans. Program:

Methods, Instance & Self Methods:

import Foundation

//Methods are functions that are associated with a particular type

class Match {

var goal = 0

func increment() { //Method 1

self.goal += 1 //Using self property to add to Current Instance

}

func increment(by amount: Int) { //Method 2

goal += amount

}

func reset() { //Method 3

goal = 0

}

func display(){

print("Total goals = \(goal)")

}

}

let match = Match()

//We call Instance method using <variableName>.methodName()

match.increment() // <- Instance Method

match.display()

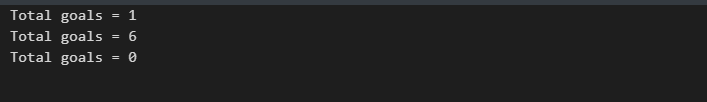
match.increment(by: 5)

match.display()

match.reset()

match.display()

Output:



Mutating Method:

import Foundation

struct Position {

var x = 0, y = 0

mutating func moveby(x posX: Int, y posY: Int) { //Mutating method

self = Position(x: x + posX, y: y + posY)

}

}

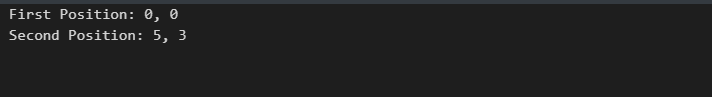
var pos1 = Position()

print("First Position: \(pos1.x), \(pos1.y)")

pos1.moveby(x: 5, y: 3)

print("Second Position: \(pos1.x), \(pos1.y)")

Output:



1. Programs to demonstrate Accessing Superclass Methods, Properties, Overriding Methods and Overriding Properties.

Ans. Program:

import Foundation

class Vehicle { //"Vehicle" is Superclass

var speed = 40

func description()

{

print("I am a bike")

}

}

class Car: Vehicle { //Accessing superClass "Vehicle"

override var speed: Int { //Over riding Property

didSet {

if speed == 100 {

print("Top Speed reached!")

}

}

}

override func description() { //Over riding Method

print("I am a car")

print("Current Speed is \(speed)")

}

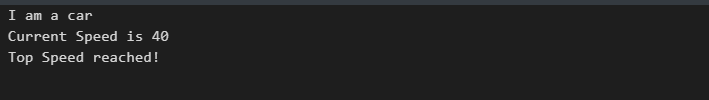
}

let car = Car()

car.description()

car.speed = 100

Output:



1. Programs to demonstrate Initializers, Default Property Values and Custom Initializers.

Ans. Program:

Initializer & Default Property value:

import Foundation

struct Airplane {

var pressure = 0.0 //Default Property Value

var altitude = 0.0

init() { //Initializer

altitude = 200.0

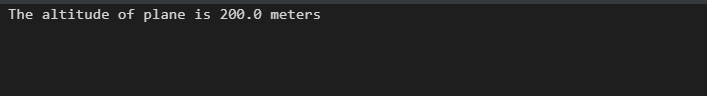
}

}

var a = Airplane()

print("The altitude of plane is \(a.altitude) meters")

Output:



Custom Initializer:

import Foundation

struct Airplane {

var pressure = 0.0 //Default Property Value

var altitude = 200.0

init(setaltitude a: Double) { //Custom Initializer

pressure = altitude / 100

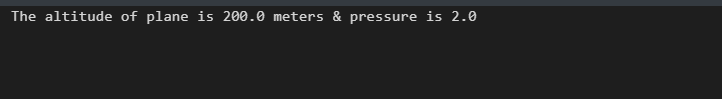
}

}

var a = Airplane(setaltitude: 200)

print("The altitude of plane is \(a.altitude) meters & pressure is \(a.pressure)")

Output:



1. Programs to demonstrate Initializer Inheritance, Overriding and Automatic Initializer Inheritance,

Ans. Program:

import Foundation

class Vehicle {

var speed = 0

var description: String {

return "speed is \(speed) kmph"

}

}

class Bike: Vehicle {

override init() { //Initializer Inheritance & Overriding

super.init()

speed = 40

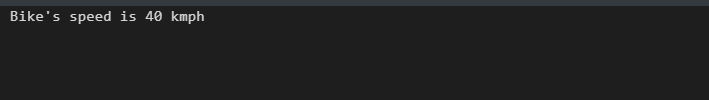
}

}

let bike = Bike()

print("Bike's \(bike.description)")

Output:



Automatic Initializer Inheritance:

import Foundation

class Food {

var name: String

init(name: String) {

self.name = name

}

convenience init() { //Automatic initializer inheritance

self.init(name: "[Unnamed]")

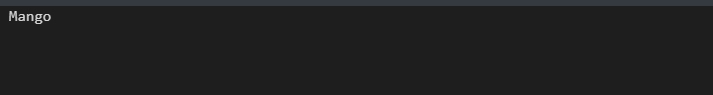
}

}

let fruit1 = Food(name: "Mango")

print(fruit1.name)

Output:



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

Ans. Program:

Failable Initializer:

import Foundation

struct Item {

let itemtype: String

init?(itemtype: String) { //Failable initializer

if itemtype.isEmpty { return nil }

self.itemtype = itemtype

}

}

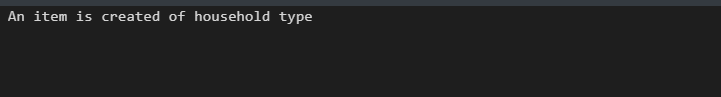
let houseitem = Item(itemtype: "household")

if let bottle = houseitem {

print("An item is created of \(bottle.itemtype) type")

}

Output:



Failable initializers for Enumerations:

import Foundation

enum TemperatureUnit { //Failable initializers for enumeration

case kelvin, celsius, fahrenheit

init?(symbol: Character) {

switch symbol {

case "K":

self = .kelvin

case "C":

self = .celsius

case "F":

self = .fahrenheit

default:

return nil

}

}

}

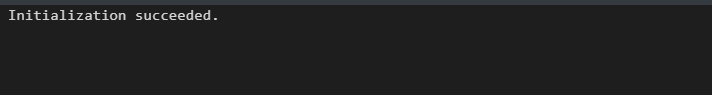
let fahrenheitUnit = TemperatureUnit(symbol: "F")

if fahrenheitUnit != nil {

print("Initialization succeeded.")

}

Output:



Overriding Failable Initializer:

import Foundation

class macBook {

var model: String?

init() {}

init?(model: String) {

if model.isEmpty { return nil }

self.model = model

}

}

class DefaultSet: macBook { //Over riding failable initializer

override init() {

super.init()

self.model = "Unknown"

}

override init(model: String) {

super.init()

if model.isEmpty {

self.model = "Unknown"

} else {

self.model = model

}

}

}

let macBook1 = DefaultSet()

print(macBook1.model)

Output:

