**Q1. Explain the difference between let and var in swift**

In Swift, **let** is used to declare constants, meaning the value assigned to it cannot be changed once it's initialized. On the other hand

**var** is used to declare variables, meaning the value assigned to it can be changed after initialization.

**Immutability with let**: When you use **let** to declare a constant in Swift, you're not just indicating that the variable's value won't change after initialization; you're also signaling to both the compiler and other developers that the variable should remain immutable throughout its lifetime. This helps enforce a functional programming style and can lead to safer, more predictable code.

**Q2. Write a program in swift to find the greatest and smallest among all**

**numbers.**

func findMinMax(numbers: [Int]) -> (min: Int, max: Int)? {

guard !numbers.isEmpty else { return nil }

var min = numbers[0]

var max = numbers[0]

for number in numbers {

if number < min {

min = number

} else if number > max {

max = number

}

}

return (min, max)

}

let numbers = [10, 5, 8, 20, 3]

if let result = findMinMax(numbers: numbers) {

print("Minimum: \(result.min), Maximum: \(result.max)")

} else {

print("Array is empty")

}

**Q3. Print all the even numbers between 1 and 30 using range and**

**where operator.**

for number in 1...30 where number % 2 == 0 {

print(number)

}

**Q4. Explain method overloading in swift.**

Method overloading in Swift allows you to define multiple methods with the same name but different parameters or return types within the same scope. This enables you to provide multiple implementations of a method tailored to handle different input types or perform different tasks while keeping the codebase clean and organized.

class MathFunctions {

// Method to add two integers

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

return a + b

}

// Method to add two doubles

func add(\_ a: Double, \_ b: Double) -> Double {

return a + b

}

// Method to concatenate two strings

func add(\_ a: String, \_ b: String) -> String {

return a + b

}

}

let math = MathFunctions()

// Calling the add methods with different parameter types

let sumIntegers = math.add(5, 3) // Calls add(Int, Int)

let sumDoubles = math.add(3.5, 2.5) // Calls add(Double, Double)

let concatenatedStrings = math.add("Hello, ", "World!") // Calls add(String, String)

print("Sum of integers: \(sumIntegers)")

print("Sum of doubles: \(sumDoubles)")

print("Concatenated strings: \(concatenatedStrings)")

**Q5. Write a program to display assertions and precondition**

1. **Assertions**: Assertions are checks that happen at runtime. They're used to verify that certain conditions are met during the execution of a program. If an assertion condition evaluates to false, the program stops executing, and an error message is logged. Assertions are primarily used for debugging purposes to detect and diagnose unexpected states or conditions in your code.
2. **Preconditions**: Preconditions, like assertions, are checks that occur at runtime. However, preconditions are used to enforce certain conditions that must be true for a particular piece of code to execute correctly. If a precondition is not satisfied, the program stops execution, and an error message is logged. Preconditions are often used to validate input parameters or ensure that the program is in a valid state before proceeding with an operation.

//Assertions

func validate(\_ marks: Int) {

assert(marks >= 0, "Marks cannot be negative")

print("marks are valid")

}

//preconditions

func division(\_ num: Int, deno: Int) -> Int {

precondition(deno != 0, "Denominator should be non-zero")

return num / deno

}

**Q6. Write a menu driven program for student management system using switch case, classes, objects and list.**

class Student {

var name: String

var age: Int

init(name: String, age: Int) {

self.name = name

self.age = age

}

}

var students: [Student] = []

func addStudent() {

print("Enter student name:")

let name = readLine()!

print("Enter student age:")

let age = Int(readLine()!)!

students.append(Student(name: name, age: age))

print("Student added successfully!")

}

func displayStudents() {

print("Students:")

for student in students {

print("Name: \(student.name), Age: \(student.age)")

}

}

func runMenu() {

var choice: Int = 0

while choice != 3 {

print("\nMenu:")

print("1. Add Student")

print("2. Display Students")

print("3. Exit")

print("Enter your choice:")

if let input = readLine(), let number = Int(input) {

choice = number

switch choice {

case 1:

addStudent()

case 2:

displayStudents()

case 3:

print("Exiting program.")

default:

print("Invalid choice. Please try again.")

}

} else {

print("Invalid input. Please enter a number.")

}

}

}

runMenu()

**Q7. Write a program to store phone numbers using dictionary and fetch**

**them by name.**

var phoneBook = [String: String]()

func addPhoneNumber() {

print("Enter name:")

let name = readLine()!

print("Enter phone number:")

let number = readLine()!

phoneBook[name] = number

print("Phone number added successfully!")

}

func fetchPhoneNumber() {

print("Enter name to fetch phone number:")

let name = readLine()!

if let number = phoneBook[name] {

print("Phone number: \(number)")

} else {

print("Phone number not found for \(name)")

}

}

func runPhoneBook() {

var choice: Int = 0

while choice != 3 {

print("\nMenu:")

print("1. Add Phone Number")

print("2. Fetch Phone Number")

print("3. Exit")

print("Enter your choice:")

if let input = readLine(), let number = Int(input) {

choice = number

switch choice {

case 1:

addPhoneNumber()

case 2:

fetchPhoneNumber()

case 3:

print("Exiting program.")

default:

print("Invalid choice. Please try again.")

}

} else {

print("Invalid input. Please enter a number.")

}

}

}

runPhoneBook()

**Q8. Write down the features of swift programming language**

* Safety: Swift eliminates many common programming errors by enforcing safe programming patterns.
* Speed: Swift is designed to be fast and optimized for performance.
* Expressive syntax: Swift has a concise and expressive syntax that is easy to read and write.
* Modern features: Swift includes modern features like closures, generics, optionals, and type inference.
* Interoperability: Swift is designed to work seamlessly with Objective-C, allowing you to use Swift and Objective-C code together in the same project.
* Memory management: Swift uses Automatic Reference Counting (ARC) to manage memory automatically.
* Protocol-oriented programming: Swift encourages the use of protocols and protocol extensions for code reuse and composition.

**Q9. Write a program to find a factorial of a number using recursion**

func factorial(\_ n: Int) -> Int {

if n == 0 {

return 1

}

return n \* factorial(n - 1)

}

let number = 5

print("Factorial of \(number) is \(factorial(number))")

**Q10. Write a program to Delete last occurrence of an element from a**

**linked list**

func deleteLastOccurrence<T: Equatable>(\_ element: T, from array: inout [T]) {

if let indexToRemove = array.lastIndex(of: element) {

array.remove(at: indexToRemove)

}

}

// Example usage:

var numbers = [1, 2, 3, 2, 4]

print("Original List:")

print(numbers) // Output: [1, 2, 3, 2, 4]

deleteLastOccurrence(2, from: &numbers)

print("List after deleting last occurrence of 2:")

print(numbers) // Output: [1, 2, 3, 4]