```
  Programatically questions
//Count the number of vowels in given string
func isVowel(char: Character) -> Bool {
   func isVowels(char: Character) -> Bool{
    return char == "a" || char == "e" || char == "i" || char == "o" || char == "u"
}

func countVowels(in string: String) -> Int {
    var vowelCount = 0
    for char in string {
        if isVowel(char: char) {
            vowelCount += 1
        }
    }

    return vowelCount
}

let inputString = "HeAllo, world!"
let vowelCount = countVowels(in: inputString)

// find second largest number of an array
func findSecondLargestNumberInArray(numbers: [Int]) -> Int? {
    guard numbers.count > 2 else{
        return nil
    }

    var largest: Int = 0
    var secondLargest: Int = 0

    for i in numbers{
        if i > largest{
            secondLargest = largest
            largest = i
        }else if i > secondLargest && i < largest{
           secondLargest = i
        }
    }
    return secondLargest
}



let num = [-1,-2,-3,-30,-4440,-230]
let result1 = findSecondLargestNumberInArray(numbers: num)
print(result1)

//
```

```
func reverseStringFunction(str: String) -> String{
    var reversedStr: String = ""
    for i in str{
        reversedStr = String(i) + reversedStr
    }
    return reversedStr
}

let result = reverseStringFunction(str: "login")
//print(result)

//

///
//find duplicate value for an array
func findDuplicateValueOfAnArray(IntegerArray: [Int]) -> [Int]{
    var duplicateArray = Set<Int>()
    var seenArray = Set<Int>()

    for i in IntegerArray{
        if seenArray.contains(i){
            duplicateArray.insert(i)
        }else{
            seenArray.insert(i)
        }
    }
    return Array(duplicateArray)
}

let number = [1,2,4,2,4,1,6,3,7,10,2,3]
let result = findDuplicateValueOfAnArray(IntegerArray: number)
print(result)
//

func removeDuplicates(_ array: [Int]) -> [Int] {
    let uniqueValues = Set(array)
    return Array(uniqueValues)
}

var numbers = [3, 1, 4, 2, 2, 1, 5, 3]
numbers = removeDuplicates(numbers)
print("Array after removing duplicates: \(numbers)")

//swap two number without using third variable
var a = 5
var b = 10
```

```
a = a + b
b = a - b
a = a - b
print(a,b)
//

// merge Two Unsorted Array
func mergeSortedArray(array1: [Int], array2: [Int]) -> [Int]{
    var mergeArray = [Int]()
    var sortedArray1 = [Int]()
    var sortedArray2 = [Int]()
    sortedArray1 = array1
    sortedArray2 = array2
    mergeArray = sortedArray1
    mergeArray.append(contentsOf: sortedArray2)
    var sortedMerge = sortedArrayFunction(array: mergeArray)
    return sortedMerge
}


let ar1 = [1,2,3,4,7,8]
let ar2 = [5,6,9,10]

let mergeArray = mergeSortedArray(array1: ar1, array2: ar2)
print(mergeArray)
//

//anagram string in swift
//anagram means done string is same character hold
//
func checkAnagram(str1: String, str2: String) -> Bool{
    guard str1.count == str2.count else { return false}

    for i in str1{
        if !str2.contains(i){
            return false
        }
    }
    return true
}
//
//anagram means done string is same character hold
func isAnagram(_ str1: String, _ str2: String) -> Bool {
    let lowerStr1 = str1.lowercased()
    let lowerStr2 = str2.lowercased()
    var isReturn = false
```

```
    // Check if the lengths of the strings are the same
    guard lowerStr1.count == lowerStr2.count else {
        return false
    }

    let charArray1 = Array(lowerStr1)
    let charArray2 = Array(lowerStr2)
    let str1Sorted = sortedArrayFunction(array: charArray1)
    let str2Sorted = sortedArrayFunction(array: charArray2)
    if str1Sorted == str2Sorted{
        isReturn = true
    }
    return isReturn
}

// Example usage
let str1 = "ajay"
let str2 = "risalat"

if isAnagram(str1, str2) {
    print("\(str1) and \(str2) are anagrams.")
} else {
    print("\(str1) and \(str2) are not anagrams.")
}

//

func factorial(_ n: Int) -> Int {
    if n <= 1 {
        return 1
    }
    return n * factorial(n - 1)
}

// Example usage
let number = 5
let result = factorial(number)
print("Factorial of \(number) is: \(result)")  // Output: 120
//


// palindrome means string ka reverse phi same ho
func isPalindrome(str: String) -> Bool{
    var reversedStr = ""
    for i in str{
        reversedStr = String(i) + reversedStr
    }
```

```
    return str.lowercased() == reversedStr.lowercased()
}

let str = "MalayalAm"
let results = isPalindrome(str: str)
print(results)

//

// Append the new element without using append method
myArray += Array(arrayLiteral: newValue)

print(myArray) // Output: [1, 2, 3, 4, 5, 6]
//
```

Closed Range (a...b):

- A closed range includes both a and b as part of the range.

```
let closedRange = 1...5 // Represents the range from 1 to 5 (1, 2, 3, 4, 5)
```

//

Half-Open Range (a..<b):

- A half-open range includes a but not b.

```
let halfOpenRange = 1..<5 // Represents the range from 1 to 4 (1, 2, 3, 4)
```

//

One-Sided Range:

- One-sided ranges are used when you only need one bound.

```
let lowerBoundRange = ..<5 // Represents the range from the start up to, but
not including, 5 (0, 1, 2, 3, 4)
let upperBoundRange = 1... // Represents the range from 1 to the end (1, 2,
3, ...)

//
```

The ~= operator in Swift is called the "Pattern Matching Operator" or "Pattern Matching Expression."

```
let value = 42

if value == 0 {
    print("Value is zero")
} else if 1...10 ~= value {
    print("Value is in the closed range of 1 to 10")
} else if value >= 11 {
    print("Value is greater than or equal to 11")
} else if value < 0 {
    print("Value is negative")
} else if 20...30 ~= value {
```

```
    print("Value is in the closed range of 20 to 30")
} else {
    print("Value doesn't match any of the specified conditions")
}

//
let array = [1,2,3,4,5,6,7,8]

for i in array{
    if 1...2 ~= i{
        print("HI")
    }else if i > 3 && i <= 7{
        print("HELLO")
    }
}
//

let value = 7

switch value {
case 0:
    print("Value is zero")
case 1...10:
    print("Value is between 1 and 10")
case let x where x % 2 == 0:
    print("Value is even")
case let x where x % 2 != 0:
    print("Value is odd")
default:
    print("Value doesn't match any pattern")
}

//
let set1: Set<Int> = [1, 2, 3]
let set2: Set<Int> = [3, 4, 5]

let mergedSet = set1.union(set2)

print(mergedSet) // Output: [5, 2, 3, 1, 4]
//

var set1: Set<Int> = [1,2,3,4]
var set2: Set<Int> = [1,2,3,4,5]


var mergeSet = set1.union(set2)
```

```swift
var mer = set1.intersection(set2)

print(mergeSet, mer)
//

//
Set sorted in swift

var set: Set<Int> = [1,2,3,4,6,5]


let sr = sortedArr(arr: Array(set))
print(sr)
//

//count number of character in swift
func numberOfCount(elemenT: String) -> Int{
    var count = 0
    for _ in elemenT{
        count += 1
    }
    return count
}

let resultt = numberOfCount(elemenT: "ababs")
print(resultt)
//

//count the number of words in swift
func countTheNumberOfWords(elemenT: String) -> Int{
    var count = 1
    for i in elemenT{
        if i.isWhitespace{
            count += 1
        }
    }
    return count
}

let resultt = countTheNumberOfWords(elemenT: "hh hjh bjhbj jknknjnjnj oppo")
print(resultt)
//

//find the common elements in swift
func findTheCommonElementOfArray(arr1: [Int], arr2: [Int]) -> [Int]{
    var commonElements: [Int] = []
```

```
    for i in arr1{
        for j in arr2{
            if i == j{
                commonElements += Array(arrayLiteral: i)
            }
        }
    }
    return commonElements
}

let res = findTheCommonElementOfArray(arr1: [1,2,3,4,6], arr2: [1,2,5,6])
print(res)
//
Count the repeated character in given string and print the frequency of
elements in array
func countTheRepeatedChar(str: String) -> [Character: Int]{
    var charCountDic = [Character: Int]()
    for i in str{
        if i.isLetter{
            if let count = charCountDic[i]{
                charCountDic[i] = count + 1
            }else{
                charCountDic[i] = 1
            }
        }
    }

    return charCountDic
}

let res = countTheRepeatedChar(str: "h uuuu rrr")
print(res)
//
find the first repeated character in string
extension String {
    func firstRepeatedCharacter() -> Character {
        var minChar: Character = " "
        for i in self {
            if minChar == i {
                return i
            }else{
                minChar = i
            }
        }
        return minChar
    }
```

```
}
let stri = "heo gekss"
print(stri.firstRepeatedCharacter())
//
func printDuplicateCharacter(str: String) -> Character? {
    var resultDic = [Character: Int]()

    for i in str{
        resultDic[i, default: 0] += 1
    }

    for j in str{
        if resultDic[j]! >= 2{
            print(j)
        }
    }
    return nil
}

print(printDuplicateCharacter(str: "rissurlopkjbeee"))
//
maximum consecutive repeating character in string
func maximumConsecutiveRepeatingChar(str: String) -> (Character, Int)?{
    var currentChar: Character = " "
    var currentCount = 0
    var maxChar: Character = " "
    var maxCount = 0

    for i in str{
        if i == currentChar{
            currentCount += 1
        }else{
            currentChar = i
            currentCount = 1
        }

        if currentCount > maxCount{
            maxCount = currentCount
            maxChar = currentChar
        }
    }

    return (maxChar, maxCount)
}

print(printDuplicateCharacter(str: "geeekk"))
//
```

```
Check reference count arc
class Test{
    var name = "Hello"
    var age = 99.0
}

let obj = Test()
let refCount = CFGetRetainCount(obj)
print(refCount)
//

var names = ["Hello", "Swift"]

names += Array(arrayLiteral: "How are you")
print(names)

let index = names[1]
print(index)

if names.isEmpty {
    print("empty")
}else{
    print("Fulll")
}

let str1 = "a"
let str2 = "b"
let combineStr = str1 + str2
print(combineStr)
//

//contains logics
func customContainsMethod(fullString: [String], findString: String) -> Bool{
    var fullChar = fullString
    var isReturn = false
    for i in fullChar{
        if i == findString{
            isReturn = true
            break
        }

    }
    return isReturn
}

let stro = ["swift is a programming languages","fsvf"]
```

```swift
let find = "swift is a programming languages"

if customContainsMethod(fullString: stro, findString: find){
    print("Main String contains sub string")
}else{
    print("not contains")
}

//even and odd number
func checkEvenOdd(nums: Int) -> Bool{
    return nums % 2 == 0
}

if checkEvenOdd(nums: 11){
    print("Even \(11)")
}else{
    print("Odd \(11)")
}

//
typealias Risalat = String

let name: Risalat = "Hello"
print(name)
//
//AssociatedTypes
protocol PrintValue{
    associatedtype ValueFromCallerSide

    func printName(str: ValueFromCallerSide)
}

class Test: PrintValue{
    typealias ValueFromCallerSide = Double
    func printName(str: ValueFromCallerSide) {
        print(str)
    }

}

let obj = Test()
obj.printName(str: 555.0)
//

Object oriented programming concept in swift
//there are many features
1> Class and Objects .... That is used to create object with method
```

Class Person{
Var name = “”
Var age: int = 0
Func sayHello(){
Print(age, name)
}
}
Let objc = Person()
Obj.name = “”swift
Obj.age = 22.0
Obj.sayHello()
//
2> Inheritance
Inheritance **is** a procees one **class** inherit from another **class is** called inheritance

```
class Test{
    var name = "Hello"

    func sayHello(){
        print(name)
    }
}

class Test2: Test{
    var data = "Inherit from another class"

    func anotherBollerCalled(){
        sayHello()
    }
}

let obj = Test2()
obj.anotherBollerCalled()
```

//
3>Encapsulation
you can use access control modifier like **private fileprivate internal public** to control the visibility and access of the properties **in** swift **is** called encapsulation

```
class BankAccount{
    private var balance: Double = 0.0

    func deposit(amount: Double){
        balance += amount
    }

    func withdraw(amount: Double){
        if amount <= balance{
```

```
            balance -= amount
        }else{
            print("Insufficient Balance")
        }
    }

    func checkBalance(){
        print(balance)
    }
}


let bank = BankAccount()
bank.deposit(amount: 3000)
bank.checkBalance()
bank.withdraw(amount: 1200)
bank.checkBalance()
bank.withdraw(amount: 200)
bank.checkBalance()
//
```
Polymorphism deff- one name having multiple form
Polymorphism is a three type method overloading and method overriding and parametric polymorphism

Method overloading: when you have multiple method with the same name in same class but they have different parameter is called method overloading.
```
class MathOpertaions{

    func sum(a: Int, b: Int) -> Int {
        return a + b
    }

    func sum(a: Double, b: Double, c: Double) -> Double {
        return a + b + c
    }

}

let obj = MathOpertaions()
let sum1 = obj.sum(a: 2, b: 3)
let sum2 = obj.sum(a: 2.0, b: 3.0, c: 4.0)
print(sum1, sum2)
```

2> Method Overriding it allow you to define method base class and then provide different implementations derived class.
```
class Shape{
    func area() -> Double{
```

```
        return 0.0
    }
}

class Circle: Shape{
    var radius: Double = 5.0

    override func area() -> Double {
        return Double.pi * radius * radius
    }
}

let obj = Circle()
print(obj.area())
//
Parametric polymorphism>
Parametric polymorphism it allow to write a generic code in swift that is used to
any type for example
func executeRequest<T: Decodable>(data: Data) throws -> T  {
    let decoder = JSONDecoder()
    do {
        let model = try decoder.decode(T.self, from: data)
        return model
    } catch {
        throw error
    }
}

struct Kheema: Codable {
    let name: String
    let age: Int
    var son: SonRohan

}

struct SonRohan: Codable{
    let name: String
    let age: Int
}

let objc = Kheema(name: "Maneesh", age: 20, son: SonRohan(name: "Rohan",
age: 12))

let encoder = JSONEncoder()
let data = try encoder.encode(objc)
print(data)
```

```
do {
    let personModel: Kheema = try executeRequest(data: data)
    print(personModel)  // Output: John Doe

} catch {
    print("Error: \(error)")
}
//

Result Type:
Result type it is a enum that is used to handle operation like sucess and failure
with an error in swift
enum NetworkError: Error{
    case noInternet
    case serverError
}


func fetchData(completion: @escaping (Result<String, NetworkError>) ->
Void){
    let success = true

    if success {
        completion(.success("Data SuccessFully Fetched"))
    }else{
        completion(.failure(.serverError))
    }
}

fetchData { result in
    switch result{
    case .success(let data):
        print("data\(data)")
    case .failure(let error):
        print(error)
    }
}

//
//delegate example in swift
protocol DataDelegte{
    func sendData(data: Any)
}

class DataSender{
```

```
    var delegate: DataDelegte?

    func sendDataToReciever(data: Any){
        delegate?.sendData(data: data)
    }

}

class DataReciever: DataDelegte{
    func sendData(data: Any) {
        print("recived data \(data)")
    }
}

let sender = DataSender()
let reciever = DataReciever()

sender.delegate = reciever
sender.sendDataToReciever(data: "Hello delegate")
//

Check prime number prime number wo number home hai jo apne app or 1 se
divide hate hai is called prime number
func isPrimeNumber(num: Int) -> Bool{
    if num <= 1{
        return false
    }
    if num <= 3{
        return true
    }
    if num % 2 == 0 || num % 3 == 0{
        return false
    }
    return true
}

let num = 44
if isPrimeNumber(num: num){
    print("is Prime Number: \(num)")
}else{
    print("is not Prime Number: \(num)")
}
//
Protocool Extension in swift
// Define a protocol named "Drawable"
protocol AddtionProtocol {
    func sum()
```

```
}

// Create a protocol extension with a default implementation for "sum()"
extension AddtionProtocol {
    func sum() {
      let a = 9
      let b = 10
      var c = a + b
      print("default implementation: \(c)")
    }

    func printFunc(){
        print("Printed functions")
    }
}

// Create a struct that conforms to the "Additionprotocol" protocol
struct SumOfNumber: AddtionProtocol {
    func sum() {
        print("Hello no arguments")
    }
}

struct SumOFArgu: AddtionProtocol {
    // No "sum()" implementation here, so it will use the default implementation.
}

// Usage
let obj1 = SumOFArgu()
let obj2 = SumOfNumber()

obj1.sum() // Output: Drawing a circle
obj2.sum() // Output: Default drawing implementation
obj2.printFunc()
//

Fibonacci sequence is a sequence in which each number is the sum of the two
preceding ones.
func fibonacci(_ n: Int) -> Int {
    if n <= 0 {
        return 0
    } else if n == 1 {
        return 1
    }

    return fibonacci(n - 1) + fibonacci(n - 2)
}
```

```swift
let num = 10
for i in 0..<num {
    let fib = fibonacci(i)
    print(fib, terminator: " ")
}

//
//calculate the power of number
func calculatePower(_ base: Int, _ power: Int) -> Int{
    if power == 0{
        return 1
    }
    return base * calculatePower(base, power - 1)
}

let result = calculatePower(5, 3)
print(result)
//
//First Non Repeating char
func firstNonRepeatingchar(in str: String) -> Character? {
    var charCountDic = [Character: Int]()

    for i in str{
        charCountDic[i, default: 0] += 1
    }

    for i in str{
        if charCountDic[i] == 1{
            return i
        }
    }
    return nil
}

let str = "hheellswift"
let res = firstNonRepeatingchar(in: str)
print(res)
//
func binarySearchAlgo(_ arr: [Int], target: Int) -> Int?{
    var left = 0
    var right = arr.count - 1

    while left <= right{
        let mid = (left + right) / 2
        if arr[mid] == target{
            return mid
```

```
        }else if arr[mid] < target{
            left = mid + 1
        }else if arr[mid] > target{
            right = mid - 1
        }
    }

    return nil
}
let result = binarySearchAlgo([2, 5, 8, 12, 16, 23, 38, 56, 72, 91], target: 5)
print("Element is presented by: \(result ?? 0) index")//
//
```

How to avoid retain cycle in swift
Answer you can use weak variable decalre that avoid

```
class Person{
    var name: String
  weak  var apartment: Apartment?
    init(name: String){
        self.name = name
    }

    deinit{
      print("Person is deinitializes")
    }
}



class Apartment{
    var name: String
    var person: Person?
    init(name: String){
        self.name = name
    }

    deinit{
        print("Apartment is deinitializes")
    }
}

//instance created

var person: Person? = Person(name: "Joh(n")
var apartment: Apartment? = Apartment(name: "Apartment")

//seting refrence that is create retain cycle
```

```
person?.apartment = apartment
apartment?.person = person
// at this point a reatin cycle exists because person and apartment refrence each other strongly

//attempt to denit

person = nil
apartment = nil
```

Retain cycle in swift
Retain cycle it occurs when two or more object have strong reference each. Other preventing them from being deallocated object by the ARC.

```
class Person{
    var name: String
    var apartment: Apartment?
    init(name: String){
        self.name = name
    }

    deinit{
      print("Name is deinitializes")
    }
}


class Apartment{
    var name: String
    var person: Person?
    init(name: String){
        self.name = name
    }

    deinit{
        print("Apartment is deinitializes")
    }
}

//instance created

var person: Person? = Person(name: "Joh(n")
var apartment: Apartment? = Apartment(name: "Apartment")
```

```
//seting refrence that is create retain cycle

person?.apartment = apartment
apartment?.person = person
// at this point a reatin cycle exists because person and apartment refrence each other strongly

//attempt to denit

person = nil
apartment = nil

//dispute setting them to nil the deinit method wont be called because of the retain cycle

//
func sortedIntersection(_ array1: [Int], _ array2: [Int]) -> [Int] {
    let sortedArray1 = array1.sorted()
    let sortedArray2 = array2.sorted()

    var result = [Int]()
    var index1 = 0
    var index2 = 0

    while index1 < sortedArray1.count && index2 < sortedArray2.count {
        let element1 = sortedArray1[index1]
        let element2 = sortedArray2[index2]

        if element1 == element2 {
            result += Array(arrayLiteral: element1)
            index1 += 1
            index2 += 1
        } else if element1 < element2 {
            index1 += 1
        } else if element1 > element2 {
            index2 += 1
        }
    }

    return result
}

let array1 = [1, 2, 2, 2,4, 3, 4, 5,8]
let array2 = [2, 2, 4, 5, 6,4]
let intersection = sortedIntersection(array1, array2)
print(intersection) // Output: [2, 2, 3, 5]
```

```swift
//

print("Start xcode")

//
func checkPalindromOfArray(arr: [Int]) -> Bool{
    var reversedArray: String = ""
    var local = ""
    for i in arr{
        reversedArray = String(i) + reversedArray
        local = local + String(i)
    }
    print(reversedArray)
    print(local)
    if reversedArray == local{
        return true
    }
    return false
}

let num = [3,2,3]
let res = checkPalindromOfArray(arr: num)
print(res)
//
Custom phone call search
func searchEnginePhoneNumber(names: [String], _ nums: [String], _ search:
String) -> (String, String)? {
    for i in names{
        if i.contains(search){
            for (index, i) in names.enumerated(){
                if i.contains(search){
                    let num = nums[index]
                    return (num, i)
                }
            }

        }else{
            for (index, i) in nums.enumerated() {
                if i.contains(search){
                    let name = names[index]
                    return (i, name)
                }
            }

        }
    }
    return nil
```

```
}



let names = ["saif", "zaid", "risalat", "saleem", "rahat"]
let nums =  ["8755445089", "6395160006", "9058683307", "8126605166",
"9870692531"]
let search = "z"

if let (num, name) = searchEnginePhoneNumber(names: names, nums, search)
{
    print("Results: \(name): \(num)")
} else {
    print("No solution found.")
}
//

Weak reference
class Person {
    var name: String
    weak var home: Home?

    init(name: String) {
        self.name = name
    }
}

class Home{
    var name = "Home"
}



var home: Home? = Home()
var person: Person? = Person(name: "Risalat")
person?.home = home

home = nil
print(person?.home?.name) // Prints nil
//
Strong reference
class Person {
    var name: String
    var home: Home?

    init(name: String) {
        self.name = name
    }
```

```
}

class Home{
    var name = "Home"
}


var home: Home? = Home()
var person: Person? = Person(name: "Risalat")
person?.home = home

home = nil
print(person?.home?.name) // Prints nil
//
Unowned reference
class Person {
    var name: String
    unowned var home: Home?

    init(name: String) {
        self.name = name
    }
}

class Home{
    var name = "Home"
}


var home: Home? = Home()
var person: Person? = Person(name: "Risalat")
person?.home = home

home = nil
print(person?.home?.name) // Prints nil

//
//MVVM Model Example
struct TestDataModel{
    var name: String?
    var age: Int?
}


class TestViewModel {
    var model: TestDataModel = TestDataModel(name: "Hello", age: 23)
}
```

```swift
class ViewController{
    var viewModel: TestViewModel = TestViewModel()

    func printModelData(){
        print(viewModel.model.name ?? "")
    }
}


let obj = ViewController()
obj.viewModel.model.name = "Model"
obj.printModelData()
//
Madam
func isPalindrom(str: String) -> Bool {
    guard !str.isEmpty else { return false }
    let array = Array(str)
    let l = str.count / 2
    for i in 0..<l {
        if array[i] != array[str.count - i - 1] {
            return false
        }
    }
    return true
}

print(isPalindrom(str: "madam"))



func sortString(str: String) -> String {
    var array = Array(str)
    let count = array.count

    for i in 0..<count {
        for j in (i + 1)..<count {
            if array[i] > array[j] {
                let temp = array[i]
                array[i] = array[j]
                array[j] = temp
            }
        }
    }
    return String(array)
}
```

```swift
func findCountOfSubString(str: String, searchStr: String) -> Int{
    var count = 0
    var temp = ""
    for i in str{
        temp += String(i)
        guard temp.count == searchStr.count else{
            continue
        }
        if temp == searchStr{
            count += 1
            temp.remove(at: temp.startIndex)
            continue
        }
        temp.remove(at: temp.startIndex)
    }
    return count
}


print(findCountOfSubString(str: "risalat", searchStr: "risalat"))

//sortedarray
func sortedArray(arr: [Int]) -> [Int]{
    var sortedArray = arr
    let n = sortedArray.count

    for i in 0..<n{
        for j in (i + 1)..<n{
            if sortedArray[i] > sortedArray[j]{
                let tem = sortedArray[i]
                sortedArray[i] = sortedArray[j]
                sortedArray[j] = tem
            }
        }
    }
    return sortedArray
}

let numbs = [4,56,77,32,1,2]
let resul = sortedArray(arr: numbs)
print(resul)
//


func sortedArray(arr: [Int]) -> Int{
```

```
    var sortedArray = arr
    let n = sortedArray.count

    for i in 0..<n{
        for j in (i + 1)..<n{
            if sortedArray[i] > sortedArray[j]{
                let tem = sortedArray[i]
                sortedArray[i] = sortedArray[j]
                sortedArray[j] = tem
            }
        }
    }
    return sortedArray[1]
}

let numbs = [4,56,77,32,1,32]
let resul = sortedArray(arr: numbs)
print(resul)
//



//MVC Design patter
//Model
struct Person{
    var name: String
    var age: Int
}

//View
class PersonView{
    func printDetails(person: Person){
        print("Detail of person: \(person.name) : \(person.age)")
    }
}

//viewController it contains business logics etc
class PersonController{
    var person: Person
    var personView: PersonView
    init(person: Person, personView: PersonView){
        self.person = person
        self.personView = personView
    }

    func updatePersonDetail(name: String, age: Int){
        person.name = name
```

```
        person.age = age
    }

    func displayPersondetail(){
        personView.printDetails(person: person)
    }
}


let person = Person(name: "Risalat", age: 22)

let personView = PersonView()

let personViewController = PersonController(person: person, personView:
personView)

personViewController.displayPersondetail()
personViewController.updatePersonDetail(name: "Swift", age: 50)
personViewController.displayPersondetail()
//

Designated vs Convenience init example
//Designated init
class Vehicle{
    var wheels: Int
    //designated init
    init(wheels: Int){
        self.wheels = wheels
    }
}

class Car: Vehicle{
    var name: String

     //designated init
    init(name: String, wheels: Int){
        //firstly initialize own property after that init super class property
        self.name = name
        super.init(wheels: wheels)
    }
}

let obj = Car(name: "Truck", wheels: 12)
print(obj.name, obj.wheels)
//

Convenience init
```

```
//Convenience init
class MyClass{
    var name: String
    var age: Int
    //desig init
    init(name: String, age: Int){
        self.name = name
        self.age = age
    }
    //convenience init
    convenience init(name: String){
        self.init(name: name, age: 0)
    }
}

let obj = MyClass(name: "Rissu")
let obj2 = MyClass(name: "Risalat", age: 20)
print(obj.name, obj2.name, obj2.age)
//

//Synchronous programming
func testSysnchronousProgramming(){
    print("Task 1")
    print("Task 2")
    print("Task 3")
}
testSysnchronousProgramming()

//Asynchronous programming
//Synchronoius Programming
func testSysnchrounousProgramming(completion: @escaping (() -> ())){
    print("Task1")
    print("Task2")
    DispatchQueue.global().async {
        completion()
    }
    print("task4")

}

testSysnchrounousProgramming {
    print("Task3")
}

//Memberwise initialisers
struct Test{
    var name: String
```

```
    var age: Int
}

let obj = Test(name: "Risalat", age: 23)
print(obj.name, obj.age)

//Failable initialisers
class Test{
    var value: Int
    init?(value: Int){
        if value < 0{
        return nil
        }
        self.value = value
    }
}

if let obj = Test(value: -88){
    print(obj.value, "Object created")
}else{
    print("failed to create object")
}
//
//
func customMax(num1: Int, num2: Int) -> Int{
    guard num1 > num2 else{
        return num2
    }
    return num1
}

//
Enums (enumerations) are a way to define a common type for a group of
related values. There are several types of enums in Swift,
//Basic Enum
enum CompassDirection {
    case north
    case south
    case east
    case west
}

let direction = CompassDirection.north
switch direction{
case .north:
    print("North")
case .south:
```

```
    print("south")
case .east:
    print("east")
case .west:
    print("west")
}
//
Enum with Associated Values:
which allow you to attach additional different type of data to each enum case
enum Result<T> {
    case success(T)
    case failure(Error)
}

enum NetworkError: Error{
    case timeout
    case noInternet
}

let successResult: Result<String> = .failure(NetworkError.noInternet)

switch successResult{
case .success(let data):
    print(data)
case .failure(let error):
    print(error)
}
//
Enum with Raw Values:
You can also assign raw values to enum cases, which can be of any type, such as integers or strings.
enum HTTPStatusCode: Int {
    case ok = 200
    case badRequest = 400
    case notFound = 404
}

let statusCode = HTTPStatusCode.badRequest.rawValue
print(statusCode)
//

//Enum with Methods
enum Shape {
    case circle(radius: Double)
    case rectangle(width: Double, height: Double)

    func area() -> Double{
```

```
        switch self {
        case .circle(let radius):
            return .pi * pow(radius, 2)
        case .rectangle(let width, let height):
            return width * height
        }
    }
}

let circle = Shape.circle(radius: 5.0)
let rectangle = Shape.rectangle(width: 4.0, height: 6.0)
let circleArea = circle.area()
let rectangleArea = rectangle.area()
//
GCD manage task concurrency like asynchronous operations for example
func testSysnchrounousProgramming(completion: @escaping (() -> ())){
    print("Task1")
    print("Task2")
    DispatchQueue.global().async {
        completion()
    }
    print("task4")

}

testSysnchrounousProgramming {
    print("Task3")
}
//
Disctionalry
Dictionary

// Creating an empty dictionary with String keys and Int values
var myDictionary: [String: Int] = [:]

// Adding key-value pairs to the dictionary
myDictionary["Alice"] = 25
myDictionary["Bob"] = 30
myDictionary["Charlie"] = 35

// Accessing values using keys
let aliceAge = myDictionary["Alice"] // Optional(25)

// Modifying a value
myDictionary["Bob"] = 32

// Removing a key-value pair
```

```
myDictionary["Charlie"] = nil

// Checking if a key exists in the dictionary
let containsAlice = myDictionary.keys.contains("Alice") // true

// Iterating over the dictionary
for (name, age) in myDictionary {
    print("\(name): \(age)")
}
//
//If sum of any two elements of array results in given value. N
func findPairInGivenTargetSum(arr: [Int], target: Int) -> (Int, Int)?{
    var result = [Int]()

    for i in arr{
        let complement = target - i
        if result.contains(complement){
            return (i, complement)
        }
        result.append(i)
    }



    return nil
}
let nums = [2,3,2,3,4,-5,6]
if let (a,b) = findPairInGivenTargetSum(arr: nums, target: -1) {
    print(a, b)
}

//
func sortedArray(arr: [Int]) -> [Int]{
    var sortedArray = arr
    let n = sortedArray.count

    for i in 0..<n{
        for j in (i + 1)..<n{
            if sortedArray[i] > sortedArray[j]{
                let tem = sortedArray[i]
                sortedArray[i] = sortedArray[j]
                sortedArray[j] = tem
            }
        }
    }
    return sortedArray
}
```

```swift
func customFilter(arr: [Int],_ condition: ((Int)->Bool)) -> [Int]{
    var filterArray = [Int]()
    for i in arr{
        if condition(i){
            filterArray.append(i)
        }
    }
    return filterArray
}
//

func findMissingNo(arr: [Int]) -> [Int] {
    guard !arr.isEmpty else { return [] }
    let array = sortedArray(arr: arr)
    let start = array[0]
    let end = array[arr.count - 1]
    var missingNumbers = [Int]()
    for item in start...end {
        if !array.contains(item) {
            missingNumbers.append(item)
        }
    }
    return missingNumbers
}
//
print(findMissingNo(arr: []))
//
//
func customFilter(arr: String,_ condition: ((Character)->Bool)) -> [Character]{
    var filterArray = [Character]()
    for i in arr{
        if condition(i){
            filterArray.append(i)
        }
    }
    return filterArray
}

func countOccurrencesOfThreeInRange(_ start: Int, _ end: Int, target: Int) ->
Int {
    let strTarget = String(target)
    var count = 0
    for number in start...end {
        let str = String(number)
        let countIn = customFilter(arr: str) { char in
            String(char) == strTarget
```

```
        }.count
        count += countIn
    }
    return count
}


let start = 1
let end = 100
let countOfThrees = countOccurrencesOfThreeInRange(start, end, target: 3)

print("The count of 3s in the range \(start) to \(end) is \(countOfThrees)")
//

func reverseStringAlsoFirstUppercasedSecondLovercased(_ input: String) ->
String {
    var reversed = ""

    for i in 0..<input.count {
        guard i != 0 else {
            reversed = String(input[input.index(input.startIndex, offsetBy:
i)].lowercased()) + reversed
            continue
        }
        guard i != (input.count - 1) else {
            reversed = String(input[input.index(input.startIndex, offsetBy:
(input.count - 1))].uppercased()) + reversed
            continue
        }
        reversed = String(input[input.index(input.startIndex, offsetBy: i)]) +
reversed
    }

    return reversed
}

let originalString = "risalat"
let reversedString =
reverseStringAlsoFirstUppercasedSecondLovercased(originalString)

print("Original String: \(originalString)")
print("Reversed String: \(reversedString)")

//fizzbuzz algorithm
//if n / 3 is fizz and n / 5 is buzz then if n 3 and 5 / both are fizzbuzz

func printFizzBuzzNumbers(n: Int){
```

```
    for i in 1...n{
        if i % 3 == 0 && i % 5 == 0{
            print("FIZZBUZZ")
        }else if i % 3 == 0{
            print("FIZZ")
        }else if i  % 5 == 0{
            print("BUZZ")
        }else {
            print(i)
        }
    }
}

printFizzBuzzNumbers(n: 20)


// merge two sorted array in swift
func mergeTwoSortedArray(arr1: [Int], arr2: [Int]) -> [Int]{
    let all = arr1 + arr2
    return all
}

print(mergeTwoSortedArray(arr1: [1,2,3,4,5,44], arr2: [6,7,8,9]))

//valid parentheses string in swift suppose ex "{()}" it is valid but "{(})" it is
wrong
func isValidParanthesis(str: String) -> Bool{
    var allbracket = [Character]()

    for i in str{
        if i == "(" || i == "{"  || i == "["{
            allbracket.append(i)
        }else{
            if allbracket.isEmpty {
                return false
            }
            let firstChar = allbracket.removeLast()
            print(firstChar)
            if (i == ")" && firstChar != "(") || (i == "}" && firstChar != "{") || (i == "]"
&&  firstChar != "["){
                return false
            }

        }
    }
    return true
}
```

```swift
print(isValidParanthesis(str: "(())"))

//
//given two string check is equal that string contains # bacspaces
func backSpaceCompare(a: String, b: String) -> Bool{
    func buildString(_ str: String) -> String{
        var result = [Character]()

        for i in str{
            if i == "#"{
                if !result.isEmpty{
                    result.removeLast()
                }
            }else{
                result.append(i)
            }
        }
        return String(result)
    }
    return buildString(a) == buildString(b)
}

print(backSpaceCompare(a: "a##c", b: "#a#c"))

//
func sortMaxElementToMin(arr: [Int]) -> [Int] {
    var array = arr
    let count = array.count

    for i in 0..<count {
        for j in (i + 1)..<count {
            if array[i] < array[j] {
                let temp = array[i]
                array[i] = array[j]
                array[j] = temp
            }
        }
    }
    return array
}


print(sortMaxElementToMin(arr: [5,3,6,88,2,1]))

//sum of didgits
func sumOfDigit(num: Int) -> Int{
```

```
    guard num >= 1 else { return 0}
    let strNum = String(num)
    var sum = 0
    for i in strNum{
        sum = sum + Int(String(i))!
    }
    return sum
}

print(sumOfDigit(num: 88))
/

//
Kadane algorithms largest sum of an array, //maximum product subarray
func largestSumOfAnArray(arr: [Int]) -> Int{
    var maxSum = 0
    var currentsum = 0

    for i in arr{
        currentsum = max(currentsum, currentsum + i)
        maxSum = max(maxSum, currentsum)
    }
    return maxSum
}

print(largestSumOfAnArray(arr: [-2,3,-4,-55,9,88,5]))
//

//
// move all zeros to end
func moveAllZerosToEnd(arr: [Int]) -> [Int]{
    var array = arr
    var j = 0

    for i in 0..<array.count{
        if array[i] != 0{
            var temp = array[i]
            array[i] = array[j]
            array[j] = temp
            j += 1

        }
    }
    return array
}

print(moveAllZerosToEnd(arr: [0,2,3,0,4,0,3,0,55,4]))
```

```swift
//

// move all zeros to start
func moveAllZerosToEnd(arr: [Int]) -> [Int]{
    var array = arr
    var j = 0

    for i in 0..<array.count{
        if array[i] == 0{
            var temp = array[i]
            array[i] = array[j]
            array[j] = temp
            j += 1

        }
    }
    return array
}

print(moveAllZerosToEnd(arr: [0,2,3,0,4,0,3,0,55,4]))
//
// sort an array 0s, 1s, 2s dutch national flag  problem
func sortZerosOnesTwos(arr: [Int]) -> [Int]{
    var newArray = arr
    var left = 0
    var mid = 0
    var right = newArray.count - 1

    for _ in 0..<newArray.count{
        guard newArray[mid] != 0 else{
            let temp = newArray[left]
            newArray[left] = newArray[mid]
            newArray[mid] = temp
            left += 1
            mid += 1
            continue
        }

        guard newArray[mid] != 1 else{
            mid += 1
            continue
        }

        if newArray[mid] == 2{
            let temp = newArray[right]
            newArray[right] = newArray[mid]
            newArray[mid] = temp
```

```
            right -= 1
        }
    }

    return newArray
}

print(sortZerosOnesTwos(arr: [1,2,0,1,0,2,0,1,0,2]))

//
func customMAx(a: Int, b: Int) -> Int{
    guard a > b else{
      return b
    }
    return a
}
//
func customContains(str: String, target: String) -> Bool{
    var isContains = false
    var temp = ""
    for i in str{
        temp += String(i)

        guard temp.count == target.count else{
            continue
        }
        if temp == target{
            isContains = true
            temp.remove(at: temp.startIndex)
            continue
        }
        temp.remove(at: temp.startIndex)
    }
    return isContains
}

print(customContains(str: "abababba", target: "abc"))
//
func reverseWordsStr(str: String) -> String{
    var reversedStr = ""
    var currentWord = ""
    var isAddedWords = false

    for i in str{
        if i == " "{
            if isAddedWords{
                reversedStr = currentWord + " " + reversedStr
```

```
            currentWord = ""
            isAddedWords = false
        }
    }else{
        currentWord += String(i)
        isAddedWords = true
    }
}

if isAddedWords{
    reversedStr = currentWord + " " + reversedStr
}
return reversedStr
}

print(reverseWordsStr(str: "swift i sa programming language"))

//

func longestPalindrom(text: String) -> String {
    let newArray = Array(text)
    guard newArray.count > 1 else { return text }
    var maxL = 1
    var start = 0
    var low = 0
    var high = 0
    for i in 0..<newArray.count {
     low = i - 1
     high = i + 1

        while high < newArray.count && newArray[high] == newArray[i] {
            high += 1
        }

        while low >= 0 && newArray[low] == newArray[i] {
            low -= 1
        }

        while high < newArray.count && low >= 0  && newArray[low] ==
newArray[high]  {
            low -= 1
            high += 1
        }

        let length = high - low - 1
        if maxL < length {
            maxL = length
```

```
            start = low + 1
        }
    }

    print(start)
    print(maxL)
    let str = String(newArray[start..<(start + maxL)])
    return str
}
//

//Elements print greater than the previous and next element in an Array
func printHigherelementFromPreviousAndNext(in array:[Int]) {
    guard array.count > 2 else {
         print("no element exist!!!")
        return
    }

    for i in 1..<(array.count - 1) {
        if array[i - 1] < array[i] && array[i] > array[i + 1] {
            print("\(array[i])")
        }
    }
}

printHigherelementFromPreviousAndNext(in: [2,3,4,23,6,6,5,8,7])
//
//given an array find if there is a triplet in the array which sum upto given
number
func tripletSum(array: [Int], target: Int) -> (Int, Int, Int)? {
    for i in 0..<(array.count - 2) {
        var set = Set<Int>()
        let d = target - array[i]
        for j in (i + 1)..<array.count {
            let r = d - array[j]

            if set.contains(r) {
                return (array[i], array[j], r)
            }
            set.insert(array[j])
        }
    }
    return nil
}

if let result = tripletSum(array: [12,3,4,1,6,9], target: 24){
    print(result)
```

```
}
//
//NGE algorithm next greater element in array
struct Stack {
    var items = [Int]()

    var isEmpty: Bool {
        return items.isEmpty
    }

    var top: Int? {
        return items.last
    }

    mutating func push(_ e:Int) {
        items.append(e)
    }
    mutating func pop() {
        items.removeLast()
    }


}

//NGE algorithm next greater element in array
func printNextGreater(in array:[Int]) {
    guard !array.isEmpty else { return }
    var s = Stack()
    s.push(array[0])

    for i in 1..<array.count {

        if s.isEmpty {
            s.push(array[i])
            continue
        }

        while let top = s.top, top < array[i] {
            print("\(top) ---> \(array[i])")
            s.pop()
        }

        s.push(array[i])
    }

    while !s.isEmpty {
        print("\(s.top ?? -9999) ---> -1 ")
```

```
            s.pop()
        }
    }

    printNextGreater(in: [3,2,5,44,3,7,8])
    //
    //given two unsorted array fing all pair from both arrays whose sum is equal to x
    func findPairWithSum(arr1: [Int], arr2: [Int], target: Int) -> [(Int, Int)]{
        var result = [(Int, Int)]()
        for i in arr2 {
            let complement = target - i
            if arr1.contains(complement) {
                result.append((complement,i))
            }
        }
        return result
    }

    print(findPairWithSum(arr1: [1,2,4,5,7], arr2: [5,6,3,4,8], target: 9))
    //
    func zerosReplaceIntofive(num: Int) -> Int{
        var temp = ""
        let strNum = String(num)
        for i in strNum{
            if i == "0"{
                temp.append("5")
            }else{
                temp.append(i)
            }
        }
        return Int(temp)!
    }

    print(zerosReplaceIntofive(num: 2005))
    //

    //
    func printDigits(n: Int) {
        var n = n

        while n != 0 {
            let d = n % 10
            print(d)
            n = n / 10
        }
    }
    printDigits(n: 125)
```

```swift
//
//reverse integers
func reverseInteger(n: Int) {
    var n = n
    var sum  = 0
    while n != 0 {
        let d = n % 10
        sum = sum * 10  + d
        print(d)
        n = n / 10
    }
    print(sum)
}
reverseInteger(n: 125)
//
//
//sumOfDigitsIntegers
func sumOfDigitsIntegers(n: Int) {
    var n = n
    var sum  = 0
    while n != 0 {
        let d = n % 10
        sum = sum + d
        print(d)
        n = n / 10
    }
    print(sum)
}
sumOfDigitsIntegers(n: 125)
//
//repleceZerosByGiven
func reverseDigitMethod(n: Int) -> Int {
    var n = n
    var reverse  = 0
    while n != 0 {
        let d = n % 10
        reverse = reverse * 10 + d
        n = n / 10
    }
    return reverse
}

func repleceZerosByGiven(n: Int, replace: Int) -> Int {
    var n = n
    var result = 0
    while n != 0 {
        let d = n % 10
```

```
        if d == 0 {
            result = result * 10 + replace
        }else{
            result = result * 10 + d
        }
        n = n / 10
    }
    return reverseDigitMethod(n: result)
}

print(repleceZerosByGiven(n: 2001, replace: 5))

//Reverse Int Array in swift
func reverseArray(arr: [Int]) -> [Int]{
    var reversed = ""
    var reversedArray = [Int]()
    for i in arr{
        reversed = String(i) + reversed
    }

    for i in reversed {
        reversedArray.append(Int(String(i))!)
    }

   return reversedArray
}

print(newArray)
print(reverseArray(arr: newArray))
//
//
func printAllDuplicateCharacters(in str: String) {
    var dic:[Character: Int] = [:]

    for i in str {
        if let count = dic[i], count > 0 {
            dic[i] = count + 1
        } else {
            dic[i] = 1
        }
    }
    print(dic)
    for i in str {
        if let count = dic[i], count >= 2 {
            print(i)
        }
    }
```

```
}

printAllDuplicateCharacters(in: "geeksforgeeks”
/////
Find second smallest value from array in swift
func findSecondSmallestValue(in array: [Int]) -> Int? {
    if array.count < 2 {
        return nil
    }

    var smallest = Int.max
    var secondSmallest = Int.max

    for number in array {
        if number < smallest {
            secondSmallest = smallest
            smallest = number
        } else if number < secondSmallest && number != smallest {
            secondSmallest = number
        }
    }

    return secondSmallest
}

// Example usage:
let numbers = [5, 3, 8, 2, 9, 1, 7]
if let secondSmallest = findSecondSmallestValue(in: numbers) {
    print("The second smallest value in the array is \(secondSmallest)")
} else {
    print("The array does not have a second smallest value.")
}
//
func reverseInts(arr: [Int]) -> [Int]{
    var newArray = arr
    let l = arr.count / 2
    var j = arr.count
    for i in 0..<l {
        let temp = newArray[i]
        newArray[i] = newArray[j - 1]
        newArray[j - 1] = temp
        j -= 1
    }
    return newArray
}

print(reverseInts(arr: [77,1,2,44,88,8,44]))
```

```swift
//
//Common characters in n strings
func commonCharactersInArrayOfString(in str: [String]) -> [Character] {
    var resultDict = [Character:Int]()
    var resultArr = [Character]()
    for word in str {
        for char in word {
            resultDict[char, default: 0] += 1
        }
    }


    for (char, count) in resultDict {
        if count >= str.count {
            resultArr.append(char)
        }
    }
    return resultArr
}

var charsArray = ["geeksforgeeks", "gemkstones", "acknowledges",
"aguelikes"]
print(commonCharactersInArrayOfString(in:charsArray))
//
//Check if given String is Pangram or not pangram means that string hold
character a to z
func isPangram(string: String) -> Bool {

    var set = Set<Character>()
    for i in string {
        if i >= "a" && i <= "z" {
            set.insert(i)
        }

        if i >= "A" && i <= "Z" {
            set.insert(Character(String(i).lowercased()))
        }
    }

    guard set.count == 26 else { return false }
    return true
}

print(isPangram(string: "abcdefghijklmnopqrstuvwxyz i am risalat"))
//
//Determine if a string has all Unique Characters
func isAllUniqueCharacters(in str: String) -> Bool {
```

```
    var dic:[Character: Int] = [:]

    for i in str {
        dic[i, default: 0] += 1
    }

    for i in str {
        if let count = dic[i], count > 1 {
            return false
        }
    }
    return true
}

print(isAllUniqueCharacters(in: "abcd10jk"))
//
//Remove duplicates from a string in O(1) extra space
func removeDuplicateCharacters(in str: String) -> String {
    var ans = ""
    for i in str {
        if !ans.contains(i) {
            ans.append(i)
        }
    }
return ans

}

print(removeDuplicateCharacters(in: "geeksforgeeks"))
//
// print non duplicate character in string swift
func printNonDuplicateCharacters(in str: String) {
    var dic:[Character: Int] = [:]

    for i in str {
        dic[i, default: 0] += 1
    }

    for i in str {
        if let count = dic[i], count == 1 {
            print(i)
        }
    }
}

printNonDuplicateCharacters(in: "geeksforgeeks")
//
```

```swift
//Print all the duplicate characters in a string
func printAllDuplicateCharacters(in str: String) {
    var dic:[Character: Int] = [:]

    for i in str {
        if let count = dic[i], count > 0 {
            dic[i] = count + 1
        } else {
            dic[i] = 1
        }
    }
    for i in str {
        if let count = dic[i], count >= 2 {
            print(i)
        }
    }
}

printAllDuplicateCharacters(in: "geeksforgeeks")
//
```