# Ionic 2

**https://www.youtube.com/watch?v=T9XoKE8w0dI**

<https://www.youtube.com/watch?v=qdeD9rYlBU4>

<http://vpt-deeplearner.tech/2017/01/21/ionic-2-an-example-of-speech-recognition-on-mobile-phones/>

<https://github.com/vijtad/SpeechRecognizer/tree/master/src/ios>

<https://www.youtube.com/watch?v=0jamhGf-8ww>

<https://www.youtube.com/watch?v=Ds5tm-6Nc9g>

<https://ionicframework.com/docs/native/text-to-speech/>

<https://ionicframework.com/docs/native/speech-recognition/>

<https://ionicframework.com/docs/intro/deploying/>

# <https://forum.ionicframework.com/t/ionic-toturial-for-building-a-release-apk/15758>

# ARC And Retain Cycle <https://youtu.be/VcoZJ88d-vM>

**AWS: https://youtu.be/i8GZod5Dkt8**

## This is Apple’s way of handling memory management.

Swift uses *Automatic Reference Counting* (ARC) to track and manage your app’s memory usage

ARC automatically frees up the memory used by class instances when those instances are no longer needed.

Reference counting only applies to instances of classes. Structures and enumerations are value types, not reference types, and are not stored and passed by reference.

How ARC Works

Every time you create a new instance of a class, ARC allocates a chunk of memory to store information about that instance. This memory holds information about the type of the instance, together with the values of any stored properties associated with that instance.

Additionally, when an instance is no longer needed, ARC frees up the memory used by that instance so that the memory can be used for other purposes instead. This ensures that class instances do not take up space in memory when they are no longer needed.

However, if ARC were to deallocate an instance that was still in use, it would no longer be possible to access that instance’s properties, or call that instance’s methods. Indeed, if you tried to access the instance, your app would most likely crash.

To make sure that instances don’t disappear while they are still needed, ARC tracks how many properties, constants, and variables are currently referring to each class instance. ARC will not deallocate an instance as long as at least one active reference to that instance still exists.

To make this possible, whenever you assign a class instance to a property, constant, or variable, that property, constant, or variable makes a *strong reference* to the instance. The reference is called a “strong” reference because it keeps a firm hold on that instance, and does not allow it to be deallocated for as long as that strong reference remains.

ARC automatically frees up the memory used by class instances when those instances are no longer needed.

For each object it keeps count of how many strong references are pointing to that object

e.g. We have Person class,Phone class and Car class And Phone and Car classes have strong references to Person class

Now if I assign Person=nil to deallocate Person from memory ARC will not allow to do so as Phone and Car classes have strong references to it.

To fix this problem we have to make them weak references

We will create two classes

1]Person 2]Phone

import Foundation

class Person{

let name:String

var Phone:Phone?

init(name:String,phone:Phone) {

self.name = name

self.Phone = phone

}

deinit{

print("\(name) is being deinitialized")

}

}

class Phone{

let name:String

var owner:Person?

init(name:String,owner:Person) {

self.name = name

self.owner = owner

}

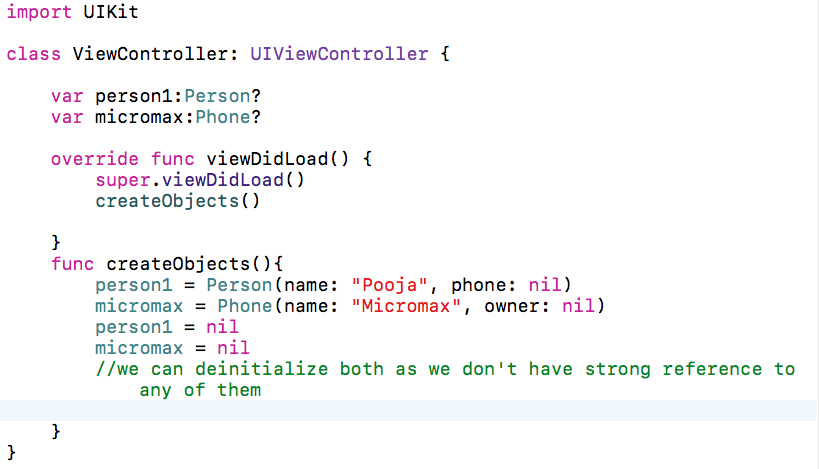
deinit {

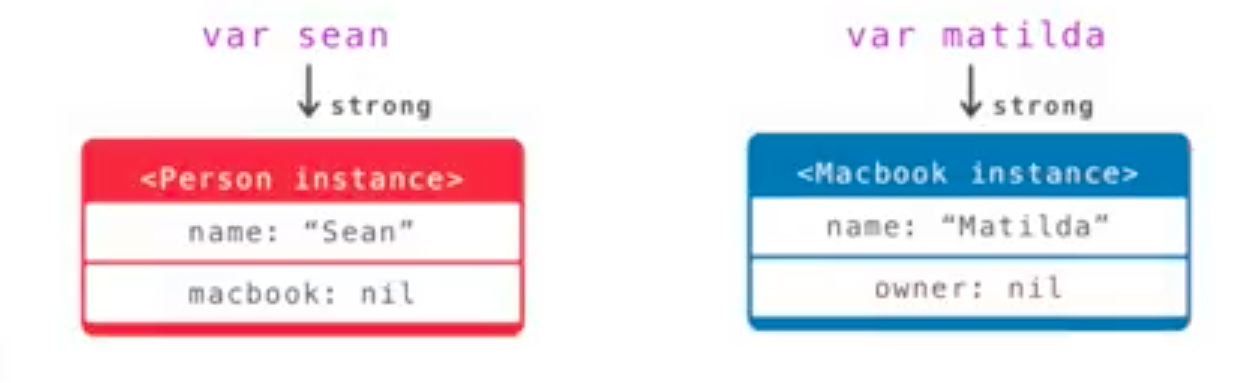
print("Phone named \(name) is being deinitialized")

}

}

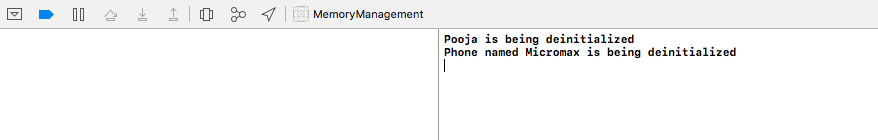
**ViewController.swift**



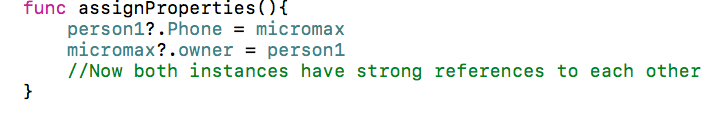


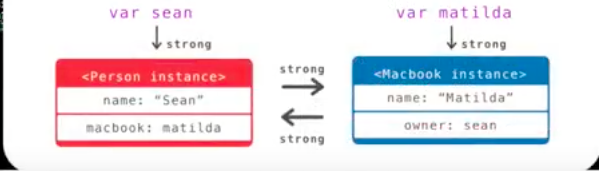
As there are no strong references to both Person and Phone they can be deinitialized.

OutPut:-

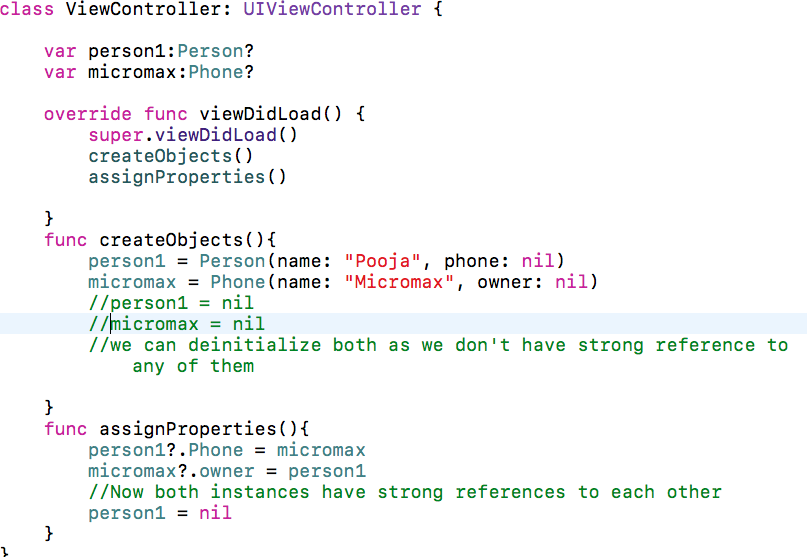


Now will create strong references for both the instances.



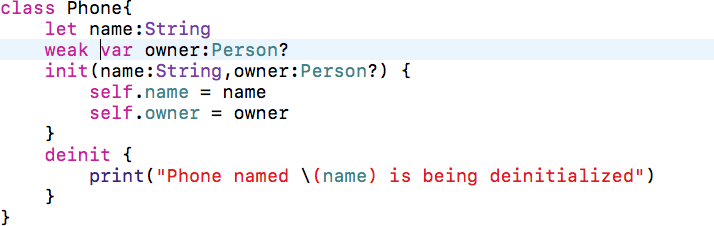


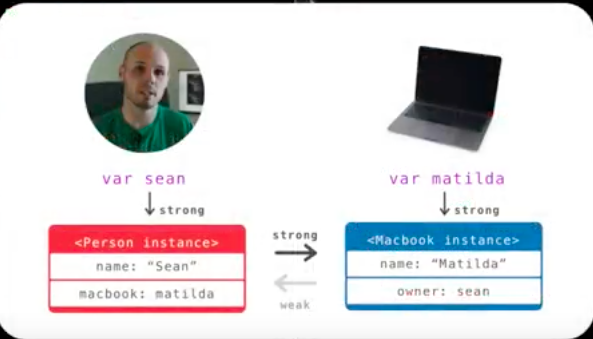
Now if we set nil for person1 instance it will not get deallocated from memory as Automatic reference count for that instance is 1.



Now deinit methods for person1 and micromax will not be called

To solve this issue will make owner in phone as weak variable





Now if we run the program it will call deinit method for person1

And micromax.owner will be nil

If two objects have strong references to each other and it becomes like never ending loop then it is called Retain cycle or memory leak.

To solve this we can make one of the reference as weak.

**Memory leaks in Closures?**

[**https://medium.com/@streem/understanding-memory-leaks-in-closures-48207214cba**](https://medium.com/@streem/understanding-memory-leaks-in-closures-48207214cba)

**Communication Patterns**

**1]Delegates**

**2]Notifications**

**Delegates/Protocol**

Delegates and protocols is 1-to-1 communication pattern.

Selection Screen(on Right) is boss – Knows all the information about what needs to be done.

This class knows what option user has selected and it passes all this info to the intern

Base Screen(on left) is Intern - does not know what to do and it only waits for orders from boss.

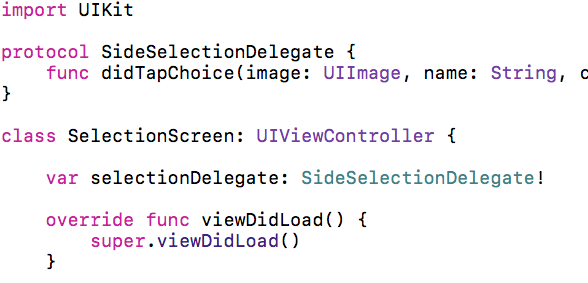
Intern gets info from boss and acts accordingly

To pass the info will first create Protocol in Boss i.e. SelectionScreen

protocol SideSelectionDelegate{

func didTap()

}



Protocol will contain just the declaration of functions i.e. list of orders for intern

Then will declare a variable in SelectionScreen i.e. Boss to hold that delegate

This delegate variable will be our intern

Now on some button clicks we our intern to do something

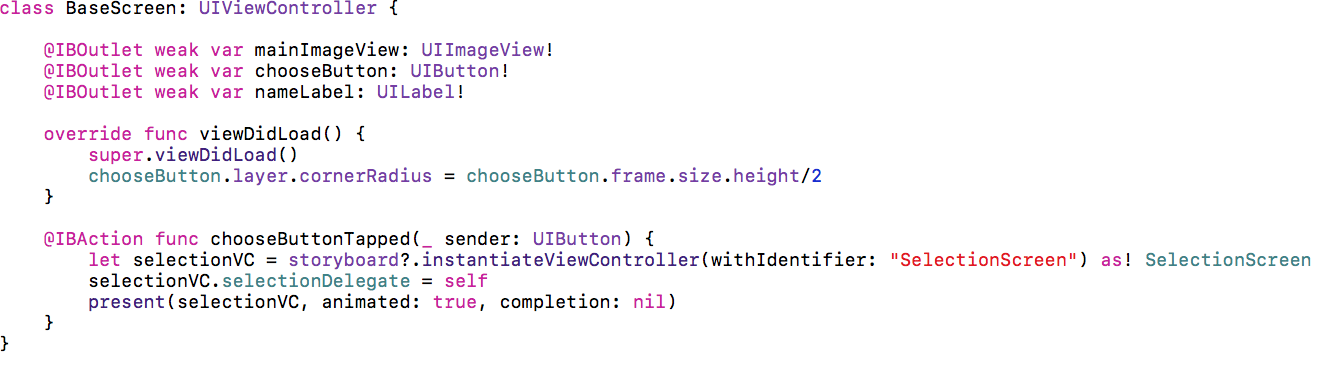


Will call didTap() on button click with delegate variable

Here the Boss i.e. selection screen knows what image to send what background color to set etc.

The Base screen i.e. intern will just get the information passed by the Boss i.e. Selection Screen and display it

Now in the BaseScreen.swift we have to set the delegate variable

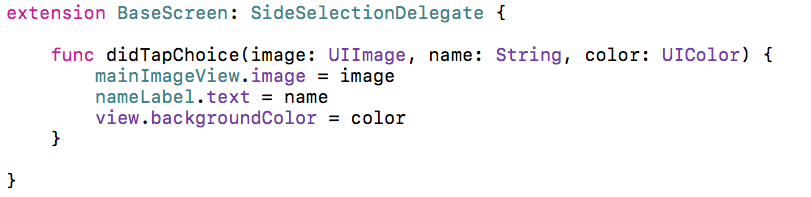


In the above code self means base screen i.e. we are saying that SelectionVC i.e. SelectionSCreen(Boss) have variable called selectionDelegate which is an intern of SelectionSCreen and Base screen is saying that it would like to be an intern of SelectionScreen i.e.Boss

Whenever you have have delegate, your intern has to conform to that delegate

We have to define didTap() here in the BaseSCreen(intern)

This function will get called whenever will click buttons in the SelectionSCreen(Boss).We don’t have to call it in the BaseScreen(intern)



**Notification And Observers**

This is one-to-many communication pattern

**View LifeCycle**

viewDidLoad(),viewWillAppear(),viewWillLayoutSubviews(),viewDidLayoutSubviews(),viewDidAppear() these are called while creating the view

And getting rid of the view is viewWillDisappear() and viewDidDisappear()

viewDidLoad() gets called very first time view is loaded into memory and it only gets called once

viewWillAppear() gets called every time the view appears.If you want to do something whenever view appears such as animation you can do it in viewWillAppear()

viewWillLayoutSubviews() and viewDidLayoutSubviews() these are called when the view is actually laying out all the subviews the constraints,the sizing and everything

viewDidAppear() meand the view is completely loaded

viewWillDisappear()

viewDidDisappear()

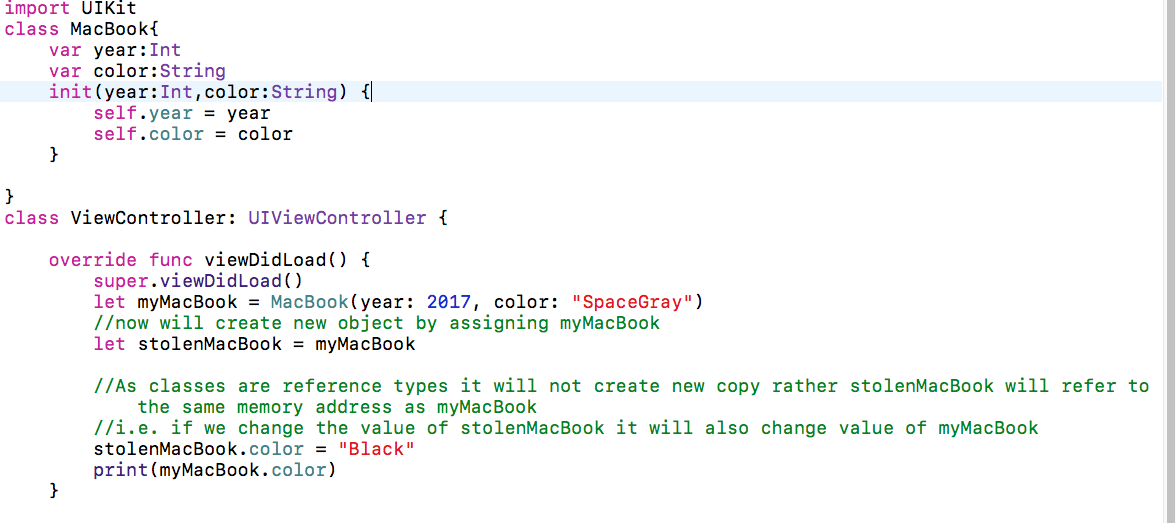
Which Apple Framework you like and Why?

**Classes Vs Structs**

Classes are reference types and structs are value types

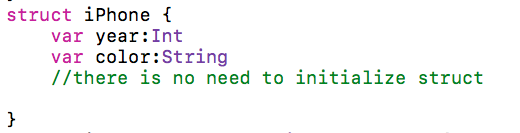
Value types of swift are passed around by copies

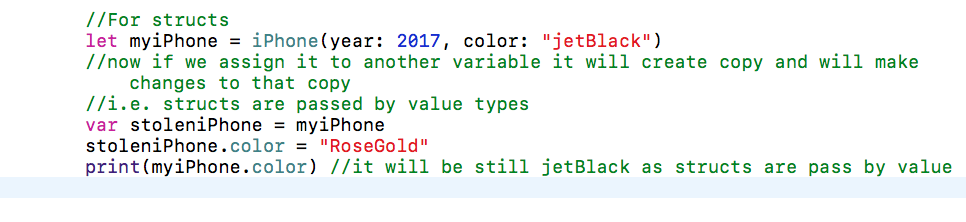
Reference types of swift are passed around by original copies



1]Classes are reference types

Now MacBook.color will be black





2]Structs are pass by value

**Atomic vs NonAutomic**

atomic

Properties are atomic by default, so if you don’t write nonatomic, it will be atomic (whether you write it or not).  Atomic basically ensures that data is written or read atomically.  So if thread A is still in the getter when thread B calls the setter, thread A will get a viable value.  It is not necessarily thread-safe, but much safer than nonatomic.  The problem with atomic though, is that it is quite slow compared to its opposite.  Also, you must either implement both the setter and the getter, or neither. You cannot have a custom getter and a synthesized setter in an atomic property.

nonatomic

This makes no such guarantees about atomicity (which is quite a cool word) as nonatomic.  If thread A is in the middle of a getter for a nonatomic NSString, and Thread B tries to set it to “Microwave”, and Thread C tries to set it to “Refrigerator”, you might get “Microgerator”, or it may just be completely unreadable and crash the program.  You never know, so if you use nonatomic, you must implement your own thread safety and atomicity.  You will more often use nonatomic properties though, because they are FAST when compared to atomic ones.

**Filter,Map And Reduce**

**Filter**

**Use filter to loop over a collection and return an Array containing only those elements that match an include condition.**

These functions iterate over collection e.g array and make changes into it and creates new array

Let iPhones = myArray.filter({return $0.type == “iPhone”})

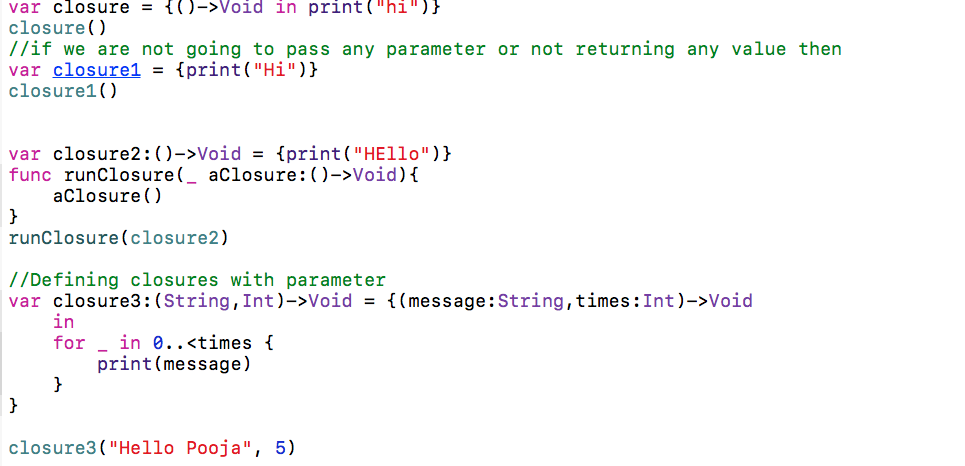
Here $0 is just the placeholder for the object that it’s iterating over

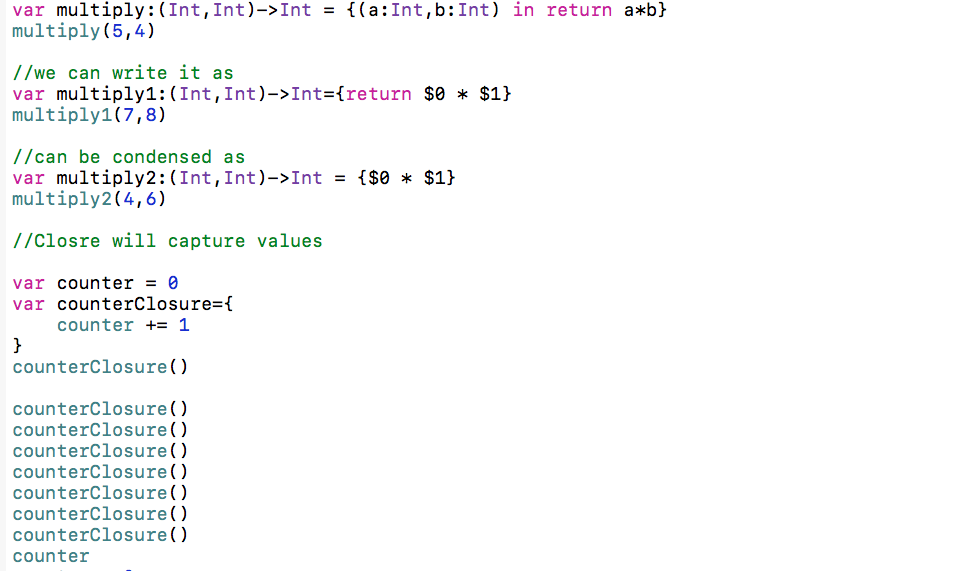
filter has closure inside

**Closure**

Closures are small blocks of code much like functions,but without names

https://youtu.be/fVF\_tNcIhfc





**Memory Leak:**

Memory leaks are blocks of allocated memory that the program no longer references.

"Zombies" in Objective-C parlance are the opposite of leaks. A leak is a bit of allocated memory that you no longer have any references to, so you can't free it. A zombie is an object that *has* been deallocated, but references to it still exist and messages are still being sent to it (which can lead to all sorts of unpredictable behavior).

http://basememara.com/memory-leaks-resource-management-swift-ios/

**map()**

**Use map to loop over a collection and apply the same operation to each element in the collection.** The map function returns an array containing the results of applying a mapping or transform function to each item:

Returns an array containing the results of mapping the given closure over the sequence’s elements.

*et cast = ["Vivien", "Marlon", "Kim", "Karl"]*

*let lowercaseNames = cast.map { $0.lowercaseString }*

*// 'lowercaseNames' == ["vivien", "marlon", "kim", "karl"]*

*let letterCounts = cast.map { $0.count }*

*// 'letterCounts' == [6, 6, 3, 4]*

**Concurrency**

**https://videos.raywenderlich.com/courses/55-ios-concurrency-with-gcd-and-operations/lessons/1**

More than one task running at same time

GCD is used to do simple jobs and Operations is used to do complex jobs

In GCD we r working with functions and Operations are objects they encapsulates data and functionality

Tasks run on Threads.

UI runs on main thread

We create units of work,tasks or operations and run them off the main thread on dispatch or operation queues

Concurrency Problems

1]Race Condition

2]Priority inversion

3]Deadlock

Race condition happen when two threads try to access same value

**XCode 8 introduces ThreadSanitizer tool TSan which helps to find potential race condition in code**

**Priority inversion means low-priority task preempts a higher-priority**

**Deadlock is where two threads are each waiting for other to release a shared resource**

**Concurrent is not same as asynchronous**

**Apple creating operation property isAsynchronous and deprecating isConcurrent**

**If we run slow task asynchronously on another queue the main queue can continue immediately**

**GCD queues manage threads**

**We can run task asynchronously on serial or concurrent queue**

**Synchronous vs Asynchronous tells you whether current queue has to wait for the task to complete**

**Serial vs concurrent tell you whether a queue has one or many threads**

**Coding Challege on HackerRank**

[Pushpanjaliam55@gmail.com](mailto:Pushpanjaliam55@gmail.com)

pushpanjalimane

abhijeet25190

**1]Input Format**

Code that reads input from stdin is provided for you in the editor. There are lines of input, and each line contains a single integer.

**Output Format**

Code that prints the sum calculated and returned by *solveMeFirst* is provided for you in the editor.

// read integers line by line

var a = Int(readLine()!)!

var b = Int(readLine()!)!

// Hint: Type print(a + b) below

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

return a+b

}

let sum = solveMeFirst(a:a,b:b)

print(sum)

Higher order Functions

Reduce

The reduce method solves the problem of combining the elements of an array to a single value

var sum = 0

for money in moneyArray {

sum = sum + money

}

Reduce is a method that let’s us quickly define this kind of operations by specifying an initial value and a method of combining elements.

To compute the sum of our array we can use:

sum = moneyArray.reduce(0,{$0 + $1})

We can take advantage of the fact that operators are methods in Swift and make using reduce even more convenient

sum = moneyArray.reduce(0,+)

**2]Simple Array Sum**

Given an array of integers, can you find the sum of its elements?

**Input Format**

The first line contains an integer, , denoting the size of the array.

The second line contains space-separated integers representing the array's elements.

**Output Format**

Print the sum of the array's elements as a single integer.

**Sample Input**

6

1 2 3 4 10 11

**Sample Output**

31

import Foundation

// number of elements

let n = Int(readLine()!)!

// read array and map the elements to integer

let arr = readLine()!.components(separatedBy: " ").map{ Int($0)! }

// find and print the sum of array

let sum = arr.reduce(0,+)

print(sum)

Using the Filter Function

**The** filter **function loops over every item in a collection, and returns a collection containing only items that satisfy an include condition.** It’s like applying an if-statement to a collection, and only getting the ones that pass the condition back.

let values = [11, 13, 14, 17, 21, 33, 22]

let even = values.filter { $0 % 2 == 0 }

print(even)

<https://learnappmaking.com/map-reduce-filter-swift-programming/>

**zip()**

Creates a sequence of pairs built out of two underlying sequences.

let arr1 = readLine()!.components(separatedBy: " ").map{ Int($0)! }

let arr2 = readLine()!.components(separatedBy: " ").map{ Int($0)! }

print(zip(arr1, arr2))

input : 5 3 4

2 5 7

output: Zip2Sequence<Array<Int>, Array<Int>>(\_sequence1: [5, 3, 5], \_sequence2: [2, 5, 7])

**filter()**

let arr1 = readLine()!.components(separatedBy: " ").map{ Int($0)! }

let arr2 = readLine()!.components(separatedBy: " ").map{ Int($0)! }

print(zip(arr1, arr2).filter(>))

output

[(5, 2)] returns pair having arr1 greater than arr2

**Compare The Triples**

Alice and Bob each created one problem for HackerRank. A reviewer rates the two challenges, awarding points on a scale from to for three categories: *problem clarity*, *originality*, and *difficulty*.

We define the rating for Alice's challenge to be the triplet , and the rating for Bob's challenge to be the triplet .

Your task is to find their *comparison points* by comparing with , with , and with .

* If , then Alice is awarded point.
* If , then Bob is awarded point.
* If , then neither person receives a point.

Comparison points is the total points a person earned.

Given and , can you compare the two challenges and print their respective comparison points?

**Input Format**

The first line contains space-separated integers, , , and , describing the respective values in triplet .

The second line contains space-separated integers, , , and , describing the respective values in triplet .

**Constraints**

**Output Format**

Print two space-separated integers denoting the respective comparison points earned by Alice and Bob.

As we want to count score for Alice and Bob will use three function

1]zip():- To zip the two arrays

2]filter():**The** filter **function loops over every item in a collection, and returns a collection containing only items that satisfy an include condition.**

**3**]count()**To count how many elements in array**

import Foundation

// Enter your code here

let arr1 = readLine()!.components(separatedBy: " ").map{ Int($0)! }

let arr2 = readLine()!.components(separatedBy: " ").map{ Int($0)! }

let AliceScore = zip(arr1, arr2).filter(>).count

let BobScore = zip(arr1, arr2).filter(<).count

print("\(AliceScore) \(BobScore)")

**A Very Big Sum**

In case of swift it will automatically convert result in long

You are given an array of integers of size . You need to print the sum of the elements in the array, keeping in mind that some of those integers may be quite large.

**Input Format**

The first line of the input consists of an integer . The next line contains space-separated integers contained in the array.

**Output Format**

Print a single value equal to the sum of the elements in the array.

**Constraints**

**Sample Input**

5

1000000001 1000000002 1000000003 1000000004 1000000005

**Output**

5000000015

**Note:**

The range of the 32-bit integer is .

When we add several integer values, the resulting sum might exceed the above range. You might need to use long long int in C/C++ or long data type in Java to store such sums.

So it will be same as sum of array elements just use arr.reduce(0,+)

**2d Array in Swift**

<https://stackoverflow.com/questions/24051490/multidimensional-arrays-in-swift>

<https://www.dotnetperls.com/2d-swift>

**Diagonal Difference**

Given a square matrix of size , calculate the absolute difference between the sums of its diagonals.

**Input Format**

The first line contains a single integer, . The next lines denote the matrix's rows, with each line containing space-separated integers describing the columns.

**Constraints**

**Output Format**

Print the absolute difference between the two sums of the matrix's diagonals as a single integer.

**Sample Input**

3

11 2 4

4 5 6

10 8 -12

**Sample Output**

15

**Explanation**

The primary diagonal is:

11

5

-12

Sum across the primary diagonal: 11 + 5 - 12 = 4

The secondary diagonal is:

4

5

10

Sum across the secondary diagonal: 4 + 5 + 10 = 19

Difference: |4 - 19| = 15

**Note:** |x| is [absolute value](https://www.hackerrank.com/external_redirect?to=https://www.mathsisfun.com/numbers/absolute-value.html) function

import Foundation

// read the integer n

let n = Int(readLine()!)!

// declare 2d array

var arr: [[Int]] = []

var primaryDiagonalSum = 0

var secondaryDiagonalSum = 0

// read array row-by-row

for \_ in 0..<n {

arr.append(readLine()!.components(separatedBy: " ").map{ Int($0)! })

}

for i in 0..<n{

primaryDiagonalSum += arr[i][i]

secondaryDiagonalSum += arr[i][n-i-1]

}

let difference = abs(primaryDiagonalSum - secondaryDiagonalSum)

print(difference)

**Where Keyword**

<http://blog.krzyzanowskim.com/2015/11/13/where-where-may-be-used/>

**PlusMinus**

Given an array of integers, calculate which fraction of its elements are *positive*, which fraction of its elements are *negative*, and which fraction of its elements are *zeroes*, respectively. Print the decimal value of each fraction on a new line.

**Note:** This challenge introduces precision problems. The test cases are scaled to six decimal places, though answers with absolute error of up to are acceptable.

**Input Format**

The first line contains an integer, , denoting the size of the array.

The second line contains space-separated integers describing an array of numbers .

**Output Format**

You must print the following lines:

* A decimal representing of the fraction of *positive* numbers in the array compared to its size.
* A decimal representing of the fraction of *negative* numbers in the array compared to its size.
* A decimal representing of the fraction of *zeroes* in the array compared to its size.

**Sample Input**

6

-4 3 -9 0 4 1

**Sample Output**

0.500000

0.333333

0.166667

**Explanation**

There are positive numbers, negative numbers, and zero in the array.

The respective fractions of positive numbers, negative numbers and zeroes are , and , respectively.

import Foundation

// number of elements

let n = Int(readLine()!)!

// read array and map the elements to integer

let arr = readLine()!.components(separatedBy: " ").map{ Int($0)! }

func getFraction(for someOperator:Character) -> Double{

switch someOperator{

case ">": return Double(arr.filter{$0>0}.count)/Double(arr.count)

case "<": return Double(arr.filter{$0<0}.count)/Double(arr.count)

case "=": return Double(arr.filter{$0==0}.count)/Double(arr.count)

default: return 0.0

}

}

let positiveFraction = getFraction(for:">")

let negativeFraction = getFraction(for:"<")

let zeroFraction = getFraction(for:"=")

print(positiveFraction, negativeFraction, zeroFraction, separator: "\n")

**Staircase**

Consider a staircase of size :

#

##

###

####

Observe that its base and height are both equal to , and the image is drawn using # symbols and spaces. *The last line is not preceded by any spaces.*

Write a program that prints a staircase of size .

**Input Format**

A single integer, , denoting the size of the staircase.

**Output Format**

Print a staircase of size using # symbols and spaces.

**Note**: The last line must have spaces in it.

import Foundation

// read the integer n

let n = Int(readLine()!)!

// print the staircase

var hash="#"

for i in 0..<n {

let spaces = String(repeating: " ", count: n-i-1)

print(spaces + hash)

hash += "#"

}

**Mini-Max Sum**

Given five positive integers, find the minimum and maximum values that can be calculated by summing exactly four of the five integers. Then print the respective minimum and maximum values as a single line of two space-separated long integers.

**Input Format**

A single line of five space-separated integers.

**Constraints**

* Each integer is in the inclusive range .

**Output Format**

Print two space-separated long integers denoting the respective minimum and maximum values that can be calculated by summing exactly *four* of the five integers. (The output can be greater than 32 bit integer.)

**Sample Input**

1 2 3 4 5

**Sample Output**

10 14

**Explanation**

Our initial numbers are , , , , and . We can calculate the following sums using four of the five integers:

* If we sum everything except , our sum is .
* If we sum everything except , our sum is .
* If we sum everything except , our sum is .
* If we sum everything except , our sum is .
* If we sum everything except , our sum is .

As you can see, the minimal sum is and the maximal sum is . Thus, we print these minimal and maximal sums as two space-separated integers on a new line.

**Hints:** Beware of integer overflow! Use 64-bit Integer.

import Foundation

// Enter your code here

let arr = readLine()!.components(separatedBy: " ").map{ Int($0)! }

let sorted = arr.sorted()

let minimumSum = sorted.dropLast().reduce(0, +)

let maximumSum = sorted.dropFirst().reduce(0, +)

print("\(minimumSum) \(maximumSum)")

**BirthdayCakeCandles**

Colleen is turning years old! Therefore, she has candles of various heights on her cake, and candle has height . Because the taller candles tower over the shorter ones, Colleen can only blow out the tallest candles.

Given the for each individual candle, find and print the number of candles she can successfully blow out.

**Input Format**

The first line contains a single integer, , denoting the number of candles on the cake.

The second line contains space-separated integers, where each integer describes the height of candle .

**Constraints**

**Output Format**

Print the number of candles Colleen blows out on a new line.

**Sample Input 0**

4

3 2 1 3

**Sample Output 0**

2

import Foundation

let n = Int(readLine()!)!

let arr = readLine()!.components(separatedBy:" ").map{Int($0)!}

let maxHeight = arr.max()

let noOfCandlesBlown = arr.filter{$0 == maxHeight}.count

print(noOfCandlesBlown)

**Time Conversion**

Given a time in [-hour AM/PM format](https://www.hackerrank.com/external_redirect?to=https://en.wikipedia.org/wiki/12-hour_clock), convert it to military (-hour) time.

**Note:** Midnight is on a -hour clock, and on a -hour clock. Noon is on a -hour clock, and on a -hour clock.

**Input Format**

A single string containing a time in -hour clock format (i.e.: or ), where and .

**Output Format**

Convert and print the given time in -hour format, where .

**Sample Input**

07:05:45PM

**Sample Output**

19:05:45

var time = readLine()!

var arr = time.components(separatedBy: ":")

var hr:Int = Int(arr[0])!

let endIndex = arr[2].index(arr[2].endIndex, offsetBy: -2)

let ampm = arr[2].substring(from:endIndex)

arr[2] = arr[2].substring(to: endIndex)

if (ampm == "PM" && hr != 12) {

hr+=12

arr[0] = String(hr)

time = arr[0] + ":" + arr[1] + ":" + arr[2]

}

else if(ampm == "AM" && hr == 12){

arr[0]="00"

time = arr[0] + ":" + arr[1] + ":" + arr[2]

}

else{

time = time.substring(to: time.index(time.endIndex, offsetBy: -2))

}

print(time)

**Grading Students**

HackerLand University has the following grading policy:

* Every student receives a in the inclusive range from to .
* Any less than is a failing grade.

Sam is a professor at the university and likes to round each student's according to these rules:

* If the difference between the and the next multiple of is less than , round up to the next multiple of .
* If the value of is less than , no rounding occurs as the result will still be a failing grade.

For example, will be rounded to but will not be rounded because the rounding would result in a number that is less than .

Given the initial value of for each of Sam's students, write code to automate the rounding process. For each , round it according to the rules above and print the result on a new line.

**Input Format**

The first line contains a single integer denoting (the number of students).

Each line of the subsequent lines contains a single integer, , denoting student 's grade.

**Constraints**

**Output Format**

For each of the grades, print the rounded grade on a new line.

**Sample Input 0**

4

73

67

38

33

**Sample Output 0**

75

67

40

33

import Foundation

// Enter your code here

let n = Int(readLine()!)!

for i in 0..<n{

let grade = Int(readLine()!)!

let modifiedGrade = (grade < 38) || (grade % 5 < 3) ? grade : grade+5-grade%5

print(modifiedGrade)

}

**Apple and Orange**

Sam's house has an apple tree and an orange tree that yield an abundance of fruit. In the diagram below, the red region denotes his house, where is the start point and is the end point. The apple tree is to the left of his house, and the orange tree is to its right. You can assume the trees are located on a single point, where the apple tree is at point and the orange tree is at point .



When a fruit falls from its tree, it lands units of distance from its tree of origin along the -axis. A negative value of means the fruit fell units to the tree's left, and a positive value of means it falls units to the tree's right.

Given the value of for apples and oranges, can you determine how many apples and oranges will fall on Sam's house (i.e., in the inclusive range )? Print the number of apples that fall on Sam's house as your first line of output, then print the number of oranges that fall on Sam's house as your second line of output.

**Input Format**

The first line contains two space-separated integers denoting the respective values of and .

The second line contains two space-separated integers denoting the respective values of and .

The third line contains two space-separated integers denoting the respective values of and .

The fourth line contains space-separated integers denoting the respective distances that each apple falls from point .

The fifth line contains space-separated integers denoting the respective distances that each orange falls from point .

**Constraints**

**Output Format**

Print two lines of output:

* On the first line, print the number of apples that fall on Sam's house.
* On the second line, print the number of oranges that fall on Sam's house.

**Sample Input 0**

7 11

5 15

3 2

-2 2 1

5 -6

**Sample Output 0**

1

1

import Foundation

// Enter your code here

var arr = readLine()!.components(separatedBy:" ").map{Int($0)!}

var s = Int(arr[0])

var t = Int(arr[1])

arr = readLine()!.components(separatedBy:" ").map{Int($0)!}

var a = Int(arr[0])

var b = Int(arr[1])

arr = readLine()!.components(separatedBy:" ").map{Int($0)!}

var m = Int(arr[0])

var n = Int(arr[1])

var appleDistances = readLine()!.components(separatedBy:" ").map{Int($0)!}

var orangeDistances = readLine()!.components(separatedBy:" ").map{Int($0)!}

var noOfApple = appleDistances.filter{s...t ~= a + $0}.count

var noOfOrange = orangeDistances.filter{s...t ~= b + $0}.count

print(noOfApple)

print(noOfOrange)

**Kangaroo**

There are two kangaroos on a number line ready to jump in the positive direction (i.e, toward positive infinity). The first kangaroo starts at location and moves at a rate of meters per jump. The second kangaroo starts at location and moves at a rate of meters per jump. Given the starting locations and movement rates for each kangaroo, can you determine if they'll ever land *at the same location at the same time*?

**Input Format**

A single line of four space-separated integers denoting the respective values of , , , and .

**Constraints**

**Output Format**

Print YES if they can land on the same location at the same time; otherwise, print NO.

**Note:** The two kangaroos must land at the same location *after making the same number of jumps*.

**Sample Input 0**

0 3 4 2

**Sample Output 0**

YES

**Explanation 0**

The two kangaroos jump through the following sequence of locations:

Thus, the kangaroos meet after jumps and we print *YES*.

**Sample Input 1**

0 2 5 3

**Sample Output 1**

NO

x1+y\*v1 = x2+y\*v2

=>(x1-x2)+y(v1-v2) = 0

=>(x1-x2) = -y(v1-v2) Removing the '-ve' sign from RHS =>(x2-x1) = y(v1-v2)

=>(x2-x1)/(v1-v2) = y ----(1)

If you multiply -1 to both the numerator and denominator, then

=>(x1-x2)/(v2-v1) = y ----(2)

Thus equation (1) and (2) are the same.

**Explanation 1**

The second kangaroo has a starting location that is ahead (further to the right) of the first kangaroo's starting location (i.e., ). Because the second kangaroo moves at a faster rate (meaning ) *and* is already ahead of the first kangaroo, the first kangaroo will never be able to catch up. Thus, we print *NO*.

import Foundation

// Enter your code here

let arr = readLine()!.components(separatedBy:" ").map{Int($0)!}

let x1 = Int(arr[0])

let v1 = Int(arr[1])

let x2 = Int(arr[2])

let v2 = Int(arr[3])

//as x2> x1 then if v2> v1 or v2==v1 then they will not meet

if(v2>v1 || v2 == v1){

print("NO")

}

else if((x1-x2)%(v2-v1) == 0){

print("YES")

}

else{

print("NO")

}

**GCD**

The *greatest common divisor* (or Greatest Common Factor) of two numbers a and b is the largest positive integer that divides both a and b without a remainder.

For example, gcd(39, 52) = 13 because 13 divides 39 (39/13 = 3) as well as 52 (52/13 = 4). But there is no larger number than 13 that divides them both.

**New In iOS 11**

Drag and Drop

With the power of Multi-Touch and iOS 11, users can quickly move text, images, and files from one app to another. Support Drag and Drop in your apps to let users move content in a way that feels natural.