GTECH IOS QUESTION AND ANSWERS

Day-2

Example1. Difference between let and var ?
-Both let & var are used to declare a variable, Let is used for fixed values & var is used for changable value
-Both let and var are for creating variables in Swift. let helps you create immutable variables (constants) while on the other hand var creates mutable variables. The difference between them is that when you create a constant using let you have to assign something to it before the first use and can't reassign it

Example2. What is the difference between below declaration
var a: Int = 100
In above values the data type is mentioned
var a = 10
In above value the data type has not mentioned.

GTECH IOS QUESTION AND ANSWERS

1. What is array and write syntax in swift
-An ordered, random-access collection.
-Arrays are one of the most commonly used data types in an app. You use arrays to organize your app’s data. Specifically, you use the Array type to hold elements of a single type, the array’s Element type. An array can store any kind of elements—from integers to strings to classes.
-Swift makes it easy to create arrays in your code using an array literal: simply surround a comma-separated list of values with square brackets. Without any other information, Swift creates an array that includes the specified values, automatically inferring the array’s Element type.
For example:

```
// An array of 'Int' elements
let oddNumbers = [1, 3, 5, 7, 9, 11, 13, 15]

// An array of 'String' elements
let streets = ["Albemarle", "Brandywine", "Chesapeake"]
```

\

2. What is dictionary and write syntax in swift

-A collection whose elements are key-value pairs.

-Dictionaries are an association of an unordered collection of key-value pairs. Each value is associated with a unique key.

-EXAMPLE

```
-var responseMessages = [200: "OK",
                         403: "Access forbidden",
                         404: "File not found",
                         500: "Internal server error"]
```

-To create a dictionary with no key-value pairs, use an empty dictionary literal ([:]).

```
var emptyDict: [String: String] = [:]
```

3. How to specify a data type with Dictionary and Array

-Dictionary can also hold any data type in key and values, but in array of dictionary can hold a single data type.

-Array can also hold any data type, but inside array brackets can hold single data type.

4. What is set and syntax

-A set stores distinct values of the same type in a collection with no defined ordering. You can use a set instead of an array when the order of items isn't important, or when you need to ensure that an item only appears once.

-The type of a Swift set is written as Set<Element>, where Element is the type that the set is allowed to store. Unlike arrays, sets don't have an equivalent shorthand form.

-EXAMPLE

```
var favoriteGenres: Set<String> = ["Rock", "Classical", "Hip hop"]
OR
var = ([        ])
```

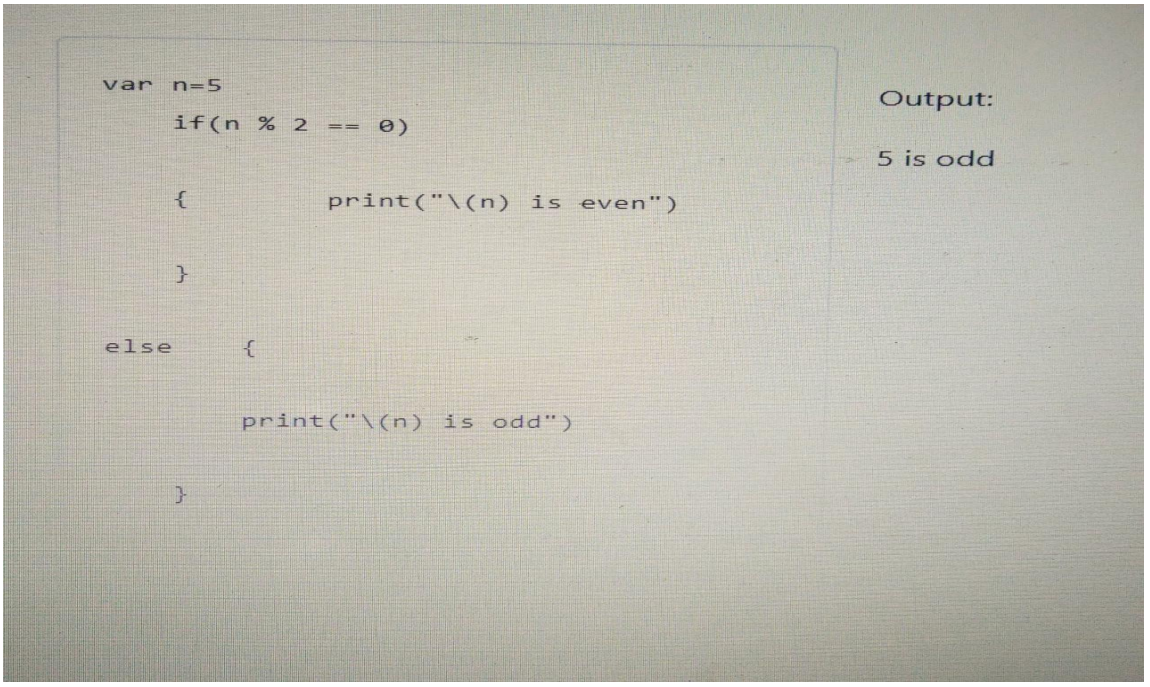
5. Difference between Array and Set

6. Difference between Array and NSArray

-An array can hold only one type of data, whereas NSArray can hold different types of data.

-An array is a value type, whereas NSArray is an immutable reference type.

7. Write a program to find Odd or Even using function with return type



8. What is Defer and how to use function inside Defer

-Swift's defer keyword lets us set up some work to be performed when the current scope exits.

-For example, you might want to make sure that some temporary resources are cleaned up once a method exits, and defer will make sure that happens no matter how that exit happens.

-Perfect ans .... compare to print and defer print

*Print will printed first

*Deferprint will be after print statement

```
func updateImage() {
  defer { print("Did update image") }

  print("Will update image")
  imageView.image = updatedImage
}

// Will update image
// Did update image
```

```
func printStringNumbers() {
  defer { print("1") }
  defer { print("2") }
  defer { print("3") }

  print("4")
}

/// Prints 4, 3, 2, 1
```

9. How to use for loop in swift

A for loop is control flow statements in Swift an its used to run a block of code a certain number of times. In this tutorial, we will learn how to use for-in loop in Swift with the help of examples.

10. Using for loop how to iterate the array in swift
We can use from [1,2,3,4,5,6,7,8,9,10]

Or
By using range operator like this loop it
we can use for [1...10]
We can use for [1..<10]
We can use for [1..>10]

11. Print 1 to 100 using for loop

```
for number in 1...100 {

    }
        print(number)
```

12. Print 100 to 1 using for loop
Try this🔥✨

```
for i in (0 ..< 101).reversed() {
    print(i)
}
```

13. Syntax for Switch statement and example program
The switch statement in Swift lets you inspect a value and match it with a number of cases. It's particularly effective for taking concise decisions based on one variable that can contain a number of possible values. Using the switch statement often results in more concise code that's easier to read.

Syntax for Switch statement and example program-mainly we used this code in segment button for purpose in post this is easy relate🔥✨

```
@IBOutlet weak var historyView: UIView!
@IBOutlet weak var popularView: UIView!

@IBAction func indexChanged(_ sender: UISegmentedControl) {
    switch segmentedControl.selectedSegmentIndex {
    case 0:
        historyView.isHidden = true
        popularView.isHidden = false
    case 1:
        historyView.isHidden = false
        popularView.isHidden = true
    default:
        break;
    }
```

14. What is Control Transfer Statements (Continue,Break,fallthrough,return)

Swift has five control transfer statments,they are

Control Transfer Statements: Keywords that allow you to transfer control from one piece of code to another

break: Breaks the current execution whenever the keyword is encountered

continue: Stops the execution of a loop at the current point, and returns control to the next iteration of the loop

fallthrough: Make a Switch run the next case after a match

return: Make a function return

throw: If a function throws an error the error is propogated from that function

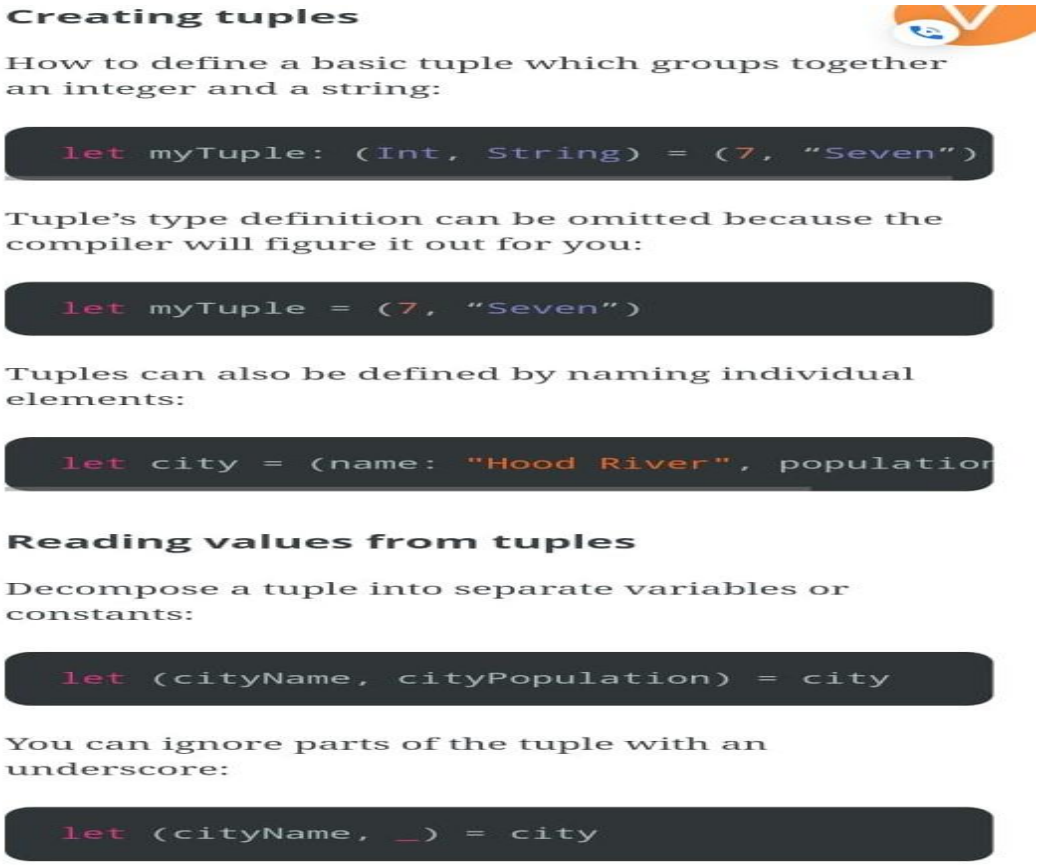
15. What is Tuples and write a example program

-A tuple is a group of different values in a single compound value In swift, a tuple can consist of multiple different types. It can support two values i.e. one of integer type, and the other of a string type. It is a legal command.

For Example:

- Let ImplementationError = (501, “Not implemented”).
- Let person = (name: “Ajay” , age: 34)

16. How to use function in tuples



17. What is Optional chaining and explain with syntax

Optional chaining is a process for querying and calling properties, methods, and subscripts on an optional that might currently be nil . If the optional contains a value, the property, method, or subscript call succeeds; if the optional is nil , the property, method, or subscript call returns nil .

Program

```
class Person {
   var residence: Residence?
   }
class Residence {
   var numberOfRooms = 1
   }
   let john = Person()
   let roomCount = john.residence!.numberOfRooms
   // this triggers a runtime error
   if let roomCount = john.residence?.numberOfRooms {
      print("John's residence has \(roomCount) room(s).")
      } else {
         print("Unable to retrieve the number of rooms.")
         }
   // Prints "Unable to retrieve the number of rooms."
```

18. What is Optional binding and explain with syntax?

Other than forced unwrapping, optional binding is a simpler and recommended way to unwrap an optional. You use optional binding to check if the optional contains a value or not. If it does contain a value, unwrap it and put it into a temporary constant or variable.

```
if let constantName = someOptional,
 let constantName2 = someOptional2 {
 // your statement goes here .....
print(constantName)
print(constantName2) }
```

19. What is Guard statement and explain with syntax?

Guard doesn't allow any code is condition is not true, it provides early exit.

A guard statement is as simple as using an if..else statement and has its own benefits and uses. Swift guard is defined as a statement that is used to transfer program control out of a scope if one or more conditions aren't met. ... Guard is commonly used inside a function, closure or loop

The syntax of the guard statement is: guard expression else { // statements // control statement: return, break, continue or throw. } Note: We must use return , break , continue or throw to exit from the guard scope.

PROGRAM
This is the part without guard

```
func fooBinding(x: Int?) {
   if let x = x where x > 0 {
      // Do stuff with x
      x.description
   }

   // Value requirements not met, do something
}
```

But now we can use guard and we can see that is possible to resolve some issues:

```
func fooGuard(x: Int?) {
   guard let x = x where x > 0 else {
      // Value requirements not met, do something
      return
   }

   // Do stuff with x
   x.description
}
```

20. Difference between if let and guard.(Explain detail)

-In if let , the defined let variables are available within the scope of that if condition but not in else condition or even below that. In guard let , the defined let variables are not available in the else condition but after that, it's available throughout till the function ends or anything.

-if let and guard let serve similar, but distinct purposes.

The "else" case of guard must exit the current scope. Generally that means it must call return or abort the program. guard is used to provide early return without requiring nesting of the rest of the function.

if let nests its scope, and does not require anything special of it. It can return or not.

In general, if the if-let block was going to be the rest of the function, or its else clause would have a return or abort in it, then you should be using guard instead. This often means (at least in my experience), when in doubt, guard is usually the better answer. But there are plenty of situations where if let still is appropriate.

21. What is Lazy property with example

- The lazy property is marked as lazy var. You can't make it lazy let because lazy properties must always be variables.
- Because the actual value is created by evaluation, you need to declare its data type up front. In the case of the code above, that means declaring the property as Int.
- Once you've set your data type, you need to use an open brace ("{") to start your block of code, then "}" to finish.
- You need to use self inside the function. In fact, if you're using a class rather than a structure, you should also declare [unowned self] inside your function so that you don't create a strong reference cycle.
- You need to end your lazy property with (), because what you're actually doing is making a call to the function you just created.

Once that code is written, you can use it like this:

var singer = Person()

print(singer.fibonacciOfAge)

22. What is Optional(?) ?

-optional allows you to write flexible and more safe code. It can be nil or some value.

- An optional in Swift is a type that can hold either a value or no value. Optionals are written by appending a ? to any type:

var name: String? = "Bertie"

Optionals (along with Generics) are one of the most difficult Swift concepts to understand. Because of how they are written and used, it's easy to get a wrong idea of what they are. Compare the optional above to creating a normal String:

var name: String = "Bertie" // No "?" after String

From the syntax it looks like an optional String is very similar to an ordinary String. It's not. An optional String is not a String with some "optional" setting turned on. It's not a special variety of String. A String and an optional String are completely different types.

Here's the most important thing to know: An optional is a kind of container. An optional String is a container which might contain a String. An optional Int is a container which might contain an Int. Think of an optional as a kind of parcel. Before you open it (or "unwrap" in the language of optionals) you won't know if it contains something or nothing.

23. What is Forced unwrapping

Forced unwrapping is one of a way to extract a value out of an optional. by simply adding exclamation point (!) at the end of the optional's name like below.

<u>Using Force Unwrapping</u>

```
let helloWorld: String? = "Hello World!"

print(helloWorld!)
// Hello World!

print(helloWorld)
// Optional("Hello World!")
```

Notice that the variable is optional with the question mark (?). In order to unwrap the optional, we force unwrap it with an exclamation point (!) and it prints out the value while on the other hand, without force unwrapping, it prints out the optional.
Do take note that you only use forced unwrapping if you are almost certain there is a value in it or it will crash.
Some of the most encouraged coding style is to safely unwrap the optional such as optional binding or nil-coalescing.

24. Difference between Single ? and Double ??

25. What is Generic Function?

With generics function you can write clear, flexible and reusable code. You avoid writing the same code twice, and it lets you write generic code in an expressive manner.
Let's take the original `addition(a:b:)` function, and turn it into a generic function. Like this:

```
func addition<T: Numeric>(a: T, b: T) -> T
{
   return a + b
}
```

26. What is Higher order function (Sort,Map,Filter)

27. Convert Int to String in Swift

Its easily possible with **string interpolation.**

Swift's string interpolation means you can convert all sorts of data – including integers – to a string in just one line of code:

let str1 = "\(myInt)"

(or)

However, the more common way is just to use the string initializer, like this:

let str2 = String(myInt)

28. Convert String to Int in swift

If you have an integer hiding inside a string, you can convert between the two just by using the integer's initializer, like this:

let myString1 = "556"

let myInt1 = Int(myString1)

Because strings might contain something that isn't a number – e.g. "Fish" rather than "556" – the Int initializer will return an optional integer, so if you want to force a value you should use nil coalescing like this:

let myInt2 = Int(myString) ?? 0

That means "attempt to convert myString to an integer, but if the conversion failed because it contained something invalid then use 0 instead."

29. What is Extension and use of Extension

-Extensions in Swift are super powerful, because they help you organize your code better.

-You use an **extension** to add new functionality to an existing class.

30. Use Extension in Color code and String

Here's a simple extension to UIColor that lets you create colors from hex strings. The new method is a failable initializer, which means it returns nil if you don't specify a color in the correct format. It should be a # symbol, followed by red, green, blue and alpha in hex format, for a total of nine characters. For example, #ffe700ff is gold. Here's the code:

```
extension UIColor {
    public convenience init?(hex: String) {
        let r, g, b, a: CGFloat

        if hex.hasPrefix("#") {
            let start = hex.index(hex.startIndex, offsetBy: 1)
            let hexColor = String(hex[start...])

            if hexColor.count == 8 {
                let scanner = Scanner(string: hexColor)
                var hexNumber: UInt64 = 0

                if scanner.scanHexInt64(&hexNumber) {
                    r = CGFloat((hexNumber & 0xff000000) >> 24) / 255
                    g = CGFloat((hexNumber & 0x00ff0000) >> 16) / 255
                    b = CGFloat((hexNumber & 0x0000ff00) >> 8) / 255
                    a = CGFloat(hexNumber & 0x000000ff) / 255

                    self.init(red: r, green: g, blue: b, alpha: a)
                    return
                }
            }
        }

        return nil
    }
}
```

If you wanted it always to return a value, change init? to be init then change the return nil line at the end to be return UIColor.black or whatever you'd like the default value to be.

To use the extension, write code like this:

```
let gold = UIColor(hex: "#ffe700ff")
```

31. What is Generics

Swift 4 language provides 'Generic' features to write flexible and reusable functions and types. Generics are used to avoid duplication and to provide abstraction. Swift 4 standard libraries are built with generics code. Swift 4s 'Arrays' and 'Dictionary' types belong to generic collections. With the help of arrays and dictionaries the arrays are defined to hold 'Int' values and 'String' values or any other types.

32. What is Closures and types

closures are self-contained blocks of functionality the can be passed around and used in code anywhere. It is headless functions. closures are functions without “Func” keyword and the function name.

33. What all are the types in Closure

Accepting parameters in closure
Returning parameters from a closure
Closures as parameters
Using Closures as parameters when they accept parameters
Using Closures as parameters when they return values
Trailing closure
Escaping closure

34. What is @escaping closure and @nonescaping closure

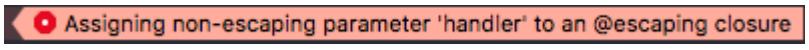
@escaping closure

When passing a closure as the function argument, the closure is being preserve to be execute later and function’s body gets executed, returns the compiler back. As the execution ends, the scope of the passed closure exist and have existence in memory, till the closure gets executed.

There are several ways to escaping the closure:

- **Storage**: When you need to preserve the closure in storage that exist in the memory, past of the calling function get executed and return the compiler back. (Like waiting for the API response)
- **Asynchronous Execution**: When you are executing the closure asynchronously on dispatch queue, the queue will hold the closure in memory for you, to be used in future. In this case you have no idea when the closure will get executed.

When you will try to use the closure with these option the swift compiler will show the error.



Lifecycle of the @escaping closure:

1. Pass the closure as function argument, during the function call.
2. Do some additional work in function.
3. Function execute the closure asynchronously or stored.
4. Function returns the compiler back.

@nonescaping closure

When passing a closure as the function argument, the closure gets execute with the function’s body and returns the compiler back. As the execution ends, the passed closure goes out of scope and have no more existence in memory.

Lifecycle of the @nonescaping closure

1. Pass the closure as function argument, during the function call.
2. Do some additional work with function.
3. Function runs the closure.
4. Function returns the compiler back.

35. What is Enum and types

An enumeration defines a common type for a group of related values and enables you to work with those values in a type-safe way within your code. Enumerations in Swift are much more flexible, and do not have to provide a value for each case of the enumeration.

Types

1.Enumerations with switch statements

2.Enum with Raw values

3.Enum with Associates values

36. What is Strong and Weak

STRONG

Let's start off with what a strong reference is. It's essentially a normal reference (pointer and all), but it's special in it's own right in that it *protects* the referred object from getting deallocated by ARC by increasing it's retain count by 1. In essence, *as long as anything* has a strong reference to an object, it will not be deallocated. This is important to remember for later when I explain retain cycles and stuff.

Strong references are used almost everywhere in Swift. In fact, the declaration of a property is strong by default! Generally, we are safe to use strong references when the hierarchy relationships of objects are linear. When a hierarchy of strong references flow from parent to child, then it's always ok to use strong references.

Here is an example of strong references at play.

```
class Kraken {
    let tentacle = Tentacle() //strong reference to child.
}
class Tentacle {
    let sucker = Sucker() //strong reference to child
}
class Sucker {}
```

WEAK

A weak reference is just a pointer to an object that *doesn't protect* the object from being deallocated by ARC. While strong references increase the retain count of an object by 1, weak references *do not*. In addition, weak references zero out the pointer to your object when it successfully deallocates. This ensures that when you access a weak reference, it will either be a valid object, or nil.

In Swift, all weak references are non-constant Optionals (think var vs. let) because the reference can and will be mutated to nil when there is no longer anything holding a strong reference to it.

For example, this won't compile:

```
class Kraken {
    //let is a constant! All weak variables MUST be mutable.
    weak let tentacle = Tentacle()
}
```

Because tentacle is a let constant. Let constants by definition cannot be mutated at runtime. Since weak variables can be nil if nobody holds a strong reference to them, the Swift compiler requires you to have weak variables as vars.

37. What is Access specifier in swift and types

Access specifier is keyword which helps in *Access control of* code block. *Access control* restricts access to the parts of your code from code in other source files and modules.

Types of Access levels in swift

1. Open & Public:

- **open** classes and class members can be subclassed and overridden both within and outside the defining module (target/framework).
- **public** classes and class members can only be subclassed and overridden within the defining module (target/framework).

2. Private:

Restricts the use of an entity to its enclosing declaration. You typically use private access to hide the implementation details of a specific piece of functionality when those details are used only within a single declaration.

3. FilePrivate:

Restricts the use of an entity to its defining source file. You typically use fileprivate access to hide the implementation details of a specific piece of functionality when those details are used within an entire file.

4. Internal:

Enables an entity to be used within the defining module (target). You typically use internal access when defining an app's or a framework's internal structure.

38. What is Class?

-class is reference type, means any changes in one reference, will effect other reference as well.

-Classes in Swift are building blocks of flexible constructs. Similar to constants, variables and functions the user can define class properties and methods. Swift provides us the functionality that while declaring classes the users need not create interfaces or implementation files. Swift allows us to create classes as a single file and the external interfaces will be created by default once the classes are initialized.

Benefits of having Classes

- Inheritance acquires the properties of one class to another class
- Type casting enables the user to check class type at run time
- Deinitializers take care of releasing memory resources
- Reference counting allows the class instance to have more than one reference

39. What is Structure

-struct is value type, means any changes in one value, will not effect the others.

-Swift lets you design your own types in two ways, of which the most common are called structures, or just *structs*. Structs can be given their own variables and constants, and their own functions, then created and used however you want.

Let's start with a simple example: we're going to create a Sport struct that stores its name as a string. Variables inside structs are called *properties*, so this is a struct with one property:

```
struct Sport {
    var name: String
}
```

That defines the type, so now we can create and use an instance of it:

```
var tennis = Sport(name: "Tennis")
print(tennis.name)
```

We made both name and tennis variable, so we can change them just like regular variables:

```
tennis.name = "Lawn tennis"
```

Properties can have default values just like regular variables, and you can usually rely on Swift's type inference.

40. Difference between Class and Structure with Examples

class is reference type, means any changes in one reference, will effect other reference as well.

struct is value type, means any changes in one value, will not effect the others.