Kotiln Interveiw Question

***Why should we use Kotlin?***

Kotlin is concise  
Kotlin is null-safe  
Kotlin is interoperable  
  
**Compact code**: Kotlin is an OOPs-based programming language with code lines that may be reduced by up to 40% when compared to Java, making it an excellent choice for software development.

**Open Source**: Kotlin for Android is open-source and uses the JVM to combine the benefits of OOPs and functional programming.

**Simple Language**: When working with Kotlin, compiling the code is simple, resulting in improved performance for Android development. It also explains which types of data functions can be used throughout the code.

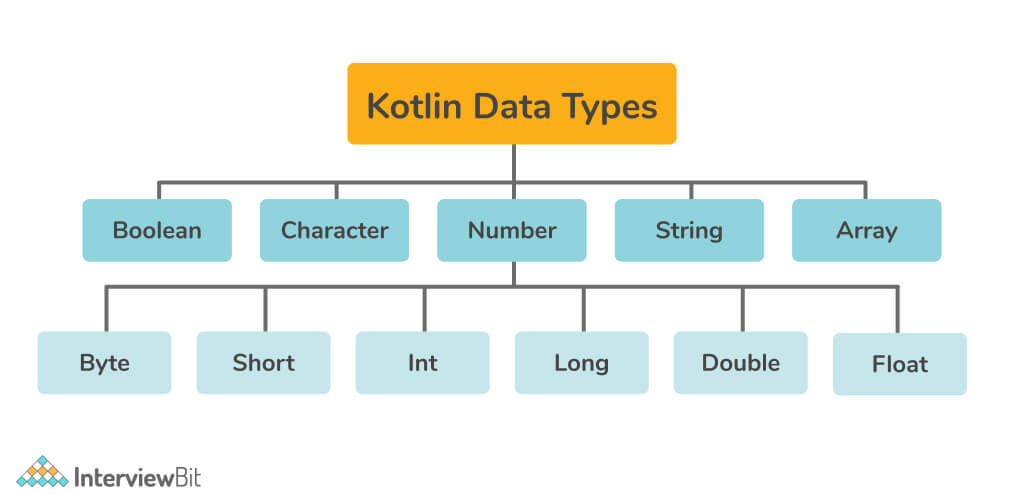
**High number of extensions**: Without modifying the code, Kotlin may support a variety of extension functions. It aids developers in making existing code more appealing and wonderful.

**Full Java Interoperability**: Java code can utilize Kotlin code, and Kotlin code can use Java code. So, if you're familiar with OOPS programming, switching to Kotlin development is simple. Also, if there are any Java-based applications, they can be used with Kotlin's environment.

**Smart Cast**: Smart casting is a technique that reduces the cost of an application while also improving its speed and performance. It uses typecasting or immutable data to manage the efficiency of programming. This technique is supported by Kotlin.

**Low Learning Curve**: Kotlin is preferred by businesses due to its low adoption cost. Most significantly, it is simple for developers to learn, especially if they have programming experience.

***Different Data Types in Kotlin***



***How are variables declared in Kotlin ?   
What are the different types of variables in Kotlin ?  
Explain with examples***

fun main(){  
 var email="VinayTShetty@gmail.com"  
 val password="123456"  
}

Different Types of variable in Kotlin are.

|  |  |
| --- | --- |
| ***Immutable Variables*** | ***Mutable variables*** |
| Immutable variables are also known as read-only variables. They are declared using the ***val*** keyword.  Once these variables have been declared, we cannot change their values | In a mutable variable, the value of the variable can be changed. We use the keyword “var” to declare such variables. |
| val password="123456" | var email="VinayTShetty@gmail.com" |

***What are data classes in Kotlin? Explain with a proper example.***

fun main(){}  
data class Test(val email:String,var password:String)

The following functions are automatically derived by the compiler for the data classes:

**equals()** - The equals() function returns true if two objects have the identical contents. It operates similarly to "==," although for Float and Double values it works differently.

**hashCode()** - The hashCode() function returns the object's hashcode value.

**copy()** - The copy() function is used to duplicate an object, changing only a few of its characteristics while leaving the rest unaltered.

**toString()** - This function returns a string containing all of the data class's parameters.

**To ensure consistency, data classes must meet the following requirements:**

At least one parameter is required for the primary constructor.  
val or var must be used for all primary constructor parameters.  
Abstract, open, sealed, or inner data classes are not possible.  
Only interfaces may be implemented by data classes.

***Explain Null Safe Operators***

***Assigining null values in Kotlin.***1.Assign null values in kotlin for Type inference is allowed.

***var username=null  
var password=null  
fun main() {  
 var email=null  
 var address=null  
 println("username = ${username}")  
 println("Passowrd = ${password}")  
 println("Email = ${email}")  
 println("Address = ${address}")  
}***

2. Assgin null values in kotlin for datatypes mentioned is Invalid.

***var username:String=null  
var password:String=null  
fun main() {  
 var email:String=null  
 var address:String=null  
 println("username = ${username}")  
 println("Passowrd = ${password}")  
 println("Email = ${email}")  
 println("Address = ${address}")  
}  
/\*\*  
e: D:\Delete\KotlinPractice\src\main\kotlin\main.kt: (1, 21): Null can not be a value of a non-null type String  
e: D:\Delete\KotlinPractice\src\main\kotlin\main.kt: (2, 21): Null can not be a value of a non-null type String  
e: D:\Delete\KotlinPractice\src\main\kotlin\main.kt: (4, 22): Null can not be a value of a non-null type String  
e: D:\Delete\KotlinPractice\src\main\kotlin\main.kt: (5, 24): Null can not be a value of a non-null type String  
 \*/***

***Nullable Operators(?)***

In Kotiln we cannot assign null values to variables with DataType.

using nullable property for assigining null values in Kotiln

***var username:String?=null  
var password:String?=null  
fun main() {  
 var email:String?=null  
 var address:String?=null  
 println("username = ${username}")  
 println("Passowrd = ${password}")  
 println("Email = ${email}")  
 println("Address = ${address}")  
}  
/\*\*  
Output  
username = null  
Passowrd = null  
Email = null  
Address = null  
 \*/***

***Error Case***

***var username:String=null  
var password:String=null  
fun main() {  
 var email:String=null  
 var address:String=null  
 println("username = ${username}")  
 println("Passowrd = ${password}")  
 println("Email = ${email}")  
 println("Address = ${address}")  
}  
/\*  
e: D:\Delete\KotlinPractice\src\main\kotlin\main.kt: (1, 21): Null can not be a value of a non-null type String  
e: D:\Delete\KotlinPractice\src\main\kotlin\main.kt: (2, 21): Null can not be a value of a non-null type String  
e: D:\Delete\KotlinPractice\src\main\kotlin\main.kt: (4, 22): Null can not be a value of a non-null type String  
e: D:\Delete\KotlinPractice\src\main\kotlin\main.kt: (5, 24): Null can not be a value of a non-null type String  
\*/***

***SafeCall(?.)***

1. Return the value if its not null else return null value.
2. Should use it, when the user is OK to show null as output.

var *username*:String?=null  
var *password*:Int?=123455  
fun main() {  
 var email:String?=null  
 var address:String?="ABC State India"  
 *println*("UserName= ${*username*?.length}")  
 *println*("Email= ${email?.*reversed*()}")  
  
 *println*("Password= ${*password*}")  
 *println*("Address= ${address}")  
}  
  
*/\*\*  
 Output  
  
 UserName= null  
 Email= null  
 Password= 123455  
 Address= ABC State India  
 \*/*

***SafeCall with let (?.let)***

1. It Executes the blocks if the value is not null.
2. If the value is null, It wont execute the block

var *username*: String? = null  
var *password*: Int? = 123455  
fun main() {  
 var email: String? = null  
 var address: String? = "ABC State India"  
  
 *username*?.*let* **{** *println*("Username= ${*username*}")  
 **}** email?.*let* **{** *println*("Email = ${email}")  
 **}** *password*?.*let* **{** *println*("pasword= ${*password*}")  
 **}** address?.*let* **{** *println*("Address= ${address}")  
 **}**}  
/\*  
OutPut  
pasword= 123455  
Address= ABC State India  
\*/

Without Safe Call Test case where null values are printed.

var *username*: String? = null  
var *password*: Int? = 123455  
fun main() {  
 var email: String? = null  
 var address: String? = "ABC State India"  
  
 *username*.*let* **{** *println*("Username= ${*username*}")  
 **}** email.*let* **{** *println*("Email = ${email}")  
 **}** *password*.*let* **{** *println*("pasword= ${*password*}")  
 **}** address.*let* **{** *println*("Address= ${address}")  
 **}**}  
  
/\*  
Output  
  
Username= null  
Email = null  
pasword= 123455  
Address= ABC State India   
\*/

***Elvis Operator(?:)***

Returns its first operand if that operand evaluates to a true value, and otherwise evaluates and returns its second operand

var *username*: String? = null  
var *password*: String? = "123455"  
fun main() {  
 var email: String? = null  
 var address: String? = "ABC State India"  
  
 val usname=*username*?.length?:-1  
 val pswrd=*password*?.length?:-1  
 val emil=email?.length?:-1  
 val addrs=address?.length?:-1  
 *println*("Username= ${usname}")  
 *println*("Password= ${pswrd}")  
 *println*("Email= ${emil}")  
 *println*("Address= ${addrs}")  
}  
  
/\*  
Output  
Username= -1  
Password= 6  
Email= -1  
Address= 15   
\*/

***Non-null Assertion Operator (!!)***

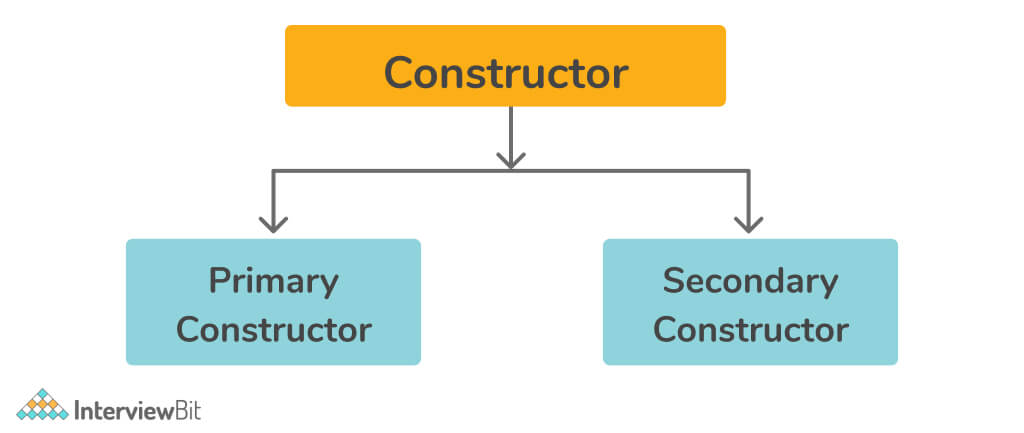
Should use when the value is compulsary is not null. Else will get a exception at runtime

var *password*: String? = "123455"  
fun main() {  
 var address: String? = "ABC State India"  
 *println*("Password= ${*password*!!.length}")  
 *println*("Address= ${address!!.length}")  
}

Error Test Case

var *username*: String? = null  
var *password*: String? = "123455"  
fun main() {  
 var email: String? = null  
 var address: String? = "ABC State India"  
 *println*("Username= ${*username*!!.length}")  
 *println*("Password= ${*password*!!.length}")  
 *println*("Email= ${email!!.length}")  
 *println*("Address= ${address!!.length}")  
}  
/\*  
OutPut  
Exception in thread "main" java.lang.NullPointerException  
at MainKt.main(main.kt:7)  
at MainKt.main(main.kt)  
\*/

***What are the different types of constructors available in Kotlin? Explain them with proper examples.***



***Primary Constructor***  
  
This type of constructor is initialised in the class header and is provided after the class name. It is declared using the “constructor” keyword. Parameters are optional in this type of constructor  
  
Ex-

class Sample constructor(val a: Int, val b: Int) {}

***Secondary Constructor***  
Secondary Constructor - Secondary constructors allow for the initialization of variables as well as the addition of logic to the class. They have the constructor keyword prefixed to them

fun main(args: Array<String>) {  
 val s1 = Sample(1, 2)  
}  
class Sample {  
 constructor(a: Int, b: Int) {  
 *println*("The first parameter value is : $a")  
 *println*("The second parameter value is : $b")  
 }  
}

***Explain the various methods to iterate over any data structure in Kotlin***  
For-loop  
whileloop  
Do-while loop  
iterator

***Function Interview Question***

***Extension Function***

fun main(args: Array<String>) {  
 // Extension function created for a class Circle  
 fun Circle.perimeter(): Double{  
 return 2\*Math.*PI*\*radius;  
 }  
 // create object for class Circle  
 val newCircle = Circle(2.5);  
 // invoke member function  
 *println*("Area of the circle is ${newCircle.area()}")  
 // invoke extension function  
 *println*("Perimeter of the circle is ${newCircle.*perimeter*()}")  
}  
class Circle (val radius: Double){  
 // member function of class  
 fun area(): Double{  
 return Math.*PI* \* radius \* radius;  
 }  
}

***Inline Function***

The inilne function is a function that is declared with the keyword ‘inline’  
The use of inline function enhances the performance of higher order function  
The inline function tells the compiler to copy parameters and function to call site.

fun main() {  
var mylambda:()->Unit=**{**-> *println*("My Lambda")**}** *test*(mylambda)   
}  
  
  
fun test(fn: () -> Unit) {  
 fn()  
 *println*("My Test inline function")  
}

***Variables***

lateinit and lazy

|  |  |
| --- | --- |
| ***lateinit*** | ***lazy*** |
| The “lateinit” keyword in Kotlin as the name suggests is used to declare those variables that are guaranteed to be initialized in the future. | When the lazy keyword is used, the object will be created only when it is called, otherwise there will be no object creation. |
| Only mutable vairables i.e var can be used for lateinit. | Immutable property can be used for lazy. i.e val |
| lateinit does not work on primitive datatypes. |  |
| lateinit can be used when the variable does not have getter and setter property. |  |

lateinit

lateinit var username:String  
fun main() {  
 username="Vinay"  
 println(username)  
}

lazy

fun main() {  
 *println*(*pi*)  
}  
val *pi* by *lazy***{** 3.14  
**}**