Que-1) Explain the different data types available in Kotlin. How do val and var differ? What is a lambda expression in Kotlin, and where can it be used?

Ans:

1)Int
2)String
3)Char
4)Byte
5)Short
6)Long
7)Double

Difference between Val and Var

val	var
it cannot be changed.	it can be reassigned to a new value.
Used when you don't want the variable to be changed .	Used when the variable might need to be changed over time.
The reference is immutable, but the object it points to can still be mutable	Both the reference and the object are mutable.

What is a lambda expression in Kotlin, and where can it be used?

→ A lambda expression in Kotlin is enclosed in curly braces `{}`.

→it can be passed around and used as an argument to other functions.

Use of lamda function :-

→Functions that take other functions (lambdas) as arguments or return

→ Lambdas can be passed as callbacks for event handling or async operations.

Que-2) Describe the principles of Object-Oriented Programming (OOP). Explain the differences between abstract class and interface in Kotlin and provide examples of when to use them.

Ans:-

Abstraction in Object-Oriented Programming (OOP) is the concept of hiding the complex implementation details of a system and exposing only the necessary features or functionalities.

Encapsulation in Object-Oriented Programming refers to the concept of bundling variables and methods functions

Inheritance in Object-Oriented Programming OOP allows one class to inherit properties and methods from another class.

Polymorphism in Object-Oriented Programming (OOP) allows objects of different classes to be treated as objects of a common superclass.

Abstract Classes:

1. We can’t create an object for abstract classs
2. All the variables (properties) and member functions of an abstract class are by default non-abstract.
3. If we declare a member function as abstract then we does not need to annotate with open keyword because these are open by default.

Example of Abstract Class:-

```
abstract class Animal(val name: String) {
    abstract fun makeSound()
    fun sleep() {
        println("$name is sleeping.")
    }
}

class Dog(name: String) : Animal(name) {
    override fun makeSound() {
        println("$name says Woof!")
    }
}
```

Interface:

→When you want to define common behavior for unrelated classes.

→When you need multiple inheritance (a class can implement multiple interfaces).

```
interface Camera {
    fun takePhoto() {
        println("Taking a photo...")
    }
}
interface MusicPlayer {
    fun playMusic() {
        println("Playing music...")
    }
}

class Smartphone : Camera, MusicPlayer {
    fun makeCall() {
        println("Making a call...")
    }
}
```