Assignment

[Module 2 – Kotlin Programming]

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: Data types:

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

Difference between Val and Var

Aspects	**Val**	**Var**
Mutability	Immutable (read-only)	Mutable (can be reassigned)
Usage	Used for constants or variables that don't change after initialization.	Used for variables whose values need to change.
Example	Val name =" Kotlin"	Var id = 101

Lambda Expression

- A lambda expression in Kotlin is defined within curly braces. The code inside the braces represents the lambda body. Here, a and b are 'function parameters', and ->

signifies the start of the 'lambda body'. The sum is a 'Val lambda'.

- Syntax:

  val lambdaName: (InputType) -> ReturnType = {parameterName -> body}

- Usage of lambda Expression
  - When passing function as parameter to another function.
  - Arrow (->) operator is used to separate body and arguments of function
  - Code becomes easier to understand when written in a functional style.

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: Principles of Object-Oriented Programming

- Abstraction
  - hiding internal details and showing functionality
  - Ex: login page
- Encapsulation
  - wrapping up of data or binding of data
  - Ex: capsule
- Inheritance
  - when an object acquires all the properties and behaviour of parent class

    - Ex: father-son
- Polymorphism
    - many ways to perform anything
    - Ex: method overloading
    - Ex: method overriding

Difference between Abstract class and interface

Abstract Classes:

- Can have both abstract and concrete methods.
- Can have properties with initializers.
- Can have constructors.
- Can be inherited by multiple classes, but only one direct parent.
- Used when you want to define a partial implementation and provide default behaviour.

Example:

```
abstract class Shape
 {
    abstract fun draw()
   fun erase()
   {
      println("Erasing shape")
   }
 }
class Circle : Shape()
```

```
 {
override fun draw()
 {
      println("Drawing a circle")
 }
}
```

Interface:

- Can only have abstract methods.
- Cannot have properties with initializers or constructors.
- Can be inherited by multiple classes.
- Used when you want to define a contract that multiple classes must adhere to.

Example:

```
interface Drawable
{
   fun draw()
}
class Square : Drawable {
   override fun draw()
   {
      println("Drawing a square")
   }
}
```