Synopsis Kotlin

Task – Take a note of today's kotlin Programming language coverd topics.

Annotations – In the Kotlin programming language, annotations serve as a powerful mechanism to add metadata to code elements, such as classes, functions, parameters, and properties, without affecting their actual functionality.

In line- By "inlining" the function code, Kotlin improves performance by reducing the overhead associated with function calls. It's like having an efficient assistant who anticipates your needs, works alongside you, and streamlines the entire process.

Regex- A regular expression, often shortened to "regex" or "regexp", is a sequence of characters that define a search pattern. Regular expressions are used to match patterns in strings, and are a powerful tool for text processing. They can be used for tasks such as validation, search-and-replace, and text parsing.

Triple – As the code above shows, Triple is a data class. That is to say, Triple is final and cannot have a subclass. Further, Triple is immutable since all three properties are defined as val. We can put three values from different types in a Triple, for example, Triple(42, "Kotlin", Long.MAX_VALUE)

Destruction – Kotlin's destructuring declarations allow you to extract fields from an object to a set of variables. Then you can use variables independently in any way out. Under the hood, the compiler uses componentN operator functions to unpack values of an object and assign them to the variables.

Operator overloading – Operator overloading Kotlin allows you to provide custom implementations for the predefined set of operators on types. These operators have predefined symbolic representation (like + or *) and precedence.

Higher Order Function – In the Use function types and lambda expressions in Kotlin codelab, you learned about higher-order functions, which are functions that take other functions as parameters and/or return a function, such as repeat() .

Implementation

```
Fun main() {

Val pattern = Regex("ll") // match ll

Val res : MatchResult? = pattern.find("Hello Hello", 5)
```

```
    Println(res?.value)
}
Fun main() {
    Var obj = Triple(1, 2, 3)
    Println(obj.toList())
}
Data class Data(val name: String, val age: Int)
Fun sendData() : Data {
    Return Data(“Amulya”, 50)
}
Fun main() {
    Val obj = sendData()
    Println(“Name is ${obj.name}”)
    Println(“Age is ${obj.age}”)
    // Destructing objects
    Val (name, age) = sendData()
    Println(“$name “ + “$age”)
}
Class Object(var objName: String) {
    // overloading the func
    Operator fun plus(b: Int) {
        objName = “Name is $objName and data is $b”
    }
    Override fun toString(): String {
        Return objName
    }
```

```
}
Fun main() {
   Val obj = Object(“Amulya”)
   Obj+10
   Println(obj)
}
Fun hof(str: String, mycall: (String) -> Unit) {
   Mycall(str)
}
Fun main() {
   Println(“Result: “)
   Hof(“Achal”, ::println)
}
```