Deprecated:

```
@Deprecated("this fucntion is deprecated", ReplaceWith("function b"))
fun a() {
   println("a")
}
fun b() {
  println("b")
}
fun main() {
   println(a())
}
```

Output: a

```
kotlin.Unit
```

Use: This annotation marks the 'a()' function as deprecated. It provides a message ("this function is deprecated").It provides a message ("this function is deprecated") that explains why the function is deprecated. Additionally, it suggests using function b instead (using the ReplaceWith parameter).

Reflection:

```
fun main() {

   val ref = RefDemo::class
   println("$ref")
   val obj = RefDemo()
   println("${obj::class}")
}

class RefDemo {

}
```

Output: class RefDemo (Kotlin reflection is not available)

```
class RefDemo (Kotlin reflection is not available)
```

Use: Reflection in programming refers to the ability of a program to examine and modify its own structure, behavior, and state at runtime. It allows a program to inspect its own code and manipulate objects, classes, functions, and other entities dynamically, rather than statically at compile time.

RegEx:

```
fun main() {

   val pattern = Regex("ll")

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

   println(res?.value)

}
```

Output: ll

Use: Regular expressions (regex) are widely used in programming for various tasks involving text processing and pattern matching.

Triple class:

```
fun main() {
   var obj = Triple(1, Hello, false)
   println(obj.toList())

}
```

Output: [1, Hello, false]

Use: the Triple class is primarily used to hold three values of potentially different types together as a single entity,to convert a Triple object to a list, you can use the toList() function which is provided as an extension function in Kotlin standard library for various collection-like classes, including Triple.

Data Class (Data):

```
data class Data(val name: String, val age: Int)
fun sendData() : Data {
        return Data("Chandu", 23)
}
fun main() {
        val obj = sendData()
        println("Name is ${obj.name}")
        println("Age is ${obj.age}")
        val (name, age) = sendData()
        println("$name " + "$age")
}
```

Output:

```
        Name is Chandu
        Age is 23
        Chandu 23
```

Use: data class is a special class used for holding data/state. It automatically generates equals(), hashCode(), toString(), and copy() methods based on the properties defined in the primary constructor.

Operator Overloading:

```
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("Ramu")
   obj+23
   println(obj)
}
```

Output: Name is Ramu and data is 23

Use: Kotlin allows you to overload operators by defining functions with specific names that correspond to the operator (e.g., plus for +, minus for -, etc.).

Higher-order function:

```
fun hof(str: String, mycall: (String) -> Unit) {
   mycall(str)
}
fun main() {
   println("Result: ")
   hof("My HOF", ::println)
}
```

Output:

```
Result:
My HOF
```

Uses: A higher-order function is a function that either takes one or more functions as parameters or returns a function. This approach demonstrates the flexibility and power of higher-order functions in Kotlin, enabling you to pass behavior around as data, which can lead to more modular and reusable code.