Kotlin Training Day 1

Kotlin Conventions

- Naming conventions: camelCase for functions and properties, PascalCase for classes.
- Prefer val (immutable) over var (mutable) wherever possible.
- Always specify data types for public API.

Null Safety

Good Scenario:

```
val name: String? = null
println(name?.length)  // null
val age: Int? = null
val youth: Int = age ?: 0  // Provides default
value if age is null
```

Bad scenario:

```
val name: String? = null
println(name?.length)  // null
val age: Int? = null
val youth: Int = age ?: 0  // Provides default
value if age is null
```

Comment: Checking nullability ensures safe code without runtime crashes.

Basic Types and Type Inference

Explicit types:

```
val balance: BigInteger =
BigInteger("100000000000")
val walletPublished: Boolean = true
val name: String = "Bob"
```

Type inference:

```
val tierLevel = 5  // Int inferred
val accountName = "Alice"  // String
inferred
```

Comment: Kotlin often infers the type from the assigned value.

Control Flow and Pattern Matching

If statment:

```
if (tierLevel > 10) {
    println("Platinum")
} else if (tierLevel > 5) {
    println("Gold")
} else {
    println("Silver")
}
```

Pattern matching with when:

```
val balance = BigInteger.valueOf(9000000)
when (balance) {
    BigInteger.ZERO -> println("No balance!")
    in BigInteger.ONE..BigInteger("10000") ->
println("Low balance")
    else -> println("Good balance!")
}
```

String Manipulation and Printing

```
val greeting = "Hello"

val entity = "World"


println("$greeting $entity!")  // String template

println(greeting + " " + entity + "!")  // Concatenation
```

Comment: String templates provide a clean way to interpolate values.

Collections

```
val customerNames = listOf("Alice", "Bob", "Charlie")

val accountBalances = listOf(1000, 500, 2000)


val totalBalance = accountBalances.sum()

val highValueCustomers = customerNames.filter { it.startsWith("A") }
```

Comment: Kotlin provides rich standard library functions for collection operations.

Kotlin Collections - Lists

```
val customerNames = listOf("Alice", "Bob", "Charlie")

val primeNumbers = listOf(2, 3, 5, 7, 11)

val filteredNames = customerNames.filter { it.startsWith("A") }

val squaredPrimes = primeNumbers.map { it * it }
```

Kotlin Collections - Arrays

```
val accountBalances = arrayOf(1000.0, 500.0, 2000.0)

val totalBalance = accountBalances.sum()
```

Functions in Kotlin

```
fun calculateInterest(balance: Double, rate: Double = 0.05): Double {
    return balance * rate
}
fun main() {
    val interest = calculateInterest(1000.0)  // Uses default rate
    val customInterest = calculateInterest(1000.0, 0.04)  // Overrides default rate
}
```

Comment: Kotlin functions can have default arguments, making function calls flexible.

Varargs and Spread Operators

```
fun printBalances(vararg balances: Double) {
    for (balance in balances) println(balance)
}
printBalances(100.0, 200.0, 300.0)
```

Spread operator:

```
val balancesArray = arrayOf(100.0, 200.0, 300.0)
printBalances(*balancesArray)
```

Comment: vararg lets you pass multiple arguments of the same type to a function. The spread operator (*) can be used to unpack an array into varargs.

Classes in Kotlin

```
class Wallet(val id: String, var balance: BigInteger)
```

usecase:

```
val aliceAccount = BankAccount("123456", BigInteger("123456")

println(aliceAccount.balance)
```

Classe Properties

```
class Wallet(_id: String, _balance: BigInteger)  {
   val id: String
   val balance: BigInteger
   init {
      id = _id
      balance =  _balance
   }
}
```

=

```
class Wallet(val id: String,
val balance: BigInteger)  {}
```

Visibility Modifiers

```
open class Bank {
    private val secretCode = "XYZ"                  // Can only be accessed within the Bank class
    protected val vaultPassword = "1234"            // Can be accessed within the Bank class and any subclasses of Bank
    internal val internalAuditNumber = "7890"       // Can be accessed within the same module that this Bank class is defined
    public val bankName = "Global Bank"             // Can be accessed from any other class, the default visibility if no modifier is specified
}
```

Comment: Kotlin provides various visibility modifiers to control access to properties and functions.

Visibility Class Function Example

```
val myBank = Bank()
println(myBank.bankName)  // Outputs: "Global Bank"

class SantanderBank : Bank() {

    fun printVaultPassword() {

        println(vaultPassword)     }

}

val santander = SantanderBank()

santander.printVaultPassword()  // Outputs: "1234"
```