**Lazy initialization in Kotlin**

In Kotlin, `lazy` is a property delegate that helps in implementing lazy initialization. Lazy initialization means that the value of a property is computed only when it's accessed for the first time. This can be useful for performance optimization and managing resources efficiently.

**Key Features of `lazy` in Kotlin**

1. \*\*Lazy Initialization\*\*:

- The value is computed only once, when it is accessed for the first time.

- Subsequent accesses return the previously computed value without recomputing it.

1. \*\*Thread-Safety Modes\*\*:

- \*\*SYNCHRONIZED\*\* (default): Ensures that the lazy initialization is thread-safe, suitable for multi-threaded environments.

- \*\*PUBLICATION\*\*: Allows multiple threads to initialize the value but guarantees that the same value will be used afterward.

- \*\*NONE\*\*: No thread-safety guarantees, suitable for single-threaded scenarios.

**Syntax and Usage**

Basic Usage:

**kotlin**

val lazyValue: String by lazy {

println("Computed!")

"Hello, Lazy Initialization!"

}

fun main() {

println(lazyValue) // Computation happens here

println(lazyValue) // Returns the cached value

}

```

**Output**:

Computed!

Hello, Lazy Initialization!

Hello, Lazy Initialization!

```

#### Custom Thread-Safety Mode

```kotlin

val lazyValue: String by lazy(LazyThreadSafetyMode.PUBLICATION) {

println("Computed!")

"Hello, Lazy Initialization!"

}

```

**Use Cases for `lazy`**

1. \*\*Expensive Computation\*\*:

- Delay the computation of a value until it is actually needed to save resources.

2. \*\*Resource-Intensive Initialization\*\*:

- Initialize objects that require significant memory or processing power only when they are first used.

3. \*\*Dependency Initialization\*\*:

- Delay the initialization of dependencies that might not always be needed.