1. Variables:
Kotlin uses 'val' for immutable (read-only) variables and 'var' for mutable variables.
Type inference allows omitting the type declaration if it can be determined from the context.

```
val immutable = 5 // Can't be reassigned
var mutable = 10; mutable = 15 // Can be reassigned
```

2. Classes and Objects:
Classes in Kotlin define blueprints for objects, encapsulating data and behavior.
Objects are instances of classes, created using the class constructor.
Kotlin allows declaring properties directly in the primary constructor.

```
class Person(val name: String, var age: Int) {
    fun introduce() = println("I'm $name, $age years old")
}

val john = Person("John", 30)
john.introduce() // Prints: I'm John, 30 years old
```

3. Data Types:
Kotlin has various built-in data types including Int, Long, Float, Double, Boolean, Char, and String.
Kotlin also supports nullable types, denoted by adding a '?' after the type.

```
val integer: Int = 42
val nullable: String? = null
```

4. Functions:
Functions in Kotlin are declared using the 'fun' keyword. They can have parameters and return values.
Single-expression functions can be written concisely using an '=' sign instead of curly braces.

```
fun greet(name: String): String { return "Hello, $name!" }
fun greetConcise(name: String) = "Hello, $name!"
```

5. Control Flow:
Kotlin provides if-else statements and when expressions for conditional logic.
For loops, while loops, and do-while loops are available for iteration.

```
val result = if (10 > 5) "Greater" else "Less"
when (result) {
    "Greater" -> println("10 is indeed greater than 5")
    else -> println("Unexpected result")
}
```

6. Inheritance:
Kotlin classes are final by default. Use 'open' keyword to allow inheritance.
Subclasses are declared using a colon after the class name.

```
open class Animal(val name: String)
class Dog(name: String) : Animal(name) {
```

```
    fun bark() = println("$name says: Woof!")
}
```

7. Ranges:
Ranges in Kotlin represent an interval of values. They are created using the '..' operator or the 'rangeTo()' function.
Ranges can be used with numbers, characters, and even custom types that implement the 'Comparable' interface.

-Numeric Ranges:
Integer and long ranges are commonly used in 'for' loops and to check if a value is within a specific interval.
They can be ascending or descending.

```
for (i in 1..5) print(i) // Prints: 12345
println((5 downTo 1).toList()) // Prints: [5, 4, 3, 2, 1]
```

-Char Ranges:
Character ranges are useful for iterating through alphabets or generating sequences of characters.
They follow the ASCII/Unicode order.

```
for (c in 'a'..'e') print(c) // Prints: abcde
println(('Z' downTo 'X').toList()) // Prints: [Z, Y, X]
```

8. Null Safety:
Kotlin's type system distinguishes between nullable and non-nullable types to prevent null pointer exceptions.
The safe call operator '?.' and the Elvis operator '?:' are used to handle potential null values safely.

```
val nullableString: String? = null
val length = nullableString?.length ?: 0
println("The length is: $length")
```