**1. What is the lambda expression of Java 8?**

Ans - The term implies its function: enabling more concise Java 8 code writing. Consider (a, b) -> a + b, a lambda expression (indicated by ->). This is equivalent to the subsequent code:

public int value(int a, int b) {

return a + b;

}

Termed an anonymous function, it mirrors a named function's code, albeit without the nomenclature.

**2. Can you pass lambda expressions to a method? When?**

Ans - Certainly, it's possible to supply a lambda expression to a method, given that the method anticipates a functional interface. For instance, if a method receives a Runnable, Comparable, or Comparator, you can employ a lambda expression, as these interfaces are all classified as functional interfaces in Java 8.

**3. What is the functional interface in Java 8?**

Ans - In Java 8, a functional interface denotes an interface containing only a single abstract method. Take, for instance, the Comparator interface, housing the singular abstract method named compare(), or the Runnable interface with its lone abstract method, run(). The java.util.function package in JDK introduces various other versatile functional interfaces, also marked with the optional @FunctionalInterface annotation.

**4. What is the benefit of lambda expressions in Java 8?**

Ans - The primary advantage of lambda expressions in Java 8 is the simplified process of transmitting a code block to a method. Previously, accomplishing this task necessitated encapsulating the code within an anonymous class, entailing a substantial amount of repetitive code.

**5. Is it mandatory for a lambda expression to have parameters?**

Ans - Indeed, a lambda expression is not obligatory to include parameters. You can create a lambda expression without parameters, as illustrated here:

() -> System.out.println("lambda without parameters");

Such code can be provided to any method that anticipates a functional interface.