1.	**What is the lambda expressions of java 8?**
Ans.	In Java 8, lambda expressions were introduced as a new language feature. Lambda expressions provide a concise way to represent anonymous functions or "function literals" in Java. They enable you to treat functionality as a method argument or code as data. (parameter list) -> { lambda body }
2.	**Can you pass lambda expression to a method? When?**
Ans.	Yes, we can pass lambda expressions as arguments to methods. This is made possible by the introduction of functional interfaces, which are interfaces with a single abstract method. Lambda expressions can be used as implementations of these functional interfaces.
3.	**What is the functional interface in Java 8?**
Ans.	a functional interface is an interface that has only one abstract method. It is designed to be used as a target for lambda expressions or method references. The **@FunctionalInterface** annotation can be used to explicitly mark an interface as a functional interface, but it is optional.
4.	**Why do we use lambda expression in java?**
Ans.	Lambda expressions are used in Java for several reasons: 1. Functional Programming: Lambda expressions enable functional programming paradigms in Java. They allow you to treat functions as first-class citizens, meaning that functions can be assigned to variables, passed as arguments to other methods, and returned as results from methods. This functional programming approach promotes code modularity, reusability, and expressiveness. 2. Concise Syntax: Lambda expressions provide a concise syntax for representing anonymous functions or function literals. They eliminate the need to write boilerplate code when implementing functional interfaces, reducing the verbosity of code. This makes the code more readable and maintainable. 3. Improved APIs: Lambda expressions enhance the flexibility and usability of APIs. They allow API designers to provide functional interfaces as method arguments, enabling developers to pass behavior in the form of lambda expressions. This promotes code customization and reduces the need for creating numerous callback classes or interfaces.

	4. Enhanced Collections and Stream API: Lambda expressions work synergistically with the enhanced Collections and Stream API introduced in Java 8. They provide a compact way to perform operations such as filtering, mapping, and reducing elements in collections or streams. This leads to more expressive and readable code when working with collections and data processing. 5. Parallel Processing: Lambda expressions facilitate parallel processing in Java. The Stream API, combined with lambda expressions, enables easy parallelization of operations on large data sets. This simplifies the development of concurrent and parallel algorithms, improving performance in multi-core environments. 6. Code Organization and Readability: Lambda expressions help in organizing and simplifying code by encapsulating behavior in a concise and focused manner. They allow you to express the intent of the code more clearly, leading to improved readability and understanding.
5. **Ans.**	**Is it mandatory for a lambda expression to have parameter?** No, it is not mandatory for a lambda expression to have parameters. The presence or absence of parameters in a lambda expression depends on the functional interface that it is associated with. A lambda expression can have zero, one, or multiple parameters, depending on the abstract method of the functional interface it is implementing. The number and types of parameters in the lambda expression must match the parameter types and order of the abstract method.