1. What is the Lambda expression of Java 8?

Ans: It is a feature in Java 8 which allows us to reduce code for Functional Interfaces, i.e, interfaces that have only one method. It is similar to Anonymous classes in Java because we are writing the code inside a class that doesn't have a name.

Example:

(int a, int b) -> System.out.println("a is " + a "and b is " + b);

Is a Lambda Expression which is a simplified form of the below code:

```
public void num(int a, int b) {
   System.out.println("a is " + a "and b is " + b);
}
```

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

Ans:Yes, we can pass a lambda expression to a method provided it is expecting a functional interface. For example, if a method is accepting a Runnable, Comparable or Comparator then we can pass a lambda expression to it because all these are functional interfaces in Java 8.

3. What is the functional interface in Java 8?

Ans:A functional interface in Java 8 is an interface with a single abstract method. For example, Comparator which has just one abstract method called compare() or Runnable which has just one abstract method called run() are Functional Interfaces. They are also annotated with @FunctionalInterface but it is optional.

There are many more general purpose functional interfaces introduced in JDK on java.util.function package.

4. Why do we use lambda expressions in Java?

Ans: The advantages of using Lambda Expressions are:

i. We can easily pass code blocks as arguments to a method.

ii. Before Java 8, Anonymous class was used which made us write more amount of code for using the interfaces. But since Java 8, the use of Lambda Expressions has reduced the code required for using Functional Interfaces.

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

Ans: No, it is not mandatory for lambda expression to have parameters. We can defined Lambda expressions with no parameters.

Example:

() -> System.out.println(" This is a Lambda Expression with no parameters");

We can pass this expression to the method which accepts a Functional Interface.