

**Q1. What is the lambda expression of Java 8?**

Ans. As its name suggests it's an expression which allows us to write more succinct code in Java 8. For example `(a,b) -> a+b` //is a lambda expression (look for that arrow ->).

**Which is equal to following code:**

```
public int value(int a , int b)
{
    return a+b;
}
```

It is also called an anonymous function because we are essentially writing the code we write in function but without name.

**Q2. 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 you can pass a lambda expression to it because all these are functional interfaces in Java 8.

**Q3. 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()`. There are many more general purpose functional interface introduced in JDK on `java.util.function` package. They are also annotated with `@FunctionalInterface` but that's optional.

**Q4. What is the benefit of lambda expression in Java 8?**

Ans. The main benefit of lambda expression in Java 8 is that it's easier to pass a code block to a method. Earlier, the only way to do this was wrapping the code inside an Anonymous class, which requires a lot of boilerplate code.

**Q5. Is it mandatory for a lambda to have parameters?**

Ans. No, it's not mandatory for a lambda expression to have parameters, we can define a lambda expression without parameters as shown below:

```
( ) -> System.out.println("Lambda expression without parameter");
```

We can pass this code to any method which accepts a functional interface.