1) What is the lambda expression of Java 8?

Ans: As its name suggests it's an expression which allows you 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 the following code:

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

It's also called an anonymous function because you are essentially writing the code you write in function but without name.

2) Can you pass lambda expressions to the method? When?

Ans: Yes, you 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.

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 compare() or Runnable which has just one abstract method called run(). There are many more general purpose functional interfaces introduced in JDK on java.util.function package. They are also annotated with @FunctionalInterfaces but that's optional.

4) Why do we use lambda expressions in Java? &
5) Is it mandatory for lambda expressions in Java?

Ans: The main benefit of lambda expression in Java 8 is that now 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.

6) Is it mandatory for a lambda expression to have parameters?

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

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

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