Q1 Given a list of integers, find out all the even numbers exist in the list using Stream functions?

```
List<Integer> myList = Arrays.asList(10,15,8,49,25,98,32);
    myList.stream()
      .filter(n -> n%2 == 0)
      .forEach(System.out::println);
```

Output: **10, 8, 98, 32**

Q2 Given a list of integers, find out all the numbers starting with 1 using Stream functions?

```
List<Integer> myList = Arrays.asList(10,15,8,49,25,98,32);
myList.stream()
    .map(s -> s + "") // Convert integer to String
    .filter(s -> s.startsWith("1"))
    .forEach(System.out::println);
```

Output: 10, 15

Q3 How to find duplicate elements in a given integers list in java using Stream functions?

```
List<Integer> myList = Arrays.asList(10,15,8,49,25,98,98,32,15);
Set<Integer> set = new HashSet();
myList.stream()
    .filter(n -> !set.add(n))
    .forEach(System.out::println);
```

Output: 98, 15

Q4 Given the list of integers, find the first element of the list using Stream functions?

```
List<Integer> myList = Arrays.asList(10,15,8,49,25,98,98,32,15);
myList.stream()
    .findFirst()
    .ifPresent(System.out::println);
```

Output: 10

Q5 Given a list of integers, find the total number of elements present in the list using Stream functions?

```
List<Integer> myList = Arrays.asList(10,15,8,49,25,98,98,32,15);
long count =  myList.stream()
              .count();
System.out.println(count);
```

Output: 9

Q6 Given a list of integers, find the maximum value element present in it using Stream functions?

```
List<Integer> myList = Arrays.asList(10,15,8,49,25,98,98,32,15);
int max =  myList.stream()
          .max(Integer::compare)
          .get();
```

```
        System.out.println(max);
```
Output: 98

Q7 Given a String, find the first non-repeated character in it using Stream functions?

```
        String input = "Java Hungry Blog Alive is Awesome";
        Character result = input.chars() // Stream of String
                        .mapToObj(s -> Character.toLowerCase(Character.valueOf((char) s))) // First
convert to Character object and then to lowercase
                        .collect(Collectors.groupingBy(Function.identity(), LinkedHashMap::new,
Collectors.counting())) //Store the chars in map with count
                        .entrySet()
                        .stream()
                        .filter(entry -> entry.getValue() == 1L)
                        .map(entry -> entry.getKey())
                        .findFirst()
                        .get();
        System.out.println(result);
```
Output: j

Q8 Given a String, find the first repeated character in it using Stream functions?

```
        String input = "Java Hungry Blog Alive is Awesome";
        Character result = input.chars() // Stream of String
                        .mapToObj(s -> Character.toLowerCase(Character.valueOf((char) s))) // First
convert to Character object and then to lowercase
                        .collect(Collectors.groupingBy(Function.identity(), LinkedHashMap::new,
Collectors.counting())) //Store the chars in map with count
                        .entrySet()
                        .stream()
                        .filter(entry -> entry.getValue() > 1L)
                        .map(entry -> entry.getKey())
                        .findFirst()
                        .get();
        System.out.println(result);
```
Output: a

Q9 Given a list of integers, sort all the values present in it using Stream functions?

```
        List<Integer> myList = Arrays.asList(10,15,8,49,25,98,98,32,15);
         myList.stream()
             .sorted()
             .forEach(System.out::println);
```

Output: 8 10 15 15 25 32 49 98 98

Q10 Given a list of integers, sort all the values present in it in descending order using Stream functions?

```
        List<Integer> myList = Arrays.asList(10,15,8,49,25,98,98,32,15);
```

```
myList.stream()
    .sorted(Collections.reverseOrder())
    .forEach(System.out::println);
```

Output: 98 98 49 32 25 15 15 10 8

Q11. To write a program in java to add prefix and suffix in a given String we will use StringJoiner class, a newly added in Java 8.
In the below program We will be adding "#" and "#" as a prefix and suffix in the string.

```
StringJoiner stringJoiner = new StringJoiner(",", "#", "#");
stringJoiner.add("Interview");
stringJoiner.add("Questions");
stringJoiner.add("Answers");
System.out.println("String after adding # in suffix and prefix :");
System.out.println(stringJoiner);
```

Q12. Java 8 Program to Print ten random numbers using forEach?

```
Random random = new Random();
 random.ints().limit(10).forEach(System.out::println);
```

Q13. Java 8 program to iterate a Stream using the forEach method?

```
List<String> str = Arrays.asList("Hello","Interview","Questions","Answers");
str.stream().forEach(System.out::println);
```

Q14. Java 8 program to find the Minimum number of a Stream?

```
Integer min = Stream.of(1, 2, 3, 4, 5, 6,7)
      .min(Comparator.comparing(Integer::valueOf))
      .get();
System.out.println("The Minimum number is: " + min);
```

Q15. Java 8 program to find the Maximum number of a Stream?

```
Integer max = Stream.of(1, 2, 3, 4, 5, 6,7)
                .max(Comparator.comparing(Integer::valueOf))
                .get();
   System.out.println("The Maximum number is: " + max);
```

Q16. Java 8 program to Count Strings whose length is greater than 3 in List?

```
List<String> stringList = Arrays.asList("Hello","Interview","Questions","Answers","Ram","for");

long count = stringList.stream().filter(str -> str.length() > 3).count();

System.out.println("String count with greater than 3 digit : " + count);
```

Q17. Java 8 program to Print Strings whose length is greater than 3 in List.

```
List<String> stringList =
```

```
Arrays.asList("Hello","Interview","Questions","Answers","Ram","for");
    stringList.stream().filter(str -> str.length() > 3).forEach(System.out::println);
```

Q18.Java 8 program to multiply 3 to all element in list and print the list?

```
List<Integer> integerList = Arrays.asList(1,2,3,4,5,6,7);
    System.out.println(integerList.stream().map(i -> i*3).collect(Collectors.toList()));
```

Q19. Java 8 program to perform concatenation on two Streams?

```
List<Integer> integerList1 = Arrays.asList(1,2,3,4);
    List<Integer> integerList2 = Arrays.asList(5,6,7);
    Stream<Integer> concatStream = Stream.concat(integerList1.stream(),
integerList2.stream());
    concatStream.forEach(System.out::print);
```

Q20. Java 8 program to remove the duplicate elements from the list?

```
List<Integer> integerList = Arrays.asList(1,2,3,4,1,2,3);
    System.out.println("After removing duplicates : ");
    integerList.stream().collect(Collectors.toSet()).forEach(System.out::print);
```

Q21.Print current date and time in Java 8 Date and Time API?

```
System.out.println("Current Date: " + java.time.LocalDate.now());
 System.out.println("Current Time: " + java.time.LocalTime.now());
 System.out.println("Current Date and Time: " + java.time.LocalDateTime.now());
```

Q22.Java 8 program to sort the given list?

```
List<Integer> integerList = Arrays.asList(4,5,6,7,1,2,3);
integerList.stream().sorted().forEach(System.out::println);
```

Q23. Write a Java 8 program to get the sum of all numbers present in a list?

```
List<Integer> integerList = Arrays.asList(4,5,6,7,1,2,3);
System.out.println(integerList.stream().mapToInt(Integer::intValue).sum());
```

Q24. Java 8 program to perfrom cube on list elements and filter numbers greater than 50.

```
List<Integer> integerList = Arrays.asList(4,5,6,7,1,2,3);
 integerList.stream().map(i -> i*i*i).filter(i -> i>50).forEach(System.out::println);
```

Q25. What is the easiest way to convert an array into a stream in Java 8?

```
String[] testarray = {"Hello", "Intellipaat", "learners"};
Stream numbers = Stream.of(testarray);
numbers.forEach(System.out::println);
```

Q26. What are some of the examples of terminal operations in Java 8?

Following are some of the examples of terminal operations in Java 8:
Count, Min ,Max, Reduce, toArray, anymatch, allMatch

Q27. Can you give examples of intermediate operations in Java 8?

The examples that are widely used in intermediate operations are:
Distinct(), Limit(long n), Filter(Predicate), Map(Function) skip(long n)

Q28. Can you name the common types of functional interfaces in the standard library?

There are many functional interface types in the standard library, and some of them are as follows:

BiFuction, BinaryOperator, Consumer, Predicate, Supplier , UnaryOperator

Q29. What is the difference between intermediate and terminal operations on Stream?

The intermediate Stream operation returns another Stream, which means you can further call other methods of Stream class to compose a pipeline.
For example after calling map() or flatMap() you can still call filter() method on Stream.
On the other hand, the terminal operation produces a result other than Streams like a value or a Collection.
Once a terminal method like forEach() or collect() is called, you cannot call any other method of Stream or reuse the Stream.

Q30. What is the difference between flatMap() and map() functions?

Even though both map() and flatMap() can be used to transform one object to another by applying a method to each element.
The main difference is that flatMap() can also flatten the Stream. For example, if you have a list of the list, then you can convert it to a big list by using the flatMap() function.

Q31. What does the peek() method do? When should you use it?

The peek() method of Stream class allows you to see through a Stream pipeline. You can peek through each step and print meaningful messages on the console. It's generally used for debugging issues related to lambda expression and Stream processing.

Q32. What do you mean by saying Stream is lazy?

When we say Stream is lazy, we mean that most of the methods are defined on Java .util.stream.Stream class is lazy i.e. they will not work by just including them on the Stream pipeline.
They only work when you call a terminal method on the Stream and finish as soon as they find the data they are looking for rather than scanning through the whole set of data.

Q33. What is a Predicate interface?

A Predicate is a functional interface that represents a function, which takes an Object and returns a boolean. It is used in several Stream methods like filter(), which uses Predicate to filter unwanted elements.
here is how a Predicate function looks like:

```
public boolean test(T object){
  return boolean;
}
```

You can see it just has one test() method which takes an object and returns a boolean. The

method is used to test a condition if it passes; it returns true otherwise false.

Q34. What are Supplier and Consumer Functional interface?
The Supplier is a functional interface that returns an object. It's similar to the factory method or new(), which returns an object.
The Supplier has get() functional method, which doesn't take any argument and return an object of type T. This means you can use it anytime you need an object.
Since it is a functional interface, you can also use it as the assignment target for a lambda expression or method reference.
A Consumer is also a functional interface in JDK 8, which represents an operation that accepts a single input argument and returns no result.
Unlike other functional interfaces, Consumer is expected to operate via side-effects. The functional method of Consumer is accept(T t), and because it's a functional interface, you can use it as the assignment target for a lambda expression or method interface in Java 8.

Q35. What is the parallel Stream? How can you get a parallel stream from a List?
A parallel stream can parallel execute stream processing tasks. For example, if you have a parallel stream of 1 million orders and you are looking for orders worth more than 1 million, then you can use a filter to do that.
Unlike sequential Stream, the parallel Stream can launch multiple threads to search for those orders on the different parts of the Stream and then combine the result.

Q36. What is the meaning of a Spliterator in Java 8?
Spliterator is a newly introduced iterator interface for Java 8. It is very efficient and handles API-related operations seamlessly across the runtime environment.

Q37. Differentiate between Spliterator and a regular iterator in Java 8.

Spliterator	**Iterator**
Introduced with Java 8	Present since Java 1.2
Used in Stream API	Used in Collection API
Helps in the iteration of streams in a parallel and sequential order	Iterates collections in a sequential order only
Example: tryAdvance()	Examples: next() and hasNext()

Q38. When is an ideal situation to use the Stream API in Java 8?
The Stream API in Java 8 can be effectively used if the Java project calls for the following operations:

- Perform database operations
- Execute operations lazily
- Write functional-style programming
- Perform parallel processing
- Use pipeline operations
- Use internal iteration

Q39. What is the easiest way to print the sum of all of the numbers present in a list using Java
List<Integer> numbers = Arrays.asList(5, 4, 10, 12, 87, 33, 75);

```
IntSummaryStatistics stats = integers.stream().mapToInt((x) —> x).summaryStatistics();
System.out.println("Sum of all numbers : " + stats.getSum());
```

Q40. What is the use of the optional keyword in Java 8?

The optional keyword is used in Java 8 to avoid the occurrence of the NullPointerException. An optional object can be created easily using the empty() static method as shown below:

```
@Test
public void whenCreatesEmptyOptional_thenCorrect() {
        Optional<String> empty = Optional.empty();
        assertFalse(empty.isPresent());
}
```

Q41. What is stream pipelining used for?

Stream pipelining is a concept that is implemented in Java 8 so that users can chain more than one operation at a time. This works on the principle of splitting the operation into two categories:

Intermediate operations: Return the instance of the stream when running

Terminal operations: Used to terminate the operation and return the final value

Q42. Differentiate between intermediate and terminal operations in Java8?

Intermediate Operation	**Terminal Operation**
Used for the transition to a new state	Used to end the process under execution
Lazy execution of code, i.e., code is not executed as soon as it is encountered	Not lazy; code is immediately executed upon encounter

Q43. Differentiate between a predicate and a function in Java 8.

Predicate	**Function**
Returns True or False	Returns an object
Used as an assignment target for lambda expressions	Can be used for both lambda expressions and method references

Q44. Can a functional interface extend/inherit another interface?

A functional interface cannot extend another interface with abstract methods as it will void the rule of one abstract method per functional interface. E.g:

```
interface Parent {
public int parentMethod();
}
@FunctionalInterface // This cannot be FunctionalInterface
interface Child extends Parent {
public int childMethod();
// It will also extend the abstract method of the Parent Interface
// Hence it will have more than one abstract method
// And will give a compiler error
}
```

Q45. What are static methods in Interfaces?

Static methods, which contains method implementation is owned by the interface and is invoked using the name of the interface, it is suitable for defining the utility methods and cannot be overridden.

Q46. What is the default method, and why is it required?
A method in the interface that has a predefined body is known as the default method. It uses the keyword default. default methods were introduced in Java 8 to have 'Backward Compatibility in case JDK modifies any interfaces. In case a new abstract method is added to the interface, all classes implementing the interface will break and will have to implement the new method. With default methods, there will not be any impact on the interface implementing classes. default methods can be overridden if needed in the implementation. Also, it does not qualify as synchronized or final.

```
@FunctionalInterface // Annotation is optional
public interface Foo() {
// Default Method - Optional can be 0 or more
public default String HelloWorld() {
return "Hello World";
}
// Single Abstract Method
public void bar();
}
```

Q47. What are the most commonly used Intermediate operations?
Filter(Predicate<T>) - Allows selective processing of Stream elements. It returns elements that are satisfying the supplied condition by the predicate.

map(Funtion<T, R>) - Returns a new Stream, transforming each of the elements by applying the supplied mapper function.= sorted() - Sorts the input elements and then passes them to the next stage.

distinct() - Only pass on elements to the next stage, not passed yet.

limit(long maxsize) - Limit the stream size to maxsize.

skip(long start) - Skip the initial elements till the start.

peek(Consumer) - Apply a consumer without modification to the stream.

flatMap(mapper) - Transform each element to a stream of its constituent elements and flatten all the streams into a single stream.

Q48. What is the stateful intermediate operation? Give some examples of stateful intermediate operations.
To complete some of the intermediate operations, some state is to be maintained, and such intermediate operations are called stateful intermediate operations. Parallel execution of these

types of operations is complex.

For Eg: sorted() , distinct() , limit() , skip() etc.

Sending data elements to further steps in the pipeline stops till all the data is sorted for sorted() and stream data elements are stored in temporary data structures

Q49. What is the most common type of Terminal operations?

collect() - Collects single result from all elements of the stream sequence.

reduce() - Produces a single result from all elements of the stream sequence

- **count**() - Returns the number of elements on the stream.
- **min**() - Returns the min element from the stream.
- **max**() - Returns the max element from the stream.

Search/Query operations

anyMatch() , noneMatch() , allMatch() , ... - Short-circuiting operations.
Takes a Predicate as input for the match condition.
Stream processing will be stopped, as and when the result can be determined.

Iterative operations

forEach() - Useful to do something with each of the Stream elements. It accepts a consumer.
forEachOrdered() - It is helpful to maintain order in parallel streams