What are interfaces in typescript ?

Ans: interfaces in typescript are similar to the interfaces in many object oriented programming languages. Here Objects don't have to explicitly implement interfaces as you would in C# or Java. Instead, interfaces define the expected properties so that the type checker can verify an object with the expected properties is being used.

_____1

How do classes in typescript implements interfaces ?

Ans: Typescript classes implements the interfaces similarly as they are implemented in c#/ java. By using implements keyword
typescript classes implements the interfaces.

Ex:

```
class Sample implements ISample{
   //your code here
}
```

_____2

What are optional properties ?

Ans: Optional properties can be defined by adding a question mark (?) after the property name when defining the property
in the interface. This is the way of expressing property that may be present, but it still treats that object as that type
if optional property is missing.

_____3

What are Function Types ?

Ans: In addition to describing an object with properties, interfaces are also capable of describing function types.
To describe a function type with an interface, we give the interface a call signature. This is like a function declaration
with only the parameter list and return type given. Each parameter in the parameter list requires both name and type.

_____4

what are hybrid types ?

Ans: Objects which can operate both as function and as an object are known as Hybrid Types.

_____5

What are indexable interfaces ?

Ans: Interface can be used to describe that indexing into an object always produces values of a certain type. You can define
the array in the iterface and then implement that in the class.

_____6

How do you extend an inteface into another interface in typescript ?

Ans: Like wise we know that we can extend an interface to a class, similarly an interface can also extend another, it's
similar to what we do in many programming languages like c#/ java.

Ex:

```
interface If1{
add(a:number,b:number): number
}

inteface If2{
sub(a:number,b:number): number
}

interface IArithmeticOperations extends If1,If2{

}
```

_____7

What are rest paramters ?

Ans: It's nothing but grouping multiple parameters into a single variable. By using (...) three dots we can achieve this.
And this technique is also called as Spread.. In JavaScript, you can work with the arguments directly using the arguments variable that is visible inside every function body.

Ex:

```
function sample(a: number, ...b:number[]): number{
        //your code here
}

console.log(sample(10,20,30,40));
```

Here in this example a will be 10, and the rest values 20,30,40 will comes under b which is an array.

_____8

How to do function overloading in tyepscript ?

Ans: Actually JavaScript does not allow overloading, so TypeScript cannot generate multiple definitions of the same function
differing only by their signature. TypeScript does not accept the two different constructors too(constructor overloading)

But if you still want to do function overloading, we need to supply multiple function types for the same function as a list of overloads.
This list is what the compiler will use to resolve function calls.

Ex:

```
function add(x: string, y: string): string;
function add(x: number, y: number): number;
function add(x: boolean, y: string): number; //Function Implementation
function add(x, y): any {
        //your implentation here
}
console.log(add(10, 20))     //valid
console.log(add("A", "B"))   //valid
console.log(add(true, "B"))  //valid
console.log(add(true,false)) //invalid
```

_____9

What is duck typing in typescript ?

Ans:

1. The core principle of typescript type-checking focussing on the shape that values have is called as duck-typing (or)
structural-subtyping.
2. In TypeScript, interfaces fill the role of naming these types, and are a powerful way of defining contracts within your code as well as contracts with code outside of your project.

Ex:

```
interface ISample{
   name :string;
}

function printLabel(sample: ISample){
   console.log(sample.name);
}

let myObj = {age: 41, name: "Sandeep Soni"};
printLabel(myObj);
```

_____10