**Why are closures useful in JavaScript? Give an example use case.**

Closures in JavaScript allow you to access variables in the inner scope from an outer function's scope. It provides a powerful way of managing global namespace. Essentially, you can encapsulate data within the function and therefore, provide controlled access to that data by keeping it hidden from the outer scope. This also enables the creation of higher order functions.

Example –

```
function foo(outer_argument)
{
        function innerFunction (inner_argument) {
                return outer_argument * inner_argument;
                }
                return inner_argument;
}

let getInnerFunc = foo (9);

console.log(getInnerFunc(5));
console.log(getInnerFunc(10));
```

In the above example, we can access the outer_argument from the inner function even before being done with the execution of foo(9). And, on the execution, produce the multiplicative answer of outer_argument and inner_argument as desired.

**When should you choose to use "let" or "const"**

"let" allows new values to be assigned to variables multiple times whereas "const" variables cannot be a reassigned a value once initialized. You would choose "let" when you expect a variable to change value (example – value changes based on user input) and you would choose "const" when you expect a variable to stay constant and predictable (example – assigning the value of gravitational constant to a "const" variable 'g').

**Give an example of a common mistake related to hoisting and explain how to fix it.**

A common mistake related to hoisting is using variables before declaring them. To avoid this mistake, one must always declare the variables before using them. Another mistake through extension is thinking var, let and const are pretty much the same in behaviour when it comes to hoisting. To fix this, we must prioritize using let and const over var to avoid unnecessary hoisting.

Example –

```
hello();
var hello = function () {
        console.log ("Hi There!");
};
```

The above function expression will lead to an error because even if the function is hoisted, it's value as a function expression is not and so the result will be 'undefined'. To correct this, simply call hello(); after the function expression.

**What will the outcome of each console.log() be after the function calls? Why?**

```
const arr = [1, 2];
function foo1(arg) {
  arg.push(3);
}
foo1(arr);
console.log(arr);    ->  Solution -> [1,2,3]

function foo2(arg) {
  arg = [1, 2, 3, 4];
}
foo2(arr);
console.log(arr);  -> Solution -> [1,2,3]

function foo3(arg) {
  let b = arg;
```

```
  b.push(3);
}
foo3(arr);
console.log(arr);  -> Solution -> [1,2,3,3]

function foo4(arg) {
  let b = arg;
  b = [1, 2, 3, 4];
}
foo4(arr);
console.log(arr);  -> Solution -> [1,2,3,3]
```