- Why are closures useful in JavaScript? Give an example use case.
  - Closures are used in JavaScript so that we can keep some variables for future use in that function. We can also keep variables hidden and expose the operation we want.

```
function createCounter(initial = 0) {
  let count = initial;

  return {
    increment() {
      count += 1;
      return count;
    },
    getValue() {
      return count;
    }
  };
}

const counter = createCounter(5);
console.log(counter.getValue());
console.log(counter.increment());
console.log(counter.increment());
```

- When should you choose to use “let” or “const”
  - Should try to use const when possible. Only us let when we want to reassign value to that variable
- Give an example of a common mistake related to hoisting and explain how to fix it.
  - console.log(count) // undefined
  - var count = 5;
  - Should bring the variable declaration to the top before accessing
- 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); // [1, 2, 3], as arr is passed to foo1 as reference, arg.push(3) just push to the array itself

function foo2(arg) {
  arg = [1, 2, 3, 4];
}
foo2(arr);
console.log(arr); // [1, 2, 3], arg = [1, 2, 3, 4] only change the argument, not the array
```

```
function foo3(arg) {
  let b = arg;
  b.push(3);
}
foo3(arr);
console.log(arr); // [1, 2, 3, 3], b = arg is reference, b still points to the
original array, pushing to the array changes it

function foo4(arg) {
  let b = arg;
  b = [1, 2, 3, 4];
}
foo4(arr);
console.log(arr); // [1, 2, 3, 3], b = arg, now b points to a different array, the
original array does not change
```