1. Find output

```
function outer() {
var x = 10;

  function inner() {
    console.log(x);
    var x = 20;
  }

  return inner;
}

var closureFunc = outer();
closureFunc();
```

```
Output : undefined.
```

2. Find output

```
function createFunctions() {
  var result = [];

  for (let i = 0; i < 5; i++) {
    result.push(function() {
      console.log(i);
    });
  }

  return result;
}

var functions = createFunctions();
functions.forEach(fn => {
  fn();
})
```

```
Output: 0
1
2
3
4
```

3. Implement a function that generates a sequence of unique IDs, starting from the given number

Ans. function createSequentialIdGenerator(num){

```
  function f1(){
          num = num+1;
          return (num);
  }
  return f1
}

const generateUniqueId = createSequentialIdGenerator(999);

console.log(generateUniqueId()); // Expected output: 1000
console.log(generateUniqueId()); // Expected output: 1001
console.log(generateUniqueId()); // Expected output: 1002
```

4. 
```
function swapKeyAndValues(obj) {
  // Your code here
}

const sampleObject = {
  key1: 'value1',
  key2: 'value2',
  key3: 'value3'
};

swapKeyAndValues(sampleObject);
console.log(sampleObject);

// Expected output:
{
  value1: 'key1',
  value2: 'key2',
  value3: 'key3'
}
```

```
Ans. function swapKeyAndValues(obj) {
  // Your code here
  for(key in obj){
      obj[obj[key]] = key;
  //delete key;
      delete obj[key];
  }
}
```

```
const sampleObject = {
  key1: 'value1',
  key2: 'value2',
  key3: 'value3'
};

swapKeyAndValues(sampleObject);
console.log(sampleObject);
```

5. Find whether all students in the class are passed in the exam
   Rule: Passed - If average marks of a student > 40 else failed

```
const students = [
  { name: 'John', marks: [70, 85, 90] },
  { name: 'Jane', marks: [60, 75, 80] },
  { name: 'David', marks: [50, 55, 65] }
];

function checkAllStudentsPassed(studentsArr) {
  // Your code here
}

const allStudentsPassed = checkAllStudentsPassed(students);

console.log(allStudentsPassed);
```

```
Ans. const students = [
      { name: 'John', marks: [70, 85, 90] },
      { name: 'Jane', marks: [60, 75, 80] },
      { name: 'David', marks: [50, 55, 65] }
  ];

  function checkAllStudentsPassed(studentsArr) {
      // Your code here
      // for (student in studentsArr){
      //   var tot = 0;
      //   for (m in student.marks){
      //    tot += m;
      //    var avg = tot/3;
      //   }
      //   if (avg < 40){
      //    return false;
      //   }
      // }
      // return true;
      function findtot(tot,num){
      return tot+num;
      }
      function find_avg(stobJ){
      //console.log(stobJ);
      var markarr = stobJ.marks;
      var total = markarr.reduce(findtot);
      var avg = total/3;
      //console.log(avg);
      return avg;
      }
      var avg_array = studentsArr.map(find_avg);
      console.log(avg_array);
      //avg_array.

  }

  const allStudentsPassed = checkAllStudentsPassed(students);

  console.log(allStudentsPassed); // Output: true
```

6. Rewrite the below code snippet using async/await

```
function getProcessedData(url) {
      return downloadData(url)
      .catch(e => {
                  return downloadFallbackData(url)
      })
      .then(value => {
                  return processDataInWorker(value)
      })
}
```

Ans.

```
async function getProcessedData(url){
      try{
      const data = await downloadData(url);
      return processDataInWorker(data);
      }
      catch(e){
      try{
```

```
                const msg = await downloadFallbackData(url);
                return processDataInWorker(msg);
        }
        catch(e){
                ;
        }
        }
}
```

7. Implement Retry method using promise

```
function simulateAsyncTask() {
  return new Promise((resolve, reject) => {
    const randomNumber = Math.random();
    setTimeout(() => {
      if (randomNumber < 0.8) {
        resolve('Success');
      } else {
        reject('Error: Task failed');
      }
    }, 500);
  });
}

function retry() {
  // Your code here
}

// Sample invocation
retry(simulateAsyncTask, 3)
  .then(result => console.log('Result:', result))
  .catch(error => console.log('Error:', error));
```

```
Ans. function simulateAsyncTask() {
  return new Promise((resolve, reject) => {
       const randomNumber = Math.random();
       console.log(randomNumber);
       setTimeout(() => {
       if (randomNumber < 0.08) {
       resolve('Success');
       } else {
       reject('Error: Task failed');
       }
       }, 500);})}
```

```
function retry(task,num) {
const p = new Promise((resolve,reject)=>{
      const p2 = task();
      p2.then(resValue=>{
      //console.log(resValue);
      resolve(resValue);
      },error=>{
      //console.log(error);
      if (num <= 0){
            reject(error);
      }
      else{
            num -= 1;
            //console.log(num);

            retry(task,num).then(res=>resolve(res),err=>reject(err));
      }

      });
})
return p;
}

// Sample invocation
retry(simulateAsyncTask, 3)
  .then(result => console.log('Result:', result))
  .catch(error => console.log('Error:', error))
```

8. Implement retry method using async await

```
Ans. function simulateAsyncTask() {
      return new Promise((resolve, reject) => {
      const randomNumber = Math.random();
      console.log(randomNumber);
      setTimeout(() => {
      if (randomNumber < 0.08) {
            resolve('Success');
      } else {
            reject('Error: Task failed');
      }
      }, 500);})}

//   function retry(task,num) {
```

```
//   const p = new Promise((resolve,reject)=>{
//    const p2 = task();
//    p2.then(resValue=>{
//         //console.log(resValue);
//         resolve(resValue);
//    },error=>{
//         //console.log(error);
//         if (num <= 0){
//              reject(error);
//         }
//         else{
//              num -= 1;
//              //console.log(num);

//
retry(task,num).then(res=>resolve(res),err=>reject(err));
//         }

//    });
//   })
//   return p;
//   }

  async function retry(task,num){
     try{
     const res = await task();
     console.log(res);
     return res;
     }
     catch(e){
     if (num <= 0)
     {
          console.log(e);
          return(e);
     }
     else
     {
          num -= 1;
          try{
               const resV = await retry(task,num);
               return resV;
          }
          catch(e){
               return e;
          }
     }
     }
  }

  // Sample invocation

  async function doo(){
     try{
     const res = await retry(simulateAsyncTask,3);
     console.log(res);
     }
     catch(e){
     console.log(e);
     }
  }
//   retry(simulateAsyncTask, 3)
//    .then(result => console.log('Result:', result))
//    .catch(error => console.log('Error:', error))
```