```
// Q-1 Which of the following is a deep clone?

// A
let cloned = JSON.parse(JSON.stringify(objectToClone));

// B
let cloned = objectToClone


 Options:
1) Both
2) A
3) None
4) B

Q2- find output:

let p = new Promise(function (resolve, reject) {
  reject(new Error("some error"));
  setTimeout(function(){
    reject(new Error("some error"));
  },1000)
  reject(new Error("some error"));
});

p.then(null, function (err) {
  console.log(1);
  console.log(err);
}).catch(function (err) {
  console.log(2);
  console.log(err);
});


options:

1)

1
2

2)

1
Error: some error
2
Error: some error

3)

1
Error: some error

4)

Error: some error
Error: some error
```

Q-3 find output:

```
let p = new Promise(function (resolve, reject) {
  setTimeout(function () {
    reject(new Error("some value"));
  }, 2000);

  resolve("some error");

  setTimeout(function () {
    reject(new Error("some value"));
  }, 2000);

  resolve("some error");

  setTimeout(function () {
    reject(new Error("some value"));
  }, 2000);
});


p.then(null, function (err) {
  console.log(1);
  console.log(err);
});

p.catch(function (err) {
  console.log(2);
  console.log(err);
});

p.finally(function () {
  console.log(1);
})

p.finally(function () {
  console.log(2);
}).then(function (val) {
  console.log(val);
})

p.then(
  function (val) {
    console.log(val);
  },
  function (err) {
    console.log(err);
  }
);
```

Options:

1) Error

2)

```
1
2
some error
some error

3) Not work

4)

1
2
some value
some value

Q4- find output

function f(x) {
    return x*x;
}

f.description = "This function returns the square"

let arr = [1, 2, 3, 4, 5]
console.log(arr.length);
arr.length = 6
arr.pop()
console.log(arr.length);
console.log(f);
console.log(f());
console.log(f());

// options:
// 1)
    // 5
    // 5
    // { [Function: f] description: 'This function returns the square' }
    // NaN
    // NaN

// 2)
    // 5
    // 6
    // { [Function: f] description: 'This function returns the square' }
    // NaN
    // NaN

// 3)
    // 5
    // 5
    // [Function: f]
    // NaN
    // NaN

// 4)
    // 5
    // 6
```

```
    // [Function: f]
    // NaN
    // NaN
```

Q-5 Create a function which deep clones the object

Q-6 find the output of the following:

```
console.log(1);

setTimeout(function () {
  console.log(3);
});

console.log(4);

setTimeout(function () {
  console.log(2);
});

Promise.resolve().then(function () {
  console.log(5);
});

console.log(6);
```

options:

1)

```
    1
    4
    6
    5
    3
    2
```

2)

```
    1
    3
    4
    2
    5
    6
```

3)

```
    1
    4
    6
    3
    2
    5
```

4)

1
3
4
2
Error

/Q-7 Without changing the structure of the code move inputs of console statements so that the output ordering is in increasing rather than jumbled

```
setTimeout(function () {
  console.log(1);
});
setTimeout(function () {
  console.log(2);
});

let p = new Promise(function (resolve, reject) {
  resolve();
});

console.log(3);

p.then(function () {
  console.log(4);
});

p.then(function () {
  console.log(5);
});

setTimeout(function () {
  console.log(6);
});
```

Q-8 find output

```
try {
  let a = null;

  b = a;

  delete a;

  b = undefined;

  console.log(a);
  console.log(b);
  console.log(c);
} catch (err) {
  console.log(err.message);
}
```

options:

1)

Error

2)

null
undefined
undeclared

3)

undefined
undefined
Error

4)

null
undefined
Error

//Q-9 Write a function that takes 2 number => num1 and num2.
The function checks if num1 is even and is divisible by num2 if both conditions are true then it returns the remainder of num1/num2 else it throws an error with the message incompatible types