1. On the difference between global scope and local scope

```
var a = 1;

function foo() {
  a = 2;
}

foo();
console.log(a);
```

- This returns 2, since a will first loook in the local scope, see that a has not been defined, and will look up in the global scope, finds a global a, and this is redefined

```
var a = 1;

function foo() {
  var a = 2;
}

foo();
console.log(a);
```

- This returns 1, since the function foo() will first look in the local scope, and return the local value of a first

```
function foo() {
  var a = 2;
}

foo();
console.log(a);
```

- In this case, there's no global a, then it's locally created and assigned 2

```
var a = 1;

function foo() {
  var a = 2;
  console.log(a);
}

function bar() {
  a = 3;
  console.log(a);
}

foo();
bar();
console.log(a);
```

- This will return 2, 3, 3

- foo() will run, where a will be found in the local scope, so a is returned
- bar() will run, a is not found in the local scope, it will go up in the global scope and finds a a, a is then reassigned to 3, and logged
- console.log(a) will return a in the global scope, and since it's been reassigned to 3, 3 is returned

2. On function scoping

```
const message = "hi";
console.log(message);
```

- The const is available as a global var

```
{
    const message = "hi";
}
console.log(message);
```

- The const is scoped locally and not available outside

```
function start() {
  const message = "hi";
}
console.log(message);
```

- Returns "message is not returned", since const is only defined within the block

- Trying to access a variable outside of an if() or for() block will create the same reference error

```
function start() {
  const message = "hi";

  if (true) {
    const another = "bye";
  }

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

  console.log(i);
}
start();
```

- Local var/const take precendence over global var/const

```
const color = 'red';

function start() {
  const message= "hi";
  const color = 'blue';
  console.log(color);
}
```

```
function stop(){
  const message="bye";
}

start();
```

- The above returns `blue`

3. Object shadows and coersions (example)[https://stackoverflow.com/questions/4750225/what-does-object-object-mean]

- There are 5 primitive types in JS: `null`, `string`, `boolean`, `undefined`, and `number`
- Three of these primitive types have object counterparts: `string`, `boolean`, and `number`, this means that the primitives can be coersed into their object counterparts
- The objects are instances of the `String`, `Number`, and `Boolean` constructors
- How do primitives have access to the methods of their object counterparts? JS coerses the primitive types to their object counterparts when required

```
var myObj = {lhs: 3, rhs: 2};
var myFunc = function(){}
var myString = "This is a sample String";
var myNumber = 4;
var myArray = [2, 3, 5];
var myUndefined = undefined;
var myNull = null;

Object.prototype.toString.call(myObj);    //"[object Object]"
Object.prototype.toString.call(myFunc);    //"[object Function]"
Object.prototype.toString.call(myString);    //"[object String]"
Object.prototype.toString.call(myNumber);    //"[object Number]"
Object.prototype.toString.call(myArray);    //"[object Array]"
Object.prototype.toString.call(myUndefined);    //"[object Undefined]"
Object.prototype.toString.call(myNull);    //"[object Null]"
```