**Q.1 Arrays function**

```
var arr1=[1,2,3,4];

var arr2=[5,6,7,8];
```

**Concat:**

```
//<p id="demo"></p>

var concatArray=arr1.concat(arr2);

console.log(concatArray);

o/p:  [1, 2, 3, 4, 5, 6, 7, 8]
```

**every**

```
function check(num){

  return num>=3;

}

console.log("Every: "+arr2.every(check));

o/p:rue
```

**Filter()**

```
function check(num){

  return num>=3;

}

console.log("Every: "+arr1.filter(check));

o/p:3,4
```

**forEach()**

```
var m=1;

arr1.forEach(mul);

function mul(num){

  console.log("mul"+m*num);
```

```
 m=m*num;
}
```

o/p:

```
mul1
 mul2
 mul6
 mul24
```

**indexOf()**

```
console.log(arr1.indexOf(3));
```

o/p:2

**join()**

```
var a=arr2.join();
console.log(a);
```

o/p:5,6,7,8

**lastIndexOf()**

```
var arr3=[1,2,3,1,4,5,3,4];
console.log(arr3.lastIndexOf(1));
```

o/p:3

**map()**

```
var newArr=arr1.map(pass);
function pass(num){
  return num*5;
}
console.log(newArr);
```

```
o/p:5 10 15 20
```

**pop()**

```
var arr3=[1,2,3,1,4,5,3,4];

arr1.pop();

console.log(arr1);

o/p:1 2 3
```

**push()**

```
arr1.push(4);

console.log(arr1);

o/p: 1 2 3 4 4
```

**reduce()**

```
var val=arr1.reduce(fun);

function fun(total,num){

  return total-num;

}

console.log(val);

o/p: -8
```

**reduceRight()**

```
var val=arr1.reduceRight(fun);

function fun(total,num){

  return total-num;

}

console.log(val);

o/p: -2
```

**reverse()**

```
arr1.reverse();

console.log(arr1);

o/p: 4 3 2 1
```

**shift()**

```
arr1.shift();

console.log(arr1);

o/p: 2 3 4
```

**slice()**

```
var tm=arr1.slice(0,2);

console.log(tm);

o/p:1 2
```

**some()**

```
var tm=arr1.some(fn);

function fn(num){

  return num>=3;

}

console.log(tm);

o/p:true
```

**splice()**

```
arr1.splice(1,0,5);

console.log(arr1);

o/p:1 5 2 3 4
```

**toString()**

```
var tm=arr1.toString();

console.log(tm);
```

**unshift()**

arr1.unshift(10);

console.log(arr1);

o/p:10 1 2 3 4

**sort()**

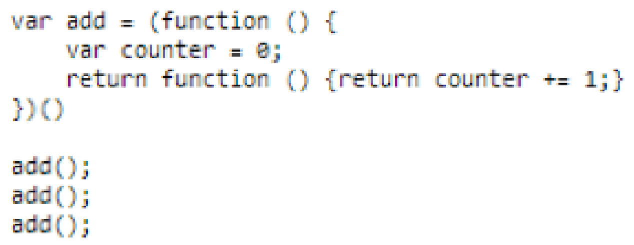
var arr3=[1,2,3,1,4,5,3,4];

arr3.sort();

console.log(arr3);

o/p: 1 1 2 3 3 4 4 5

**Q.2**



```
var add = (function () {
    var counter = 0;
    return function () {return counter += 1;}
})()

add();
add();
add();
```

**Ans:**

Output=3.

Example is based on the “closure”. We could have defined the count variable without the use of nested function but other code can also change the counter without calling add(). So counter should be local to add function.

JavaScript supports nested functions and have access to the scope above them.

The var add is assigned the return value of self-invoking function.

Because self-invoking function only runs once, it initialize the value of counter to zero and returns expression.

Add() function is called three times and it'll give output by adding one to the counter in parent scope.

**3. Difference between '\n ' and '\r'.**

**\r** is carriage return, which moves where are you typing on the page back to the left.

**\n** is new line, which moves page up a line.

**\n** acts as an end of line terminator in UNIX.

**\r** acts as an end of line terminator in MAC.

**\r\n** acts an end of line terminator in WINDOWS and DOS.