Javascript Assignment

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

   The output for this function is 3 because this is a self-invoking function and runs only once .The counter is changed only when the add function is called.

2. A.) Every():

```
> let result = [10, 5, 20, 100].every(function(number){
      return number < 150
  });
<- undefined
> result
<- true
```

   B.) Concat():

```
> arr1=[1,3,5,7,9];
<- ▶ (5) [1, 3, 5, 7, 9]
> a=[2,4,6,7,9];
<- ▶ (5) [2, 4, 6, 7, 9]
> arr1.concat(a);
<- ▶ (10) [1, 3, 5, 7, 9, 2, 4, 6, 7, 9]
```

   C.) Filter():

```
let ans=[10,5,20,100].filter(function(ayy){
return ayy<50
});
undefined
ans;
▶ (3) [10, 5, 20]
```

D.) IndexOf():

```
> let str="This is Snigdha";
  str.indexOf("me");
<- -1
> let str="This is Snigdha";
  str.indexOf("is");
<- 2
```

E.) Join():

```
> let arr=["How","are","you","?"]
  arr.join(" ");
<- "How are you ?"
```

F.) LastIndexOf():

```
> let str="This is Snigdha";
  str.lastIndexOf("is");
<- 5
```

G.) Map():

```
> let arr=[1,2,3,4,5,6,7]
  newarr=arr.map(function(numb){
  return numb*2
  });
<- ▶ (7) [2, 4, 6, 8, 10, 12, 14]
```

H.) Pop():

```
> let arr1=[1,2,3,4,5,6]
  arr1.pop();
<- 6
```

i.)Push():

```
> let s=["A","B","C","D","E","F"]
  s.push("G");
<- 7
> s;
<- ▶ (7) ["A", "B", "C", "D", "E", "F", "G"]
```

J.) Reverse():

```
> s;
◂ ▸(7) ["A", "B", "C", "D", "E", "F", "G"]
> s.reverse();
◂ ▸(7) ["G", "F", "E", "D", "C", "B", "A"]
```

K.) Shift():

```
> s;
◂ ▸(6) ["F", "E", "D", "C", "B", "A"]
> s.shift();
◂ "F"
> s;
◂ ▸(5) ["E", "D", "C", "B", "A"]
```

L.) Slice():

```
> let b=[1,2,3,4,5,6,7,8,9]
  b.slice(1);
◂ ▸(8) [2, 3, 4, 5, 6, 7, 8, 9]
```

M.) Some():

```
> let b=[1,2,3,4,5,6,7,8,9]
  r=b.some(function(num){
  return num<5
  });
◂ true
> r;
◂ true
```

N.) Sort():

```
> let c=[4,3,6,5,8,1,9,2];
◂ undefined
> c.sort();
◂ ▸(8) [1, 2, 3, 4, 5, 6, 8, 9]
```

O.) Splice():

```
> c;
◂ ▸(8) [1, 2, 3, 4, 5, 6, 8, 9]
> c.splice(2,4);
◂ ▸(4) [3, 4, 5, 6]
```

P.) ToString():

```
> c;
◂ ▸(4) [1, 2, 8, 9]
> c.toString();
◂ "1,2,8,9"
```

Q.) Unshift():

```
> let d=[3,4,5,6,7,8,9]
  d.unshift(1,2);
◂ 9
> d;
◂ ▸(9) [1, 2, 3, 4, 5, 6, 7, 8, 9]
>
```

3.

```
> function solve(str){
  var re=new RegExp(/^lion/);
  var re1=new RegExp(/cat$/);
  var re3=new RegExp(/ab+c/);
  if(re.test(str)){
  return true+"\n"+str.search("lion"); }
  else if(re1.test(str)){
  return true+"\n"+str.search("cat"); }
  else if(re2.test(str)){
  return true+"\n"+str.search("ab/+c");}
  else {
  return false ;}
  };
◂ undefined
> solve("Rahul is having a cat");
◂ "true
  18"
```

4.

```
> function sol(arr){
  var b=[];
  var c=[]
  var a=arr.sort();
  for(i=0;i<a.length;i++){
  c[i]=a[i]*10;
  if(a[i]%3==0){
  b.push(a[i]);}
  };
    return a+"\n"+b +"\n"+c
  };
< undefined
> sol([2,5,1,4,9,6,8,7,3]);
< "1,2,3,4,5,6,7,8,9
  3,6,9
  10,20,30,40,50,60,70,80,90"
```

5. == : This checks whether two operands are same or not. It does not checks their data types but just the value.
   Eg. 1==”1” // False

   === : This checks whether the two operands as well as their data types is equal or not.
   Eg. 1===”1” // True