AU Javascript Assignment

Arhan Relan

1. Implementation of all array functions

```
> var a1 = ["Ram", "Shyam", "Geeta", "Sita"]
  var a2 = ["Rahul", "Raj", "Nishtha", "Muskan"]
  var a3 = [1,2,3,4]
  var a4 = [5,6,7,8]
  a1.concat(a2);
◂ ▶ (8) ["Ram", "Shyam", "Geeta", "Sita", "Rahul", "Raj", "Nishtha", "Muskan"]
> a4.every((currVal)=>currVal >4)
◂ true
> a3.filter((currVal)=> currVal>=2)
◂ ▶ (3) [2, 3, 4]
> a2.forEach(i => console.log(i))
  Rahul                                                    VM928:1
  Raj                                                      VM928:1
  Nishtha                                                  VM928:1
  Muskan                                                   VM928:1
◂ undefined
> a1.indexOf("Sita")
◂ 3
> a4.join()
◂ "5,6,7,8"
> a4.push(6)
◂ 5
> a4.lastIndexOf(6)
◂ 4
> a4.map((i)=> {return ++i})
◂ ▶ (5) [6, 7, 8, 9, 7]
> a4.pop
◂ ƒ pop() { [native code] }
> a4.pop()
◂ 6
```

```
> var temp=a3.reduce(function(acc,curr){return acc+curr;})
  console.log(temp);
  10
  undefined
> var temp2=a4.reduceRight(function(acc,curr){return acc-curr;});
  console.log(temp2);
  -10
  undefined
> a1.reverse()
  ▶ (4) ["Sita", "Geeta", "Shyam", "Ram"]
> a1.shift()
  "Sita"
> a1.unshift("Pappu")
  4
> a1
  ▶ (4) ["Pappu", "Geeta", "Shyam", "Ram"]
> a4.some((currVal)=>currVal>10)
  false
> a4
  ▶ (4) [5, 6, 7, 8]
> a1
  ▶ (4) ["Pappu", "Geeta", "Shyam", "Ram"]
> a1.slice(1,3)
  ▶ (2) ["Geeta", "Shyam"]
> a2
  ▶ (4) ["Rahul", "Raj", "Nishtha", "Muskan"]
> a2.splice(2,2)
  ▶ (2) ["Nishtha", "Muskan"]
> a2
  ▶ (2) ["Rahul", "Raj"]
> a2.sort()
  ▶ (2) ["Rahul", "Raj"]
> a1.sort()
  ▶ (4) ["Geeta", "Pappu", "Ram", "Shyam"]
> a2.toString()
  "Rahul,Raj"
>
```

2. Explanation of the output of the following code

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

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

This code returns the following output:

3

It can be seen that since add () function is called three times, it is simply an implementation of a counter. Since it is a self-invoking function, it runs only once it returns the expression inside the body of the returned function whenever add () is called.

In this way, we are able to initialise the value of counter as 0 only once while ensuring that the value of the counter can not be changed arbitrarily and only by using the add () function. This type of function called **Javascript Closure** can ensure that certain function variables can be **private**, in this case counter, and yet be shared by functions using the self-invoking function add ().

3. Differences between \n and \r.

- \n stands for newline character whereas \r stands for carriage return.
- As the name suggests, \n moves to the start of a new line explicitly whereas \r moves to the start of the ongoing line only and not a new line implicitly due to space constraints.
- Owing their terminology to the working of typewriters, the carriage returned to the rightmost position after reaching the end whereas entered to a new line on moving up the paper roll. Thus even in OS nowadays \r is followed by a \n.
- In C++ and most languages \n is defined as the new line character.
- Different operating systems handle new line differently. Windows uses \n\r to signify that the enter key was pressed whereas Linux and modern Mac OS use \n to convey the same.
- Nowadays \r and \n are used interchangeably in character-mode terminals