Accolite University Feb Batch 1- JS Assignment

- Meet Shah
(INT626)

1. **Array methods**

   i. concat():
```
var car1 = [“Toyota”, “BMW”];
var car2 = [“KIA”, “Hyundai”];
var car = car1.concat(car2);   // ["Toyota", "BMW", "KIA", "Hyundai"]
```

   ii. every():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.every( (item)=> {
        return item.length>=0;});    // true
```

   iii. filter():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.filter( (item)=>{
        return item.length>3;});    // ["Toyota", "Hyundai"]
```

   iv. forEach():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.forEach( (item,index)=>{
console.log(index+":"+item+" ");});    // 0:Toyota 1:BMW 2:KIA 3:Hyundai
```

   v. indexOf():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.indexOf("KIA");     // 2
```

   vi. join():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.join();    // Toyota,BMW,KIA,Hyundai
```

   vii. map():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.map( (item)=> {
        return item+" Cars";
        });        // ["Toyota Car", "BMW Car", "KIA Car", "Hyundai Car"]
```

   viii. push():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.push("BMW");       // ["Toyota", "BMW", "KIA", "Hyundai", "BMW" ];
```

   ix. lastIndexOf():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai", "BMW" ];
car.lastIndexOf("BMW");        // 4
```

x. pop():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai", "BMW" ];
car.pop();        // "BMW"
```

xi. reduce():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.reduce( (item,item1)=>{
        return item+" "+item1;});        // Toyota BMW KIA Hyundai
```

xii. reduceRight():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.reduceRight( (item,item1)=>{
        return item+" "+item1;});        // Hyundai KIA BMW Toyota
```

xiii. reverse():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.reverse();          // ["Hyundai", "KIA", "BMW", "Toyota"]
```

xiv. shift():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.shift();            // "Toyota"
car;                    // ["BMW", "KIA", "Hyundai"]
```

xv. slice():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.slice(1,3);         // ["BMW", "KIA"]
```

xvi. some():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.some( (item)=> {
        return item.length>3;});         // true
```

xvii. toSource():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.toSource();         // ["Toyota", "BMW", "KIA", "Hyundai"];
```

xviii. sort():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.sort();             // ["BMW", "Hyundai", "KIA", "Toyota"]
```

xix. splice():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.splice(3,1, "Jaguar");      // ["Hyundai"];
car;                            // ["Toyota", "BMW", "KIA", "Jaguar"]
```

xx. toString():
```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.toString();                 // Toyota, BMW, KIA, Hyundai
```

xxi. unshift():

```
var car = ["Toyota", "BMW", "KIA", "Hyundai"];
car.unshift("Jaguar");
car;                    // ["Jaguar", "Toyota", "BMW", "KIA", "Hyundai"]
```

2. **Slide 52 explanation**

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

add();  // 1
add();  // 2
add();  // 3
```

Here the output on calling the add() function 3 times will be 1, 2, 3. This is an example of closure which makes it possible for the function to have private variables.

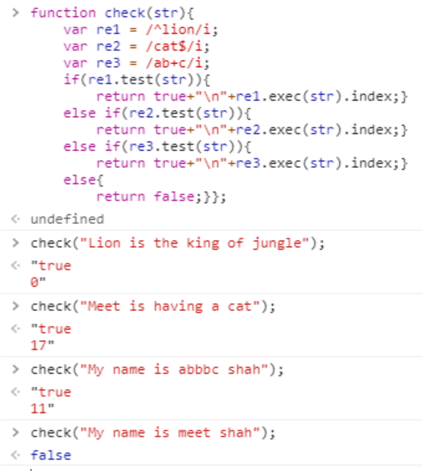
The initialisation of “counter = 0” line is executed only once because it is inside a self-invoking function and returns a function whose value is assigned to the variable add.

So first the parent function is executed and the counter is set to 0, after which the counter is protected by the scope of anonymous function and each time the function add is called it returns an increment of counter and assigns its value to variable add.

3. **Create a function which takes input as a string and returns true if**
   - **a) String starts with lion**
   - **b) String ends with cat**
   - **c) String has abc (b can be n>=1 times) anywhere in between the string.**

And also print the location of a/b/c if true or else return false.

```
function check(str){
    var re1 = /^lion/i;
    var re2 = /cat$/i;
    var re3 = /ab+c/i;
    if(re1.test(str)){
        return true+“\n”+re1.exec(str).index;}
    else if(re2.test(str)){
        return true+“\n”+re2.exec(str).index;}
    else if(re3.test(str)){
        return true+“\n”+re3.exec(str).index;}
    else{
        return “false”;}};
```

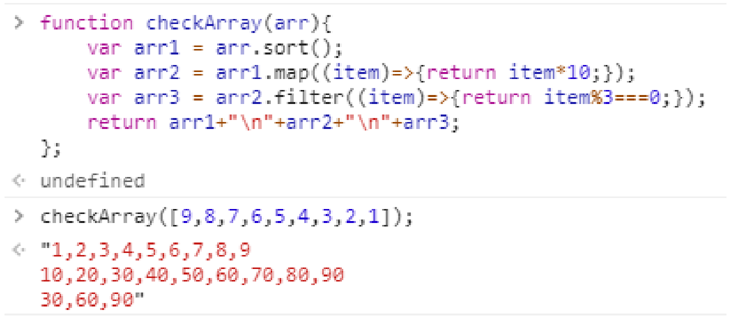


```
> function check(str){
      var re1 = /^lion/i;
      var re2 = /cat$/i;
      var re3 = /ab+c/i;
      if(re1.test(str)){
          return true+"\n"+re1.exec(str).index;}
      else if(re2.test(str)){
          return true+"\n"+re2.exec(str).index;}
      else if(re3.test(str)){
          return true+"\n"+re3.exec(str).index;}
      else{
          return false;}};
< undefined
> check("Lion is the king of jungle");
< "true
  0"
> check("Meet is having a cat");
< "true
  17"
> check("My name is abbbc shah");
< "true
  11"
> check("My name is meet shah");
< false
```

4. Create a function which takes array as an input and

   a) Sort array in ascending order
   b) Multiply each number by 10
   c) Return those numbers which are divisible by 3

```
function checkArray(arr){
   var arr1 = arr.sort();
   var arr2 = arr1.map((item)=>{return item*10;});
   var arr3 = arr2.filter((item)=>{return item%3===0;});
   return arr1+"\n"+arr2+"\n"+arr3;
};
```



```
> function checkArray(arr){
      var arr1 = arr.sort();
      var arr2 = arr1.map((item)=>{return item*10;});
      var arr3 = arr2.filter((item)=>{return item%3===0;});
      return arr1+"\n"+arr2+"\n"+arr3;
  };
<· undefined
> checkArray([9,8,7,6,5,4,3,2,1]);
<· "1,2,3,4,5,6,7,8,9
  10,20,30,40,50,60,70,80,90
  30,60,90"
```

5. Difference between == and === with a small example

== → equal value

Checks the equality of two operands without considering their type. It tests abstract equality.

=== → equal value and equal type

Compares equality of two operands with their types. It tests strict equality.

For e.g.:

5 == “5”   // true

Here, 5 is an integer and “5” is a string, but == operator just checks their value which is 5 and hence returns true.

5 === “5”   // false

Here, although 5 and “5” both have their values equal but one is of type integer and other is of type string hence the === operator being a strict one doesn’t do type conversion and returns false.