OOJS Javascript Assessment

Q1. Create a hierarchy of person, employee and developers.

CODE:

```
function person()
{
    this.gender = 'male';
};
function employee()
{
    this.age = 22;
};
employee.prototype = new person();
function developer()
{
    this.dept = "MERN";
};
developer.prototype = new employee();
var obj = new developer();

console.log(obj.dept); //MERN
console.log(obj.age);  //22
console.log(obj.gender); //male
```

```
> function person()
  {
      this.gender = 'male';
  };
< undefined
> function employee()
  {
      this.age = 22;
  };
< undefined
> employee.prototype = new person();
< ▶ person {gender: "male"}
> function developer()
  {
      this.dept = "MERN";
  };
< undefined
> developer.prototype = new employee();
< ▶ person {age: 22}
> var obj = new developer();
< undefined
> console.log(obj.dept);
  MERN
< undefined
> console.log(obj.age);
  22
< undefined
> console.log(obj.gender);
  male
```

```
> obj
< ▼ developer {dept: "MERN"}
    dept: "MERN"
  ▼ __proto__: person
      age: 22
    ▼ __proto__: person
        gender: "male"
      ▼ __proto__:
        ▼ constructor: ƒ person()
            arguments: null
            caller: null
            length: 0
            name: "person"
          ▶ prototype: {constructor: ƒ}
          ▶ __proto__: ƒ ()
            [[FunctionLocation]]: VM597:1
          ▶ [[Scopes]]: Scopes[2]
        ▶ __proto__: Object
```

Q2. Given an array, say [1,2,3,4,5]. Print each element of an array after 3 secs.

This can be done by using setInterval and clearInterval

```
var arr = [1,2,3,4,5];
var i =0;
function printArr(){
    if(i>=arr.length)
    {
        clearInterval(iterate);
        return;
    }
    else
    {
        console.log(arr[i]);
    }
     i++;
};
var iterate = window.setInterval(printArr, 3000);
```

```
> var arr = [1,2,3,4,5];
<- undefined
> var i =0;
<- undefined
> function printArr(){
      if(i>=arr.length)
      {
          clearInterval(iterate);
          return;
      }
      else
      {
          console.log(arr[i]);
      }
       i++;
  };
<- undefined
> var iterate = window.setInterval(printArr, 3000);
<- undefined
  1
  2
  3
  4
  5
> |
```

Q3. Explain difference between Bind and Call (example).

- In bind function we need to call the bindend function externally whereas in call function the reference is automatically called when binded.

BIND:

```
let customer1 = { name: 'shiv', email: 'shiv@gmail.com' };
let customer2 = { name: 'pat', email: 'pat@hotmail.com' };
function greeting(text) {
      console.log(`${this.name}`);
}
let helloShiv = greeting.bind(customer1);
let helloPat = greeting.bind(customer2);

helloShiv(); // Hello shiv
helloPat(); // Hello pat
```

CALL:

```
let customer1 = { name: 'shiv', email: 'shiv@gmail.com' };
let customer2 = { name: 'pat', email: 'nat@hotmail.com' };
function greeting(text) {
      console.log(`${text} ${this.name}`);
}
greeting.call(customer1, 'Hello'); // Hello shiv
greeting.call(customer2, 'Hello'); // Hello pat
```

- bind(this): Returns a new function whose this value is bound to the provided value.

  call(this [, arg1, arg2...]): Calls a function with a provided this. Further arguments are provided as a comma separated list.

BIND:

```
var person = {
  name: 'Abhishek',
  print: function () {
    console.log('Name is: ', this.name); // this.name means person.name in this context
  }
};
var p = person.print;
p(); // won't print the name

p = p.bind(person);
p(); // will print 'Abhishek'
```

CALL:

```
var sayHello = function (greeting) {
  greeting = greeting || 'Hello';
  console.log(greeting, this.name);
};

var abhi = {name: 'Abhishek'};
sayHello.call(abhi); // Hello Abhishek

var anil = {name: 'Anil'};
sayHello.call(anil, 'Hiiiii'); // Hiiiii Anil
```

Q4. Explain 3 properties of argument object.

- **arguments** is an Array-like object accessible inside functions that contains the values of the arguments passed to that function.

```
function func1(a, b, c) {
  console.log(arguments[0]);
  // output: 1
  console.log(arguments[1]);
  // output: 2
  console.log(arguments[2]);
  // output: 3
}
func1(1, 2, 3);
```

1. The `arguments` object is a local variable available within all functions.
2. It has entries for each argument the function was called with, with the first entry's index at `0`.
3. The `arguments` object is not an Array.

Q5. Create a function which returns number of invocations and number of instances of a function.

```
var count = 0;
function MyObj() {
    count++;
    console.log("Number Of Calls :" + count);
  MyObj.numInstances = (MyObj.numInstances || 0) + 1;
}
new MyObj();
new MyObj();
MyObj.numInstances;
```

```
var count = 0;
function MyObj() {
    count++;
    console.log("Number Of Calls :" + count);
  MyObj.numInstances = (MyObj.numInstances || 0) + 1;
}
new MyObj();
new MyObj();
MyObj.numInstances;
```

```
Number Of Calls :1
```

```
Number Of Calls :2
```

```
2
```

Q6. Create a counter using closures.

```
function counter(num) {
    var count = function() { console.log(num); }
    num++;
    return count;
}
var callCounter = counter(44);
callCounter();
```

```
function counter(num) {
    var count = function() { console.log(num); }
    num++;
    return count;
}
var callCounter = counter(44);
callCounter();
```

```
45
```

```
undefined
```

Q7. Explain 5 array methods with example.

forEach() Method:

- The forEach() method calls a function once for each element in an array, in order.

```
var cars = ["Saab", "Volvo", "BMW"];
cars.forEach(function(item,index){
    console.log(item[index]);
});
```

```
cars.forEach(function(item,index){
    console.log(item);
})
```

```
Saab
Volvo
BMW
```

indexOf() Method:

- The indexOf() method searches the array for the specified item, and returns its position.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var a = fruits.indexOf("Apple");
console.log(a);
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var a = fruits.indexOf("Apple");
console.log(a);
```

```
2
```

shift() Method:

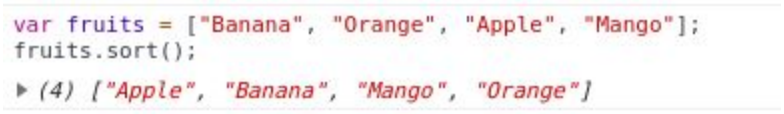
- The shift() method removes the first item of an array.The change is permanent in the array.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.shift();
```

sort() Method:

- The sort() method sorts the items of an array.
- The sort order can be either alphabetic or numeric, and either ascending (up) or descending (down).

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.sort();
```



reverse() Method:

- The reverse() method reverses the order of the elements in an array.
- This method will change the original array.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
```

```
fruits.reverse();
```