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

Ans.

JavaScript objects have a link to a prototype object. When trying to access a property of an object, the property will not only be sought on the object but on the prototype of the object, the prototype of the prototype, and so on until either a property with a matching name is found or the end of the prototype chain is reached.

```
» function person(name,age){
  this.name=name;
  this.age=age;
  }

  function employee(salary){
  this.salary=salary;
  }

  function department(dept){
  this.dept=dept;
  }

  employee.prototype=new person('Ali',24);
  department.prototype=new employee(4000);
  var depart=new department("FEEN");
  console.log(depart);

  ▼ {…}                                          debugger eval code:17:1
    dept: "FEEN"
    ▼ <prototype>: {…}
      salary: 4000
      ▼ <prototype>: {…}
        age: 24
        name: "Ali"
        ▼ <prototype>: {…}
          ▶ constructor: function person()
          <prototype>: Object { … }
← undefined
```

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

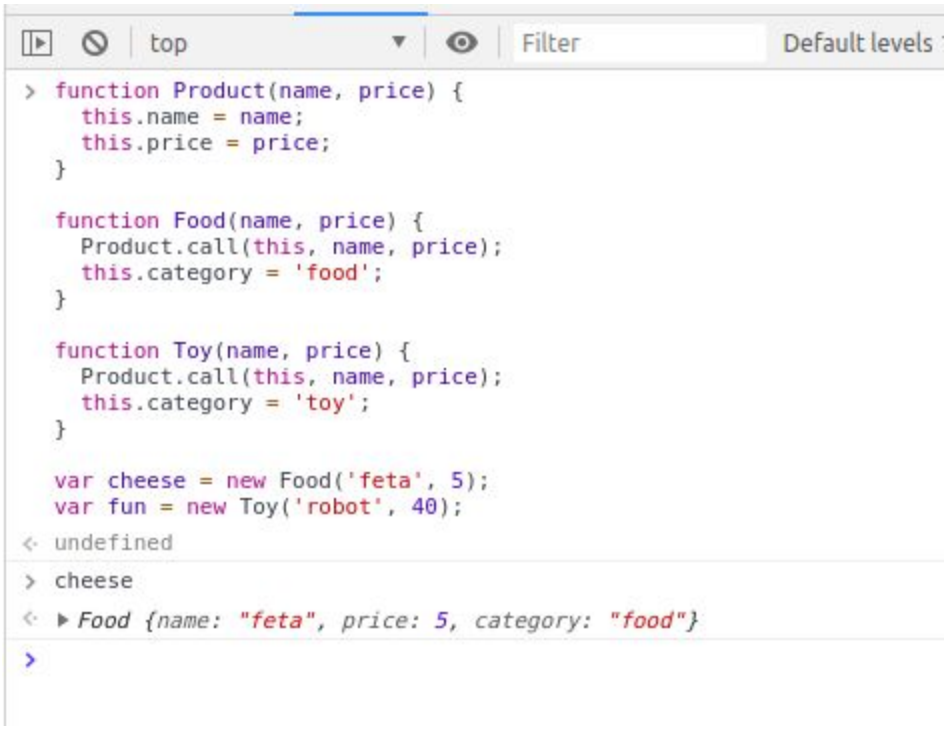
```
 var arr=[1,2,3,4,5];

function print(i){
        if(i<arr.length){
        setTimeout(function(){console.log(arr[i]);
print(++i);},3000); }}

print(0);
```

```
VM1673:3 1
VM1673:3 2
VM1673:3 3
VM1673:3 4
VM1673:3 5
```

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

Ans.

The call() method calls a function with a given this value and arguments provided individually. The simplest use of bind() is to make a function that, no matter how it is called, is called with a particular this value.



The bind() method creates a new function that, when called, has its this keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called.

```
var obj1={
x:9,
getX:function(){
return this.x;
}
}

var unbounded=obj1.getX;
console.log(unbounded());

var bounded=unbounded.bind(obj1);
console.log(bounded());
```

```
undefined                                    VM475:
9                                            VM475:1
```

Q4. Explain 3 properties of argument object.

Ans

Properties of argument object:

- The arguments object is a local variable available within all non-arrow functions. You can refer to a function's arguments inside that function by using its arguments object. It has entries for each argument the function was called with, with the first entry's index at 0. For example, if a function is passed 3 arguments, you can access them as follows:

```
arguments[0] // first argument
arguments[1] // second argument
arguments[2] // third argument
```

- The arguments object is not an Array. It is similar, but does not have any Array properties except length. For example, it does not have the pop() method. However, it can be converted to a real Array:

```
var args = Array.prototype.slice.call(arguments);
// Using an array literal is shorter than above but allocates an empty array
 var args = [].slice.call(arguments);
```

- it doesn't have Array's built-in methods like forEach() and map().

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

Ans.

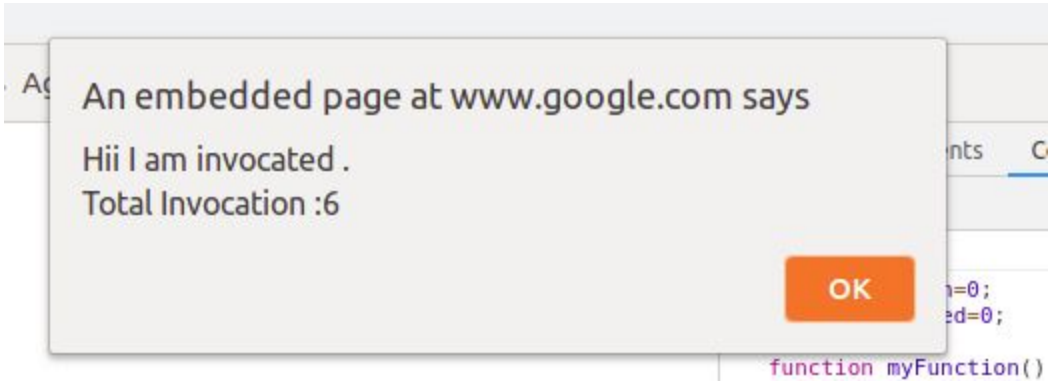
```
let invocation=0;
let instaciated=0;

function myFunction() {
invocation++;
alert("Hii I am invocated .\nTotal Invocation :"+invocation);
}
function check(c) {
var val;
if(c==1)
val=new myFunction();
else
val=myFunction();
if(val){
alert('I am also instanciated\nTotal :'+(++instaciated));
}
}
```

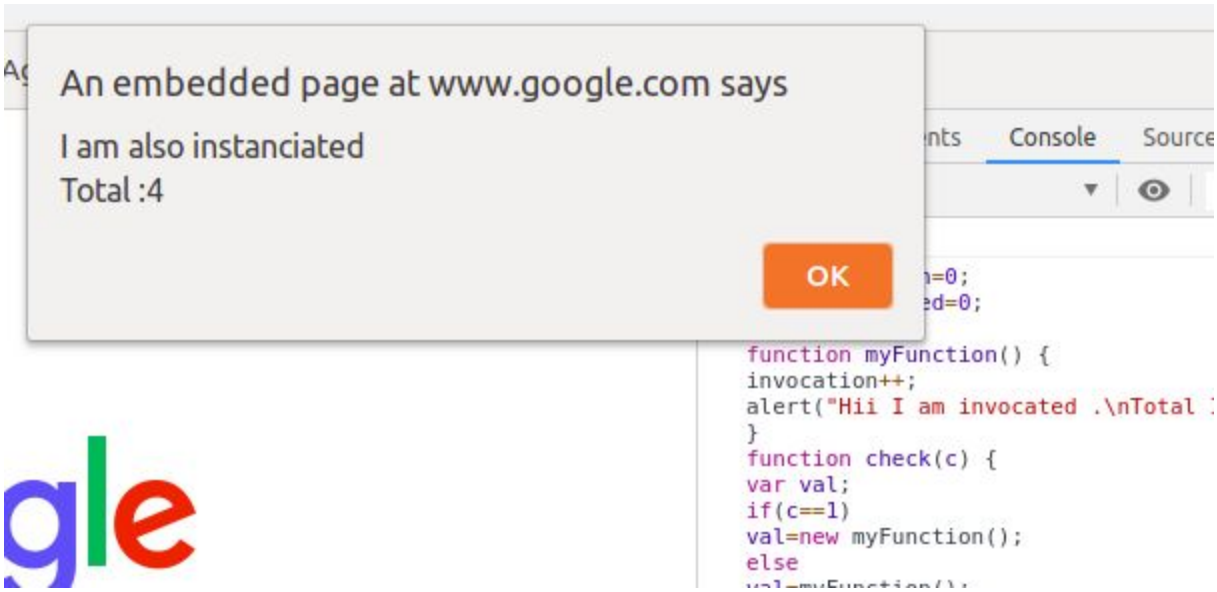
```
undefined
```

```
check(0)
```

```
undefined
```

```
check(1)
```





Q6. Create a counter using closures.

Ans.

A closure is an expression (typically a function) that can have free variables together with an environment that binds those variables (that "closes" the expression).

- The inner function can be accessed only from statements in the outer function.
- The inner function forms a closure: the inner function can use the arguments and variables of the outer function, while the outer function cannot use the arguments and variables of the inner function.

```
> function outside(){

  var count=0;
  function middle(){
  var counter= ++count;

     function inside(){
         console.log("Total Closure - "+ ++counter);}
  inside();

  }

  middle();
  }

  outside();
```

```
  Total Closure - 2                                          VM1859:8
```

```
<- undefined
```