Lab 08

JavaScript – The Good Parts

Objective: To get familiar with different good parts available for programming which are typical of JS language.

Lab Task:

1 *Functions*

a. Create a function called *calculator* and accept three parameters: *op1, op1, operator.*
b. *op1, op2* are numerical values and *operator* is a function.
c. From within the *calculator* function call *operator* and pass it two parameters *op1, op2.*
d. Save the result returned by *operator* into a variable and return that result from *calculator* function.
e. Validate the result by logging the result.

Code:

C: > Users > computer house > Desktop > New folder > JS lab4.js > ...

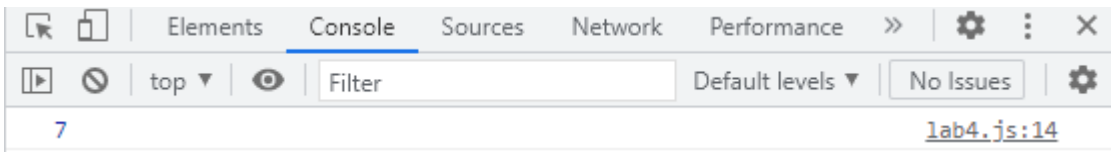
```
// Task#01

function calculator(op1, op2, func) {
    var operand1 = parseInt(op1);
    var operand2 = parseInt(op2);
    var result = func(operand1, operand2);
    return result;
}

function operator(operand1, operand2) {
    var result = operand1 + operand2;
    return result;
}
console.log(calculator(2, 5, operator));

```

Output:



2 *this keyword*

a. Create a function called *getFullName.*
b. Using *this* keyword return full name string built by concatenating firstname and lastname.
c. Create two objects containing firstname and lastname.
d. Call *getFullName* using function built-in apply method and passing it the created objects one by one.
e. Validate if full name is being returned correctly using *console.log* method.

Code:

```
99  //Task no 2
100 var student1 = {
101     firstName: 'Shakir',
102     lastName: 'khan',
103 };
104 var student2 = {
105     firstName: 'Huzaifa',
106     lastName: 'Ahmed',
107 };
108 function getFullName(){
109     return this.firstName + " " + this.lastName;
110 }
111 console.log(getFullName.apply(student1));
112 console.log(getFullName(student2));
```

Output:

```
Shakir Khan                                   lab4.js:29
Rukhsar Naz                                   lab4.js:30
```

3 *Closure*

a. Create a *Timer* closure and make it tick (console log) after every 1 second.
   i. Create a private *counter* variable
   ii. Create a private *tick* method and use *setInterval* to increment the *counter* after 1 second and console.log *counter* variable. If the *counter* exceeds *limit* stop. You may need to accept the *limit* parameter in *tick* method.
   iii. Create a *start* method and call tick method from it. Accept the *limit* parameter.
   iv. Return the *start* method by creating an object and assigning the start method as a public API on the returned object.

Code:

```
// Task#03

var timer = (function () {
    var counter = 0;
    function tick(limit) {
        setInterval(
            function () {
                if (counter < limit) {
                    counter++;
                    console.log(counter);
                }
            }, 1000
        )
    }
    function start(limit) {
        tick(limit);
    }
    return {
        start: start
    }
}());
timer.start(3);
```

Output:

```
1                                                    lab4.js:41
2                                                    lab4.js:41
3                                                    lab4.js:41
>
```

a. Create *Student* module using closure and two child sub-modules *Courses* and *Result*.
b. Add properties to each module (parent module and child sub-modules)
c. Log each property via module and sub-module properties.

Code:

```
var student = (function () {
    var courses = (function () {
        var name = "oop";
        var courseName = function () {
            return name + " is course";
        }
        return {
            courseName
        };
    }());
    var result = (function () {
        var grade = "C";
        var gradeResult = function () {
            return grade + " is Result!";
        }
        return {
            gradeResult
        };
    }());
    return {
        courses,
        result
    }
}());

console.log(student.courses.courseName());
console.log(student.result.gradeResult());
```

Output:

```
oop is course                                   lab4.js:82
C is Result!                                    lab4.js:83
```

a. Create function called *getProgramResults* and create a promise called *runProgramPromise* inside it, and await its result using *async* and *await.*

Code:

```
//Task#05
async function getProgramResults()
{
    async function runProgramPromise(){
        return new Promise((resolve,reject) =>{
            setTimeout(function(){
                resolve('passed');

            }, 1000);



        });

    }
    const result = await runProgramPromise();
    console.log(result);

}
getProgramResults();
```

Output:

```
passed                                                    lab4.js:100
```

a. Create a base class called *Program* and two child classes called *TeacherProgram* and *StudentProgram*
b. Provide run function in *Program* base class.
c. Provide *debug* function in child *StudentProgram* class.
d. Provide *release* function in child *TeacherProgram*.
e. Inherit *Program* class behavior by creating new object of it and assigning it to the *prototype* property of each child class.

Code:

```
//Task#06

function Program(){
    this.run = function(){
        return "Run function is running!";
    };
}
function TeacherProgram(){
    this.release = function(){
        return "Release function is running!";
    }
}
function StudentProgram(){
    this.debug = function(){
        return "Debug function is running";
    }
}
TeacherProgram.prototype = new Program();
StudentProgram.prototype = new Program();

let teacherProgram = new TeacherProgram();
let studentProgram = new StudentProgram();

console.log(teacherProgram.release());
console.log(studentProgram.debug());
console.log(studentProgram.run());
```

Output:

```
Release function is running!                    lab4.js:128
Debug function is running                       lab4.js:129
Run function is running!                        lab4.js:130
```

Home Task

1 *Module*

a. Create a module called *Class*.
b. Create three sub-modules called *Teacher*, *Notes* and *Lecture*.
c. Add relevant members to the containing (*Class* module) and to each sub-module.
d. Create instances of the *Class* module and log the members

CODE:

```
var Class = (function(){
    var Teacher = (function(){
    var name = "Teachers";
    var teach = function() { return `${name}  is teaching`; };
    return{
        teach
    };
}());
    var Notes = (function(){
        var name = "PME";
        var notes = function() { return `${name} notes`; };

return {
    notes
    };
}());

var Lecture = (function(){
    var name = "Info Sec";
    var lec  = function() { return `${name} lecture`; };

    return {
        lec
    };
}());

return {
Teacher,
Notes,
Lecture
};
}());
```

```
console.log(Class.Teacher.teach());
console.log(Class.Notes.notes());
console.log(Class.Lecture.lec());
```

OUTPUT:

```
Teachers  is teaching                    lab4.js:194
PME notes                                lab4.js:195
Info Sec lecture                         lab4.js:196
```

2.Asynchronicity

a. Create a *ClassAlarm* function.
b. Return an *AlarmPromise* promise from that function.
c. Call *resolve* after the class time is over with the help of *setTimeout* function after 30 minutes and pass in the string message 'Class is over' to *resolve* function.
d. *Await* the result using *await* keyword and console.log that returned message.

CODE:

```
//hom
let ClassAlarm  =3;
let ClassAlarmPromise = new Promise((resolve, reject) => {
    if(ClassAlarm == 3) {
        resolve("on time");
    } else {
        reject("not on time");
    }
});

ClassAlarmPromise.then((result) => {
    console.log(result);
}).catch((error) => console.log(error));

let ClassAlarmToken = setTimeout(function(){
    console.log("alarm time");
}, 100);
```

OUTPUT:

```
on time                                  lab4.js:142
alarm time                               lab4.js:146
```

3.Functional Inheritance

a. Create a base class called *Gadget* and define following properties
   i. startTime date and time property.
   ii. salePrice numeric property.
   iii. *Start* and *End* methods.

CODE:

```
//2
function Gadget(sale, time) {
    this.sale = sale;
    this.time = time;
    this.startTime = function() {
        return "Time : ${this.sale},${this.time}";
    };
}
let gadget = new Gadget(8, "9 pm");
console.log(gadget.sale);
console.log(gadget.time);
console.log(gadget.startTime);
```

OUTPUT:

```
8                                                    lab4.js:157
9 pm                                                 lab4.js:158
```