D15B/37 **Mikil Lalwani** **IP Lab 5**

Aim -

Experiment to study the basics of JavaScript.

Theory-

1. DOM

   The Document Object Model (DOM) is the data representation of the objects that comprise the structure and content of a document on the web. This guide will introduce the DOM, look at how the DOM represents an HTML document in memory and how to use APIs to create web content and applications.

   What is the DOM?

   The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

   A web page is a document that can be either displayed in the browser window or as the HTML source. In both cases, it is the same document but the Document Object Model (DOM) representation allows it to be manipulated. As an object-oriented representation of the web page, it can be modified with a scripting language such as JavaScript.

   For example, the DOM specifies that the querySelectorAll method in this code snippet must return a list of all the <p> elements in the document:

```
const paragraphs = document.querySelectorAll("p");

// paragraphs[0] is the first <p> element

// paragraphs[1] is the second <p> element, etc.

alert(paragraphs[0].nodeName);
```

   All of the properties, methods, and events available for manipulating and creating web pages are organized into objects. For example, the document object that represents the document itself, any table objects that implement the HTMLTableElement DOM interface for accessing HTML tables, and so forth, are all objects.

   The DOM is built using multiple APIs that work together. The core DOM defines the entities describing any document and the objects within it. This is expanded upon as needed by other APIs that add new features and capabilities to the DOM. For example, the HTML DOM API adds support for representing HTML documents to the core DOM, and the SVG API adds support for representing SVG documents.

DOM and JavaScript

The previous short example, like nearly all examples, is JavaScript. That is to say, it is written in JavaScript, but uses the DOM to access the document and its elements. The DOM is not a programming language, but without it, the JavaScript language wouldn't have any model or notion of web pages, HTML documents, SVG documents, and their component parts. The document as a whole, the head, tables within the document, table headers, text within the table cells, and all other elements in a document are parts of the document object model for that document. They can all be accessed and manipulated using the DOM and a scripting language like JavaScript.

The DOM is not part of the JavaScript language, but is instead a Web API used to build websites. JavaScript can also be used in other contexts. For example, Node.js runs JavaScript programs on a computer, but provides a different set of APIs, and the DOM API is not a core part of the Node.js runtime.

The DOM was designed to be independent of any particular programming language, making the structural representation of the document available from a single, consistent API. Even if most web developers will only use the DOM through JavaScript, implementations of the DOM can be built for any language, as this Python example demonstrates:

```
# Python DOM example

import xml.dom.minidom as m

doc = m.parse(r"C:\Projects\Py\chap1.xml")

doc.nodeName # DOM property of document object

p_list = doc.getElementsByTagName("para")
```

2.Regular Expression-

Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects. These patterns are used with the exec() and test() methods of RegExp, and with the match(), matchAll(), replace(), replaceAll(), search(), and split() methods of String. This chapter describes JavaScript regular expressions.

Creating a regular expression

You construct a regular expression in one of two ways:

Using a regular expression literal, which consists of a pattern enclosed between slashes, as follows:

```
const re = /ab+c/;
```

Regular expression literals provide a compilation of the regular expression when the script is loaded. If the regular expression remains constant, using this can improve performance.

Or calling the constructor function of the RegExp object, as follows:

```
const re = new RegExp('ab+c');
```

Using the constructor function provides runtime compilation of the regular expression. Use the constructor function when you know the regular expression pattern will be changing, or you don't know the pattern and are getting it from another source, such as user input.

Writing a regular expression pattern

A regular expression pattern is composed of simple characters, such as /abc/, or a combination of simple and special characters, such as /ab*c/ or /Chapter (\d+)\.\d*/. The last example includes parentheses, which are used as a memory device. The match made with this part of the pattern is remembered for later use, as described in Using groups.

Code-

1. Grade calculator-

```
function grade() {
  let m = document.getElementById("marks1").value;
  if (m > 100 || m < 0) {
    document.getElementById("marks1").value = "";
    alert("Enter marks in range of 0 to 100!");
  } else if (m > 90) {
    alert("Grade O!");
  } else if (m > 80) {
    alert("Grade A!");
  } else if (m > 70) {
    alert("Grade B!");
  } else if (m > 60) {
    alert("Grade C!");
  } else if (m > 50) {
```

```
  alert("Grade D!");
} else {
  alert("Grade F!");}
```

2. Calculator-

```
        while (true) {
          let num1 = parseInt(prompt("Enter 1st number:"));
          let num2 = parseInt(prompt("Enter 2nd number:"));
          let choice = parseInt(
            prompt(
              "Select operation:\n 1)Addition\n 2)Subtraction\n 3)Multiplication\n 4)Division") );
          switch (choice) {
            case 1:
              alert(num1 + num2);
              break;
            case 2:
              alert(num1 - num2);
              break;
            case 3:
              alert(num1 * num2);
              break;
            case 4:
              alert(num1 / num2);
              break;
            default:
              alert("Select valid operation!");}}
```

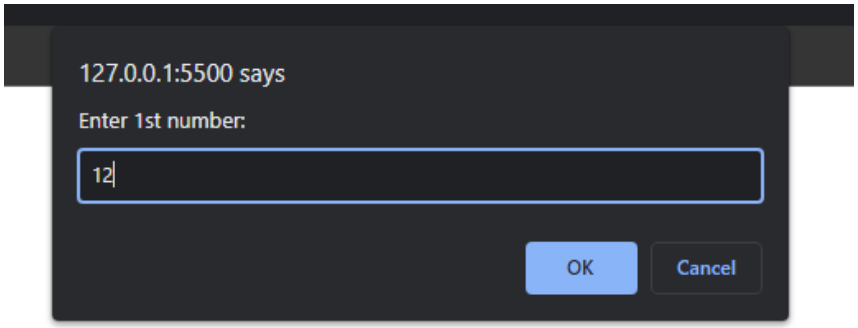
3. Regex-

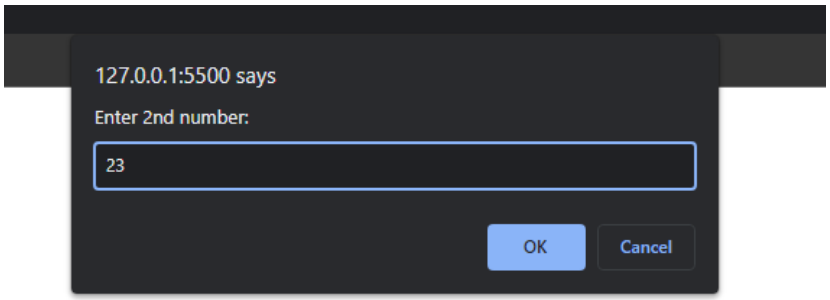
```
function check() {
  let no = document.getElementById("pno").value;
  const pnum = /^[0-9]{10,10}$/;
  if (!pnum.test(no)) {
    alert("Enter a valid number!");  }
  let id = document.getElementById("mail").value;
  const mid = /([a-zA-Z0-9,_\.]+)@([a-zA-Z0-9]{2,3})(.com$)/;
  if (!mid.test(id)) {
    alert("Enter valid email!");  }
  const psd = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/;
  let pswd = document.getElementById("pass").value;
  console.log(pswd);
  if (!psd.test(pswd)) {
    alert("Enter a valid password!");  }}
```

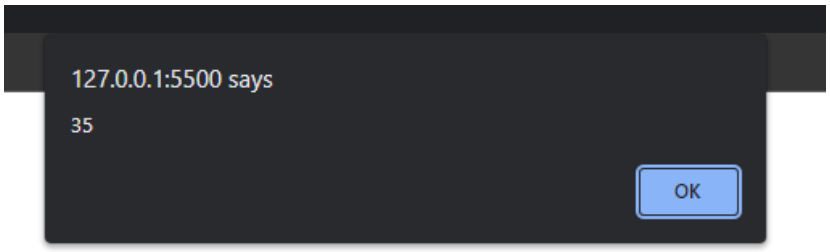
Screenshots-

Grade Calculator

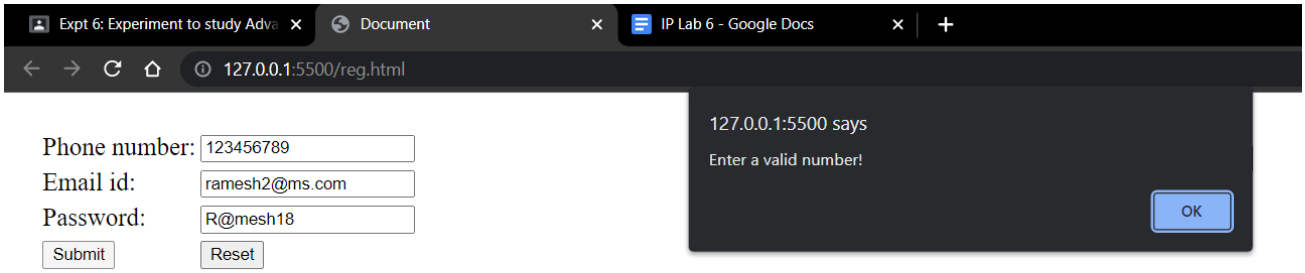
Calculator

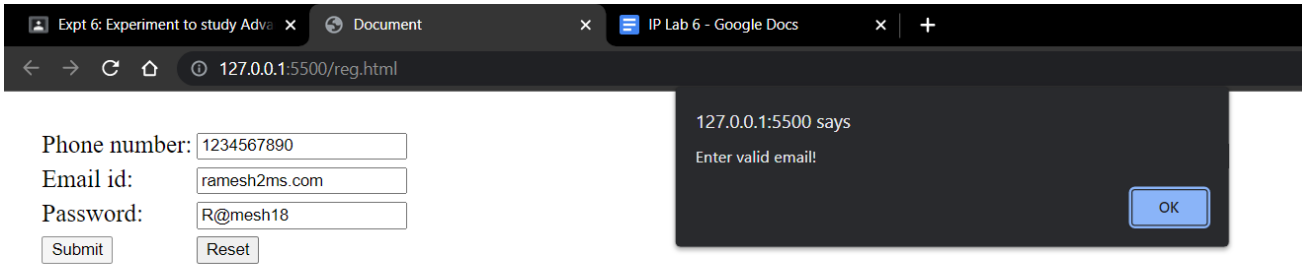


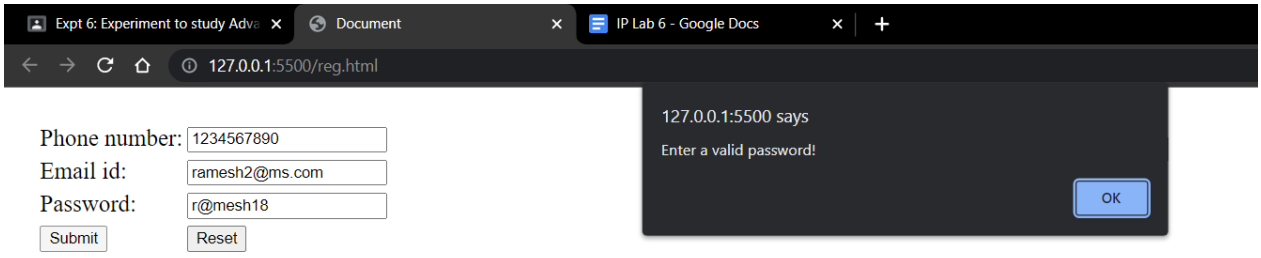




Password Validation







Conclusion-

We have successfully completed the experiment.