Unit 5 : Regular Expressions , Rollover and Frames

(Weightage – 14marks)

Unit –V Regular Expression, Rollover and Frames	5a. Compose relevant regular expression for the given character pattern search. 5b. Develop JavaScript to implement validations using the given regular expression. 5c. Create frames based on the given problem. 5d. Create window object as per	5.1 Regular Expression - language of regular expression, finding non matching characters, entering a range of characters, matching digits and non digits, matching punctuations and symbols, matching words, replacing a the text using regular expressions, returning the matched characters, regular expression object properties.
	the given problem. 5e. Develop JavaScript for creating rollover effect for the given situation.	5.2 Frames – create a frame, invisible borders of frame, calling a child windows, changing a content and focus of a child window, writing to a child window, accessing elements of another child window. 5.3 Rollover – creating rollver, text rollver, Multiple actions for rollover, more efficient rollover.

Question – State what is regular Expression .or Describe Regular Expression .

Regular Expression

What is a Regular Expression?

A **regular expression** (regex) is a sequence of characters that forms a search pattern. It's commonly used in text processing tasks like searching, replacing, or validating strings. A regular expression (regex) is a pattern for matching character combinations in strings. It helps validate or find specific text patterns.

- **Search pattern**: A regex pattern can be a single character, a word, or a complex string combination.
- **Usage**: Common functions for regex are search(), replace(), and test()

Syntax: `/pattern/modifiers;`

- **Pattern**: This is the string that defines what to search for. It could be any combination of letters, numbers, or special characters.
- **Modifiers**: Optional flags that adjust the behavior of the search, such as making it case-insensitive or allowing global matching across multiple lines.
- **g**: Global search (find all matches, not just the first).
- **i**: Case-insensitive search.
- **m**: Multi-line matching

Example:

- `/abc/`: Searches for the sequence "abc".
- `/abc/i`: Searches for "abc" regardless of case (i.e., it will match "ABC", "aBc", etc.).

```
let text = "Hello world! Hello again.";

let regex = /hello/gi; // Finds all "hello" regardless of case

let result = text.match(regex); // ["Hello", "Hello"]
```

Language of Regular Expressions

Brackets in Regular Expressions:

- Brackets `[]` are used to specify a set or range of characters to match. Here's a breakdown of the examples shown:

Sr No	Expression	Description
1	`[...]`	Specifies to find **any one character** between the brackets.
2	`[^...]`	Specifies to find **any one character not** between the brackets.
3	`[0-9]`	Specifies to find **any decimal digit** from 0 through 9.
4	`[a-z]`	Specifies to find **any lowercase character** from a through z.
5	`[A-Z]`	Specifies to find **any uppercase character** from A through Z.
6	`[a-Z]`	Specifies to find **any character** from lowercase a through uppercase Z (this includes both lower and uppercase letters).

Example Usage:

1. **Find a vowel (a, e, i, o, u) in a string:**

```javascript
let text = "Hello world";
let regex = /[aeiou]/gi;
let result = text.match(regex); // Matches "e" and "o" in "Hello"
```

2. **Find any digit:**

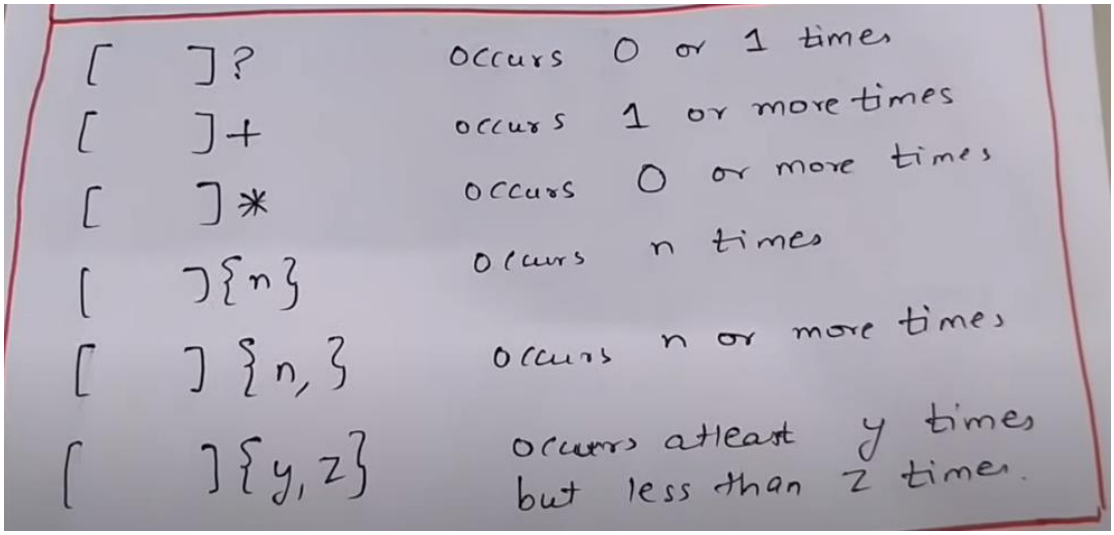
```javascript
let text = "There are 7 apples and 12 oranges.";
let regex = /[0-9]/g;
let result = text.match(regex); // Matches "7", "1", "2"
```

3. **Find any non-alphabetical character:**

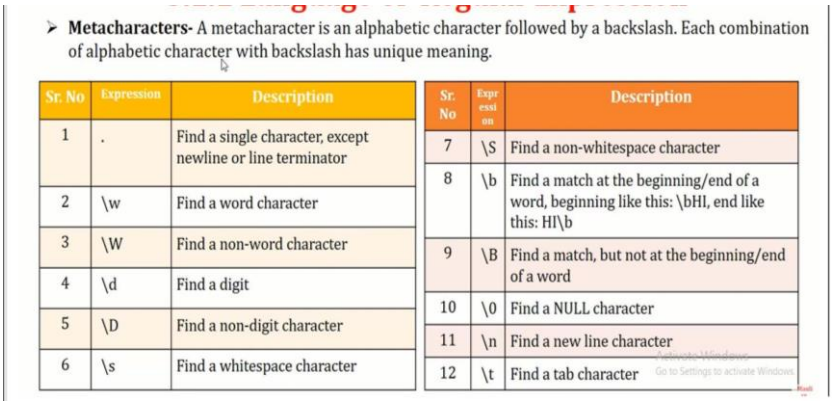
```javascript
let text = "Hello! How are you?";
let regex = /[^a-zA-Z]/g;
let result = text.match(regex); // Matches "!", " ", " ", "?", etc.
```

Describe Quantifier with suitable example (4m)

Quantifiers in regular expressions define **how many times** a character, group, or character class should appear in the input string. They allow you to control the **frequency** of matching occurrences.



Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{X}	Matches any string that contains a sequence of *X* *n*'s
n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s
n{X,}	Matches any string that contains a sequence of at least X *n*'s
n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it
?=n	Matches any string that is followed by a specific string *n*
?!n	Matches any string that is not followed by a specific string *n*



➢ **Metacharacters-** A metacharacter is an alphabetic character followed by a backslash. Each combination of alphabetic character with backslash has unique meaning.

Sr. No	Expression	Description
1	.	Find a single character, except newline or line terminator
2	\w	Find a word character
3	\W	Find a non-word character
4	\d	Find a digit
5	\D	Find a non-digit character
6	\s	Find a whitespace character

Sr. No	Expression	Description
7	\S	Find a non-whitespace character
8	\b	Find a match at the beginning/end of a word, beginning like this: \bHI, end like this: HI\b
9	\B	Find a match, but not at the beginning/end of a word
10	\0	Find a NULL character
11	\n	Find a new line character
12	\t	Find a tab character

RegExp Object Properties

Property	Description
constructor	Returns the function that created the RegExp object's prototype
global	Checks whether the "g" modifier is set
ignoreCase	Checks whether the "i" modifier is set
lastIndex	Specifies the index at which to start the next match
multiline	Checks whether the "m" modifier is set
source	Returns the text of the RegExp pattern

- **RegExp Function-** The RegExp consist of following functions to perform operation related to regular expression :

Sr. No	Expression	Description
1	compile()	Deprecated in version 1.5. Compiles a regular expression
2	exec()	Tests for a match in a string. Returns the first match
3	test()	Tests for a match in a string. Returns true or false
4	toString()	Returns the string value of the regular expression

1. compile() (Deprecated)

Description: In earlier versions, compile() was used to compile a regular expression, making it reusable with a new pattern or flags. However, this method is deprecated and no longer recommended or supported in most modern JavaScript environments.

Alternative: You can create a new regular expression by directly assigning it to a new RegExp object.

```
// Old way (deprecated, do not use)
let regex = /abc/;
regex.compile("xyz");  // This is deprecated and should be avoided

// New way (recommended)
let regex = new RegExp("xyz");  // Creates a new RegExp to match "xyz"
console.log(regex.test("xyz")); // Output: true
```

2. exec()

Description: exec() is used to search for a match in a string. It returns an array of information about the first match found, or null if no match is found. The returned array contains the matched text and additional information about the match (e.g., index and input).

```
const regex = /hello/;
const text = "hello world";

// Using exec to find a match
const result = regex.exec(text);

console.log(result);
// Output: [ 'hello', index: 0, input: 'hello world', groups: undefined ]
```

3. test()

Description: test() checks if a match for the pattern exists in a string. It returns true if there's a match and false otherwise. This is useful for simply checking if a pattern is present without needing detailed match information.

```
const regex = /world/;
const text = "hello world";

// Using test to check for a match
const isMatch = regex.test(text);

console.log(isMatch); // Output: true
```

4.toString()

Description: toString() returns the string representation of the regular expression. This can be useful if you want to see the pattern in a readable format or when debugging.

```
const regex = /hello/;

// Using toString to get the string representation
console.log(regex.toString()); // Output: "/hello/"
```

5.search()

The search method in JavaScript is used to find the position of the first match of a regular expression in a string. It returns the **index** of the match or **-1** if no match is found.'

Unlike exec, the search method only returns the position of the first match, not an array of details.

Syntax: string.search(regex)

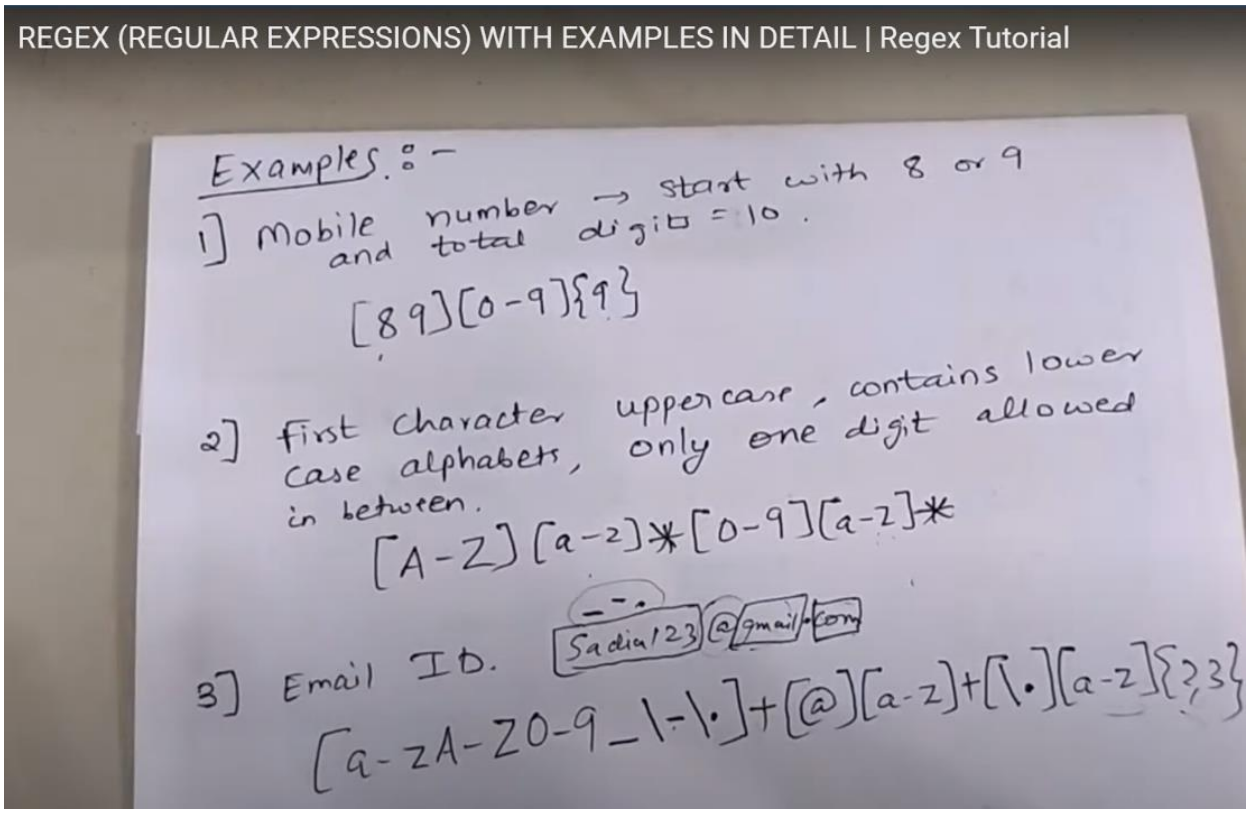
- **string**: The string to search within.
- **regex**: A regular expression pattern.

```
const text = "hello world";

const regex = /world/;


const position = text.search(regex);

console.log(position); // Output: 6
```



Programs 1:

Write a JavaScript function to check the first character of a string is uppercase or not.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Uppercase Checking</title>
    </head>
    <body>
        <script>
            function touppercase(str) {
                const regex = /^[A-Z]/;
                if (regex.test(str)) {
                    document.write("String first letter is
uppercase");
                } else {
                    document.write("String first letter is not
uppercase");
                }
            }
            var s = prompt("Enter the string: ");
            touppercase(s);
        </script>
    </body>
</html>
```

(b) Form regular expressions for following :

(i) Validation of email address.

(ii) Validation of adhaar card. Format is dddd – dddd – dddd

(iii) Validation of phone number. Format is (ddd) – (ddddddddd)

i)/[A-Za-z0-9\-\._]+[@][a-z]+[\.][a-z]{2,3}/

ii)/[0-9]{4}[\-][0-9]{4}[\-][0-9]{4}/ or /^\d{4}-\d{4}-\d{4}$/;

iii)/([\+][0-9]{2})[\-]([0-9]{10})/

Program 2 – Write a JavaScript function to check whether a given address is a valid IP address or not. (4marks)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Validsate IP ADDRESS</title>
    </head>
    <body>
        <script>
            function ipaddress(str) {
                const regex = /[0-9]{3}[\.][0-9]{3}[\.][0-9]{3}[\.][0-9]{3}/;
                if (regex.test(str)) {
                    document.write("Ip ADDRESSS IS VALID");
                } else {
                    document.write("Ip ADDRESSS IS INVALID");
                }
            }
            var s = prompt("Enter the IP address in format (ddd.ddd.ddd.ddd): ");
            ipaddress(s);
        </script>
    </body>
</html>
```

Program 3

Write a JavaScript function that checks whether a passed string is palindrome or not.
Note: Any other relevant logic shall be considered
function isPalindrome(str) {
str = str.replace(/[^A-Za-z0-9]/g, '').toLowerCase();
return str === str.split('').reverse().join('');
}
console.log(isPalindrome("A man, a plan, a canal, Panama")); // Output: true
console.log(isPalindrome("racecar")); // Output: true
console.log(isPalindrome("hello")); // Output: false
	4M
	Correct logic 2M
	Correct syntax 2M

Program 4 : Write a JavaScript that accepts a string and searches for the pattern “MSBTE” in the given string using regular expressions. If the pattern is found, JavaScript will display that “Pattern is found” else display “Pattern is not found”.(4m)

```html
<!DOCTYPE html>
<html>
<head>
    <title>Pattern Search</title>
</head>
<body>
    <form>
        <label>Enter a string: </label>
        <input type="text" name="string" id="pattern">
        <input type="button" value="Search" onclick="patternsearch()">
    </form>

    <script>
        function patternsearch() {
            // Get the input value from the text field
            var str = document.getElementById("pattern").value;
            // Regular expression to search for "MSBTE"
            var regex = /MSBTE/i;

            // Check if the pattern is found in the string
            if (regex.test(str)) {
                // Show an alert message if the pattern is found
                alert("Pattern found successfully");
            } else {
                // Show an alert message if the pattern is not found
                alert("Pattern not found");
            }
        }
    </script>
</body>
</html>
```

Program 5 : Write a Java script that displays textboxes for accepting name & email ID & a submit button. Write Java script code such that when the user clicks on submit button (1)Name Validation (2) Email ID validation

```
<!DOCTYPE html>
<html>
<head>
    <title>Name and Email Validation</title>
</head>
<body>
    <h2>Enter Your Details</h2>

    <form id="userForm">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name"><br><br>

        <label for="email">Email ID:</label>
        <input type="text" id="email" name="email"><br><br>

        <input type="button" value="Submit" onclick="validateForm()">
    </form>

    <script>
        function validateForm() {
            var name = document.getElementById("name").value;
            var email = document.getElementById("email").value;
            var nameRegex = /^[A-Za-z\s]+$/;
            var emailRegex = /^[a-zA-Z0-9._-]+@[a-z]+\.[a-z]{2,3}$/;

            // Name validation
            if (!nameRegex.test(name)) {
                alert("Invalid name! Name should contain only letters and spaces.");
                return false;
            }

            // Email validation
            if (!emailRegex.test(email)) {
                alert("Invalid email! Please enter a valid email address.");
                return false;
            }

            // If both validations pass, show a success message
            alert("Form submitted successfully!");
            return true;
        }
    </script>
</body>
</html>
```

Program 6 : Write a javascript program to validate user accounts for multiple set of user ID and password (using swith case statement).(4m)

```
<form id="loginForm">
        <label for="userID">User ID:</label>
        <input type="text" id="userID" name="userID"><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password"><br><br>

        <input type="button" value="Login" onclick="validateUser()">
    </form>
  <script>
        function validateUser() {
            var userID = document.getElementById("userID").value;
            var password = document.getElementById("password").value;
            switch(userID) {
                case "user1":
                    if (password === "password1") {
                        alert("Login successful! Welcome User1.");
                    } else {
                        alert("Invalid password for User1.");
                    }
                    break;

                case "user2":
                    if (password === "password2") {
                        alert("Login successful! Welcome User2.");
                    } else {
                        alert("Invalid password for User2.");
                    }
                    break;

                case "user3":
                    if (password === "password3") {
                        alert("Login successful! Welcome User3.");
                    } else {
                        alert("Invalid password for User3.");
                    }
                    break;

                case "admin":
                    if (password === "adminpass") {
                        alert("Login successful! Welcome Admin.");
                    } else {
                        alert("Invalid password for Admin.");
                    }
                    break;

                default:
                    alert("Invalid User ID.");
            }
        }
    </script>
```

Program 7 : Write HTML code to design a form that displays textboxes for accepting UserID and Aadhar No. and a SUBMIT button. UserID should contain 10 alphanumeric characters and must start with Capital Letter. Aadhar No. should contain 12 digits in the format nnnn nnnn nnnn. Write JavaScript code to validate the UserID and Aadhar No. when the user clicks on SUBMIT button.(6m)

```
<!DOCTYPE html>
<html>
<head>
    <title>UserID and Aadhar Validation</title>
</head>
<body>
    <h2>User Registration Form</h2>

    <form id="registrationForm">
        <label for="userID">User ID (10 alphanumeric characters, starts with
capital letter):</label>
        <input type="text" id="userID" name="userID"><br><br>

        <label for="aadharNo">Aadhar No. (Format: nnnn nnnn nnnn):</label>
        <input type="text" id="aadharNo" name="aadharNo"><br><br>

        <input type="button" value="Submit" onclick="validateForm()">
    </form>

    <script>
        function validateForm() {
            var userID = document.getElementById("userID").value;
            var aadharNo = document.getElementById("aadharNo").value;
            var userIDRegex = /^[A-Z][a-zA-Z0-9]{9}$/;
            var aadharRegex = /^[0-9]{4} [0-9]{4} [0-9]{4}$/;

            // UserID validation
            if (!userIDRegex.test(userID)) {
                alert("Invalid UserID! UserID must be 10 alphanumeric characters
and start with a capital letter.");
                return false;
            }

            // Aadhar No. validation
            if (!aadharRegex.test(aadharNo)) {
                alert("Invalid Aadhar No.! Aadhar No. must be 12 digits in the
format nnnn nnnn nnnn.");
                return false;
            }

            // If both validations pass
            alert("Form submitted successfully!");
            return true;
        }
    </script>
</body>
</html>
```

(a) Write HTML script that will display following structure

Name	
Email	
Pin Code	
	Submit

Write the JavaScript code for below operations :

(1) Name, Email & Pin Code should not be blank.

(2) Pin Code must contain 6 digits & it should not be accept any characters.

Program 8 :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Form Validation</title>
</head>
<body>

    <form id="myForm" onsubmit="return validateForm()">
        <table border="1">
            <tr>
                <td>Name</td>
                <td><input type="text" id="name"></td>
            </tr>
            <tr>
                <td>Email</td>
                <td><input type="email" id="email"></td>
            </tr>
            <tr>
                <td>Pin Code</td>
                <td><input type="text" id="pincode"></td>
            </tr>
            <tr>
                <td ></td>
                <td>
                    <input type="submit" value="Submit">
                </td>
            </tr>
        </table>
    </form>

    <script>
        function validateForm() {
            // Get input values
            var name = document.getElementById("name").value.trim();
            var email = document.getElementById("email").value.trim();
            var pincode = document.getElementById("pincode").value.trim();
```

```
 // Check if fields are empty
            if (name === "" && email === "" && pincode === "") {
                alert("Name, Email & Pin Code should not be blank.");
                return false;
            }

            // Validate Pin Code with regex
            var pinCodePattern = /^\d{6}$/; // [0-9]{6}
            if (!pinCodePattern.test(pincode)) {
                alert("Pin Code must contain exactly 6 digits and should not accept any
other characters.");
                return false;
            }

            // If all checks pass
            alert("sucessfully submited");
            return true;

        }
    </script>

</body>
</html>
```