SECTION1 DESIGN AND ANALYSIS OF ALGORITHMS LAB

Session 1: Implementation ofSimple Algorithms

Introduction

The focus of the section is to implement small problems such as Euclid's algorithm for GCD, polynomial expression evaluation through Horner' method, algorithms for evaluation of exponentiation and simple sorting algorithms. Selection sort begins by finding the least element in the array which is moved to the first position. Then the second least element is searched for and moved**to** the second position in the array. This process continues until the entire array is sorted.

Objectives

The main objectives of the algorithms are to :

- Implement Euclid's algorithm to find GCD
- Implement Horner's method to evaluate polynomial expression
- Compare the performance of Horner's method with brute force method
- Implement simple sorting algorithms

Implement multiplication of two matrices

Problems for Implementations:

Q1. Implement Euclid's algorithm to find GCD (15265, 15)and calculate the number of times **mod** and assignment operations will be required.

Q2(i). Implement Horner's rule for evaluating polynomial P(x)= $6x^6+5x^5+4x^4-3x^3+2x^2+8x-7$ at x = 3. Calculate how many times (i) multiplications and addition operations will take (ii) how many times the loop will iterate

Q2(ii) Apply a brute force method to implement the above polynomial expression(Q2(i)) and compare it with Horner's method in terms of number of multiplication operations

Q3.Implement multiplication of two matrices A[4,4] and B[4,4]

and calculate (i) how many times the innermost and the outermost loops will run (ii) total number of multiplications and additions in computing the multiplication

Q4. Implement left to right and right to left binary exponentiation methods for the following problems:

(i) 4^{512} (ii) 3^{31}

Calculate the followings for problems (i) and (ii)

- How many times the loops will be executed?
- How many times the multiplication and additions operations will be executed?

Q5. Implement Bubble Sort algorithm for the following list of numbers:

55 25 15 40 60 35 17 65 75 10

Calculate (i) a number of exchange operations (ii) a number of times comparison operations (iii) a number of times the inner and outer loops will iterate?

Q6. Implement Selection Sort algorithm on a similar set of data as in Q5 and (i)perform similar operations on the above example (ii) make a comparison between the two algorithms in terms of best case and worst case complexities

Session 2: Fractional Knapsack Problem

Introduction

In Greedy algorithm, the solution that looks best at a moment is selected with the hope that it will lead to the optimal solution. This is one of the approaches to solve optimization problems. This is an optimization problem which we want to solve it through greedy technique. In this problem , a Knapsack (or bag) of some capacity is considered which is to be filled with objects. We are given some objects with their weights and associated profits. The problem is to fill the given Knapsack with objects such that the sum of the profit associated with the objects that are included in the Knapsack is maximum. The constraint in this problem is that the sum of the weight of the objects that are included in the Knapsack should be less than or equal to the capacity of the Knapsack. However, the objects can be included in fractions to the Knapsack because of which this problem is termed as Fractional Knapsack problem.

Objectives

The main objectives of the session are to:

- Implement Fractional Knapsack Problem
- Test the implementation on different problem instances

- Implement greedy technique in general
- Problems for Implementations

Implement Fractional Knapsack algorithm and find out optimal result for the following problem instances:

Q1 (P_1, P_2, P3, P4, P5, P6, P_7) = (15, 5, 20, 8, 7, 20, 6)

(W_1, W_2, W3, W4,W5,W6,W7) = (3, 4, 6, 8, 2, 2, 3)

Maximum Knapsack Capacity = 18

Q2 (P_1, P_2, P_3, P_4, P_5) = (20, 30,40, 32, 55)

(W_1, W_2, W_3, W_4, W_5) = (5, 8, 10 12, 15)

Maximum Knapsack Capacity = 20

Q3 (P_1, P_2, P_3, P_4, P_5, P_6, P_7) = (12, 10, 8, 11, 14, 7, 9)

(W_1, W_2, W_3, W_4, W_5 , W_6, W_7) = (4, 6, 5, 7, 3, 1, 6)

Session 3 : Task Scheduling Algorithm

Introduction

A task scheduling problem is formulated as an optimization problem in which we need to determine the set of tasks from the given tasks that can be accomplished within their deadlines along with their order of scheduling such that the profit is maximum. So, this is a maximization optimization problem with a constraint that tasks must be completed within their specified deadlines.

Objectives

The main objectives of this session are to:

- Implement a task scheduling algorithm
- Test the algorithm on different problem instances
- Differentiate between a brute force approach and an efficient task scheduling algorithm

Problems for Implementation

Q1. Apply a brute force approach to schedule three jobs J1, J2 and J3 with service times as 5,8,12 respectively. The actual service time units are not relevant to the problems. Make all possible job schedules , calculate the total times spent in jobs by the system. Find the optimal schedule (total time). If there are N jobs , what would be the total number of job schedules?

Q2.Implement the task scheduling algorithm on your system to minimize the total amount of time spent in the system for the following problem:

Job	Service Time
1	5
2	10
3	7
4	8

Q3. Consider the following jobs, deadlines and profits. Implement the task scheduling algorithm with deadlines to maximize the total profits.

Jobs	Deadlines	Profits
1	3	50
2	4	20
3	5	70
4	3	15
5	2	10
6	1	47
7	1	60

Session 4: Huffman's Coding Algorithm

Introduction

Huffman coding is a greedy algorithm that is used to compress the data. Data can be sequence of characters and each character can occur multiple times in a data called as frequency of that character. Basically compression is a technique to reduce the size of the data. Huffman coding compress data by 70-80%. Huffman algorithm checks the frequency of each data, represent those data in the form of Huffman tree and build an optimal solution to represent each character as a binary string. Huffman coding is also known as variable length coding.

Objectives

The main objectives of this session are to:

- Implement Huffman's Coding algorithm.
- Test the implementation on different problem instances
- Construct the optimal binary prefix code
- Problems for Implementation

Q1 Apply Huffman's algorithm to construct an optimal binary prefix code for the letters and its frequencies in the following table.

Letters	A	B	I	M	S	X	Z
Frequency	10	7	15	8	10	5	2

Show the complete steps.

Q2. Implement Huffman's coding algorithm and run on the problem instance of Q1.

Q3 What is an optimal binary tree and Huffman code for the following set of frequencies? Find out an average number of bits required per character. Show the complete steps.

A:15 B:25 C:5 D:7 E:10 F:13 G:9

Q4. A file contains only colon, spaces, new line character and digits in the following frequency:

colon (80), space (500), new line (110), commas (500), 0 (300), 1 (200), 2 (150), 3(60), 4 (180), 5 (240), 6 (170), 7 (200), 8 (202).

Using the Huffman's algorithm to construct an optimal binary prefix code.

Session 5: Divide and Conquer Technique

Introduction

In Divide and conquer approach, the original problem is divided into two or more sub-problems recursively, till it is small enough to be solved easily. Each sub-problem is some fraction of the original problem. Next, the solutions of the sub-problems are combined together to generate the solution of the original problem.

Objectives

The main objectives of this session are to:

- Write recurrence relation of a problem
- Implementations of Binary Search, Merge Sort and Quick Sort algorithms
- Draw a tree of recursive calls of recursive calls

Problems for Implementations:

Q1 Implement a recursive binary search algorithm on your system to search for a number 100 in the following array of integers. Show the processes step by step:

10 35 40 45 50 55 60 65 70 100

Draw recursive calls to be made in this problem

Q2 Suppose that we are required to search among 512 million items in a list using binary search algorithm. What is the maximum number of searches needed to find a given item in the list or coming to the conclusion that the item is not found in the list.

Q3. Implement Merge Sort algorithm to sort the following list show the process step by step.

200 150 50 100 75 25 10 5

Draw a tree of recursive calls in this problem.

Q4 Implement Quick Sort's algorithm on your machine to do sorting of the following list of elements

12 20 22 16 25 18 8 10 6 15

Show step by step processes.

Q5. Examine the performance of Quick Sort algorithm implemented in the previous problem for the following list in terms of a number of comparisons, exchange operations and the number of times the loop will iterate?

6 8 10 12 15 16 18 20 22 25

Q6. Implement the Stassen's multiplication algorithm two matrices A and B of n* n, where n is a power of 2 on different problem instances and compare it(all the instances) in terms of a number of multiplications and additions required.

Session 6: Single Source Shortest Path Algorithm

Introduction

Dijkstra's algorithm solves the single-source shortest path problem when all edges have non-negative weights. It is a very similar to Prim's algorithm and always choose the path that are optimal right now but not for global optimizations

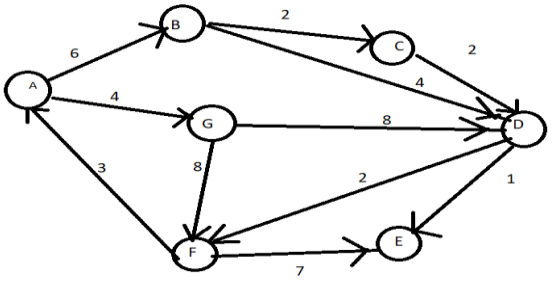
Objectives

Main objectives are to:

- Implement Dijkstra's algorithm to implement a single source shortest path using greedy technique
- Apply the implemented algorithm on different problem instances
- Represent a graph
- Find out the similarities in implementations of Prim's algorithm and Dijkstra's algorithm

Problems for Implementations:

Implement Dijkstra's algorithm to find the single source shortest path algorithm from different sources to the rest of nodes in the following graph and show allthe intermediate processes:



Q1 Find the shortest path from A to the rest of vertices

Q2 Find the shortest path from B to the rest of nodes

Q3. Find the shortest path from A to the rest of vertices but there are negative weights(-8) between G and F and (-3) between F and A

Session 7: Minimum Cost Spanning Tree

Introduction

A connected sub graph **S** of Graph **G**(**V**, **E**)is said to be spanning tree if and only if it contains all the vertices of the graph **G** and have minimum total weight of the edges of **G**. Spanning tree must be acyclic. A Spanning tree whose weight is minimum over all spanning trees is called a minimum spanning tree or MST

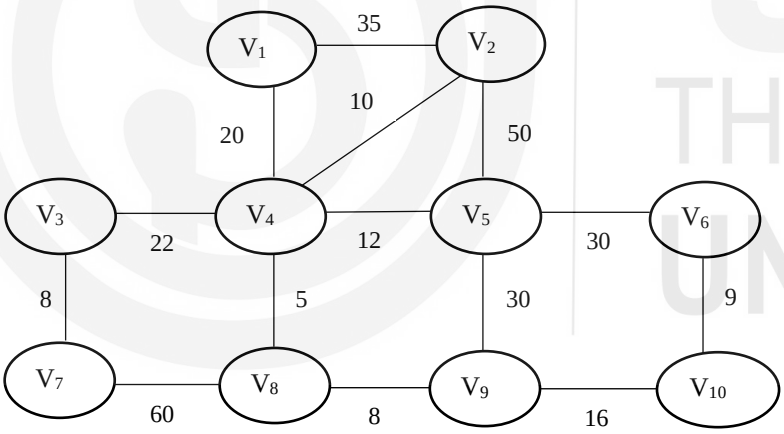
Objectives

The main objectives of the session are to:

- Implement Prim's algorithm to find a minimum cost spanning tree
- Implement Kruskal's algorithm to find a minimum cost spanning tree
- Differentiate between the implementations of two algorithms
- Implementation of disjoint data structure

Problem for Implementations:

Q1. Implement Prim's algorithm to find a minimum cost spanning tree (MCST) in the following graph. Show all the processes.



Q2. Implement Kruskal's algorithm to find a MCST for the following graph. Show all the processes.

Q3. Analyze the performance of both the algorithms on different problem instances and write a brief report

Session 8: Implementation of Binomial Coefficient Algorithm

Introduction

Binomial Coefficient is a part of permutation and combination. Computing binomial coefficients is non optimization problem but can be solved using dynamic programming. **Binomial Coefficient** is the coefficient in the

Binomial Theorem which is an arithmetic expansion. It is denoted as c(n, k) which is equal to $\frac{n!}{k! \times (n-k)!}$where ! denotes factorial.

gives combinations to choose k elements from n-element set. We can compute n_{c_k} for any n and k using the recurrence relation as follows:

$$n_{C_k} = \begin{cases} 1, & if\ k=0\ or\ n=k \\ n-1C_{k-1} + n-1C_k, & for\ n>k>0 \end{cases}$$

It is possible to compute binomial coefficient in linear time $O(n \times k)$using Dynamic Programming.

Objectives

The main objectives of this section are to:

- understand the application of binomial coefficient
- define binomial coefficient recursively
- solve the binomial coefficient problem through divide and conquer and dynamic programming techniques
- do performance analysis of both techniques on different problem instances for large and small values of n and k

Problems for Implementation:

Q1. Implement a binomial coefficient problem using Divide and Conquer technique

Q2. Implement a binomial coefficient problem using Dynamic Programming technique

Q3. Study the performance of both implementations using five problem instances in terms of the efficiency of both the approaches for large and small values of n and k in the problem instances.

Session 9: Floyd and Warshall's Algorithm for All Pair Shortest Path Algorithms

Introductions

Unlike the single source shortest path algorithm by Dijkstra's, Floyd Warshall's all pair shortest path algorithm computes the shortest path between any two vertices of a graph at one go. Floyd Warshall Algorithm uses the Dynamic Programming (DP) methodology. Unlike Greedy algorithms which always looks for local optimization, DP strives for global optimization that means DP does not rely on the intermediate(local) best results.

Main objectives of this session are to:

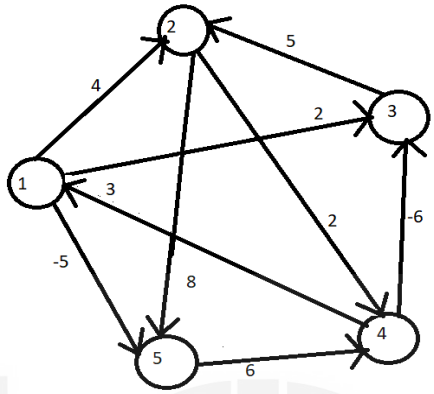
- Implement Floyd and Warshall's algorithm using dynamic programming

- Analyze the performance of the algorithm using different graph instances(large graphs, small graphs

Problems for Implementations

Q1. Apply the Floyd and Warshall's algorithm for the following graph. Show the matrix D^5 of the graph and find out the shortest path .

Q2. Apply the Floyd and Warshall's algorithm to compute the shortest path to the following graph. Compute D^5 matrix of the graph



Q3. Implement the all pair shortest path algorithms using different graphs

Session 10: Chained Matrix Multiplication

Introduction

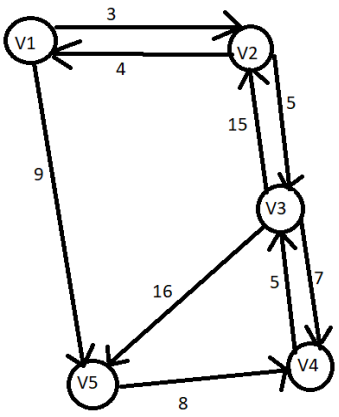
The problem of matrix-chain-multiplication is the process of selecting the optimal pair of matrices in every step in such a way that the overall cost of multiplication would be minimal. If there are total N matrices in the sequence then the total number of different ways of selecting matrix-pairs for multiplication will be ${}^{2n}C_n/(n+1)$. We need to find out the optimal order for multiplying n matrices.

Objectives

The main objectives of the sessions are to:

- List all the different orders in which we can multiply matrices
- Implement the chained matrix multiplications using dynamic programming technique

Problems for Implementations:



Q1. List different orders for evaluating the product of A,B,C,D,E matrices.

Q2. Find the optimal order for evaluating the product A * B * C *D *E where

- A is (10 * 4)
- B is (4 *5)
- C is (5 * 20)
- D is (20 * 2)
- E is (2* 50)

Q3. Implement the chained matrix multiplication and print the optimal parentheses algorithms and study the performance of the algorithms on different problem instances.

Session 11: Optimal Binary Search Tree

Introduction

The problem of optimal binary search tree is, given a keys and their probabilities, we have to generate a binary search tree such that the searching cost should be minimum. Using dynamic programming we can find the optimal binary search tree without drawing each tree and calculating the cost of each tree. Thus, dynamic programming gives better, easiest and fastest method for trying out all possible binary search tree and picking up best one without drawing each sub tree

Objectives

The main objectives of this session are to:

- Determine the cost and structure of an optimal binary search tree
- Implement an optimal binary search tree and study the performance of the algorithm using different problem instances

Problems for Implementation:

Determine the cost and structure of an optimal binary search tree for a set of n=7 keys with the following properties. Show the step by step processes.

Q1.

i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

Q2.

I	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Q3 Implement the optimal binary search tree algorithm on your system and study the performance of the algorithm on different problem instances

SECTION 2 WEB DESIGN LAB

Structure of sessions **Page Nos.**

Introduction

JavaScript is a popular web programming language. It is generally used to validate the forms. JavaScript is based on an object-oriented paradigm. In this lab, you will learn the advanced JavaScript concepts with hands-on practices.

The lab manual contains ten sessions. The first two sessions cover the basic concepts on HTML forms and CSS. Then, you will be redirected to learn the advanced JavaScript concepts. Each session introduces the topics & with examples. Some programming exercises are provided for the hands-on practise. From session 3 onwards, you will be able to learn the advanced JavaScript concepts like HTML DOM, error handling, object-oriented programming, cookies, and JSON.

Objectives

After completion of this course, student will be able to learn

- Basics of HTML forms and CSS
- The HTML DOM functionality, events and navigations
- Error Handling in JavaScript
- asynchronous programming in JavaScript

Session 1: Preliminaries

1.0 Introduction

JavaScript has nothing to do with Java, is dynamically typed and has C-like syntax. JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities. In this session we will introduce the basic JavaScript constructs, conditional statement and control flow statements.

1.1 Objectives

The objectives of this session are:

- Understand the basic JavaScript constructs
- Understand conditional statements and control flow statements

1.2 Enabling JavaScript

JavaScript can be included in html in two ways: Internal and external file.

- **Internal JavaScript to HTML file:**

  JavaScript can be added to head, body or both sections.

```
<script>
  x = 3;
</script>
```

- **Reference external file:**

```
<head>
<script type = "text/javascript" src = "filename.js" ></script>
</head>
```

OR

```
<head>
```

```
<script src="http://example.com/script.js"></script>

</head>
```

1.3 Conditionals and Loops:

Javascript provides conditional statements i.e. if. . . else, nested if. . . .else and switch. . .case. Following are the syntax of conditional statements in Javascript:

- **if. . . else**

  Syntax:

  ```
  if (condition1)

  {
    //  statements to be executed if condition1 is true
  } else if (condition2) {
    //  statements to be executed if the condition1 is false and condition2 is true
  } else

  {
    //  block of code to be executed if both condition1 and condition2 are false

  }
  ```

  Example 1

  ```
  if (str == "Hello")
  {    // if-else
   alert("Hi");      // popup dialog
  }
  else
  {
  alert("something is wrong!");
  }
  ```

  Output:
  Based on the value of str, popoup dialog box will be shown.

- **switch. . . . case**

  Syntax:

  ```
  switch(expression)
  {
    case p:
      // code block
      break;
    case q:
      // code block
      break;
    default:
      // code block
  }
  ```

Example 2

```
switch (name)
{     // switch statement
  case "Bob":
alert("Hi Bob!")
    break
  case "Joe":
alert("Hey Joe.")
    break
  default: alert("Do I know you?")
}
```

Output:

Based on the value of the variable name, the corresponding case statement will be executed, if none of the case is true, the default case will be executed.

Loops

Loops are used to execute certain statements multiple number of times. In javascript while and for loops are available to use.

- **while loop:**

  Syntax:

  ```
  while (condition)
  {
    // statements to be executed if condition is true
  }
  ```

  Example:

  ```
  while (i<n)
  {         // the basics
   // do something
  i++;
  }
  ```

  Output:

  The code block "do something" within while loop is executed until the condition i<n, is true.

- **for loop:**

  Syntax:

  ```
  for (statement1; statement2; statement3)
  {
    // for loop code block to be executed if statement2 is true
  }
  ```

  Where,

  Statement1 is executed only once, before the execution of the loop code block.

  Statement2 is the condition statement. The loop code block is executed only when this condition is true.

Statement3 gets executed each time after execution of the loop code block.

Example 3

```
for (var i=0; i<n; i++)
{
  // loop code block
}
```

Output:

Here,

1. 1st vari=0 statement is executed,

2. i<n is executed,

3. if in step 2 (i<n) statement is true, loop code block is executed.

4. after finishing loop code block, i++ is executed.

This completes 1st cycle.

5. Now, omitting statement1 (var i=0), statement2 (i<n) is executed.

6. If statement2 (in step 5) is true, loop code block is executed.

7. After finishing loop code block statement3 (i++) is executed.

This completes 2nd cycle.

Now, 2nd cycle is executed until the condition meets false value.

1.4 Functions in JavaScript

Similar to other programming languages, javascript also enables the use of functions. Functions are the code block written to meet a specific task.

Syntax:

```
function function_name(parameter1, parameter2, parameter3)
{
  // function code block to be executed
}
```

Example 4

```
function foo(x,y)
```

```
{
    //function code block to be executed
 return x*y;
}
```

1.5 Exercises

1. Write a JavaScript code to perform the two-digit arithmetic operations: Addition, Subtraction, Multiplication and Division.

2. Write a JavaScript code to print 1 for 1 time, 2 for 2 times, 3 for 3 times and so on upto 10.

3. Design a web page that describes you. Your page must include following details (extra details and creativity are welcome):

- Your personal details, i.e. name, gender etc.
- 2-3 lines describing yourself as About Me. Make the most important word bold to emphasize them.
- Details of classes in tabular form you are taking right now (as per your time table).
- Details of your favourite movies, books, or TV shows, in reverse order(using CSS), list atleast 3 from each. You can use images, hyperlinks, colourful text to make this section attractive.
- Include a section showing your interest and also not of your interest jobs. In this section also you can use images, showing you're happy in doing your interest jobs and the other to represent you when you're sad.
- In this section show something interesting about one or more people of your neighbours.

2.0 Session 2: HTML Forms

2.0 Introduction

This session provides basic understanding of creating and using the HTML form. We will learn, how to use Form elements in HTML to collect the information from user such as input, text field, label, radio button, text area, submit button etc. It would be desired to practise all the examples during the session as well as exercises.

2.1 Objectives

The objectives of this session are as follows:

- Understand the HTML Form element.
- Use the form element in HTML
- Use HTML to acquire user input and process it.
- Create forms with basic elements like textboxes checkboxes, radio buttons and submit buttons.
- Create forms using HTML5 elements like E-mail address field and form validation.

2.2 HTML Forms Elements

In HTML, user input is collected using the HTML Form element. The data collected from user is sent to server for processing. **HTML Form**: a document, capable of storing data on a web server, provided by the user at run time. It is usually used to collect information like: user name, user password, contact details like: mobile number, e-mail id etc. Some of the commonly used elements in an **HTML form** are: input box, text box, check box, radio buttons, submit buttons etc.

Basic Elements of HTML forms:

1. The <form> Element:To make a dynamic HTML webpage, in real time user inputs are required. In HTML, the element <form> is used trough which user inputs are acquired.

The format of <form> element is as follows:

```
<form>
.
form elements
.
</form>
```

User inputs can be acquired in many forms like: submit button, text, radio button, checkbox etc. Some of them are describe below:

The <input> Element:

It is used to acquire user inputs

Following is the syntax of <input> element:

```
<input type="">:
```

Here, type defines the way to acquire the user input i.e. text, radio button, checkbox, submit button.

Let's discuss the commonly used attributes of input element: -

type="text"

It creates a text field, used to acquire user input in the form of text.

Example:

```
<form>
 <input type="text" id="lname" name="lname">
</form>
```

type="radio"

It creates radio button, used to acquire the user selection choice. Radio buttons are used for selecting one out of many options.

Example:

```
<p>Choose an option:</p>
<form>
  <input type="radio" id="opt1" name="opt" value="Option1">
  <label for="opt1">Option1</label><br>
  <input type="radio" id="opt2" name="opt" value="Option2">
  <label for="opt2">Option2</label><br>
```

```
  <input type="radio" id="opt3" name="opt" value="Option3">
  <label for="opt3">JavaScript</label>
</form>
```

type="checkbox"

It creates checkbox used to acquire the user choices. checkboxes are used for selecting none or more than one choice.

Example:

```
<form>
  <input type="chkbox" id="car1" name="car1" value="Maruti">
  <label for="car1"> I own a Maruti Car</label><br>
  <input type="chkbox" id="Car2" name="Car2" value="Hyundai">
  <label for="Car2"> I own a Hyundai Car</label><br>
  <input type="chkbox" id="Car3" name="Car3" value="Ford”>
  <label for="Car3"> I Own a Ford Car</label>
</form>
```

type="submit"

It creates a submit button, which can be used to submit the form.

Example:

```
<form action="/actions.php">
  <label for="f_name">First name:</label><br>
  <input type="text" id="f_name" name="f_name" value="John"><br>
  <label for="l_name">Last name:</label><br>
  <input type="text" id="l_name" name="l_name" value="Li"><br><br>
  <input type="submit" value="Submit">
</form>
```

type="button"

It creates a clickable button.

type="email"

In HTML5 new input field to acquire Email address from user was introduced.

Syntax:

```
<input type="email">
```

Example:

```
<form>
<input type="email">
<input type="submit" value="Submit">
</form>
```

To allow multiple Email Ids at once, use following syntax:

```
<input type="email" multiple>
```

While entering multiple Email Ids use , to separate each other.

type="search"

Whether it is Google, social sites or e-commerce sites, searching for desired content is what most of the user perform today. HTML5 provides a search field to use in your website.

Example:

```
<form >
```

```
<input type="search" name="search">
</form>
```

type="url"

Like Email field HTML5 also provide the url field to acquire web addresses. Browser also performs simple validation to check the syntax correctness of the url entered.

Syntax:

```
<input type="url" name = “url>
```

type = “date”

Upto HTML4, the date pickers were to be constructed with JavaScript libraries. HTML5 enables this functionality possible within the browser.

To show the calendar, following syntax can be used:

```
<input type = “date” id="dob" name="dob">
```

To show the month and year only:

```
<input type = “month” id="expiry" name="expiry" required>
```

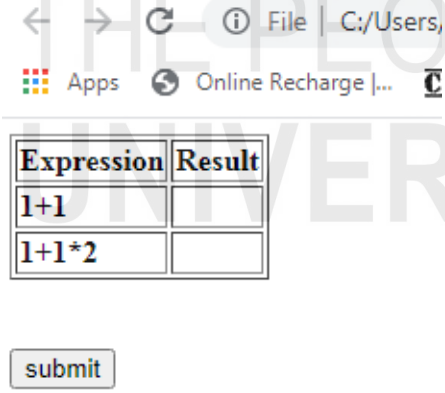
To show the time only:

```
<input type=”time” id="time" name="time">
```

Note: The name attribute in <input> element is very important. The value stored in the input element will not be sent if the name attribute is missing.

2.3 Exercises

1. Create an HTML webpage, as shown in figure. The user can specify any expression as input. Use form element to collect the user data.



2. Create a scrolling list that shows five items from the following list of PC processors: Intel i7, Motorola 68000, AMD Athlon , Intel 8088, AMD Ryzon, Intel Pentium MMX, Intel i3, Intel Pentium II, Intel Pentium III, Intel i5, Intel Celeron, PowerPC G3, PowerPC G4,.

3. Design a webpage for a software company to accept on-line job applications. The webpage must be designed to use form with elements:

- Position applied for (autofocus), name, nationality, date of birth (selected from an auto picker), address (in a text area), telephone number and email (required).
- Educational history and qualifications.

- Work experience/employment/training in terms of employer history and number of years of experience selected from a slider. Set maximum years of experience to 10 years.
- Personal statement.
- Two referees including names, occupation, relationship, address, telephone.

Session 3: Cascading Style Sheet (CSS)

3.0 Introduction

Web pages belong to the same website, each page should have a consistent look in order to provide familiarity for the user. Style sheets are used in desktop publishing to provide consistency when formatting text. When a big project containing large number of web pages and elements, is handled by a team of web developers, requires extra efforts to maintain similar design and style among all the developers. To make this task easy, CSS is used to style and layout web pages. Using CSS, style attributes of HTML elements such as: font, color, size, and spacing of your content, split it into multiple columns, or add animations and other decorative features can be set.
This session is a recap session for CSS covering the basics of how it works, what the syntax looks like, and how you can start using it to add styling to HTML.

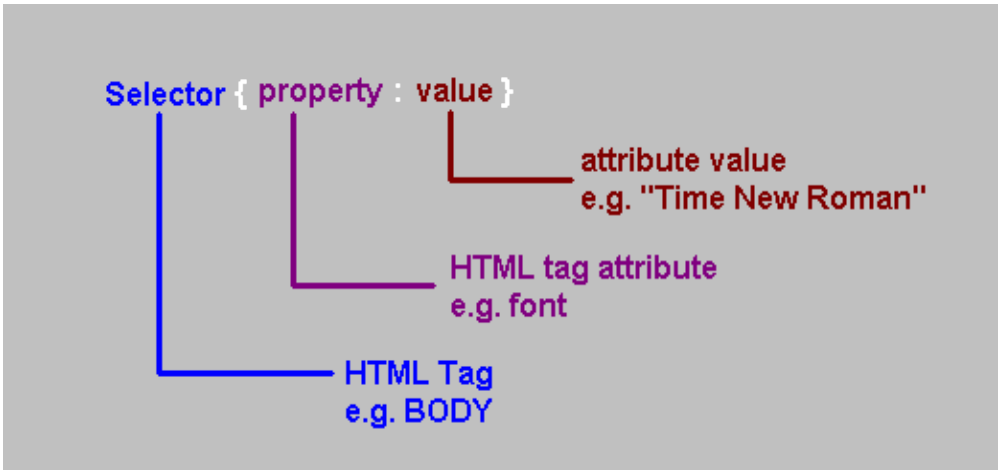
3.1 Objectives

After completing this session, one should be able to:

- Create CSS for a webpage
- Embed CSS inline and external into an HTML code.

3.2 What is CSS?

A style sheet is a collection of rules describing the format and style of the HTML elements/tags. CSS can be embedded into HTML code as: internal CSS and external CSS.
Structure of CSS:



3.2.1 Internal CSS:

Internal CSS is included within the <head> element. It can be used to style a single HTML page only. It is accessible to all the elements of that HTML document i.e. it is globally defined in the HTML file.
Syntax:

```
<style>
HTML_tag
{
  style attributes
}
</style>
```

Example:

```
<html>
<head>
<style>
body
{
 background-color: green;
}

h1
{
 margin-left: 20px;
color: red;

}
</style>
</head>
<body>

<h2>It is heading under internal CSS</h2>
<p> It is paragraph under internal CSS.</p>
</body>
</html>
```

3.2.2 Inline CSS:

Inline CSS is specific to the element. It is included within the element to which it has to be applied. It is element specific style design.
Syntax:

```
<HTML_Tag style="style_attributes">Content of Tag</HTML_Tag>
```

Example:

```
<html>
<body>

<h2 style="text-align:right; color:red;"> It is heading under inline CSS </h2>
<p style="text-align:justify; color:green;"> It is paragraph under inline CSS.</p>
```

```
</body>
</html>
```

3.2.3 External CSS:

External CSS is written in a file with extension .css and can be included in any HTML page as desired to be style with it. For website with many web pages, it is very easy to maintain and modify the style and design of the whole website using external style sheet, by just updating the one file: CSS. Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

Syntax:

```
<link rel="stylesheet" href="external_css.css">
```

This should be included within <head> tag.

Example:

```
<html>
<head>
<link rel="stylesheet" href="external_css.css">
</head>
<body>

<h2> It is heading under external CSS</h2>
<p> It is paragraph under external CSS.</p>

</body>
</html>
```

The external_css.css file looks like this:

```
body
{
 background-color: lightblue;
}

h2
{
 color: navy;
 margin-left: 20px;
}
```

Here, in this body and h2 are the HTML elements for which style attributes are defined in this CSS file.

When multiple styles are available to a web page, all of them will "cascade" into a new "virtual" style sheet by the virtue of the following rules, with first one having the highest priority:

1. Inline style
2. External and internal style sheets
3. Browser default

That is, an inline style has the highest priority, and will override all other three styles.

3.4 Exercises

1. Create an HTML website to display the time table of all the 4 years of B. Tech. (even semester of each year). It should contain a home page where link to each year's time table is listed. On click to any year time table a new web page should be open displaying the corresponding time table. Apply the following effects on the table using CSS:

- Common to all:
  - Display day names (Mon, Tue etc...) in bold format with the first letter in the dayname in uppercase.
  - Display lunch slightly in bigger font other than the remaining text.
- Specific to each year:
  - For 1st year
    - Apply blue as the background colour and white for the colour of the text in the table header.
  - For 2nd year
    - Apply green as the background colour and white for the colour of the text in the table header.
  - For 3rd year
    - Apply purple as the background colour and white for the colour of the text in the table header.
  - For 4th year
    - Apply yellow as the background colour and black for the colour of the text in the table header.

2. Create an HTML webpage to show personal information i.e. name, class, qualifications, photo,address etc. Make use of tables as when possible. Apply the following styleinformation using CSS:

- Display the heading of the page in Arial font and with font size of 18px.
- Align all the field names like Name, Class, Photo etc to left in the table.
- Apply yellow color as background colour in left side cells contains field names like Name, Class etc...
- Set college logo as background image to the web page.

Session 4: JavaScript HTML DOM

4.0 Introduction

In this session, we will discuss the DOM(Document Object Model). DOM is a standard defined by W3C (World Wide Web Consortium). W3C DOM is an interface, which allows programs and scripts to access, add, modify and even remove the *content, structure, and style of an html document. It is a language independent interface.*

The W3C DOM is categorized into 3 parts as follows:

- Core DOM – it is the standard model for all document types
- XML DOM – it is the standard model for XML documents
- HTML DOM – it is the standard model for HTML documents

In this Section we will be discussing about the HTML DOM only.

4.1 Objectives:

After completing this session one should be able to:

- Add, modify or delete the existing content of HTML elements
- Modify a CSS style in the html page

react to existing HTML DOM events in the html page.

4.2 HTML DOM

HTML DOM stands for Hyper Text Markup Language Document Object Model. When a browser loads an html page, it creates a DOM of this html page. HTML DOM provides ability to JavaScript to access and modify any element of an HTML document.

HTML DOM is constructed in the form of a tree to show the parent-child relationship of various elements of the HTML page.

HTML DOM considers:

- The entire document as a document node
- Each HTML element as element node
- Text content within an HTML element as text nodes
- Comments as comment nodes

Below is an example of HTML DOM for the shown HTML page.

code 1:

```
<html>
<head>
<title>My Title></title>
<body>

<p>My first paragraph.
<h1>My First Heading</h1>
</p>
<!This is comment>
</body>
</html>
```

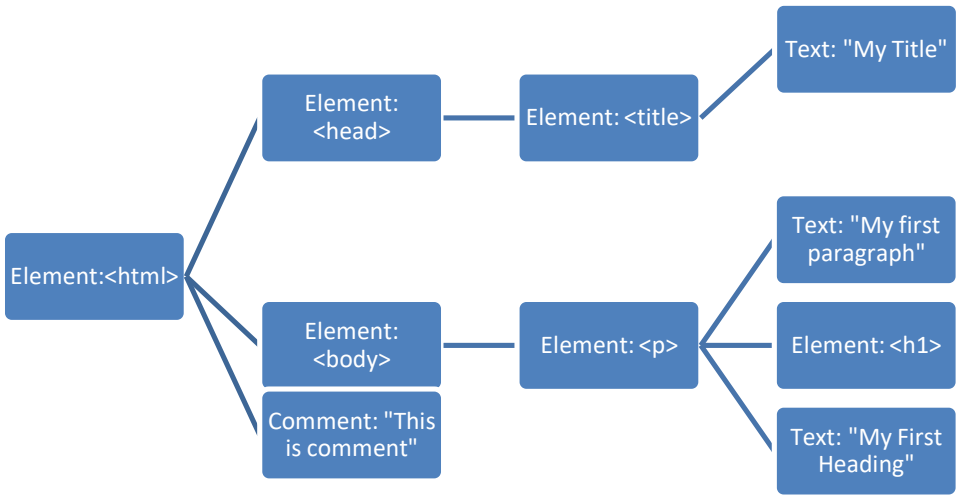


Figure 1: HTML DOM of the HTML page shown in code 1.

From the above HTML code and figure, we can read as:

- <html> is the root node of the page and is the parent node of <head> and <body>
- <head> is the first child of <html>
- <body> is the last child of <html>
- <head> has a child node: <title>
- <title> has a child (a text node): "DOM Tutorial"
- <body> has two children: <h1> and <p>
- <h1> has one child (a text node): "DOM Lesson one"
- <p> has one child: "Hello world!"
- <h1> and <p> are siblings

The HTML DOM is a standard for how to get, change, add, or delete HTML elements.Following are the capabilities of JavaScript with HTML DOM. It can:

- add new HTML element or attributes of an element in the html page.
- modify any of the HTML elements or attributes of an element in the html page.
- modify a CSS style in the html page
- remove any of the HTML elements or attributes of an element in the html page.
- react to any of the existing HTML events in the html page.
- add new HTML events in the html page.

4.2.1 HTML DOM Methods and Properties

JavaScript objects consist of properties and methods. When a web browser renders an HTML document, it creates a DOM of HTML document. As we know in DOM every HTML element is treated as a JavaScript Object which has a number of method and properties associated with it.

HTML DOM Methods: They are the actions could be performed on Elements of HTML page.

HTML DOM properties: These are the **values** or attributes Elements of HTML page that could be set or updated.

4.2.2 The DOM Programming Interface

The DOM programming interface is the actions or methods and values or properties of an object/HTML Element.

As discussed above the **property** of an object is a value that can be get or set i.e. updating the content of an HTML element.To update an HTML element, first we need to access it by finding that particular element in the whole document or the web page.As we have already learned the methods to access or find an HTML element in the HTML course, an element can be accessed by its ID, Tag Name or by the Class Name. Out of these thewidely used method is accessing the element by its ID i.e.

document.getElementById(id)

Following are the properties used to change an HTML Elements:

Property	Description
element.innerHTML = *new content*	Update the content of element: element (between opening and closing tags) with content: new content
element.attribute = new attribute value	Update the attribute values of the element: element with the new attribute values
element.style.*property* = *new style*	Update the style of the element: element with the new style

Following are the methods used to add or remove HTML Elements:

Method	Description
document.createElement("*element_name"*)	Create a new element: element_name under the element: document. Ex: document.createElement("p"); It creates a new element <p> under the body part.
document.removeChild(child_*element*)	Remove the element: child_element whose parent is element: document.
document.appendChild(child_*element*)	Add element: child_element under

	the element: document.
document.replaceChild(*new, old*)	Replace element: old, with element: new of the parent element document.

Following are the methods used to add event handlers:

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	It adds an onclick event to the element with ID: id,whose code is written in function: function()

Example 1: Write a javascript to change the contents of the paragraph element.
Solution:
<p> with id="para1".

```
<html>
<body>
<p id="para1">This is to be updated using innerHTML</p>
<script>
document.getElementById("para1").innerHTML = "We have replaced using
innerHTML";
</script>
</body>
</html>
```

Result:

We have replaced using innerHTML

Explanation:

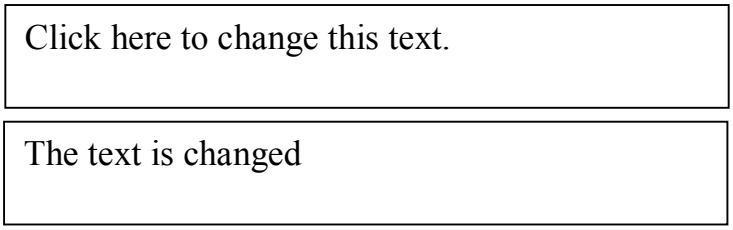
The id of an element is used to access an HTML element. Here, the method getElementById is used to find the element with id="para1".
The contents of any HTML element (even <html> and <body>) can be accessed and updated using the innerHTML property.

Example 2: Write an HTML code with JavaScript in which the contents of the paragraph element
Solution:
<p> with id="para1"changes on click event.

```
<!DOCTYPE html>
<html>
<body>
<p id="para1" onclick="Fun()">Click here to change  this text.</p>
<script>
function Fun()
```

```
{
document.getElementById("para1").innerHTML = "The text is changed";
}
</script>
</body>
</html>
```

Result: Initially below page is shown:



On mouse click on the text, below page is shown:

Explanation:
The id of an element is used to access an HTML element. Here, the method getElementById is used to find the element with id="para1".

4.2.3 Updating style of an HTML Element

Following syntax is used to change the style of an HTML element using HTML DOM:
document.getElementById(element_*id*).style.*property* = *new style*

Example:
Changing the color of content of the element: h2 using HTML DOM.

```
<!DOCTYPE html>
<html>
<head>
<title>Style change</title>
</head>
<body>

<h2 id="heading1">Old heading in red color</h2>
<p> the heading color changed from red to green</p>
<script>
document.getElementById("heading1").style.color = "green";
</script>

</body>
</html>
```

Result: As shown below the color of the content of h2 element is changed from red to green.

Old heading in red color

the heading color changed from red to green

4.3 Exercises

1. Design an HTML page using JavaScript as shown below. When Try button is pressed, the Full Name should be shown below the Try button. (Hint: Use concatenation to get Full Name.

Accessing all the elements of HTML Form with document.forms

First name: Bhagat
Last name: Singh

Click on "Try" button to show the value of elements in the form.

Try

On Click on Try:

Accessing all the elements of HTML Form with document.forms

First name: Bhagat
Last name: Singh

Click on "Try" button to show the value of elements in the form.

Try

Bhagat
Singh

2. Modify the below snippet of the code to change the src attribute of the image element to “pqr.jpg” using HTML DOM.

```
<img id="pic" src="old.jpg">
<script>
//Write your code here.
</script>
```

3. Write the JavaScript code to change the color of the HTML element h2 to green, on click the “change color” button.

Session 5: JavaScript HTML DOM events and Event Listener

5.0 Introduction

JavaScript is capable of executing a script on occurrence of an event in HTML document i.e. on mouse click event on an HTML element like button. The JavaScript code is added to the event attribute of the HTML element. In this session, you will learn the about adding or removing the event listeners and methods about event listener.

5.1 Objectives

After completing this session, one should be able to:

- Add, modify or delete the HTML DOM events
- Work with event listeners
- Perform actions based on events using JavaScript

5.2 DOM Events

Following are the list of events commonly used in HTML DOM:

- Mouse click on an element, like: button, anchor, image etc.
- On load of a web page.
- On load of an image.
- On mouse movement over an element.
- On change in input field value.
- On submission of HTML form.
- On key press from keyboard etc.

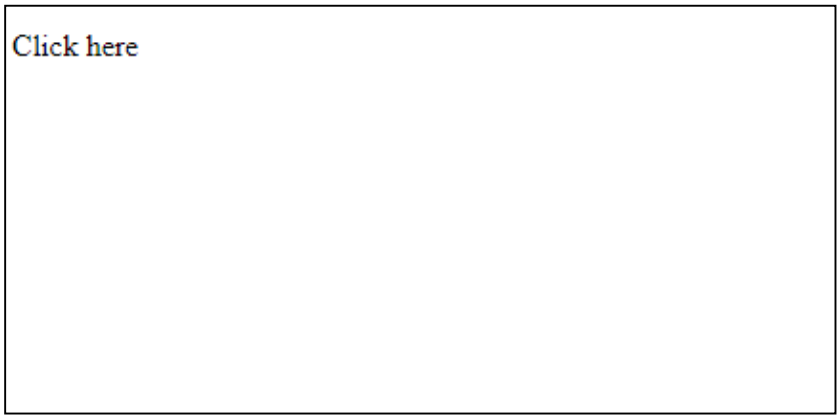
Example 1: Write a javascript code to change the contents of an element like header tag or paragraph tag or a division tag etc when user clicks on it.

Solution:

```
<!DOCTYPE html>
<html>
<head>
<title> DOM Events</title>
</head>
<body>
<p id="para1", onclick="abc()">Click here</p>
<script>
function abc()
{
  para1.innerHTML = "This is changed text after click";
}
```

```
</script>
</body>
</html>
```

Result: Before click:



After click:

5.2.1 Adding an event listener of an element

Event Listeners are attached to elements and fired when certain event occurs.addEventListener() is a method which is used to attach an event handler to an element. This method adds an event handler in addition to the existing ones i.e. existing event handlers are not overwritten. In this manner more than one event handlers (even of the same type) can be attached to one element.

When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

Similarly, an existing event handler to an element can be removed using the method: removeEventListener().

Syntax

element.addEventListener(*event, function, useCapture*);

Where,

event: is the type of the event

function: in this the actions to be performed on event are defined.

useCapture: it is a Boolean value and is optional field. it is used to specify whether to use event bubbling or event capturing.

They specify the order of event propagation in the HTML DOM. Consider following snippet of HTML page:

```
<div>
<p>
        </p>
</div>
```

Both the elements: div and p are implemented with event handlers.

What will be the order of execution of events, if <p> element is clicked 1st?

In *bubbling* the inner most element's event is handled 1st and then the outer element's event i.e. the event of <p> element is handled 1st and then the event of element. To use bubbling method, the value of useCapture is passed as Boolean False.

In *capturing* the outer most element's event is handled 1stand then the inner element's event i.e. the event of element handled 1st and then the event associated with <p> element. To use capturing method, the value of useCapture is passed as Boolean True.

By defaultbubbling propagation is used i.e. useCapture is set to false.

Example2

Add an event handler to the element button, where onclick an alert message is shown.

Solution:

```
<!DOCTYPE html>
<html>
<head>
<title>Add Event Listener</title>
</head>
<body>
<h3>JavaScript HTML DOM addEventListener()</h3>
<p>Click on submit button below to which addEventListener() method is associated
and shows an alert message on button click.</p>
<button id="btn1">Submit</button>
<script>
document.getElementById("btn1").addEventListener("click", abc);

function abc() {
  alert ("Event handler!");
}
</script>
</body>
</html>
```

Example 3

Add two click events to the element: button, where in first clickan alert message "This is first click" is shown and in second click an alert message "This is second click" is shown.

Solution:

```
<!DOCTYPE html>
<html>
<head>
<title>Adding Multiple Event Listener to an Element</title>
</head>
<body>
<h3> Adding Multiple Event Listener to an Element </h3>
<p>Click on submit button below to which to see the effect of multiple event listeners added to an element.</p>
<button id="btn1">Submit</button>
<script>
document.getElementById("btn1").addEventListener("click", abc1);
document.getElementById("btn1").addEventListener("click", abc2);

function abc1()
{
  alert ("This is first click");
}
function abc2()
{
  alert ("This is second click");
}
</script>
</body>
</html>
```

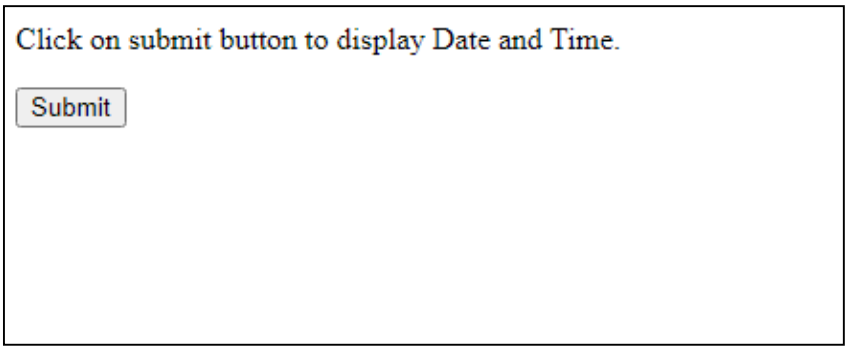
In the function called on an event, parameters can be passed. Parameter passing is done with the help of an "anonymous function".
Syntax:
element.addEventListener("click", function(){ my_Function(a, b); });

5.2.2 Remove an event listener of an element

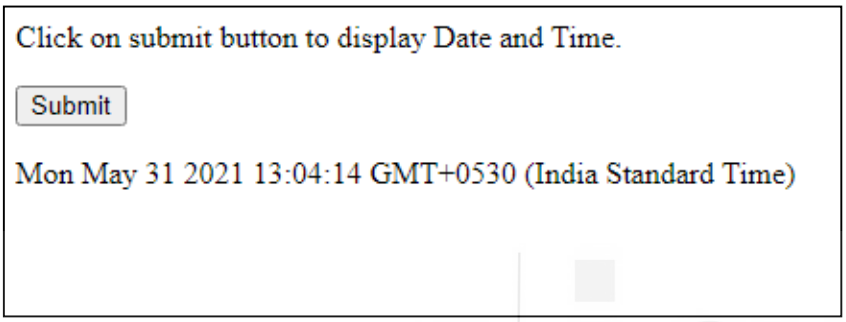
The method: removeEventListener() is used to removes an event handler of an element attached with the addEventListener() method.
Syntax:
element.removeEventListener("event", function);

5.3 Exercises

1. Design an HTML page as shown below. Implement the event "On button click display the date" using HTML DOM events.
Before click on submit button:



After click on submit button:



2. Design an HTML page to shown that, when this page is loaded it must check the cookies status of user's browser and display a message accordingly in the popup box i.e. if cookies are enabled show: "Cookies are enabled" and if not then show: "Cookies are disabled". Implement the event "On load" using HTML DOM events.

3.Whenever user input a text in the text box in any case (i.e. upper or lower case), automatically it should be converted to uppercase. Design an HTML page to implement above functionality using HTML DOM event "onChange".

4. Design HTML page to implement HTML DOM events by changing an image on the following events: onmouseoveronmouseout, onmousedown, and onmouseup Events.

Session 6: HTML DOM Navigations

6.0 Introduction

As discussed earlier in the Introduction section, the HTML DOM interprets the HTML page as the nodes with respect to the elements. These nodes can be navigated with the help of the node tree using node relationships.

6.1 Objectives

After completing this session, one should be able to:

- Add the navigations between nodes
- Create and remove HTML DOM elements

6.2

Following node properties can be used to navigating between nodes in JavaScript code:

- parentNode
- childNodes[*nodenumber*]
- firstChild
- lastChild
- nextSibling
- previousSibling

A node can be accessed either by its ID or by its class or by navigation properties of node.
Considering the HTML code below, following are the ways nodes can be accessed using navigational properties of node:

```
<!DOCTYPE html>
<html>
<head>
<title>1st child of title element.</title>
</head>
<body>
<h3 id="heading1"> 1st child of body element (which is the last child or 2nd child of the root element </h3>
<p id="para1">Second or the last child node of body element.</p>
</body>
</html>
```

1. Using ID of the element:

```
var elemt = document.getElementById("heading1").innerHTML;
```

2. Using the parent – child relationship:

```
var elemt = document.getElementById("heading1").firstChild.nodeValue;
```

Here, document.getElementById("heading1") returns the access id of heading1 element, and document.getElementById("heading1").firstChild points to the 1st child which is the text content of heading1 element, and document.getElementById("heading1").firstChild.nodeValue returns the value of firstChild element of heading 1 element which is "1st child of body element (which is the last child or 2nd child of the root element" in this case".
nodeValue is the property of node to access the value of the node.

The first child can also be accessed as follows:

```
var myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```

Where, childNodes is an array which stores the list of child nodes of an element.
Other than nodeValue, some of the commonly used node property is: nodeName. It specifies the name of the tag of an HTML element or node, which is always retuned in uppercase.

6.3 Creating and removing HTML elements using HTML DOM

An HTML element is created 1st and then appended before or after to an existing element.

The following syntax is used to create a new HTML element:

```
var new_element = element.createElement("tag_name");
```

This creates a new element with tag name: tag_name, and returns the id of this as return value.

Example 3

var para1 = document.createElement("p");
This will create a new element <p> with id = para1. At present this element is not associated with any element of the HTML document.
Note that, <> symbols are not used while specifying the name of the new element to be created.
To add the text to this <p> element, following command can be used:
var text_node = document.createTextNode("A new text node is created");
By this, we have created two new nodes represented by variables: para1 and text_node.
First this text node is to be added to the <p> node, which can be done using following syntax:
para1.appendChild(text_node);
Finally, the <p> element has to be appended to an existing HTML document's element. This can be done as follows:
Considering, there exists a element with id =d1 to which this newly created <p> element is to be added.
document.getElementById("d1").appendChild(para1);
The appendChild() adds the new element as the last child of the parent.
The insertBefore() method can be used instead of appendChild() to add a child at a specific position among the children of parent.
Syntax:
element.insertBefore(new_element,existing_element_before_new_is_inserted);
Here, the new_element is inserted as the child of the element:element, at the position: before the element: existing_element_before_new_is_inserted.

6.3.1 Removing an HTML element

To remove an existing HTML element, remove() method is used.
Syntax:
element.remove()
The element: element will be dropped.

6.3.2 Removing a child element

To remove a child element of a parent removeChild() method is used.
Syntax:
parent.removeChild(child);
The element: child of the parent node: parent will be dropped.

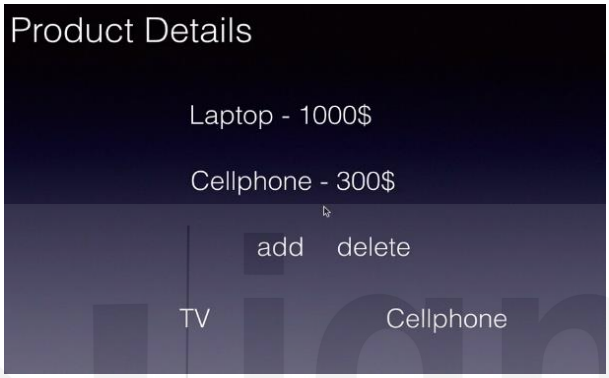
6.4 Exercises

1. Which function on the document object should be used to retrieve a HTML element uniquely?

2. Which JavaScript function can be used to dynamically add a event listener to a HTML Element?

3. Which method should be used to add a new HTML child element?

4. Using the DOM API, create a product details page which shows the prices of a laptop and a cell phone. You may paragraph tag to wrap these elements. When the user clicks on add button, you should able to add TV with price details and when you click on delete button then cell phone details will be deleted.



Session 7: JavaScript Error Handling

7.0 Introduction

When executing a code or script, errors may occur. These errors can be mistakes made in coding by the programmer, errors caused by providing wrong input, and due to unforeseeable things.

7.1 Objectives

After completing this session, one should be able to:

- Understand error handling mechanism
- Implement the different error cases with nested try catch blocks

7.2 Java Script error handling methods

The main terms that are used in JavaScript to handle the errors are describe as follows:
try: set of statements required to be executed in which the errors may occurred and are required to be addressed.

syntax:

```
try
{
    set of statements to be executed.
}
```

catch: set of statements written to handle the error occurred in try segment.
Syntax:

```
catch
{
        set of statements to be executed when error occurs in try segment to handle
        this error.
}
```

throw: set of statements used to create custom errors.

Syntax:

```
throw "Empty field";    // throw a text
throw 100;              // throw a number
```

finally: set of statements executed, after try and catch segments, irrespective of their result.

Syntax:

```
finally
{
        set of statements executed, after try and catch segments, irrespective of their
        result.
}
```

Putting all together:

```
try
{
 set of statements to be executed.
}
catch(err)
{
 set of statements to be executed when error occurs in
 try segment to handle this error.
}
finally
{
 set of statements executed, after try and catch
 segments, irrespective of their result.
}
```

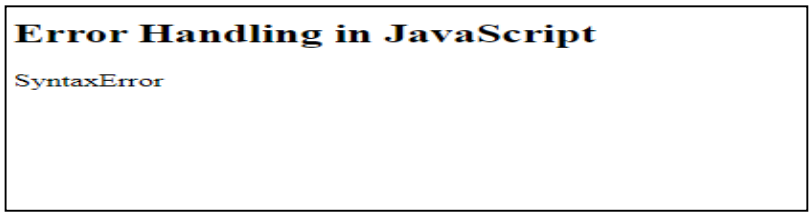
Example 1

In following HTML code error handling is used to address any error in the code.

```
<html>
<head>
<title>Error Handling in javascript</title>
</head>
<body>

<h2>Error Handling in JavaScript</h2>
<p id="para1"></p>
<script>
try
{
 eval("2=x");
}
```

```
catch(err)
{
document.getElementById("para1").innerHTML = err.name;
}
</script>
</body>
</html>
```

Result:



Explanation:
as the function eval(), evaluates the expression passed as argument. In the code in try segment, eval("2=x") shows a syntax error, due to the expression 2=x passed to evaluate which is handled by the catch segment.

A built-in error object is defined in JavaScript which provides error information about the error occurred.
The error object has two properties: name and message.
Name property: can set or return name of the error occurred.
Message property: can set or return an error message in the form of a string.
The error name property can define following values:

Error Name	Description
EvalError	It shows that there exists an error in the eval() function
RangeError	It shows that a number has assigned a value "out of its range".
ReferenceError	It shows that a variable is accessed that has not been declared
SyntaxError	It shows that a syntax error is found in the code
TypeError	A type error has occurred. Ex. trying to convert a number to uppercase
URIError	It shows that illegal characters are used in a URI (Uniform Resource Identifier) function

7.3 Exercises

1. Write an HTML code using JavaScriptto add two numbers taken as input from user and display the result of division operation on them i.e. a/b. If an error occurs, show the error. Use error handling concepts in JavaScript.

2. A teacher has created a gradeLabs function that verifies if student programming labs work. This function loops over an array of JavaScript objects that should contain a student property and runLab property. The runLab property is expected to be a function containing the student's code. The runLab function is called and the result is compared to the expected result. If the result and expected result don't match, then the lab is considered a failure.

3. There is an array of animals. The user is asked to enter the index for the animal they want to see.
i) If the user enters an index that does NOT contain an animal, the code will throw a TypeError when name is referenced on an undefined value.

ii) Update the above program to print out the index the user entered. We want this message to be printed EVERY time the code runs

4. Assume you have a function `primitiveMultiply` that in 20 percent of cases multiplies two numbers and in the other 80 percent of cases raises an exception of type `MultiplicatorUnitFailure`. Write a function that wraps this clunky function and just keeps trying until a call succeeds, after which it returns the result.
Make sure you handle only the exceptions you are trying to handle.

Session 8: JavaScript Classes

8.0 Introduction

In this session, we are going to explore JavaScript Classes. These classes were introduced in ES6. These classes are used to support object-oriented paradigm in JavaScript.

8.1 Objectives

After completing this session, one should be able to:

- understand object-oriented architecture
- define the objects and implement the methods
- make use of getters and setters

8.2 Java script classes

JavaScript Classes are templates for creating Objects.JavaScript classes have two components class expressions and class declarations.Similar to the function expressions, class expressions can also be defined in two ways: named and unnamed.The basic difference between class expression and class declaration method of defining class is that when a class is defined as class expression, it runs as soon as it is defined in contrast to class declaration. That is, use class declarations when the class is to be made available throughout the code with global scope and class expressions is used when we want to limit the scope of the class to where it is available.

Class expression syntax:

```
<script>
// unnamed method
```

```
let Bike = class
{
//let is variable type in JavaScript
  constructor(make)
  {
this.make = make;
  }
  var bmodel()
  {
        document.write("This is inside the function");
  }
};
document.write(Bike.name);
// output: "Bike"
</script>
```

Here, the class name is the variable name Bike.

```
<script>
// named method
let Bike = class Bike2
{
//let is variable type in JavaScript
  constructor(make)
{
this.make = make;
}
};
document.write(Bike.name);
// output: "Bike2"
</script>
```

Here, the class name cannot be printed using "Bike2.name" instead of "Bike.name" because the scope of class name "Bike2"is limited within the class body.

Class declaration syntax:

```
<script>
class Bike
{
  constructor(make)
   {
this.make = make;
   }
  var bmodel()
 {
        document.write("This is inside the function");
  }
};
document.write(Bike.name);
// output: "Bike"
</script>
```

The script above creates a class “Bike" with initial property model.

A JavaScript class is **not** an object rather it’s a **template** for objects.
Once we have created class it can be used by creating object of it as follows:
var object_name = new class_name(Arg1_value, Arg2_value);

When an object of a class is created, the constructor method of its class is called automatically. When a constructor method is not defined by the programmer, JavaScript will add a default empty constructor by its own. After the constructor method, class methods are created within the class. Class methods and class properties can be access as follows:

class_object.class_method(Arguments list);
class_object.class_property;

Example 1: Create a Class with name Bike having two properties: B_make and B_year and a method with name "age", that returns how many years old the bikeis.
Solution:

```
<script>
class Bike {
  constructor(B_make, B_year)
{
    this.B_make = B_make;
    this.B_year = B_year;
  }
  age()
{
    vardt = new Date();
    var x = dt.getFullYear() - this.B_year
    return x;
  }
}

varobject_bike = new Bike("Honda", 2021);
document.write ="My bike is " + object_bike.age() + " years old.";
</script>
```

Arguments can also be passed to the Class methods as follows:

object_name.class_method(Argument_list);

8.2.1 Getters and Setters Function

Classes in JavaScript have the functionality of getter and setter functions. Using gettermethodwecan get some data from a class and using setter method some fields of the class can be assigned desired values. The “get” keyword is used before the getter functions name and “set” keyword for setter methods.

Example 2:

```
<html>
<head>
<title>Getter and Setter in JavaScript</title>
</head>
<body>
<script>
class Bike
{
  constructor(make)
  {
this.make = make;
  }
  get bname()
{
    return this.make;
  }
  set bname(x)
{
this.make = x;
  }
}

var obj_bike = new Bike('TVS');
obj_bike.bname = 'Honda';
document.write(obj_bike.bname);
</script>
</body>
</html>
```

Result:

Honda

Explanation:

As we have defined getter function as “get bname() and setter function as “set bname(x)”, the value of class property make can be set directly.

The important point here to note is that even though setter and getter defined with name “bname” is method but we have not used parentheses () of function with “bname”. Another point to note is that the name of getter and setter should not be same as the class property names.

8.2.2 Class Inheritance

In JavaScript, properties or methods of a parent class can be included in the child class using inheritance. A child class can inherit all the properties and methods of a parent class using the keyword “extends”.
Inheritance extends the reusability of the code in JavaScript.

Example 3: Create a class: BikeType to define whether it is a racer bike or a normal bike, which inherits the methods of the class Bike.

Solution:

```
<html>
<body>

<script>
class Bike
{
  constructor(make)
 {
this.make = make;
 }
present()
  {
    return 'My Bike is of type ' + this.make;
  }
}

class BikeType extends Bike
{
constructor(make, bike_type)
 {
    super(make);
this.make = make;
this.bike_type = bike_type;
  }
show()
  {
    return this.present() + ', it is a ' + this.bike_type;
 }
}

var object_bike = new BikeType("Hero", "Racer");
document.write(object_bike.show());
</script>
</body>
</html>
```

Result:

My Bike is of type Hero, it is a Racer

Explanation:

The function super() used in the constructor of the child class: BikeTyperefers to the parent class: Bike.

Parent class constructor can be called when the super() method is called in the constructor method of the derived or the child class. By doing so, the parent's properties and methods can be accessed from the child class.

8.3 Exercises

1. Let us assume you need to organize a library of some electronic manuals and novels for a particular company. For each book, you need to store following information:

- The title
- The author
- The copyright date
- The ISBN
- The number of pages
- The number of times the book has been checked out.
- Whether the book has been discarded

Company wants to perform certain actions when the any book is outdated. The manual books must be exited after 5 years while a novel book should be exited if its checked out time crossed 100.

Task 1. Construct three classes that hold the information needed by company. One class should be a **Book** class and two child classes of the Book class called **Manual** and **Novel**. Each class will contain two methods. One will be a constructor. The other one will either be in charge of disposal of the book or updating the property related to the number of times a book has been checked out.

Task 2: Create an object of the Novel class and Manual class with some valid details.

Task 3: Write the method to count the duration of a manual book and checkout time for novel book so that they can be discarded based on the duration.

2. Create a object 'product' with properties: ID, Name, Description and Price. Then create function for this object called displayProductDetails(). This function when invoked should display the name, description and discounted price of the product. For the calculation of discounted price you need to create another function CalculateDisc(percentage). The discount percentage would be given by user.

Session 9: JavaScript Asynchronous programming

9.0 Introduction

Welcome to the session 8. In this session, we are going to learn the Asynchronous programming in JavaScript and explore the Async keyword.

9.1 Objectives

After completing this session, one should be able to:

- Gain familiarity with what asynchronous JavaScript is, how it differs from synchronous JavaScript, Differentials asynchronous JavaScript and Synchronous JavaScript
- Learn different use cases

9.2 Asynchronous Function

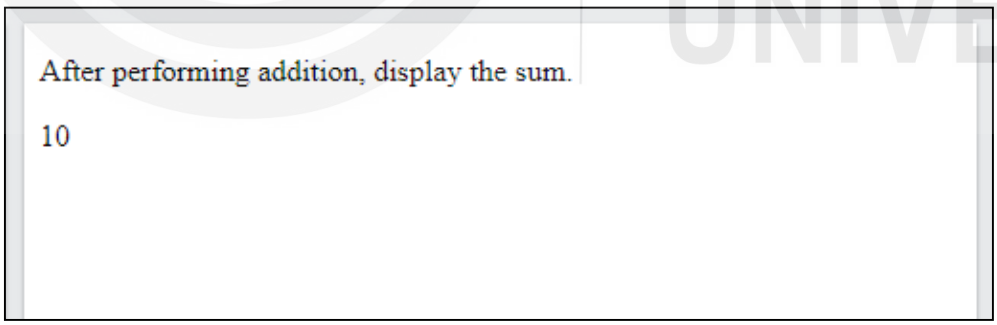
In JavaScript the functions are not executed in the sequence they are defined, rather executed in the sequence they are defined.

Sometimes functions are dependent on each other and are to be executed in a certain sequence. A function may need output of another function as input Therefore it cannot be executed until the output of another function is available. This dependency cannot be implemented by simply defining the order of function calling.
JavaScript implements Callbacks to provide more control on the sequence of function execution. A Callback is a function which is passed as an argument to another function. When a function is passed as an argument to a function, parenthesis are not used for the function passed as an argument.

Example 1:

```
<html>
<head>
<title>JavaScript Callbacks</title>
</head>
<body>
<p>After performing addition, display the sum.</p>
<script>
function show(data)
{
document.write(data);
}
function add(x, y, abc)
{
  var sum = x + y;
abc(sum);
}
add(3, 7, show);
</script>
</body>
</html>
```

Result:



Explanation:
The function show() is callbacks through the function add() by passing as an argument. First, add() function is called and executed, then within add() function abc() is linked with the show() function which is called by add() once the result of addition is available.
Asynchronous functions are implemented with Callbacks in JavaScript. In JavaScript the method setTimeout() is a typical example of asynchronous function.

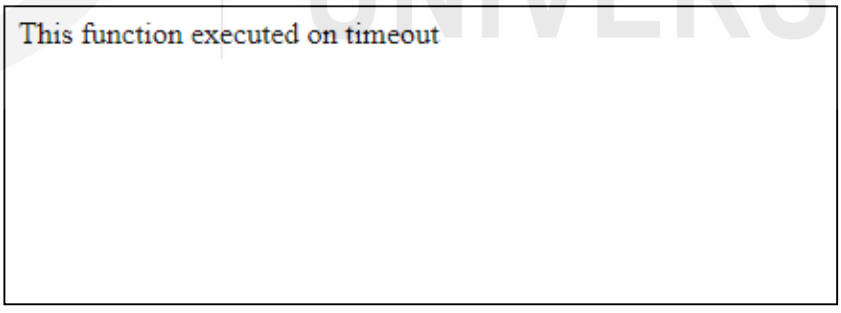
With setTimeout(myCallback, timeout_interval), the name of the callback function can be passed as an argument which will be executed after the expiry of timeout_interval.

Example 2:

```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript Callbacks with SetTimeout()</title>
</head>
<body>
<p>Please wait for 5 sec (5000 milisecond).</p>
<script>
setTimeout(abc, 5000);
function abc()
{
document.write("This function executed on timeout");
}
</script>
</body>
</html>
```

Result:

After 5 seconds:



9.3 Exercises

1. Implement a clock in HTML which shows the live time in the format HH:MM:SS. Use the JavaScript setInterval() method and a callback function by displaying the system time every second.

2. Sometimes we use external resource files in our HTML page. The content of an external file cannot be used until loaded completely. This can be implemented with JavaScript Callback. Design a web page which loads an external HTML file on the

event: onLoad. Hint: use the inbuilt class XMLHttpRequest() to use its functions: open(), onLoad() etc. to load an external HTML file.

3.The village crows own an old scalpel that they occasionally use on special missions—say, to cut through screen doors or packaging. To be able to quickly track it down, every time the scalpel is moved to another nest, an entry is added to the storage of both the nest that had it and the nest that took it, under the name "scalpel", with its new location as the value.This means that finding the scalpel is a matter of following the breadcrumb trail of storage entries, until you find a nest where that points at the nest itself.

Write an async function locateScalpel that does this, starting at the nest on which it runs. You can use the anyStorage function defined earlier to access storage in arbitrary nests. The scalpel has been going around long enough that you may assume that every nest has a "scalpel" entry in its data storage.

Session 10: JavaScript Cookies

10.0 Introduction

Cookies are small piece of user data, stored as small text files, on the user/client-side computer. Cookies are used to store user information on user's computer to use it when user visits the website next time.

Cookies are stored in the form name-value pairs as:
user_name = Devis Smith

When a user submits a request to access a web page asURL to the browser, the browser on behalf of user sends a request forthedesired web page to concerned server. Here, before sending this request, browser checks for the cookies belonging to the web page, if found are also added to the request. From this request with cookies the server gets the required data to "remember" information about users.

10.1 Objectives

In this session, we are going to learn about cookies and sessions. The main objectives of this session are as follows:

- Learn to handle cookies
- Delete, Send, and Receive cookies
- Learn session tracking

10.2 Cookies in JavaScript Creating and deleting Cookie in JavaScript

In JavaScript cookies can be created, read, modified, and deleted using document.cookie property.
A cookie is set or created using the statement as follows:
document.cookie = "username=Devis Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/; domain=site.com";

Here, this cookie stores user name in username, expires is the time after which this cookie will be deleted automatically. In a cookie it is optional to mention the expiry date and time. When expiry date and time is not mentioned in a cookie it will be deleted when the browser is closed.
The path parameter tells the browser what path the cookie belongs to. If path parameter is not set, by default it belongs to the current page.
The domain parameter is set to the root domain.
An existing cookie can be modified using the same statement as it was created. The cookie parameters will be overwritten with the newly defined values.
Reading a Cookie in JavaScript
A cookie can be read by the following statement:
var c = document.cookie;
This statement will return cookies in the form of a single string as follows: cookie1=value; cookie2=value; cookie3=value;

Deleting a Cookie in JavaScript

Deletion of a cookie in JavaScript is very easy. To delete a cookie all we need is to set its expiry parameter to a past date. The following cookie will be deleted:
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
We must specify the path parameter to ensure the deletion of right cookie.

The Cookie String

The string returned by document.cookie is not a normal string instead it is the name-value pair. A new cookie set does not overwrite the older cookies i.e. after setting the new cookie on reading document.cookie again the following result will be displayed:
cookie1 = value; cookie2 = value;
Important here to note is that: when we want to access value of a particular cookie, a JavaScript method is to be written which searches for the desired cookie value in the cookie string.

Setting a JavaScript Cookie

Let us learn setting of a cookie with an example.

- A cookie will be set which stores the name of a visitor.
- When a visitor visits the web page for the first time, he/she will be prompted to provide his/her name which will be stored in a cookie.
- This cookie will be used when the visitor visits the web page next time to show a welcome message.

Example 1: Implementing the above problem statement:
To implement this, following three javascript functions are to be created to:

1. create and set a cookie value
2. get a cookie value
3. check a cookie value

Function to create and set a cookie
The function below creates a cookie and sets values to its parameters to store the visitor's name:

```
function set_Cookie(cookie_name,cookie_value,exp_days) {
  var dt = new Date();
dt.setTime(dt.getTime() + (exp_days*24*60*60*1000));
  var age = "expires=" + dt.toGMTString();
document.cookie = cookie_name + "=" + cookie_value + ";" + age + ";path=/";
}
```

Explanation:

The function set_Cookie() creates a cookie with name of the cookie ascookie_name, value as cookie_value, and the age in days after, for which this cookie will alive.

Once we have set the cookie, now we will write a function that returns the value of the cookie

```
function get_Cookie(cookie_name)
{
  var c_name = cookie_name + "=";
  var decodedCookie = decodeURIComponent(document.cookie);
  var c1 = decodedCookie.split(';');
for(var i = 0; i< c1.length; i++)
 {
   var c2 = c1[i];
   while (c2.charAt(0) == ' ')
    {
     c2 = c2.substring(1);
}
   if (c2.indexOf(c_name) == 0)
{
     return c2.substring(c_name.length, c2.length);
   }
  }
  return "";
}
```

As of now the cookie is created/set and a function is available to get the cookie. Now whenever a visitor visits a web page the server always checks whether the cookie is available or not.

Here is the function to check whether a cookie is set:

If the cookie is not found, it will set the cookie by prompting user provide name, and set a cookies by calling the set_cookie() function to store the username cookie for 365 days:

```
function check_Cookie()
{
  var usr=get_Cookie("username");
  if (usr != "")
   {
alert("Welcome again " + usr);
} else
   {
usr = prompt("Please enter your name:","");
    if (usr != null &&usr != "")
{
set_Cookie("username", usr, 30);
```

```
    }
  }
}
```

Putting all these three function together to run and see how cookies works:

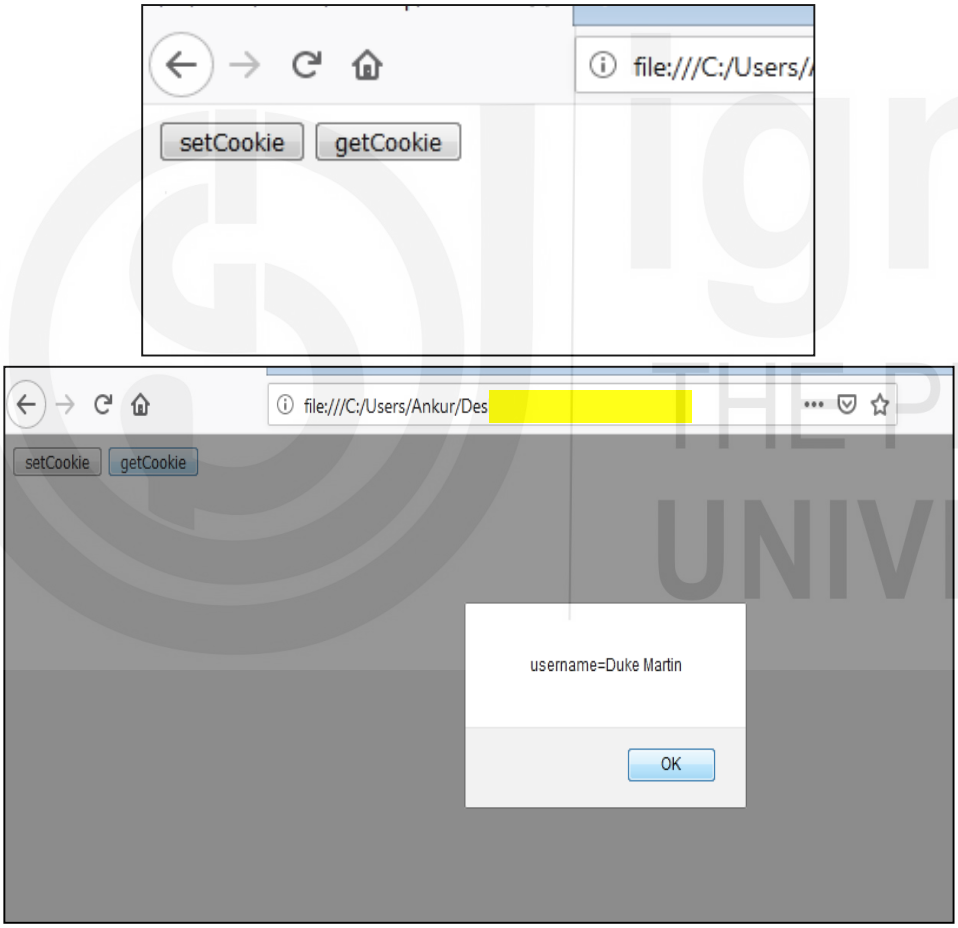
```
<!DOCTYPE html>
<html>
<head>
<script>
function set_Cookie(cookie_name,cookie_value,exp_days)
{
  var dt = new Date();
dt.setTime(dt.getTime() + (exp_days*24*60*60*1000));
  var age = "expires=" + dt.toGMTString();
document.cookie = cookie_name + "=" + cookie_value + ";" + age + ";path=/";
}
function get_Cookie(cookie_name)
{
  var c_name = cookie_name + "=";
  var decodedCookie = decodeURIComponent(document.cookie);
  var c1 = decodedCookie.split(';');
for(var i = 0; i< c1.length; i++)
{
    var c2 = c1[i];
    while (c2.charAt(0) == ' ')
{
      c2 = c2.substring(1);
    }
    if (c2.indexOf(c_name) == 0)
{
      return c2.substring(c_name.length, c2.length);
    }
  }
 return "";
}

function check_Cookie()
{
  var usr=get_Cookie("username");
  if (usr != "")
{
alert("Welcome again " + usr);
  } else
{
usr = prompt("Please enter your name:","");
     if (usr != null &&usr != "")
{
set_Cookie("username", usr, 30);
     }
  }
```

```
}
</script>
</head>
<body onload="check_Cookie()"></body>
</html>
```

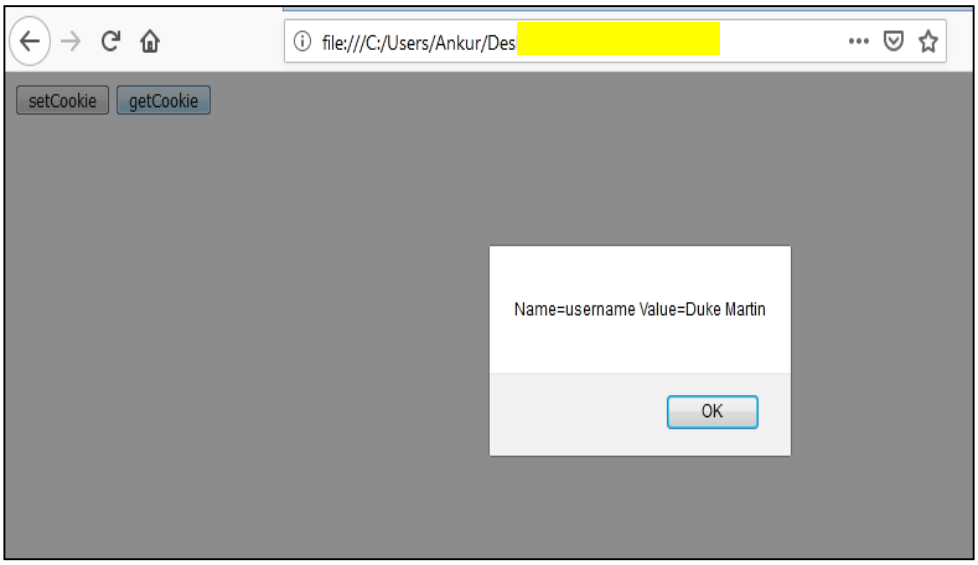
Result:
For the first time, a prompt box is shown asking to enter the name.
Once the name is provided, afterwards, whenever we reload the page, it shows a welcome message with the name entered.

10.3 Exercises

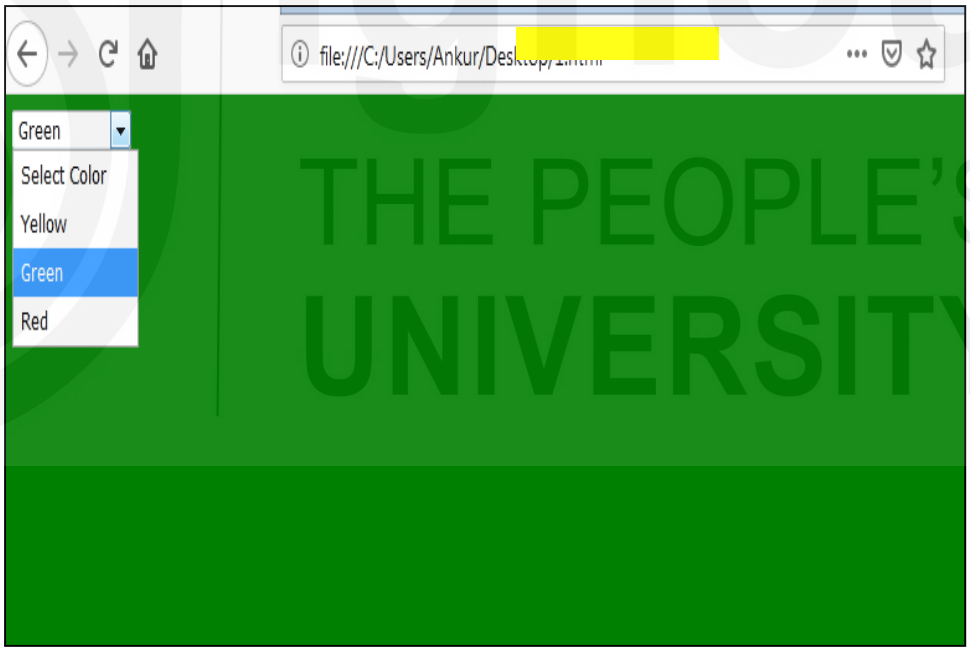
1.Create an HTML web page, as shown below. The cookie will be set on pressing setCookie button and the stored cookie value will be displayed on pressing getCookie button.



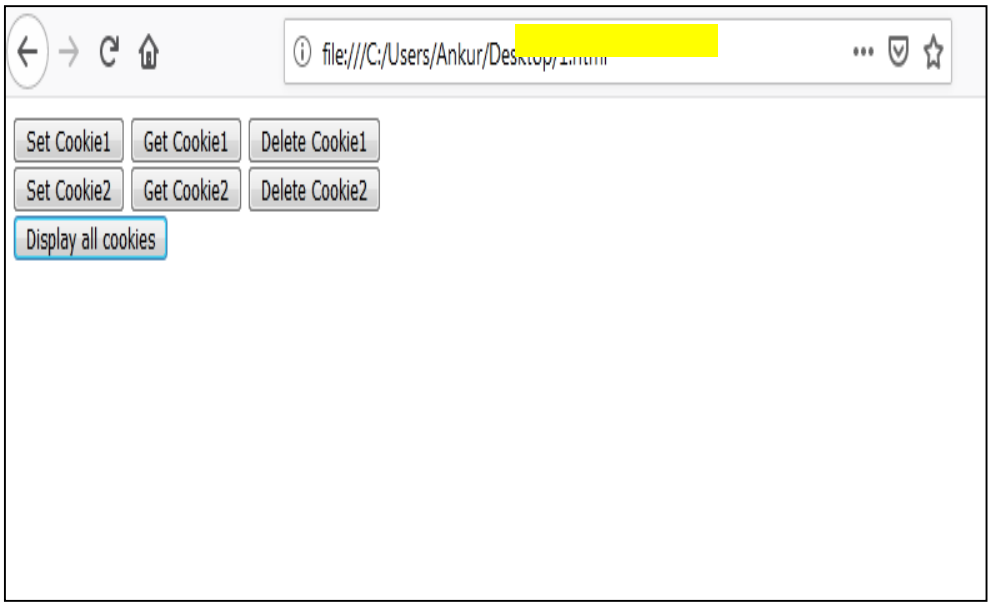
2.Considering exercise 1, modify the web page to display the cookie's name-value pair separately as shown in figure below.

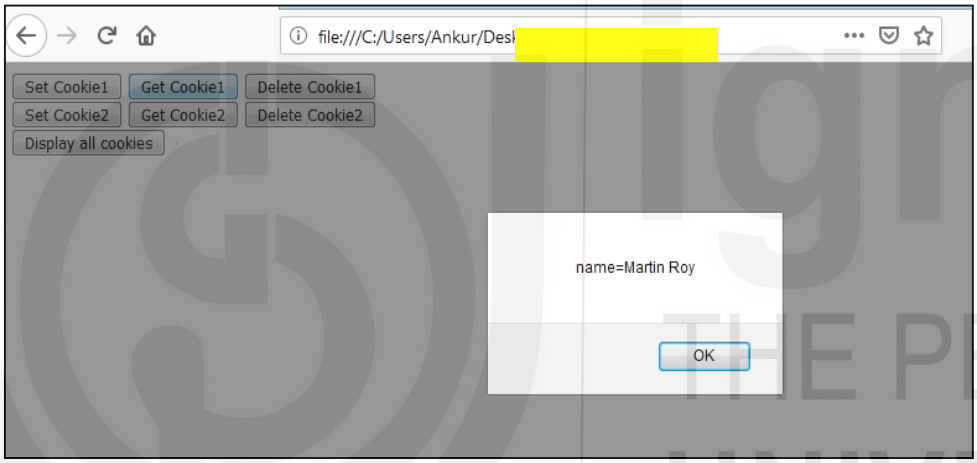


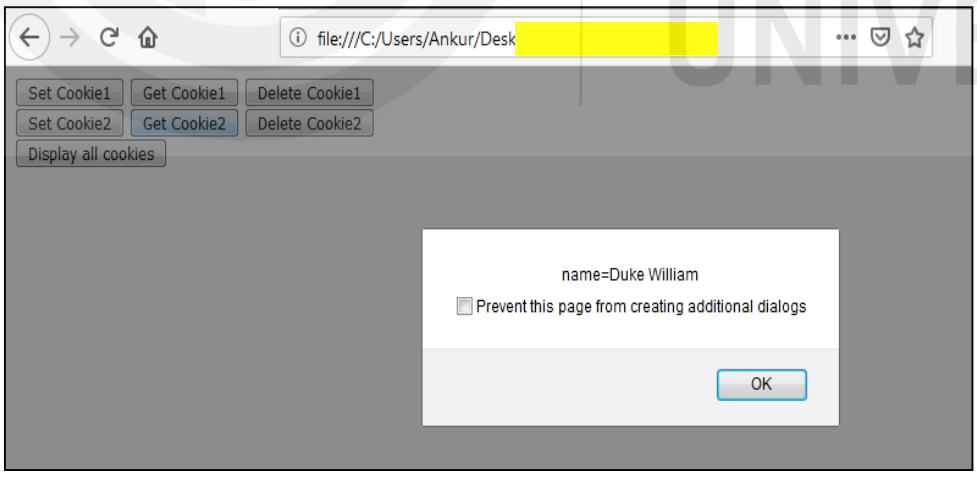
3. A list of color is provided in the form of drop down list. The background color of the web page changes as per the selection of the color from the list by the user. At the same time the chosencoloris passed to the cookie. The cookie stores the last choice of a user in a browser which will be used on reloading the web page to display the same color as background color.

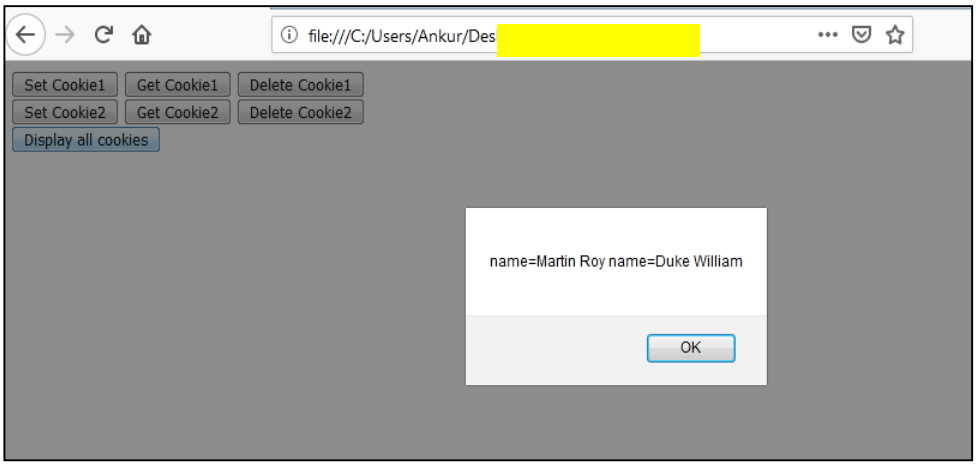


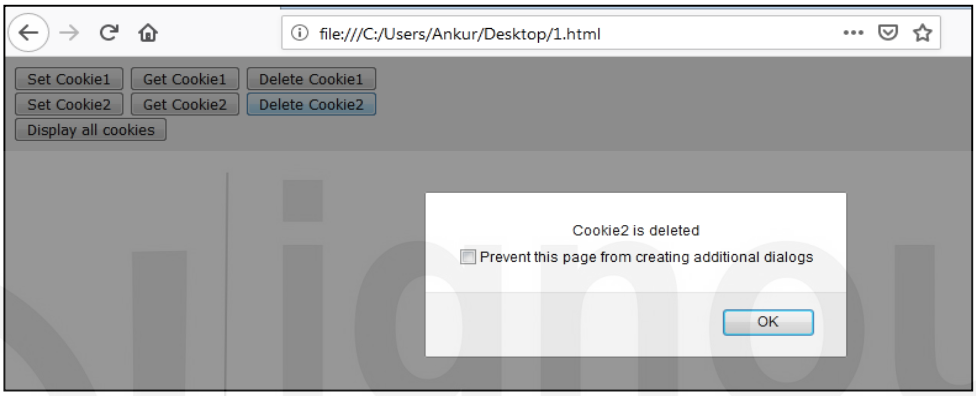
4. Create an HTML web page, as shown below. The cookie1 and cookie2 will be set on pressing Set Cookie1or Set Cookie2 button and the stored cookie value will be displayed on pressing Get Cookie1 or Get Cookie2 button respectively. On the other hand selectively cookie can be deleted by pressing Delete Cookie1 or Delete Cookie2 button. Display all cookies button will show all the stored cookies.


Set Cookie1
Get Cookie1
Delete Cookie1
Set Cookie2
Get Cookie2
Delete Cookie2
Display all cookies


Set Cookie1
Get Cookie1
Delete Cookie1
Set Cookie2
Get Cookie2
Delete Cookie2
Display all cookies
name=Martin Roy
OK


Set Cookie1
Get Cookie1
Delete Cookie1
Set Cookie2
Get Cookie2
Delete Cookie2
Display all cookies
name=Duke William
Prevent this page from creating additional dialogs
OK


file:///C:/Users/Ankur/Des
Set Cookie1
Get Cookie1
Delete Cookie1
Set Cookie2
Get Cookie2
Delete Cookie2
Display all cookies
name=Martin Roy name=Duke William
OK


file:///C:/Users/Ankur/Desktop/1.html
Set Cookie1
Get Cookie1
Delete Cookie1
Set Cookie2
Get Cookie2
Delete Cookie2
Display all cookies
Cookie2 is deleted
Prevent this page from creating additional dialogs
OK