Advance JavaScript

BccFalna.com
097994-55505

Kuldeep Chand

In this EBook, I have not only covered Simple Client Side Programming Concepts of JavaScript and Web Development but also various Advance Concepts like **Anonymous Functions, JavaScript OOPS, JSON, AJAX, Clousers,** etc…

After learning JavaScript, you can very easily move to various JavaScript Frameworks like **jQuery, Prototype**, etc… for fast and easy Client Side Development.

If you really want to be a Programmer as a Professional Developer, you will sure need to learn JavaScript because now each and everything is being developed on the basics of JavaScript.

Like HTML5, which is the latest technology for web development, have been divided in various parts for various kinds of tasks to fulfill and for fulfilling various kinds of requirements, we need to use HTML5 API like **Geo Location**, and that is available only in JavaScript API Format.

So for learning JavaScript Properly in easy to understand HINDI Language with hundreds of Example Programs, this is the only EBook for you. Just read and learn by fun.

Advance JavaScript In Hindi

■■■
Kuldeep Chand

BetaLab Computer Center
Falna

Advance JavaScript in Hindi

Lead Editors: **Kuldeep Chand**

Distributed to the book trade worldwide by BetaLab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

e-mail *bccfalna@gmail.com*

or

visit *http://www.bccfalna.com*

For information on translations, please contact BetaLab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

Phone **097994-55505**

This book is dedicated to those

who really wants to be

a

PROFESSIONAL DEVELOPER

INDEX OF CONTENTS

Table of Contents

JAVASCRIPT INTRODUCTION

JAVASCRIPT INTRODUCTION

किसी भी प्रकार की Programming Language में Program या Software Develop करते समय कई Basic Steps Follow करने होते हैं। लेकिन हमेंशा सबसे पहले हमें किसी Text Editor में अपनी Language से संबंधित Codes लिखकर कोई Program Create करना होता है। इस प्रकार के Codes को हम जिस File में लिखते हैं, उस File को **Source File** कहा जाता है, क्योंकि Program से संबंधित मूल Codes इसी Source File में होते हैं और यदि हमें हमारे Program में कोई Modification करना हो, तो हम वह Modification इसी Source File में करते हैं।

Source File केवल एक **Plain Text File** ही होती है, जिसमें हम हमारे समझने योग्य English Language में Programming Language से संबंधित Codes लिखते हैं। लेकिन Computer एक Electronic Machine मात्र है, जो हिन्दी, अंग्रेजी, Chinese जैसी उन भाषाओं को नहीं समझता जिन्हें हम Human Beings Real Life में समझते हैं, बल्कि वह केवल Binary Language या अन्य शब्दों में कहें तो Machine Language को ही समझता है। जबकि परेशानी ये है कि हम Human Beings Computer की Machine Language को आसानी से नहीं समझ सकते।

इस स्थिति में एक ऐसे Inter-Mediator की जरूरत होती है, जो हमारी English जैसी भाषा में लिखे गए Codes को Computer के समझने योग्य Machine Language में Convert कर सके और Computer द्वारा हमारे Program के आधार पर Generate होने वाले Output या Result को हमारे समझने योग्य English जैसी भाषा में Convert कर सके। इस प्रकार के Inter-mediator को Computer की भाषा में **Compiler** या **Interpreter** कहते हैं।

Compiler व Interpreter दोनों ही एक प्रकार के **Software** मात्र होते हैं, लेकिन इनका मूल काम हमारे Program के Codes को Computer के समझने योग्य मशीनी भाषा में और मशीनी भाषा में Generate होने वाले Results को हमारे समझने योग्य English जैसी भाषा में Convert करना होता है। इस प्रकार से Programming की दुनियां में मूल रूप से दो प्रकार की Programming Languages हैं:

1 पहले प्रकार की Programming Languages को Compiler Based Programming Languages कहते हैं, जिसके अन्तर्गत “C”, “C++” जैसी Languages आती हैं। इस प्रकार की Languages की मूल विशेषता ये है कि इस प्रकार की Programming Languages में हम जो Program Create करते हैं, उन्हें Compile करने पर वे Program पूरी तरह से **Machine Codes** में Convert हो जाते हैं, जिन्हें हमारा Computer Directly Run करता है।

   Compiler Based Programming Languages की मूल विशेषता ये होती है कि जब हम हमारे किसी Program को उसके Compiler द्वारा Compile कर लेते हैं, तो एक नई Executable File बनती है, जिसमें केवल Computer के समझने योग्य Machine Codes होते हैं और इस File को Run करने के लिए अब हमें हमारी Source File की जरूरत नहीं रहती।

   ये Executable File पूरी तरह से Current Computer Architecture व Operating System पर आधारित होती है। यानी यदि हम किसी Program को उस Computer पर Compile करें जिस पर **Windows** Operating System Run हो रहा हो, और Generate होने वाली Executable File को हम किसी दूसरे ऐसे Computer पर Run करने की कोशिश करें, जिस पर **Linux** Operating System हो, तो हमारा Program Linux Operating System पर Run नहीं होगा, क्योंकि Compiler Based Programming Language के Compiler द्वारा Generate होने वाली File हमेंशा अपने Operating System व Computer Architecture पर Depend होती है इसलिए पूरी तरह से Portable नहीं होती।

लेकिन चूंकि Compiler Based Programming Language में Program को Compile करने पर एक नई Executable File बन जाती है, जो कि पूरी तरह से Current Operating System व Computer Architecture पर आधारित होती है, इसलिए इस Executable File को अब उसके Source File की जरूरत नहीं रहती।

यानी एक बार किसी Program को Compile करके उसकी Executable File प्राप्त कर लेने के बाद अब यदि हम उसकी Source File को Delete भी कर दें, तब भी उसकी Executable File के आधार पर Computer हमारे Program को Run करेगा।

लेकिन यदि हमें हमारे Program में कोई Modification करना हो, तो हमें फिर से उस Program की Source File की जरूरत होगी, जिसे हमने Compile किया था और Modification करने के बाद हमें फिर से अपनी Source File को Compile करके एक नई Executable File Create करनी होगी, तभी हमारा Computer हमारे Modified Program को समझ सकेगा।

यानी Compiler Based Programming Languages को अपने Source Program की जरूरत केवल एक बार उस समय होती है, जब Source Program को Compile करके Executable File Create किया जाता है।

2 जबकि दूसरी प्रकार की Programming Languages को **Interpreter** Based Programming Language कहते हैं और इस प्रकार की Programming Languages की मुख्य विशेषता ये होती है कि Interpreter Based Programming Languages कभी भी Machine Depended Executable Files Create नहीं करते, इसलिए हमेंशा अपनी Source File पर Depend होते हैं।

यानी हालांकि Compiler व Interpreter दोनों ही हमारे Program को Machine Codes में Convert करते हैं, ताकि हमारा Computer उसे समझ सके, लेकिन Compiler Based Programming Language अपने Computer Architecture व Operating System पर Dependent एक नई Executable File Create करता है, इसलिए उसे अपनी Source File की जरूरत नहीं रहती। जबकि Interpreter Based Programming Language किसी भी तरह की नई Executable File Create नहीं करता। परिणामस्वरूप Interpreter Based Programming Language को हमेंशा अपनी Source File की जरूरत रहती है और यदि हम Source File को Delete कर दें, तो हमारा Program भी हमेंशा के लिए खत्म हो जाता है।

चूंकि Interpreter Based Programming Languages की कोई Executable Create नहीं होती, इसलिए इनमें बने हुए Programs को Run होने के लिए हमेंशा किसी न किसी Host Environment की जरूरत होती है, जिनमें Interpreter Based Languages के Programs Run होते हैं।

इसी वजह से किसी भी Interpreter Based Programming Language में यदि किसी प्रकार का परिवर्तन करना हो, तो उसकी Source File को ही Modify करना होता है और जब हम उस Modified Source File को फिर से Interpret करते हैं, हमें उसका Modification तुरन्त Reflect हो जाता है, जबकि Compiler Based Languages में हमें Source Filed में Modification करने के बाद उसे फिर से Compile करना जरूरी होता है, अन्यथा Modification का कोई Effect हमें Executable Program में दिखाई नहीं देता।

Interpreter व Compiler दोनों ही प्रकार की Programming Languages की एक विशेषता व एक कमी है। चूंकि Compiler Based Programs की हमेंशा एक Executable File बनती

है, जो कि पूरी तरह से Current Computer Architecture व Operating System पर Depend होती है, इसलिए Compiler Based Programs की Speed हमेंशा Interpreter Based Programs की तुलना में Fast होती है, क्योंकि Interpreter Based Programs की तरह इन्हें बार–बार Machine Codes में Convert नहीं होना पडता।

लेकिन Interpreter Based Program किसी भी Computer Architecture व Operating System पर बिना Recompile किए हुए ज्यों के त्यों बार–बार Run हो सकते हैं। यानी ये Portable होते हैं क्योंकि ये हमेंशा अपने **Host Environment** में Current Computer Architecture व Operating System के आधार पर बार–बार हर बार Interpret होते हैं यानी Machine Codes में Covert होते हैं और Program Run होने के बाद इनके Machine Codes समाप्त हो जाते हैं।

"C", "C++" जैसी Programming Languages, Compiler Based Programming Languages हैं, जबकि HTML, CSS, XML, JavaScript, ASP आदि Interpreter Based Markup व Client Side Scripting Languages हैं, जो हमेंशा किसी Host Environment में Run होते हैं। यानी इनका अलग से कोई Inter-Mediator Software नहीं होता बल्कि इनका Interpreter इनके Host Environment के अन्दर ही होता है।

Host Environment वह Software होता है, जिनमें विभिन्न Interpreter Based Programming Languages के Programs Run होते हैं। उदाहरण के लिए Web Browser वह Host Environment होता है, जहां HTML, XML, CSS, JavaScript आदि के Programs Run होते हैं और हमें इनका Output एक Rendered Web Page के रूप में दिखाई देता है।

जैसाकि हमने पहले भी कहा कि JavaScript एक Client Side में Run होने वाली Interpreter Based Scripting Language है और Interpreter Based होने की वजह से JavaScript का अलग से कोई Interpreter Software नहीं होता, बल्कि JavaScript Programs जिस Software में Run होते हैं, उन Software में ही JavaScript के Engine को Build किया गया होता है।

सामान्यतः Web Browsers ही JavaScript का Host Environment होते हैं, लेकिन इसका मतलब ये नहीं है कि JavaScript के Programs केवल Web Browser में ही Run हो सकते हैं। वास्तव में सच्चाई ये है कि जिस किसी भी Software में JavaScript Engine Embedded होता है, हर उस Software में JavaScript के Programs Run हो सकते हैं।

इसीलिए JavaScript केवल Web Browser में ही Use नहीं किया जाता बल्कि JavaScript Engine को कई अन्य Platforms में भी Embed किया गया है, जहां JavaScript के Programs Run हो सकते हैं।

उदाहरण के लिए Adobe Flash एक प्रकार का Animation Software है, जहां Programming Language के रूप में **ActionScript** को Use किया जाता है। ये भी एक प्रकार की JavaScript Language ही है। इसी तरह से Adobe PDF Reader में भी JavaScript Supported है।

वर्तमान समय में विभिन्न प्रकार के Web Development IDEs उपलब्ध हैं, जैसेकि Adobe DreamWeaver, Eclipse, NetBeans आदि, इनमें भी JavaScript Engine Embedded है, इसलिए ये भी JavaScript के Host Environments हैं।

यानी हम जिस Software को Use कर रहे हैं, यदि उसमें **ECMAScript Standard** आधारित कोई भी Scripting Language Supported है, तो वह एक प्रकार से JavaScript का भी **Host Environment** है।

चूंकि JavaScript का सबसे ज्यादा प्रयोग Web Pages व Web Applications को **Interactive** (User Interaction Supported) बनाने के लिए किया जाता है, इसलिए इस पुस्तक में हमारे लिए Web Browsers ही JavaScript का **Host Environment** है।

History of JavaScript

JavaScript को सबसे पहले 1995 में Netscape Navigator के Developers ने अपने Web Browser में Client Side Validation के लिए Develop किया था। Netscape तो Market से पूरी तरह से जा चुका है, लेकिन उसकी Develop की गई JavaScript Language अभी भी Market में है और आगे भी लम्बे समय तक रहने वाली है क्योंकि अब ये Language न केवल Client Side Validation के लिए उपयोगी है, बल्कि कई जगहों पर इसे Server Side Scripting Language के रूप में भी Use किया जाता है।

1992 के आसपास **Nombas** नाम की एक Company ने जिसे बाद में **Openware** नाम की Company ने खरीद दिया, एक Scripting Language Develop करना शुरू किया, जिसका नाम C-Minus-Minus रखा गया था। CMM इसलिए, क्योंकि ये लगभग पूरी तरह से C व C++ Language पर आधारित थी, लेकिन आसानी से Web Browsers में Client Side Requirements को पूरा कर सकती थी और Developers इसे आसानी से सीख सकते थे।

कुछ समय बाद Nombas ने इस Language का नाम CMM से बदलकर **ScriptEase** रख दिया। जब Netscape Navigator Market में Popular होने लगा, तो Nombas ने इसी Language का एक नया Version Develop किया जो कि Web Page में Embed हो सकता था। शुरूआत में इस Embedding Process को Espresso Pages कहा जाता था और यही World Wide Web का पहला Client Side Scripting Language बना।

Internet पर लोगों का रूझान बढने की वजह से Web Page की Size भी बढने लगी जिससे Network का Traffic भी बढने लगा क्योंकि ज्यादातर Validation व Interactivity के कामों को पूरा करने के लिए बार–बार Web Browser को Web Server से Request करनी पडती थी। इसलिए Netscape ने महसूस किया कि Web Server का Interaction कम करने के लिए एक ऐसी Scripting Language की जरूरत है जो Web Browser में ही ज्यादातर Validation के कामों को पूरा कर दे।

इस जरूरत को ध्यान में रखते हुए **Brendan Eich** जो कि Netscape Navigator को Develop कर रहे थे, ने **LiveScript** नाम की एक Client Side Scripting Language को अपने Web Browser में Include किया। उसी समय **Sun Microsystems** अपनी Programming Language "**Java**" को Develop कर रहा था और लोगों में Java बहुत Popular हो रही थी, इसलिए Netscape Navigator ने Official Release के बाद LiveScript का नाम बदल कर **JavaScript** कर दिया, ताकि लोग ये समझकर इस Language पर भी ध्यान दें कि JavaScript, Java से संबंधित ही कोई Language है ताकि JavaScript भी Popular हो जाए और हुआ भी ऐसा ही।

Netscape व उसके **JavaScript** की सफलता के साथ ही Microsoft ने भी Web Browser Technology में कदम रखा और अपनी स्वयं की JavaScript जैसी Scripting Language बनाई जिसका नाम **JScript** रखा गया।

इस समय तक वास्तव में **JavaScript, JScript** व **ScriptEase** तीन Client Side Scripting Languages हो गई थीं, जो कि किसी भी तरह से एक Unique Standard को Follow नहीं कर रही थीं।

चूंकि इन Client Side Scripting Language की Popularity बहुत कम समय में बहुत ज्यादा हो गई थी, इसलिए इस Language को भी Standardized करने की जरूरत महसूस की गई, ताकि Scripting Language Develop करने वाली सभी Companies उन Standards के आधार पर ही अपनी Scripting Language को Develop करें व Web Developers को अलग–अलग Web Browsers के लिए अलग–अलग तरह की Scripting Languages न सीखनी पडे।

इसलिए 1997 में को **European Computer Manufactures Association (ECMA)** को JavaScript 1.1 को Standardized करने का एक Proposal भेजा गया और इस Association ने **Netscape, Sun, Microsoft, Borland** व अन्य Companies, जो कि Client Side Scripting Language Develop करने में Interested थीं, के सदस्यों की एक Technical Committee गठित की ताकि JavaScript को **Cross Platform, Vendor Neutral** Scripting Language बनाने के लिए उसके Syntax व Semantics को Standardize किया जा सके।

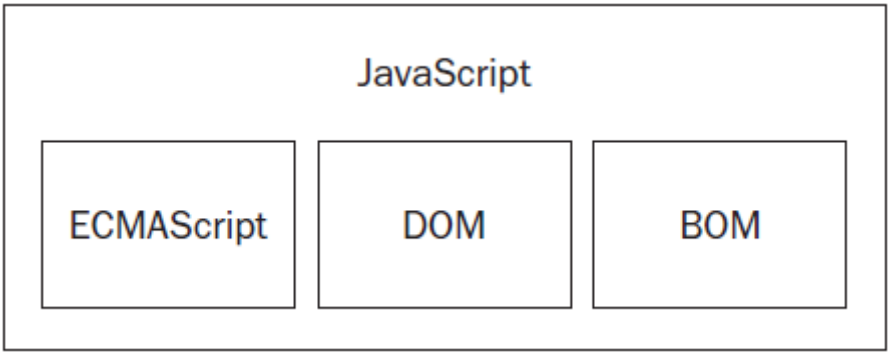
फल स्वरूप इस Committee ने अन्तिम रूप से **ECMAScript-262** नाम का एक Standard तैयार किया और JavaScript का नाम बदलकर **ECMAScript** हो गया। यानी आज की जो JavaScript है वह वास्तव में JavaScript नहीं बल्कि ECMAScript है।

आगे आने वाले कुछ सालों में *International Organization for Standardization and International Electotechnical Commission* (**ISO/IEC**) ने भी ECMAScript को एक Standard की तरह Accept कर लिया और फिर बनने वाले सभी Web Browsers में JavaScript के Implementation के लिए ECMAScript को आधार के रूप में उपयोग में लिया जाने लगा।

JavaScript Implementation

चूंकि सामान्यतः ECMAScript व JavaScript दोनों को एक ही समझा जाता है, जबकि JavaScript, ECMS-262 से कुछ ज्यादा है। एक Complete JavaScript Implementation के तीन हिस्से होते हैः

1. **The Core (ECMAScript)**
2. **The Document Object Model (DOM)**
3. **The Browser Object Model (BOM)**



ECMAScript

ECMA-262 में Define किया गया ECMAScript किसी Web Browser से Tied नहीं होता। वास्तव में इस Language में Input Output के लिए कोई Method नहीं है। ये Standard केवल एक Specification है जो विभिन्न Companies को एक आधार देता है कि उन्हें JavaScript को किस प्रकार से Implement करना चाहिए, ताकि वह विभिन्न अन्य Web Browsers के Standard के समरूप रहे। Web Browsers केवल वह **Host Environment** होते हैं, जिसमें **ECMAScript Implementation** Exist होता है।

एक **Host Environment** ECMAScript के Implementation का आधार होता है और ये Host हमेंशा कोई Web Browser ही हो, ऐसा जरूरी नहीं है। इसीलिए Adobe Company ने इस Specification के आधार पर अपनी Scripting Language Develop की है जिसका नाम **ActionScript** है और इस Scripting Language के Codes का प्रयोग करके ही Adobe Flash में **Cross-Browser Animation** Create किया जाता है। यानी **ActionScript** Scripting Language का भी आधार ECMAScript ही है।

इसीलिए यदि आप इस पुस्तक को अच्छी तरह से समझते हैं तो आप बडी ही आसानी से ActionScript Programming को भी सीख सकते हैं और Adobe Flash में ऐसे Applications Create कर सकते हैं जिनमें Animation का प्रयोग किया जाता है।

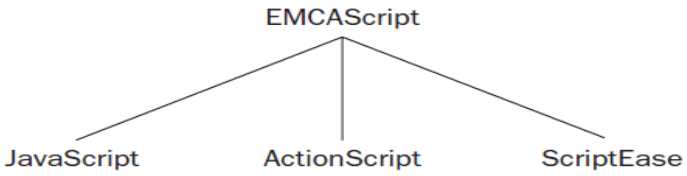
ECMAScript के Implementation के साथ ही विभिन्न Web Browsers अपने स्वयं के भी कुछ Extensions Develop करते हैं, ताकि Web Browsers को Users ज्यादा बेहतर तरीके से Web Browsing के लिए Use कर सकें।

DOM यानी Document Object Model भी एक Extension ही होता है जो अपने Core के रूप में ECMAScript के Type व Syntax को Use करता है तथा Host Environment, जो कि Web Browser भी हो सकता है और कोई अन्य Software भी, Additional Functionality Provide करता है। सामान्यतः अन्य Host Environments के रूप में **ScriptEase** व **Adobe Flash** को समझा जा सकता है।

ECMA-262 वास्तव में किसी Web Browser को Reference नहीं करता बल्कि इसका Specification किसी भी Scripting Language के निम्न Parts को Describe करता है, जिसे हम **Core JavaScript** भी कह सकते हैं:

1. **Syntax**
2. **Types**
3. **Statements**
4. **Keywords**
5. **Reserved Words**
6. **Operators**
7. **Objects**

ECMAScript केवल किसी Language के Implementation का Description मात्र है, इसलिए JavaScript वास्तव में ECMAScript को Implement करता है, ECMAScript स्वयं कोई Programming Language नहीं है बल्कि इसके आधार पर अन्य Scripting Language Develop की गई हैं, जिनमें से कुछ Most Poplar Implementations निम्नानुसार हैं:



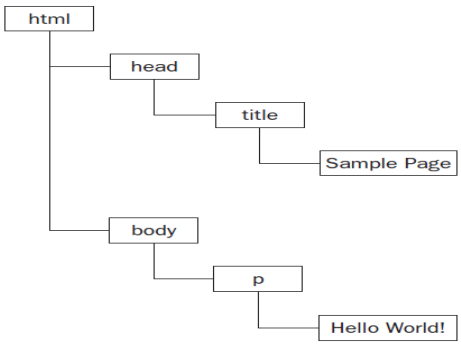
वर्तमान समय में ECMAScript का 5th Version आ चुका है, लेकिन इसे पूरी तरह से विभिन्न Web Browsers में Implement नहीं किया गया है। वर्तमान समय में Internet Explorer, FireFox, Safari, Chrome व Opera जो कि सबसे ज्यादा Use किए जाने वाले Web Browsers हैं, ने **ECMAScript3.1** Specification को पूरी तरह से Implement किया है।

Document Object Model(DOM)

DOM एक *Application Programming Interface* **(API)** है, जिसे XML के लिए Define किया गया था ताकि HTML Documents को Extend किया जा सके। DOM किसी भी Document को Memory में **Nodes की एक Hierarchy** के रूप में Model करता है। HTML या XML Document का हर Element या Tag, Attribute व Text आदि DOM के Nodes को Represent करते हैं। उदाहरण के लिए निम्न HTML Code देखिए:

```
<html>
   <head>
      <title>Sample page</title>
   </head>
   <body>
      <p>Hello World! </p>
   </body>
</html>
```

जब ये HTML Code Web Browser की Memory में Load होता है, तब निम्नानुसार Form में विभिन्न HTML Elements की एक Hierarchy बन जाती है:



किसी Document के विभिन्न Elements के Memory में इस तरह से Model होने की व्यवस्था को ही DOM या **Document Object Model** कहा जाता है, जिसमें Document के विभिन्न Elements DOM के एक **Node** को Represent करते हैं और हर Node एक **Object** की तरह व्यवहार करता है, जिसकी स्वयं की कुछ ***Properties*** व ***Behaviors*** होते हैं।

Document के विभिन्न Contents की एक Tree बनाकर DOM, किसी Web Developmer को अपने Document पर पूरी तरह से Control करने की सुविधा प्रदान करता है क्योंकि JavaScript

जैसी किसी Scripting Language का प्रयोग करके Web Developer अपने Document के किसी Node को Remove कर सकता है, DOM में नया Node Add कर सकता है, किसी अवांछित Node को Replace कर सकता है अथवा DOM API का प्रयोग करते हुए किसी Node को Modify कर सकता है।

चूंकि Web Browser में Document Render होने से पहले उस Document का DOM Tree Create होता है, जो कि उस Document का **In-Memory Model** होता है और Web Browser के Window में वही दिखाई देता है, जो DOM Tree में होता है, इसलिए DOM में किए जाने वाले परिवर्तनों का Effect तुरन्त Web Browser में Reflect होता है।

इसलिए DOM Tree किसी भी Client Side Scripting Language के लिए एक मुख्य Source होता है, जिस पर वह Scripting Language विभिन्न प्रकार के Operations Perform करके Document को ज्यादा Interactive बनाने में सक्षम हो पाता है।

चूंकि DOM को विभिन्न Companies ने अपने–अपने Web Browsers में अपनी सुविधानुसार अलग–अलग तरीकों से Develop किया था, इसलिए Web को Cross Platform यानी Platform Independent बनाए रखने के लिए व सभी Web Browsers में किसी Document को एक जैसा दिखाने के लिए फिर से एक Standard तरीके की जरूरत को महसूस किया गया।

फलस्वरूप एक नया Organization अस्तित्व में आया जिसका नाम World Wide Web Consortium (**W3C**) था। ये Organization विभिन्न प्रकार के Web Related Standards Develop करने का काम करता है। इस Organization में विभिन्न बडी कम्पनियों जैसे कि Microsoft, Google, Yahoo, AOL आदि के Members Participate करते हैं और Web किस दिशा में आगे बढेगा इस बात का निर्णय लेकर Standards Create करते हैं।

DOM के आज तक में कुल तीन Levels W3C द्वारा Define किए गए हैं। DOM Level 1 सबसे पहले October 1998 में Recommend किया गया था। इस DOM के दो हिस्से **DOM Core** व **DOM HTML** थे।

DOM Core किसी XML Based Document को Structure करने की सुविधा प्रदान करता है ताकि Developers किसी XML Document के विभिन्न हिस्सों को आसानी से Access कर सकें तथा **DOM HTML** वास्तव में DOM Core का ही एक Extension है, जिसमें HTML के साथ कुछ Specific Objects व Methods को Add करके HTML को Extend किया गया है।

DOM JavaScript नहीं है और **ECMAScript** की तरह ही इसे भी कई अन्य Programming Languages में Implement किया गया है। हालांकि Web Browsers में DOM को ECMAScript का प्रयोग करके Implement किया गया है और अब ये DOM JavaScript Language का एक सबसे बडा व सबसे महत्वपूर्ण हिस्सा है।

DOM भी ECMAScript की तरह ही केवल एक Specification है। जिस तरह से ECMAScript के आधार पर विभिन्न प्रकार की Scripting Languages को Develop किया गया है, उसी तरह से DOM के आधार पर विभिन्न प्रकार की Programming Languages में किसी Document को Access व Manipulate करने के तरीकों को Develop किया जाता है ताकि एक Programming Language में Develop किया गया Document किसी दूसरी Programming Language में भी आसानी से उपयोग में लिया जा सके।

हालांकि DOM Level 1 का मूल उद्देश्य किसी Document को Structure करना था, ताकि Developers JavaScript जैसी Client Side Scripting Language द्वारा Document के विभिन्न

हिस्सों को आसानी से Access व Manipulate कर सकें जबकि DOM Level 2 को Develop करने का मूल उद्देश्य DOM के साथ Mouse व User Interface Events, Ranges, Traversals, तथा Cascading Style Sheets को Support करवाना था, ताकि Document को न केवल बेहतर तरीके से Structure किया जा सके बल्कि उसे आसानी से Style भी किया जा सके। साथ ही उसे Interactive भी बनाया जा सके। इसलिए DOM Level 1 के Core को **XML Namespaces** को Support करने के लिए Extend किया गया। DOM Level 2 में निम्न नए Modules को Extend किया गया थाः

1 **DOM Views**
2 **DOM Events**
3 **DOM Styles**
4 **DOM Traversal and Range**

Document की Styling करने से पहले व Styling करने के बाद एक ही Document के कई Views हो जाते हैं। इन Views को Handle करने के लिए **DOM Views** का Concept Describe किया गया।

Document को User के लिए ज्यादा Interactive बनाने के लिए विभिन्न प्रकार के Events व Event Handlers को **DOM Events** के रूप में Describe किया गया।

Document की Styling को Control करने व Document के Structure से अलग रखने के लिए **DOM Styles** को Describe किया गया ताकि Document की Styling को Control, Manage व Handle करना आसान हो सके।

DOM Traversal and Range को Describe करके DOM को Access, Manipulate व Traverse करने के लिए नए Descriptions को Define किया गया।

वर्तमान समय में **DOM Level 3** को Describe किया जा रहा है, जिसमें ऐसे Methods को Support किया जा रहा है ताकि Web Browser या Host Environment के Document को Local Device पर Save किया जा सके व Local Device से Host Environment में Load किया जा सके।

एक तरह से देखा जाए, तो अब Web Technology पूरी तरह से Desktop Technology के समकक्ष आने वाली है। क्योंकि DOM Level 2 तक किसी भी Document को Local Device में Save नहीं किया जा सकता था, इसीलिए कोई भी User केवल वही Document देख सकता था, या वैसे ही किसी Document को Access कर सकता था, जैसा Developer ने उसे अधिकृत किया था।

लेकिन DOM Level 3 के पूर्ण Implementation के बाद ये बात पूरी तरह से बदल जाएगी। क्योंकि उस स्थिति में User अपनी इच्छानुसार किसी Document को Modify कर सकेगा और अपने Personal Device पर Save कर सकेगा। जिससे एक ही Document को अलग–अलग Users अपनी इच्छानुसार अलग–अलग तरीके से Access व Manipulate कर सकेंगे।

DOM Level 3 का Implementation धीरे–धीरे होने लगा है और **HTML5** DOM Level 3 व **CSS3** का ही एक Implementation है, जो कि धीरे–धीरे विभिन्न Web Browsers में Support किया जाने लगा है।

इन मूल DOMs के अलावा कुछ अन्य DOMs भी हैं, जिन्हें अलग प्रकार की जरूरतों को पूरा करने के लिए Define किया गया है। उदाहरण के लिए **SVG 1.0** व **MathML 1.0** का अपना DOM है। SVG Host Environment में Graphics Develop करने से संबंधित Standards को Handle करता है, जबकि MathML Mathematics से संबंधित Functions, Formulas आदि को Handle करता है। इसी तरह से **SMIL** के लिए Document में Multimedia Integration से संबंधित DOM को Specify किया गया है।

इनके अलावा अन्य Languages ने अपनी जरूरत के अनुसार अपना स्वयं का DOM Develop किया है। उदाहरण के लिए Mozilla ने **XML** का प्रयोग करके **XUL (XML User Interface Language)** विकसित किया है, जिसका प्रयोग Mozilla व Firefox Web Browsers के Front End को Develop करने के लिए किया गया है। लेकिन इस Language व ऐसी ही कई और Languages को W3C ने Standard के रूप में Accept नहीं किया है, जिन्हें अलग–अलग Companies ने XML के आधार पर अपनी Specific जरूरतों को पूरा करने के लिए Develop किया है।

Browser Object Model (BOM)

Web Browsers के शुरूआती दिनों में Standards बनने से पहले विभिन्न Web Browsers बनाने वाली Companies ने अपने–अपने Web Browsers में एक Specific तरह का **Browser Object Model** बनाया था, जो Web Browser को Access व Manipulate करने की सुविधा देता था। BOM का प्रयोग करके Web Developers अपने Web Page से अपने Web Browser को Access करने की क्षमता प्राप्त करते थे।

चूंकि विभिन्न Web Browser बनाने वाली Companies अपने Web Browsers को अपनी इच्छानुसार बनाती हैं, इसलिए यही एक ऐसा हिस्सा है जहां विभिन्न Companies के Web Browsers में JavaScript Implementation का कोई Standard नहीं है।

प्राथमिक रूप से BOM Web Browser **Window** व **Frames** के साथ Deal करता है लेकिन सामान्यतः Browser Specific Extensions को JavaScript में Develop किया जाता है जो कि BOM के एक हिस्से की तरह काम करता है। कुछ Extensions निम्नानुसार हैं, जो लगभग सभी Web Browsers में Common हैं हालांकि उनको अलग–अलग तरीके से Implement किया गया है:

1. नया Window Popup करने की Capability
2. Web Browser Window को *Move*, *Resize* या *Close* करने की Capability
3. **navigator** Object जो कि Web Browser से संबंधित Detailed जानकारी देता है।
4. **location** Object जो कि Web Browser में Loaded Web Page की Detained जानकारी देता है।
5. **screen** Object हो कि User के Computer के Screen Resolution की Detailed जानकारी देता है।
6. Cookies का Support भी एक Extension के रूप में Web Browser के BOM का हिस्सा होता है।
7. **XMLHttpRequest** तथा Internet Explorer का ActiveXObject भी Web Browser के BOM की Capabilities का ही एक हिस्सा है।

चूंकि BOM के लिए कोई Standard नहीं है, इसलिए सभी Web Browsers में BOM का Implementation पूरी तरह से Web Browser बनाने वाली Company की नीतियों पर आधारित होता है। फिर भी सभी Web Browsers में **window** व **navigator** Object जरूर होता है लेकिन

इन Objects की Properties व Methods को अलग–अलग Web Browsers अपनी इच्छानुसार तय करते हैं।

अलग–अलग Standards के साथ JavaScript के भी कई Versions विभिन्न Web Browsers में Implement किए गए हैं। वर्तमान समय में लगभग सभी Web Browsers JavaScript 2.0 Version को Support कर रहे हैं।

JavaScript के Version बढने के साथ उसके Features जैसे कि Keywords, Syntaxes, Features आदि भी Change होते हैं। JavaScript 2.0 वास्तव में **ECMAScript 3.1** Proposal का ही Implementation है।

चूंकि ECMAScript का 5th Version भी आ चुका है, तो जाहिर सी बात है कि जैसे–जैसे Web Browsers, ECMAScript के इस 5th Version को Support करने लगेंगे, JavaScript का एक और नया Version भी आएगा।

Web Browsers

चूंकि **JavaScript**, वास्तव में **BOM** (*Browser Object Model*), **Core ECMAScript** व **DOM** (*Document Object Model*) तीनों का Combination है, इसलिए JavaScript को समझने के लिए हमें इन तीनों को Best तरीके से समझना होगा और जैसाकि हमने पहले भी कहा है कि इस पुस्तक में Web Browser ही हमारा Host Environment है, इसलिए Web Browser को अच्छी तरह से समझे बिना हम JavaScript को उसकी पूरी ताकत के साथ उपयोग में नहीं ले सकते।

Web Browser एक ऐसा माध्यम होता है जो किसी Web Application या Web Document को Download करता है, Render करता है व Execute करता है। Web Browsers दो तरह के होते हैं। पहले प्रकार के Web Browsers केवल Text Browser होते हैं जो केवल Text Content को ही Render करते हैं। **lynx** एक ऐसा ही Web Browser है जो कि http://lynx.isc.org/ Website पर Free Available है।

जबकि दूसरे प्रकार के Web Browsers Text के अलावा विभिन्न प्रकार के Multimedia जैसे कि Sound, Audio, Video, Images, Animations आदि को भी Render करने में सक्षम होते हैं। *Google Chrome, Mozilla Firefox, Apple Safri, Internet Explorer, Opera* आदि सबसे ज्यादा Use होने वाले इस Group के Modern Web Browsers के Examples हैं।

Engines

चूंकि एक Web Browser विभिन्न प्रकार के Resources जैसेकि HTML Document, CSS Stylesheets, Multimedia Plugins, आदि को आपस में व्यवस्थित तरीके से Organize करके User के सामने Present करता है, इसलिए इन विभिन्न प्रकार के Resources को Process करने के लिए एक Web Browser में विभिन्न प्रकार के Resource Processors होते हैं, जिन्हें **Engines** कहा जाता है।

ये Engines ही किसी CSS Style को किसी HTML Element पर Apply करते हैं अथवा किसी Element पर Click करने पर Trigger होने वाले Event को Response करते हैं। यानी ये Engines ही Internally विभिन्न प्रकार के HTML, CSS, JavaScript, XML आदि Codes को Process करते हैं और हमारे सामने एक Well Organized Web Page **Render** करते हैं। Engines की कार्यप्रणाली को हम एक Car के उदाहरण द्वारा बेहतर तरीके से समझ सकते हैं।

जिस प्रकार से किसी Car की Body उसका बाहरी ढांचा मात्र होता है और उस Car की Body के Good Looking होने का मतलब ये नहीं होता कि वह Car वास्तव में Efficient व Powerful है बल्कि उस Car में जो Engine होता है, वह Engine ही उस Car की Efficiency व Power तय करता है।

ठीक इसी प्रकार से कोई Web Browser कितना अच्छा दिखाई दे रहा है अथवा Web Browser का User Interface कितना अच्छा है, इस बात से हम Web Browser की Inner Working व Power का पता नहीं लगा सकते, बल्कि Web Browser की Efficiency व Power पूरी तरह से उसमें Use किए गए **Engines** पर निर्भर करती है, जो कि किसी भी Web Page के विभिन्न Resources (HTML, XML, CSS, JavaScript Codes) को Process करके Render करने का काम करते हैं।

किसी Web Page का पूरी तरह से Process होकर Web Browser में पूरी तरह से Load होने की प्रक्रिया को Web Page का **Render** होना कहते हैं।

किसी भी Web Browser में मूल रूप से हमेंशा दो प्रकार के **Engines** होते हैं:

1. **Rendering Engine -** इसे सामान्यतः Layout Engine भी कहते हैं जो कि HTML व CSS Codes को Process करके एक Page को Screen पर व्यवस्थित तरीके से Organize करके Visible या Show करता है।
2. **JavaScript Engine -** ये Engine, JavaScript Codes को समझकर Process व Execute करता है, जिसका Effect Web Page व Web Browser के **Chrome** पर Reflect करता है।

Web Browser का वह हिस्सा जिससे User Interact करता है, Web Browser का **Chrome** कहलाता है। किसी Web Browser का Menubar, Bookmark Toolbar, Web Browser का Frame, Web Browser का Title Bar, Standard Toolbar आदि Web Browser के Chrome का हिस्सा होते हैं।

Web Browsers के ये Engines, User Inter से पूरी तरह से अलग होते हैं। यानी कोई भी User Interface Element जैसेकि Menubar, Standard Toolbar या Navigation Bar इन Engines से Directly Connected नहीं होता।

विभिन्न प्रकार के **Rendering** व **JavaScript** Engines को अलग–अलग प्रकार की Companies, Organizations या Individuals ने Develop किया है और उन्होंने ही ये तय किया है कि कोई Web Page उनके Web Browser में किस प्रकार का दिखाई देगा। इसलिए यदि हम एक ही Web Page जैसे कि http://www.google.com के Home Page को अलग–अलग Web Browsers में Open करें, तो समान Home Page भी अलग–अलग Web Browsers में Exactly समान दिखाई नहीं देता।

चूंकि Web Browsers के Engines, Web Browser के User Interface से पूरी तरह से अलग रहते हैं इसलिए Technically ऐसा सम्भव है कि एक ही **Rendering** या **JavaScript Engine** को Use करते हुए दो बिल्कुल अलग Web Browsers या Software (**Host Environment**) Create किए जा सकते हैं, जो कि एक दूसरे से बिल्कुल भिन्न दिखाई देते हों जबकि विभिन्न Web Browsers के User Interface को हम JavaScript Engines के Container की तरह समझ सकते हैं।

यानी JavaScript Engine किसी Web Browser में ठीक उसी तरह से Exist होता है, जिस तरह से किसी Car में उसका Engine Exist होता है और Web Browser का User Interface उस JavaScript Engine के Skin या Body की तरह होता है और जिस तरह से समान प्रकार का Engine Use करते हुए अलग–अलग प्रकार की Body की Car बनाई जा सकती है, उसी तरह से समान प्रकार का JavaScript व Rendering Engine Use करते हुए, अलग–अलग प्रकार के Web Browser User Interface बनाए जा सकते हैं।

वर्तमान समय में बहुत ज्यादा उपयोग में लिए जाने वाले विभिन्न Web Browsers के **Rendering Engines** को हम निम्न सारणी अनुसार समझ सकते हैं:

Rendering Engine	**Web Browser**
Trident	Microsoft Internet Explorer
Gecko	Mozilla Firefox
Presto	Opera browser
WebKit	Apple Safari (including iPhone), Google Chrome, Nokia (for mobile devices)

इसी तरह से वर्तमान समय में बहुत ज्यादा उपयोग में लिए जाने वाले विभिन्न Web Browsers के **JavaScript Engines** को हम निम्न सारणी अनुसार समझ सकते हैं:

JavaScript Engine	**Web Browser**
Jscript	Microsoft Internet Explorer
SpiderMonkey	Mozilla Firefox (up to and including version 3.5)
TraceMonkey	Mozilla Firefox (version 3.6)
JavaScriptCore	Apple Safari (up to and including version 3.2)
Nitro	Apple Safari (version 4)
V8	Google Chrome
Futhark	Opera

जैसाकि उपरोक्त सारणी द्वारा हम समझ सकते हैं कि एक ही Web Browser में हम Rendering Engine व JavaScript Engines के अलग–अलग Combinations को Use कर सकते हैं।

उदाहरण के लिए Mozilla Firefox ने अपने Firefox 3.5 Version तक ***SpiderMonkey*** नाम के JavaScript Engine को Use किया है जबकि बाद के Versions में ***TraceMonkey*** नाम के Version को Use करना शुरू कर दिया है।

विभिन्न **JavaScript Engine** Develop करने वाले Developers का मूल उद्देश्य यही है कि उनका Engine ज्यादा से ज्यादा तेज गति से JavaScript Codes को Process करे, ताकि Web Browsers Based Web Applications, जो कि JavaScript पर निर्भर हों, उसी Speed से Run हो सकें, जिस Speed से Compiler Based Executables Run होते हैं। इसलिए कई मायनों में Web Browser एक प्रकार से नया Operating System बनते जा रहे हैं।

इससे पहले कि हम आगे बढें, Web Browser की कार्यप्रणाली को भी थोडा बेहतर तरीके से समझना उपयोगी रहेगा, क्योंकि Web Browser के **Request** व **Response** Message से संबंधित कई प्रकार के Web Browser Related Objects होते हैं, जिन्हें JavaScript द्वारा Access व Manipulate करने की जरूरत पडती है।

Web Page – Request and Response

HTTP वह Protocol या Software Piece है, जो Web Browser के Addressbar में Specify किए जाने वाले Web Address के Resource को Web Browser में Load करने का काम करता है।

यानी Web Server व Web Browser के बीच जो Data Transfer होता है, उसे Handle करने का काम **HTTP** (*Hyper Text Transfer Protocol*) करता है। इस Protocol के अन्तर्गत **Web Browser** एक **Client** होता है, जो किसी Web Resource के लिए Request करता है जबकि **Web Host** वह **Server** होता है, जो Web Browser से आने वाली Request को पूरा करते हुए उसे उसका Required Web Resource Available करवाता है।

Web पर उपलब्ध किसी भी File (HTML, XML, CSS, JavaScript, Image, Sound, Video etc…) को **Web Resource** कहा जाता है।

जब Web Browser के Address Bar में कोई URL (Uniform Resource Locator) जैसे कि http://www.bccfalna.com Specify किया जाता है या किसी Web Page पर Specified किसी Hyperlink को Click किया जाता है, तो Web Browser एक **Request Message** Create करके उसे Web Server पर भेज देता है। जिसके बदले में Web Server उस Resource को Web पर Search करता है और एक **Response Message** के साथ वह Resource Web Browser को Available करवाता है। इस प्रकार से Client व Server के बीच HTTP के माध्यम से Resources का Transfer होता रहता है।

HTTP Request Message

जब Web Browser किसी URL के लिए कोई Request करता है, तो Request के रूप में एक Plain Text HTTP Request Message Create होता है, जिसे Web Server पर Send किया जाता है। इस Request Message में उस Resource की Information होती है, जिसे Web Server से प्राप्त करके Current Web Browser में Load किया जाना होता है।

उदाहरण के लिए यदि हम Web Browser के Address Bar में http://wwww.google.com Type करके Enter Key Press करते हैं, तो Web Browser निम्नानुसार HTTP Request Message Create करता है:

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:19.0) Gecko/20100101 Firefox/19.0
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Language: en-gb,en;
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;
Keep-Alive: 300
Connection: keep-alive
Cookie: PREF=ID=980395a10a8f6655:U=c31bdc3844339937:…
```

इस Request में हर Line का Code एक प्रकार का Header Message है और हर Line Web Server को Request किए गए Resource से संबंधित विभिन्न प्रकार की जरूरी जानकारियां देता है। चलिए, इस Request Message को थोडा समझने की कोशिश करते हैं।

इस Header या Request Message में सबसे पहले वह Action या Method Define होता है, जिसका प्रयोग करते हुए Request Message को Web Server पर Send किया जाना है।

HTTP में हम मूल रूप से 8 प्रकार के Actions या Methods को उपयोग में लेते हुए Web Server से किसी Resource की Request कर सकते हैं। लेकिन सामान्यतः जब हम Web Browser द्वारा किसी Resource की Request करते हैं, तब वह Request **GET** या **POST** Method द्वारा की जाती है। फिर भी विभिन्न प्रकार के Request Methods की Details निम्नानुसार हैं:

GET Method

किसी भी Webpage के हमेंशा दो हिस्से होते हैं, जिन्हें **Head Part** व **Body Part** के नाम से जाना जाता है। Head Part में हमेंशा Meta Information होते हैं, जो कि Basically Search Engines व Web Browser के Chrome से संबंधित होते हैं, जबकि Body Part में Web Page के Actual Contents होते हैं।

इस Method को Use करने पर Specified URL पर स्थित Page के Content का HTML Format Body Return होता है।

POST Method

इस Method का प्रयोग सामान्यतः HTML Form में किया जाता है, जिसमें किसी Data को फिर से Process होने के लिए Web Server पर भेजना होता है।

HEAD Method

ये Method **GET** Method के समान ही काम करता है। दोनों में मूल अन्तर केवल इतना है कि GET Method Use करने पर Requested HTML Page की Body भी Return होती है, जबकि HEAD Method Use करने पर Requested HTML Page का केवल Head Part ही Return होता है, जिसमें Web Browser से संबंधित Metadata Information होती है।

इस Method का प्रयोग हम तब करते हैं, जब हमें केवल Response के साथ आने वाले Metadata को ही प्राप्त करना होता है अथवा इस बात का पता लगाना होता है कि Specified URL Actually Exist है या नहीं।

PUT Method

इस Method को Use करके हम किसी Web Server पर स्थित किसी Resource को Update कर सकते हैं। ये सामान्यतः POST Method के समान काम करता है, लेकिन ये केवल उसी स्थिति में Server के किसी Resource को Modify कर सकता है, जबकि Server इस बात की Permission देता हो।

DELETE Method

इस Method को Use करके हम किसी Web Server पर स्थित किसी Resource को Delete कर सकते हैं, लेकिन ये केवल उसी स्थिति में Server के किसी Resource को Delete कर सकता है, जबकि Server इस बात की Permission देता हो।

TRACE Method

ये Method, Web Server पर Sender द्वारा आने वाली Request को फिर से उसी Sender को भेज देता है। इस Method का प्रयोग करके हम इस बात का पता लगा सकते हैं कि Request के दौरान कौन–कौन से Servers, Services आदि Client व Server के बीच बनने वाले Connection के Chain में Involve हो रहे हैं।

OPTIONS Method

इस Method को Use करके हम किसी Particular URL पर Available विभिन्न Actions या Methods का पता लगा सकते हैं, जिसे वह URL Support करता है। यदि हम URL को एक Wildcard Character (*) की तरह Specify करते हैं, तो Web Server हमें उस Resource पर Perform हो सकने वाले सभी Actions (Methods) की List Response के रूप में Return करता है।

अब यदि हम हमारे उपरोक्त उदाहरण के Request Message की पहली Line को देखें, जो कि निम्नानुसार है:

GET / HTTP/1.1

तो हम समझ सकते हैं कि ये Line Web Server को इस बात की Information देगा कि Web Browser को Request किए जाने वाले Page का HTML Markup यानी Body Part चाहिए। जबकि Line में दिखाई देने वाला "/" Character इस बात को Specify कर रहा है कि Web Browser को Specified Domain के Root Page या Home Page की जरूरत है और इस जरूरत को HTTP/1.1 यानी HTTP Protocol के 1.1 Version के Rules को Use करते हुए पूरा करना है।

Host: www.google.com

Request Message की ये Line Web Server को बताता है कि Web Browser जिस Host से Resource या Home Page की Request कर रहा है, वह Host www.google.com है।

User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:19.0) Gecko/20100101 Firefox/19.0

Request Message की ये Line उस Web Browser की Information दे रहा है, जिससे Request Perform की गई है। सामान्यतः User-Agent Header में Current Web Browser की Information होती है।

ये Header Line इस बात को Specify कर रहा है कि Perform होने वाली Request **Mozilla/5.0** Web Browser से Perform की गई है, जो कि **Windows NT 6.2** Operating System यानी Windows-8 पर Installed है, जबकि **WOW64** इस बात को Specify कर रहा है कि Installed Windows Operating System **64bit** का है।

Web Browser के Operating System की Information के बाद **Gecko/20100101** इस बात को Specify कर रहा है कि Current Web Browser Gecko Based Web Browser है, जिसका नाम Firefox है और Version 19.0 है।

Accept: text/html,application/xhtml+xml,application/xml;

Request Message की ये Header Line इस बात को Specify कर रहा है कि Current Web Browser किस–किस प्रकार के Document को Support करता है। यानी Current Web Browser किन File Types या **MIME Types** (Multipurpose Internet Mail Extensions) को हमारे समझने योग्य Format में Convert करके Render कर सकता है।

उपरोक्त Header इस बात को Specify कर रहा है कि Current Web Browser HTML, XHTML व XML Types के Documents को इस तरह से Render करने में सक्षम है, जिस तरह से वह हमें यानी Human Beings को समझ में आता है।

Accept-Language: en-gb,en;

ये Header Line Web Server को इस बात की जानकारी देता है कि Current Web Browser किस Locale व Languages के लिए Currently Configured है। ये Line इस बात को Specify कर रहा है कि Current Web Browser English Language व UK Locale के लिए Configured है क्योंकि "**gb**" UK Locale को Represent करता है।

जबकि Backup के लिए केवल "**en**" Specified है, जो कि इस बात की Information है कि बिना किसी Geographical Locale की स्थिति में Default रूप से ये Web Browser English Language को Support करता है। Web Server इस Information को उस स्थिति में Ignore कर देता है, जब Web Browser द्वारा Requested Page केवल एक ही Language Version में Available हो।

Accept-Encoding: gzip,deflate

ये Header Web Server को इस बात की जानकारी दे रहा है कि Current Web Browser किस प्रकार के Encoding के Content को Accept कर रहा है। यदि Web Browser द्वारा Specified Encoding Type को Web Server Support न करता हो, तो Web Server स्वयं Standard Encoding Use कर लेता है। लेकिन यदि Web Server, Web Browser Accepted Encoding को समझता है, तो वह Response Content को उसी Format में Compress करके Send करता है, ताकि Response Content की Size कम हो जाए व Content ज्यादा तेजी से Web Server तक पहुंच सके।

उपरोक्त Header में **gzip** व **deflate** Compression Format को Current Web Browser Support करता है। जिसका मतलब ये है कि यदि Web Server इन Compression Formats को Support करता है, तो वह Requested Resource यानी HTML, CSS, JavaScript आदि Files को इनमें से किसी Format में Compress करके Web Browser को Send कर सकता है, जिससे Documents के Web Browser में Download होने की Speed काफी तेज हो जाती है।

Accept-Charset: ISO-8859-1,utf-8;

ये Header Message Web Server को इस बात की जानकारी देता है कि Current Web Browser ISO-8859-1 व utf-8 Character Sets की Encoding को Accept करता है, जिनमें लगभग दुनियां कि किसी भी भाषा के अक्षरों व Symbols के लिए Specific Code समाहित हैं। यानी Web Browser दुनियां की किसी भी भाषा के अक्षरों व Symbols को Web Browser में Render करने में सक्षम है।

Keep-Alive: 300

ये Header Message Web Server को इस बात की जानकारी देता है कि Web Browser व Web Server के बीच Data Transfer के लिए जो Connection बनेगा, वह Connection 300 Seconds यानी 5 Minutes तक Available रहेगा। परिणामस्वरूप यदि 300 Seconds की अवधि में Current Web Browser से फिर से उसी Web Server पर कोई Request Send की जाती है, तो Web Server पर फिर से नया Connection Open करने की जरूरत नहीं रहेगी। लेकिन यदि Request 300 Seconds के बाद की जाती है, तो Client व Server के बीच का Connection Lost हो जाएगा और Web Browser व Server के बीच फिर से एक नया Connection Open होगा।

Connection: keep-alive

ये Connection Header Information इस बात को Specify करता है कि Client व Server के बीच किस प्रकार का Connection बनेगा। सामान्यतः **HTTP/1.1** Protocol के साथ **keep-alive** सबसे Common रूप से Use होने वाला Connection Type होता है।

Cookie: PREF=ID=980395a10a8f6655:U=c31bdc3844339937:…

HTTP Cookie Client Computer पर Locally Store होने वाले Text Based Data होते हैं, जिनका प्रयोग सामान्यतः Web Server द्वारा किसी Client Computer को Uniquely Identify करने अथवा Session Create करने के लिए किया जाता है। Cookies को सामान्यतः उस Web Site द्वारा Client Computer पर Place किया जाता है, जिसकी Request Web Browser करता है, ताकि Web Site अपने हर Viewer को Personalized Information दे सके।

चूंकि HTTP एक **Connectionless Protocol** है, यानी Client द्वारा एक बार Request करने और Server द्वारा उस Request को पूरा कर दिए जाने के बाद Client व Server दोनों एक दूसरे से पूरी तरह से अनजान हो जाते हैं, इसलिए उस स्थिति में Cookies का प्रयोग करके इस बात की जानकारी को Maintain किया जाता है कि किस Client ने Web Server पर किस Resource की Request की है। यानी एक बार एक Request पूरी हो जाने के बाद Cookies ही Client व Server के बीच एक दूसरे को फिर से Identify करने का माध्यम होते हैं।

HTTP Response Message

जब एक बार किसी Web Browser से किसी Resource की Request की जाती है, तो उस Request को पूरा करने के लिए पिछले Session में Discuss किए अनुसार एक HTTP Request Message बनता है जिसमें विभिन्न प्रकार की Header Information होती हैं। ये Request

Message ही Web Server पर पहुंचता है, जिसे Web Server Receive करके इसके Data को Process करता है और बदले में एक HTTP Response Message Create करता है।

यदि हम उपरोक्त उदाहरण को ही आगे बढाऐं, तो Request के बदले में Create होने वाला Response Message निम्नानुसार हो सकता है:

```
HTTP/1.x 200 OK
Cache-Control: private, max-age=0
Date: Fri, 29 Mar 2013 12:42:14 GMT
Expires: -1
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 2520

<html><head>
... rest of HTML for Google's home page ...
</body></html>
```

चलिए, जिस तरह से हमने Request Message के विभिन्न Headers को One by One समझा, उसी तरह से हम इस Response Message के भी सभी Headers को One by One समझने की कोशिश करते हैं।

HTTP/1.x 200 OK

Response Message का ये पहला Header Request करने वाले Web Browser को इस बात की जानकारी देता है कि Request को पूरा करने के लिए किस HTTP Protocol का प्रयोग हुआ है।

Version की Information के बाद एक Short Description Information Web Browser को Return होता है, जो कि Web Browser को इस बात की जानकारी देता है कि उसकी Request की Processing का क्या परिणाम रहा।

उपरोक्त Header में ये Status Code **200** व Description **OK** है, जो इस बात को Indicate कर रहा है, कि Web Browser द्वारा Perform की गई Request सही तरीके से Process हो गई है और Web Server ने Request किया गया Resource Return कर दिया है।

Web Server द्वारा अलग–अलग परिस्थितियों में अलग–अलग प्रकार के **Status Code** व **Description** Return होते हैं, जिनके बारे में हम आगे Detail से जानेंगे।

Cache-Control: private, max-age=0

हर Web Browser में एक Local File Cache होता है, जो Recently Requested Files को Store करके रखता है, ताकि यदि फिर से उन्हीं Files की Request हो, तो Web Browser उन Files को फिर से Download न करके अपने Cache से ही उन्हें प्राप्त कर ले, ताकि Request ज्यादा तेजी से Perform हो जाए और Response ज्यादा तेजी से प्राप्त हो जाए।

Cache-Control Header Web Browser को Currently Requested Resource से संबंधित कुछ Parameters देता है, जो Web Browser को इस बात की जानकारी देता है कि Web

Browser कितने समय तक Web Server से आने वाले Resources को Cache करके रख सकता है।

हमारे उदाहरण में **Cache-Control** Header के साथ "**private**" Specified है, जो Web Browser के लिए इस बात का Indication है कि Current Resources केवल Current User के लिए ही है और यदि Current User किसी LAN यानी Local Area Network पर है, तो LAN पर उपलब्ध अन्य Users के लिए वह Resource Available नहीं होगा।

परिणामस्वरूप यदि LAN पर काम करने वाले अन्य Users उसी Web Page को Open करेंगे, तो उनके लिए हर Resource फिर से Download होगा और उनके लिए अलग से Cache होगा, फिर भले ही सभी Users समान Web Browser Program को ही Share क्यों न कर रहे हों।

साथ ही "**max-age**" Property Web Browser को इस बात की जानकारी देता है कि Web Browser कितने Seconds तक Requested Page को Cache करके रख सकता है। हमारे उदाहरण में "**max-age**" का मान 0 है, जो इस बात का Indication है कि Web Browser Google के Homepage को 0 Seconds के लिए ही Cache करेगा या दूसरे शब्दों में कहें तो Cache नहीं करेगा।

यदि हम किसी Web Page को Web Browser में Cache करवाना नहीं चाहते, तो हम **Cache-Control** Header में Value के रूप में "**no-cache**" मान Specify करके ऐसा कर सकते हैं।

Date: Fri, 29 Mar 2013 12:42:14 GMT

Response Message के इस Header में उस समय की Information होती है, जब HTTP Response Message, Web Server से Sent किया गया होता है।

Expires: -1

ये Header Web Browser को इस बात की Information देता है कि Requested Page किस Date-Time पर Old हो जाएगा और यदि फिर से उस Page को Load किया जाएगा, तो वह Web Browser के Cache से Load न होकर फिर से Download होगा। यानी Requested Web Page कब Expire हो जाएगा, इस बात की जानकारी इस Header में होती है।

Current Header में हमें -1 दिखाई दे रहा है, जो इस बात का Indication है कि Requested Page पहले से ही काफी पुराना हो चुका है और अगली बार यदि इस Page का Request किया जाएगा, तो Web Browser इस Page को Cache से Load करने के बजाय Web Server से Page की एक नई Copy Download करेगा।

Content-Type: text/html; charset=UTF-8

ये Header Requested Document के MIME Type को Represent करता है, जो Current Example में इस बात को Specify कर रहा है कि Currently Requested Page एक Plain Text HTML Page है जो कि एक **UTF-8** Character Set Supported Page है।

Content-Encoding: gzip

ये Header Web Browser को ये बात Specify कर रहा है कि Web Server से Return होने वाला Document **gzip** Format में Compressed है, जिसे Web Browser Extract करके Render करेगा।

Web Server उसी स्थिति में किसी Resource को Compress करता है, जबकि Web Browser उस Compression Mode को Accept करता है। चूंकि Request Message में हमने देखा था कि Web Browser **gzip** व **deflate** Compression Mode को Support करता है, इसलिए Web Server ने Requested Document **gzip** Format में Return किया है।

यदि Request Message में **Accept-Encoding** Header में Specified Compression Mode को Web Server Support नहीं करता, तो Web Server Return होने वाले Resource को Compress करके Send नहीं करता बल्कि Normal Content की तरह ही Send करता, जिससे Web Page या Resource के Web Browser में Load होने की Speed कुछ धीमी हो जाती।

Server: gws

ये Header Web Browser को इस बात की जानकारी दे रहा है कि Return होने वाला Resource किस Web Server से Return हो रहा है। उपरोक्त उदाहरण में ये **gws** है जो कि Google का स्वयं का Web Server है।

Content-Length: 2520

ये Header Return होने वाले Resource की Length या Size को Bytes के रूप में Return करता है। यानी ये Header Web Brower को इस बात की जानकारी देता है कि Web Server से आने वाले Resource की Size किया है।

इन सभी Response Headers के बाद अन्त में Requested Resource का Actual Content होता है, जो कि Current Example में एक HTML Document है। यदि Requested Resource कोई Image File होता, तो यहां पर Text Representation के रूप में वह Image Download होना शुरू होता, जो Download होने के बाद Current Web Browser व Operating System द्वारा Image Format में Convert होकर Web Browser में Display होता है।

अब आप समझ सकते हैं कि जब आप Web Browser के Address Bar में कोई URL Specify करके Keyboard पर Enter Key Press करते हैं या Web Page पर दिखाई देने वाले किसी Hyperlink को Click करते हैं, तब उपरोक्तानुसार दोनों Request व Response Messages में कितनी सारी Header Information Send व Receive होती है और इसी Header Information के आधार पर Web Browser व Web Server के बीच Information का Transfer होता है।

HTTP Status Codes

सभी HTTP Status Codes 3-Digit Numbers होते हैं, जो किसी Request के बदले में Web Server द्वारा किए गए Response को Represent करते हैं और इस बात को Indicate करते हैं कि Request सही तरीके से Fulfill हुई या Client द्वारा किसी अन्य Action की जरूरत है, ताकि Requested Data को उपयुक्त तरीके से Successfully Locate किया जा सके। यहां हम कुछ Common Status Codes के बारे में Discuss कर रहे हैं, जबकि HTTP के सभी Status

Codes की जानकारी *http://en.wikipedia.org/wiki/List_of_HTTP_status_codes* पर प्राप्त कर सकते हैं।

200+ (Success)

200 से 299 के बीच के सभी Status Codes इस बात को Indicate करते हैं कि Web Server द्वारा Request Message को ठीक से Receive करके Process कर लिया गया है तथा Web Browser को किसी प्रकार का Content Return कर दिया गया है। इस Range में सामान्य रूप से उपयोग में आने वाले Status Codes का Description निम्नानुसार हैः

200 OK

ये Status Code इस बात को Represent करता है कि Request Successful रहा तथा Response Message में Requested Data Exist है।

201 Created

ये Status Code इस बात को Represent करता है कि POST या PUT Method के अनुसार Server पर नया Resource Create हो गया है।

204 No Content

ये Status Code इस बात को Represent करता है कि Request Successful रहा लेकिन Requested URL पर Response Message में Return करने के लिए कोई Data Exist नहीं है।

206 Partial Content

ये Status Code इस बात को Represent करता है कि Request Successful रहा लेकिन Requested Data ज्यादा होने के कारण अथवा Network Failure या User Cancellation के कारण Content Web Browser में पूरी तरह से Download नहीं हो सका और पूरा Data प्राप्त करने के लिए Web Browser को फिर से Request करना होगा अथवा Download को Resume करना होगा।

300+ (Redirection)

300 से 399 के बीच के सभी Status Codes इस बात को Indicate करते हैं कि Web Server द्वारा Request Message को ठीक से पूरा करने के लिए Client को एक Extra Step लेना होगा व किसी अन्य URL पर Redirect करना होगा अन्यथा कोई Content Return नहीं होगा। इस Range में सामान्य रूप से उपयोग में आने वाले Status Codes का Description निम्नानुसार हैः

301 Moved Permanently

ये Status Code इस बात को Represent करता है कि Requested URL किसी अन्य Location पर Permanently Move कर दिया गया है। यानी Requested Resource Current URL पर Available नहीं है बल्कि किसी अन्य URL Location पर उपलब्ध है।

302 Found

ये Status Code इस बात को Represent करता है कि Requested URL किसी अन्य Location पर Temporarily Move कर दिया गया है। यानी Requested Resource Current URL पर Currently Available नहीं है लेकिन भविष्य में फिर से इस URL पर वह Resource Available हो सकता है।

304 Not Modified

ये Status Code इस बात को Represent करता है कि Requested URL को पहले भी Request किया गया है और तब से अब तक उसमें किसी तरह का कोई Modification नहीं किया गया है। इसलिए Client को Web Browser के Cache में Stored Resource को Locally ऩेम करना चाहिए।

400+ (Client Error)

400 से 499 के बीच के सभी Status Codes इस बात को Indicate करते हैं कि Web Browser द्वारा भेजे गए Request Message में किसी तरह की Error थी इसलिए Web Server Requested Resource को Return नहीं कर पाया। ये Codes इस बात को Indicate करते हैं कि वांछित Response प्राप्त न हो पाने का Fault *Client Side* में है न कि Server Side में। इस Range में सामान्य रूप से उपयोग में आने वाले Status Codes का Description निम्नानुसार हैः

400 Bad Request

ये Status Code इस बात को Represent करता है कि Request Message उपयुक्त Format में न होने की वजह से Web Server उसे ठीक से समझ ही नहीं पाया।

401 Unauthorized

ये Status Code इस बात को Represent करता है कि Request Message के साथ Web Server को उपयुक्त **Username** व **Password** भी चाहिए, क्योंकि Requested Content एक Restricted Content है।

403 Forbidden

ये Status Code इस बात को Represent करता है कि Web Server ने Client की Request को Refuse कर दिया है। ऐसा तब हो सकता है, जब Web Server पर उस **IP Address** को Block या Blacklisted कर दिया गया हो, जिस पर Installed Web Browser से Request Perform किया गया है।

404 Not Found

ये Status Code इस बात को Represent करता है कि Requested URL Current Location पर Available नहीं है, लेकिन भविष्य में इस Location पर कोई Content हो सकता है, इसलिए Web Browser भविष्य में फिर से इस URL की Request कर सकता है।

405 Method Not Allowed

ये Status Code इस बात को Represent करता है कि Request Message जिस तरह का Interaction Current URL के साथ करना चाहता है, उस प्रकार का Interaction Specified URL पर Allowed नहीं है। ये Code तब Generate हो सकता है, जब User Google के Homepage को DELETE Method द्वारा Delete करने की कोशिश करे।

410 Gone

ये Status Code "**404 Not Found**" Status Code की तरह ही काम करता है। अन्तर केवल इतना है कि ये Status Code इस बात को Represent करता है कि Specified URL को फिर से Try नहीं करना चाहिए।

सामान्यतः ये Status Code, Search Engine Spider के लिए उपयोगी होती है, क्योंकि यदि Search Engine Spiders को ये Status Code प्राप्त होता है, तो Search Engine Spiders

Specified URL को अपने Index से हमेंशा के लिए Remove कर सकते हैं, ताकि वे फिर से इस URL पर न आऐं।

413 Request Entity Too Large

ये Status Code इस बात को Represent करता है कि Request Message इतना बडा है कि Web Server उसे Process नहीं कर सकता। ये Status Code तब Return हो सकता है, जब कोई HTML Form अपनी Limit से ज्यादा Data Web Server पर Process होने के लिए Submit कर देता है।

414 Request URL Too Long

ये Status Code इस बात को Represent करता है कि Request Message में Specified URL Acceptable Size से ज्यादा बडा है।

500+ (Server Error)

500 से 599 के बीच के सभी Status Codes इस बात को Indicate करते हैं कि Web Browser द्वारा भेजा गया Request Message पूरी तरह से ठीक था लेकिन Server की किसी समस्या के कारण Request पूरी नहीं हो सकी। ये Codes इस बात को Indicate करते हैं कि वांछित Response प्राप्त न हो पाने का Fault *Server Side* में है न कि Client Side में। इस Range में सामान्य रूप से उपयोग में आने वाले Status Codes का Description निम्नानुसार हैः

500 Internal Server Error

ये Status Code सामान्यतः तब Return होता है, जब Server Side में कोई Script Run हो रही होती है और उस Script में किसी तरह का Error Trigger हो जाता है।

501 Not Implemented

ये Status Code सामान्यतः तब Return होता है, जब Server HTTP Method को ठीक से समझ नहीं पाता या Support नहीं करता।

502 Bad Gateway

ये Status Code सामान्यतः **Proxy Server** द्वारा तब Return होता है, जब Client व Server के बीच Data Transfer ठीक से नहीं हो पाता। जिसका मतलब ये है कि Web Server या तो Request Message को ठीक से समझ नहीं पाता अथवा इस बात के लिए Sure नहीं होता कि Web Server द्वारा Return होने वाला Response Data Web Client तक पहुंचेगा या नहीं

503 Service Unavailable

ये Status Code सामान्यतः तब Return होता है, जब या तो Web Server **Overload** हो जाता है या फिर **Scheduled Maintenance Period** में होता है।

504 Gateway Timeout

ये Status Code सामान्यतः तब Return होता है, जब Web Client व Web Server के बीच स्थित Proxy Server, Client व Destination के बीच Messages को ठीक तरह से Forward नहीं कर पा रहा होता है।

Status Codes व **Request/Response** से सम्बंधित उपरोक्त Discussion में बताए गए Concepts हमारे लिए तब उपयोगी होते हैं, जब हम AJAX Technology को Use करते हैं और

AJAX वर्तमान समय में एक बहुत ही उपयोगी तकनीक है, जिसका प्रयोग करके हम Powerful ***Dynamic Websites*** व ***Web Applications*** Create कर सकते हैं।

Development Environment Setup

आप कोई भी नई Programming Language सीखना चाहते हों, सीखने का सबसे बेहतर तरीका यही है कि उस Language से संबंधित Basics व Fundamentals को छोटे–छोटे Programs बनाते हुए सीखा जाए और Program बनाने के लिए हमें हमेंशा किसी न किसी Text Editor या IDE की जरूरत होती है।

JavaScript भी एक प्रकार की Programming Language या ज्यादा बेहतर शब्दों में कहें, तो एक प्रकार की **Client Side Scripting Language** है, इसलिए इससे पहले कि हम इस Language को समझें, हमें JavaScript Programs को Develop करने से सम्बंधित Basic Environment Setup करने की जरूरत है, ताकि पुस्तक में आगे आने वाले Program Codes की Working को आसानी से समझा जा सके।

अन्य सभी Programming, Scripting व Markup Languages की तरह ही JavaScript Programs को भी हम एक Simple Text Editor में लिख सकते हैं, लेकिन चूंकि JavaScript एक Interpreter Based Programming Language है और JavaScript का Interpreter सामान्यतः Web Browser के अन्दर ही In-Built होता है, इसलिए सामान्यतः JavaScript को Web Pages को Interactive बनाने के लिए Use किया जाता है।

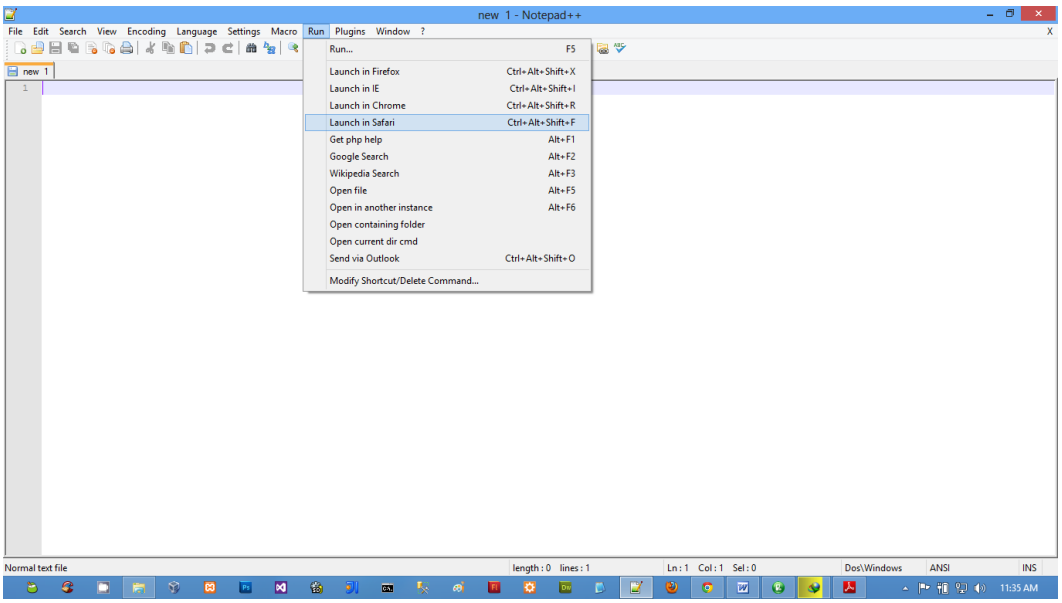
परिणामस्वरूप JavaScript Codes मूल रूप से Web Pages के लिए ही उपयोगी होते हैं और Web Pages Create करने के लिए जितने भी **IDE** (*Integrated Development Environment*) वर्तमान में उपलब्ध हैं, उन सभी को JavaScript Codes को लिखने के लिए Use किया जा सकता है। जैसे *Adobe DreamWeaver, Microsoft Visual Studio, NetBeans, Eclipse* आदि।

चूंकि किसी भी Program को Develop करने में कई Steps Involved होते हैं, जैसे कि Source Codes लिखना, उन्हें Compile या Interpret करना, Bugs को Identify करना, उन्हें Debug करना, Maintain करना, Test करना व Deploy करना। इन सभी कामों को एक ही स्थान पर पूरा करने के लिए यदि कोई Software बना लिया जाए, तो उस Software को **IDE** (Integrated Development Environment) कहते हैं।

हालांकि IDE किसी भी Program को Develop करने में काफी मदद करते हैं, लेकिन फिर भी यदि हम कोई नई Language सीखने के लिहाज से देखें, तो IDE फायदा करने के स्थान पर नुकसान करते हैं।

यानी यदि JavaScript आपके लिए बिल्कुल नई Language है, तो किसी IDE को Use करने के स्थान पर Simple Text Editor का प्रयोग करते हुए JavaScript Codes लिखना आपके लिए ज्यादा फायदेमन्द रहेगा और **Notepad++** किसी भी नई Programming Language को सीखने के लिए मेरा Favorite Text Editor है, जबकि Client Side Web Technologies (HTML, CSS, JavaScript, etc…) IDE के रूप में मुझे Eclipse आधारित **Aptana Studio** पसन्द है।

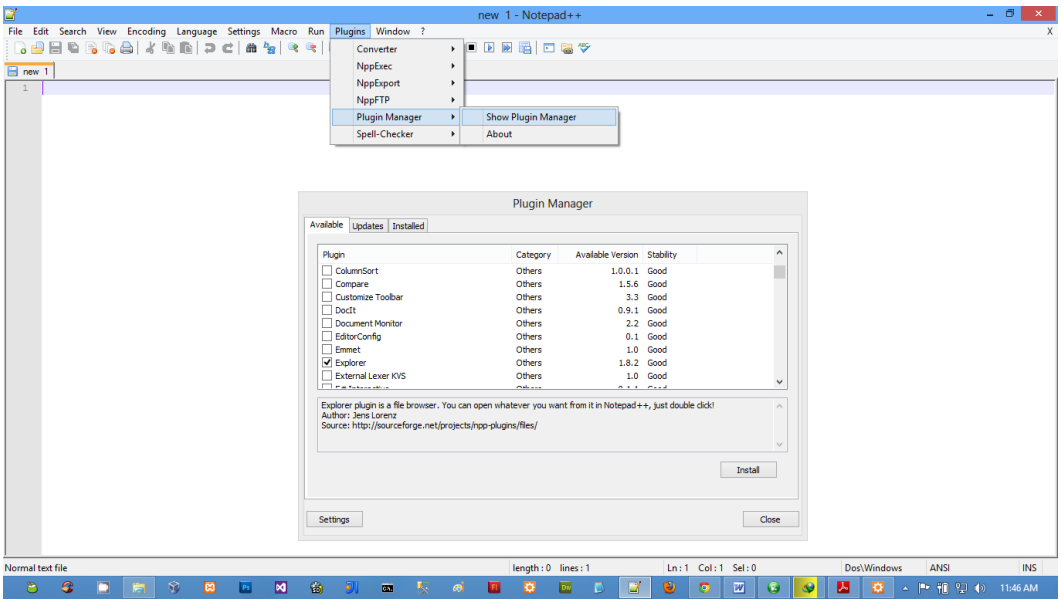
तो सबसे पहले **Notepad++** Text Editor को *http://www.notepad-plus-plus.org /download/* Website से Download करके अपने Computer पर Install कीजिए। ये Text Editor Free Available है। Install करके Open करने पर ये कुछ निम्नानुसार दिखाई देता है



चूंकि HTML, CSS व JavaScript जैसी Scripting Languages, Web Browser में ही Interpret होते हैं और इनका Server Side से कोई Direct Connection होना जरूरी नहीं होता, इसलिए JavaScript Codes के Effects को समझने के लिए हमें हमारे Local Computer पर किसी Local Web Server को Install करने की जरूरत नहीं है, बल्कि JavaScript Codes हमेंशा किसी न किसी HTML Web Page से Link या HTML Web Page में Embed होते हैं, इसलिए जैसे ही हम Web Page को किसी Web Browser में Open करते हैं, JavaScript Interpret होने लगता है।

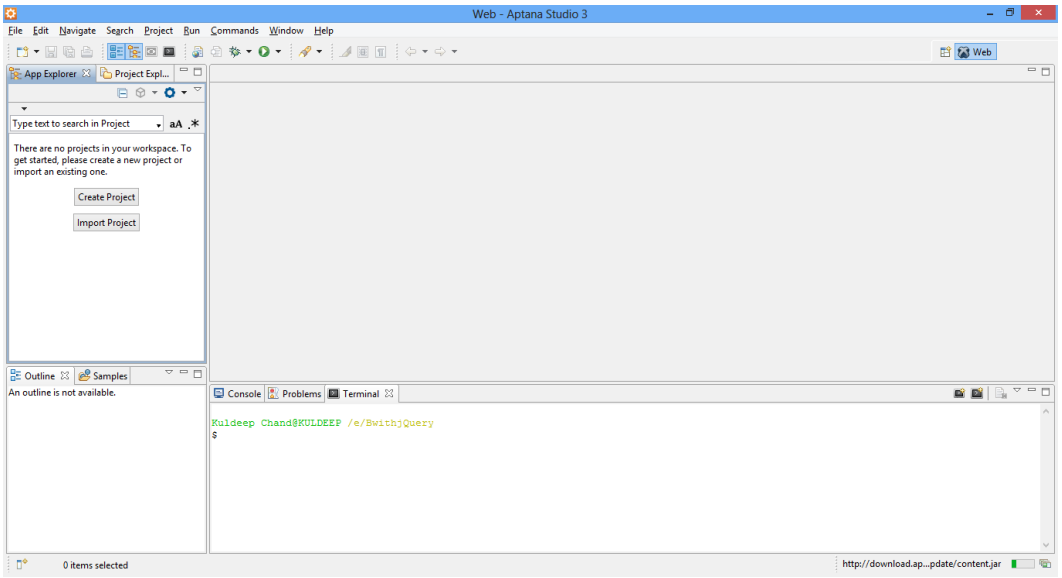
यानी JavaScript Programming सीखने के लिए हमें किसी External Software की जरूरत नहीं है। हमें केवल एक Text Editor की जरूरत है, जहां हम अपने HTML, CSS व JavaScript Codes को लिख सकें व एक Web Browser की जरूरत है, जहां हम हमारे JavaScript Codes के Output को देख सकें।

Notepad की Capabilities को Extend करने के लिए हम इसमें अपनी जरूरत के अनुसार विभिन्न प्रकार के Plug-ins को भी Install कर सकते हैं। विभिन्न प्रकार के Plug-ins Install करने के लिए हमें Notepad++ के **Plug-in** Menu के "**Plugin Manager**" Sub-Menu के "**Show Plugin Manager**" Option को Click करना होता है और हमारे सामने निम्नानुसार एक Dialog Box Open होता है, जिसमें हम उन Plug-ins को Select करके Install कर सकते हैं, जिन्हें हम हमारे Notepad++ Text Editor में Include करना चाहते हैं जबकि Install होने के बाद उस Plugin को उपयोग में लेने के लिए के लिए भी हमें इसी "**Plug-in**" Menu में ही जाना होता है।



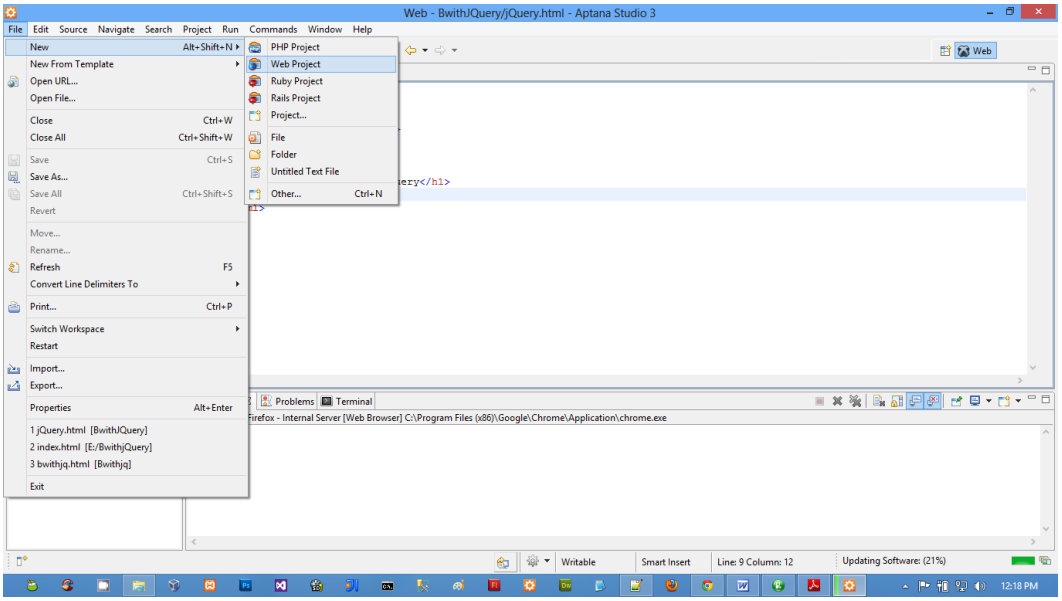
इसके अलावा यदि आप कोई IDE Use करना चाहते हैं, तो आप मेरा Favorite IDE **Aptana Studio** Use कर सकते हैं। ये एक Advance IDE है, इसलिए इसे इसकी पूरी क्षमता के साथ Use करने के लिए आपको कुछ Configuration करने की जरूरत पड सकती है।

ये एक ऐसा IDE है, जिसे Use करने पर आप अपना सारा Code एक ही स्थान पर लिख सकते हैं और उसे इसी Studio में उपलब्ध Internal Web Browser में Run करके उसका Output भी इसी Browser में देख सकते हैं। इस IDE को आप http://www.aptana.com/products/studio3/download Website से Download कर सकते हैं और ये भी पूरी तरह से Free है। Install करके Open करने पर ये IDE कुछ निम्नानुसार दिखाई देता है:

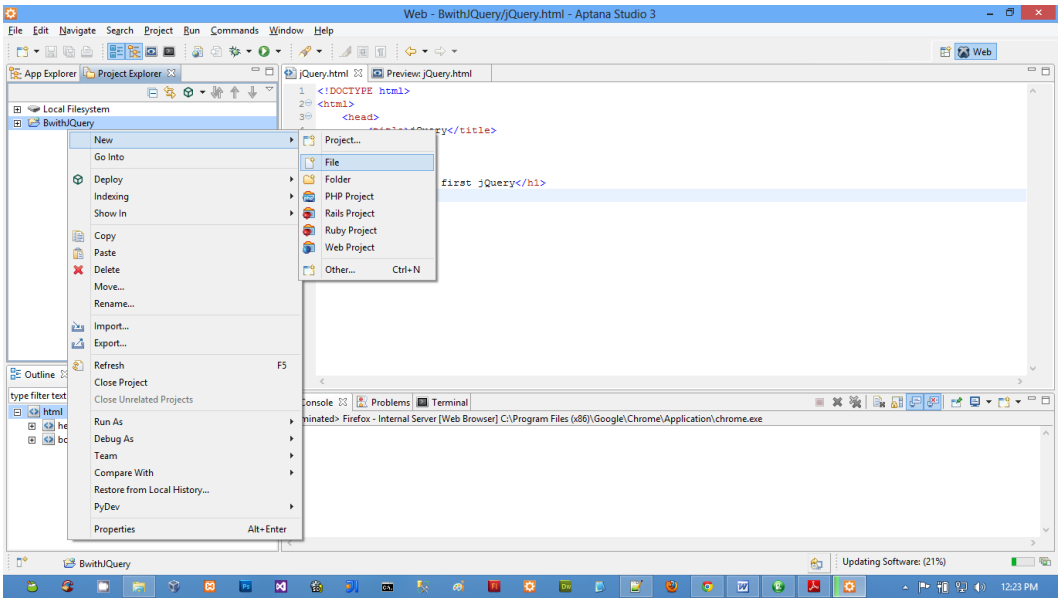


इस IDE की विशेषता ये है कि इस IDE में ही एक Local Web Server व Internal Web Browser भी है। जिसकी वजह से हमें हमारे Program को Test या Debug करने के लिए ***Text Editor*** व ***Web Browser*** के बीच Switch नहीं करना पडता।

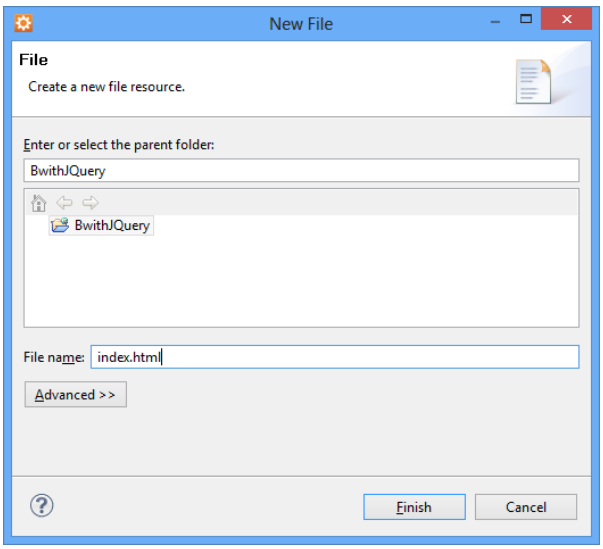
इस IDE को Use करने के लिए सबसे पहले हमें निम्न चित्रानुसार Option को Click करके एक नया **Web Project** Create करना होता है:



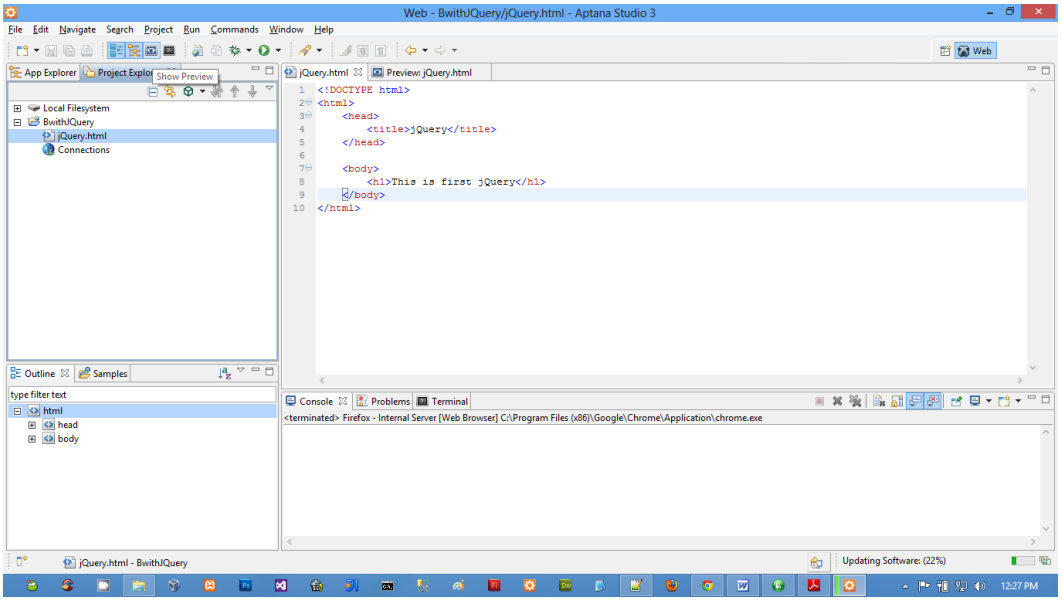
एक Dialog Box Display होता है, जहां हमें हमारे Project का नाम Specify करके Next Button पर नहीं बल्कि **Finish** Button पर Click करना होता है। ऐसा करते ही एक नया Project Create हो जाता है, जिसे हम IDE के Left Side में दिखाई देने वाले **"Project Explorer"** Tab में देख सकते हैं। फिर निम्न चित्रानुसार नई File Create करना होता है:



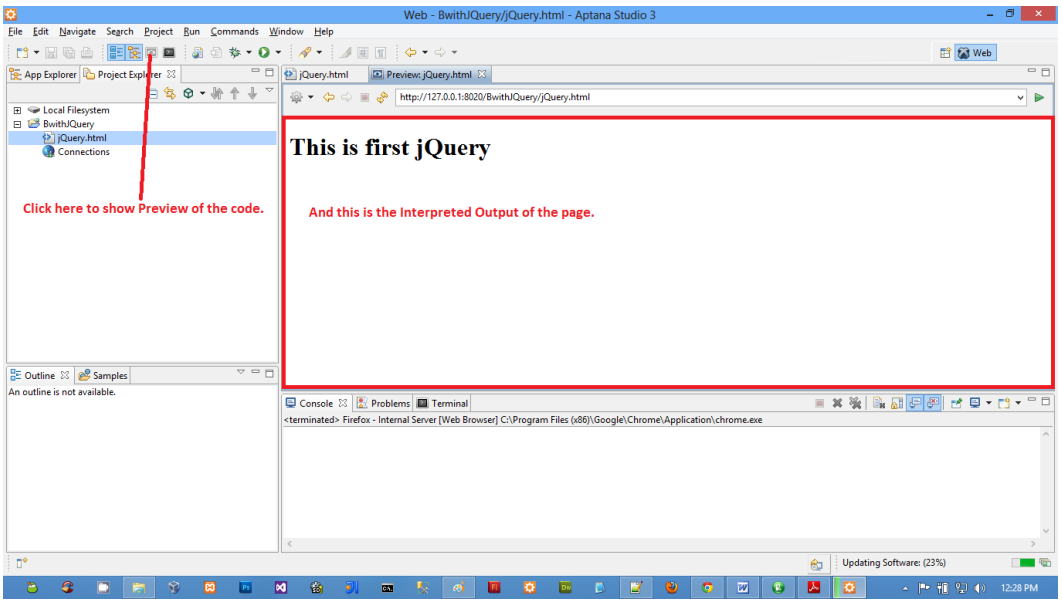
चित्र में दिखाए अनुसार "New => File" पर Click करते ही हमारे सामने एक Dialog Box आता है, जिसमें हमें हमारी File का नाम जैसे कि "**index.html**" Specify करके "**Finish**" Button पर Click करना होता है:



इस प्रकार से हमारे Project में एक नई File Add हो जाती है, जिसमें हम निम्नानुसार HTML, CSS या JavaScript Code लिख सकते हैं:



इस Web Page में HTML, CSS, JavaScript Codes लिखने के बाद उसका Output देखने के लिए हमें अगले चित्र में दिखाए अनुसार IDE के Standard Toolbar में दिए गए "**Show Preview**" Icon को Click करना होता है और हमें हमारे Page का Output निम्न चित्रानुसार दिखाई देने लगता है:



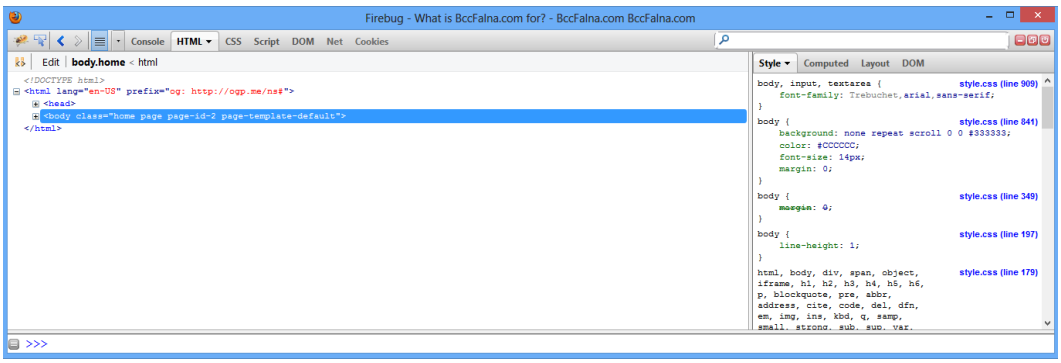
इस IDE के अलावा भी कई और IDEs हैं, जिनका प्रयोग Web Pages Create करने के लिए किया जा सकता है। उदाहरण के लिए आप NetBeans भी Use कर सकते हैं, जो कि मूल रूप से Java Development के लिए Oracle Company द्वारा Provide किया गया IDE है, लेकिन हम इसे Web Development के लिए भी Use कर सकते हैं और ये IDE भी काफी Powerful व Free Available हैं।

जबकि आप Adobe DreamWeaver या Microsoft Visual Studio IDE का भी प्रयोग Web Development के लिए कर सकते हैं, लेकिन ये IDE Free नहीं हैं बल्कि काफी महंगे हैं।

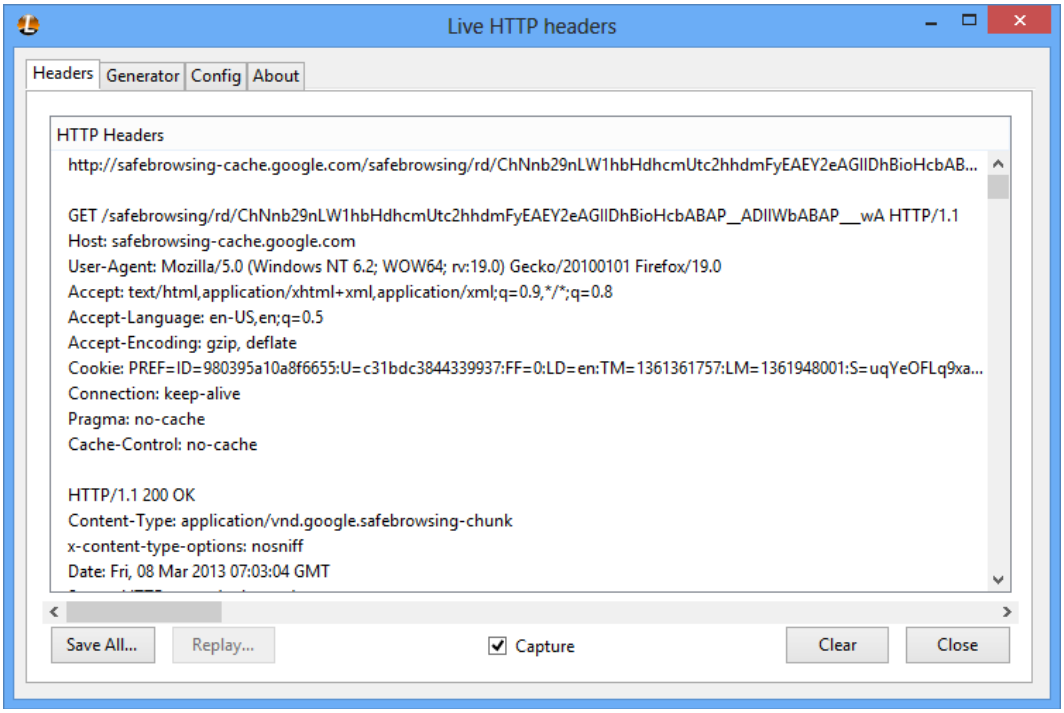
हालांकि आप **Notepad++** या **Aptana Studio** का प्रयोग करके Web Pages Create कर सकते हैं, जिनमें JavaScript Codes लिखकर उनका Effect समझ सकते हैं, लेकिन फिर भी Development के समय विशेष रूप से Codes की Debugging करते समय व Language की Internal Working को बेहतर तरीके से समझने के लिए हमें कुछ और Special प्रकार के Tools की अक्सर जरूरत पडती है और सामान्य रूप से ये **Tools,** Web Browser बनाने वाली Companies ने Default रूप से अपने Web Browser में दे रखा होता है, जिसे "**Developer Tools**" कहते हैं, और विभिन्न Web Browsers में सामान्यतः इन्हें **F12** Function Key Press करके **On/Off** किया जा सकता है।

फिर भी Developer Tools के मामले में Mozilla Firefox Web Browser सबसे Powerful Developer Tools Plugin के रूप में Install करने की सुविधा देता है और ये Tool भी **F12** Function Key द्वारा Enable/Disable कर सकते हैं। यानी यदि आप Mozilla Firefox Web Browser Use कर रहे हैं, तो निम्न Tools को अपने Web Browser में Extension के रूप में जरूर Install करें:

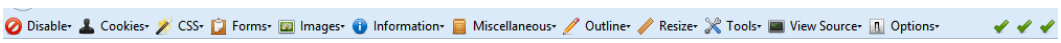
1ण् http://www.getfirebug.com/
ये Tool वास्तव में सभी Web Developers के लिए एक बहुत ही उपयोगी Tool है, क्योंकि ये Tool Web Page Development व Debugging से संबंधित लगभग जरूरी Tools का एक Collection है।



2ण http://livehttpheaders.mozdev.org/
इस Tool का प्रयोग करके हम Web Browser व Web Server के बीच Transfer होने वाले Message की Details प्राप्त कर सकते हैं।

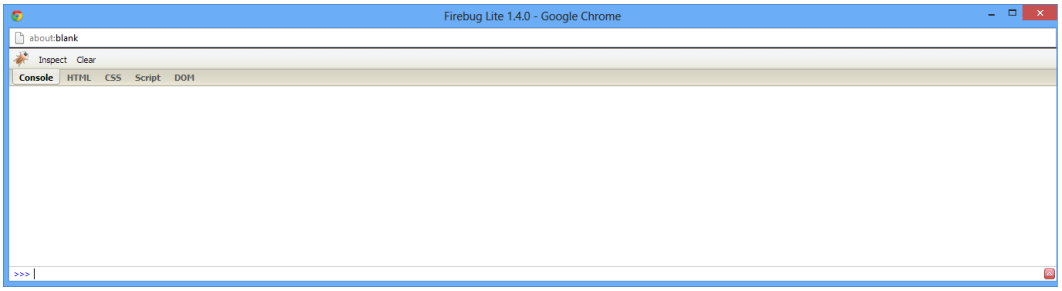


3ण http://chrispederick.com/work/web-developer/
ये Tool Current Web Page से सम्बंधित लगभग सभी Elements की जानकारी व उन्हें Handle व Control करने की सुविधा देता है।

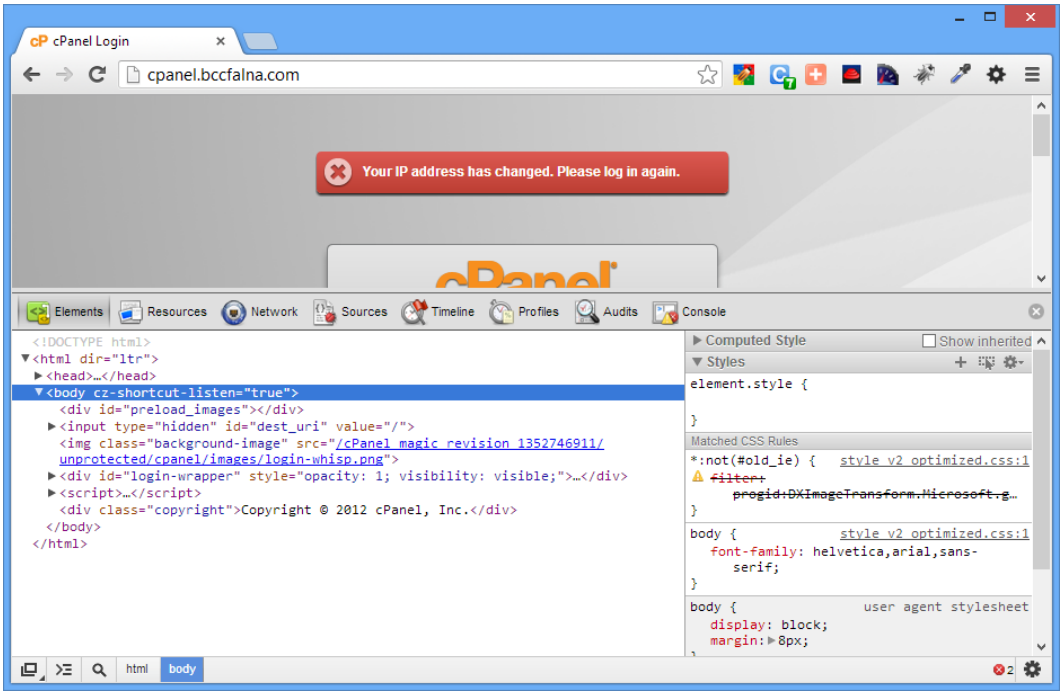


यदि आप Firefox Web Browser को ज्यादा उपयोग में लेते हैं, तो इन तीनों Tools के साथ कुछ समय व्यतीत करना आपके लिए काफी फायदेमन्द रहेगा। लेकिन यदि आप Google Chrome Web Browser को ज्यादा उपयोग में लेते हैं, तो उपरोक्त सभी Tools के Lite या Alternative

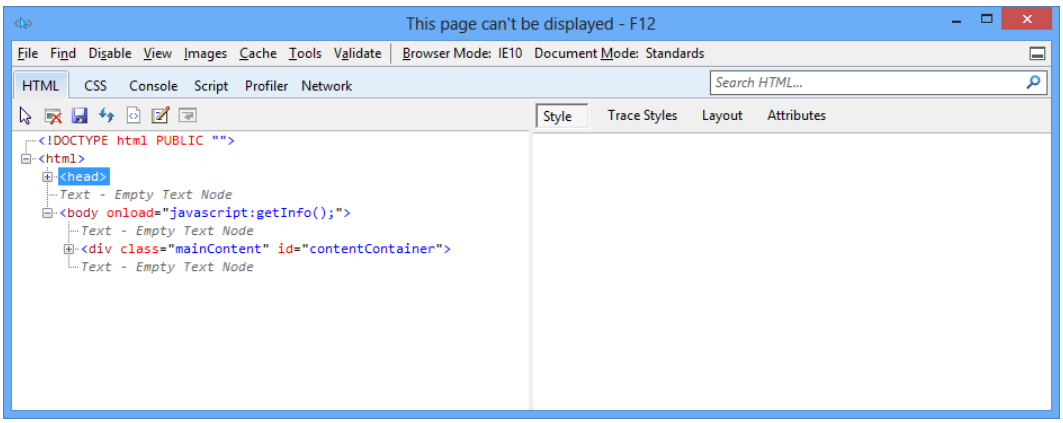
Versions, Google Chrome Web Browser के लिए भी Plug-in के रूप में Available हैं, जो कि **F12** Function Key Press करने पर कुछ निम्नानुसार दिखाई देते हैं:



Google Chrome के लिए Firebug Tool का ये एक Lite Version है। इसके अलावा Google Chrome का स्वयं का भी एक Developer Tool है, जिस को उस स्थिति में **F12** Function Key द्वारा Activate किया जा सकता है, जबकि आपने Google Chrome में "**Firebug Lite**" Version को Install न किया हो। लेकिन यदि आपने "Firbug" के Lite Version Extension को Install किया है, तो इस Default **Developer Tools** को Open करने के लिए आपको Google Chrome के **Tools** Menu में जाकर "**Developer Tools**" Option को Click करना होगा। ये Tool कुछ निम्नानुसार दिखाई देता है।



इसके अलावा Microsoft ने अपने Latest Web Browser के साथ भी अपना एक Developer Tool Provide किया है और वह भी **F12** Function Key द्वारा ही Activate होता है, जो कि लगभग Firebug Took की Exact Copy है। ये Tool कुछ निम्नानुसार दिखाई देता है:



इनके अलावा Apple Safari व Opera Web Browsers का भी अपना Develop Tool है। उपरोक्त सभी Tools देखकर आप समझ ही गए होंगे कि ये सभी Tools लगभग एक समान ही हैं। इसलिए आप चाहे जो Web Browser Use कर रहे हों, आपको इन Tools को अच्छी तरह से Use करना आना ही चाहिए।

वैसे भी यदि आप Web Developer बनना चाहते हैं, तो आपके Computer में सभी Modern Web Browsers Installed होने चाहिऐं और आपको अपने Web Page को सभी Modern Web Browsers में Test करना चाहिए, ताकि आपको पता चल सके कि एक ही Web Page अलग–अलग Web Browsers में कितना अलग दिखाई दे सकता है।

इसके अलावा हालांकि हमने कुछ Tools के बारे में Discuss किया, लेकिन विभिन्न प्रकार की Requirements को पूरा करने हेतु विभिन्न Web Browsers के बहुत सारे Tools Plug-in के रूप में Available हैं, जिन्हें सुविधानुसार जरूर उपयोग में लेना सीखना चाहिए।

साथ ही इन अलग–अलग Web Browsers के "**Developer Tools**" में भी कुछ Special Types के अलग–अलग Options हैं, जो किसी दूसरे Web Browser के Developer Tool में उपलब्ध नहीं हैं। इसलिए सभी Web Browsers के Developer Tools को अच्छी तरह से समझना आपके लिए उपयोगी रहेगा

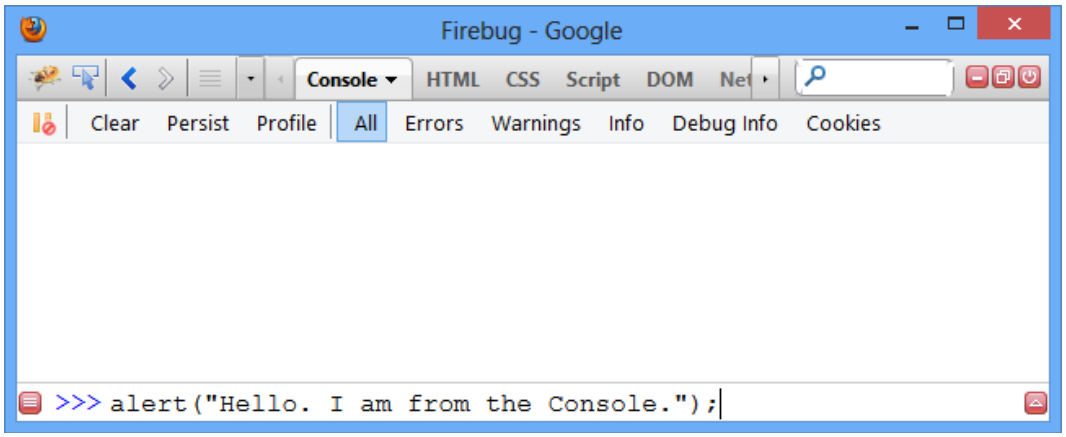
उदाहरण के लिए Internet Explorer का Developer Tools Use करके यदि हम Web Page के किसी Code में Change करते हैं, तो हम उस Code को Hard Disk पर एक अलग File के रूप में Save करके रख सकते है, जबकि ये सुविधा किसी भी अन्य Web Browser के Developer Tools में नहीं है।

यानी यदि आप कोई IDE या Text Editor Use न करें, तो आप सीधे ही Internet Explorer के Developer Tools को एक IDE की तरह Use करते हुए भी JavaScript Codes को Interpret कर सकते हैं, नया Web Page Create कर सकते हैं, उसकी Stylesheet बना सकते हैं।

यानी हर Web Browser के Developer Tools की अपनी विशेषता है इसलिए आपको सभी Web Browsers के Developer Tools के बारे में ज्यादा से ज्यादा जानना चाहिए ताकि आपको पता रहे कि किसी Specific Type की Requirement को पूरा करने के लिए आपको कौनसे Web Browser के Developer Tools की जरूरत है।

Developer Tools Console

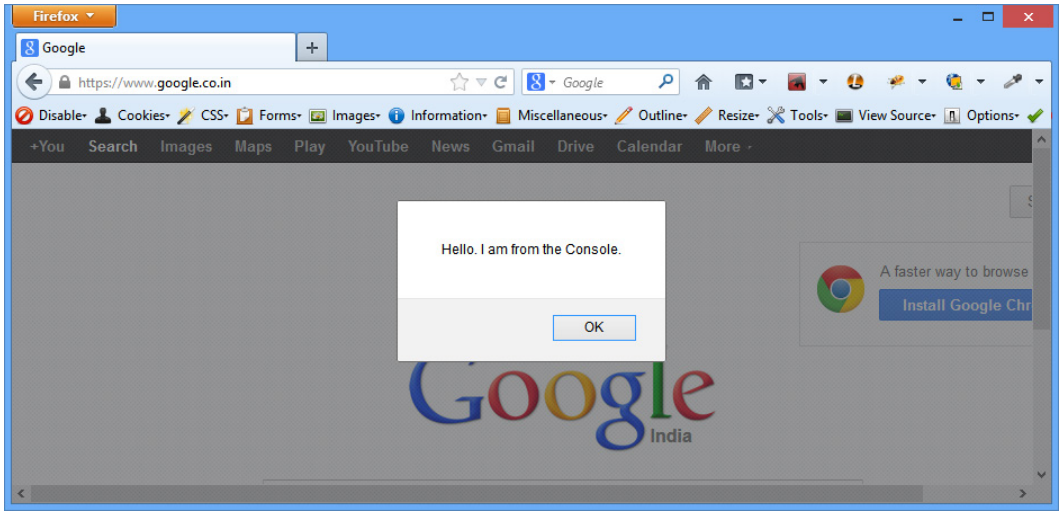
जब आप विभिन्न Web Browsers के Developer Tools या Firebug Tool को Inspect करेंगे, तो आप देखेंगे कि उन सभी Tools में "**Console**" नाम का एक Tab है। ये वह स्थान है, जहां पर आप Directly JavaScript Codes लिखकर सीधे ही Web Browser में Code की Functionality का प्रभाव देख सकते हैं। यही नहीं, Web Browser में Loaded किसी भी Web Page को इस Console में JavaScript Code लिखकर उस Page पर JavaScript Code के Effect को देखा जा सकता है।



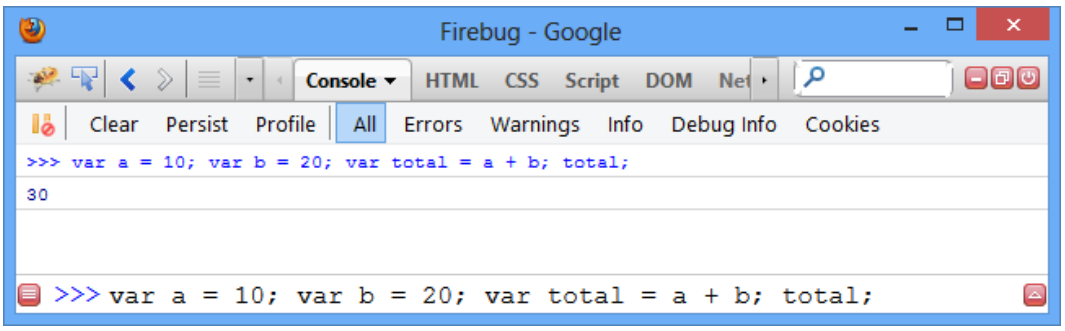
जैसाकि उपरोक्त चित्र में हम देख सकते हैं कि हमने **>>>** Prompt पर निम्न Code लिखा है:

```
>>> alert("Hello. I am from the Console.");
```

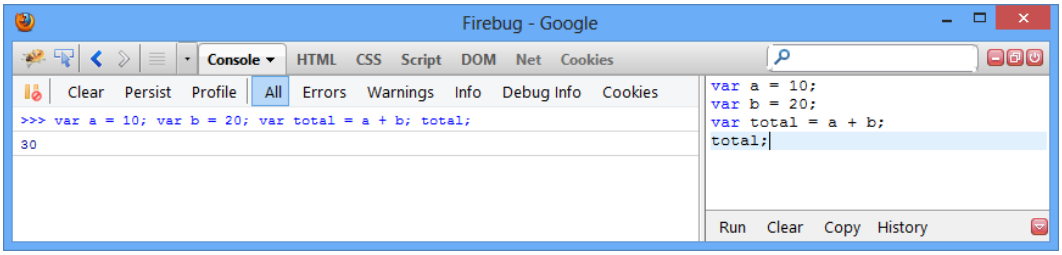
जैसे ही हम इस Code को लिखकर Enter Key Press करते हैं, Web Browser में निम्न चित्रानुसार एक **Alert Window** Display होने लगता है, जिसमें उपरोक्त Code में लिखा गया Message ही होता है:



यदि हम एक से ज्यादा Lines का JavaScript Code लिखना चाहें, तो सभी Codes के अन्त में "**Semicolon**" का प्रयोग करते हुए Multiple Lines का Code लिख सकते हैं। जैसे:

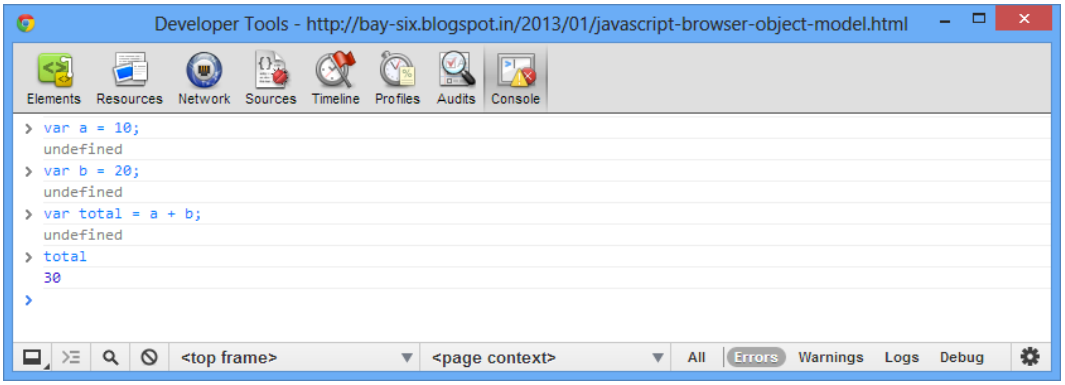


यदि हम चाहें तो इस Firebug Tool के इस Console को Multiline Mode में भी Open कर सकते हैं। इसके लिए हमें इस Tool के Bottom Right में दिखाई देने वाले Red Color के Button को Click करना होता है। परिणामस्वरूप यही Console निम्नानुसार Multiline Mode में दिखाई देने लगता है:

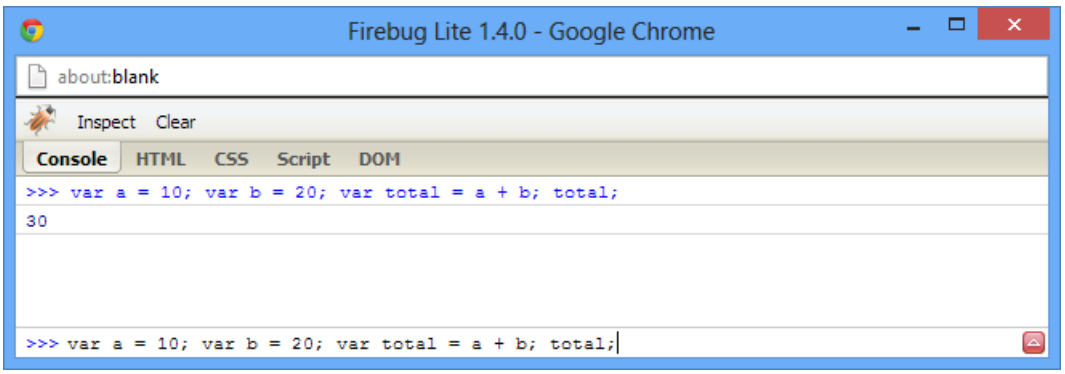


जब हम Firebug के Console को Multiline Mode में Use करते हैं, तो JavaScript Code को Interpret करने के लिए हमें Enter Key के स्थान पर **Ctrl+Enter** Key Combination को Use करना पडता है।

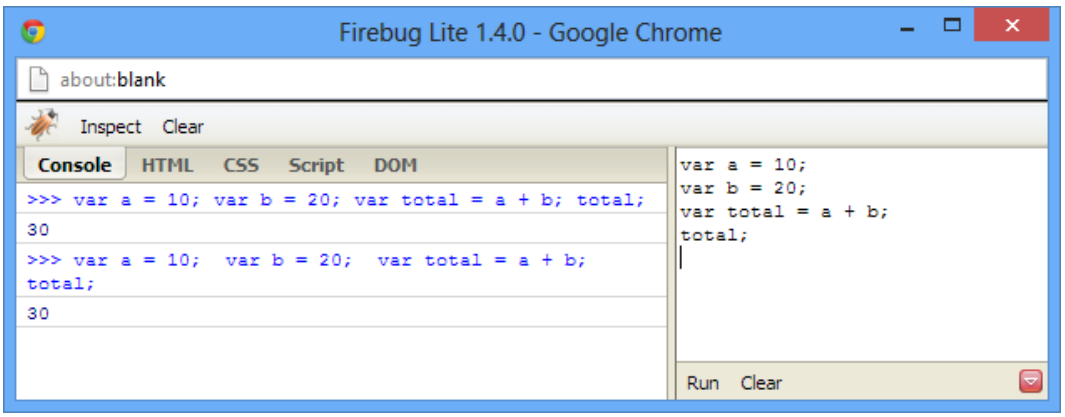
इसी तरह से Google Chrome Web Browser में यदि **Ctrl+Shift+J** Key Combination Use किया जाए, तो हमारे सामने निम्नानुसार Google Chrome का Default JavaScript Console Display हो जाता है, जिसे हम ठीक Firebug Console की तरह ही Use कर सकते हैं:



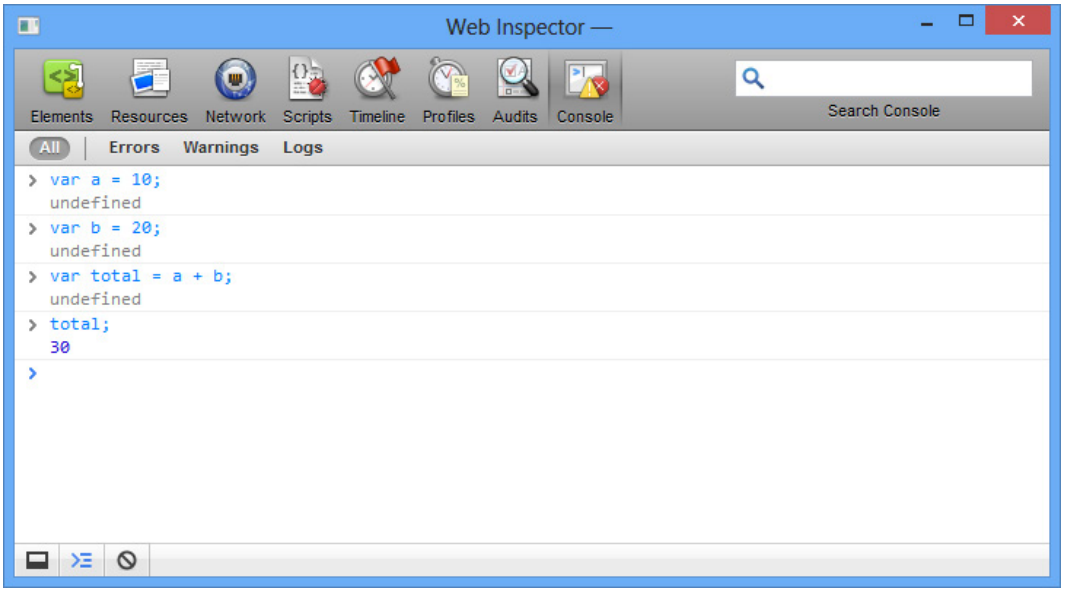
जबकि यदि हम चाहें तो Google Chrome के Firebug Lite Version के Console को Function Key **F12** द्वारा Activate कर सकते हैं और ये हमें निम्नानुसार दिखाई देता है:



जबकि इसी को Multiline Mode में Open करने पर ये निम्नानुसार दिखाई देता है:

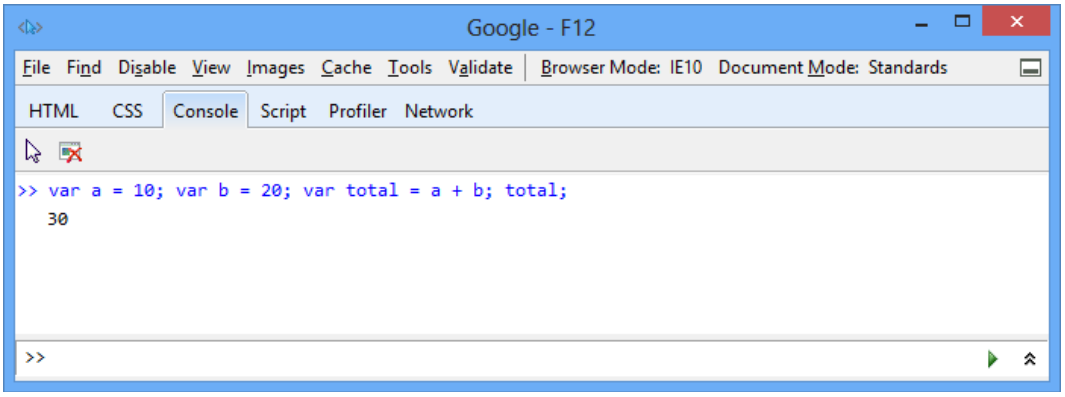


इसी तरह से **Apple Safari** Web Browser का Developer Tool Exactly Google Chrome के Default Developer Tool की तरह दिखाई देता है, जो कि कुछ निम्नानुसार होता है:

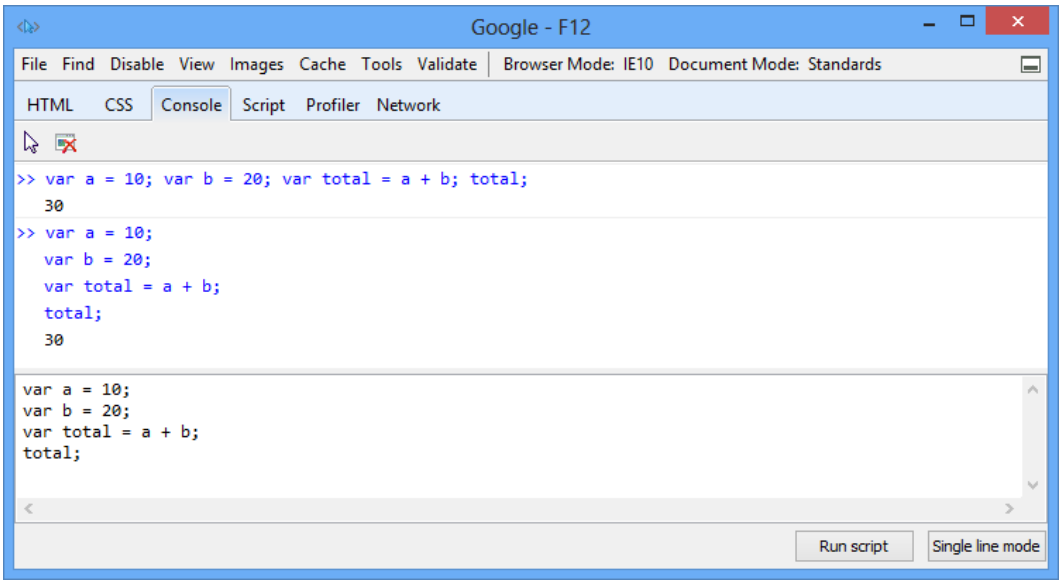


Apple Safari Web Browser के Developer Tools को Activate करने के लिए हम **Ctrl+Alt+I** Key Combination का प्रयोग कर सकते हैं।

इसी तरह से हम Internet Explorer का भी Console Use कर सकते हैं, जो कि निम्नानुसार होता है:



चूंकि, ये Developer Tool लगभग पूरी तरह से Firebug के समान है, इसलिए हम इसे भी Multiline Mode में Use कर सकते हैं, जिसके लिए हमें इस Developer Tools के Bottom Right में दिखाई देने वाले Up-Arrow को Click करना होता है। Multiline Mode में ये निम्नानुसार दिखाई देता है:



Multiline Mode में पूरे JavaScript Code को Run करने के लिए हमें **Ctrl+Enter** Key Combination को Use करना होता है अन्यथा हम "**Run script**" Button पर Click भी कर सकते हैं।

विभिन्न Web Browsers के **Developer Tools** सामान्यतः Web Browser की Screen से **Dock** या Connected रहते हैं, लेकिन हम इन्हें Web Browser के Window से **Un-Dock** या अलग भी कर सकते हैं और अलग करने के बावजूद ये उस Web Page से Linked रहते हैं, यानी उस Web Page की Information से Associated रहते हैं, जिस पर इन्हें Activate किया गया था।

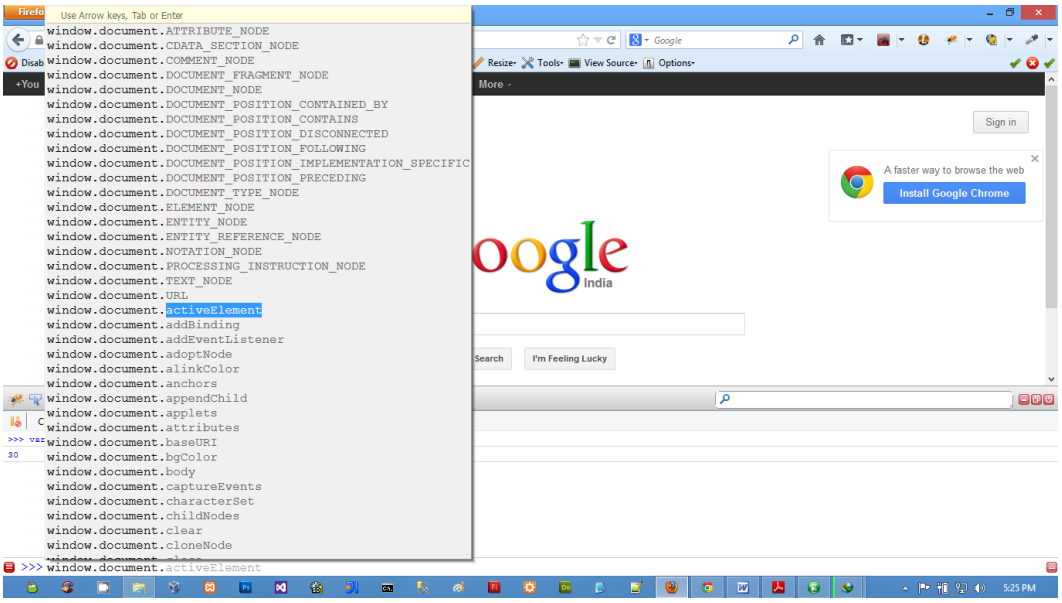
हालांकि विभिन्न Web Browsers के सभी **Consoles** बिना किसी तरह की परेशानी के JavaScript Codes को Interpret करने में पूरी तरह से सक्षम हैं, फिर भी **Firebug** मेरा

Favorite Console है, क्योंकि ये हमें उस समय Automatically **IntelliSense** की सुविधा देता है, जब हम इसमें अपना JavaScript Code Type कर रहे होते हैं।

यानी ये हमें Professional IDE की तरह Code Type करते समय Code Hint देता रहता है, जिससे हमें JavaScript Codes को याद रखने अथवा पूरा Code Type करने की जरूरत नहीं रहती, बल्कि हम Code को Select करके **Tab** Key या **Enter** Key को Press कर सकते हैं।

साथ ही केवल इसी Console में JavaScript Codes विभिन्न **Color Text** के अनुसार दिखाई देते हैं, जिससे JavaScript Codes लिखना काफी आसान हो जाता है। जबकि अन्य Web Browsers के Consoles में सारे JavaScript Codes **Plain Text** की तरह दिखाई देते हैं।

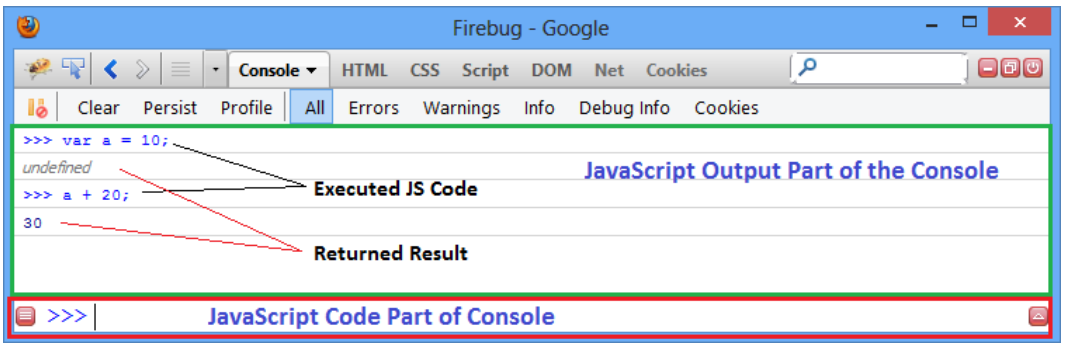
यानी हम **Firebug Console** को एक प्रकार से JavaScript IDE की तरह उपयोग में ले सकते हैं। इस सुविधा को निम्न चित्रानुसार समझा जा सकता है:



लेकिन ये सुविधा हमें केवल तब प्राप्त होती है, जब हम Single Line Console Mode में होते हैं। Multiline Console Mode में ये Feature काम नहीं करता, हालांकि Multiline Mode में भी JavaScript Codes विभिन्न Color में दिखाई देते हैं, जो कि एक Extra Benefit है।

सभी Web Browsers के *Developer Tools* के **Console Window** के दो हिस्से होते हैं। पहला हिस्सा वह हिस्सा होता है, जहां हम JavaScript Codes लिखते हैं, जबकि दूसरा हिस्सा वह हिस्सा होता है, जहां उस JavaScript Code के Interpret होने पर Return होने वाला Output Display होता है।

इस Output Window में ऊपर की तरफ वह JavaScript Code दिखाई देता है, जो Run हुआ है, जबकि नीचे की ओर उस Code के Run होने से Return होने वाला Output दिखाई देता है। इसे हम निम्न चित्रानुसार बेहतर तरीके से समझ सकते है:



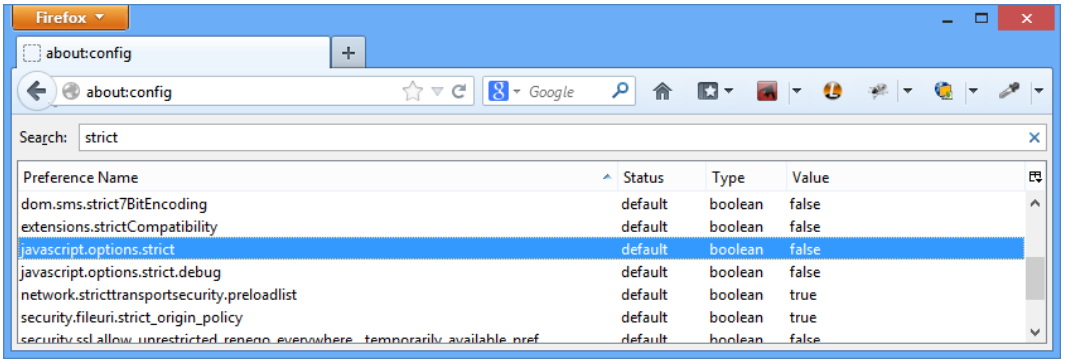
यानी हम इस Console को विभिन्न प्रकार के **Core JavaScript** Codes को Test करने के लिए भी Use कर सकते हैं और हमें हर JavaScript Code के Effect को देखने के लिए अलग से Web Page Create करने की जरूरत नहीं है, जिससे JavaScript सीखने के लिहाज से समय की काफी बचत हो जाती है।

जब हम Firebug के Console को JavaScript Codes के लिए Use कर रहे होते हैं, तब हमें Firefox Web Browser में एक Configuration Setting जरूर करनी चाहिए, ताकि Firefox Web Browser किसी भी तरह का **Warning Message** जरूर Display करे।

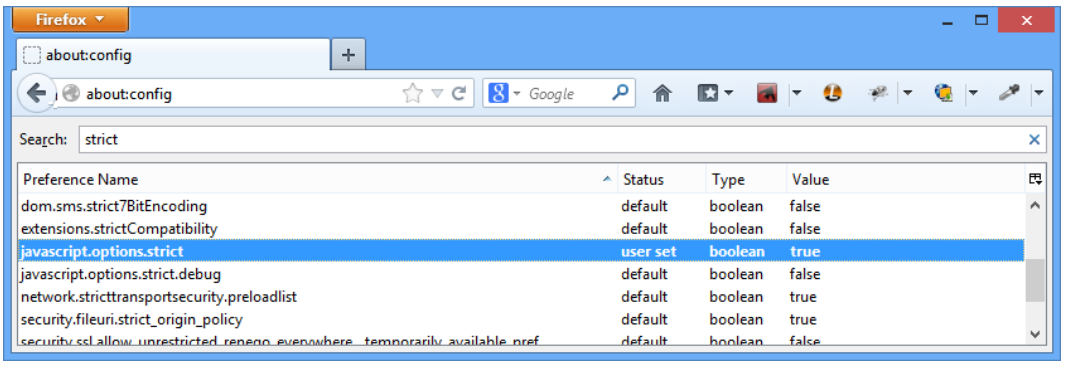
क्योंकि सामान्यतः Warning Message Web Page पर किसी तरह का Direct प्रभाव नहीं डालते, इसलिए Default रूप से Firefox में Warning Message **Disabled** रहता है। लेकिन यदि हमारे Web Page में बहुत ज्यादा Warning Messages हों, तो हमारे Web Page के Web Browser में Load होने की Speed काफी कम हो जाती है।

इसलिए जहां तक हो सके, हमें JavaScript से संबंधित सभी Warning Message को भी Resolve करना चाहिए। यदि हम ऐसा करते हैं, तो हमारा JavaScript Code पूरी तरह से Bug Proof बनेगा। ये Setting करने के लिए हमें निम्न Steps Follow करने होते हैं:

1. Firefox Web Browser के ।ककतमेइंत में **about:config** लिखकर Enter करें।
2. एक Configuration Page दिखाई देगा जिसमें **javascript.options.strict** Option को Select करके Double Click करें।



3. ऐसा करते ही इस Option की Value Property में **false** के स्थान पर निम्न चित्रानुसार **true** Set हो जाएगा:



JavaScript Console उस समय बहुत उपयोगी होते हैं, जब हम JavaScript सीख रहे होते हैं। क्योंकि सीखते समय हम छोटे–छोटे Programs बनाते हैं व उन Programs के Statements के Execution के तरीके व कार्यप्रणाली को समझने की कोशिश करते हैं। उस समय लिखे गए Codes किसी भी तरह से Practically या Professionally Useful नहीं होते।

इसलिए इन Codes को Web Pages में लिखकर उन Web Pages को Run करके JavaScript Codes का Effect देखने में ज्यादा समय लगता है। जबकि Web Browser में हम किसी भी Web Page को Load करके उस पर JavaScript Codes के Effect देखने के लिए उन JavaScript Codes को सीधे ही इन Console में लिख सकते हैं, जिससे समय की बचत होती है।

साथ ही कई बार हमारे JavaScript Codes में ऐसे Bugs होते हैं, जो यदि Web Page में Embedded हों, तो उन्हें Find करना आसान नहीं होता। ऐसे में इस प्रकार के Buggy Codes को हम सीधे ही इन Console में लिखकर Test कर सकते हैं और Codes के पूरी तरह Debug हो जाने पर उन्हें मुख्य Web Page में Embed कर सकते हैं, जिससे इस बात की Surety हो जाती है कि Web Browser के Console में आपने जो JavaScript Code लिखकर Test किया है, वह पूरी तरह से Bug Free है। यानी ये Console मूल रूप से JavaScript Codes को Test व Debug करने में हमारी काफी मदद करते हैं।

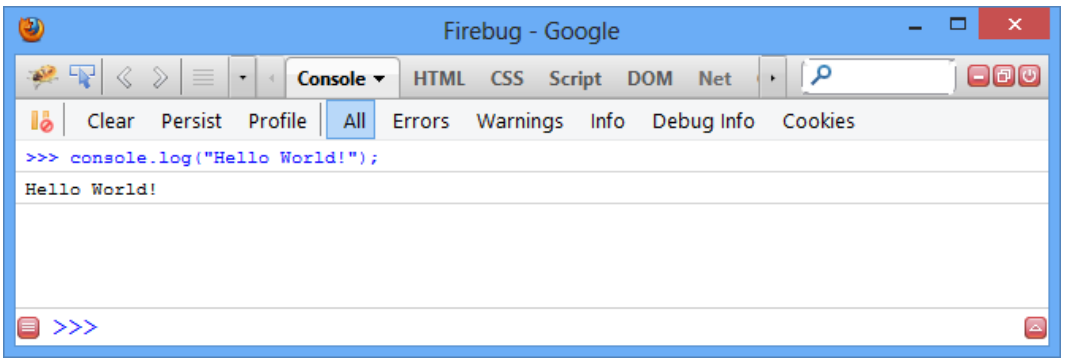
Display Message in Console

जब हम Console में कोई Message या किसी Variable अथवा Object की Values को Display करना चाहते हैं, तो हमें इस **console** Object के **log()** Method को Use करना होता है और इस log() Method में हम जो भी Variable या String (*Text written in Single or Double Quote Pair*) Specify करते हैं, उस Variable में Stored Value या String हमें Console Window के Output में दिखाई देती है।

उदाहरण के लिए यदि हम "**Hello World**" String को Console Window में Display करना चाहें, तो हम Console Interpreter में निम्नानुसार Code लिखकर ऐसा कर सकते हैं:

```
console.log("Hello World!");
```

जब हम ये Code लिखकर Enter Key Press करते हैं, तो हमें निम्नानुसार Output प्राप्त होता है:



JavaScript in Webpage

उपरोक्त Section के Discussion से अब हम इस स्थिति में हैं कि विभिन्न Core JavaScript Codes को Web Browser के **Developer Tools** के **Console Window** में लिखकर Test कर सकते हैं।

लेकिन चूंकि इस पुस्तक में Web Browser ही हमारे JavaScript Codes का Host Environment है और Web Browser में हमेंशा Web Pages Load होते हैं, जिनमें JavaScript Codes को Use करके हम Web Pages को ज्यादा Interactive बनाते हैं। इसलिए JavaScript को किसी Web Page के साथ किस प्रकार से Use किया जाता है, इस बात को समझना बहुत जरूरी है। पुस्तक के इस Section में हम इस विषय में जानेंगे।

JavaScript को HTML Document में Use करने के लिए हमें **<script>** Element को Use करना होता है। सामान्यतः हम JavaScript को दो तरीकों से HTML Web Page से Link कर सकते हैं। पहले तरीके में JavaScript Codes को सीधे ही HTML Web Page में <script> Element के बीच Enclose किया जाता है जबकि दूसरे तरीके में सभी JavaScript Codes को एक External File में Specify किया जाता है और केवल उस JavaScript File को HTML Document में Link किया जाता है।

<script> Element

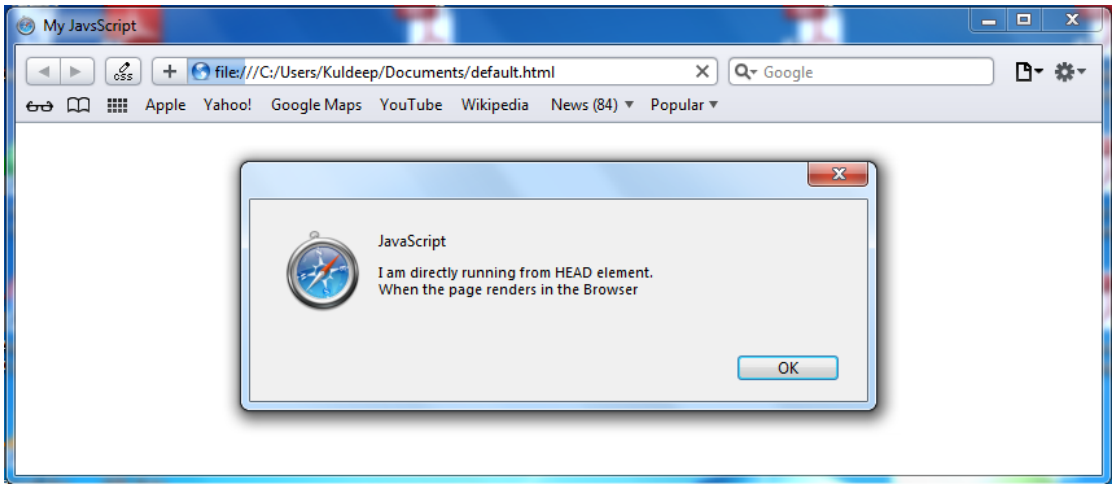
जब हम किसी JavaScript Code को सीधे ही अपने HTML Page में Embed करना चाहते हैं, तब भी हम **<head>** व **<body>** दोनों स्थानों पर JavaScript Code को Specify कर सकते हैं। जब हम HTML Document के Head या Body में कोई JavaScript Code Specify करते हैं, तो वह JavaScript Code Web Page के Web Browser में Render होते समय ही Run हो जाता है।

जबकि किसी JavaScript Code को किसी External File में Store करने पर ऐसा नहीं होता और हमें External JavaScript File के विभिन्न JavaScript Codes को HTML Document में किसी न किसी Event के Response में Call करना पडता है। किसी JavaScript Code को सीधे ही HTML Page में Embed करने के लिए हमें <script> Element को निम्नानुसार Specify करना होता है:

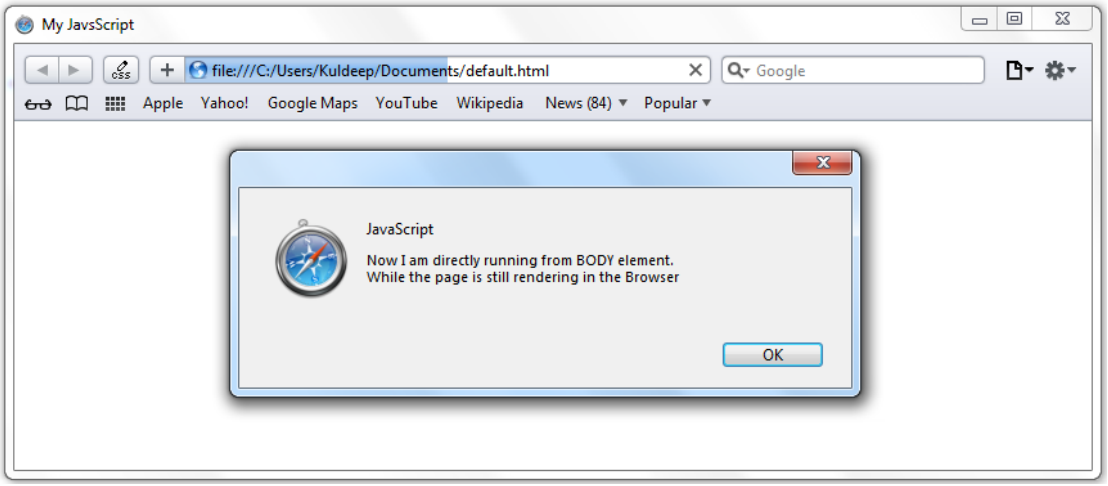
```
<!DOCTYPE html>
<html>
```

```
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <title>My JavaScript</title>
    <script>
        alert("I am directly running from HEAD element.\n
                When the page renders in the Browser");
    </script>
</head>

<body>
    <script>
        alert("Now I am directly running from BODY element.\n
                While the page is still rendering in the Browser");
    </script>
</body>

</html>
```

चूंकि HTML5 में JavaScript को Client Side Scripting Language के रूप में पूर्ण मान्यता मिल गई है, इसलिए हमें **<script>** Element में किसी और Attribute को Specify करने की जरूरत नहीं है बल्कि हम सीधे ही उपरोक्त Code अनुसार किसी JavaScript Code को **<script>** Element के बीच Enclose कर सकते हैं।



जब Web Page को Render किया जाता है, तब सबसे पहले उपरोक्त चित्रानुसार HTML Page का Head Section Render होता है और हमें उपरोक्त Dialog Box दिखाई देता है। आप चित्र के Address Bar में देख सकते हैं कि अभी भी Web Page पूरी तरह से Load नहीं हुआ है, क्योंकि इस Alert Dialog Box को जब तक Close नहीं किया जाएगा, तब तक Web Page आगे Render नहीं होगा।



जैसे ही हम पहले Alert Dialog Box के OK Button पर Click करते हैं, एक और Alert Dialog Box Display होता है, जो कि हमने HTML Document की Body में Specify किया था। अभी भी आप उपरोक्त चित्र के Address Bar को देख सकते हैं, जिसमें हमारा Web Page लगभग आधा Render हो चुका है, लेकिन पूरी तरह से तभी Render होगा, जब हम इस Body Element के Alert Dialog Box को भी Close कर देंगे।

इस तरह से हम किसी HTML Document में Inline JavaScript Codes को Head व Body में Specify कर सकते हैं।

किसी HTML Document के अन्दर किसी JavaScript Code को Specify करने पर वह JavaScript Code भी HTML Code की तरह ***Top to Bottom*** व ***Left to Right*** Flow होता है और साथ ही साथ Interpret भी होता रहता है। जिसकी वजह से सभी Executable JavaScript Codes Execute भी होते रहते हैं। यानी Inline JavaScript Codes पूरे Web Page के Web Browser में Load होने का Wait नहीं करते बल्कि Sequential Form में Execute होते रहते हैं।

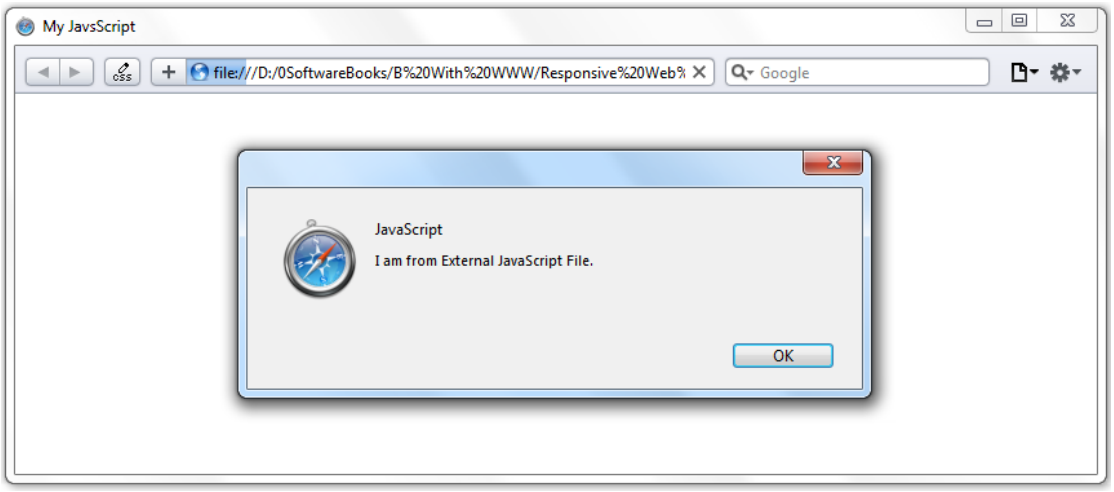
जब हम किसी JavaScript Code को किसी External JavaScript File में Store रखते हैं और फिर उस File को अपने HTML Document में Link करना चाहते हैं, तब भी हमें **<script>** Element का ही प्रयोग करना होता है, लेकिन इस बार हमें निम्नानुसार <script> Element के **src** Attribute में उस JavaScript File के URL को Specify करना होता है, जिसे हम हमारे Current HTML Document के साथ Link करना चाहते हैं।

```
<!-- index.html -->
<!DOCTYPE html>
<html>
    <head>
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
        <title>My JavaScript</title>
        <script src="myJS.js"></script>
    </head>

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

```
//myJS.js
alert("I am from External JavaScript File.");
```

उपरोक्त Code अनुसार हम हमारे HTML Document में myJS.js नाम की External JavaScript File को Link कर रहे हैं और External JavaScript File में हमने उपरोक्त Code अनुसार एक Alert Message Specify किया है।

इस बार जब हम हमारे Web Page को Render करते हैं, तब भी उपरोक्त JavaScript Code हमें हमारे Web Page में निम्नानुसार एक Alert Dialog Box के साथ दिखाई देता है:



जैसाकि उपरोक्त चित्र के Address Bar में देख सकते हैं कि अभी भी Web Page पूरी तरह से Web Browser में Load नहीं हुआ है और हमने जो Code External JavaScript File में लिखा है, वह Code Run हो रहा है और हमें एक Alert Dialog Box दिखाई दे रहा है।

ऐसा इसीलिए हो रहा है, क्योंकि Web Page हमेंशा Top to Bottom Render होता है और जैसे ही Web Browser को कोई Script या Link Element मिलता है, Web Browser उस Link वाले Resource पर पहुंच जाता है और उसे Web Browser की Memory में Download करना शुरू कर देता है।

चूंकि JavaScript Codes Web Browser के लिए Executable Codes होते हैं, इसलिए JavaScript Code में Specified Alert Function Execute हो जाता है और हमें उपरोक्त चित्रानुसार Alert Dialog Box दिखाई देने लगता है।

ध्यान रखने वाली बात ये है कि Head Element में जिन Scripts या Stylesheet Files को Link करने के लिए <script> या <link> Element का प्रयोग किया जाता है, जब तक उन Resources के सभी Codes Web Browser में Download नहीं हो जाते, तब तक Web Browser आगे नहीं बढता।

यानी जब तक हम उपरोक्त चित्र में दिखाए गए Alert Dialog Box के **OK** Button पर Click नहीं करते, तब तक हमारा Web Page आगे नहीं बढता और Web Browser में Render नहीं होता।

चूंकि JavaScript का काम हमेंशा पूरा Web Page Load हो जाने के बाद शुरू होता है, इसलिए किसी JavaScript File को कभी भी Web Page के Head में Link नहीं करना चाहिए, बल्कि पूरा

HTML Document Specify करने के बाद Closing Body Element से Just पहले <script> Element का प्रयोग करके External JavaScript File को Link करना चाहिए।

External JavaScript File को इस प्रकार से HTML Document से Link करने के दो फायदे हैं। पहला फायदा ये है कि जब तक HTML Document पूरी तरह से Render नहीं हो जाता, तब तक JavaScript File Web Browser में Download नहीं होती, जो कि बिल्कुल सही व्यवस्था है क्योंकि JavaScript वह अन्तिम File होनी चाहिए, जिसे Web Browser में Load होना चाहिए।

दूसरा फायदा ये है कि Web Page User के सामने जल्दी Render होता है। यदि हम Scripts को Head Element में Link करें, तो जब तक पूरी Script Web Browser में Download नहीं हो जाएगी, तब तक Web Page आगे Render नहीं होगा। इसलिए Head Element में JavaScript File को Link करने पर User को लगता है Web Page काफी बडा है या Web Site काफी Slow है, इसलिए सामान्यतः User आपकी Site से Skip कर सकता है, जो कि अच्छी बात नहीं है।

Performance की दृष्टि से Yahoo, Google, Facebook, Bing, Apple आदि सभी बडी कम्पनियां इसी तरह से विभिन्न JavaScript Files को अपने Web Page में Specify करती हैं। हालांकि ये सनातन सत्य नियम नहीं है और आप अपनी Script को Head या Body में कहीं भी Link कर सकते हैं। लेकिन फिर भी किसी भी Script File को हमेंशा निम्नानुसार Closing Body Element से Just पहले Specify करने पर Web Page ज्यादा तेजी से Web Browser में Render होता है।

```
<!-- index.html -->
<!DOCTYPE html>
<html>

<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <title>My JavaScript</title>
</head>

<body>
    <!-- Whole Page Content -->

    <!-- script just before closing body element. -->
    <script src="myJS.js"></script>
</body>

</html>
```

जब हम किसी External JavaScript File को अपने HTML Document में Link करना चाहते हैं, तब ये जरूरी नहीं होता है कि हमारी External JavaScript File का Extension **.js** ही हो। Web Browser कभी भी किसी Resource का Extension Check नहीं करता। इस वजह से हम किसी Script Element में किसी Text File को भी Specify कर सकते हैं, लेकिन शर्त बस ये है कि उस External File का चाहे जो भी Extension हो, उसमें Code हमेंशा JavaScript के ही होने चाहिए।

उदाहरण के लिए यदि हम हमारे उपरोक्त Example Code में **myJS.js** File का नाम बदलकर निम्नानुसार **myJS.txt** कर दें, तो भी हमारा Web Browser इस File के Code को भी पिछले Example की तरह ही Run करेगा और हमें दिखाई देने वाले Output में किसी तरह को कोई फर्क नहीं आएगा।

```
<!-- index.html -->
<!DOCTYPE html>
<html>

<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <title>My JavaScript</title>
    <script src="myJS.txt"></script>
</head>

<body>
</body>

</html>

//myJS.txt
alert("I am from External JavaScript File.");
```

जब हम किसी <script> Element में किसी External Source File को Specify करते हैं, तब हमें उस Script Element के बीच किसी तरह का कोई JavaScript Code Specify नहीं करना चाहिए। जैसेः

```
<script src="myJS.txt">
    alert("Hi, It's not right way to write Inline JavaScript Code");
</script>
```

उपरोक्त Script Code में हमने एक External JavaScript Code File को Specify किया है साथ ही इस Element के बीच Inline JavaScript Code भी Specify किया, जो कि गलत है। यदि हमें Inline Code Specify करना ही हो, तो हमें ये काम निम्नानुसार करना होता हैः

```
<script src="myJS.txt"></script>
<script>
    alert("Hi, It's not right way to write Inline JavaScript Code");
</script>
```

JavaScript की एक सबसे बडी विशेषता ये है कि हम जिस तरह से किसी External Domain से किसी Image को अपनी Web Browser में Import कर सकते हैं, उसी तरह से हम किसी अन्य Domain पर Specified JavaScript Codes को भी अपनी Web Site के लिए Use कर सकते हैं और जब हम ऐसा करते हैं, तब उस External Domain पर Specified External JavaScript उसी तरह से हमारे Web Page के लिए Execute होता है, जिस तरह हमारे स्वयं के Domain पर Specified JavaScript File के Codes Executable होते हैं।

उदाहरण के लिए यदि हम Google के CDN पर Places **jQuery** की JavaScript File को अपने Web Page में उपयोग में लेना चाहें, तो हमें केवल निम्नानुसार <script> Element में **src** Attribute को jQuery की JavaScript File के Path से Associate करना होगा और बिना उस JavaScript File को अपने Web Server पर Place किए हुए, हम उस JavaScript File के सभी Functions को अपने Web Page के लिए Use कर सकते हैंः

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.3/jquery.min.js">
```

```
</script>
```

अब सबसे महत्वपूर्ण बात और वो ये है कि हालांकि हम किसी भी HTML Document में Page Level Inline JavaScript को Use कर सकते हैं, लेकिन हमें ऐसा नहीं करना चाहिए। चाहे एक ही Line का JavaScript Code क्यों न हो, हमें हमेंशा एक External JavaScript File Create करनी चाहिए और उस External File में ही अपने JavaScript Codes को लिखना चाहिए।

External File के बहुत सारे फायदे हैं। सबसे पहली बात तो ये है कि जिस तरह से HTML के Structure व Style को एक दूसरे के साथ Mix नहीं करना चाहिए उसी तरह से HTML के Structure व Behavior को भी एक दूसरे के साथ Mix नहीं करना चाहिए।

यानी एक Web Page के सभी Front Parts (HTML, CSS, JavaScript) आदि को अलग–अलग ही रखना चाहिए। ऐसा करने का मुख्य फायदा ये है कि हम हमारी Web Site को ज्यादा बेहतर तरीके से Maintain कर सकते हैं। हमारा हमारी Web Site पर ज्यादा Control होता है और हम JavaScript File के किसी भी एक Function में Change या Modification करके उसका प्रभाव Web Site के सभी Web Pages पर Apply कर सकते हैं, जबकि Internal JavaScript Codes होने की स्थिति में हमें एक–एक Web Page को Open करके उसमें Modification करना होगा।

चूंकि JavaScript File, Images व CSS File की तरह Web Browser में Cache होती हैं, इसलिए जब पहली पर हमारी Web Site को Open किया जाता है, तभी हमारी सभी JavaScript Files, User के Web Browser में Save हो जाती हैं और जब User हमारी Web Site के अन्य Pages को Open करता है, तब वे Web Pages काफी तेजी से Open होते हैं क्योंकि उन Web Pages से Associated JavaScript File पहले से ही Web Browser में मौजूद होती हैं। यानी External JavaScript Files Use करने पर हमारी Web Site ज्यादा Fast हो जाती है।

जबकि Inline JavaScript का प्रयोग करने पर हर Web Page के Load होने पर हर बार JavaScript Code भी Web Browser में Load होता है, भले ही सभी JavaScript Codes समान ही क्यों न हो। इस स्थिति में Inline JavaScript Codes को Use करने पर Web Page की Speed Slow हो जाती है।

तीसरा एक और फायदा ये है कि Inline JavaScript Use करने पर हमें जरूरत के अनुसार विभिन्न प्रकार के Symbol Encodings को Use करना पडता है। उदाहरण के लिए JavaScript Codes में “<” या “>” इन Comparison Symbols को यदि Inline JavaScript में Use करना पडे, तो हमें इनके स्थान पर **“<”** या **“>”** Codes का प्रयोग करना पडता है। अथवा हमें **CDATA Element** का प्रयोग करना पडता है, जबकि External Files में ऐसी कोई Restrictions नहीं होती हैं।

<noscript> Element

कई बार किन्हीं कारणों से User के Web Browser में JavaScript **Disabled** रहता है। इस स्थिति में हम JavaScript के जो भी Codes लिखते हैं, वे Codes अपना कोई Effect प्रदर्शित नहीं करते।

ऐसे में हम इस Element का प्रयोग करके User को इस बात की जानकारी दे सकते हैं कि उसका Web Browser JavaScript को Support नहीं कर रहा है इसलिए या तो वह अपने Web Browser में JavaScript को Enable करे अथवा किसी दूसरे Web Browser में Web Site को Open करे।

इस जरूरत को हम सामान्यतः HTML के <noscript> Element का प्रयोग करके पूरा करते हैं। <noscript> Element को Web Page में निम्नानुसार Use किया जाता हैः

```
<!-- index.html -->
<!DOCTYPE html>
<html>

<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <title>My JavaScript</title>
    <script src="myJS.txt"></script>
</head>

<body>
    <noscript>
        <p> This Web Page requires JavaScript Enabled Web Browser. </p>
    </noscript>
</body>

</html>
```

<noscript> Element को हमेंशा Body Element के बीच ही Use किया जाता है और इस Element में हम किसी भी HTML Element को Use कर सकते हैं। इस Element के बीच Enclosed Content सिर्फ और सिर्फ दो स्थितियों में ही User को दिखाई देते हैंः

1. जब User का Web Browser JavaScript को Support नहीं करता।
2. जब User के Web Browser में JavaScript को Disabled किया गया होता है।

किसी भी अन्य स्थिति में User को <noscript> Element के बीच Enclosed Content दिखाई नहीं देता।

Object Oriented Programming System Fundamental

JavaScript एक Object Oriented Programming Language है। इसलिए Object Oriented Programming System के Fundamental को समझे बिना हम JavaScript को बेहतर तरीके से नहीं समझ सकते। इसलिए पहले हम Object Oriented Programming System के Basic Elements को समझने की कोशिश करेंगे, जो कि निम्नानुसार हैंः

1. Object (Method and Property)
2. Class
3. Encapsulation
4. Aggregation and Composition
5. Reusability or Inheritance
6. Polymorphism

Objects

दुनियां की किसी भी वस्तु (Physical or Logical) को हम **Object** मान सकते हैं। Object वास्तव में किसी चीज का एक Representation होता है और इस Representation को किसी Programming Language की मदद से Express किया जाता है। Object कुछ भी हो सकता है।

ये कोई Physical **Car** हो सकता है अथवा कोई Logical **Bank A/c** हो सकता है। यानी हम किसी भी चीज को एक Object मान सकते हैं।

हर Object Basically दो मूल भूत चीजों का बना होता है, जिन्हें *Properties* व *Methods* के नाम से जाना जाता है। यानी दुनियां के हर Object की कुछ **Characteristics** हो सकती हैं, जिन्हें उसकी Appearance व Stat के रूप में जाना जाता है। जैसे *Height, Width, Length, Color, Name* आदि।

जबकि दुनियां कर हर Object किसी न किसी तरह का Action Perform करता है। Object द्वारा Perform किए जा सकने वाले Actions को Object का **Method** कहते हैं। जैसे *उठना, बैठना, चलना, गायब होना* आदि।

यदि हम Object को एक Analogy द्वारा समझने की कोशिश करें, तो Book एक **Object** है। Book के Pages की संख्या, Author का नाम, Book की Price आदि Book की Properties हैं और Book को खरीदा जा सकता है, बेचा जा सकता है, आदि उस Book के **Methods** हो सकते हैं।

यदि Programming Language की भाषा में समझें तो Window, Menubar, Toolbar, Button आदि सभी **Objects** के उदाहरण हैं। इन सभी Object की कुछ न कुछ Properties हैं। उदाहरण के लिए Button एक Object है और Button पर दिखाई देने वाला नाम, उस Object की एक **Property** है। जबकि Button को Click करने पर Button पर दिखाई देने वाले नाम का Change हो जाना, Button का एक **Method** है जबकि **Click** होना एक Operating System **Event** है।

Class

दुनियां के सभी Object किसी न किसी एक Group से संबंधित होते हैं। यानी हम दुनियां के सभी Objects को Categorized कर सकते हैं। उदाहरण के लिए दो पैरों पर चलने वाले जीवों को हम **Human Being Class** का Object मान सकते हैं जबकि चार पैरों वाले जीवों को हम **Animal Class** का Object मान सकते हैं।

यानी Class वास्तव में एक Blueprint या Description या Prototype या Modal होता है और उस Prototype या Modal को Follow करने वाली सभी चीजें उस Class का Object होते हैं।

Object का दूसरा नाम Instance भी है, इसलिए यदि हम चाहें तो ऐसा भी कह सकते हैं कि **Rohan** नाम का व्यक्ति **Human Being** Class का Object है क्योंकि उसमें Human Being Class की Description के सारे Features हैं। या फिर हम Rohan को **Human Being** Class का **Instance** भी कह सकते हैं।

Class वास्तव में एक प्रकार का Description या Blueprint मात्र होता है। इसलिए एक बार Class Define कर लेने के बाद हम उस Class के जितने चाहें उतने Object या Instance Create कर सकते हैं। ठीक उसी तरह से जिस तरह से हम किसी घर का Blueprint Create कर लेने के बाद उस Blueprint के आधार पर जितने चाहें उतने एक समान घर Create कर सकते हैं।

चूंकि Object Oriented Programming System केवल एक Concept है और इस Concept को जिस Programming Language में Implement किया जाता है, उस Programming Language को Object Oriented Programming Language कहा जाता है। लेकिन इसका

मतलब ये नहीं है कि सभी Object Oriented Programming Languages एक समान OOPS Pattern को Follow करें।

इसीलिए JavaScript जिस Object Oriented Programming Pattern को Follow करता है, उसे **Prototypes Pattern** के नाम से जाना जाता है और इस Pattern में ठीक उसी तरह की **Class** Create नहीं होती है, जैसी Classical Object Oriented Programming Languages **C++** व **Java** में होती है। बल्कि JavaScript में हर Object किसी दूसरे Object के आधार पर बनता है क्योंकि एक Object के सभी Features बनने वाले हर नए Object में होते हैं।

इसलिए जिस Object के आधार पर नया Object Create होता है, उस मूल Object को सभी अन्य Object के **Prototype** के रूप में Represent किया जाता है।

यानी C++ या Java जैसी Languages में हम एक Class Create करते हैं और फिर उस Class का Instance Create करते हैं, जो कि Object को Represent करता है। जबकि JavaScript में हम एक Object Create करते हैं और फिर उस Object का एक नया Instance Create करते हैं जो कि ठीक वैसा ही Object होता है, जैसा पहला वाला Object था। यानी JavaScript में एक Object किसी दूसरे Object के लिए Blueprint या Description या Class या **Prototype** या Modal का काम करता है।

Encapsulation

ये OOPS का एक ऐसा Concept है जिसमें इस बात को Represent किया जाता है कि एक Object वास्तव में **Properties** व **Methods** के Combination का एक Unit होता है। यानी दुनियां का कोई भी Object ऐसा नहीं हो सकता, जिसके केवल Characteristics हों और Methods न हों। यानी Object की Appearance व Stat हो लेकिन वह Object कुछ काम न करता हो।

सरल शब्दों में कहें तो दुनिया के हर Object की कुछ न कुछ Properties होती हैं और हर Object कुछ न कुछ काम करता है। Object की Properties को हम Object का **Data** कह सकते हैं जबकि Object अपने Data पर जिन Operations को Perform कर सकता है, उन Operations को हम Object का **Method** कह सकते हैं।

चूंकि कोई भी Object हमेंशा Properties व Methods यानी Data व Data पर Perform होने वाले Operations दोनों का एक Combined Unit होता है। इसलिए Programming Language में भी Object की Properties व Methods को एक Unit के रूप में Define किया जाता है।

किसी Object की Properties व Methods को एक Unit के रूप में Define करने की प्रक्रिया को **Encapsulation** कहा जाता है। यानी Encapsulation को Programming Term के रूप में हम निम्नानुसार Specify कर सकते हैं:

1. **Data** (Stored in Properties)
2. Operations to Perform on Data (**Methods**)

OOPS Based Object Oriented Programming Languages में Encapsulation करने का मूल उद्देश्य **Data Hiding** करना होता है, ताकि किसी Object के **Data** पर केवल उसी Object के **Methods** Operation Perform कर सकें।

Aggregation or Composition

जब बहुत सारे Objects को आपस में जरूरत के अनुसार Combine करके एक नया Object Create किया जाता है, तो इस प्रक्रिया को Object Oriented Programming System में **Aggregation** या **Composition** कहा जाता है।

Aggregation एक ऐसी प्रक्रिया है, जिसमें किसी समस्या से संबंधित विभिन्न Objects को कई छोटे–छोटे Objects में Divide कर दिया जाता है, ताकि उन्हें Manage व Develop करना आसान रहे। फिर उन सभी Objects को आपस में Combine करके Problem को बेहतर तरीके से Solve किया जाता है।

उदाहरण के लिए एक Computer System बहुत सारे Units जैसे कि Keyboard, Mouse, Monitor, Micro Processor, RAM, Motherboard आदि का बना हुआ Complex Unit है। हालांकि हम Computer System को एक Single Object के रूप में Identify करते हैं। लेकिन Internally ये बहुत सारे अन्य छोटे–छोटे Objects का Combination होता है। यही है Aggregation या Composition, जिसमें बहुत सारे स्वतंत्र Unit या Objects आपस में मिलकर एक ज्यादा Complex Object Define करने में सक्षम होते हैं।

Inheritance or Reusability

Inheritance एक ऐसी प्रक्रिया है, जिसको Use करके हम समान प्रकार के Codes बार–बार Create करने के बजाय उन्हें एक बार Create करके बार–बार Reuse करने में सक्षम हो पाते हैं।

उदाहरण के लिए यदि हम एक Watch Object Create करने के लिए Code लिखते हैं, जिसमें केवल Hour व Minutes को Handle किया जाता है और भविष्य में हमें ऐसे Watch Object की जरूरत पडती है, जिसमें Hour व Minutes के साथ Seconds को भी Handle करना है।

तो हमें पूरा Code फिर से लिखने की जरूरत नहीं होती है। बल्कि Hour व Minutes को Handle करने की Functionality को हम पिछले Codes को ज्यों का त्यों Reuse करते हुए प्राप्त कर लेते हैं और हमें केवल Seconds को Manage करने के लिए ही नया Code लिखने की जरूरत पडती है।

इस प्रकार के Coding Pattern को प्राप्त करने की सुविधा हमें Object Oriented Programming System के **Inheritance** या **Reusability** Concept से प्राप्त होती है।

Classical Object Oriented Programming Language में हम Class को Inherit करके ये सुविधा प्राप्त करते हैं। लेकिन चूंकि JavaScript **Prototype** Pattern Based Object Oriented Programming Language है, इसलिए इसमें ये सुविधा प्राप्त करने के लिए हमें एक Object को किसी दूसरे Object से Inherit करना पडता है।

Polymorphism

"***Single Statement Multiple Form***" Polymorphism को Represent करने का One Line Statement है। इस Concept के अन्तर्गत विभिन्न प्रकार की Classes के Objects के लिए समान Methods को Call किया जाता है। लेकिन सभी Objects के लिए उनकी Class के Methods Call होते हैं।

यानी Program में Create किए जाने वाले विभिन्न Class के Objects के लिए Call किए जाने वाले Methods का नाम तो समान होता है। लेकिन जब Object के लिए Dot Operator का प्रयोग करते हुए समान नाम के Method को Call किया जाता है, तो Object जिस Class का होता है, उस Object के लिए उसी Class का Method Call होता है न कि किसी दूसरे Object की Class का।

इस प्रकार से एक ही Program Code Statement अलग–अलग Object के लिए अलग–अलग परिस्थिति में अलग–अलग Method को Call करता है। इस प्रक्रिया को **Polymorphism** कहा जाता है।

यदि हम उपरोक्त सभी Concepts को एक सामान्य उदाहरण द्वारा समझने की कोशिश करें, तो इस प्रकार से समझ सकते हैं कि मानलो :

Rahul एक व्यक्ति (Object) है।	***Object***
Rahul का Date of Birth 10 Jan 1980, रंग गोरा, वजन 60KG है।	***Properties***
Rahul चल सकता है, बात कर सकता है, सो सकता है।	***Methods***
Rahul **Programmer** Class का एक Instance है।	***Class Pattern (in Classical OOP)***
Rahul **Programmer** Object पर आधारित दूसरा Object है। यानी जो Rohit है वही Rahul है, जबकि Rohit एक Programmer है, इसलिए Rahul भी Programmer है।	***Prototype Pattern (in Prototype OOP)***
Rahul की एक Date of Birth (Data) हैं, जिसके आधार पर वह अपनी उम्र Calculate (Method) करता है।	***Encapsulation***
Rahul अपनी उम्र कैसे Calculate करता है, इसकी जानकारी Rahul के अलावा किसी को नहीं है, क्योंकि Rahul की Date of Birth केवल Rahul को ही पता है।	***Data Hiding***
Rahul **Web Development Team** Object का हिस्सा है, जिसमें Rajesh और Mukesh भी काम करते हैं।	***Aggregation Composition***
Rahul, Rajesh and Mukesh तीनो च्मतेवद व्इरमबज पर आधारित हैं।	***Inheritance***
Rahul:Talk, Rajesh:Talk व Mukesh:Talk के रूप में हम तीन अलग Person Object के लिए **Talk** नाम का समान Method Call कर सकते हैं। लेकिन हम जिस Object के साथ इस Method को Call करते हैं, वह Object Talk यानी बात करता है।	Polymorphism Method Overriding

उपरोक्त सभी Concepts, Object Oriented Programming System के Concepts हैं। यदि उपरोक्त सारांश सारणी से भी आपको Object Oriented Programming System का Basic Concepts ठीक से समझ में न आए हों, तो इन Concepts को बेहतर तरीके से समझने के लिए आप हमारी अन्य पुस्तकें "**C++ Programming Language in Hindi**" व "**Java Programming Language in Hindi**" को पढ सकते हैं। इन दोनों पुस्तकों में Object Oriented Programming System को बहुत ही Detail में समझाया गया है।

BOM

BROWSER OBJECT MODEL

BOM – THE BROWSER OBJECT MODEL

हालांकि हर Web Browser में एक JavaScript Engine होता है, जो JavaScript Codes को Process व Interpret करता है, फिर भी हमारा JavaScript Program कभी भी Web Browser से Directly Interact नहीं कर सकता, बल्कि JavaScript हमें Objects का एक ऐसा Collection Provide करता है, जो कि हमारे JavaScript Program व Web Browser के बीच Intermediary की तरह काम करते हैं।

यानी हमें हमारे Web Browser के साथ जब भी कोई Interaction करना होता है, तो हम हमारे Web Browser को JavaScript द्वारा Directly Access नहीं करते बल्कि हम JavaScript द्वारा Provided किसी Appropriate Object के साथ Interaction करते हैं और वह Object अपने Web Browser के साथ Interact करता है। Objects के इसी Collection को **BOM** या **Browser Object Model** कहा जाता है, जिनका मुख्य Objective, Web Browser व JavaScript के बीच एक Simple व Consistent Interface Provide करना मात्र होता है।

जैसाकि हमने पहले भी कहा है कि JavaScript मूल रूप से तीन हिस्सों का बना हुआ है। पहला हिस्सा Core ECMAScript का है, दूसरा हिस्सा DOM का है व तीसरा हिस्सा BOM का है और BOM, JavaScript का वह हिस्सा है, जो JavaScript को Web Browser के साथ Interact करने की सुविधा प्रदान करता है।

लेकिन एक JavaScript Developer के लिए BOM ही सबसे ज्यादा परेशानी पैदा करने वाला हिस्सा भी है क्योंकि BOM का कोई Standard बनने से पहले ही Web Browsers बनाने वाली Companies ने अपने Web Browsers बना दिए थे और सभी Companies ने अपने Web Browser को अपनी सुविधानुसार अलग–अलग Patterns व Technology का प्रयोग करते हुए Develop किया था। जिसका परिणाम ये हुआ कि एक Standard Web Browser (BOM) कैसा होना चाहिए, इस विषय में कोई Standard बनने से पहले ही BOM बन चुका था।

अतः Web Browsers (BOM) बनने के बाद में BOM का Standard Develop नहीं किया जा सकता था। क्योंकि ऐसा करने पर सभी Companies को अपने पुराने Web Browsers को फिर से पूरी तरह से Change करना पडता, जो कि सम्भव नहीं था। इसलिए BOM (Web Browsers) का कोई Specific Standard नहीं बन पाया।

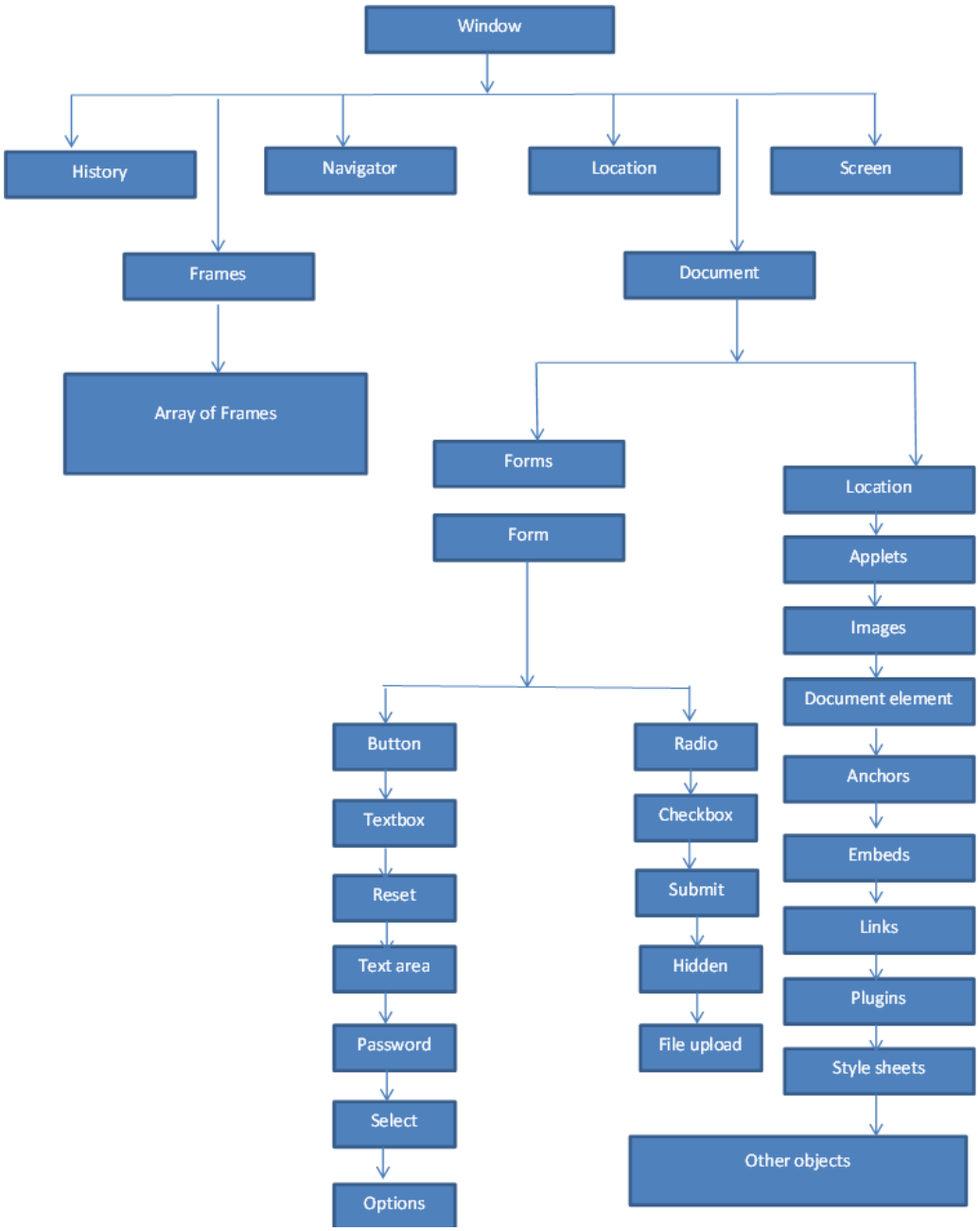
परिणामस्वरूप आज भी विभिन्न Web Browsers के BOM आपस में पूरी तरह से एक दूसरे के Compatible नहीं हैं। जिसकी वजह से जब हम JavaScript का प्रयोग करते हुए Web Browser यानी BOM से Interact करते हैं, तब अलग–अलग Web Browsers में समान Functionality प्राप्त करने के लिए भी हमें अलग–अलग प्रकार के Codes लिखने पडते हैं।

हम Web Browser को ही BOM (Browser Object Model) भी कह सकते हैं। यानी BOM व Web Browser दो अलग चीजें नहीं हैं बल्कि एक ही चीज के दो अलग नाम हैं। इसलिए पुस्तक के इस Section में हम विभिन्न Companies द्वारा Develop किए गए Web Browsers के Model के Common Objects के बारे में जानेंगे।

हालांकि विभिन्न Companies ने अपने Web Browser में बहुत सारे ऐसे Features Add किए हैं, जो कि केवल उसी Company के Web Browser में Available हैं। लेकिन हम उन Specific Features के बारे में चर्चा नहीं करेंगे क्योंकि हम जब भी कोई Web Site Create करते हैं, तो हम यही चाहते हैं कि उस Web Site का हर Web Page सभी Web Browsers में एक समान दिखाई दे व एक समान रूप से Behave करे।

इसलिए यदि हम किसी Particular Company के Web Browser के Specific Features को Use करते हुए अपने Web Page में कोई Functionality Add करेंगे, तो वह Functionality केवल उसी Web Browser में Reflect होगी अन्य में नहीं।

हालांकि लगभग सभी Web Browsers को अलग–अलग Companies, Organizations या Individuals ने Create किया है, लेकिन फिर भी सभी Web Browsers का BOM कुछ हद तक समान हैं, जिन्हें हम निम्नानुसार एक Tree Structure के रूप में Represent कर सकते हैं:



जैसाकि हम उपरोक्त चित्र में देख सकते हैं कि **window** Object पूरे BOM यानी Web Browser का Top Level Object है। दूसरे शब्दों में कहें, तो *window* Object हमारे Current Web Browser के Window को Represent करता है। यानी हम हमारे Web Browser में

JavaScript के माध्यम से जो भी Functionality प्राप्त करना चाहते हैं, वे सभी Functionality हमें *window* Object के माध्यम से ही प्राप्त होती हैं क्योंकि *window* Object किसी भी Web Browser का Root Object होता है।

window Object किसी भी Web Browser के किसी Instance को Represent करता है। यानी यदि हम एक ही Web Browser में कई Tabbed Windows Open करें, तो हर Tabbed Window एक **window** Instance होता है।

window Object किसी Web Browser में दो काम करता है। पहला Web Browser के साथ **JavaScript Interface** की तरह और दूसरा Core JavaScript यानी ECMAScript **Global Object** की तरह। यानी किसी Web Page में Define किया गया हर Object, Variable व Function, Web Browser के इस **window** Object को एक Global Object की तरह Use करता है। दूसरे शब्दों में कहें, तो Global Object को हम Program का **Global Scope** भी कह सकते हैं।

Global Scope

यानी जब हम हमारे JavaScript Program में कोई ऐसा Object, Variable या Function Create करते हैं, जो कि Current Web Page के किसी भी हिस्से के लिए Accessible रहता है, तो इस प्रकार के Variable, Object या Function को Global Variable, Global Object या Global Function कहा जा सकता है और इस प्रकार के सभी Variables, Objects व Functions, **window** Object की ***Properties*** व ***Methods*** की तरह window Object से Associate हो जाते हैं।

उदाहरण के लिए JavaScript में जब भी हम कोई Variable Create करना चाहते हैं, तो हम **var** Keyword का प्रयोग करते हुए निम्न Statement Use करते हैं:

```
var age = 31;
```

यदि हम किसी Variable को किसी Function या Object की Body के अन्दर Declare करने के अलावा, पूरी JavaScript File में कहीं भी Declare करते हैं, तो वह Variable एक प्रकार से **Global Variable** की तरह Declare होता है, जिसे Current Web Page में कहीं भी उपयोग में लिया जा सकता है।

वह Global Variable, Current Web Page के किसी भी हिस्से में लिखे गए JavaScript Code के लिए उपलब्ध रहता है, क्योंकि वास्तव में वह Global Variable हमेंशा **window** Object की Property बन जाता है। इसलिए जब तक Current Window Memory में Loaded रहता है तब तक वह Variable भी Window में Loaded Current Web Page के लिए Globally Available रहता है।

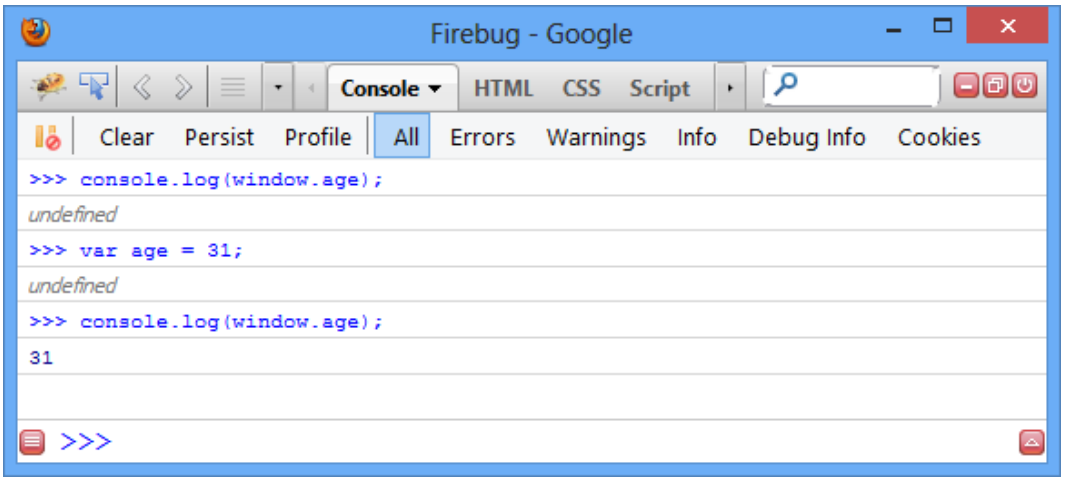
चूंकि एक Object वास्तव में **Properties** व **Methods** का Encapsulated Unit होता है, इसलिए Object की Property व Method को Use करने के लिए हमें Dot (.) Operator को Use करना पडता है।

ठीक इसी प्रकार से **window** भी एक Object है और यदि हम ये जानना चाहते हैं कि उपरोक्त Statement के माध्यम से हमारे द्वारा Create किया गया **age** नाम का Variable, *window* Object की Property बना या नहीं, तो इस बात की जानकारी प्राप्त करने के लिए हम *window*

Object के साथ Dot Operator का प्रयोग करते हुए age नाम के Variable को Specify कर सकते हैं।

यानी हम **window.age** Statement को *console.log()* में Specify करके इस बात का पता लगा सकते हैं कि *window* Object की ***age*** Property में कोई मान है या नहीं

यदि ***age*** नाम का Variable वास्तव में *window* Object की Property बना होगा, तो उसमें Stored मान *console.log()* Statement द्वारा Output में Display हो जाएगा, लेकिन यदि ऐसा नहीं हुआ होगा, तो *console.log()* Method Output के रूप में "*undefined*" शब्द Display करेगा, जो इस बात का Signal होता है कि हम *window* Object की किसी ऐसी Property को Access करने की कोशिश कर रहे हैं, जो कि Exist नहीं है। Global Scope की इस Functionality को समझने के लिए हम Firebug Console में निम्न Code लिख सकते हैं:



उपरोक्त चित्र में हमने सबसे पहले *console.log(window.age);* Statement लिखकर ये जानने की कोशिश की है कि age नाम की Property पहले से window Object के लिए Available है या नहीं। और जैसाकि आप देख सकते हैं कि इस Statement के Run होने पर हमें Return Value के रूप में "***undefined***" मान प्राप्त होता है, जो इस बात का Signal है कि window Object के लिए age नाम की Property पहले से Exist नहीं है।

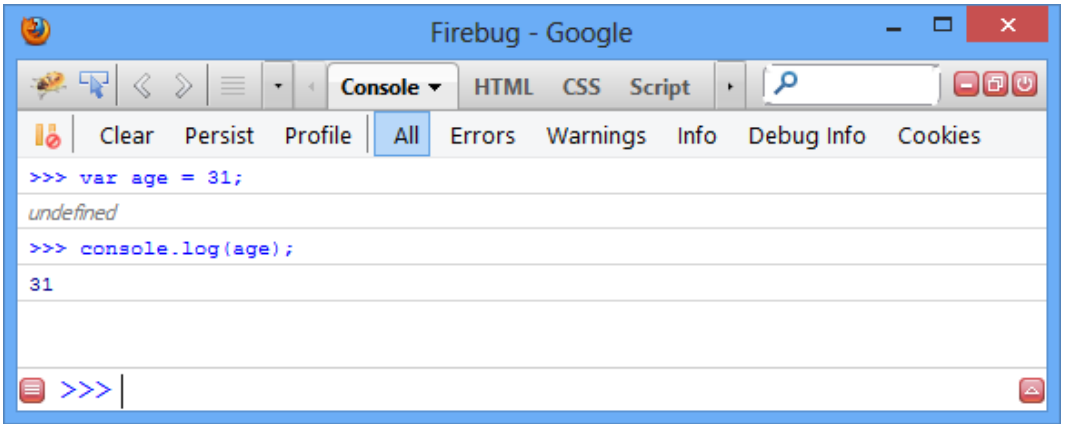
फिर हमने *var age = 31;* Statement द्वारा age नाम का एक नया Variable Create किया है। चूंकि age नाम का Variable भी पहले से Exist नहीं है, इसलिए ये Statement Execute होने के बाद भी हमें Return Value के रूप में "*undefined*" मान प्राप्त होता है।

अन्त में हमने फिर से *console.log(window.age);* Statement लिखकर ये जानने की कोशिश की है कि age नाम की Property पहले से window Object के लिए Available है या नहीं। और जैसाकि आप देख सकते हैं कि इस बार हमें Output के रूप में मान 31 प्राप्त हो रहा है, जो age नाम के हमारे द्वारा Create किए गए Variable का मान है।

इस मान का Return होना इसी बात को Indicate कर रहा है कि अब ***age*** नाम का Variable, *window* Object की Property बन गया है। अन्यथा ये Statement Execute होने पर हमें फिर से "***undefined***" मान प्राप्त होता।

इस प्रकार से ये बात साबित होती है कि हम JavaScript में जितने भी Variables, Objects व Functions Create करते हैं, वे सभी *window* Object की *Properties* व *Methods* बनते हैं और Web Browser में Currently Loaded Web Page के लिए Globally Available रहते हैं।

ऐसा नहीं है कि हम जिन भी Variables, Objects या Functions को Create करते हैं, उन सभी को ***window.propertyName*** या ***window.methodName*** Statement द्वारा ही Access कर सकते हैं। यदि हम चाहें तो age नाम के Variable को निम्नानुसार बिना *window* Object Specify किए हुए भी Use कर सकते हैं:



जैसाकि आप उपरोक्त चित्र में आप देख सकते हैं कि यदि हम *console.log(age);* Statement Use करते हैं, तो बिना *window* Object को Specify किए हुए भी हम ***age*** नाम के Variable की Value को प्राप्त कर सकते हैं।

ऐसा इसलिए होता है, क्योंकि *window* Object एक Global Object को Represent करता है और Global Scope के सभी Variables, Properties व Methods पूरे JavaScript Program में Available रहते हैं। इसीलिए हम ***age*** नाम के Variable को Directly भी Access कर सकते हैं और *window* Object की Property के रूप में भी Access कर सकते हैं।

ध्यान रखने वाली बात ये है कि JavaScript के कई ऐसे Objects हैं जो Global होते हैं, जैसे ***location, navigator, history, screen, document*** आदि। लेकिन वास्तव में ये सभी **window** Object की Properties हैं, जिनके बारे में हम आगे विस्तार से जानेंगे।

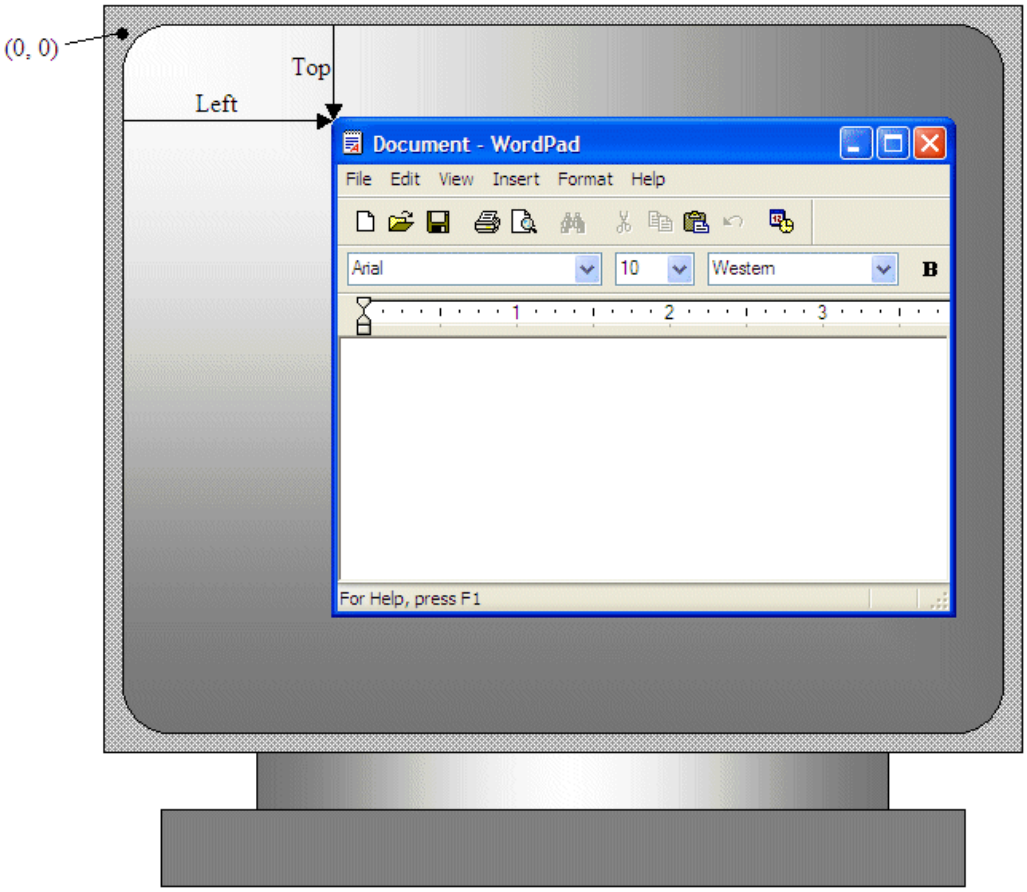
जैसाकि हमने पहले भी कहा कि विभिन्न Companies ने अपने BOM को अलग–अलग तरीकों से Develop किया है, इसलिए सभी Web Browsers एक समान काम नहीं करते। अतः Windows Mobile के लिए Develop किया गया Internet Explorer Web Browser अपने **window** Object के साथ *window.property = value;* Statement द्वारा किसी नए Property या Method को Directly Create करने की सुविधा नहीं देता। हालांकि Globally Declare किए जाने वाले सभी Variables व Functions, इसमें भी *window* Object की Properties व Methods बन जाते हैं।

Window Position

किसी Web Browser के Window की Position को विभिन्न window Properties व Methods द्वारा Determine व Change किया जा सकता है। FireFox को छोडकर सभी Web Browsers

screenLeft व ***screenTop*** नाम की दो Properties द्वारा Current Web Browser के Window की Position Return करते हैं जबकि FireFox में यही Positions ***screenX*** व ***screenY*** Properties द्वारा Return करता है।

हमारे Computer System का जो Monitor होता है, उसे JavaScript में **screen** Object द्वारा Represent किया जाता है, जबकि हमारा Web Browser Screen पर जितने Area में दिखाई देता है, वह Area Web Browser का Window होता है। इसे हम निम्न चित्र द्वारा बेहतर तरीके से समझ सकते हैं:



इस प्रकार से यदि हमें ये जानना हो कि हमारा Web Browser हमारे Screen की Left Position से कितने Pixel Right में Placed है, तो हम निम्न Code लिखकर इस बात का पता लगा सकते हैं:

For Firefox

```
>>> window.screenX                    //Output: 407
>>> window.screenY                    //Output: 60
```

For IE, Safari, Chrome, Opera

```
>>> window.screenLeft                 //Output: 407
>>> window.screenTop                  //Output: 60
```

ये Statements Console में Execute करते समय यदि Web Browser Maximized हो, तो Return Value के रूप में हमें 0 ही प्राप्त होता है, क्योंकि Maximized होने पर Web Browser का Window, **Screen** के **Top** व **Left** से 0 Pixel की दूरी पर होता है।

चूंकि विभिन्न Web Browsers को अलग–अलग Companies ने अलग–अलग Patterns के आधार पर Develop किया है, इसलिए सभी Web Browsers में Window के **Top** व **Left** Coordinates को हम कभी भी Accurately Determine नहीं कर सकते।

फिर भी हम किसी Window को ***moveTo()*** व ***moveBy()*** window Methods का प्रयोग करके Web Browser के Window को किसी Exact Coordinate पर Move जरूर कर सकते हैं।

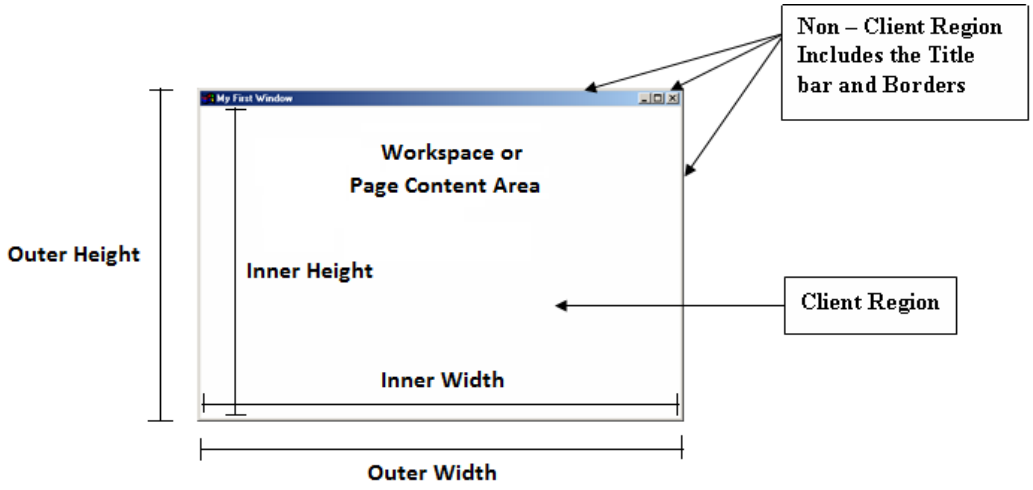
ये दोनों ही Methods Parameters के रूप में दो Arguments Accept करते हैं, जो कि क्रमशः ***x*** व ***y*** Coordinate को Represent करते हैं। सामान्यतः सभी Web Browsers में ये दोनों **window** Methods By Default Disabled रहते हैं।

Window Size

इससे पहले कि हम Web Browser के Window की Size से संबंधित Properties के बारे में बात करें, पहले हमें किसी भी GUI Window की Anatomy या Structure से सम्बंधित विभिन्न Terms को समझना जरूरी है, ताकि हम Window के Size से सम्बंधित विभिन्न Properties को समझ सकें।

किसी भी GUI Operating System में दिखाई देने वाले Window के मूल रूप से दो हिस्से होते हैं, जिन्हें **Client Region** व **Non-Client** Region कहा जाता है। Client Region को सामान्यतः Workspace या Page Content Area कहते हैं। यह वही Area होता है, जहां User को Web Page दिखाई देता है। जबकि Non-Client Area के अन्तर्गत Title Bar व Borders आते हैं। Non-Client Area वह Area होता है, जहां User किसी भी तरह का कोई Interaction नहीं कर सकता।

Title Bar व Border यानी सभी Non-Client Region सहित Window की जो Height होती है, उसे Window की Actual **Height** या **Outer Height** कहते हैं जबकि Border व Title Bar यानी Non-Client Region को घटाने के बाद जो Height बचती है, उसे **Inner Height** कहते हैं।



इसी तरह से Left व Right की Border की Width सहित Window की जो Width होती है, उसे Window की Actual **Width** या **Outer Width** कहते हैं जबकि Left व Right Side की Border को घटाने के बाद जो बचता है, उसे Inner Width कहते हैं।

Page Content Area की Height व *Inner Height* तथा Page Container Area की Width व *Inner Width* में केवल इतना अन्तर होता है कि Page Content Area में Border Included रहता है, जबकि Inner Height व Inner Width में Border Included नहीं होता।

किसी भी Window की Size का Accurate तरीके से पता लगाना भी पूरी तरह से सभी Web Browsers में सम्भव नहीं है, क्योंकि अलग–अलग Web Browsers अलग–अलग तरीके से Window की Size Return करते हैं। फिर भी Firefox, Safari, Opera और Chrome *innerWidth, innerHeight, outerWidth* व *outerHeight* नाम की चार Properties Provide करते हैं, जिनका प्रयोग JavaScript के माध्यम से Window की Size ज्ञात करने के लिए किया जा सकता है।

Firefox व Safari में **outerWidth** व **outerHeight** Properties, **Browser Window** का Dimension Return करते हैं।

जबकि Opera में **outerWidth** व **outerHeight** Properties Browser Window के **Page View Container** का Dimension Return करते हैं जबकि *innerHeight* व *innerWidth* Properties, Page View Area को Return करते है, जिसमें Border Included नहीं होता।

Google Chrome में *outerWidth* व *outerHeight* Properties Viewport की Size Return करते हैं जो कि वास्तव में **innerWidth** व **innerHeight** Properties Return होने वाले मान के समान ही होता है।

यानी Google Chrome में *outerWidth* व *outerHeight* Properties Browser Window के Dimension Return नहीं करते बल्कि Page View Area या Page Content Area की Height व Width को ही Return करते हैं।

इसी तरह से Internet Explorer अपने Browser Window की कोई Information अपने window Object द्वारा Return नहीं करता, बल्कि Web Browser में Loaded Page के Viewable Area की Information DOM के माध्यम से Return करता है।

जबकि *document.documentElement.clientWidth* व *document.documentElement .clientHeight* ये दोनों Properties सभी Web Browsers के Page Viewport यानी Client Region की **Height** व **Width** Provide करते हैं।

हालांकि IE6 में ये दोनों Properties तभी काम करती हैं, जबकि वह Standard Mode में हो। यदि IE6 Quirks Mode में हो, तो इन Properties के स्थान पर *document.body.clientWidth* व *document.body.clientHeight* Properties को Use करना पडता है।

इसी तरह से Google Chrome जब Quirks Mode में होता है, तो इन Information को प्राप्त करने के लिए **clientWidth** व **clientHeight** Property को *document.body* तथा *document.documentElement* दोनों Properties के साथ उपयोग में लिया जा सकता है।

सारांश के रूप में कहें तो यहां भी विभिन्न Web Browser के Window की Size को Determine करने का भी कोई Standard तरीका नहीं है।

Intervals and Timeouts

JavaScript एक Single Threaded Language है। यानी इसमें एक बार में केवल एक ही Code को Interpret किया जा सकता है, लेकिन ये हमें **Timeouts** व **Intervals** दो ऐसी सुविधाऐं प्रदान करता है, जिनका प्रयोग करके हम अपने Codes के Execution की Scheduling कर सकते हैं।

Timeouts ऐसी सुविधा है, जिसमें हमारा JavaScript Code Specify किए गए समय के बाद Automatically Execute हो जाता है। जबकि Intervals सुविधा द्वारा हम किसी Code को निश्चित समयावधि के Interval पर बार–बार Execute कर सकते हैं।

किसी Code को एक निश्चित समयावधि के बाद Automatically Execute करने के लिए हम Web Browser के window Object के **setTimeout()** Method को उपयोग में ले सकते हैं। ये Method Parameter के रूप में दो Arguments Accept करता है, जहां पहला Argument वह Code होता है, जिसे Execute होना है, जबकि दूसरा Argument Milliseconds में वह समयावधि होता है, जितने समय के बाद पहले Argument के रूप में Specify किए गए Code को Execute होना है। इस Method को हम निम्नानुसार उपयोग में ले सकते हैं:

```
>>> setTimeout("console.log('Hello')", 3000);
```

जब इस Code को Console Window में Type करके Run किया जाता है, तो Console Window में तुरन्त ही "**Hello**" Print नहीं हो जाता, बल्कि इस Code को Run करने के 3 Second बाद "Hello" Print होता है। यानी ***setTimeout()*** Method द्वारा हम Execute होने वाले Code की Time Scheduling कर सकते हैं।

यदि हम उपरोक्त Code को ही निम्नानुसार Modify करते हुए *setTimeout()* Method के स्थान पर *setInterval()* Method को Specify कर दें व दूसरे Parameter के रूप में 1000 Specify कर दें:

```
>>> setInterval("console.log('Hello')", 1000);
```

तो इस Code को Execute करने पर ये Code Console Window में हर एक Second के बाद "Hello" Message Print करता है और तब तक इसी Code को बार–बार Run करता रहता है, जब तक कि हम Current Web Page को Reload न कर दें अथवा Web Browser को Close न कर दें।

सामान्यतः इन दोनों Methods को Use करके JavaScript में विभिन्न प्रकार के Animations Create किए जाते हैं।

एक बार किसी Code को Run करने के लिए *setTimeout()* Method का प्रयोग करके यदि Schedule कर दिया जाए और फिर किसी कारणवश यदि हमें किसी विशेष परिस्थिति में उस Code को Execute होने से पहले ही रोकना हो, तो हम *setTimeout()* Method को Cancel करने के लिए **clearTimeout()** Method को Use कर सकते हैं। जबकि **setInterval()** Method के Execution को रोकने के लिए हमें **clearInterval()** Method को Use करना होता है।

setTimeout() व *setInterval()* दोनों ही Methods Execute होते समय अपना एक Unique ID Return करते हैं। इस ID को किसी Variable में Store करके उन IDs को ही *clearTimeout()* या *clearInterval()* Method में Parameter के रूप में Pass किया जाता है। ये दोनों Methods

उन Codes को Execute होने से रोक देते हैं, जिनका ID इनमें Parameter के रूप में Pass किया गया होता है।

setTimeout() व *setInterval()* Methods को Use करते समय हमें पहले Parameter के रूप में Executable Code को एक String के रूप में Specify करने के स्थान पर एक Anonymous Function के रूप में Specify करना चाहिए। यानी यदि हम उपरोक्त Code को ही उपयोग में लेना चाहें, तो हमें वही Code निम्नानुसार लिखना चाहिए–

```
>>> setTimeout(function(){
        console.log('Hello')
    }, 3000);
```

```
>>> setInterval(function(){
        console.log('Hello')
    }, 1000);
```

हालांकि उपरोक्त दोनों Codes पिछले वाले दोनों Codes की तुलना में काफी अच्छे हैं, लेकिन फिर भी ये दोनों ही Codes एक बार Execute होने के लिए Schedule हो गए, तो इन्हें किसी भी स्थिति में रोका नहीं जा सकता।

क्योंकि ये दोनों ही Methods जब Execute होते हैं, तो अपने Code का एक Unique ID Return करते हैं और उस Unique ID को जब तक हम किसी Variable में Store न करें, तब तक हम इन Codes को *clearTimeout()* या *clearInterval()* Method में Parameter के रूप में Pass करके इन्हें रोक नहीं सकते। इसलिए वास्तव में हमें उपरोक्त Code को भी निम्नानुसार लिखना चाहिए:

```
>>> var id1 = setTimeout(function(){
        console.log('Hello')
    }, 3000);
```

```
>>> var id2 = setInterval(function(){
        console.log('Hello')
    }, 1000);
```

अब ये Codes पूरी तरह से Controllable हैं। यदि किसी विशिष्ट परिस्थिति में हमें उपरोक्त Codes को Execute होने से रोकना हो तो हम ये काम निम्नानुसार **clearTimeout()** व **clearInterval()** Methods का प्रयोग करके कर सकते हैं:

```
>>> setTimeout(function(){
        console.log('Hello')
    }, 3000);

    clearTimeout(id1);
```

```
>>> setInterval(function(){
        console.log('Hello')
    }, 1000);

    clearInterval(id2);
```

setTimeout() व ***setInterval()*** Functions को **DOM Styles** व **DOM Events** के साथ Property Use करके ही JavaScript के माध्यम से Web Pages में विभिन्न प्रकार के Animations Define किए जाते हैं।

लेकिन Pure JavaScript का प्रयोग करते हुए विभिन्न प्रकार के Animations Create करने के लिए बहुत सारी बातों का ध्यान रखना होता है क्योंकि सभी DOM Methods को सभी Web Browsers समान रूप से Support नहीं करते।

इसलिए सामान्यतः सामान्य प्रकार की ज्यादातर जरूरतों को पूरा करने के लिए हम Pure JavaScript का प्रयोग करने के स्थान पर बडी ही आसानी से ***jQuery, DOJO, YUI*** जैसे JavaScript Library यानी JavaScript Frameworks का प्रयोग कर सकते हैं क्योंकि इन Frameworks को Pure JavaScript में ही Design किया गया है और ये काफी Mature Frameworks हैं।

इसलिए इन Frameworks को Use करके हम वास्तव में अपने समय की बचत ही रहे होते हैं। क्योंकि जिन कामों को हम Pure JavaScript का प्रयोग करने के लिए अपने स्वयं के Cross-Browser Functions Create करेंगे, उन Functions को इन Libraries के रूप में पहले से ही Define किया जा चुका है, जिन्हें हम बिना किसी परेशानी के बडी ही आसानी से Reuse कर सकते हैं।

लेकिन इसका मतलब ये नहीं है कि हमें JavaScript को ठीक से समझने की जरूरत नहीं है। वास्तव में ये सभी Frameworks JavaScript पर ही आधारित हैं, इसलिए यदि हम JavaScript के विभिन्न Fundamentals को समझ लें, तो हमें इन Frameworks की Internal Working को समझने में काफी मदद मिलती है, जिसकी वजह से हम इन Frameworks को ज्यादा बेहतर तरीके से Use करने में सक्षम हो जाते हैं।

System Dialog Boxes

Web Browsers **alert(), confirm()** व **input()** Methods द्वारा तीन System Dialogs Display करने में सक्षम है। ये Dialogs Web Browser में Loaded Web Page से संबंधित नहीं होते और न ही इनमें किसी तरह का कोई HTML Code होता है। साथ ही इनकी Appearance पूरी तरह से Operating System व Web Browser की Settings पर निर्भर होती है।

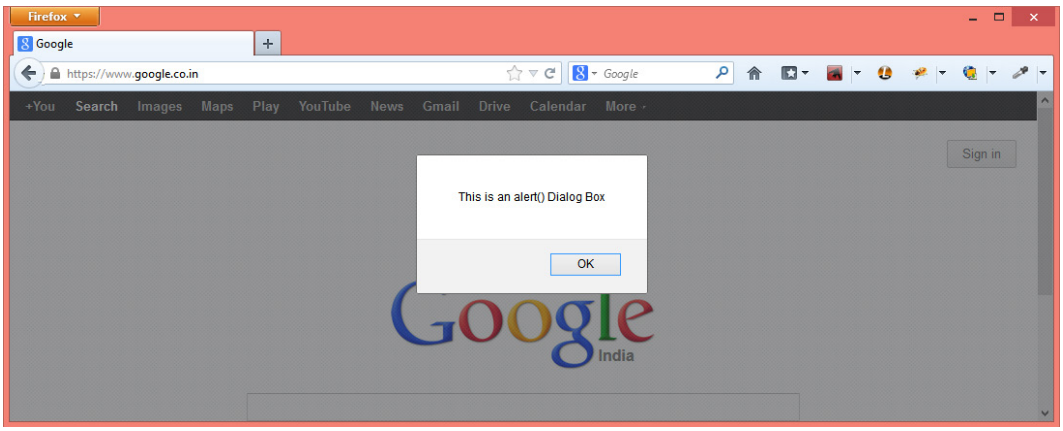
ये तीनों ही Dialog Box Synchronous व Modal हैं यानी जब ये Memory में Load होते हैं, तो इनके बाद लिखे गए सारे Codes का Execution तब तक के लिए रूक जाता है, जब तक कि हम इन्हें Close नहीं कर देते है।

alert() Method – Alert Dialog Box

alert() Method का प्रयोग किसी Specific Message को Display करने के लिए किया जा सकता है। सामान्यतः JavaScript Codes के Flow को समझने या उन Codes की Debugging करने के लिए हम ***alert()*** Method द्वारा Alert Dialog Box को एक Pause की तरह Use करते हैं। सामान्यतः किसी JavaScript Error को Alert Box द्वारा Render किया जाता है। जैसेः

```
>>> alert("This is an alert() Dialog Box");
```

जैसे ही हम इस Code को Execute करते हैं, हमें इसका Output Web Browser में निम्नानुसार दिखाई देने लगता हैः



Alert Dialog Box पर केवल एक "**OK**" Button होता है। यानी हम इस Dialog Box को Web Browser से हटाने के लिए OK Button पर Click करने के अलावा और कुछ नहीं कर सकते।

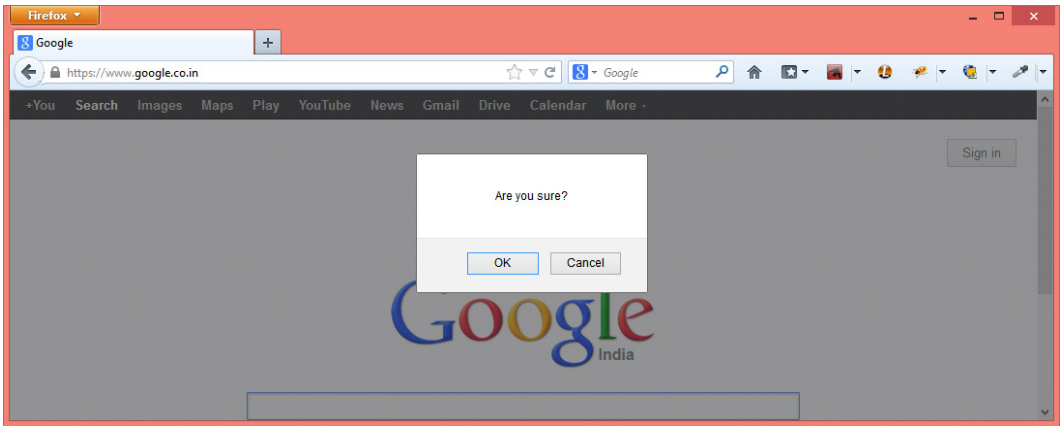
confirm() Method – Confirm Dialog Box

ये एक दूसरे प्रकार का Dialog Box है लेकिन इस Dialog Box में हमें "OK" व "Cancel" नाम के दो Buttons प्राप्त होते हैं। ये Dialog Box ***true*** या ***false*** में से किसी एक Value को भी Return करता है।

यानी यदि हम इस Dialog Box के **OK** Button पर Click करते हैं, तो ये Dialog Box ***true*** Value Return करता है, जबकि **Cancel** Button पर Click करने पर ये Dialog Box ***false*** Value Return करता है। इसे हम निम्नानुसार उपयोग में ले सकते हैं:

```
>>> confirm("Are you sure?");          //Output: true
>>> confirm("Are you sure?");          //Output: false
```

ये Dialog Box Screen पर निम्नानुसार दिखाई देता है:



prompt() Method – Input Dialog Box

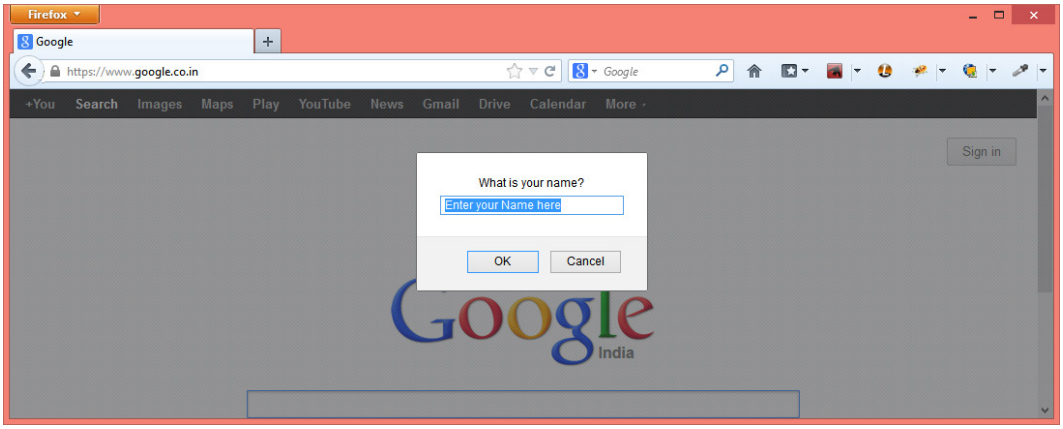
ये Dialog Box एक Special प्रकार का Dialog Box है, जिसे हम User से किसी प्रकार का Input प्राप्त करने के लिए Use कर सकते हैं। इस Dialog Box पर भी **OK** व **Cancel** नाम के

दो Buttons दिखाई देते हैं साथ ही एक Text Box भी दिखाई देता है, जिसमें User किसी तरह का Data Input कर सकता है।

जब User इस Box पर दिखाई देने वाले Text Box में कोई Text Input करके **OK** Button पर Click करता है, तो ये Method Input किए गए Text को Return करता है, जबकि यदि User इस Dialog Box के **Cancel** Button को Click करता है, तो ये Method ***null*** Value Return करता है।

```
>>> prompt("What is your name?", "Enter your Name here");
"Enter your Name here"
```

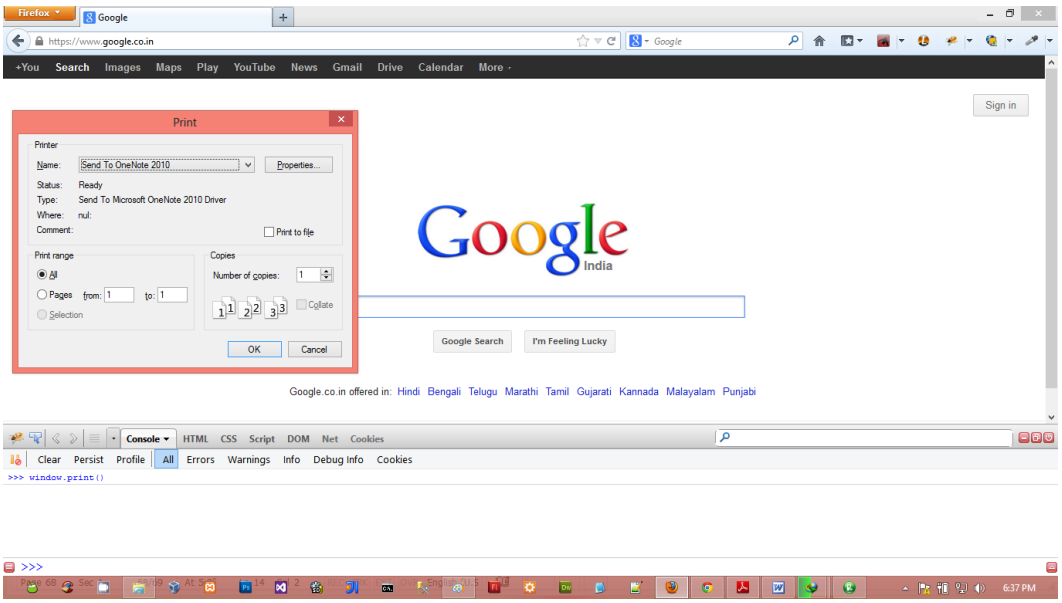
जब ये Code Run होता है, तो हमें Web Browser में निम्नानुसार Prompt Dialog Box दिखाई देता है:



इनके अलावा BOM का **window** Object हमें **find()** व **print()** नाम के दो और Method Provide करता है, जिनका प्रयोग हम Currently Loaded Web Page किसी Content को खोजने अथवा Current Web Page को Printer पर Print करने के लिए कर सकते हैं। इन्हें हम निम्नानुसार Use कर सकते हैं:

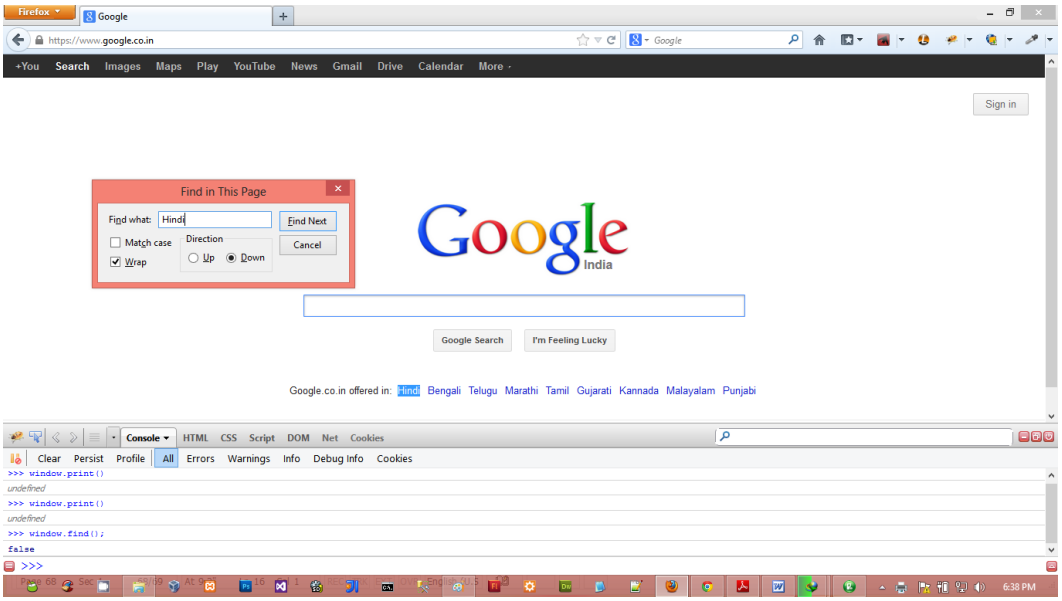
```
>>> window.print();
      //OR
>>> print();
```

ये Method Web Page पर Print Dialog Box Open करता है, जिसे हम निम्न चित्र में देख सकते हैं:



इसी तरह से हम **find()** Method को भी निम्नानुसार Console Window द्वारा Open कर सकते हैं:

```
>>> window.find();
      //OR
>>> find();
```



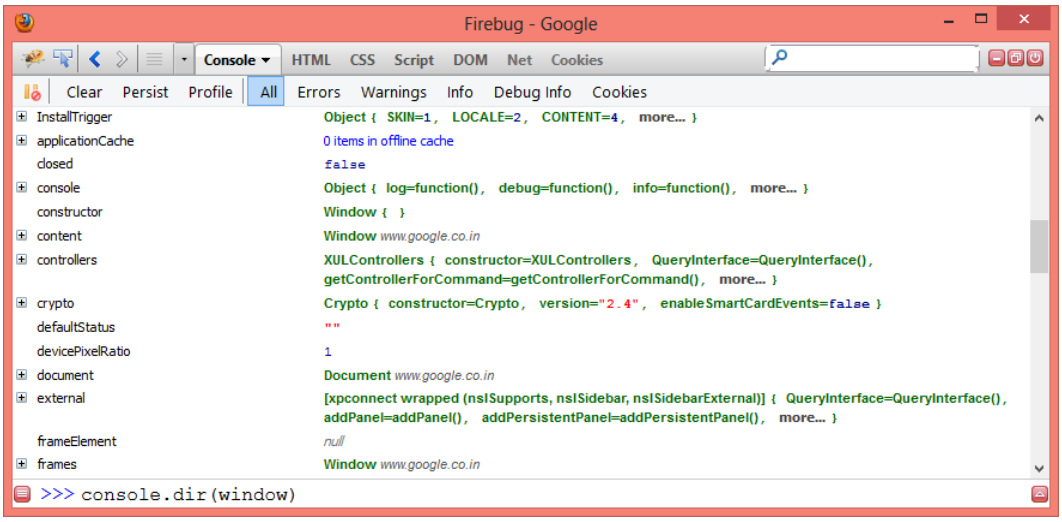
Web Browser के *window* Object की और भी बहुत सारी Properties हैं जिनमें से कुछ सभी Web Browsers के लिए Common हैं जबकि कुछ विभिन्न Web Browsers के लिए अलग–अलग हैं। Web Browser के *window* Object की सभी Properties व Methods की List देखने के लिए हम Firebug के *console.dir()* Method का प्रयोग कर सकते हैं।

इस Method में हम Web Browser के जिस Object को Parameter के रूप में Pass करते हैं, ये Method उस Object के सभी Properties व Methods की List Display कर देता है।

उदाहरण के लिए यदि हम window Object के सभी Properties व Methods की List देखना चाहें, तो हम Firebug Console में निम्न *console.dir()* Method को निम्नानुसार Use कर सकते हैं:

```
console.dir(window)
```

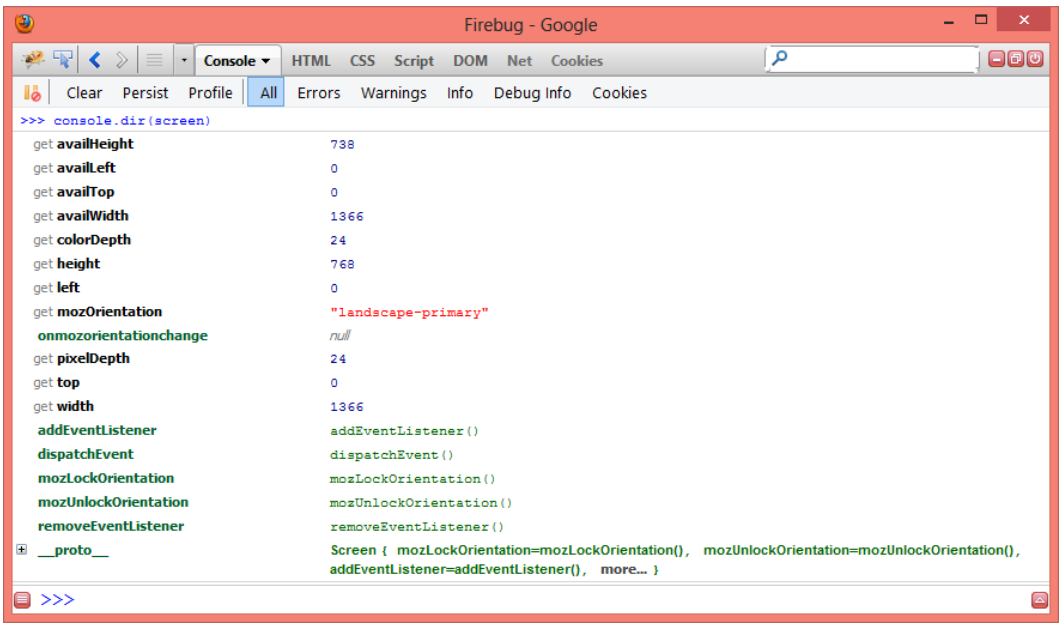
इस Statement के Execute होने पर हमें Web Browser के window Object के सभी Properties व Methods की List निम्नानुसार दिखाई देने लगती है:



इसी तरह से यदि हम **window** Object के Sub-Object **screen** के सभी Properties व Methods की List प्राप्त करना चाहें, तो *console.dir()* Method को निम्नानुसार Use कर सकते हैं:

```
console.dir(screen)
```

और हमें निम्नानुसार Output प्राप्त होता है:



Location Object

location Object, BOM का एक और बहुत ही महत्वपूर्ण Object है, जो हमें Web Browser के **window** Object में Currently Loaded Web Page से संबंधित बहुत सारी महत्वपूर्ण Information व Navigation Functionality Provide करता है। **location** Object की विभिन्न Properties व Methods को हम Firebug द्वारा निम्नानुसार *console.dir(location);* Statement द्वारा Display कर सकते हैं:

location Object इसलिए भी Special व Unique Object है क्योंकि ये *window* व *document* दोनों Object की Property है। यानी हम इसे Point करने के लिए **window.location** भी लिख सकते हैं और **document.location** भी लिख सकते हैं।

जैसाकि उपरोक्त चित्र में हम देख सकते हैं कि इस Object की विभिन्न Properties में Currently Loaded Web Page से संबंधित विभिन्न प्रकार की Special Information हैं, जिन्हें हम हमारी जरूरत के अनुसार अपने Web Page में उपयोग में ले सकते हैं। **location** Object की विभिन्न Properties को हम निम्नानुसार समझ सकते हैं:

hash Property

Currently Loaded Web Page के URL में यदि कोई #String हो, तो वह String इस Property में Store होती है। ये एक प्रकार का On-Page Anchor होता है। यानी जब हम कोई ऐसा Hyperlink Create करते हैं, जिसे Click करने पर हम उसी Page के किसी अन्य हिस्से पर पहुंच जाते हैं, तो उस अन्य हिस्से की Position को Identify करने के लिए किसी Element के **id** Attribute में हम जो String Specify करते हैं, वही String **hash** Property में Store होती है।

उदाहरण के लिए User जब *http://*www.bccfalna.com/index.html#CProgramming URL पर Click करता है, तो वह *index.html* Page के उस Element पर पहुंचता है, जिसके **id** Attribute में Value के रूप में "***#CProgramming***" String को Value के रूप में Specify किया गया है।

```
>>> location.hash                //Output: #CProgramming
```

host Property

इस Property में Currently Loaded Web Page के Host का नाम व यदि उपलब्ध हो, तो Port Number भी Stored रहता है। जैसेः

```
>>> location.host                //Output: www.google.co.in
                                 //Output: www.google.com:8080
```

hostname Property

इस Property में Currently Loaded Web Page के Host का नाम बिना Port Number के Stored रहता है। जैसेः

```
>>> location.hostname            //Output: www.google.co.in
```

pathname Property

इस Property में Currently Loaded Web Page का Path Stored रहता है। यदि Web Browser में केवल Root Level Domain को Specify किया गया हो, तो इस Property में **"/"** Stored रहता है। जैसेः

```
>>> location.pathname            //Output: www.google.co.in
                                 //Output: www.google.com:8080
```

port Property

इस Property में Currently Loaded Web Page के URL का केवल Port Number Stored रहता है। यदि ULR के साथ कोई Port Number Specified न हो, तो ये Empty String को Hold करता है। जैसेः

```
>>> location.port                //Output: ""
                                 //Output: "8080"
```

protocol Property

इस Property में Currently Loaded Web Page को Web Browser में Load करने के लिए Use होने वाले Protocol की जानकारी होती है। सामान्यतः इसमें "**http:**" या "**https**:" ही Stored रहता है। जैसेः

```
>>> location.protocol                    //Output: https
```

search Property

इस Property में Currently Loaded Web Page के साथ यदि कोई Query String हो, तो वह Query String Store होती है। Query String हमेंशा "**?**" Mark के बाद का URL होता है और Query String URL तब बनता है, जब हम HTML Form से किसी Data को Web Server पर Process होने के लिए **GET** Method का प्रयोग करते हुए SEND करते हैं। जैसेः

```
>>> location.search                      //Output: ?name=Kuldeep&age=31
```

assign() Method

location Object का प्रयोग करके हम कई तरीकों से Web Browser की Location को Change कर सकते हैं। यानी *location* Object में हम जो भी URL Specify कर देते हैं, Web Browser उसी URL के Resource को Web Browser के window Object में Load करने लगता है। *location* Object में किसी Location को Set करने के लिए हम JavaScript के **assign()** Method को निम्नानुसार नेम कर सकते हैंः

```
>>> location.assign("http://www.bccfalna.com/");
```

जैसे ही ये Statement Execute होता है, Web Browser में वह Web Server Load होने लगता है, जिसे हमने **assign()** Method में Argument के रूप में Specify किया है।

यदि हम चाहें, तो निम्न Statements द्वारा सीधे ही Web Browser में नया Web Page Load कर सकते हैंः

```
>>> window.location = "http://www.bccfalna.com/";
>>> window.href = "http://www.bccfalna.com/";
```

यदि हम उपरोक्त दोनों में से किसी भी JavaScript Statement को Interpret करें, तो ये दोनों Statements Internally **location** Object के **assign()** Method को Call करते हैं और Web Browser के **window** Object में http://www.bccfalna.com/ Web Page को Load कर देते हैं और जैसे ही नया Web Page, Current Web Browser के window Object में Load होता है, **location** Object की विभिन्न Properties (*hash, search, host, pathname, port, etc…*) नए URL की विभिन्न Values से Fill हो जाती हैं।

location Object की *hash* Property के अलावा यदि कोई भी अन्य Property की Value Change होती है, तो Web Browser का Web Page को फिर से Reload करता है।

replace() Method

उपरोक्त में से किसी भी तरीके को Use करके यदि **location** Object के URL में किसी भी तरह का Change किया जाता है, तो BOM के **history** Object में उस URL की एक Entry हो जाती है। ताकि जरूरत होने पर User Web Browser के Back Button को Click करके फिर से पिछले Page पर जा सके।

लेकिन यदि हम चाहें, तो BOM के इस Default Behavior को **location** Object के **replace()** Method का प्रयोग करके Change कर सकते हैं। ये Method उस URL को Argument के रूप में Accept करता है, जिसे Web Browser के *window* Object में Load करना है, लेकिन **history** Object में किसी प्रकार की कोई Entry नहीं करता।

replace() Method को Call करने के बाद हम Web Browser के Back Button को Use करके पिछले URL पर फिर से नहीं जा सकते। इस Functionality को हम Firebug Console में निम्न Statement Run करके Check कर सकते हैं:

```
>>> window.replace("http://www.bccfalna.com/");
```

reload() Method

reload() Method, location Object का अन्तिम Method है, जो Current Web Page को Web Browser के window Object में फिर से Reload कर देता है। जब इस Method को बिना किसी Argument के Call किया जाता है, तो यदि वह Web Page, Web Browser के Cache में Exist हो, तो वहीं से Load होता है।

जबकि यदि इस हम Web Page को फिर से Web Server से प्राप्त करना चाहते हैं, तो हमें इस Method में Parameter के रूप में "**true**" Value को Specify करना होता है। इस Method को हम दोनों तरीकों से निम्नानुसार लिख सकते हैं:

```
>>> window.reload();              // Reloads possible from cache.
>>> window.reload(true);          // Reloads back from the server.
```

जब किसी JavaScript Program में एक बार **reload()** Method Execute हो जाता है, तो फिर उस Program में उस Statement से आगे लिखा गया कोई भी Statement Execute नहीं होता।

navigator Object

इस Object को सबसे पहले Netscape Navigator 2.0 में Develop किया गया था, जिसे बाद में आने वाले लगभग सभी Web Browsers में एक Standard Object की तरह Implement किया गया।

ये Object, Current Web Browser के **Identification** से संबंधित Information को Hold करता है। हालांकि कुछ अन्य Web Browsers इस Requirement को पूरा करने के लिए अन्य Alternative या Similar Ways Provide करते हैं। उदाहरण के लिए IE में **window.clientInformation** तथा Opera में **window.opera** Object वही काम करते हैं जो अन्य Web Browsers में **navigator** Object करता है।

navigator Object लगभग सभी JavaScript Enabled Web Browsers में एक समान काम करता है। लेकिन अन्य BOM Objects की तरह ही हर Web Browser के **navigator** Object में Current Web Browser Specific Properties का Set भी होता है। इसलिए इस पुस्तक के इस Section में हम केवल उन्हीं *Properties* व *Methods* के बारे में जानेंगे, जो लगभग सभी Web Browsers के ***navigator*** Object में समान रूप से उपलब्ध हैं।

appCodeName Property

ये Web Browser के नाम को Represent करता है। सामान्यतः इसमें Value के रूप में "**Mozilla**" ही Stored रहता है, भले ही वह Browser Mom-Mozilla Browsers ही क्यों न हो।

appName Property

ये Web Browser के Full Name को Represent करता है।

appVersion Property

ये Web Browser के Version को Represent करता है। हालांकि ये Actual Web Browser Version से सम्बंधित नहीं होता।

cookieEnabled Property

यदि Current Web Browser में Cookie Enabled हो, तो इसमें ***true*** Value Stored रहता है जबकि Disabled होने की स्थिति में इसमें ***false*** Value होता है।

javaEnabled() Method

यदि Current Web Browser में Java Enabled हो, तो इसमें ***true*** Value Stored रहता है जबकि Disabled होने की स्थिति में इसमें ***false*** Value होता है।

mimeTypes Property

ये एक Array होता है, जिसमें Current Web Browser में Registered सभी Supported MIME Types की Information होती है।

onLine Property

ये Boolean Property है, जिसमें ***true*** या ***false*** Value Stored रहता है। यदि Web Browser Use करने वाला User Internet से Connected हो, तो इस Property में ***true*** Store होता है। जबकि यदि User Offline Mode में Web Browser Use कर रहा हो, तो इस Property में ***false*** Stored रहता है।

platform Property

इस Property में Current Web Browser के Operating System या System Platform की Information Stored रहती है।

Plugins Property

ये Property एक Array होता है, जिसमें Current Web Browser में Installed सभी Plug-ins की Information Stored रहती है। IE में इस Array में Current Page पर उपलब्ध सभी **<embed>** Elements की Information Stored रहती है।

userAgent Property

ये Property Current Web Browser के User Agent की Information को एक String के रूप में Store करके रखता है।

screen Object

screen Object भी BOM यानी Web Browser के *window* Object की एक Property है और इसका प्रयोग मूल रूप से Client Web Browser के Screen की Capabilities का पता लगाने के लिए ही किया जाता है। ये Object Client Computer के Display या Monitor से संबंधित Information Provide करता है।

सभी अन्य Objects की तरह ही अलग–अलग Companies ने अपने Web Browser के *screen* Object में बहुत सारी Web Browser Specify Properties को Specify किया है, इसलिए यहां हम केवल उन Properties के बारे में ही बात करेंगे, जिन्हें लगभग सभी Modern Web Browsers के ***screen*** Objects Common रूप से Support करते हैं।

availHeight Property

इस Property में User के Computer के Display Screen की Web Browser के लिए Used **Height** Pixels की संख्या के रूप में Stored रहती है। चूंकि Operating System पर दिखाई देने वाला **Task Bar**, Web Browser के Window के लिए Use नहीं होता, इसलिए Screen की Full Height में से Task Bar व अन्य System Related Bars की Height घटाने के बाद जो Value बचती है, वह Value इस Property में Stored रहती है।

availWidth Property

इस Property में User के Computer के Display Screen की **Width** Pixels की संख्या के रूप में Stored रहती है। चूंकि हम अपने Operating System पर दिखाई देने वाले **Task Bar** को अपने Desktop के Left या Right में पर Vertically भी Place कर सकते हैं, उस स्थिति में Task Bar द्वारा Reserved Screen Width, Web Browser के Window के लिए Use नहीं होती, इसलिए Screen की Full Width में से Task Bar व अन्य System Related Bars की Width घटाने के बाद जो Value बचती है, वह Value इस Property में Stored रहती है।

height Property

इस Property में User के Computer के Display Screen की **Height** Pixels की संख्या के रूप में Stored रहती है।

width Property

इस Property में User के Computer के Display Screen की **width** Pixels की संख्या के रूप में Stored रहती है।

उपरोक्त दोनों **height** व **width** Properties में वास्तव में User के Computer के Resolution की Information होती है। यदि User के Computer का Display **768 X 1366** के Resolution पर काम कर रहा हो, तो **availHeight** Property में 768 व **availWidth** Property में 1366 मान Store हो जाता है।

pixelDepth Property

इस Property में User के Computer के Display Screen के Pixels को Display करने के लिए Use किए गए Bits यानी Pixel Depth की Information Stored रहती है।

history Object

User जब Web Browser Open करता है तब से लेकर जब तक User Web Browser में Surfing करता है, तब तक के Current Web Page की Navigational History को **history** Object Hold करता है।

चूंकि **history** Object, **window** Object की एक Property है, इसलिए हर Web Browser Window, Tab व Frame का अपना अलग history Object होता है क्योंकि हर Web Browser, Tab व Frame का अपना अलग window Object होता है।

इस Object का प्रयोग करके हम Current Web Page से Forward या Backward में Navigation कर सकते हैं और इसके लिए हमें Web Browser में Current Loaded Web Page के URL की Information होना जरूरी नहीं होता।

history Object द्वारा Web Browser में Forward या Backward Navigation करने के लिए ये Object हमें **go()** नाम का एक Method Provide करता है, जो कि Integer Value के रूप में एक Argument Accept करता है।

यदि ये Number Positive हो, तो Navigation Forward Direction में होता है, जबकि Negative Number होने की स्थिति में Backward Navigation होता है। इसे हम निम्नानुसार Use कर सकते हैं:

```
>>> window.go(-1);          // Go to previous page.

>>> window.go(1);           // Go to next page.

>>> window.go(3);           // Go 3 page forward.

>>> window.go(-2);          // Go 2 page back.
```

यदि Web Browser की **history** में कोई Information न हो, तो ये Method कुछ भी नहीं करता। यानी यदि User ने Web Browser Open ही किया हो, तो history Object में कोई Information नहीं होती।

go() Method के अलावा हम **back()** व **forward()** Method को Use करके भी अगले व पिछले Page पर Move कर सकते हैं।

history Object की एक **length** Property होती है, जिसमें कुल Navigate किए गए URLs की संख्या होती है। यानी यदि User ने Web Browser Open करने के बाद कुल 10 Web Pages Surf किए हों, तो इस length Property में Value के रूप में मान 10 Stored रहता है।

चूंकि विभिन्न Web Browsers के Model यानी BOM यानी को अलग–अलग Companies, Individuals या Organizations ने अपनी सुविधानुसार Develop किया है, इसलिए इनमें कोई Standard नहीं है।

फिर भी BOM से Related जिन Objects के बारे में हमने जानने की कोशिश की है, वे सभी Objects लगभग सभी Web Browsers में उपलब्ध होते हैं और थोडे–बहुत Extra Features के साथ उनके वे **Properties** व **Methods** Common रूप से Available रहते हैं, जिनके बारे में हमने इस अध्याय में जाना है।

चूंकि Web Applications के लिए Web Browser ही JavaScript का Environment Host होता है, इसलिए विभिन्न Web Browsers के BOM के Features अलग–अलग हो सकते हैं। लेकिन जब हम Web Browser के अलावा अन्य JavaScript Host Environments की बात करते हैं, जैसेकि Adobe Flash जो कि **ActionScript** Programming Language को Host करता है, तो ActionScript जैसी उन Languages में उनके Host Environments विभिन्न प्रकार के जरूरी Features Provide करते हैं।

इसीलिए यदि आप JavaScript सीखते हैं, तो Adobe Flash की Programming सीखना आपके लिए काफी आसान रहता है क्योंकि उस स्थिति में आपको Adobe Flash के केवल Fundamental Basics को ही सीखना होता है, जबकि अन्य Programming Features तो दोनों ही Languages में एक समान होने की वजह से आपको उन्हें फिर से सीखने की जरूरत नहीं रहती है।

यानी JavaScript में से यदि BOM के Features को Remove कर दिया जाए, तो जितना JavaScript बचता है, वह JavaScript व ActionScript पूरी तरह से एक समान है। जिसका मतलब ये है कि Web Browser BOM व Adobe Flash जैसे Software के Features को यदि छोड दिया जाए, तो JavaScript व ActionScript में कोई विशेष अन्तर नहीं रहता और उस स्थिति में JavaScript या ActionScript कहने के स्थान पर यदि इस Scripting Language को ECMAScript Language कहें, तो ज्यादा बेहतर होगा, क्योंकि ActionScript व JavaScript दोनों में से यदि **Adobe Flash** व **BOM** को हटा दें, तो जो बचता है, वह **ECMAScript** ही है।

Document Writing

JavaScript में किसी Web Page या Document Object में Content को Write करने के लिए हमें ***write()*** व ***writeln()*** नाम के दो Methods प्राप्त होते हैं, जो किसी Web Page के Output Stream में किसी Content को लिखने की सुविधा देते हैं। ये दोनों ही Methods Argument के रूप में एक String Accept करते हैं और उस String को ज्यों का त्यों Currently Loaded Web Page में Text की तरह Write कर देते हैं।

write() व ***writeln()*** इन दोनों Methods में केवल एक ही अन्तर है कि जब हम writeln() Method को Use करते हैं तो ये Method Automatically String के अन्त में एक New Line ले लेता है और अगला Content हमेंशा एक New Line में दिखाई देता है, जबकि ***write()*** Method

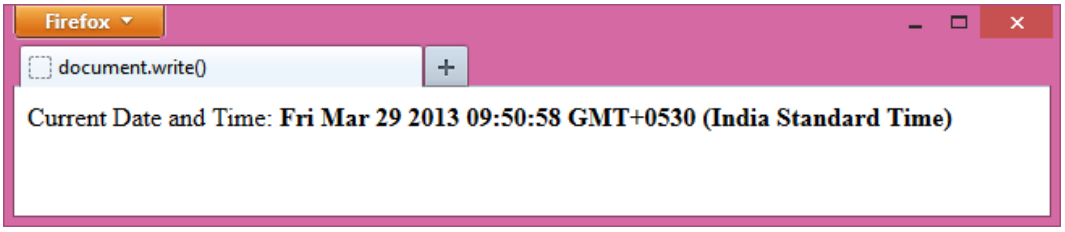
Automatically New Line नहीं लेता बल्कि यदि जरूरत हो, तो हमें स्वयं "**\n**" Constant को Use करके New Line को Specify करना पडता है।

इन दोनों Methods को Use करके हम Dynamically Load हो रहे Page में अपना Content Add कर सकते हैं। लेकिन जब हम JavaScript सीख रहे होते हैं, तब इन दोनों Methods को Use करके हम सामान्यतः अपने Programs का Output Web Page पर Render करते हैं। इन दोनों Methods को हम निम्नानुसार Use कर सकते हैं:

File Name: document.write().html

```
<!DOCTYPE html>
<html>
    <head><title>document.write()</title></head>

    <body>
        <script>
            document.write("Current Date and Time: <strong>" + Date() +"</strong>");
        </script>
    </body>
</html>
```

जब हम इस HTML Web Page को Web Browser में Load करते हैं तो ये Web Page निम्नानुसार Render होता है:



जैसाकि हम देख सकते हैं कि हमारे Web Page में कोई Static Content नहीं है, फिर भी Web Page में Current Date and Time दिखाई दे रहा है और क्योंकि Content को हमने Current Web Page में *document.write()* Method द्वारा Dynamically Insert किया है।

आप देख सकते हैं कि इस HTML Web Page में हमने ***write()*** Method में किस तरह से JavaScript के **Date()** Method को String के साथ **+** Operator द्वारा Concatenate किया है साथ ही हमने Specify किए गए Argument में **** Element को भी Specify किया है, जिसे Web Browser Parse करके दिखाई देने वाले Date and Time को Bold Face में Render कर रहा है।

यानी हम इस Method में किसी भी JavaScript Function अथवा Variable की Value को String के साथ Concatenate करके HTML Elements के बीच Enclose कर सकते हैं और Web Browser इस ***write()*** Method को Execute करते समय JavaScript Codes को भी Run करता है साथ ही सभी HTML Elements को भी Parse करके Render करता है।

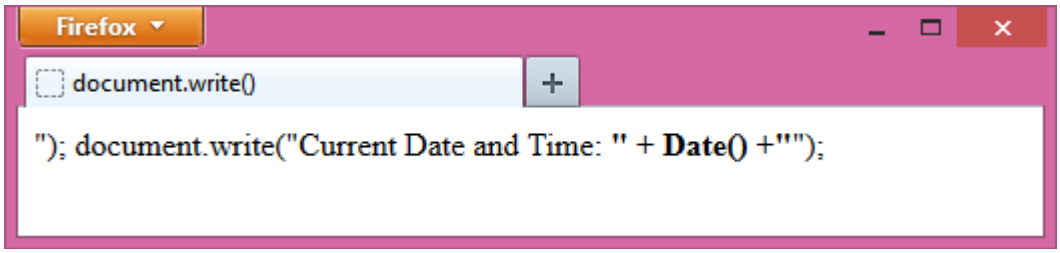
सामान्यतः ***write()*** व ***writeln()*** Methods को किसी External Resource को Dynamically Current Web Page में Include करने के लिए Use किया जाता है। जब हम इन Methods को Use करके किसी JavaScript File को अपने Web Page में Dynamically Add करना चाहते हैं, तब हमें इस बात को ध्यान में रखना होता है कि हम "**</script>**" Element को String की तरह

Specify न करें, क्योंकि ऐसा करने पर ये String Web Browser को Confuse कर देगा और Web Browser समझेगा कि हमारे Inline JavaScript Code का अन्त हो रहा है, परिणामस्वरूप ***write()*** Method के बाद के JavaScript Codes एक प्रकार से Error की तरह दिखाई देंगे। इसे हम निम्नानुसार HTML Code द्वारा समझ सकते हैं:

File Name: *document.write() with script Element.html*

```
<!DOCTYPE html>
<html>
    <head><title>document.write()</title></head>

    <body>
        <script>
            document.write("<script type=\"text/javascript\" src=\"file.js\">" + "</script>");
            document.write("Current Date and Time: <strong>" + Date() +"</strong>");
        </script>
    </body>
</html>
```

यदि हम उपरोक्तानुसार ***write()*** Method का प्रयोग करते हुए किसी External JavaScript File को **<script>** Element का प्रयोग करते हुए Current Document Dynamically में Attach करने की कोशिश करें, तो हमें निम्नानुसार Output प्राप्त होता है:



ऐसा Output इसलिए प्राप्त होता है, क्योंकि जब उपरोक्त Web Page का JavaScript Code Run होता है, तो ***write()*** Method में </script> Tag की Parsing के समय JavaScript Interpreter को लगता है कि यहीं पर हमारे Web Page के Inline JavaScript Code का अन्त हो रहा है। परिणामस्वरूप हमारा Web Browser </script> Tag के आगे के Content को Normal Content की तरह ही Web Browser में Parse कर देता है।

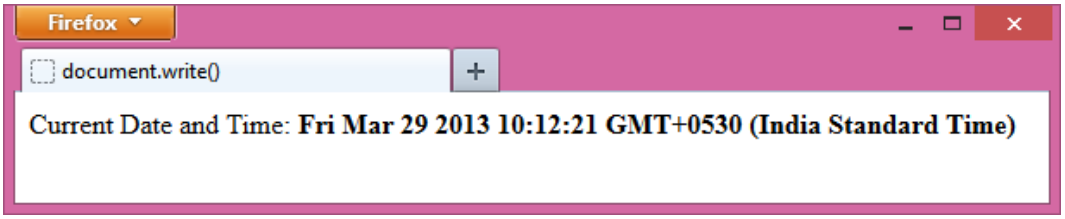
इस प्रकार की समस्या से बचने का तरीका ये है कि हम Closing </script> Tag के Slash को Backslash के साथ <\/script> तरीके से Specify किया जाए, ताकि JavaScript Interpreter उसे Closing </script> Tag की तरह Treat करते हुए Parse न करे बल्कि एक String की तरह DOM Tree में Add करे। इस Trick को Use करते हुए हम हमारे उपरोक्त Web Page को निम्नानुसार Re-Create कर सकते हैं:

File Name: *document.write() with proper script Element.html*

```
<!DOCTYPE html>
<html>
    <head><title>document.write()</title></head>

    <body>
        <script>
            document.write("<script type=\"text/javascript\" src=\"file.js\">" + "<\/script>");
```

```
            document.write("Current Date and Time: <strong>" + Date() +"</strong>");
        </script>
    </body>
</html>
```

जैसाकि निम्न Output में हम देख सकते हैं कि अब ये Code Normal तरीके से काम कर रहा है और ऐसा इसलिए हो रहा है क्योंकि हमने Closing </script> Tag को **"<\/script>"** तरीके से Specify किया है।



write() Method को यदि हम पूरा Web Page Load होने के बाद किसी Event के Response में Call करें, तो हमारा पूरा Web Page Content Overwrite हो जाता है। इसे समझने के लिए निम्न उदाहरण देखते हैं:

File Name: *document.write() before loading whole page.html*

```
<!DOCTYPE html>
<html>
    <head><title>document.write()</title></head>

    <body>
        <h1>This is heading which is normal.</h1>
        <script>
            document.write("Current Date and Time: <strong>" + Date() +"</strong>");
        </script>
    </body>
</html>
```

इस Web Page को Render करने पर हमें निम्नानुसार Output प्राप्त होता है:

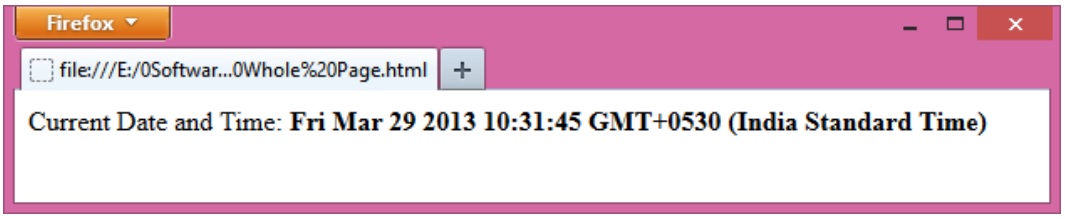
लेकिन यदि हम उपरोक्त Web Page को निम्नानुसार Modify करते हुए ***write()*** Method को पूरा Web Page Content Load होने के बाद Use करें:

File Name: *document.write() after loading whole page.html*

```
<!DOCTYPE html>
<html>
```

```
<head><title>document.write()</title></head>

<body>
    <h1>This is heading which is normal.</h1>
    <script>
        window.onload = function(){
            document.write("Current Date and Time: <strong>" + Date() +"</strong>");
        }
    </script>
</body>
</html>
```

तो जैसाकि हम निम्न Output चित्र में देख सकते हैं कि हमें केवल वही Content दिखाई दे रहा है, जो JavaScript Code द्वारा Generate हो रहा है। जो Heading Content Web Page में Static रूप से Specify किया गया था, वह दिखाई नहीं दे रहा है और ऐसा इसी वजह से होता है, क्योंकि ***write()*** Method को पूरा Web Page पूरी तरह से Load होने के बाद Call किया गया है।



इस Web Page के JavaScript Code में हमने JavaScript Event Handling Code को Use किया है। Events के बारे में विस्तार से हम आगे समझेंगे लेकिन फिर भी यहां कुछ Fundamental समझ लेना जरूरी है, ताकि Event Handling Chapter से पहले बनाए गए सभी JavaScript Codes को आप बेहतर तरीके से समझ सकें।

GUI Programming यानी Graphical User Interface Programming में जो भी Software या Application Develop किए जाते हैं, वे **Event Driven Programming** Concept पर आधारित होते हैं।

Event एक प्रकार की घटना होती है, जिसे हमारे Computer का Operating System समझता है और उस घटना के अनुसार Respond करता है।

उदाहरण के लिए जब हम किसी Software के Window के Minimize Button पर Click करते हैं, तो **click** Event Trigger होता है और इस Event के Response में हमारे Computer का Operating System उस Window को Minimize कर देता है, जिसके Minimize Button पर हमने Click किया होता है।

इसी तरह से जब हम किसी Application के किसी Form Window पर दिखाई देने वाले Text Box में कोई Character Type करते हैं, तो Keyboard का **keypress** Event Trigger होता है और इस Event के Response में हमारे Computer का Operating System उस Text Box में वह Character Display कर देता है, जिसे हमने Keyboard पर Press किया होता है।

इसी तरह से विभिन्न प्रकार के Applications में विभिन्न प्रकार के Events Trigger होते हैं, जिन्हें Operating System द्वारा Handle किया जाता है और क्योंकि Web Browser भी एक प्रकार का

Application Software ही है, इसलिए Web Browser में भी विभिन्न प्रकार के Events Trigger हो सकते हैं।

उदाहरण के लिए हम किसी Web Page पर दिखाई देने वाले किसी Hyperlink को Click कर सकते हैं और जब हम Click करते हैं, तो **click** Event Fire होता है। परिणामस्वरूप हमारा Web Browser इस Click Event के Response में उस Hyperlink से Associated Resource को Web Browser में Load कर देता है, जिस पर हमने Click किया था।

इसी प्रकार से जब हम हमारे Web Browser को Minimize, Maximize, Restore, Close, Resize आदि करते हैं अथवा Web Page के किसी Element पर Mouse से Click, Double Click करते हैं अथवा Mouse Pointer को किसी HTML Element पर Move करते हैं अथवा Keyboard से किसी Button को Press करते हैं, तो विभिन्न प्रकार के Events Trigger होते हैं।

इसलिए यदि हम चाहें, तो इन Events के Trigger होने के Response में ऐसा JavaScript Code लिख सकते हैं, जो केवल उसी स्थिति में Execute होता है, जब वह Event Trigger होता है, जिसके साथ उस JavaScript Code को Attach किया गया होता है।

इस प्रकार की Programming Technique जिसमें किसी ***Action*** के ***Reaction*** में यानी किसी ***Event*** के ***Response*** में क्या होना चाहिए, इस बात को निश्चित करते हुए Programming की जाती है, को **Event Driven Programming** कहते हैं और जब हम Web Browser में Event Driven Programming करते हैं, तब हम ऐसा JavaScript Code Create करते हैं, जो केवल तभी Run होता है, जब उस JavaScript Code से Associated Event Fire होता है।

चूंकि किसी HTML Web Page पर जितने भी Elements होते हैं, उन सभी Elements के साथ हम विभिन्न प्रकार के Mouse व Keyboard Events को Associate कर सकते हैं। इसी तरह से Web Browser के विभिन्न Objects के साथ भी हम विभिन्न प्रकार के Events को Associate कर सकते हैं।

जब हम हमारे JavaScript Code को केवल <script> Element के बीच Specify करते हैं, तो Web Page के Load होते समय जो भी JavaScript Code, JavaScript Interpreter को मिलता है, JavaScript Interpreter उसे Run कर देता है।

लेकिन जब हम हमारी सुविधानुसार किसी JavaScript Code को किसी Event के Response में तब Run करवाना चाहते हैं जब कोई Specific Event Fire होता है, तो इस जरूरत को पूरा करने के लिए हमें दो काम करने पडते हैं:

1. हमें किसी Event को उस Object के साथ Specify करना होता है, जिस पर **Event** Fire होगा।
2. Fire होने वाले Event को **Response** करने के लिए हमें एक Event Handler Function Create करना होगा।

अब हम हमारे पिछले JavaScript Program के Code को समझने की कोशिश करते हैं। चूंकि जैसाकि हम जानते हैं कि यदि हम हमारे JavaScript को निम्नानुसार लिखते:

```
document.write("Current Date and Time: <strong>" + Date() +"</strong>");
```

तो ये JavaScript Code ठीक उसी समय Execute हो जाता, जब Web Browser में Web Page Parse होकर Render होता क्योंकि Web Browser में सारे Codes ***Up to Down*** व

Left to Right Parse होते हैं। इसलिए JavaScript Interpreter जैसे ही इस JavaScript Code पर पहुंचता, वह उसे Run कर देता और Web Browser में *Current Date and Time Display* हो जाता।

लेकिन हमने हमारे पिछले Web Page में JavaScript Code को निम्नानुसार लिखा है:

```
window.onload = function(){
    document.write("Current Date and Time: <strong>" + Date() +"</strong>");
}
```

ये JavaScript Code वास्तव में Execute नहीं होता बल्कि Memory में Store हो जाता है। क्योंकि ये Code सामान्य JavaScript Code नहीं है बल्कि एक Event Handler JavaScript Function Code है जिसे Web Browser के **window** Object की **onload** Event Property के साथ Attach किया गया है जो इस बात का Signal है कि ये JavaScript Function तब Execute होगा, जब Web Browser में Current Web Page पूरी तरह से Load हो जाएगा।

जैसाकि हमने पहले कहा कि Web Browser व Web Browser में Loaded विभिन्न HTML Elements विभिन्न प्रकार के Events Trigger करते हैं। इसलिए जब किसी Web Browser में कोई Web Page पूरी तरह से Load हो जाता है, तब Web Browser "**load**" नाम का एक Event Fire करता है, जो इस बात का Signal होता है कि Current Web Page, Web Browser में पूरी तरह से Load हो चुका है।

Web Browser के **window** Object के साथ इस **load** Event का उपयोग करते हुए हम चाहते हैं कि जब Web Browser में कोई Web Page पूरी तरह से Load हो जाए, तब निम्न JavaScript Code Run हो न कि तब जब Web Page, Current Web Browser में Load हो रहा हो:

```
document.write("Current Date and Time: <strong>" + Date() +"</strong>");
```

इस जरूरत को पूरा करने के लिए हमने ***window.onload*** Property के साथ एक JavaScript Function को Attach किया है जो कि एक Event Handler Code है।

यानी हम चाहते हैं कि **Date** तब Display हो, जब Web Page पूरी तरह से Load हो चुका हो। इसलिए हमने Date Display करने से सम्बंधित *document.write()* Method को एक Event Handler Function के अन्दर Define किया है। (Function के बारे में हम आगे विस्तार से पढेंगे।)

हमारा जो Event Handler Code यहां पर हमने Define किया है, वह Code एक Anonymous Function है। जब JavaScript Interpreter इस Code को Read करता है, तो वह उसे Memory में Store कर देता है और वह Anonymous Function Memory में जिस जगह पर Store होता है, उस जगह का एक Pointer Return करता है। इस Return होने वाले Pointer को ***window.onload*** Property में Store कर दिया जाता है।

परिणामस्वरूप हमारा Web Page जैसे ही Web Browser के **window** Object में पूरी तरह से Load हो जाता है, Web Browser "**load**" नाम का Event Fire करता है। परिणामस्वरूप JavaScript Interpreter **window** Object की **onload** Property को Check करता है कि उसमें किसी Executable Code का Pointer Stored है या नहीं।

चूंकि हमने एक Anonymous Function को **window** की ***onload*** Property में Assign किया है जो कि एक Executable Code है, इसलिए JavaScript Interpreter तुरन्त उस Memory Location पर पहुंचता है, जिसका Pointer **window** के ***onload*** Property में Stored है और चूंकि इस Memory Location पर निम्नानुसार एक Executable JavaScript Statement होता है:

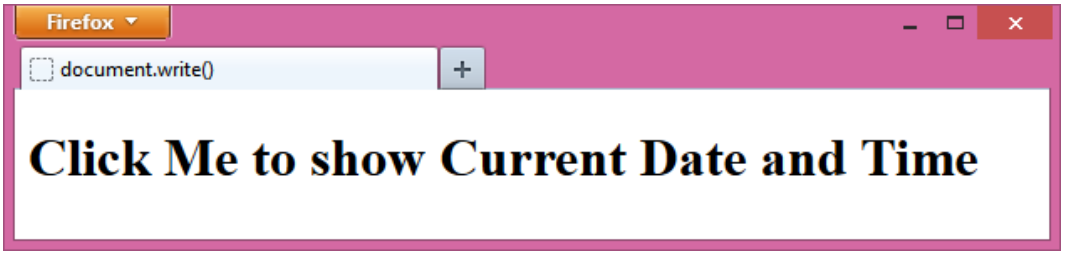
```
document.write("Current Date and Time: <strong>" + Date() +"</strong>");
```

इसलिए JavaScript इस Executable Statement को Execute कर देता है। परिणामस्वरूप Currently Loaded Web Page में जो भी Content होता है, **write()** Method उसे Overwrite करके उसके स्थान पर ***Current Date and Time*** Display कर देता है।
इसी प्रकार से हम यदि चाहें कि *Current Date and Time* तब Display नहीं होना चाहिए, जब Web Page, Web Browser के Window Object में पूरी तरह से Load हो जाए, बल्कि तब होना चाहिए, जब हम Web Page पर दिखाई देने वाले Heading पर Click करें। तो इस जरूरत को पूरा करने के लिए हम निम्नानुसार **<h1>** Element के साथ अपने JavaScript Code को Attach कर सकते हैं:

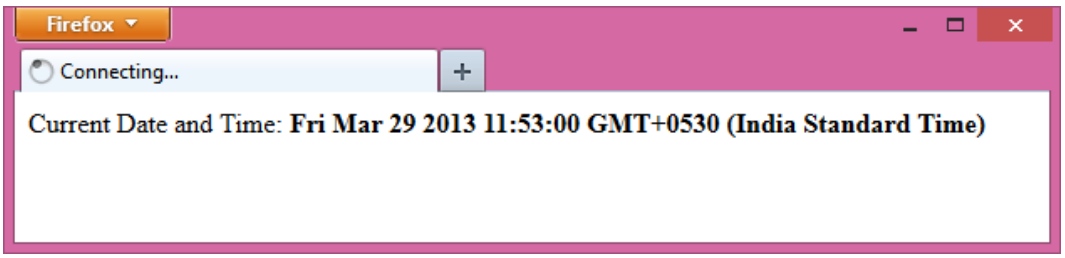
File Name: ***document.write() with click event on Heading1.html***

```
<!DOCTYPE html>
<html>
    <head><title>document.write()</title></head>

    <body>
        <h1 id="clickMe">Click Me to show Current Date and Time</h1>

        <script>
            var clickMeDOMTreeReference = document.getElementById("clickMe");

            clickMeDOMTreeReference.onclick = function(){
                document.write("Current Date and Time: <strong>" + Date() +"</strong>");
            }
        </script>
    </body>
</html>
```

जब हम इस Web Page को Render करते हैं तो इसका Output निम्नानुसार Render होता है:



लेकिन जैसे ही हम इस दिखाई देने वाले Heading1 Content "*Click Me to show Current Date and Time*" Message पर Click करते हैं, हमारा Web Page Content Change होकर निम्नानुसार "**Current Date and Time**" से Overwrite हो जाता है:



चलिए, इस Web Page के JavaScript Code को भी थोडा समझ लेते हैं। जब हम किसी HTML Element के साथ किसी Event Handler को Associate करना चाहते हैं, तब हमें दो काम करने पडते हैं:

1. जिस HTML Element के साथ हमें Event Handler को Associate करना होता है, उस Element में हमें **id** Attribute को Specify करना जरूरी होता है।
2. हमें उस HTML Element का Reference प्राप्त करना होता है, जिसके साथ Event Handler Function को Attach करना है।

हालांकि पिछले Web Page में हमने सीधे ही **window** Object के साथ ***onload*** Event Handler को Attach कर दिया था, क्योंकि *window* ,d Global Object होने की वजह से पूरे JavaScript Code में कहीं भी Accessible रहता है।

लेकिन हमारे Web Page के Elements, Global Objects की तरह व्यवहार नहीं करते बल्कि हमारे पूरे Web Page के सभी Elements, Attributes, Texts आदि हमारे Computer की Memory में Nodes की एक Hierarchy के रूप में Organize रहते हैं, जिसे **DOM Tree** कहते हैं।

इसलिए जब हम किसी HTML Element के साथ किसी Event Handler को Attach करना चाहते हैं, तो सबसे पहले हमें उस HTML Element का Reference या Pointer प्राप्त करना होता है जिसके साथ हम हमारे Event Handler को Attach करना चाहते हैं और **DOM Tree** (*In-Memory Representation of Web Page*) में से किसी Element के Reference या Pointer को प्राप्त करने के लिए हमें ***getElementById()*** Method को Use करना होता है।

getElementById() JavaScript द्वारा Provided एक ऐसा Method है, जो Argument के रूप में उस Element के **ID** को Accept करता है, जिसका Reference या Pointer Return करना होता है।

इसीलिए हमने निम्नानुसार JavaScript Code द्वारा सबसे पहले उस **<h1>** Element का DOM Tree **Reference** प्राप्त किया है, जिसके साथ हमें हमारे Click Event Handler Code को Attach करना है:

```
var clickMeDOMTreeReference = document.getElementById("clickMe");
```

और इस <h1> Element का Reference हम इसीलिए प्राप्त कर पा रहे हैं, क्योंकि हमने हमारे <h1> Element के **id** Attribute में निम्नानुसार "**clickMe**" मान Specify किया है:

```
<h1 id="clickMe">Click Me to show Current Date and Time</h1>
```

इस प्रकार से जब हमें *clickMeDOMTreeReference* Variable में <h1> Element का Reference प्राप्त हो जाता है, तो हम इस <h1> Element के साथ निम्नानुसार Code द्वारा एक Click Event Handler Associate कर सकते हैं:

```
clickMeDOMTreeReference.onclick = function(){
    document.write("Current Date and Time: <strong>" + Date() +"</strong>");
}
```

परिणामस्वरूप जब हम हमारे Current Web Page को Web Browser में Load करके दिखाई देने वाले Heading पर Click करते हैं, तो JavaScript Interpreter उस JavaScript Anonymous Function Code को Run कर देता है, जिसका Reference clickMeDOMTreeReference Variable में Stored होता है।

यानी अब हमारा JavaScript Code तब Run नहीं होता, तब Web Page पूरी तरह से Web Browser में Load हो जाता है, बल्कि तब Run होता है, जब हम Web Browser में दिखाई देने वाले Heading पर Click करते हैं।

इस पुस्तक में Event Handling से सम्बंधित पूरा एक Chapter है, जिसमें विभिन्न प्रकार के Event Handling Methods को Detail से Discuss किया गया है। फिर भी उपरोक्त Event Handling तरीके को ठीक से समझना आपके लिए उपयोगी रहेगा, ताकि आगे आने वाले Chapters में दिए गए विभिन्न प्रकार के Codes व Concepts के आधार पर आप स्वयं अपना Event Driven Program बना सकें।

JAVASCRIPT OR ECMASCRIPT FUNDAMENTALS

JAVASCRIPT OR ECMASCRIPT FUNDAMENTALS

किसी भी Programming Language की तरह JavaScript का भी एक Fundamental या Core Part है, जिसे समझे बिना हम इस Language को इसकी पूरी Power के साथ उपयोग में नहीं ले सकते।

सामान्यतः सभी Programming Languages में ये Core Part लगभग एक जैसा ही होता है इसलिए यदि आपने पहले कोई भी Programming Languages सीखी है, तो इस Part को Clear करने में आपको बहुत ही कम समय लगेगा।

सामान्यतः JavaScript के Core Part के अन्तर्गत हमें ये समझना होता है कि JavaScript के Codes किस तरह से लिखे जाते हैं, Operators, Data Types, Functions, Looping, Conditions आदि किस तरह से काम करते हैं, आदि बातों को जानना होता है और इस Chapter में हम सबसे पहले इन्हीं बातों को जानेंगे।

Syntax

JavaScript के Syntax पूरी तरह से C Language पर आधारित है इसलिए यदि आपने "C" Language या इस पर आधारित किसी अन्य Language जैसे कि "C++", "Java", Perl आदि में से किसी भी एक Language को सीखा है, तो आप बडी ही आसानी से इस Language को Capture कर लेंगे

लेकिन यदि आप Programming के बारे में कुछ भी नहीं जानते, तो आपको कम से कम "C" Language जरूर सीखनी चाहिए क्योंकि "C" Language लगभग सभी अन्य Modern Languages की **Mother Language** है यानी लगभग सभी Modern Languages "C" Language के Concepts पर ही आधारित हैं।

Case Sensitive

JavaScript एक Case Sensitive Language है। यानी इस Language में हम जितने भी Identifiers Create करते हैं अथवा जो भी Identifiers, Functions, Constant, Variables आदि पहले से Predefined हैं, वे सभी Case Sensitive हैं। यानी JavaScript में Small Case Letters व Capital Case Letters में लिखा गया एक ही नाम अलग–अलग माना जाता है।

Identifiers

किसी भी Programming Languages में किसी Variable, Function, Property, Object, Constant आदि को पहचानने के लिए उसका एक नाम Specify किया जाता है। इस नाम को ही **Identifier** कहते हैं। JavaScript में Identifier Define करने के लिए हमें निम्न नियमों को Follow करना होता हैः

1. Identifier का नाम हमेंशा किसी Upper Case या Lower Case **Character, Underscore** या **Dollar** Sign से शुरू होना चाहिए।
2. Identifier के नाम में किसी **Special Symbol, Keyword** या **Reserve Word** का प्रयोग नहीं करना चाहिए।

3 Identifier के नाम में **Digits** का प्रयोग किया जा सकता है, लेकिन Digits का प्रयोग कभी भी नाम की शुरूआत में नहीं होना चाहिए।

ECMAScript के अनुसार सभी Identifiers का नाम Camel Case में होना चाहिए। यानी नाम की शुरूआत Small Case Letter से व बाकी के सभी शब्द Capital Case Letter से। जैसेः

```
basicSalary
firstCar
```

Comments

ECMAScript "C" Language की तरह ही दो तरह से Comments Specify करने की सुविधा देता हैः

Single Line Comment

```
// This is single Line Comment
```

Multi Line Comment

```
/* This
   is
   multiline
   comment
*/
```

Comments का प्रयोग Programmer अपनी जरूरत के अनुसार लिखे गए विभिन्न Codes को ज्यादा बेहतर तरीके से समझने व याद रखने के लिए करता है। ये Comments Web Browser में Render नहीं होते। यानी Comments केवल Programmer अपनी सुविधा के अपने Codes को ज्यादा Understandable बनाने लिए लिखता है।

Statements

चूंकि Web Browser किसी भी JavaScript Code को Line by Line Interpret करता है, इसलिए Code की हर Line को एक **Statement** कहा जाता है।

ECMAScript में हर Statement का अन्त एक Semicolon से किया जाना जरूरी होता है, यानी जहां पर भी Web Browser को Semicolon प्राप्त होता है, वह Statement का अन्त समझता है।

Block Statements

Looping व Conditional Statements ऐसे Statements होते हैं, जो एक से ज्यादा Statements का Group होते हैं। सामान्यतः इन Statements को **Block Statements** कहा जाता है। किसी Block को हमेंशा Opening व Closing Curly Braces के Pair के बीच लिखा जाता है। जैसेः

```
if (condition) {
    alert('Hi, this is in a if Block');
}
```

Block के अन्दर लिखे गए Statements का अन्त Semicolon से होता है लेकिन चूंकि किसी एक Block के अन्दर दूसरा Block Nested हो सकता है, इसलिए Nested Block का अन्त भी Semicolon से नहीं होता। जैसेः

```
if (condition) {
    alert('Hi, this is in a if Block');

    if (condition) {
        alert('Hi, I am nested Block');
    }
}
```

Keywords and Reserved Words

ECMAScript में कुछ नामों या शब्दों को Reserve किया गया है तथा कुछ नामों या शब्दों का Web Browser के JavaScript Interpreter के लिए Special Meaning होता है। इसलिए इन नामों को हम सामान्य **Identifier** की तरह Use नहीं कर सकते।

उदाहरण के लिए जब हम **if** शब्द का प्रयोग करते हैं, तब JavaScript Interpreter को ये Instruction मिलता है कि यदि Parenthesis के बीच Specify की गई Condition **True** हो, तो Block के बीच Specified सभी Statements को Execute करना है अन्यथा नहीं। इसलिए सामान्य परिस्थितियों में हम **if** शब्द को Identifier की तरह Use नहीं कर सकते। Web Site में Specify किए गए विभिन्न Keywords निम्नानुसार हैंः

break	else	new	var
case	finally	return	void
catch	for	switch	while
continue	function	this	with
default	if	throw	
delete	in	try	
do	instanceof	typeof	

इसी तरह से ECMAScript में कुछ शब्दों को Future में Keywords की तरह Use करने के लिए Reserve रखा है। इसलिए हम इन नामों को भी Identifiers की तरह उपयोग में नहीं ले सकते। ये Reserved Words निम्नानुसार हैंः

abstract	enum	int	short
boolean	export	interface	static
byte	extends	long	super
char	final	native	synchronized
class	float	package	throws
const	goto	private	transient
debugger	implements	protected	volatile
double	import	public	

Variables

Variables ऐसे नाम होते हैं जिन्हें एक Programmer की तरह हम हमारे Program में विभिन्न प्रकार के Codes या Values को Identify करने के लिए Assign करते हैं। JavaScript में किसी

Variable को Define करने के लिए हमें "**var**" Operator के साथ उस नाम का प्रयोग करना होता है, जिसे हम हमारे JavaScript Program में Variable की तरह Use करना चाहते हैं।

Variables ऐसे Memory Locations होते हैं, जिनका मान पूरे JavaScript Program के दौरान समय–समय पर जरूरत के अनुसार बदलता रहता है। जैसे यदि हम किसी Image की Height व Width को 300px Set करते हैं और हम चाहते हैं कि जरूरत के अनुसार User उस Image की Size को कम या ज्यादा कर सके, तो हमें इस Size को एक Variable में Store करना होता है।

चूंकि Image की Size बदलने के लिए User किसी न किसी तरह का नया मान Specify करेगा, इस स्थिति में Size का मान बदलेगा और जो मान बदल सकता है, उसे Hold करने के लिए हमें Variable Specify करना होता है। Variable Define करने के लिए हम निम्न Syntax का प्रयोग कर सकते हैं:

```
var variableIdentifierName;
```

variableIdentifierName के स्थान पर हम उस नाम को Specify करना होता है, जिसे हम Variable की तरह Use करना चाहते हैं। उदाहरण के लिए यदि हम किसी Employee की Basic Salary को अपने Program में Store करना चाहते हैं, तो हमें निम्नानुसार Variable Define करना होगा:

```
var basicSalary;
```

ये Statement Web Browser के JavaScript Interpreter को ये Instruction देता है कि हम एक ऐसा Memory Location चाहते हैं, जहां पर हम बदल सकने वाले मानों को Store करेंगे और उस Memory Location को अपने Program में Access करने के लिए हम ***basicSalary*** शब्द का प्रयोग **नाम** की तरह करेंगे।

चूंकि इस Reserve की जाने वाली Memory Location को हम *basicSalary* नाम से Identify करेंगे, इसलिए ये नाम एक **Identifier** है। जबकि इस Memory Location पर Program के Execution के दौरान जो मान Store किया जाएगा, वह मान समय–समय पर जरूरत के अनुसार बदल सकता है, इसलिए ये एक **Variable** Identifier है।

जब हम किसी Variable को Define करते हैं, तब उसमें किसी तरह का कोई मान नहीं होता है और जब किसी Variable में कोई मान नहीं होता है, तब Variable की इस बिना मान वाली स्थिति को "***undefined***" शब्द द्वारा Represent किया जाता है। यानी जब किसी Variable में कोई मान नहीं होता, तब उसमें "***undefined***" होता है।

हम किसी Variable को Define करते समय ही उसमें किसी न किसी तरह का मान Specify कर सकते हैं। ऐसा करने के लिए हमें "**=**" Operator को निम्नानुसार Use करना होता है:

```
var message = "Hi";
```

ये Statement जो Memory Location Reserve करेगा, उस Memory Location को Identify करने के लिए हम *message* नाम का प्रयोग करेंगे जबकि इस Memory Location के Reserve होते ही, इसमें "**Hi**" शब्द Store हो जाएगा। इसलिए इस बार ये Variable Undefined नहीं है।

ECMAScript में जब हम किसी Variable में कोई Value **Initialize** या **Assign** करते हैं, तब उस **Value** के **Data Type** के अनुसार वह Variable Automatically उस **Data Type** का हो जाता है।

यानी हमें किसी Variable के साथ अलग–अलग प्रकार के Keywords को Specify करके JavaScript Interpreter को ये नहीं बताना होता कि हम उस Variable में किस प्रकार का मान Store करेंगे, जैसाकि "C", "C++", "Java" आदि Programming Languages में बताना पडता है।

बल्कि JavaScript Interpreter इतना समझदार है कि हम जैसे ही किसी मान को किसी Variable में Initialize या Assign करते हैं, JavaScript स्वयं इस बात का पता लगा लेता है कि हमने किस प्रकार का मान Variable में Store किया है और JavaScript स्वयं उस Variable को उस प्रकार का Define कर देता है। जैसे–

```
var message = "Hi";
```

इस Statement में हमने *message* Variable में **"Hi"** मान Specify किया है, जो कि एक **String** है। इसलिए JavaScript इस मान को *message* Variable में Store करते ही, *message* Variable को **String Data Type** का Variable Define कर देता है।

हालांकि हम किसी एक Variable में एक प्रकार का मान Specify करने के बाद दूसरे किसी Statement में अन्य प्रकार का मान भी Assign कर सकते हैं। जैसेः

```
var message = "Hi";
message = 200;
```

लेकिन हमें ऐसा नहीं करना चाहिए क्योंकि JavaScript Interpreter पहले *message* नाम के Variable को एक String प्रकार का Variable Define करेगा और Just अगले Statement में उसे एक Numerical Type के Variable में Convert कर देगा।

जब हम किसी Variable को Define करते समय उसके साथ "**var**" Operator का प्रयोग नहीं करते हैं, तब वह Variable हमेंशा **Global Variable** की तरह व्यवहार करता है यानी वह Variable, Web Browser के BOM के **window** Object की Property बन जाता है।

जबकि **var** Operator का प्रयोग करने पर वह Variable एक **Local Variable** की तरह व्यवहार कर सकता है, जबकि उसे किसी Function की Body में Define किया गया हो। Functions के बारे में हम आगे विस्तार से जानेंगे।

Local व **Global** किसी Variable के Scope को Represent करते हैं। यानी किसी Program में कोई Variable Program में किस स्थान पर Use करने के लिए उपलब्ध रहेगा और कहां पर Variable समाप्त हो जाएगा, इस बात को Variable का **Scope** कहा जाता है, जिसके बारे में हम आगे विस्तार से जानेंगे।

JavaScript में भी एक ही Statement द्वारा एक से ज्यादा Variables Define करने के लिए हम Comma Operator का प्रयोग निम्नानुसार कर सकते हैंः

```
var basic, salary, bonus;
```

यदि हम Variables को Initialize भी करना चाहें, तो उपरोक्त Declaration निम्नानुसार भी किया जा सकता है:

```
var message="Hello!",
    salary,
    bonus=12233.50;
```

इन सभी को तीन अलग Lines में लिखने की बजाय हम निम्नानुसार एक Single Line में भी लिख सकते हैं:

```
var message="Hello!", salary, bonus=12233.50;
```

Initialization V/s Assignment

जब हम किसी Variable को Define करते समय ही उसमें किसी Value को Specify करके ये तय कर देते हैं कि वह Variable किस प्रकार का यानी किस **Data Type** का है, तो इस प्रक्रिया को **Initialization** कहा जाता है। जैसे:

```
var message = "Hi";
```

जबकि किसी Variable को Define करने के बाद Program की अगली Line में अथवा पूरे Program में कहीं अन्य स्थान पर जब उस Variable में Value Specify किया जाता है, तो इस प्रक्रिया को **Assignment** करना कहा जाता है। जैसे:

```
var message;
message = "Hi";
```

How to Get this Ebook in PDF Format

ये पुस्तक केवल **PDF Format Ebook** के रूप में ही Available है और आप इस पुस्तक को केवल हमारी **Official Website** (http://www.bccfalna.com/) से ही खरीद सकते हैं। इसलिए यदि आपको ये पुस्तक पसन्द आ रही है और आप इसे PDF Format Ebook के रूप में खरीदना चाहते हों, तो आप इस पुस्तक को Online खरीदने के लिए निम्नानुसार दिए गए **3 Simple Steps** Follow कर सकते हैं:

Select Purchasing EBooks

सबसे पहले **http://www.bccfalna.com/how-to-pay/** Link पर Click कीजिए। जैसे ही आप इस Link पर Click करेंगे, आप हमारी Website के निम्नानुसार **Order Page** पर पहुंच जाऐंगे :

इस Page पर आपको उन पुस्तकों को Select करना है, जिन्हें आप खरीदना चाहते हैं। आप जैसे–जैसे पुस्तकें Select करते जाऐंगे, आपको उनका **Total Amount, Discount** व **Total Payable Amount** उपरोक्त चित्रानुसार दिखाई देने लगेगा, जहां Total Payable Amount ही वह Amount है, जो आपको अपनी Selected EBooks को खरीदने के लिए **Pay** करना होगा।

पुस्तकें Select करने के बाद इसी Page पर दिखाई देने वाले "**Order Details**" Form में आपको निम्न चित्रानुसार अपना *Name*, *Email Address* व *Mobile Number* Specify करके "**Order Now**" पर Click करते हुए उपरोक्त Selected EBooks का Order Place करना होगा:

चूंकि ये सारी पुस्तकें Physical Books नहीं बल्कि **PDF Format Ebooks** हैं। इसलिए ये पुस्तकें आपको आपके Email पर ही भेजी जाऐंगी, जिन्हें आप अपने Email के माध्यम से अपने Computer पर Download करके अपने PDF Supported *Computer, Mobile, Smart Phone, Tablet PC, Net-Book, Notebook* या *Laptop* जैसी किसी भी Device के माध्यम से पढ सकते हैं अथवा यदि आप चाहें, तो अपने Printer द्वारा इन पुस्तकों का Hard Copy Printout निकाल सकते हैं।

इसलिए जरूरी है कि उपरोक्त "**Order Details**" Form पर आप जो **Email Address** व **Mobile Number** Specify करते हैं, वह Working और एकदम सही हो। क्योंकि किसी भी तरह की परेशानी की स्थिति में हम आपको आपके **Mobile Number** पर ही Contact करते हैं।

Pay "Total Payable Amount"

जैसे ही आप "**Order Now**" Button पर Click करेंगे, आपको एक Email मिलेगा, जिसमें आप द्वारा Order की गई EBooks की Details होगी। Selected पुस्तकों का Order Place करने के बाद अब आपको "**Total Payable Amount**" का Payment करना होगा।

यदि आपके पास **Net-Banking** या **Mobile-Banking** की सुविधा है, तो आप Payment करने के लिए अपने Account में Login करके निम्न में से किसी भी Bank A/c में Payment Deposit कर सकते हैं:

भारतीय स्टेट बैंक
State Bank of India
With you - all the way

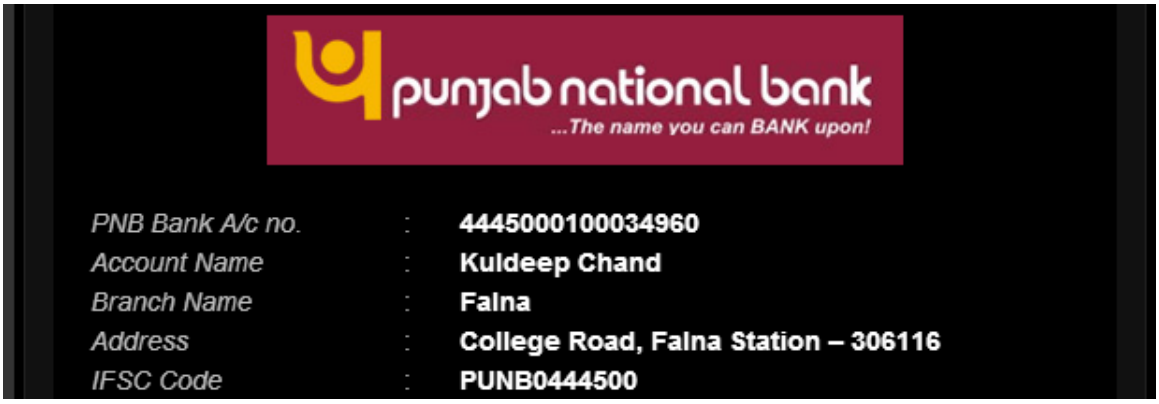
SBI Bank A/c no.	:	**31154882587**
Account Name	:	**Namita Mishra**
Branch Name	:	**Falna**
Address	:	**Near Railway Crossing, Falna Station – 306116**
IFSC Code	:	**SBIN0007868**

बैंक ऑफ़ बड़ौदा
Bank of Baroda
India's International Bank

BOB Bank A/c no.	:	**35260100003212**
Account Name	:	**Namita Sharma**
Branch Name	:	**Falna**
Address	:	**Sanderao Road, Falna, Dist. Pali (Raj.)- Pin-306116**
IFSC Code	:	**BARB0FALNAX**

स्टेट बैंक ऑफ बीकानेर एण्ड जयपुर
State Bank of Bikaner and Jaipur
The Bank with a vision

SBBJ Bank A/c no.	:	**61089986732**
Account Name	:	**Kuldeep Chand Mishra**
Branch Name	:	**Bali**
Address	:	**Sr. Secondary School Road, Bali- 306701**
IFSC Code	:	**SBBJ0010193**

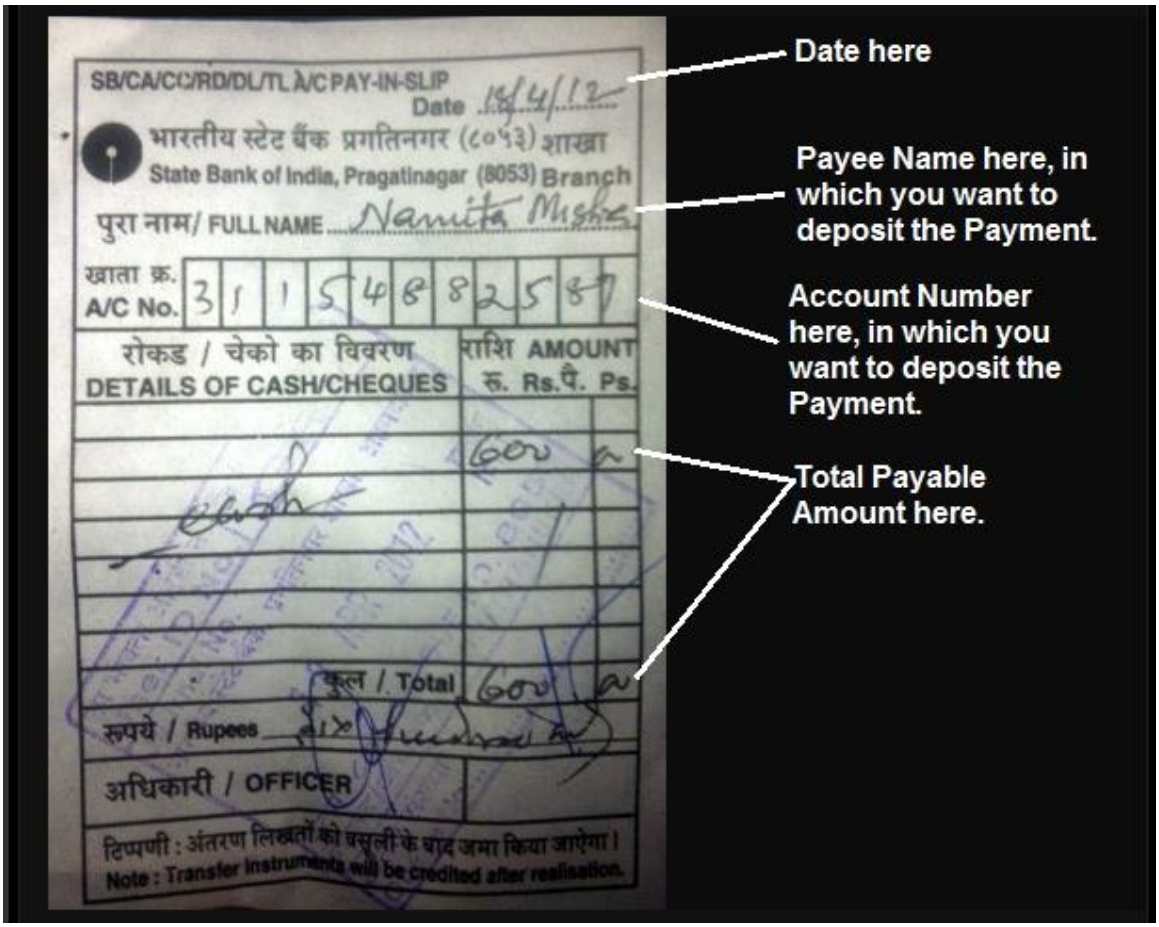


जब आप Net-Banking के माध्यम से Payment करना चाहते हैं, तो आपको लगभग **8 से 24 घण्टे पहले** हमारे उस Account को **Beneficiary** के रूप में अपने Bank A/c से Link करना पडता है, जिसमें आप Payment Deposit करना चाहते हैं।

यदि आपके पास Net-Banking या Mobile-Banking की सुविधा नहीं है, तो आप हमारे किसी भी Bank A/c में *Total Payable Amount*, **Direct Deposit** भी कर सकते हैं।

जब आप **Direct Deposit** करना चाहते हैं, तब आपको आपके किसी भी नजदीकी Bank Branch में जाकर एक *Payment Deposit Slip* Fill-Up करना होता है, जिसमें आपको हमारे किसी भी Bank A/c की Information को Fill करना होता है, **जबकि Payment Deposit करवाने के लिए उसी Bank में आपका स्वयं का Account होना जरूरी नहीं है।**

उदाहरण के लिए यदि आप हमारे SBI Bank A/c में अपनी Selected पुस्तकों का *Total Payable Amount* Pay करने के लिए Bank में जाकर Direct Deposit करना चाहते हैं, तो आप जो **Payment Deposit Slip** Fill-Up करेंगे, वह निम्न चित्रानुसार करना होता है:



इस चित्र द्वारा आप समझ सकते हैं कि Payment, Direct Deposit करने के लिए आपको हमारे किसी Bank A/c की Information को *Payment Deposit Slip* में Specify करना होता है, इसलिए उस Bank में आपका स्वयं का Bank A/c होना जरूरी नहीं होता।

Net-Banking, Mobile-Banking व **Direct Deposit** के अलावा किसी अन्य माध्यम से भी आप Payment कर सकते हैं। उदाहरण के लिए कुछ Banks अपनी ATM Machine द्वारा Direct Payment Transfer करने की सुविधा Provide करते हैं।

यदि आपके Bank का ATM Machine इस तरह से Payment Transfer करने की सुविधा देता है, तो आपको Bank में जाकर Payment Deposit Slip के माध्यम से Payment करने की जरूरत नहीं होती, बल्कि आप Bank के ATM Machine से भी Directly हमारे किसी भी Bank A/c में **Total Payable Amount** Transfer कर सकते हैं। इसी तरह से यदि आप चाहें, तो हमारे किसी भी Bank A/c में Check द्वारा भी Amount Direct Deposit कर सकते हैं।

यानी आप किसी भी तरीके से हमारे किसी भी Bank A/c में *Total Payable Amount* Deposit कर सकते हैं। लेकिन हम **Money-Order, Demand-Draft** या **Check** जैसे Manual माध्यमों से Payment Accept नहीं करते, क्योंकि इस तरह का Payment Clear होने में बहुत समय लगता है। जबकि Direct Deposit या Mobile अथवा Net-Banking के माध्यम से तुरन्त Payment Transfer हो जाता है, जिससे हम आपको आपकी Purchased EBooks **20** से **30 Minute** के दरम्यान आपके Order में Specified **Email** पर Send कर देते हैं।

Confirm the Payment

जब आप अपनी Selected पुस्तकों को खरीदने के लिए उपरोक्तानुसार किसी भी तरीके से *"Total Payable Amount"* हमारे किसी भी Bank A/c में Deposit कर देते हैं, तो Payment Deposit करते ही आपको हमें उसी Mobile Number से एक **Call/Miss Call/SMS** करना होता है, जिसे आपने Order Place करते समय "**Order Form**" में Specify किया था।

इसी Mobile Number के माध्यम से हमें पता चलता है कि आपने किन पुस्तकों के लिए Order किया है और उनका *Total Payable Amount* कितना है। साथ ही हमें ये भी पता चल जाता है कि आप द्वारा Purchase की जा रही पुस्तकें किस Email Address पर Send करनी है।

आपके *Total Payable Amount* को हम Net-Banking के माध्यम से अपने Bank A/c में Check करते हैं और यदि आपका *Total Payable Amount* हमारे किसी भी Bank A/c में Deposit हुआ होता है, तो हम आपको **30 Minute** के दरम्यान आपकी Ordered EBooks आपके Email पर Send कर देते हैं, जिसे आप अगले दिन 12AM तक Download कर सकते हैं।

यदि अभी भी आपको कोई बात ठीक से समझ में न आ रही हो या किसी भी तरह का Confusion हो, तो आप **097994-55505** पर **Call/Miss Call/SMS** कर सकते हैं। यथा सम्भव तुरन्त आपको Callback किया जाएगा और आपकी समस्या या Confusion का Best Possible Solution करने की कोशिश की जाएगी।

उम्मीद है, इस पुस्तक के Sample Chapters का Demo भी आपको पसन्द आया होगा और हमें पूरा विश्वास है कि पूरी पुस्तक आपको और भी ज्यादा पसन्द आएगी।