3. Advanced Client Side Development

IT3406 – Web Application Development II

Level II - Semester 3

3.1 JavaScript Basics

Lesson Outline

- Introduction to JavaScript
  - Importance of JavaScript
  - Integrating JavaScript code
- Variables
- Data types
  - Basic data types
  - Arrays
- JavaScript operators
- Program flow control structures
  - Conditional statements
  - Loops
- JavaScript functions

Introduction to JavaScript

Why JavaScript is important?

- JavaScript adds dynamic nature into the front-end of a web site
- It is more popular scripting language to provide better user interaction in the front-end
- JavaScript code can easily be embedded into HTML 5
- The JavaScript code can alter features of the web page that the HTML5 and CSS3 code produce
- JavaScript code allows you to alter the text that appears on your web page “on the fly,” without requiring your site visitors to reload the web page

Why JavaScript is important?

- JavaScript adds greater flexibility to customize the front end
  - Capable of dynamically changing the web page layout.
  - Can manipulate the look and feel of the website components through Cascading Style Sheets (CSS).
  - Changing the visibility of components depending on the requirement (through CSS).
  - Helps to incorporate dynamic nature to the website.

Where to put JavaScript code?

- How to integrate JavaScript code with HTML5

- You can integrate and run JavaScript code in two different ways,
  - Embedding the JavaScript code directly into the web page
  - Creating an external JavaScript file that the browser downloads and runs

Option 01: Embedding JavaScript

- *"SCRIPT"* element is used to directly embed JavaScript code into the web page
- It has an opening and closing tag that surrounds your JavaScript code

- Eg:

```
<SCRIPT>

    JavaScript code goes here ..

</SCRIPT>
```

Option 01: Embedding JavaScript

- The browser is capable of identifying the *"script"* tag
- Browser executes the code written in JavaScript using the internal JavaScript *Interpreter*
- *Optionally, the "type"* attribute can be specified. Not necessarily in HTML 5.

```
<script type="text/javascript">
```

Embed code in the <HEAD> tag

- If the JavaScript code embeds in the *head* section, the script runs before the page content is executed,

IMPORTANT:

- When you run the test, your browser may not run the JavaScript code or it may prompt you to allow the code to run.
- Some browsers have built-in security features to block running JavaScript code embedded in a web page.
- You'll need to consult your browser documentation on how to enable JavaScript code.

Example for embedded code in the <HEAD> section

```
<!DOCTYPE html>
<html>
<head>
        <title>Testing JavaScript in the Head Section</title>
        <script>
        alert(“This executes before loading the content.");
        </script>
</head>
<body>
        <h1>This is the web page</h1>
</body>
<html>
```

Embed JS in the <BODY> tag

```
<!DOCTYPE html>
<html>
<head>
        <title>Testing JavaScript in the Body Section</title>
</head>
<body>
        <h1>This is before executing the JS code</h1>
        <script>
                alert("This is the JavaScript program!");
        </script>
        <h1>This is after executing the JS code</h1>
</body>
<html>
```

Compare the output

- Embedding JavaScript code inside the body element of a web page slows down the loading process of the content,

Best Practice

"If the code location is not crucial, it is best to place the *script* element at the end, after the HTML5 code"

Option 02: Using external JS file

Usage:

- Using an external file is a good design aspect when the same code repeatedly applicable to many number of web pages.
- Keeping the repeating JS code separately improves maintainability of the code.

Can be an absolute, relative or full path to a remote machine location

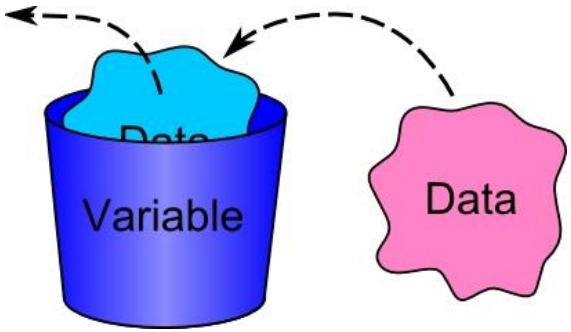
```
<script src="javascript_code.js"></script>
```

Variables in JavaScript

Variables in JavaScript

- *Variable* holds data temporarily in the memory until a program complete its execution.
- These temporary data storage locations help manipulating data, retrieving data at later stages and displaying it to the user.

- *Variables* are names that represent storage locations in the computer memory.



Rules for defining JS variables

Remember to adhere following rules when defining variables.

- Variable names can contain letters, numbers, underscores, and dollar signs
- Variable names must begin with a letter, an underscore, or a dollar sign
- Variable names are case sensitive
- Do not use JavaScript keywords as variable names

Declare and initialize a variable

- Use *var* keyword to declare the variable
- Give a meaningful name (best practice)
- Now you can initialize the variable using a value

Example: var name; name = “Kamal”;

Set aside a place in memory for storing data and call that location ***name***

Use the JavaScript ***assignment operator*** to assign a value ***“Kamal”*** to the variable

JavaScript Data Types

JavaScript data types

Basic data types

- **Numbers:** Either integer values (Eg: 5) or floating point values (Eg: 5.345) .
- **Strings:** A series of characters, strung together in memory one after the other.

JavaScript basics

Arrays

- Arrays allow us to store a list of items into a single variable.
- For example:

```
var marks = [80, 50, 65];
```

- Individual items are called elements of the array.
- Using the index of the element, the value can be extracted.
- Syntax: marks[1] will give the number 50.

JavaScript *operators* for data manipulation

JavaScript mathematical operators

Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulus (the remainder of a division operation)
++	Increment (increases the value by 1)
––	Decrement (decreases the value by 1)

JavaScript Boolean operators

Operator	Description
&&	logical AND
||	logical OR
!	logical NOT

Program flow control structures

Program flow control structures

- Flow control structures are used to alter the default flow of code execution in a JavaScript program.
- These structures contain conditional statements from which a certain condition is evaluated.
- Depending on the result of the evaluation, the program would execute the code.

Program flow control structures

- Conditional statements
  - If block
  - If else block
  - Switch statement
- Loops
  - Do.. While
  - While
  - For
  - For.. In

if statement

- The condition is evaluated whether *true* or *false.*
- The code inside the block only executes if the condition is evaluated as *true.*

```
if (condition) {
    code goes here…
}
```

```
if (5>2) {
    alert("success");
}
```

else statement

- With the ***if*** statement, if the condition is not met, the interpreter just skips the code you specify in the code block.
- The ***else*** statement allows you to specify code to run if the condition evaluates to a false value.

```
if (type == “Lory”) {
    message = “Sorry, you are not allowed to park here”;
    status = “failed”;
} else {
    message = “You may begin the game”;
    status = “success”;
}
```

switch statement

- Switch statement is similar to the else if statement.
- JavaScript interpreter evaluates the *expression* and the result will be matched against the case statement.
- *break* statement breaks the flow and exits from the block.

```
switch (expression) {
        case match1:
                statements
                break;
        case match2:
                statements
                break;
        default:
        statements
}
```

switch statement contd..

- Each case statement specifies a different possible result of the expression.
- If there is a match, the interpreter runs the statements contained in that section.
- The *break* statement forces the interpreter to skip over the remaining case statement sections.
- If none of the case results matches, the interpreter runs the statements under the default statement.

Loops in JavaScript

- When the same block of code runs multiple times, it is called a *loop.*
- Typically, one or more variables changes values in each iteration of the loop .
- Loop ends once a specific criteria meets.

Loops in JavaScript

- JavaScript supports several looping structures as shown in the table below,

Statement	Description
`do..,while`	Executes a block of statements and checks a condition at the end
`for`	Checks a condition, executes a block of statements, and then alters a specified variable
`for...in`	Executes a block of statements for each element contained in an array
`while`	Checks a condition and then executes a block of statements

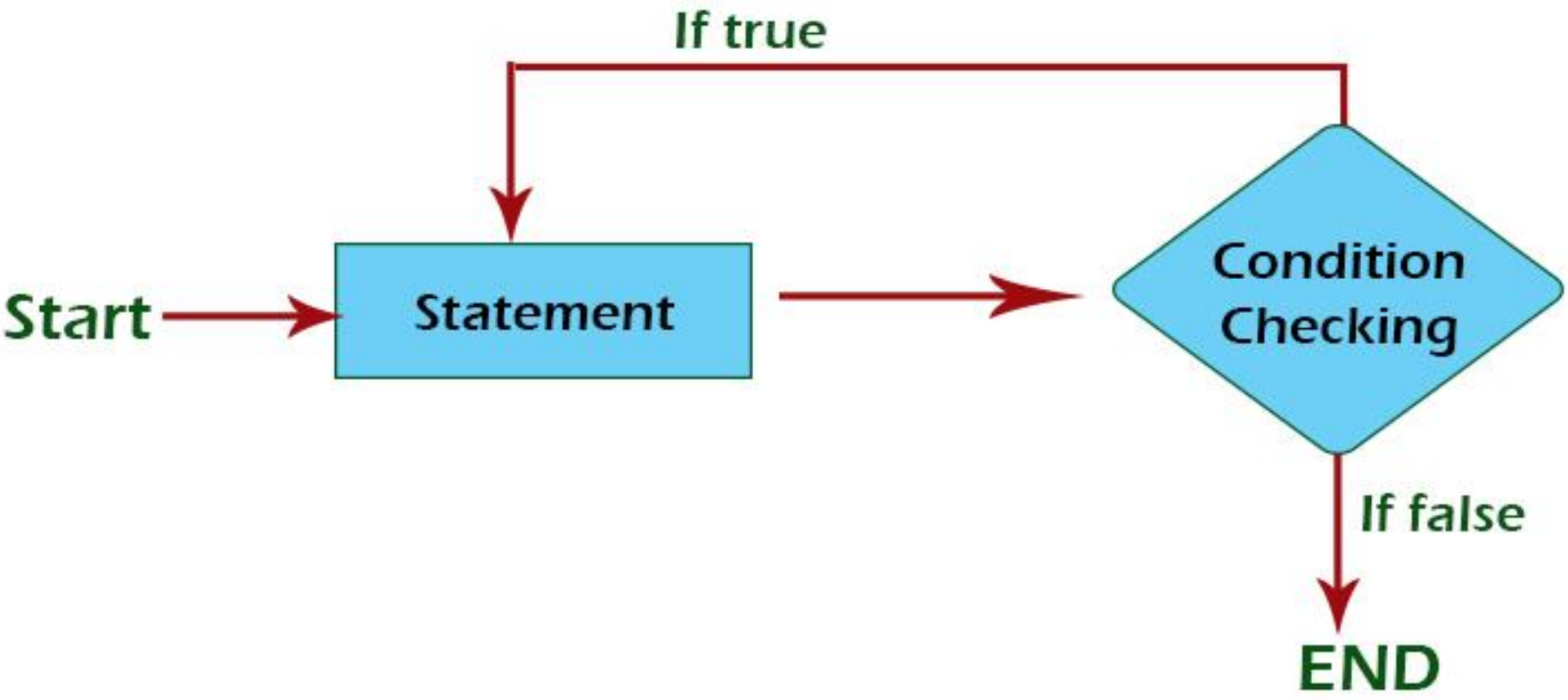
The *do… while* loop

- Do while loop executes the first iteration without checking a condition.
- Do while loop executes at least once.
- At the end of the block, it tests a condition to determine whether to repeat or not.

The *do… while* loop syntax

```
do
{
    statements..
}
while (condition);
```

The *do… while* loop flowchart



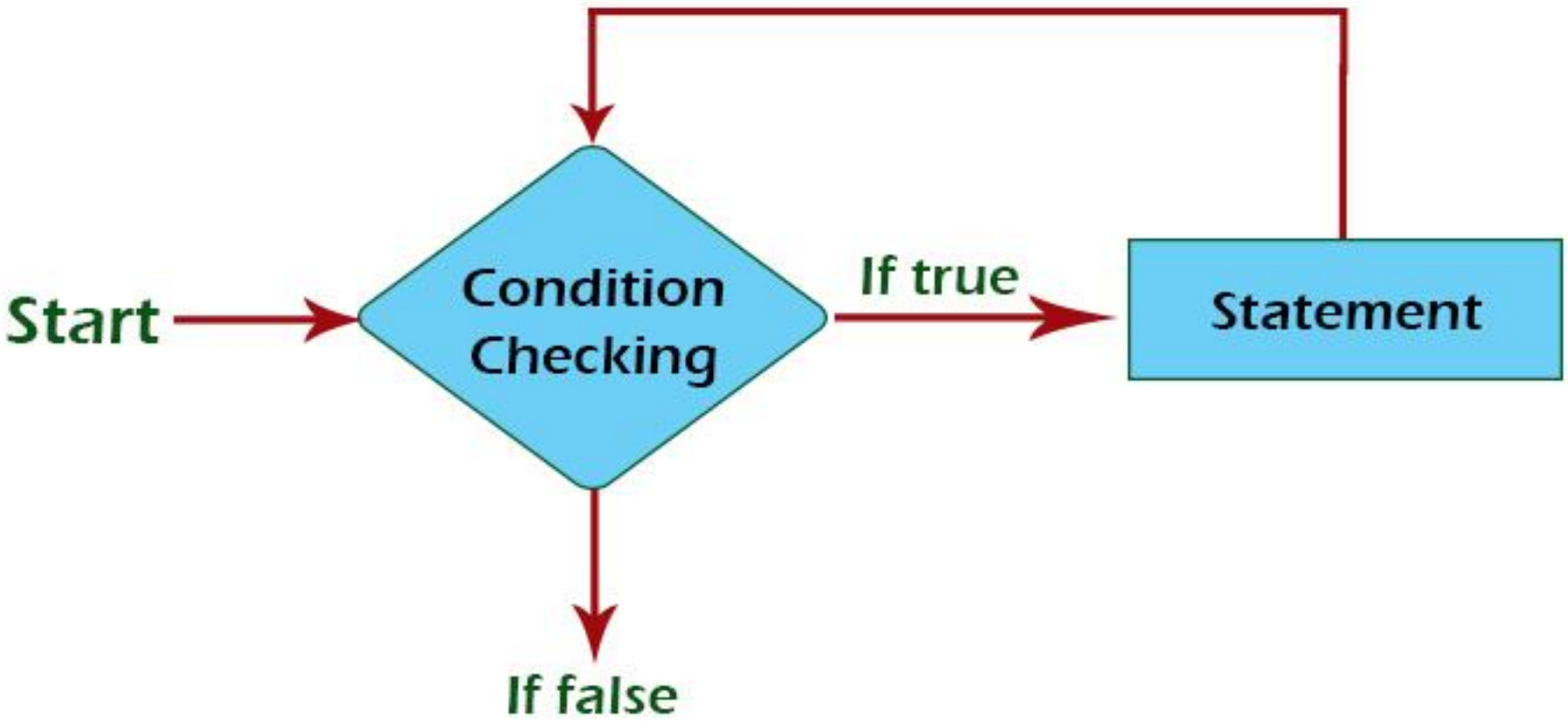
The *while* loop

- *while* loop acts opposite to the *do.. while* loop
- While loop checks the condition at the very beginning
- None of the iterations would continue without evaluating the condition

The *while* loop syntax

```
while (boolean condition)
{
   loop statements...
}
```

The *while* loop flowchart



The *for* loop syntax

Sets initial values to variable

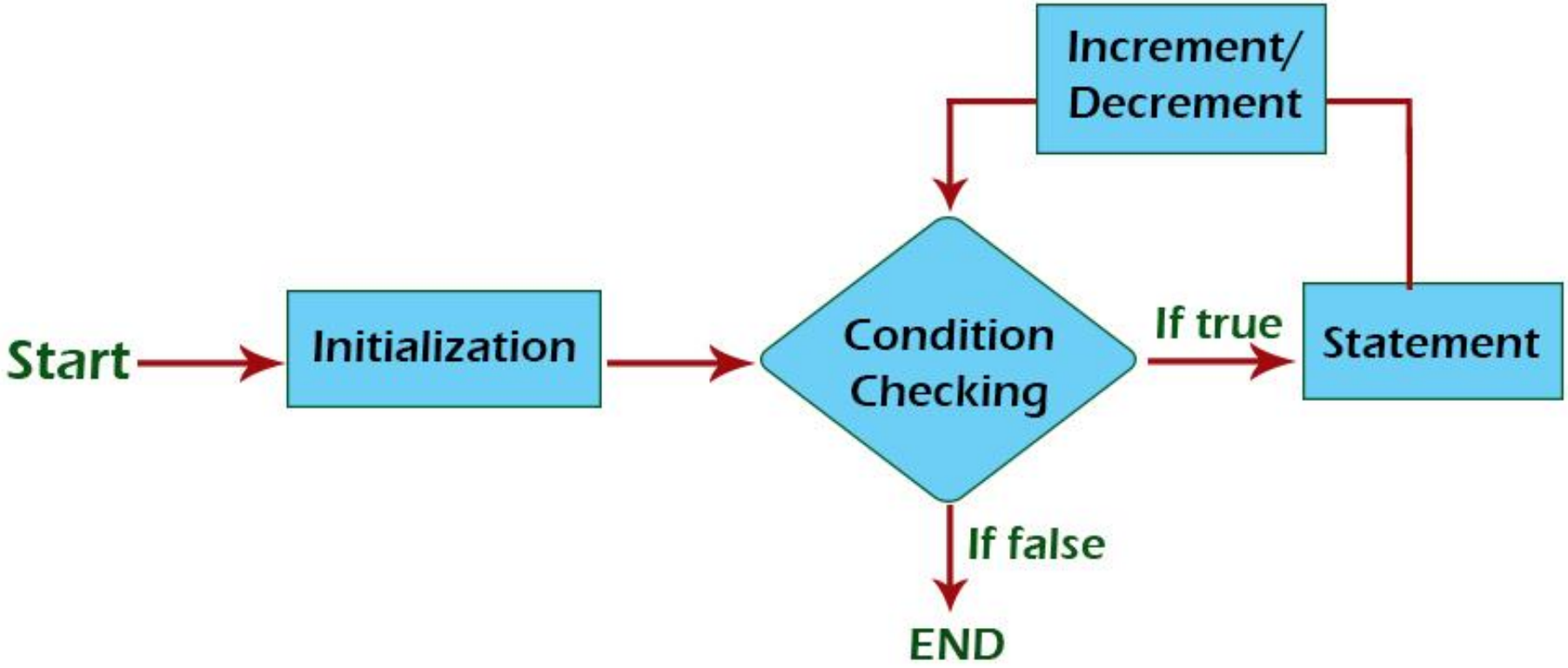
Condition to be evaluated

Variable to be changed in each iteration

```
for(statement1; condition; statement2) {
    Statements to be executed

}
```

The *for* loop flowchart



The *for.. in* loop

- The *for...in* loop helps to iterate over an array of which the number of values are not certain
- It executes till the end of the array extracting individual data values per iteration
- The *for...in* statement ends when it runs out of data elements in the array
- Elements are accessed based on the respective index in the array

The *for.. in* loop syntax

```
for (variableName in Object)
{
    statement(s)
}
```

JavaScript functions

What is a functions

- A function is a block of organized, reusable code that is used to perform a single action
- If we have a code segment (complex/simple) repeats over and over in different places, we can write a function and reuse the function without repeating the same code chunk
- Functions make the code simple and maintainable

Syntax of a JavaScript function

```
function name(param1, param2, ...) {
    function code
    return value;
}
```

JavaScript functions

- Return Type:
  - If a function returns a value, the type of the returning value becomes the return type of the function
  - If it does not return the keyword *void* is used
- Function Name:
  - The function name and the parameter list together forms the signature of the function
  - Name is important for calling the funtion

JavaScript functions

- Parameter List:
  - A parameter is like a placeholder.
  - Parameters/arguments have a type, order and number of parameters
  - Parameters are optional and a function can be defined without any parameters
- Function Body:
  - The function body contains a code block to perform the task

How to use functions

- JavaScript functions are called using the name of the function and passing the relevant parameters
- For example;

```
var result = add_values(5,3);
```

Here, the result variable will store the ***value 8*** after calling the ***add_values()*** function

Summary

- JavaScript is a language which can be used to manipulate front-end dynamic behavior.
- Variables are used to store data temporarily to support executing the program in memory.
- Data types specify which kind of data cab ne stored in a particular variable.
- JavaScript operators help in manipulating data on variables.
- Mathematical and Boolean operators are two broader categories of operators.

Summary

- JavaScript flow control statements are used to alter the program execution flow.
- Conditional statements and looping structures help mainly in flow controlling.
- JavaScript functions can be used to reuse code and improve maintainability.

3.2 Document Object Model (DOM)

Lesson Outline

- Introduction to Document Object Model
- DOM tree structure
- JavaScript and DOM
- DOM object properties
- Finding HTML elements
- Accessing form data using JS objects

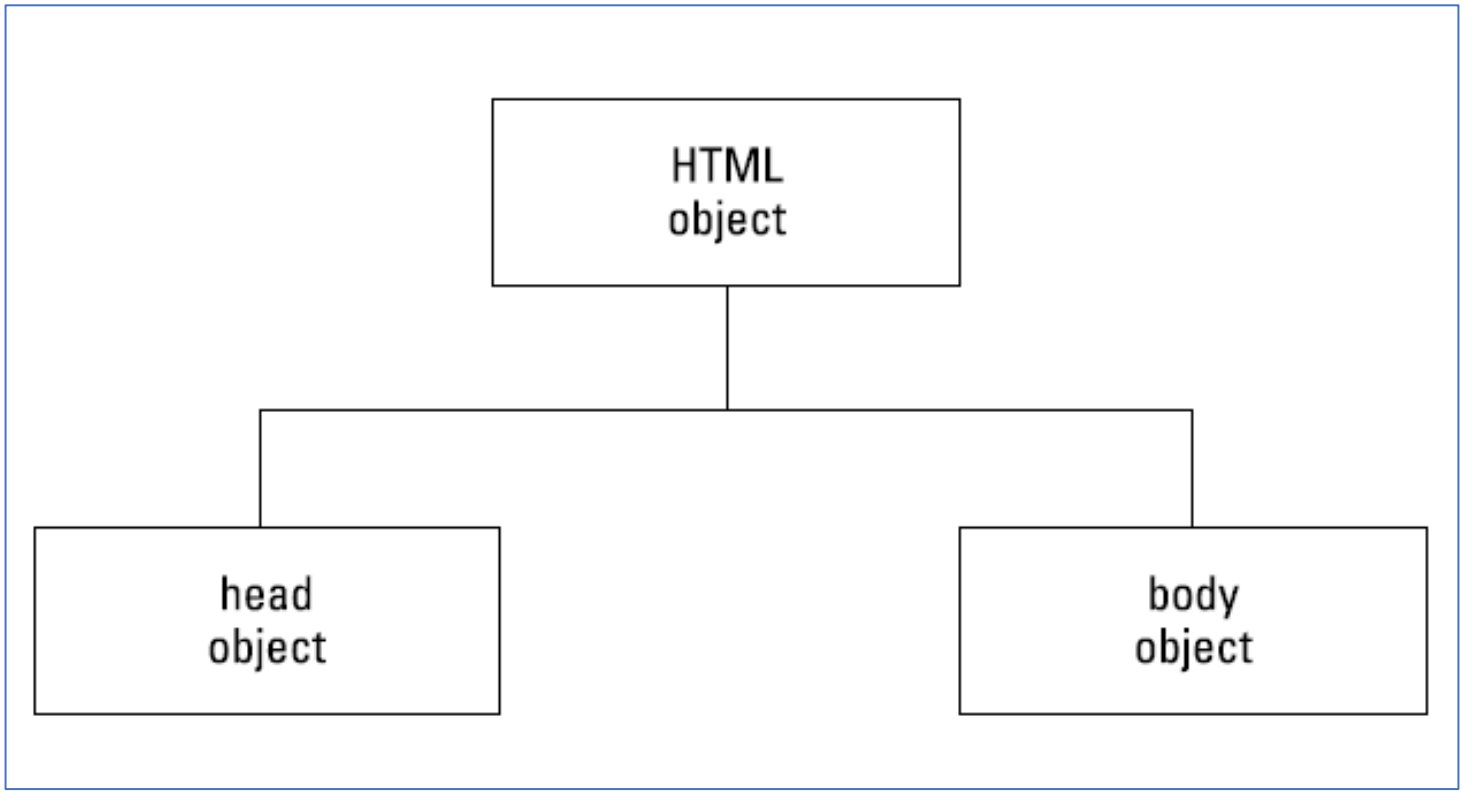
Introduction to DOM

Document Object Model

- HTML document has a number of elements.
- DOM helps finding and locating these elements in a standard way.
- DOM provides a hierarchical tree structure of elements.
- By traversing through the tree structure, one can reach the desired element.

DOM tree structure

- Start of the tree structure is always ***html*** element



DOM tree structure contd..

- Under the top/root element all the other elements are referred to as child nodes.
- In the html document *head* comes first and it is named as ***first child*** of the html element.
- *body* comes after the *head* and it is named as ***second child*** of the html element.

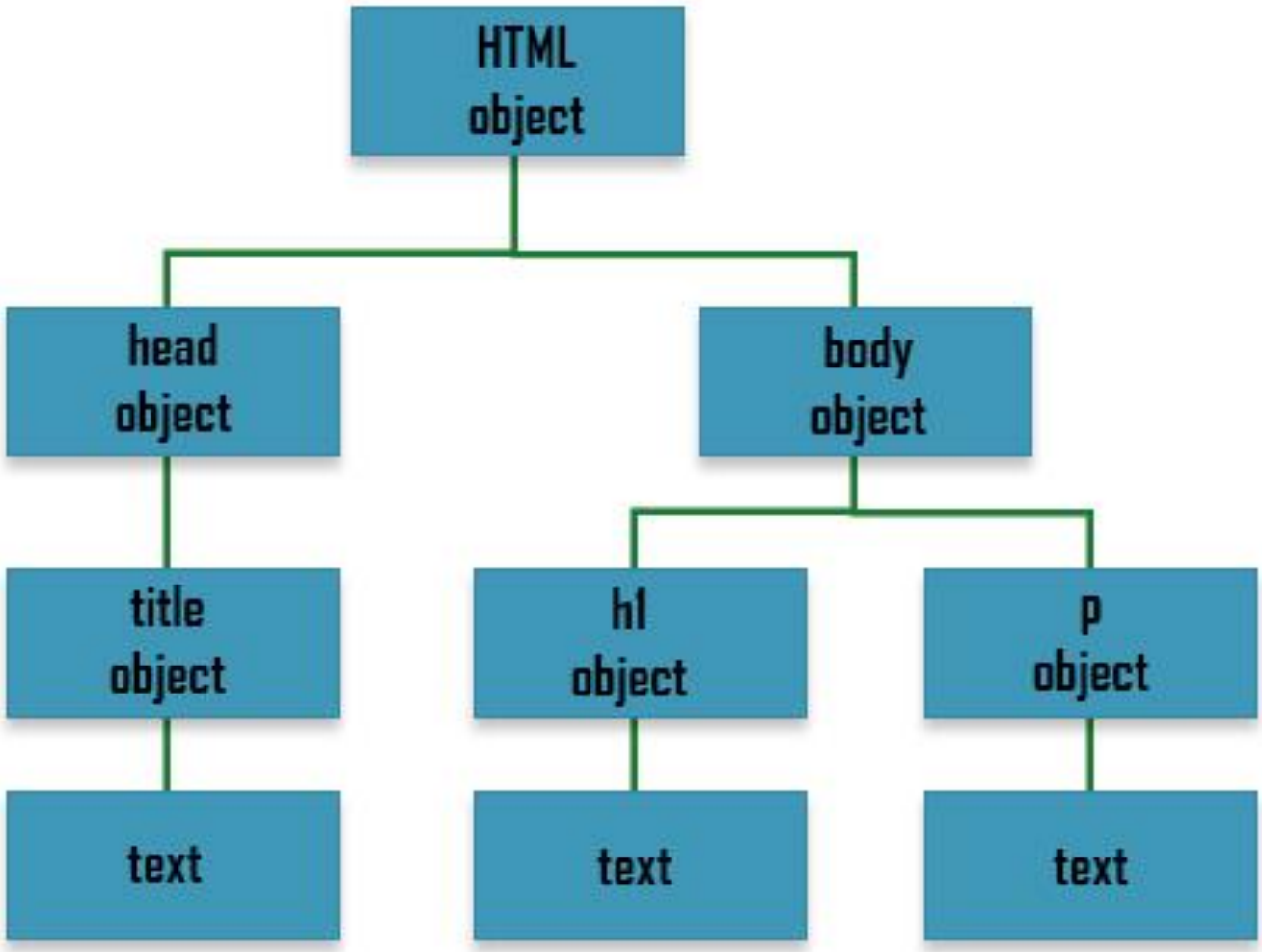
***This convention of naming the elements in DOM is important.*

Example code

```
<!DOCTYPE html>
<html>
<head>
<title>Sample DOM web page</title>
</head>
<body>
<h1>This is the heading of the web page</h1>
<p>This is sample text</p>
</body>
</html>
```

Source: w3shools

DOM tree



JavaScript and DOM

- The browser uses the DOM tree to keep track of all the HTML5 elements, their content, and the styles on the web page
- JavaScript has full access to the DOM tree created by the browser as it also works in the client side
- JavaScript is capable of modifying the DOM hierarchy
- This capability of modifying the DOM helps to add dynamic nature to the front end of the web applications

JavaScript and DOM

- JavaScript treats each element in DOM as an object
- Objects have *Properties* and *Methods*

Properties: Properties define information about the object
Methods: Methods are actions to take with the objects

JavaScript and DOM

JavaScript has a special object which refers to the entire DOM of an HTML document

Using *document.property, any property of the document object can be referenced*

Eg: var url = document.URL;

Document object properties

Property	Description
activeElement	Returns the element that currently has the focus of the web page window
anchors	Returns a list of all the anchor elements on the web page
body	Sets or retrieves the body element of the web page
cookie	Returns all cookie names and values set in the web page
characterSet	Returns the character set defined for the web page
documentElement	Returns the DOM object for the html element of the web page
documentMode	Returns the mode used by the browser to display the web page
domain	Returns the domain name of the server used to send the document
embeds	Returns a list of all the embed elements in the web page
forms	Returns a list of all the form elements in the web page
head	Returns the head element for the web page
images	Returns a list of all the img elements in the web page
lastModified	Returns the time and date the web page was last modified
links	Returns a list of all the anchor and area elements in the web page
title	Sets or retrieves the title of the web page
URL	Returns the full URL for the web page

Document object methods

Method	Description
`createElement()`	Adds a new element object
`createTextNode()`	Adds a new text object
`getElementbyId(id)`	Returns an element object with the specified id value
`getElementsByClass Name(class)`	Returns a list of elements with the specified class name
`getElementsByTagname(tag)`	Returns a list of elements of the specified element type
`hasFocus()`	Returns a true value if the web page has the window focus
`write(text)`	Sends the specified text to the web page
`writeln(text)`	Sends the specified text to the web page, followed by a new line character

JavaScript DOM objects

- Previous slides describe the document properties and methods.
- Apart from that, JavaScript also has properties and methods that apply to each element object in the document.
- These properties and methods can be used to expand the capabilities of JavaScript when interacting with the DOM elements.

JavaScript DOM object properties

Property	Description
attributes	Returns a list of the object's attributes
childElementCount	Returns a list of the number of child objects the object has
childNodes	Returns a list of the object's child nodes, including text and comments
children	Returns a list of only the object's child element object nodes
classList	Returns a list of the class name attributes of an object
className	Sets or returns the value of a class attribute of an object
firstChild	Returns the first child object for the object
id	Sets or returns the id value of the object
innerHTML	Sets or returns the HTML content of the object
lastChild	Returns the last child object for the object
nodeName	Returns the name of the object
nodeType	Returns the element type of the object
nodeValue	Sets or returns the value for the object
nextSibling	Returns the next object at the same level in the tree as the object
parentNode	Returns the parent object for the object
previousSibling	Returns the previous object at the same level in the tree as the object
style	Sets or returns the value of the style property for the object

JavaScript DOM object methods

Method	Description
appendChild(*object*)	Adds a new child object to an existing object
blur()	Removes the page focus from an object
click()	Simulates a mouse click on the object
cloneNode	Duplicates an object in the DOM
contains(*object*)	Returns a true value if the object contains the specified object
focus()	Places the window focus on the object
getAttribute(*attr*)	Returns the value for the specified object attribute
getElementsByClassName(*class*)	Returns a list of objects with the specified class name
getElementsByTagName(*tag*)	Returns a list of objects with the specified tag name
hasAttribute(*attr*)	Returns true if the object contains the specified attribute
hasAttributes()	Returns true if the object contains any attributes

JavaScript DOM object methods

`hasChildNodes()`	Returns `true` of the object contains any child objects
`insertBefore(object)`	Inserts the specified object before the object
`removeAttribute(attr)`	Removes the specified attribute from the object
`removeChild(object)`	Removes the specified child object from the parent object
`replaceChild(object)`	Replaces the child object with the specified object
`setAttribute(attr)`	Sets the specified attribute of the object to the specified value
`toString()`	Converts the object to a string value

Finding elements using JS objects

Challenge:

- As your web pages become more complicated, it may contain possibly thousands of elements
- Finding a specific element and changing or modifying it will be a real challenge
- Two different ways to find an element
  - Using a unique feature assigned to the element to jump directly to it
  - navigate your way down to the element's object from a specific point in the DOM tree

Use element id to find an element

- Assign the element a unique id attribute value
- Reference the elements in your JavaScript code by using the *getElementById()* method

```
<script>
function changeit() {
    var answer = prompt("Enter some new text");
    var spot = document.getElementById("here");
    spot.innerHTML = answer;
}
</script>
```

```
<p id="here">This is the original text</p>
```

Walking the tree to find element

- Different properties of child elements can be used to search for an element in DOM hierarchy,
  - Use the firstChild property
  - Use the nextSibling property
- We can alternatively use firstChild, lastChild, nextSibling, or previousSibling properties to reach wherever we want in the page.

Working with form data

- JavaScript objects are capable of accessing the content of form elements.
- For example, Text boxes, Text Areas, Check boxes and radio buttons.
- Let us look at how JavaScript uses DOM tree to access and work with form elements.

Text Box

- Use the *value* attribute of the object to read any text that may already be in the text box.
- Accessing data in a text box

```
var textbox = document.getElementById("test");
var data = textbox.value;
```

- Writing data to a text box

```
var textbox = document.getElementById("test");
var answer = prompt("Enter text to change");
textbox.value = answer;
```

Text Box DOM properties

Property	Description
autocomplete	Sets or retrieves the value of the autocomplete attribute
autofocus	Sets or retrieves whether the text box gets the window focus when the web page loads
defaultValue	Sets or retrieves the default value assigned to the text box
disabled	Sets or retrieves whether the text box is disabled in the form
form	Retrieves the parent form the text box belongs to
list	Retrieves the data list associated with the text box
maxLength	Sets or retrieves the maximum length of the text box
name	Sets or retrieves the name attribute for the text box
pattern	Sets or retrieves the pattern attribute for the text box
placeholder	Sets or retrieves the placeholder attribute for the text box
readOnly	Sets or retrieves whether the text box is read only
required	Sets or retrieves whether the text box is a required field in the form
size	Sets or retrieves the value of the size attribute for the text box
type	Retrieves the type of element the text box is
value	Sets or retrieves the value attribute for the text box

Text Area

- For text area elements also we use *value* property.
- Same as we did for text box except few unique DOM properties,
- **cols**: Sets or retrieves the number of columns assigned to the text area
- **rows**: Sets or retrieves the number of rows assigned to the text area
- **wrap:** Sets or retrieves whether text can auto-wrap within the text area

Check Boxes

- Check box is used to capture whether a particular option is selected or not.
- In order to check the condition we can use DOM *checked* property.

Example:

```
var pizza = document.getElementById("pizzabox");
if (pizza.checked) {
   alert("your pizza will be delivered shortly");
}
```

Checkbox DOM properties

Property	Description
autofocus	Sets or retrieves whether the check box gets the focus when the web page loads
checked	Sets or retrieves the state of the check box
defaultChecked	Retrieves the default state of the check box
defaultValue	Retrieves the default value assigned to the check box
disabled	Sets or retrieves whether the check box is disabled

Radio buttons

- Working with radio buttons is always a complicated matter.
- All the radio buttons in the same group use the same name property.
- Browser handles them as a group.
- Only one radio button in the group can be selected at any time.
- Handling data from a radio button requires using the *checked* and *value* object properties.

3.3 : Asynchronous JavaScript and XML (AJAX)

Lesson Outline

- Introduction to AJAX
- Establishing connection to server
- XMLHttpRequest class methods
- XMLHttpRequest class properties
- Caching and AJAX

Introduction to AJAX

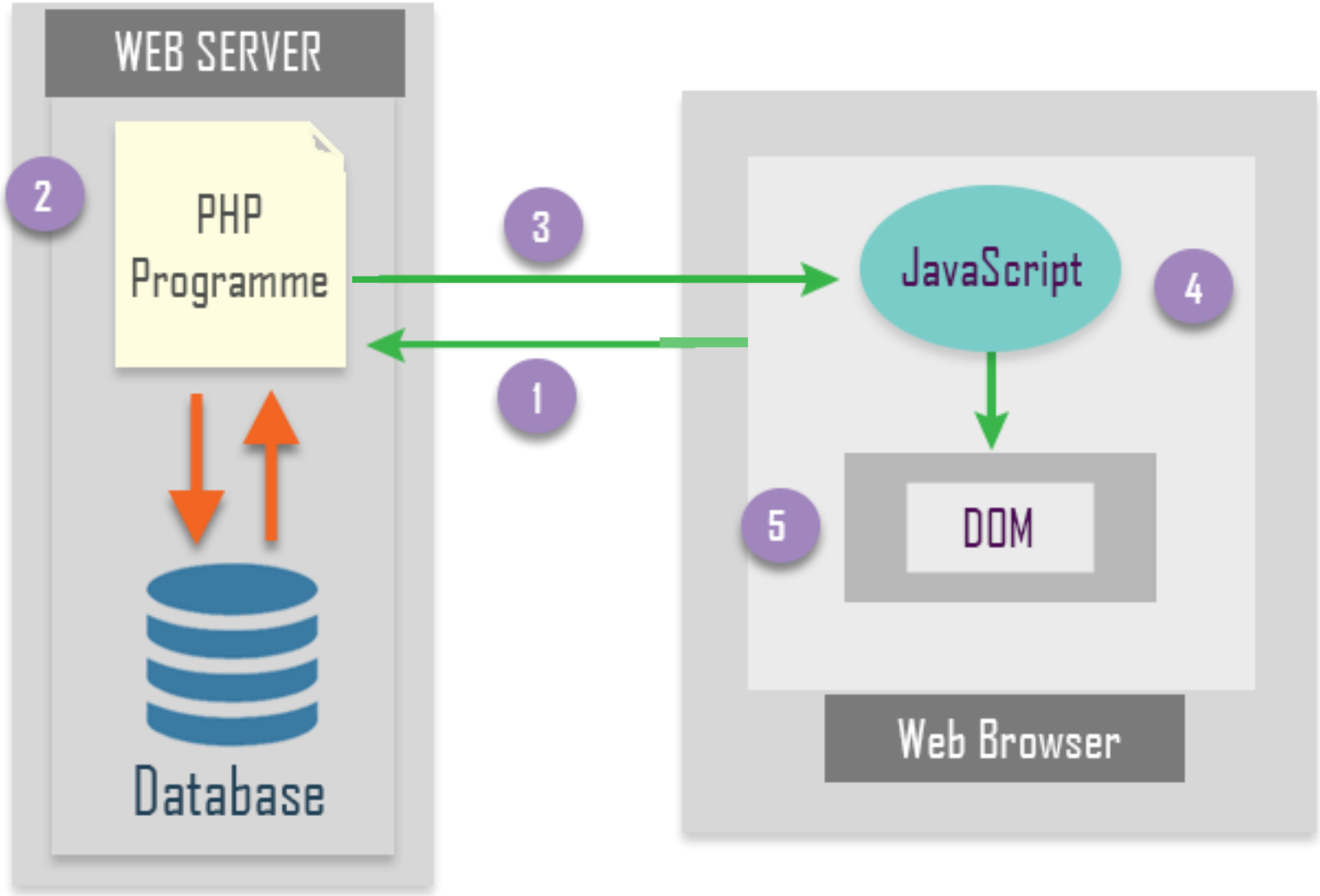
Asynchronous JavaScript and XML

- AJAX combines several existing web languages and standards.
- AJAX helps to produce dynamic content on a web page.

Technologies associated with AJAX

- JavaScript
- Server side scripting language (PHP,JSP, etc..)
- Extensible Markup Language (XML)
- HTML and CSS
- Document Object Model (DOM)

Putting everything together



Putting everything together

Step 01: JavaScript communicates with Web server

Step 02: Web server runs PHP program

Step 03: PHP program sends data through XML

Step 04: JS uses HTML and CSS for styling and positioning data

Step 05: JavaScript uses DOM to place data in the web page

IMPORTANT: AJAX created a special JavaScript object that can communicate with web servers

Communication between JavaScript and the web server

- Initially XMLHTTP ActiveX object was introduced by Microsoft to connect with the server and retrieving web page's content.
- Most of the other browsers did not adhere to the standards of Microsoft.
- As a result new XMLHttpRequest object introduced.
- XMLHttpRequest object method was more popular and supported by almost all browsers.

XMLHttpRequest class methods

Methods defined in XMLHttpRequest object class

Following methods support in establishing and communicating between JavaScript and server.

Method	Description
abort()	Cancels an existing request that is waiting for a response
getAllResponseHeaders()	Retrieves the HTTP header information returned by the web server
getResponseHeader()	Retrieves information from a specific HTTP header
open(*method*,*url*,*async*,*user*,*pass*)	Opens a connection to the specified web server
send(*string*)	Sends a request to the web server
setRequestHeader()	Adds HTML variable/value pairs for the request

Establishing the connection

- Use open() method to define connection between browser and server.
- send() method of the XMLHttpRequest object sends the request to the server.

```
var con = new XMLHttpRequst();
con.open("GET", "search.php" , true);
con.send();
```

Parameters of open() method

Parameter 1:

- Specifies the method (GET or POST)

Parameter 2:

- The URL to send the request to

Parameter 3:

- Specifies the connection type (Synchronous or Asynchronous)

Synchronous : Waits till response arrives to continue

Asynchronous: Does not wait till the response

Difference of using GET and POST

- GET: Parameters of the request are added to the URL itself

```
con.open("GET", "myprog.php?id=100&name=rich", true);
con.send();
```

- POST: Parameters are sent within the send() method

```
con.open("POST", "myprog.php", true);
con.send("id=100&name=rich");
```

XMLHttpRequest class properties

XMLHttpRequest class properties

Property	Description
onreadystatechange	Defines a callback function that the browser triggers when the HTTP connection changes state
readyState	Contains the connection status of the HTTP connection
responseText	Contains the response sent by the web server in text format
responseXML	Contains the response sent by the web server in XML format
status	Contains the numeric HTTP response code from the web server
statusText	Contains the text HTTP response string from the web server

States managed by *readyState* property

State 0: The connection has not been initialized

State 1: The connection to the server has been established

State 2: The server received the HTTP request message

State 3: The server is processing the HTTP request

State 4: The server sent the response

Caching and AJAX

AJAX and Cached pages

- Web browsers are capable of caching the response returned by a specific URL.
- It is important to reduce the amount of data the browser must download from the server each time.
- Indirectly it cases minimizing the time to load a web page.

Is it problematic caching?

- When caching is applied to HTTP requests, sent by the XMLHttpRequest object, there is an issue.
- Assume, you use the same URL to retrieve dynamic data.
  - What causes the error??
  - Instead of dynamic data to be received, always the cached data will be given
- Simply, the cached response will be used as the valid response for the URL .

How to solve cache issue with AJAX

- Solution is to create a unique URL for each HTTP request.
- This can be done by adding a large random number as a GET variable/value pair.

```
var random = Math.floor(Math.random() * 1000);
var myurl = "myprog.php?x=" + random;
con.open("GET", myurl, true);
```

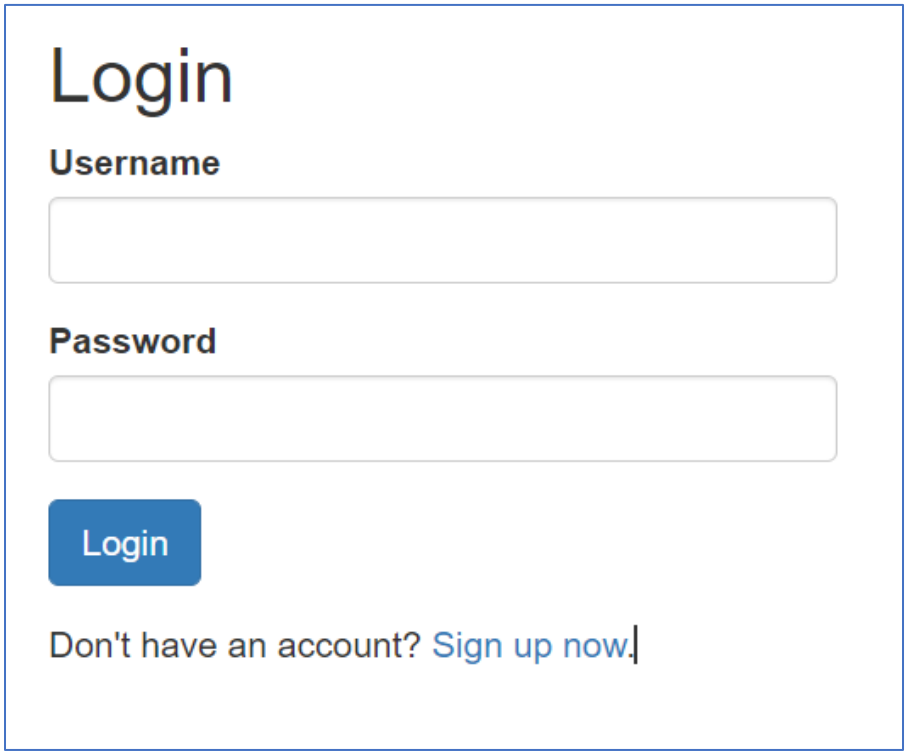
3.4 Client-side Validation with JavaScript

JavaScript for validations

- JavaScript code is used for client side validation as it runs on browser.
- JavaScript is not success on all the time as the user can disable JavaScript in the browser.
- Some of the brewers or versions may not support
- JavaScript is capable of whether the form field asre filled with values, are they within the expected bounds, format of the input, etc.

JavaScript validations with a login form example

- Create a login form with the fields username and password as shown here.



JavaScript validations with a login form example

- HTML code to create the login form

```
<form action="welcome.php" method="post" onsubmit="return validate(this)">

    <div class="form-group">
        <label>Username</label>
        <input type="text" name="username" class="form-control" value="">
        <span class="help-block"></span>
    </div>

    <div class="form-group">
        <label>Password</label>
        <input type="password" name="password" class="form-control">

    </div>

    <div class="form-group">
        <input type="submit" class="btn btn-primary" value="Login">
    </div>
    <p>Don't have an account? <a href="register.php">Sign up now</a>.</p>
</form>
```

JavaScript validations with a login form example

- Onsubmit() function calls the JavaScript validation upon clicking the ***submit*** button of the form

```
<form action="welcome.php" method="post" onsubmit="return validate(this)">

    <div class="form-group">
        <label>Username</label>
        <input type="text" name="username" class="form-control" value="">
        <span class="help-block"></span>
    </div>

    <div class="form-group">
        <label>Password</label>
        <input type="password" name="password" class="form-control">

    </div>

    <div class="form-group">
        <input type="submit" class="btn btn-primary" value="Login">
    </div>
    <p>Don't have an account? <a href="register.php">Sign up now</a>.</p>
</form>
```

Validate *username* field

- Within the *script* tag write the following code to validate the username field of the form.

```
function validateUsername(field)
{
    if (field == "") return "No Username was entered.\n"
    else if (field.length < 5)
        return "Usernames must be at least 5 characters.\n"
    else if (/[^a-zA-Z0-9_-]/.test(field))
        return "Only a-z, A-Z, 0-9, - and _ allowed in Usernames.\n"
    return ""
}
```

Validate *username* field

- The *validateUsername* function allows only the characters a-z, A-Z, 0-9, _ and – as the input for the username field.
- Further it ensures that usernames are at least five characters long.
- If it is empty, it returns an error.
- Next, if the username entered is nor empty, but shorter than five characters, it returns an error message.

Validate *username* field

- By passing regular expression to *test* function, it matches any character that is *not* one of those allowed in the regular expression.
- The defined regular expression will be matched against the field value.
- If any character which is outside the definition of the regular expression, the function returns true.
- Then the *validateUser* function returns an error message.

Validate *password* field

- Within the *script* tag write the following code to validate the password field of the form.
- See the defined rules using regular expressions to check the format of the input field .

```
function validatePassword(field)
{
    if (field == "") return "No Password was entered.\n"
    else if (field.length < 6)
        return "Passwords must be at least 6 characters.\n"
    else if (!/[a-z]/.test(field) || ! /[A-Z]/.test(field) || !/[0-9]/.test(field))
        return "Passwords require one each of a-z, A-Z and 0-9.\n"
    return ""
}
```

Validate *password* field

- First the function checks whether field is empty
  - if it is empty, it returns an error
  - Next, if the password entered is shorter than six characters, it returns an error message
- Here the expression specifies three requirements for a good password
  - a lowercase, uppercase, and numerical character
- The test function is called three times, once for each of these cases
- In case of an absence of any defined criteria, test method returns false
- Otherwise, the empty string is returned.

Putting all JavaScript in a separate file

- It is good to separate the JS content from the HTML code
  - It makes the maintenance easier than having everything together
- Link the external JS file into the code using *script* tag

External JS File Name

```
<script src="validate_functions.js"></script>
```

Regular Expressions

- Regular expressions are more important to define the validation rules.
- Pattern matching is the main principal behind regular expressions.
- Regular expression metacharacters are the key to define rules for pattern matching in a more simplified way.

Regular expression metacharacters

Metacharacters	Description
/	Begins and ends the regular expression
.	Matches any single character except the newline
*element**	Matches *element* zero or more times
element+	Matches *element* one or more times
element?	Matches *element* zero or one times
[*characters*]	Matches a character out of those contained within the brackets
[^*characters*]	Matches a single character that is not contained within the brackets
(*regex*)	Treats the *regex* as a group for counting or a following *, +, or ?
left|*right*	Matches either *left* or *right*
[*l*-*r*]	Matches a range of characters between *l* and *r*

Regular expression metacharacters

Metacharacters	Description
^	Requires match to be at the string's start
$	Requires match to be at the string's end
\b	Matches a word boundary
\B	Matches where there is not a word boundary
\d	Matches a single digit
\D	Matches a single nondigit
\n	Matches a newline character
\s	Matches a whitespace character
\S	Matches a nonwhitespace character

Regular expression metacharacters

Metacharacters	Description
`\t`	Matches a tab character
`\w`	Matches a word character (`a-z`, `A-Z`, `0-9`, and _)
`\W`	Matches a nonword character (anything but `a-z`, `A-Z`, `0-9`, and _)
`\x`	Matches *x* (useful if *x* is a metacharacter, but you really want *x*)
`{n}`	Matches exactly *n* times
`{n,}`	Matches *n* times or more
`{min,max}`	Matches at least *min* and at most *max* times

Activity

- Implement the following regular expressions and see how it validates the pattern.

`rec[ei][ei]ve`	Either of *receive* or *recieve* (but also *receeve* or *reciive*)
`rec[ei]{2}ve`	Either of *receive* or *recieve* (but also *receeve* or *reciive*)
`rec(ei|ie)ve`	Either of *receive* or *recieve* (but not *receeve* or *reciive*)
`cat`	The word *cat* in *I like cats and dogs*
`cat|dog`	The word *cat* in *I like cats and dogs* (matches either *cat* or *dog*, whichever is encountered first)

Using regular expressions in JS

- JS uses regular expressions in two methods frequently.
  - **test method** (tells whether the argument matches the regular expression)
  - **replace method** (takes the second parameter for the string to replace the text that matches)
- you have already seen) and replace. Whereas test just, replace takes a second parameter: the string to.

Examples: *test()* and *replace()*

test()

```
document.write(/cats/i.test("Cats are funny. I like cats."))
```

- If the word cats appear in the string, it returns true.

replace()

```
document.write("Cats are friendly. I like cats.".replace(/cats/gi,"dogs"))
```

- Above statement replaces both occurrences of the word *cats* with the word *dogs.*

Note:

- (/g) defines the search as global to find all occurrences.
- (/i) defines to be case-insensitive.

3.5 : MVC Architecture and Tradeoffs

Lesson Outline

- MVC Architecture
  - Model
  - View
  - Controller
  - Communication among components in MVC
- Alternative Approaches to OOP Web Development
  - MVP method
  - MVVM method

MVC Architecture

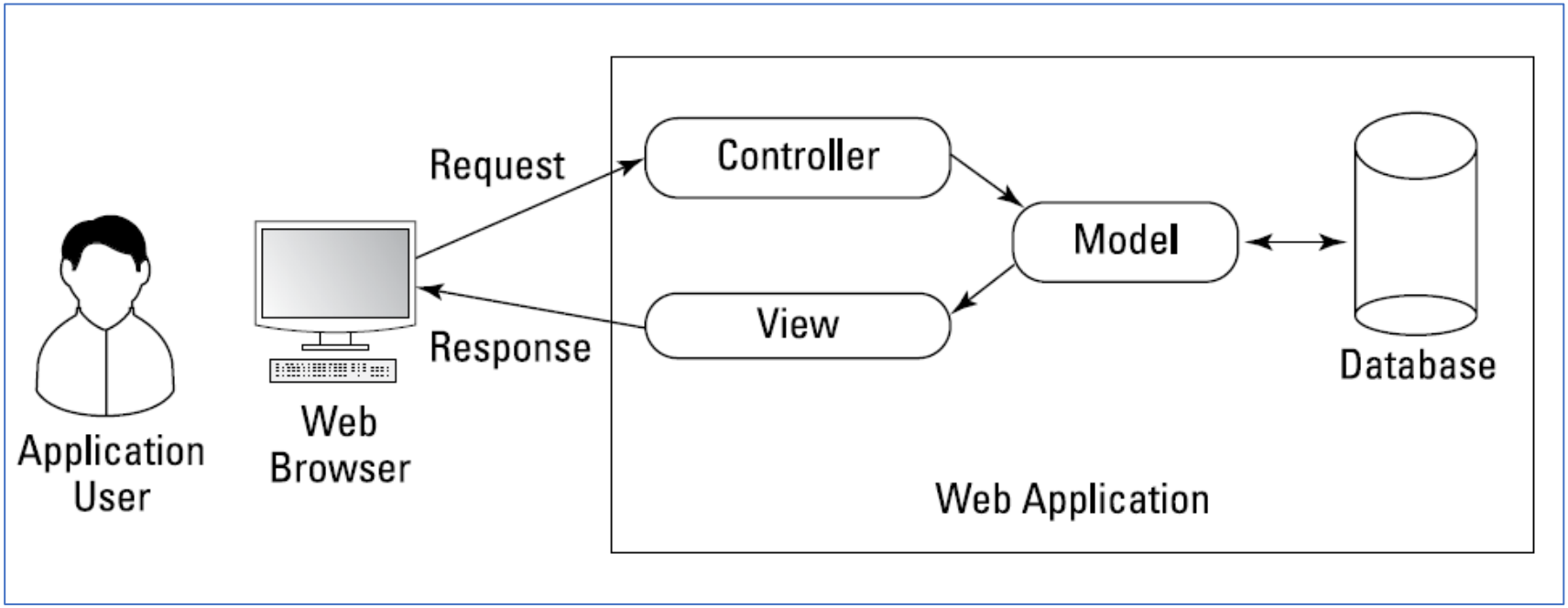
MVC Architecture

- MVC stands for *model–view–controller.*
- MVC method splits object oriented program code into multiple parts.
- It makes it easier to code and implement the web site using Object Oriented Programming (OOP).
- Separating the user view and data processing. components help developers to efficiently code and easily maintain the code at a later stage

MVC Architecture

- **The model:** One or more classes that interact with the application data.
  - This helps in implementing the business logic to process data, store and manipulate.
- **The view:** A class that displays the application data in the graphical environment.
- **The controller:** Works as an intermediate for view and model.
  - Listens for user input and passes the input to the appropriate model class methods for processing.

MVC Architecture Component Interaction Diagram



The Model

- The model component contains majority of the server-side coding (PHP).
- It provides a common interface between the application and any data.
- The code for model resides between the application and the database tables.
- Works with data storing, retrieving and manipulating as required.

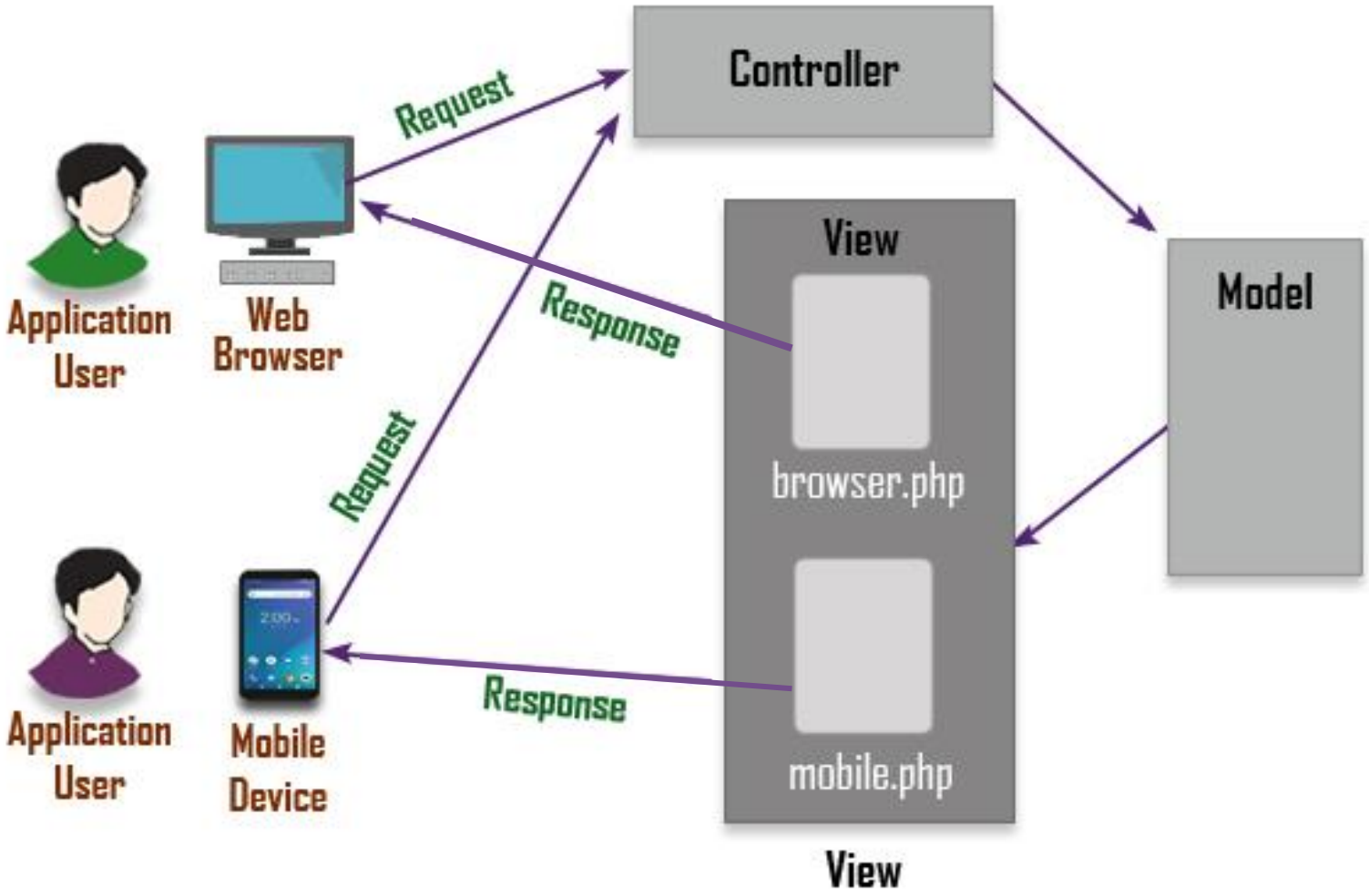
The Model

- Most MVC model implementations use a technique called *object-relational mapping* (ORM) to provide this interface.
- The ORM class is responsible for handling the methods for all interaction with the underlying table (CRUD Operations):
  - Creating new data records
  - Reading existing data records
  - Updating existing data records
  - Deleting existing data records

The View

- The view component is responsible for all the output from the application.
- It takes the raw data provided by the model component and formats it in a way that's visually pleasing to the application user.
- For our web applications, the view component is where all the HTML and CSS code resides.
- Helps to create applications that support both desktop and mobile environments with less effort.

Views depending on the screen size



Views depending on the screen size

- MVC architecture provides easy way of creating applications that support multiple devices.
- All the other processes same except the view generation.
- For example;
  - Devices on the diagram submit the same HTTP request to the controller, which forwards both requests to the model.
  - The model sends the same responses to the view, but the view processes the responses differently (see previous slide diagram).

The Controller

- The controller accepts requests from the application user and sends them to the components required to satisfy the request.
- The controller uses *routing* to determine which model class method to run based.
- on the client browser's request. Routing maps the specific HTTP GET or POST.
- request received from a client browser to a specific model class method.

The controller

- MVC controllers utilize the *rewrite rules* feature of the webservers .
- Through rewrite rules, the url turns in to clean to help clean up the format of the request URL. Rewrite rules allow you to.
- customize the format of the URL to pass information in a cleaner-looking format.
- than what the standard GET method uses.

Communication sequence in MVC

1. The controller receives the request from users
2. The request will be passed to the appropriate class method implemented in the model
3. The model class method performs the appropriate action related to data
4. The model class method passes any resulting data or status to the view
5. The view sends a response back to the website with the data

What issues exist in MVC architecture

- The controller handles client requests but not responsible for returning the responses.

- The model retrieves data but the view is responsible for the formatting data.

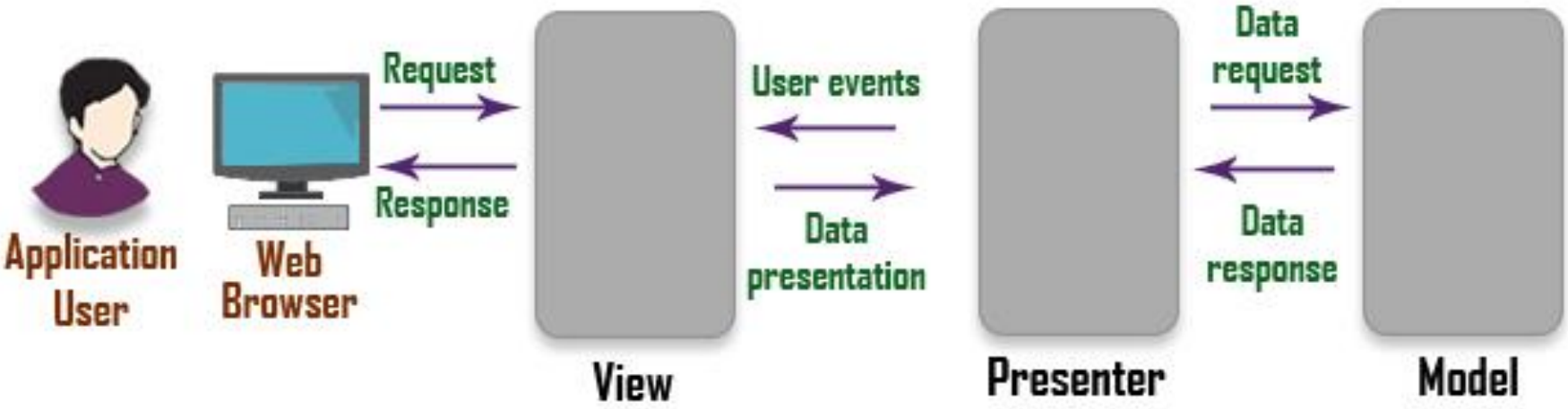
- It is better if the view has the capability to format the data according to the browser environment in the client's end.

Other Architecture Models

Model-View-Presenter (MVP) model

- The *model–view–presenter* (MVP) method is another popular method of creating object-oriented web applications.
- In the MVP method, the view handles both the request and response parts of the process, taking on the MVC controller's function of communicating with the client.
- This eliminates the communication issues identified in MVC method.

MVP diagram



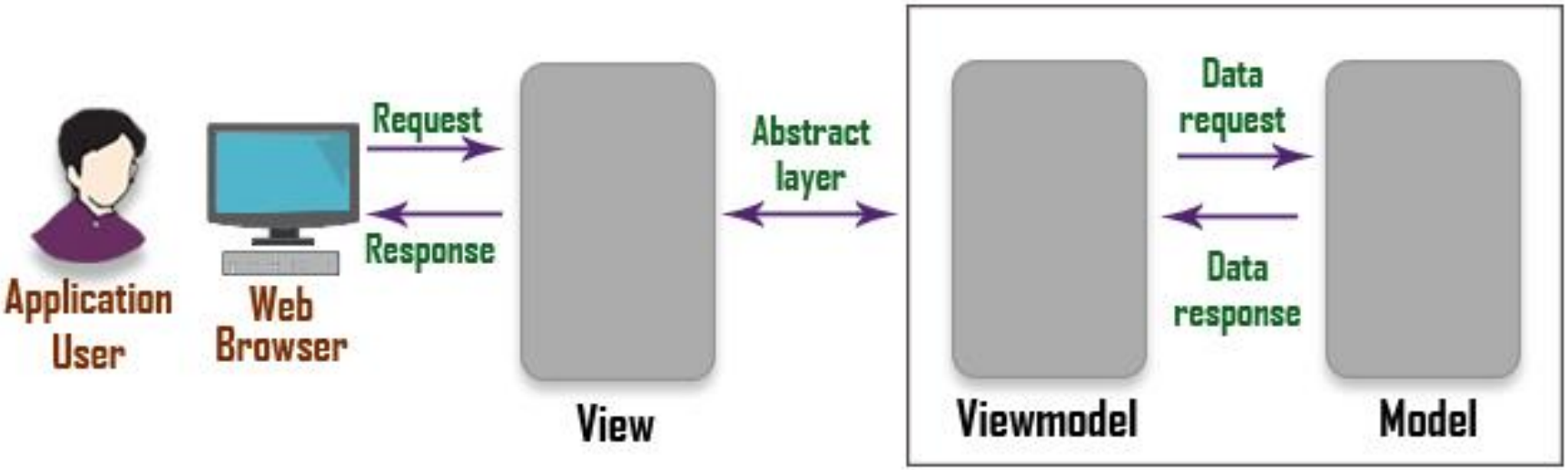
MVP method

- The presenter acts as the middleman similar to MVC architecture.
- It lies in between the model and the view.
- The client requests and calls the appropriate model class methods.
- The result after processing the request, the response is sent to presenter.
- Presenter passes the response to the view.

Model-View-View Model (MVVM) method

- Similar to the MVP method viewmodel acts as a middleman between the view and the model .
- In presenter module it manipulates data in the MVP method.
- MVVM method the viewmodel does not manipulate the data.
- Viewmodel just provides an interface between the view and the model.

MVVM method diagram



MVVM method

- The viewmodel creates an abstract layer between the view and the model.
- This abstract layer allows the programmers working on the view code.
- The mechanism of abstracting helps programmers to work on separate parts without worrying the way data are being processed in the underlying layers.
- Code bases become easily manageable and maintainable.

End