

# SCRIPTING LANGUAGE (CAC254)

**Sulav Nepal**

Email: [nep.sulav@live.com](mailto:nep.sulav@live.com)

Contact No.: 9849194892

**Master's in Computer Information System (MCIS) – Pokhara University**  
**Bachelor of Science, Computer Science & Information Technology (B.Sc. CSIT) – Tribhuvan University**  
**Microsoft Technology Associate (MTA): Windows Server Administration Fundamentals**  
**Microsoft Certified Technology Specialist (MCTS): Windows Server 2008 R2, Server Virtualization**  
**Microsoft Specialist (MS): Programming in HTML5 with JavaScript and CSS3**  
**Microsoft Students Partner (MSP) 2012 for Nepal**

# Syllabus

## UNIT 1: Client Side Scripting

- *Introduction*
- *Need of Client Side Scripting Language*
- *Formatting and Coding Conventions*
- *JavaScript Files*
- *Comments*
- *Embedding JavaScript in HTML*
- *Using Script Tag*
- *NoScript Tag*
- *Operators*
- *Control Structures*
- *Array and For Each Loop*
- *Defining and Invoking Functions*
- *Built in Objects*
- *Date Objects*
- *Interacting with the Browser*
- *Windows & Frames*
- *Document Object Model*
- *Event Handling*
- *Forms*
- *Cookies*
- *Handling Regular Expression*
- *Client Side Validations*

# UNIT 1

## *CLIENT SIDE SCRIPTING*

# JavaScript

- *HTML to define the content of web pages; CSS to specify the layout of web pages; JavaScript to program the behavior of web pages.*
- *JavaScript is a case sensitive scripting language.*
- *A scripting language is a lightweight programming language.*
- *JavaScript is usually embedded directly into HTML pages.*

# JavaScript

- *JavaScript is an interpreted language (that scripts execute without preliminary compilation).*
- *Everyone can use JavaScript without purchasing a license.*
- *Java and JavaScript are two completely different languages in both concept & design.*
- *Java (developed by Sun Microsystems) is a powerful and much more complex programming language — in the same category as C and C++.*

# What can JavaScript do?

- **JavaScript gives HTML designers a programming tool** – HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax. Almost anyone can put small “snippets” of code into their HTML pages.
- **JavaScript can put dynamic text into an HTML page** – a JavaScript statement like this:  
`document.write("<h1>"+name+"</h1>")` can write a variable text into an HTML page.
- **JavaScript can react to events** – a JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element.
- **JavaScript can read and write HTML elements** – a JavaScript can read and change the content of an HTML element.

# What can JavaScript do?

- **JavaScript can be used to validate data** — a JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing.
- **JavaScript can be used to detect the visitor's browser** — a JavaScript can be used to detect the visitor's browser, and — depending on the browser — load another page specifically designed for that browser.
- **JavaScript can be used to create cookies** — a JavaScript can be used to store and retrieve information on the visitor's computer.

# JavaScript – Syntax

- *Use JavaScript to write text on a web page:*

`<body>`

`<script type="text/javascript"> document.write("HelloWorld!");`

`</script>`

`</body>`

- *Add HTML tags to the JavaScript*

`<body>`

`<script type="text/javascript"> document.write("<h1>HelloWorld!</h1>");`

`</script>`

`</body>`



# JavaScript – Comments

- *Single line comments*

```
<body>
<h1 id="myH"></h1>
<p id="myP"></p>
<script>
// Change heading:
document.getElementById("myH").innerHTML = "JavaScript Comments";
// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>
</body>
```

- *Multiple lines comments*

```
<body>
<h1 id="myH"></h1>
<p id="myP"></p>
<script>
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
*/
document.getElementById("myH").innerHTML = "JavaScript Comments";
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>
</body>
```

# JavaScript – Comments

- *Single line comments to prevent execution*

```
<body>
<h2>JavaScript Comments</h2>
<h1 id="myH"></h1>
<p id="myP"></p>
<script>
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>
<p>The line starting with // is not executed.</p>
</body>
```

- *Multiple lines comments to prevent execution*

```
<body>
<h2>JavaScript Comments</h2>
<h1 id="myH"></h1>
<p id="myP"></p>
<script>
/*
document.getElementById("myH").innerHTML = "Welcome to my Homepage";
document.getElementById("myP").innerHTML = "This is my first paragraph.";
*/
document.getElementById("myP").innerHTML = "The comment-block is not executed.";
</script>
</body>
```

# What can JavaScript do? – Example 1

```
<html>
```

```
<body>
```

```
<h2>What Can JavaScript Do?</h2>
```

```
<p id="demo">JavaScript can change HTML content.</p>
```

```
<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello  
JavaScript!">Click Me!</button>
```

```
</body>
```

```
</html>
```

## Output:

### What Can JavaScript Do?

JavaScript can change HTML content.

Click Me!

### What Can JavaScript Do?

Hello JavaScript!

Click Me!

# What can JavaScript do? – Example 2

```
<html>
```

```
<body>
```

```
<h2>What Can JavaScript Do?</h2>
```

```
<p>JavaScript can change HTML attribute values.</p>
```

```
<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>
```

```
<button onclick="document.getElementById('myImage').src='https://s3-us-west-2.amazonaws.com/s.cdpn.io/93927/pic_bulbon.gif'">Turn on the light</button>
```

```

```

```
<button onclick="document.getElementById('myImage').src='https://s3-us-west-2.amazonaws.com/s.cdpn.io/93927/pic_bulboff.gif'">Turn off the light</button>
```

```
</body>
```

```
</html>
```

# What can JavaScript do? – Example 2

**Output:**

## What Can JavaScript Do?

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.



Turn on the light

Turn off the light

## What Can JavaScript Do?

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.



Turn on the light

Turn off the light

# Where to Insert JavaScript?

- **JavaScript in Head**

```
<html>
<head>
<script>
function myFunction()
{
document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>
<h2>JavaScript in Head</h2>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>
</body>
</html>
```

# *Where to Insert JavaScript?*

## Output:

### JavaScript in Head

A Paragraph.

Try it

### JavaScript in Head

Paragraph changed.

Try it

# Where to Insert JavaScript?

- *JavaScript in Body*

```
<html>
```

```
<body>
```

```
<h2>JavaScript in Body</h2>
```

```
<p id="demo">A Paragraph.</p>
```

```
<button type="button" onclick="myFunction()">Try it</button>
```

```
<script>
```

```
function myFunction()
```

```
{
```

```
  document.getElementById("demo").innerHTML = "Paragraph changed.";
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```



# *Where to Insert JavaScript?*

## Output:

### JavaScript in Body

A Paragraph.

Try it

### JavaScript in Body

Paragraph changed.

Try it

# Where to Insert JavaScript?

- *JavaScript in both Head & Body*

```
<html>
```

```
<head>
```

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

```
function message()
```

```
{alert("This alert box was called with the onload event"); }
```

```
</script>
```

```
</head>
```

```
<body onload="message()">
```

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

```
document.write("This message is written by JavaScript");
```

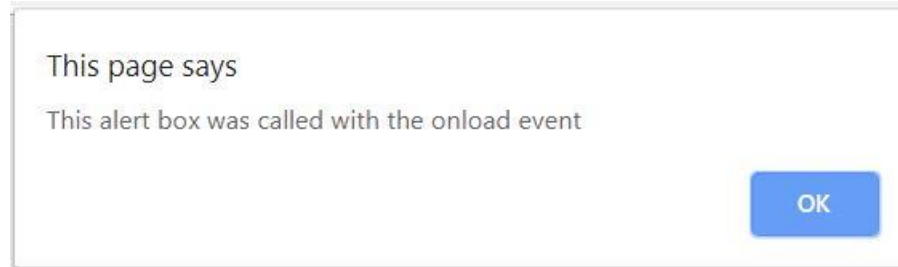
```
</script>
```

```
</body>
```

```
</html>
```

# *Where to Insert JavaScript?*

**Output:**



This message is written by JavaScript

# Using an External JavaScript

- If you want to run the same JavaScript on several pages, without having to write the same script on every page, you can write a JavaScript in an external file and save the file with a **.js** file extension.

- The external script cannot contain the `<script></script>` tags.

```
function myFunction()  
{ document.getElementById("demo").innerHTML = "Paragraph changed."; }
```

- Using an External JavaScript

```
<body>  
<h2>External JavaScript</h2>  
<p id="demo">A Paragraph.</p>  
<button type="button" onclick="myFunction()">Try it</button>  
<p>(myFunction is stored in an external file called "myScript.js")</p>  
<script src="myScript.js"></script>  
</body>
```

# JavaScript Statements

- *JavaScript statements are commands to the browser.*

```
<p id="demo"></p>
```

```
<script>
```

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```

```
</script>
```

- *JavaScript code is a sequence of statements.*

```
<p id="demo"></p>
```

```
<script>
```

```
var x, y, z; // Statement 1
```

```
x = 5; // Statement 2
```

```
y = 6; // Statement 3
```

```
z = x + y; // Statement 4
```

```
document.getElementById("demo").innerHTML = "The value of z is " + z +  
".";
```

```
</script>
```

# JavaScript Statements

- *JavaScript statements are separated with semicolon.*

```
<p id="demo1"></p>
```

```
<script>
```

```
var a, b, c;
```

```
a = 5;
```

```
b = 6;
```

```
c = a + b;
```

```
document.getElementById("demo1").innerHTML = c;
```

```
</script>
```

# JavaScript Statements

- *Multiple statement on one line is allowed.*

```
<p id="demo1"></p>
```

```
<script>
```

```
var a, b, c; a = 5; b = 6; c = a + b;
```

```
document.getElementById("demo1").innerHTML = c;
```

```
</script>
```

- *JavaScript statements can be grouped together in code blocks.*

```
<button type="button" onclick="myFunction()">Click Me!</button>
```

```
<p id="demo1"></p>
```

```
<p id="demo2"></p>
```

```
<script>
```

```
function myFunction()
```

```
{    document.getElementById("demo1").innerHTML = "Hello Dolly!";
```

```
    document.getElementById("demo2").innerHTML = "How are you?";
```

```
}
```

```
</script>
```

# JavaScript Keywords

- In JavaScript you cannot use these reserved words as variables, labels, or function names:*

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield



# JavaScript Data Types

- *JavaScript data types are dynamic. i.e. the same variable can be used to hold different data types.*

```
var x;           // Now x is undefined
x = 5; // Now x is a Number
x = "John";      // Now x is a String
```

- *JavaScript evaluates expressions from left to right.*

- *Different sequences can produce different results.*

```
var x = 16 + "Volvo"; // results: 16Volvo
var x = "Volvo" + 16; // results: Volvo16
var x = 16 + 4 + "Volvo"; // results: 20Volvo
```

- *In the last example, JavaScript treats 16 and 4 as numbers, until it reaches "Volvo". As a result, it adds 16 & 4 and then concatenates Volvo with the result.*

# JavaScript Variables

- *JavaScript variables are containers for storing data values.*
- *A variable can have a short name, like `x`, or a more descriptive name, like `carname`.*
- *Rules for JavaScript variable names:*
  - *Variable names are case sensitive (`y` & `Y` are two different variables).*
  - *Variable names must begin with a letter or the underscore character.*
- *You can declare JavaScript variables with the `var` statement.*  
*`var x; var x=5;`*  
*`var carname; var carname="volvo";`*
- *After the execution of the statements above, the variable `x` will hold the value `5`, and `carname` will hold the value `volvo`.*

# JavaScript Operators

- *The assignment operator (=) assigns a value to a variable.*

*var x = 10;*

- *The addition operator (+) adds numbers.*

*var x=5; var y=2; var z = x+y;*

- *The multiplication operator (\*) multiplies numbers.*

*var x=5; var y=2; var z = x\*y;*

# JavaScript Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division (Quotient)
%	Modulus (Division Remainder)
++	Increment
--	Decrement

# JavaScript Arithmetic Operators

- The *addition (+)* operator

```
<body>
<p id="demo"></p>
<script>
var x = 5; var y = 2; var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>
</body>
```

- The *increment (++)* operator

```
<body>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
var y = 6;
document.getElementById("demo1").innerHTML = y;
var x = ++y;
document.getElementById("demo2").innerHTML = x;
</script>
</body>
```

# JavaScript Assignment Operators

Operator	Example	Same As
=	$x=y$	$x=y$
+=	$x+=y$	$x=x+y$
-=	$x-=y$	$x=x-y$
*=	$x*=y$	$x=x*y$
/=	$x/=y$	$x=x/y$
%=	$x\%=y$	$x=x\%y$
**=	$x**=y$	$x=x**y$

# JavaScript Assignment Operators

- The `+=` and `**=` operators

`<body>`

`<p id="demo1"></p>`

`<p id="demo2"></p>`

`<script>`

`var x = 10; x += 5;`

`document.getElementById("demo1").innerHTML = x;`

`var y = 5; y **= 3;`

`document.getElementById("demo2").innerHTML = y;`

`</script>`

`</body>`

# JavaScript String Operators

- The `+` operator can also be used to add (concatenate) strings.  
`var txt1="John"; var txt2="Doe"; var txt3=txt1+" "+txt2;`
- The result of `txt3` will be:  
`John Doe`
- The `+=` assignment operator can also be used to add (concatenate) strings.  
`var txt1="What a very "; txt1+="nice day";`
- The result of `txt1` will be:  
`What a very nice day`
- Adding two numbers, will return the sum, but adding a number and a string will return a string.  
`var x=5+5; var y="5"+5; var z="Hello"+5;`
- The result of `x`, `y`, & `z` will be:  
`10      55      Hello5`



# JavaScript Comparison Operators

Operator	Description
<code>==</code>	Equal to
<code>===</code>	Equal value and equal type
<code>!=</code>	Not equal
<code>!==</code>	Not equal value; not equal type
<code>&gt;</code>	Greater than
<code>&lt;</code>	Less than
<code>&gt;=</code>	Greater than or equal to
<code>&lt;=</code>	Less than or equal to
<code>?</code>	Ternary operator

# JavaScript Comparison Operators

- The `===` and `!==` operators

```
<body>
```

```
<p id="demo1"></p>
```

```
<p id="demo2"></p>
```

```
<script>
```

```
var x = 5;
```

```
document.getElementById("demo1").innerHTML = (x === 6);
```

```
var y = 5;
```

```
document.getElementById("demo2").innerHTML = (y !== 6);
```

```
</script>
```

```
</body>
```

# JavaScript Logical Operators

<i>Operator</i>	<i>Description</i>
<code>&amp;&amp;</code>	<i>Logical and</i>
<code>  </code>	<i>Logical or</i>
<code>!</code>	<i>Logical not</i>

# Popup Boxes – Alert Box

- An alert box is often used if you want to make sure information comes through to the user.
- When an alert box pops up, the user will have to click “OK” to proceed.  
*alert(“sometext”);*

- For example:

```
<head>
<script type="text/javascript">
function show_alert()
{alert("I am an alert box!"); }
</script>
</head>
<body>
<input type="button" onclick="show_alert()" value="Show alert box" />
</body>
```

# *Popup Boxes – Alert Box*

**Output:**

Show alert box

This page says  
I am an alert box!

OK

# Popup Boxes – Confirmation Box

- *A confirm box is often used if you want the user to verify or accept something.*
- *When a confirm box pops up, the user will have to click either “OK” or “Cancel” to proceed.*
- *If the user clicks “OK”, the box returns true; and if the user clicks “Cancel”, the box returns false.*

*confirm(“sometext”);*

# Popup Boxes – Confirmation Box

- For example:

```
<head>
```

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

```
function show_confirm()
```

```
{
```

```
var r=confirm("Press a button");
```

```
if (r==true) { document.write("You pressed OK!"); }
```

```
else { document.write("You pressed Cancel!"); }
```

```
}
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<input type="button" onclick="show_confirm()" value="Show confirm box" />
```

```
</body>
```

# Popup Boxes – Confirmation Box

**Output:**

Show confirm box

This page says  
Press a button

OK

Cancel

You pressed OK!

You pressed Cancel!



# Popup Boxes – Prompt Box

- *A prompt box is often used if you want the user to input a value before entering a page.*
- *When a prompt box pops up, the user will have to click either “OK” or “Cancel” to proceed after entering an input value.*
- *If the user clicks “OK”, the box returns the input value; and if the user clicks “Cancel”, the box returns null.*

*prompt(“sometext”, “defaultvalue”);*

# Popup Boxes – Prompt Box

- For example:

```
<body>
```

```
<button onclick="myFunction()">Try it</button>
```

```
<p id="demo"></p>
```

```
<script>
```

```
function myFunction() {
```

```
var txt;
```

```
var person = prompt("Please enter your name:", "Harry Potter");
```

```
if (person == null || person == "") {txt = "User cancelled the prompt.";} 
```

```
else {txt = "Hello " + person + "! How are you today?";}
```

```
document.getElementById("demo").innerHTML = txt;
```

```
}
```

```
</script>
```

```
</body>
```

# Popup Boxes – Prompt Box

## Output:

Try it



This page says

Please enter your name:

Harry Potter

OK Cancel

Try it

Hello Harry Potter! How are you today?

Try it

User cancelled the prompt.

# Functions

- *A function is simply a block of code with a name, which allows the block of code to be called by other components in the scripts to perform certain tasks.*
- *Functions can also accept parameters that they use to complete their task.*
- *JavaScript actually comes with a number of built-in functions to accomplish a variety of tasks.*

- **Creating Custom Functions:**

```
function name_of_function(argument1,argument2,...,arguments)  
{  
    .....  
    // Block of code  
    .....  
}
```

# Functions

- **Calling Functions:**

- *There are two common ways to call a function; from an event handler and from another function.*
- *Calling a function is simple. You have to specify its name followed by the pair of parenthesis*

*<Script Type = "Text / JavaScript">*

*name\_of\_function(argument1, argument2, ..., arguments)*

*< / Script >*

# A Simple Function

```
<html>
<head>
<script>
function myFunction()
{
  document.getElementById("demo").innerHTML = "Hello World!";
}
</script>
</head>
<body>
<p>When you click "Try it", a function will be called.</p>
<p>The function will display a message.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
</body>
</html>
```

# *A Simple Function*

## Output:

When you click "Try it", a function will be called.

The function will display a message.

Try it

When you click "Try it", a function will be called.

The function will display a message.

Try it

Hello World!

# Function that Returns a Value

```
<html>
<body>
<p>This example calls a function which performs a calculation and returns the
result:</p>
<p id="demo"></p>
<script>
var x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;
function myFunction(a, b)
{
    return a * b;
}
</script>
</body>
</html>
```

## Output:

This example calls a function which performs a calculation and returns the result:

12



# JavaScript Objects

- *In JavaScript, almost everything is an object*
- *Booleans, Numbers, and Strings (if defined with the **new** keyword) can be objects.*
- *Dates, Maths, Arrays, Functions, Regular expressions, and Objects are always objects.*
- *All JavaScript values, except primitives, are objects.*
- *A primitive value is a value that has no properties or methods.*
- *A primitive data type is data that has a primitive value.*
- *JavaScript defines 5 types of primitive data types: **string**, **number**, **boolean**, **null**, & **undefined**.*
- *Primitive values are immutable (they are hardcoded and therefore cannot be changed).*

# JavaScript Objects Properties

- JavaScript variables can contain single values.  
*var person = "John Doe";*
- Objects are variables too; but objects can contain many values.
- The values are written as *name:value* pairs (name and value separated by a colon). — *creating an object using object literal*  
*var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};*
- A JavaScript object is a collection of *named values*.
- The named values, in JavaScript objects, are called *properties*.
- In above example, "firstName", "lastName", "age", and "eyeColor" are *properties*; whereas "John", "Doe", "50", and "blue" are their respective *values*.

# Math Object in JavaScript

- The *Math* object allows you to perform mathematical tasks.
- The *Math* object includes several mathematical constants and methods.
- All properties and methods of *Math* can be called by using *Math* as an object without creating it.

```
var pi_value = Math.PI;  
var sqrt_value = Math.sqrt(16);
```

```
<body>
```

```
<h2>JavaScript Math.PI</h2>
```

```
<p>Math.PI returns the ratio of a circle's circumference to its diameter:</p>
```

```
<p id="demo"></p>
```

```
<script>
```

```
document.getElementById("demo").innerHTML = Math.PI;
```

```
</script>
```

```
</body>
```

# Math Object in JavaScript

## Properties

- *Math.E* — returns Euler's number (approx. 2.718)
- *Math.LN2* — returns the natural logarithm of 2 (approx. 0.693)
- *Math.LN10* — returns the natural logarithm of 10 (approx. 2.302)
- *Math.LOG2E* — returns the base-2 logarithm of *E* (approx. 1.442)
- *Math.LOG10E* — returns the base-10 logarithm of *E* (approx. 0.434)
- *Math.PI* — returns *PI* (approx. 3.14159)

## Methods

- *abs(x)* — returns the absolute value of *x*
- *ceil(x)* — returns *x*, rounded upwards to the nearest integer
- *floor(x)* — returns *x*, rounded downwards to the nearest integer
- *log(x)* — returns the natural logarithm (base *E*) of *x*
- *max(x,y,z,...,n)* — returns the number with the highest value
- *min(x,y,z,...,n)* — returns the number with the lowest value
- *pow(x,y)* — returns the value of *x* to the power of *y*
- *sqrt(x)* — returns the square root of *x*
- *random(x)* — returns a random number between 0 & 1
- *round(x)* — rounds *x* to the nearest integer

# Window Object in JavaScript

- The window object represents the browser's window.
- Two properties *window.innerHeight* and *window.innerWidth* can be used to determine the size (in pixels) of browser window.

```
<body>
```

```
<p id="demo"></p>
```

```
<script>
```

```
var w = window.innerWidth;
```

```
var h = window.innerHeight;
```

```
var x = document.getElementById("demo");
```

```
x.innerHTML = "Browser inner window width: " + w + ", height: " + h +  
".";
```

```
</script>
```

```
</body>
```

# Date Object in JavaScript

- *The Date object is used to work with dates and times.*
- *Date objects are created with the Date() constructor.*
- *Date object methods in JavaScript:*
  - *getDate()* — returns the day of the month (from 1 – 31)
  - *getDay()* — returns the day of the week (from 0 – 6)
  - *getFullYear()* — returns the year (four digits)
  - *getHours()* — returns the hour (from 0 – 23)
  - *getMilliseconds()* — returns the milliseconds (from 0 – 999)
  - *getMinutes()* — returns the minutes (from 0 – 59)
  - *getMonth()* — returns the month (from 0 – 11)
  - *getSeconds()* — returns the seconds (from 0 – 59)

# Date Object in JavaScript

```
<html>
<body>
<h2>JavaScript new Date()</h2>
<p id="demo"></p>
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
// document.getElementById("demo").innerHTML = d.getDate();
// document.getElementById("demo").innerHTML = d.getDay();
// document.getElementById("demo").innerHTML = d.getFullYear();
// document.getElementById("demo").innerHTML = d.getHours();
// document.getElementById("demo").innerHTML = d.getMilliseconds();
// document.getElementById("demo").innerHTML = d.getMinutes();
// document.getElementById("demo").innerHTML = d.getMonths();
// document.getElementById("demo").innerHTML = d.getSeconds();
</script>
</body>
</html>
```

# String Methods in JavaScript

- *String methods helps us to work with strings.*
- *The **length** property returns the length of a string.*
- *The **indexOf()** method returns the index of the **first** occurrence of a specified text in a string.*
- *The **search()** method searches a string for a specified value and returns the position of the match.*
- *The **slice()** method extracts a part of a string and returns the extracted part in a new string.*
- *The **replace()** method replaces a specified value with another value in a string.*
- *The **toUpperCase()** method converts a string to upper case.*
- *The **toLowerCase()** method converts a string to lower case.*



# String Methods in JavaScript

```
<html>
<body>
<h2>JavaScript String Properties</h2>
<p id="demo"></p>
<script>
var txt = "The quick brown fox jumps over the lazy dog.";
document.getElementById("demo").innerHTML = txt.length;
/*
var pos = txt.indexOf("fox");           document.getElementById("demo").innerHTML = pos;

var pos = txt.search("jumps");          document.getElementById("demo").innerHTML = pos;

var res = txt.slice(4,9);               document.getElementById("demo").innerHTML = res;

var txt = txt.replace("quick", "slow"); document.getElementById("demo").innerHTML = txt;

var txt = txt.toUpperCase();            document.getElementById("demo").innerHTML = txt;
*/
</script>
</body>
</html>
```

# Number Methods in JavaScript

- The `Number()`, can be used to convert JavaScript variables to numbers.
- The `parseInt()` parses a string and returns a whole number. Spaces are allowed but only the first number is returned.
- The `parseFloat()` parses a string and returns a number. Spaces are allowed but only the first number is returned.
- The `toExponential()` method returns a string representing the number object in exponential notation.
- The `toString()` method returns a string representing the specified object. The `toString()` method parses its first argument, and attempts to return a string representation in the specified radix.
- The `toFixed()` returns a string, with the number written with a specified number of decimals.
- The `toPrecision()` returns a string, with a number written with a specified length.

# Number Methods in JavaScript

```
<script type="text/javascript">  
x = true;  
document.write(Number(x) + "<br>");  
x = false;  
document.write(Number(x) + "<br>");  
x = "10"  
document.write(Number(x) + "<br>");  
x = "10 20"  
document.write(Number(x) + "<br>");  
document.write(parseInt("12") + "<br>");  
document.write(parseInt("12.33") + "<br>");  
document.write(parseInt("12 6") + "<br>");  
document.write(parseInt("12 years") + "<br>");  
document.write(parseInt("month 12"));  
</script>
```

# Number Methods in JavaScript

`<body>`

`<p id="demo"></p>`

`<script>`

`var x = 9.656;`

`document.getElementById("demo").innerHTML =`

`x.toExponential() + "<br>" +`

`x.toExponential(2) + "<br>" +`

`x.toExponential(4) + "<br>" +`

`x.toExponential(6);`

`</script>`

`</body>`

# Number Methods in JavaScript

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

```
var num = new Number(15);
```

```
document.write("num.toString() is: " + num.toString() + "<br />");
```

```
document.write("num.toString(2) is: " + num.toString(2) + "<br />");
```

```
document.write("num.toString(4) is: " + num.toString(4) + "<br />");
```

```
</script>
```

# Number Methods in JavaScript

```
<body>
<p id="demo"></p>
<script>
var x = 9.656;
document.getElementById("demo").innerHTML =
  x.toFixed(0) + "<br>" +
  x.toPrecision() + "<br>" +
  x.toFixed(2) + "<br>" +
  x.toPrecision(2) + "<br>" +
  x.toFixed(4) + "<br>" +
  x.toPrecision(4) + "<br>" +
  x.toFixed(6) + "<br>" +
  x.toPrecision(6);
</script>
</body>
```

# Boolean Methods in JavaScript

- A JavaScript Boolean represents one of two values: **true** or **false**.
- **Boolean(5 > 3)** and **5 > 3** returns same output as true.

<body>

<p>Display the value of Boolean(5 > 3):</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>

function myFunction()

{

document.getElementById("demo").innerHTML = Boolean(5 > 3);

}

</script>

</body>

# Boolean Methods in JavaScript

- Everything with a “*value*” is **TRUE**.

```
<body>
<p id="demo"></p>
<script>
var b1 = Boolean(100);
var b2 = Boolean(3.14);
var b3 = Boolean(-15);
var b4 = Boolean("Hello");
var b5 = Boolean('false');
var b6 = Boolean(1 + 7 + 3.14);
document.getElementById("demo").innerHTML =
"100 is " + b1 + "<br>" +
"3.14 is " + b2 + "<br>" +
"-15 is " + b3 + "<br>" +
"Any (not empty) string is " + b4 + "<br>" +
"Even the string 'false' is " + b5 + "<br>" +
"Any expression (except zero) is " + b6;
</script>
</body>
```



# Boolean Methods in JavaScript

- Everything without a “*value*” is **FALSE**.
- The Boolean value of *0* (zero) is **FALSE**.
- The Boolean value of *-0* (minus zero) is **FALSE**.
- The Boolean value of “ ” (empty string) is **FALSE**.
- The Boolean value of *undefined* is **FALSE**.
- The Boolean value of *null* is **FALSE**.
- The Boolean value of *false* is **FALSE**.
- The Boolean value of *NaN* is **FALSE**.

# Regular Expression in JavaScript

- Regular expressions are patterns used to match character combinations in strings.
- In JavaScript, regular expressions are also objects.
- Regular expressions can be used to perform all types of *text search* and *text replace*.
- There are two main string methods: *search()* and *replace()*.
- *search()* method uses an expression to search for a match, and returns the position of the match.
- *replace()* method uses an expression to return a modified string where the pattern is replaced.

# Regular Expression in JavaScript

*<body>*

*<p id="demo"></p>*

*<script>*

*var str = "The quick brown fox jumps over the lazy dog";*

*var n = str.search(/brown/i);*

*document.getElementById("demo").innerHTML = n;*

*</script>*

*</body>*

# Regular Expression in JavaScript

*<body>*

*<button onclick="myFunction()">Click to replace</button>*

*<p id="demo">The quick brown fox jumps over the lazy dog</p>*

*<script>*

*function myFunction()*

*{*

*var str = document.getElementById("demo").innerHTML;*

*var txt = str.replace(/quick/i, "slow");*

*document.getElementById("demo").innerHTML = txt;*

*}*

*</script>*

*</body>*

# Form Method Property

- *It is used to change the method for sending form data.*
- *The method property sets or returns the value of method attribute in a form.*
- *The syntax for setting the method property is:*  
*`formObject.method = get | post`*
- ***get** — appends the form-data to the URL:*  
*`URL?name=value&name=value` (this is default).*
- ***post** — sends the form-data as an HTTP post transaction.*

# Form Method Property

```
<body>
<h2>Contact Us</h2>
<p>Please fill in this form and send us.</p>
<form action="form2-get.php" method="get">
<p>
<label for="inputName">Name:<sup>*</sup></label>
<input type="text" name="name" id="inputName">
</p>
<p>
<label for="inputEmail">Email:<sup>*</sup></label>
<input type="text" name="email" id="inputEmail">
</p>
<p>
<label for="inputSubject">Subject:</label>
<input type="text" name="subject" id="inputSubject">
</p>
<p>
<label for="inputComment">Message:<sup>*</sup></label>
<textarea name="message" id="inputComment" rows="5" cols="30"></textarea>
</p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>
</body>
```

# Form Method Property

`<body>`

`<h1>ThankYou</h1>`

`<p>Here is the information you have submitted:</p>`

`<ol>`

`<li><em>Name:</em> <?php echo $_GET["name"]?></li>`

`<li><em>Email:</em> <?php echo $_GET["email"]?></li>`

`<li><em>Subject:</em> <?php echo $_GET["subject"]?></li>`

`<li><em>Message:</em> <?php echo $_GET["message"]?></li>`

`</ol>`

`</body>`

# JavaScript Arrays

- An array is a special variable that can hold more than one value at a time.
- A list of items, storing the fruits in a single variable could be like:  
`var fruit1 = "Apple";`  
`var fruit2 = "Banana";`
- This might look possible for few items but isn't feasible for hundreds of item.
- The solution is an array which can replace multiple declaration in a single line like:  
`var fruits = ["Apple", "Banana"];`
- To access an array element, we can refer to the index number like:  
`document.getElementById("demo").innerHTML = fruits[0];`
- The **length** property of an array returns the length of an array.
- The **push()** method is used to add a new element to an array.



# JavaScript Arrays

```
<html>
<body>
<p id="demo"></p>
<script>
var fruits = ["Apple", "Banana", "Cherry", "Orange", "Mango"];
document.getElementById("demo").innerHTML = "The first element of an array is " + fruits[0];
// document.getElementById("demo").innerHTML = "The last element of an array is " +
fruits[fruits.length-1];
// document.getElementById("demo").innerHTML = "The elements of an array are " + fruits;
// document.getElementById("demo").innerHTML = "The length of an array is " + fruits.length;
/*
fruits.push("Lemon");
document.getElementById("demo").innerHTML = "The elements of an array after adding Lemon are "
+ fruits;
*/
</script>
</body>
</html>
```

# JavaScript Object Constructors

- Sometimes, we need a “blueprint” for creating many objects of same “type”.
- In such situation, the better option to create an “object type” is to use an *object constructor function*.
- All the objects of same type are created by calling the constructor function with the **new** keyword.

# JavaScript Object Constructors

```
<html>
<body>
<h2>JavaScript Object Constructors</h2>
<p id="demo"></p>
<script>
  // Constructor function for Person objects
  function Person(first, last, age, eye) { this.firstName = first; this.lastName = last; this.age = age; this.eyeColor = eye; }
  // Create two Person objects
  var myFather = new Person("John", "Doe", 50, "blue");
  var myMother = new Person("Sally", "Rally", 48, "green");
  // Display age
  document.getElementById("demo").innerHTML = "My father is " + myFather.age + ". My mother is " + myMother.age + ".";
  /*
  document.getElementById("demo").innerHTML = "My father name is " + myFather.firstName + " " + myFather.lastName + ".
  He is " + myFather.age + " years old with " + myFather.eyeColor + " eyes. My mother name is " + myMother.firstName + " " +
  myMother.lastName + ". She is " + myMother.age + " years old with " + myMother.eyeColor + " eyes.";
  */
</script>
</body>
</html>
```

# JavaScript Conditionals

- *Conditional statements are used to perform different actions based on different conditions.*
- *In JavaScript we have the following conditional statements:*
  - *Use **if** to specify a block of code to be executed, if a specified condition is true.*
  - *Use **else** to specify a block of code to be executed, if the same condition is false.*
  - *Use **else if** to specify a new condition to test, if the first condition is false.*
  - *Use **switch** to specify many alternative blocks of code to be executed.*

# JavaScript Conditionals (if)

- *Syntax:*

```
if (condition) {  
  // block of code to be executed if the condition is true  
}
```

- *Example:*

```
<body>  
<p>Display "Good day!" if the hour is less than 18:00:</p>  
<p id="demo">Good Evening!</p>  
<script>  
if (new Date().getHours() < 18)  
{document.getElementById("demo").innerHTML = "Good day!";}  
</script>  
</body>
```

# JavaScript Conditionals (else)

- *Syntax:*

```
if (condition) {  
  //block of code to be executed if the condition is true  
} else {  
  //block of code to be executed if the condition is false  
}
```

- *Example:*

```
<body>  
<p>Click the button to display a time-based greeting:</p>  
<button onclick="myFunction()">Try it</button>  
<p id="demo"></p>  
<script>  
function myFunction() {  
  var hour = new Date().getHours();  
  var greeting;  
  if (hour < 18) {  
    greeting = "Good day";  
  } else {  
    greeting = "Good evening";  
  }  
  document.getElementById("demo").innerHTML = greeting;  
}  
</script>  
</body>
```

# JavaScript Conditionals (else if)

- **Syntax:**

```
if(condition1) {  
  //block of code to be executed if the condition1 is true  
} else if(condition2) {  
  //block of code to be executed if the condition1 is false and condition2 is true  
} else {  
  //block of code to be executed if the condition1 is false and condition2 is false  
}
```

- **Example:**

```
<body>  
<p>Click the button to get a time-based greeting:</p>  
<button onclick="myFunction()">Try it</button>  
<p id="demo"></p>  
<script>  
function myFunction() {  
  var greeting;  
  var time = new Date().getHours();  
  if(time < 12) {  
    greeting = "Good morning";  
  } else if(time < 18) {  
    greeting = "Good day";  
  } else {  
    greeting = "Good evening";  
  }  
  document.getElementById("demo").innerHTML = greeting;  
}  
</script>  
</body>
```

# JavaScript Conditionals (switch)

## Syntax:

```
Switch (expression) {  
  case x:  
    //block of code  
    break;  
  case y:  
    //block of code  
    break;  
  default:  
    //block of code  
}
```

## Example

```
<p id="demo"></p>  
<script>  
  var day;  
  switch (new Date().getDay()) {  
    case 0:  
      day = "Sunday";  
      break;  
    case 1:  
      day = "Monday";  
      break;  
    case 2:  
      day = "Tuesday";  
      break;  
    case 3:  
      day = "Wednesday";  
      break;  
    case 4:  
      day = "Thursday";  
      break;  
    case 5:  
      day = "Friday";  
      break;  
    case 6:  
      day = "Saturday";  
  }  
  document.getElementById("demo").innerHTML = "Today is " + day;  
</script>
```



# JavaScript Loops

- *Loops can execute a block of code a number of times.*
- *JavaScript supports different kinds of loops:*
  - *for* — loops through a block of code a number of times
  - *for/in* — loops through the properties of an object
  - *for/of* — loops through the values of an iterable object
  - *while* — loops through a block of code while a specified condition is true
  - *do/while* — also loops through a block of code while a specified condition is true

# JavaScript Loop (for)

- *Syntax:*  
*for (statement1; statement2; statement3) {*  
*//block of code to be executed*  
*}*
- *Statement 1 is executed (one time) before the execution of the code block.*
- *Statement 2 defines the condition for executing the code block.*
- *Statement 3 is executed (every time) after the code block has been executed.*
- *Example:*  
*<body>*  
*<p id="demo"></p>*  
*<script>*  
*var num = "";*  
*var i;*  
*for (i = 0; i < 5; i++)*  
*{ num += "The number is " + i + "<br>"; }*  
*document.getElementById("demo").innerHTML = num;*  
*</script>*  
*</body>*

# JavaScript Loop (for/in)

- *Example:*

```
<body>
```

```
<p id="demo"></p>
```

```
<script>
```

```
var txt = "";
```

```
var person = {fname:"John", lname:"Doe", age:25};
```

```
var x;
```

```
for (x in person)
```

```
{
```

```
txt += person[x] + " ";
```

```
}
```

```
document.getElementById("demo").innerHTML = txt;
```

```
</script>
```

```
</body>
```

# JavaScript Loop (for/of)

- *Example:*

*<body>*

*<script>*

*var cars = ['BMW', 'Volvo', 'Mini'];*

*var x;*

*for (x of cars)*

*{*

*document.write(x + "<br >");*

*}*

*</script>*

*</body>*

# JavaScript Loop (while)

- *Syntax:*  
*while (condition) {*  
*//block of code to be executed*  
*}*
- *Example:*  
*<body>*  
*<p id="demo"></p>*  
*<script>*  
*var text = "";*  
*var i = 0;*  
*while (i < 10)*  
*{*  
*text += "<br>The number is " + i;*  
*i++;*  
*}*  
*document.getElementById("demo").innerHTML = text;*  
*</script>*  
*</body>*

# JavaScript Loop (do/while)

- *Syntax:*

```
do {  
  // block of code to be executed  
} while (condition)
```

- *Example:*

```
<body>  
<p id="demo"></p>  
<script>  
  var text = ""  
  var i = 0;  
  do {  
    text += "<br>The number is " + i;  
    i++;  
  }  
  while (i < 10);  
  document.getElementById("demo").innerHTML = text;  
</script>  
</body>
```

# JavaScript Error Handling

- *When executing JavaScript code, different errors can occur.*
- *Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.*
- *There are four statements to handle errors:*
  - *The **try** statement lets you test a block of code for errors.*
  - *The **catch** statement lets you handle the error.*
  - *The **throw** statement lets you create custom errors.*
  - *The **finally** statement lets you execute code, after try and catch, regardless of the result.*

# JavaScript Error Handling – try, catch, & throw

- The *try* statement allows you to define a block of code to be tested for errors while it is being executed.
- The *catch* statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The *throw* statement allows you to create a custom error.
- If you use *throw* together with *try* and *catch*, you can control program flow and generate custom error messages.



# JavaScript Error Handling – try, catch, & throw

- The JavaScript statements *try* and *catch* come in pairs:

```
try {  
    // block of code to try  
}  
catch(err) {  
    // block of code to handle errors  
}
```

- Example:

```
<body>  
<p id="demo"></p>  
<script>  
try { adddler("Welcome guest!"); }  
catch(err) { document.getElementById("demo").innerHTML = err.message; }  
</script>  
</body>
```

# JavaScript Error Handling – finally

- The *finally* statement lets you execute code, after try and catch, regardless of the result:

- Syntax:

```
try{  
//block of code to try  
}  
catch(err) {  
//block of code to handle errors  
}  
finally{  
//block of code to be executed regardless of the try/catch result  
}
```

# JavaScript Error Handling – try, catch, & throw (Example)

```
<body>
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>
<script>
function myFunction() {
  var message, x;
  message = document.getElementById("p01");
  message.innerHTML = "";
  x = document.getElementById("demo").value;
  try {
    if(x == "") throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10) throw "is too high";
    if(x < 5) throw "is too low";
    if(x >= 5 && x <= 10) throw "is proper input";
  }
  catch(err) { message.innerHTML = x + " " + err; }
  finally { document.getElementById("demo").value = ""; }
}
</script>
</body>
```

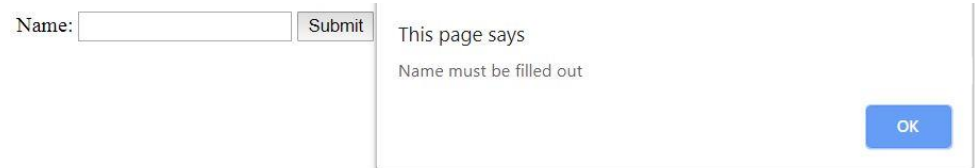
# Form Validation

- *It is the process of checking that a form has been filled in correctly before it is processed or not.*
- *For example, if your form has a box for the user to type their email address, you might want your form handler to check that they've filled in their address before you deal with the rest of the form.*
- *There are two main methods for validating forms: **server-side** (using CGI scripts, ASP, etc.), and **client-side** (usually done using JavaScript).*
- *Server-side validation is more secure but often more tricky to code and it also increases load of server computer, whereas client-side validation is easier to do and quicker too because the browser doesn't have to connect to the server to validate the form, so the user finds out instantly if they've missed out that required field).*
- *Client-side validation also decreases the load of server computer and hence server computer can focus on business logic processing.*

# Form Validation: Checking Non-Empty

```
<html>
<head>
<script>
function validateForm() {
    var x = document.forms["myForm"]["fname"].value;
    if (x == "") {
        alert("Name must be filled out");
        return false;
    }
}
</script>
</head>
<body>
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()" method="post">
    Name: <input type="text" name="fname">
    <input type="submit" value="Submit">
</form>
</body>
</html>
```

**Output:**

A screenshot of a web browser showing a form validation alert. The form has a label "Name:" followed by an empty text input field and a "Submit" button. An alert dialog box is displayed over the form, with the title "This page says" and the message "Name must be filled out". The dialog has an "OK" button.

# Form Validation: Checking Numbers

```
<script type='text/javascript'>
function validate()
{
var patt=/^[0-9]+$ /;
var v= document.getElementById('elem').value;
if(v.match(patt))
{ alert("valid entry"); }
else
{
alert("Invalid entry:");
document.getElementById('elem').value="";
document.getElementById('elem').focus();
}
}
</script>
<form>
Required Field: <input type='text' id='elem' />
<input type='button' onclick="validate()" value='Check' />
</form>
```

## Output:

Required Field:

# Form Validation: Checking All Letters

```
<script type='text/javascript'>
function validate()
{
var patt=/^[a-zA-Z]+$ /;
var v= document.getElementById('elem').value;
if(v.match(patt))
{ alert("valid entry"); }
else
{
alert("Invalid entry:");
document.getElementById('elem').value="";
document.getElementById('elem').focus();
}
}
</script>
<form>
Required Field: <input type='text' id='elem' />
<input type='button' onclick="validate()" value='Check' />
</form>
```

## Output:

Required Field:

# Form Validation: Radio Buttons

```
<script type='text/javascript'>
function validate()
{
var gender=document.getElementsByName("gen");
if(gender[0].checked==false && gender[1].checked==false)
{ alert("You must choose Gender"); }
else {
if(gender[0].checked==true)
alert("Male");
else
alert("Female");
}
}
</script>
<form>
Select Gender:
<input type=radio name='gen'>Male
<input type=radio name='gen'>Female
<input type='button' onclick='validate()' value='Check' />
</form>
```

## Output:

Select Gender: ☐ Male ☐ Female



# Form Validation: Selection Made

```
<script type='text/javascript'>
function validate()
{
var si=document.getElementById('con').selectedIndex;
var v= document.getElementById('con').options[si].text;
if(v=="Please Choose")
{ alert("You must choose the country"); }
else
{ alert("Your Country is:"+v); }
}
</script>
<form>
Select Country: <select id='con'>
<option>Please Choose</option> <option>Nepal</option>
<option>India</option> <option>China</option>
</select>
<input type='button' onclick='validate()' value='Check' />
</form>
```

## Output:

Select Country: Please Choose ▼ Check

This page says  
You must choose the country

OK

Select Country: Nepal ▼ Check

This page says  
Your Country is:Nepal

OK

# Form Validation: Email

- *Every email is made up of 5 parts:*
  - *A combination of letters, numbers, periods, hyphens, plus signs, and/or underscores.*
  - *The @ symbol.*
  - *A combination of letters, numbers, hyphens, and/or periods.*
  - *A period (.).*
  - *The top level domain (com, net, org, gov, ...).*

# Form Validation: Email

```
<script type='text/javascript'>
function validate()
{
var patt=/^[\\w\\-\\.\\+]+\\@[a-zA-Z0-9\\.\\-]+\\. [a-zA-z0-9]{2,4}$ /;
var v= document.getElementById('elem').value;
if(v.match(patt))
{ alert("valid Email"); }
else
{
alert("Invalid Email"); document.getElementById('elem').value="";
document.getElementById('elem').focus();
}
}
</script>
<form>
Email ID: <input type='text' id='elem' />
<input type='button' onclick="validate()" value='Check' />
</form>
```

**Output:**

Email ID:

# Handling Cookies in JavaScript

- Cookies are data, stored in small text files, on your computer.
- When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.
- Cookies were invented to solve the problem “*how to remember information about the user?*”
  - When a user visits a web page, his/her name can be stored in a cookie.
  - Next time the user visits the page, the cookie remembers his/her name.
- Cookies are saved in name-value pairs like: *username = John Doe*
- When a browser requests a web page from a server, cookies belonging to the page are added to the request; this way the server gets the necessary data to remember information about users.
- JavaScript can create, read, and delete cookies with the *document.cookie* property.

# Handling Cookies in JavaScript

- *Examples of cookies are:*
  - **Name Cookie** — *the first time a visitor arrives to your web page, he/she must fill in his/her name. The name is then stored in a cookie. Next time the visitor arrives at your page, he/she could get a welcome message like “Welcome John Doe!”. The name is retrieved from the stored cookie.*
  - **Password Cookie** — *the first time a visitor arrives to your web page, he/she must fill in a password. The password is then stored in a cookie. Next time the visitor arrives at your page, the password is retrieved from the cookie.*
  - **Date Cookie** — *the first time a visitor arrives to your web page, the current date is stored in a cookie. Next time the visitor arrives at your page, he/she could get a message like “Your last visit was on Tuesday July 30, 2019!”. The date is retrieved from the stored cookie.*

# Create a Cookie with JavaScript

- *With JavaScript, a cookie can be created like this:*

```
document.cookie = "username=John Doe";
```

- *You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:*

```
document.cookie = "username=John Doe; expires=Sat, 29 Feb 2020  
12:00:00 UTC";
```

- *With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.*

```
document.cookie = "username=John Doe; expires=Sat, 29 Feb 2020  
12:00:00 UTC; path=/";
```

# Read a Cookie with JavaScript

- *With JavaScript, a cookie can be read like this:*

```
var x = document.cookie;
```

# Change a Cookie with JavaScript

- *With JavaScript, you can change a cookie the same way as you create it:*

```
document.cookie = "username=John Smith; expires=Sat, 29 Feb 2020  
12:00:00 UTC; path=/";
```

- *The old cookie is overwritten*

# Delete a Cookie with JavaScript

- *Deleting a cookie is very simple as you don't have to specify a cookie value when you delete it.*
- *Just set the expires parameter to a passed date:*

```
document.cookie = "username; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";
```

***END OF UNIT ONE***



# Syllabus

## *UNIT 2: Server Side Scripting with Database Connectivity*

- *Introduction to server side scripting*
- *PHP Introduction*
- *Basic PHP Syntax*
- *Comments in PHP*
- *Variables*
- *PHP Operators*
- *Control Structure*
- *Array*
- *For Each Loop*
- *Functions*
- *Form Handling*
- *PHP \$\_POST*
- *PHP \$\_GET*
- *PHP \$\_REQUEST*
- *PHP date() Functions*
- *PHP include File*
- *File Handling*
- *File Uploading*
- *PHP Sessions*
- *Sending Emails*
- *PHP Cookies*

# UNIT 2

## *SERVER SIDE SCRIPTING WITH DATABASE CONNECTIVITY*

# PHP

- *PHP is an acronym for **Hypertext Preprocessor**.*
- *PHP is a widely-used open source scripting language that are executed on the server side.*
- *PHP borrowed its primary syntax from C++ & C.*
- *Many of the programming techniques you've previously learned will work in PHP (assignments, comparisons, loops, etc.) with little to no syntax difference.*
- *There are, however, major changes in how data is manipulated in relationship to C/C++.*

# PHP

- *C/C++ are type-specific languages, requiring the user to define a specific, singular type for each variable that they use.*
- *PHP commonly assigns its variables “**by value**”, meaning a variable takes on the value of the source variable (expression) and is assigned to the destination variable.*
- *A variable can therefore change its type “on the fly”.*
- *Therefore, variables are not declared as they are in most type-specific languages like C/C++.*

# What can you do with PHP?

- *Generate pages and files dynamically.*
- *Create, open, read, write and close files on the server.*
- *Collect data from a web form such as user information, email, phone number, etc.*
- *Send emails to the users of your website.*
- *Send and receive cookies to track the visitor of your website.*
- *Store, delete, and modify information in your database.*
- *Restrict unauthorized access to your website.*
- *Encrypt data for safe transmission over internet.*
- *..... and many more.*

# Advantages of PHP over Other Languages

- **Easy to learn** — PHP is easy to learn and use. For beginner programmers who just started out in web development, PHP is often considered as the preferable choice of language to learn.
- **Open source** — PHP is an open-source project. It is developed and maintained by a worldwide community of developers who make its source code freely available to download and use.
- **Portability** — PHP runs on various platforms such as Microsoft Windows, Linux, Mac OS, etc. and it is compatible with almost all servers used today such as Apache, IIS, etc.
- **Fast Performance** — Scripts written in PHP usually execute or runs faster than those written in other scripting languages like ASP, Ruby, Python, Java, etc.
- **Vast Community** — Since PHP is supported by the worldwide community, finding help or documentation related to PHP online is extremely easy.

# PHP Syntax

- *PHP has quite easy syntax, if you are familiar with any C-type language.*
- *It has all the same structures that you are familiar with other programming languages.*
- *A PHP script can be placed anywhere in the document.*
- *It starts with `<?php` and ends with `?>`*  
*`<?php`*  
*`//PHP codes;`*  
*`?>`*
- *The default file extension for PHP files is `.php`.*

# Simple PHP Example

```
<html>  
<body>  
<h1>My first PHP page</h1>  
<?php  
echo "HelloWorld!";  
echo "<br>";  
print "<br>";  
print "This is my first program in PHP.";  
?>  
</body>  
</html>
```



# Comments in PHP

- To write a single-line comment either start the line with two slashes (//) or a hash symbol (#).
- However to write multi-line comments, start the comment with a slash followed by an asterisk (/\*) and end the comment with an asterisk followed by a slash (\*/).

```
<?php
```

```
// This is a single line comment
```

```
# This is also a single line comment
```

```
/*
```

```
This is a multiple line comment block
```

```
that spans across more than
```

```
one line
```

```
*/
```

```
echo "Hello, world!";
```

```
?>
```

# Case Sensitivity in PHP

- *Variable names in PHP are case-sensitive.*
- *As a result the variables \$color, \$Color, and \$COLOR are treated as three different variables.*
- *However the keywords, functions and classes name are case-insensitive.*
- *As a result calling the gettype() or GETTYPE() produce the same result.*

# PHP Variables

- *In PHP, a variable does not need to be declared before adding a value to it.*
- *PHP automatically converts the variable to the correct data type, depending on its value.*
- *After declaring a variable it can be reused throughout the code.*
- *The assignment operator (=) is used to assign value to a variable.*
- *In PHP, variable can be declared as*  
*`$var_name = value;`*

# PHP Variables

```
<html>
<body>
<h1>Declaring Variables</h1>
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
echo $txt;
echo "<br>";
echo $x;
echo "<br>";
echo $y;
?>
</body>
</html>
```

# Naming Conventions for PHP Variables

- *All variables in PHP start with a \$ sign, followed by the name of the variable.*
- *A variable name must start with a letter or the underscore character .*
- *A variable name cannot start with a number.*
- *A variable name in PHP can only contain alpha-numeric characters and underscores (A-z, 0-9, and ).*
- *A variable name cannot contain spaces.*

# Defining a Constant in PHP

- *A constant is a name or an identifier for a fixed value.*
- *Constants are like variables, except that once they are defined, they cannot be undefined or changed.*
- *Constants are very useful for storing data that doesn't change while the script is running.*
- *Common examples of such data include configuration settings such as database username and password, website's base URL, company name, etc.*
- *Constants are defined using PHP's **define()** function, which accepts two arguments: the name of the constant, and its value.*
- *Naming conventions for PHP Constants are similar to that of PHP Variables.*

# Defining a Constant in PHP

```
<html>
```

```
<body>
```

```
<h1>Defining a Constant</h1>
```

```
<?php
```

```
define("SITE_URL", "https://www.texasintl.edu.np/");
```

```
echo "Thank you for visiting - " . SITE_URL;
```

```
?>
```

```
</body>
```

```
</html>
```

# PHP echo & print Statements

- The *echo* statement can output one or more strings.
- In general terms, the echo statement can display anything that can be displayed to the browser, such as string, numbers, variables values, the results of expressions, etc.
- Like echo, you can also use the *print* statement to display output to the browser.
- Both *echo* & *print* statement works exactly the same way except that the print statement can only output one string, and always returns 1.
- That's why the echo statement is considered marginally faster than the print statement since it doesn't return any value.
- Since echo and print both are language construct and not actually a function (like if statement), you can use it without parentheses. For example. *echo* or *echo()*, and *print* or *print()*.



# PHP Data Types

- *The values assigned to a PHP variable may be of different data types including simple string and numeric types to more complex data types like arrays and objects.*
- *PHP supports total eight primitive data types:*
  - *Integer*
  - *Float*
  - *String*
  - *Booleans*
  - *Array*
  - *Object*
  - *Resource &*
  - *NULL*

# PHP Strings

- *A string is a sequence of letters, numbers, special characters and arithmetic values or combination of all.*
- *The simplest way to create a string is to enclose the string characters in single or double quotation marks (' or ").*
- *Strings enclosed in single-quotes are treated almost literally, whereas the strings delimited by the double quotes replaces variables with the string representations of their values as well as specially interpreting certain escape sequences.*
- *The escape-sequence replacements are:*
  - *\n is replaced by the newline character.*
  - *\r is replaced by the carriage-return character.*
  - *\t is replaced by the tab character.*
  - *\\$ is replaced by the dollar sign itself (\$).*
  - *\\" is replaced by a single double-quote (").*
  - *\\ is replaced by a single backslash (\).*

# PHP Strings – Single & Double Quoted

```
<?php
```

```
$my_str = "World";
```

```
// Displays: HelloWorld!
```

```
echo "Hello, $my_str!<br>";
```

```
// Displays: Hello, $my_str!
```

```
echo 'Hello, $my_str!<br>';
```

```
// Displays: Hello\tWorld!
```

```
echo '<pre>Hello\tWorld!</pre>';
```

```
// Displays: Hello World!
```

```
echo "<pre>Hello\tWorld!</pre>";
```

```
// Displays: I'll be back
```

```
echo 'I\'ll be back';
```

```
?>
```

# Calculating the Length of a String `strlen()`

```
<?php
$my_str = 'Welcome to PHP classes';
echo "The length of the string is " . strlen($my_str);
?>
```

# Counting Number of Words in a String `str_word_count()`

```
<?php
$my_str = 'The quick brown fox jumps over the lazy dog.';
echo str_word_count($my_str);
?>
```

# Replacing Text within Strings

## *str\_replace()*

```
<?php  
$my_str = 'If the facts do not fit the theory, change the facts.';  
echo str_replace("facts", "truth", $my_str, $count);  
echo "<br>";  
echo "The text was replaced $count times.";  
?>
```

## Reversing a String – *strrev()*

```
<?php  
$my_str = 'You can do anything, but not everything.';  
echo strrev($my_str);  
?>
```

# Operators in PHP – Arithmetic

Operator	Description	Example	Result
+	Addition	$\$x + \$y$	Sum of $\$x$ and $\$y$
-	Subtraction	$\$x - \$y$	Difference of $\$x$ and $\$y$
*	Multiplication	$\$x * \$y$	Product of $\$x$ and $\$y$
/	Division	$\$x / \$y$	Quotient of $\$x$ and $\$y$
%	Modulus	$\$x \% \$y$	Remainder of $\$x$ and $\$y$

```
<?php
$x = 10;
$y = 4;
echo($x + $y); // Outputs: 14
echo($x - $y); // Outputs: 6
echo($x * $y); // Outputs: 40
echo($x / $y); // Outputs: 2.5
echo($x % $y); // Outputs: 2
?>
```

# Operators in PHP – Assignment

Operator	Description	Example	Is the same as
=	Assign	$\$x = \$y$	$\$x = \$y$
+=	Add and assign	$\$x += \$y$	$\$x = \$x + \$y$
-=	Subtract and assign	$\$x -= \$y$	$\$x = \$x - \$y$
*=	Multiply and assign	$\$x *= \$y$	$\$x = \$x * \$y$
/=	Divide and assign quotient	$\$x /= \$y$	$\$x = \$x / \$y$
%=	Divide and assign modulus	$\$x \% = \$y$	$\$x = \$x \% \$y$

```
<?php
$x = 10;
echo $x;
// Outputs: 10
$x = 20;
$x += 30;
echo $x;
// Outputs: 50
```

```
$x = 50;
$x -= 20;
echo $x;
// Outputs: 30
$x = 5;
$x *= 25;
echo $x;
// Outputs: 125
```

```
$x = 50;
$x /= 10;
echo $x;
// Outputs: 5
$x = 100;
$x %= 15;
echo $x;
// Outputs: 10
?>
```

6/18/2023 3:35 PM

# Operators in PHP – Comparison

Operator	Name	Example	Result
==	Equal	$\$x == \$y$	True if $\$x$ is equal to $\$y$
===	Identical	$\$x === \$y$	True if $\$x$ is equal to $\$y$ ; and they are of the same type
!=	Not equal	$\$x != \$y$	True if $\$x$ is not equal to $\$y$
<>	Not equal	$\$x <> \$y$	True if $\$x$ is not equal to $\$y$
!==	Not Identical	$\$x !== \$y$	True if $\$x$ is not equal to $\$y$ ; or they are not of the same type
<	Less than	$\$x < \$y$	True if $\$x$ is less than $\$y$
>	Greater than	$\$x > \$y$	True if $\$x$ is greater than $\$y$
>=	Greater than or equal to	$\$x >= \$y$	True if $\$x$ is greater than or equal to $\$y$
<=	Less than or equal to	$\$x <= \$y$	True if $\$x$ is less than or equal to $\$y$



# Operators in PHP – Comparison

```
<?php
```

```
$x = 25;
```

```
$y = 35;
```

```
$z = "25";
```

```
var_dump($x == $z); // Outputs: boolean true
```

```
var_dump($x === $z); // Outputs: boolean false
```

```
var_dump($x != $y); // Outputs: boolean true
```

```
var_dump($x !== $z); // Outputs: boolean true
```

```
var_dump($x < $y); // Outputs: boolean true
```

```
var_dump($x > $y); // Outputs: boolean false
```

```
var_dump($x <= $y); // Outputs: boolean true
```

```
var_dump($x >= $y); // Outputs: boolean false
```

```
?>
```

# Operators in PHP

## Increment & Decrement

Operator	Name	Effect
<code>++\$x</code>	Pre-increment	Increments $x$ by one, then returns $x$
<code>\$x++</code>	Post-increment	Returns $x$ , then increments $x$ by one
<code>--\$x</code>	Pre-decrement	Decrements $x$ by one, then returns $x$
<code>\$x--</code>	Post-decrement	Returns $x$ , then decrements $x$ by one

```
<?php
```

```
$x = 10;
```

```
echo ++$x; // Outputs: 11
```

```
echo $x; // Outputs: 11
```

```
$x = 10;
```

```
echo $x++; // Outputs: 10
```

```
echo $x; // Outputs: 11
```

```
$x = 10;
```

```
echo --$x; // Outputs: 9
```

```
echo $x; // Outputs: 9
```

```
$x = 10;
```

```
echo $x--; // Outputs: 10
```

```
echo $x; // Outputs: 9
```

```
?>
```

# Operators in PHP – Logical

Operator	Name	Example	Result
<i>and</i>	<i>And</i>	<i>\$x and \$y</i>	<i>True if both \$x and \$y are true</i>
<i>or</i>	<i>Or</i>	<i>\$x or \$y</i>	<i>True if either \$x or \$y is true</i>
<i>xor</i>	<i>Xor</i>	<i>\$x xor \$y</i>	<i>True if either \$x or \$y is true, but not both</i>
<i>&amp;&amp;</i>	<i>And</i>	<i>\$x &amp;&amp; \$y</i>	<i>True if both \$x and \$y are true</i>
<i>  </i>	<i>Or</i>	<i>\$x    \$y</i>	<i>True if either \$x or \$y is true</i>
<i>!</i>	<i>Not</i>	<i>!\$x</i>	<i>True if \$x is not true</i>

# Operators in PHP – String

Operator	Description	Example	Result
.	Concatenation	<code>\$str1.\$str2</code>	Concatenation of <code>\$str1</code> and <code>\$str2</code>
<code>.=</code>	Concatenation assignment	<code>\$str1.=\$str2</code>	Appends the <code>\$str2</code> to the <code>\$str1</code>

```
<?php
$x = "Hello";
$y = "World!";
echo $x . $y;
// Output: HelloWorld!

$x .= $y;
echo $x;
// Output: HelloWorld!

?>
```

# PHP Conditional Statements

- The *if* statement

```
if (condition){  
    // code to be executed  
}
```

- The *if ... else* statement

```
if (condition){  
    // code to be executed if condition is true  
} else{  
    // code to be executed if condition is false  
}
```

# PHP Conditional Statements

- The *if . . . elseif . . . else* statement

```
if(condition1){  
    //code to be executed if condition1 is true  
} elseif(condition2){  
    //code to be executed if condition2 is true  
} else{  
    //code to be executed if both conditions are false  
}
```

- The *switch . . . case* statement

```
switch(n){  
    case label1:  
        //code to be executed if n=label1  
        break;  
    case label2:  
        //code to be executed if n=label2  
        break;  
    ...  
    default:  
        //code to be executed if n is different from all labels  
}
```

# PHP Conditional Statements – Example

- The *if* statement

```
<?php
$d = date("D");
echo $d;
echo "<br>";
if($d == "Fri"){
    echo "Have a nice weekend!";
}
?>
```

# PHP Conditional Statements – Example

- The *if . . . else* statement

```
<?php
$d = date("D");
if($d == "Fri"){
    echo "Have a nice weekend!";
} else{
    echo "Have a nice day!";
}
?>
```



# PHP Conditional Statements – Example

- The *if ... elseif ... else* statement

```
<?php
$d = date("D");
if($d == "Fri"){
    echo "Have a nice weekend!";
} elseif($d == "Sun"){
    echo "Have a nice Sunday!";
} else{
    echo "Have a nice day!";
}
?>
```

# PHP Conditional Statements – Example

- The *switch ... case* statement

```
<?php
$today = date("D");
switch($today){
    case "Mon":
        echo "Today is Monday. Clean your
house.";
        break;
    case "Tue":
        echo "Today is Tuesday. Buy some
food.";
        break;
    case "Wed":
        echo "Today is Wednesday. Visit a
doctor.";
        break;
    case "Thu":
        echo "Today is Thursday. Repair your
car.";
        break;
```

```
case "Fri":
    echo "Today is Friday. Party
tonight.";
    break;
case "Sat":
    echo "Today is Saturday. Its
movie time.";
    break;
case "Sun":
    echo "Today is Sunday. Do some
rest.";
    break;
default:
    echo "No information available
for that day.";
    break;
}
?>
```

# PHP Ternary Operator (?)

- The *if ... else* statement

```
<?php
```

```
$age=18;
```

```
if($age < 18){
```

```
    echo 'Child'; // Display Child if age is less than 18
```

```
} else{
```

```
    echo 'Adult'; // Display Adult if age is greater than or equal to 18
```

```
}
```

```
?>
```

- The *ternary operator* statement

```
echo ($age < 18) ? 'Child' : 'Adult';
```

# PHP Arrays

- *Arrays are complex variables that allow us to store more than one value or a group of values under a single variable name.*

- *Storing the colors one by one in a variable could look like this:*

```
<?php  
$color1 = "red";  
$color2 = "green";  
$color3 = "blue";  
?>
```

- *But what if you want to store more than three colors?*
- *It is quite hard, boring, and a bad idea to store each colors in a separate variable.*
- *And here array comes into play.*

# PHP Arrays

- *Types of Arrays*
  - *Indexed Array* — array with a numeric key
  - *Associative Array* — array where each key has its own specific value
  - *Multidimensional Array* — array containing one or more arrays within itself

# PHP Arrays – Indexed Arrays

- *An indexed (or numeric) array stores each array element with a numeric index.*

- *Storing the colors one by one in a variable could look like this:*

```
<?php
$color1 = "red";
$color2 = "green";
$color3 = "blue";
?>
```

- *The following example shows how an indexed array is created:*

```
<?php
$colors = array("red", "green", "blue");
echo $colors[3];
?>
```

# PHP Arrays – Associative Arrays

- *In an associative array, the keys assigned to values can be arbitrary and user defined strings.*

- *In the following example the array uses keys instead of index numbers:*

```
<?php
```

```
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);
```

```
?>
```

- *The following example displays the associative array:*

```
<?php
```

```
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);
```

```
foreach($ages as $key_arr=>$val_arr)
```

```
echo $key_arr . "=" . $val_arr . "<br>";
```

```
?>
```

# PHP Arrays – Multidimensional Arrays

- *An array in which each element can also be an array and each element in the sub-array can be an array or further contain array within itself and so on.*

```
<?php
$contacts = array(
    array(
        "name" => "Peter Parker",
        "email" => "peterparker@mail.com",
    ),
    array(
        "name" => "Harry Potter",
        "email" => "harrypotter@mail.com",
    )
);
echo "His/Her name is ".$contacts[0]["name"] . " and email-id is: " . $contacts[0]["email"];
echo "<br>";
echo "His/Her name is ".$contacts[1]["name"] . " and email-id is: " . $contacts[1]["email"];
?>
```



# Sorting PHP Arrays

- *PHP comes with a number of built-in functions designed specifically for sorting array elements in different ways like alphabetically or numerically in ascending or descending order.*
- ***sort()*** and ***rsort()*** — *for sorting indexed arrays*
- ***asort()*** and ***arsort()*** — *for sorting associative arrays by value*
- ***ksort()*** and ***krsort()*** — *for sorting associative arrays by key*

# Sorting PHP Arrays

- **Sorting Indexed Arrays**

```
<?php
// Define array
$colors = array("Red", "Green", "Blue", "Yellow");
// Sorting and printing array
sort($colors);    // Ascending Order
print_r($colors);
echo "<br>";
rsort($colors);   // Descending Order
print_r($colors);
?>
```

- **Output**

```
Array ( [0] => Blue [1] => Green [2] => Red [3] => Yellow )
Array ( [0] => Yellow [1] => Red [2] => Green [3] => Blue )
```

# Sorting PHP Arrays

- **Sorting Associative Arrays by Value**

```
<?php
// Define array
$age = array("Peter"=>20, "Harry"=>14, "John"=>45, "Clark"=>35);
// Sorting array by value and print
asort($age);    // Ascending Order
print_r($age);
echo "<br>";
arsort($age);    // Descending Order
print_r($age);
?>
```

- **Output**

```
Array ( [Harry] => 14 [Peter] => 20 [Clark] => 35 [John] => 45 )
Array ( [John] => 45 [Clark] => 35 [Peter] => 20 [Harry] => 14 )
```

# Sorting PHP Arrays

- **Sorting Associative Arrays by Key**

```
<?php
// Define array
$age = array("Peter"=>20, "Harry"=>14, "John"=>45, "Clark"=>35);
// Sorting array by key and print
ksort($age);    // Ascending Order
print_r($age);
echo "<br>";
krsort($age);    // Descending Order
print_r($age);
?>
```

- **Output**

```
Array ( [Clark] => 35 [Harry] => 14 [John] => 45 [Peter] => 20 )
Array ( [Peter] => 20 [John] => 45 [Harry] => 14 [Clark] => 35 )
```

# PHP Loops

- *Loops are used to execute the same block of code again and again, until a certain condition is met.*
- *The basic idea behind a loop is to automate the repetitive tasks within a program to save the time and effort.*
- *PHP supports four different types of loops:*
  - *while* — *loops through a block of code until the condition is evaluate to true.*
  - *do ... while* — *the block of code executes once and then condition is evaluated. If the condition is true the statement is repeated as long as the specified condition is true.*
  - *for* — *loops through a block of code until the counter reaches a specified number.*
  - *foreach* — *loops through a block of code for each element in an array.*

# PHP Loops – while

- *Syntax:*

```
while(condition){  
    // Code to be executed  
}
```

- *Example:*

```
<?php  
$i = 1;  
while($i <= 3){  
    $i++;  
    echo "The number is " . $i . "<br>";  
}  
?>
```

# PHP Loops – do...while

- *Syntax:*

```
do{  
    // Code to be executed  
}  
while(condition);
```

- *Example:*

```
<?php  
$i = 1;  
do{  
    $i++;  
    echo "The number is " . $i . "<br>";  
}  
while($i <= 3);  
?>
```

# PHP Loops – for

- *Syntax:*

```
for(initialization; condition; increment){  
    // Code to be executed  
}
```

- *Example:*

```
<?php  
for($i=1; $i<=3; $i++){  
    echo "The number is " . $i . "<br>";  
}  
?>
```



# PHP Loops – foreach

- *Syntax:*

```
foreach($array as $value){  
    // Code to be executed  
}
```

- *Example:*

```
<?php  
$colors = array("Red", "Green", "Blue");  
foreach($colors as $value){  
    echo $value . "<br>";  
}  
?>
```

# PHP Functions

- *A function is a self-contained block of code that performs a specific task.*
- *PHP has a huge collection of internal or built-in functions that you can call directly within your PHP scripts to perform a specific task, like `gettype()`, `print_r()`, `var_dump`, etc.*
- *In addition to the built-in functions, PHP also allows you to define your own functions.*
- *It is a way to create reusable code packages that perform specific tasks and can be kept and maintained separately from main program.*

# PHP Functions – Advantages

- *Functions reduces the repetition of code within a program*
  - *It allows you to extract commonly used block of code into a single component.*
  - *Now you can perform the same task by calling this function wherever you want within your script without having to copy and paste the same block of code again and again.*
- *Functions makes the code much easier to maintain*
  - *Since a function created once can be used many times, so any changes made inside a function automatically implemented at all the places without touching the several files.*

# PHP Functions – Advantages

- *Functions makes it easier to eliminate the errors*
  - *When the program is subdivided into functions, if any error occur you know exactly what function is causing the error and where to find it.*
  - *Therefore, fixing errors becomes much easier.*
- *Functions can be reused in other application*
  - *A function is separated from the rest of the script*
  - *So it's easy to reuse the same function in other applications just by including the php file containing those functions.*

# PHP Functions – Example

```
<?php
```

```
// Defining function
```

```
function whatIsToday(){  
    echo "Today is " . date("l");
```

```
}
```

```
whatIsToday();
```

```
?>
```

# PHP Functions (with parameters) – Example

```
<?php
```

```
// Defining function
```

```
function getSum($num1, $num2){
```

```
$sum = $num1 + $num2;
```

```
echo "Sum of the two numbers $num1 and $num2 is : $sum";
```

```
}
```

```
// Calling function
```

```
getSum(10, 20);
```

```
?>
```

# PHP date() method

- d* - The day of the month (from 01 to 31)
- D* - A textual representation of a day (three letters)
- l* (lowercase 'l') - A full textual representation of a day
- w* - A numeric representation of the day (0 for Sunday, 6 for Saturday)
- z* - The day of the year (from 0 through 365)
- F* - A full textual representation of a month (January through December)
- m* - A numeric representation of a month (from 01 to 12)
- M* - A short textual representation of a month (three letters)
- n* - A numeric representation of a month, without leading zeros (1 to 12)
- t* - The number of days in the given month
- L* - Whether it's a leap year (1 if it is a leap year, 0 otherwise)
- Y* - A four digit representation of a year
- y* - A two digit representation of a year
- a* - Lowercase am or pm
- A* - Uppercase AM or PM
- g* - 12-hour format of an hour (1 to 12)
- G* - 24-hour format of an hour (0 to 23)
- h* - 12-hour format of an hour (01 to 12)
- H* - 24-hour format of an hour (00 to 23)
- i* - Minutes with leading zeros (00 to 59)
- s* - Seconds, with leading zeros (00 to 59)

# PHP GET and POST

- *GET and POST are the methods that send information to Server.*
- *Both GET and POST are HTTP methods*
- *Both methods pass the information differently and have different advantages and disadvantages.*
- *GET method carries request parameter appended in URL string **WHEREAS**, POST method carries request parameter in message body which makes it more secure way of transferring data from client to server.*
- *Both GET and POST are treated as \$\_GET and \$\_POST. These are superglobals, which means that they are always accessible, regardless of scope – and can be accessed from any function, class, or file without having to do anything special.*



# PHP GET Method

- In GET method the data is sent as URL parameters that are usually strings of name and value pairs separated by ampersands (&).
- In general, a URL with GET data will look like this:  
*<http://www.example.com/action.php?name=john&age=24>*
- The bold parts in the URL are the GET parameters and the italic parts are the value of those parameters.
- More than one parameter=value can be embedded in the URL by concatenating with ampersands (&).
- One can only send simple text data via GET method.
- The GET method is restricted to send up to 1024 characters only.

# PHP GET Method

## Advantages & Disadvantages

- Since the data sent by the GET method are displayed in the URL, it is possible to bookmark the page with specific query string values.
- The GET method is not suitable for passing sensitive information such as the username and password, because these are fully visible in the URL query string as well as potentially stored in the client browser's memory as a visited page.
- Because the GET method assigns data to a server environment variable, the length of the URL is limited. So there is a limitation for the total data to be sent.
- PHP provides the super global variable `$_GET` to access all the information sent either through the URL or submitted through an HTML form using the `method="GET"`

# PHP POST Method

- *In POST method the data is sent to the server as a package in a separate communication with the processing script.*
- *Data sent through POST method will not be visible in the URL.*
- *There is no restriction on data size to be sent in the POST method.*
- *The POST method can be used to send ASCII as well as binary data.*
- *The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using secure HTTP you can make sure that your information is secure.*

# PHP POST Method

## Advantages & Disadvantages

- *It is more secure than GET because user-entered information is never visible in the URL query string or in the server logs.*
- *There is a much larger limit on the amount of data that can be passed and one can send text data as well as binary data (uploading a file) using POST.*
- *Since the data sent by the POST method is not visible in the URL, it is not possible to bookmark the page with specific query.*
- *PHP provides the super global variable **`$_POST`** to access all the information sent via post method or submitted through an HTML form using the **`method="POST"`***

# Handling Errors

- *Sometimes your application will not run as it is supposed to do, resulting an error.*
- *There are number of reasons that may cause errors, for example:*
  - *The web server might run out of disk space.*
  - *A user might have entered an invalid value in a form field.*
  - *The file or database record that you were trying to access may not exist.*
  - *The application might not have permission to write to a file on the disk.*
  - *A service that the application needs to access might be temporarily unavailable.*

# Handling Errors

- *These types of errors are known as runtime errors, because they occur at the time the script runs.*
- *They are distinct from syntax errors that needs to be fixed before the script will run.*
- *A professional application must have the capabilities to handle such runtime error gracefully.*
- *Usually this means informing the user about the problem more clearly and precisely.*

# Handling Errors – die() Function

```
<?php
```

```
// Try to open a non-existent file
```

```
$file = fopen("sample.txt", "r");
```

```
?>
```

- *If the file does not exist you will get an error like this:*

**Warning:** *fopen(sample.txt): failed to open stream: No such file or directory in C:\xampp1\htdocs\project\DEMO.php on line 3*

# Handling Errors – die() Function

- *If we follow some simple steps we can prevent the users from getting such error message.*

```
<?php
if(file_exists("sample.txt")){
    $file = fopen("sample.txt", "r");
} else{
    die("Error:The file you are trying to access doesn't exist.");
}
?>
```

- *Now if you run the above script, you will get the error message like this:*

*Error:The file you are trying to access doesn't exist.*



# Sessions & State

- *A session is a way to store information (in variables) to be used across multiple pages.*
- *Unlike a cookie, the information is not stored on the users computer.*
- *When you work with an application, you open it, do some changes, and then you close it. This is much like a session.*
- *The computer knows who you are and it knows when you start application and when you end it.*
- *But on the internet there is a problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain **state**.*
- *Session variables solve this problem by storing user information to be used across multiple pages.*
- *By default, session variables last until the user closes the browser.*
- *So, session variables hold information about one single user, and are available to all pages in one application.*

# PHP Session – Start

- A session is started with the `session_start()` function.
- The `session_start()` function must be the very first thing in your document before any HTML tags.

```
<?php
```

```
// Start the session
```

```
session_start();
```

```
?>
```

```
<?php
```

```
// Set session variables
```

```
$_SESSION["favcolor"] = "green";
```

```
$_SESSION["favanimal"] = "cat";
```

```
echo "Session variables are set.";
```

```
?>
```

# PHP – Get Session Variable Values

- We create another page and will access the session information that we set on the first page.

```
<?php
session_start();
?>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . "<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
```

- Another way to see all the session variable values that has been set:

```
<?php
session_start();
?>

<?php
print_r($_SESSION);
?>
```

# PHP Session – Modify

- *To change a session variable, just overwrite it.*

```
<?php
```

```
session_start();
```

```
?>
```

```
<?php
```

```
// to change a session variable, just overwrite it
```

```
$_SESSION["favcolor"] = "yellow";
```

```
print_r($_SESSION);
```

```
?>
```

# PHP Session – Destroy

- To remove all global session variables and destroy the session, we use *session\_unset()* and *session\_destroy()*.

```
<?php
```

```
session_start();
```

```
?>
```

```
<?php
```

```
// remove all session variables
```

```
session_unset();
```

```
// destroy the session
```

```
session_destroy();
```

```
echo "All session variables are now removed, and the session is destroyed."
```

```
?>
```

# Database Connectivity

- *In order to store or access the data inside a MySQL database, you first need to connect to the MySQL database server.*
- *PHP offers two different ways to connect to MySQL server: **MySQLi** (Improved MySQL) and **PDO** (PHP Data Objects) extensions.*
- *While the PDO extension is more portable and supports more than twelve different databases, MySQLi extension as the name suggests supports MySQL database only.*
- *MySQLi extension however provides an easier way to connect to, and execute queries on, a MySQL database server.*

# Connecting to MySQL Database

- In PHP, you can easily connect to MySQL database using the *mysqli\_connect()* function.
- All communication between PHP and the MySQL database server takes place through this connection.
- The connection to the MySQL database server will be closed automatically as soon as the execution of the script ends.
- However, if you want to close it earlier, you can do this by simply calling the PHP *mysqli\_close()* function.

# Connecting to MySQL Database

- *Basic Syntax (MySQLi, Procedural way)*

```
$link = mysqli_connect("hostname", "username", "password", "database");
```

- *Basic Syntax (MySQLi, Object Oriented way)*

```
$mysqli = new mysqli("hostname", "username", "password", "database");
```

- *Basic Syntax (PHP Data Objects way)*

```
$pdo = new PDO("mysql:host=hostname;dbname=database", "username", "password");
```



# Connecting to MySQL Database

```
<?php
$link = mysqli_connect("localhost", "root", "");
// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Print host information
echo "Connect Successfully. Host info: " . mysqli_get_host_info($link);
// Close connection
mysqli_close($link);
?>
```

# PHP MySQL Create Database

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt create database query execution
$sql = "CREATE DATABASE demo";
if(mysqli_query($link, $sql)){
echo "Database created successfully";
} else{
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

# PHP MySQL Create Table

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt create table query execution
$sql = "CREATE TABLE persons(
id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
first_name VARCHAR(30) NOT NULL,
last_name VARCHAR(30) NOT NULL,
email VARCHAR(70) NOT NULL UNIQUE
)";
if(mysqli_query($link, $sql)){
    echo "Table created successfully. ";
} else{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

# PHP MySQL Insert Data

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt insert query execution
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES ('Peter', 'Parker', 'peterparker@mail.com')";
if(mysqli_query($link, $sql)){
echo "Records inserted successfully.";
} else{
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

# PHP MySQL Insert Multiple Data

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt insert query execution
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES
('John', 'Rambo', 'johnrambo@mail.com'),
('Clark', 'Kent', 'clarkkent@mail.com'),
('John', 'Carter', 'johncarter@mail.com'),
('Harry', 'Potter', 'harrypotter@mail.com')";
if(mysqli_query($link, $sql)){
    echo "Records added successfully.";
} else{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

# PHP MySQL Select Query

```
<?php
/* Attempt MySQL server connection. Assuming you are
running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt select query execution
$sql = "SELECT * FROM persons";
if($result = mysqli_query($link, $sql)){
if(mysqli_num_rows($result) > 0){
echo "<table>";
echo "<tr>";
echo "<th>id</th>";
echo "<th>first_name</th>";
echo "<th>last_name</th>";
echo "<th>email</th>";
echo "</tr>";
```

```
while($row = mysqli_fetch_array($result)){
echo "<tr>";
echo "<td>" . $row['id'] . "</td>";
echo "<td>" . $row['first_name'] . "</td>";
echo "<td>" . $row['last_name'] . "</td>";
echo "<td>" . $row['email'] . "</td>";
echo "</tr>";
}
echo "</table>";
// Free result set
mysqli_free_result($result);
} else{
echo "No records matching your query were found.";
}
} else{
echo "ERROR: Could not able to execute $sql. " .
mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

# PHP MySQL Update Data in Table

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt update query execution
$sql = "UPDATE persons SET email='peterparker_new@mail.com' WHERE id=1 ";
if(mysqli_query($link, $sql)){
echo "Records were updated successfully.";
} else {
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

# PHP MySQL Delete Data in Table

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt delete query execution
$sql = "DELETE FROM persons WHERE first_name='John'";
if(mysqli_query($link, $sql)){
echo "Records were deleted successfully.";
} else{
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```



# PHP MySQL Delete Table

```
<?php
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt delete table query execution
$sql = "DROP TABLE persons;";
if(mysqli_query($link, $sql)){
    echo "Table deleted successfully.";
} else{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

# Cookies

- *A cookie is a small text file that lets you store a small amount of data (nearly 4KB) on the user's computer.*
- *They are typically used to keeping track of information such as username that the site can retrieve to personalize the page when user visits the website next time.*
- *Each time the browser requests a page to the server, all the data in the cookie is automatically sent to the server within the request.*

# Setting a Cookie in PHP

- The *setcookie()* function is used to set a cookie in PHP.
- Make sure you call the *setcookie()* function before any output generated by your script otherwise cookie will not set.
- The basic syntax of this function is as follow:  
*setcookie(name, value, expire, path, domain, secure);*
- If the expiration time of the cookie is set to 0, or omitted, the cookie will expire at the end of the session. i.e. when the browser closes.

```
<?php  
// Setting a cookie  
setcookie("username", "John Carter", time()+30*24*60*60);  
echo "Cookie has been set.";  
?>
```

# Setting a Cookie in PHP

- All the arguments except the name are optional.
- You may also replace an argument with an empty string ("") in order to skip that argument, however to skip the expire argument use a zero (0) instead, since it is an integer.

Parameter	Description
name	The name of the cookie.
value	The value of the cookie.
expires	The expiry date of the cookie. After this time, cookie will become inaccessible. The default value is 0.
path	Specify the path on the server for which the cookie will be available. If set to / , the cookie will be available within the entire domain.
domain	Specify the domain for which the cookie is available. E.g. <a href="http://www.example.com">www.example.com</a>
secure	Indicates that the cookie should be sent only if a secure HTTPS connection exists.

# Accessing Cookies Values in PHP

- The PHP `$_COOKIE` superglobal variable is used to retrieve a cookie value.
- The individual cookie value can be accessed using standard array notation.

```
<?php
// Accessing an individual cookie value
echo $_COOKIE["username"];
?>
```

- An example for accessing cookie:

```
<?php
// Verifying whether a cookie is set or not
if(isset($_COOKIE["username"])){
    echo "Hi " . $_COOKIE["username"];
} else {
    echo "Welcome Guest!";
}
?>
```

# Removing Cookies in PHP

- You can delete a cookie by calling the same `setcookie()` function with the cookie name and any value (such as an empty string).

- However this time you need to set the expiration date in the past.

```
<?php
```

```
// Deleting a cookie
```

```
setcookie("username", "", time()-3600);
```

```
?>
```

- You should pass exactly the same path, domain, and other arguments that you have used when you first created the cookie in order to ensure that the correct cookie is deleted.

# *File Handling in PHP – Opening*

- The PHP *fopen()* function is used to open a file.
- The basic syntax of this function is:  
*fopen(filename, mode)*
- The first parameter passed to *fopen()* specifies the name of the file you want to open and the second parameter specifies which mode the file should be opened.
- For example:  

```
<?php  
$handle = fopen("data.txt", "r");  
echo "File opened successfully. ";  
?>
```

# File Handling in PHP – Opening

- If you try to open a file that doesn't exist, PHP will generate a warning message.
- To avoid these error messages you should always implement a simple check whether a file or directory exists or not before trying to access it, with the PHP `file_exists()` function.

```
<?php
$file = "data.txt";
// Check the existence of file
if(file_exists($file)){
    echo "The file $file exists. " . "<br>";
    // Attempt to open the file
    $handle = fopen($file, "r") or die("ERROR: Cannot open the file.");
    if($handle){
        echo "File opened successfully.";
        // Closing the file handle
        fclose($handle);
    }
} else{
    echo "ERROR: The file $file does not exist.";
}
?>
```



# File Handling in PHP – Opening

- The file may be opened in one of the following modes:

Modes	What it does
<i>r</i>	<i>Open the file for reading only</i>
<i>r+</i>	<i>Open the file for reading and writing</i>
<i>w</i>	<i>Open the file for writing only and clears the contents of file. If the file doesn't exist, PHP will attempt to create it.</i>
<i>w+</i>	<i>Open the file for reading and writing and clears the contents of file. If the file doesn't exist, PHP will attempt to create it.</i>
<i>a</i>	<i>Append. Opens the file for writing only. Preserves file content by writing to the end of the file. If the file doesn't exist, PHP will attempt to create it.</i>
<i>a+</i>	<i>Read / Append. Opens the file for reading and writing. Preserves file content by writing to the end of the file. If the file doesn't exist, PHP will attempt to create it.</i>
<i>x</i>	<i>Opens the file for writing only. Return FALSE and generates an error if the file already exists. If the file doesn't exist, PHP will attempt to create it.</i>
<i>x+</i>	<i>Open the file for reading and writing; otherwise it has the same behavior as 'x'</i>

# *File Handling in PHP – Closing*

- *Once you've finished working with a file, it needs to be closed.*
- *The **fclose()** function is used to close the file, as seen in the previous example.*
- *Although PHP automatically closes all open files when script terminates, but it's a good practice to close a file after performing all the operations.*

# ***File Handling in PHP – Reading***

- The *fread()* function can be used to read a specified number of characters from a file.
- The basic syntax of this function can be given as:  
*fread(file handle, length in bytes)*
- This function takes two parameter — a file handle and the number of bytes to read.
- The following example reads 20 bytes from the “data.txt” file including spaces.  
*fread(\$handle, "20");*
- The *fread()* function can be used in conjugation with the *fread()* function to read the entire file at once.
- The *fread()* function returns the size of the file in bytes.  
*fread(\$handle, filesize(\$file));*

# ***File Handling in PHP – Reading***

- *The easiest way to read the entire contents of a file in PHP is with the **readfile()** function.*

- *This function allows you to read the contents of a file without needing to open it.*

***readfile(\$file)***

- *Another way to read the whole content of a file without needing to open it is with the **file\_get\_contents()** function.*

***file\_get\_contents(\$file)***

# File Handling in PHP – Reading

```
<?php
$file = "data.txt";
// Check the existence of file
if(file_exists($file)){
    // Open the file for reading
    $handle = fopen($file, "r");
    // Reading the entire file
    $content = fread($handle, filesize($file));
    // Display the file content
    echo $content;
    // Closing the file handle
    fclose($handle);
} else{
    echo "ERROR: File does not exist.";
}
?>
```

```
<?php
$file = "data.txt";
// Check the existence of file
if(file_exists($file)){
    // Reads and outputs the entire file
    readfile($file);
} else{
    echo "ERROR: File does not exist.";
}
?>
```

# *File Handling in PHP – Writing*

- *The **fwrite()** function can be used to write data to a file or append to an existing file using PHP.*
- *The basic syntax of this function can be given as:*  
***fwrite(file handle, string)***
- *This function takes two parameter — a file handle and the string of data that is to be written.*
- *The following example writes the message to the “data.txt” file.*  
***fwrite(\$handle, "This line will be written in the file");***
- *An alternative way is using the **file\_put\_contents()** function which is the counterpart of **file\_get\_contents()** function and provides an easy method of writing the data to a file without needing to open it.*

# *File Handling in PHP – Writing*

```
<?php  
$file = "data.txt";  
// String of data to be written  
$data = "The quick brown fox jumps over the lazy dog."  
// Open the file for writing  
$handle = fopen($file, "w");  
// Write data to the file  
fwrite($handle, $data);  
// Closing the file handle  
fclose($handle);  
echo "Data written to the file successfully."  
?>
```

# File Handling in PHP – Renaming

- The *rename()* function can be used to rename a file or directory using PHP.

```
<?php
```

```
$file = "data.txt";
```

```
// Check the existence of file
```

```
if(file_exists($file)){
```

```
    // Attempt to rename the file
```

```
    if(rename($file, "newfile.txt")){
```

```
        echo "File renamed successfully.";
```

```
    } else{
```

```
        echo "ERROR: File cannot be renamed.";
```

```
    }
```

```
} else{
```

```
    echo "ERROR: File does not exist.";
```

```
}
```

```
?>
```



# *File Handling in PHP – Deleting*

- The *unlink()* function can be used to delete a file or directory using PHP.

```
<?php  
$file = "newfile.txt";  
// Check the existence of file  
if(file_exists($file)){  
    // Attempt to delete the file  
    if(unlink($file)){  
        echo "File removed successfully."  
    } else{  
        echo "ERROR: File cannot be removed."  
    }  
} else{  
    echo "ERROR: File does not exist."  
}  
?>
```

# Form Handling in PHP

```
<html>
<head>
<title>Contact Form</title>
</head>
<body>
<h2>Contact Us</h2>
<p>Please fill in this form and send us.</p>
<form action="process-form.php" method="post">
<p>
<label
for="inputName">Name:<sup>*</sup></label>
<input type="text" name="name" id="inputName">
</p>
<p>
<label
for="inputEmail">Email:<sup>*</sup></label>
<input type="text" name="email" id="inputEmail">
</p>
<p>
<label for="inputSubject">Subject:</label>
<input type="text" name="subject" id="inputSubject">
</p>
<p>
<label for="inputComment">Message:<sup>*</sup></label>
<textarea name="message" id="inputComment" rows="5" cols="30"></textarea>
</p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>
</body>
</html>
```

# Form Handling in PHP

```
<html>
<head>
<title>Contact Form</title>
</head>
<body>
<h1>ThankYou</h1>
<p>Here is the information you have submitted:</p>
<ol>
<li><em>Name:</em> <?php echo $_POST["name"]?></li>
<li><em>Email:</em> <?php echo $_POST["email"]?></li>
<li><em>Subject:</em> <?php echo $_POST["subject"]?></li>
<li><em>Message:</em> <?php echo $_POST["message"]?></li>
</ol>
</body>
</html>
```

# ***DATABASE***

# CREATE TABLE

- *Creating a basic table involves naming the table and defining its columns and each column's data type*
- *The SQL **CREATE TABLE** statement is used to create a new table.*
- *Syntax:*  
*CREATE TABLE table\_name(  
    column1 datatype,  
    column2 datatype,  
    ...  
    columnN datatype,  
    PRIMARY KEY (one or more columns)  
);*

# CREATE TABLE

- ***CREATE TABLE** is the keyword telling the database system what you want to do.*
- *In this case, you want to create a new table.*
- *The unique name or identifier for the table follows the **CREATE TABLE** statement.*
- *Then in brackets comes the list defining each column in the table and what sort of data type it is.*

# CREATE TABLE– Example

```
CREATE TABLE CUSTOMERS(  
    ID INT NOT NULL,  
    NAME VARCHAR (20) NOT NULL,  
    AGE INT NOT NULL,  
    ADDRESS CHAR (25),  
    SALARY DECIMAL (18,2),  
    PRIMARY KEY (ID)  
);
```

- The following code block is an example, which creates a **CUSTOMERS** table with an **ID** as a primary key and **NOT NULL** are the constraints showing that these fields cannot be NULL while creating records in this table.
- You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use the **DESC** command.

# CREATE TABLE– Example

Name	Null?	Type
-----	-----	-----
ID	NOT NULL	NUMBER (38)
NAME	NOT NULL	VARCHAR2 (20)
AGE	NOT NULL	NUMBER (38)
ADDRESS		CHAR (25)
SALARY		NUMBER (18, 2)

- Now, you have *CUSTOMERS* table available in your database which you can use to store the required information related to customers.



# ***DROP or DELETE Table***

- *The SQL **DROP TABLE** statement is used to remove a table definition and all the data, indexes, triggers, constraints, and permission specifications for that table.*
- *NOTE — you should be very careful while using this command because once a table is deleted then all the information available in that table will also be lost forever.*
- *Syntax:*  
***DROP TABLE table\_name;***

# DROP or DELETE Table – Example

- Let us first verify the CUSTOMERS table and then we will delete it from the database.

*DESC CUSTOMERS;*

Name	Null?	Type
-----	-----	-----
ID	NOT NULL	NUMBER (38)
NAME	NOT NULL	VARCHAR2 (20)
AGE	NOT NULL	NUMBER (38)
ADDRESS		CHAR (25)
SALARY		NUMBER (18, 2)

- This means that the CUSTOMERS table is available in the database, so let us now drop it as shown below.

*DROP TABLE CUSTOMERS;*

- Now, if you would try the DESC command, then you will get the following error:

*ERROR:*

*ORA-04043: object CUSTOMERS does not exist*

# INSERT Query

- The SQL **INSERT INTO** statement is used to add new rows of data to a table in the database.
- Syntax:  
**INSERT INTO TABLE\_NAME (column1, column2, ..., columnN) VALUES (value1, value2, ..., valueN);**  
Here, column1, column2, ..., columnN are the names of the columns in the table into which you want to insert the data.
- You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.
- The SQL **INSERT INTO** syntax will be as follows:  
**INSERT INTO TABLE\_NAME VALUES (value1, value2, ..., valueN);**

# INSERT Query – Example

- The following statements would create seven records in the **CUSTOMERS** table:

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (1, 'Anil', 32, 'Kathmandu', 2000.00);
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (2, 'Sandace', 25, 'Pokhara', 1500.00);
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (3, 'Prashant', 23, 'Chitwan', 2000.00);
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (4, 'Santosh', 25, 'Biratnagar', 6500.00);
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (5, 'Shreeram', 28, 'Lumbini', 8500.00);
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (6, 'Krishna', 22, 'Lamjung', 4500.00);
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (7, 'Sonam', 22, 'Bhojpur', 10000.00);
```

# INSERT Query – Example

- The alternative way to create same seven records in **CUSTOMERS** table:

```
INSERT INTO CUSTOMERS VALUES (1, 'Anil', 32, 'Kathmandu', 2000.00);  
INSERT INTO CUSTOMERS VALUES (2, 'Sandace', 25, 'Pokhara', 1500.00);  
INSERT INTO CUSTOMERS VALUES (3, 'Prashant', 23, 'Chitwan', 2000.00);  
INSERT INTO CUSTOMERS VALUES (4, 'Santosh', 25, 'Biratnagar', 6500.00);  
INSERT INTO CUSTOMERS VALUES (5, 'Shreeram', 28, 'Lumbini', 8500.00);  
INSERT INTO CUSTOMERS VALUES (6, 'Krishna', 22, 'Lamjung', 4500.00);  
INSERT INTO CUSTOMERS VALUES (7, 'Sonam', 22, 'Bhojpur', 10000.00);
```

# INSERT Query – Example

- All the statements from earlier slides would produce the following records in the **CUSTOMERS** table as shown below:

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

# SELECT Query

- The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table.
- These result tables are called result-sets.
- Syntax:  
*SELECT column1, column2, ..., columnN FROM table\_name;*  
Here, column1, column2, ... are the fields of a table whose values you want to fetch.
- If you want to fetch all the fields available in the field, then you can use the following syntax:  
*SELECT \* FROM table\_name;*

# SELECT Query – Example

- Consider the CUSTOMERS table having the following records:
- The following code is an example, which would fetch the ID, Name, and Salary fields of the customers available in CUSTOMERS table.  
  
**SELECT ID, NAME, SALARY  
FROM CUSTOMERS;**
- This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	SALARY
1	Anil	2000
2	Sandace	1500
3	Prashant	2000
4	Santosh	6500
5	Shreeram	8500
6	Krishna	4500
7	Sonam	10000



# SELECT Query - EXAMPLE

- If you want to fetch all the fields of the CUSTOMERS table, then you should use the following query:

*SELECT \* FROM CUSTOMERS;*

- This would produce the result as shown below:

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

# WHERE Clause

- The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables.
- If the given condition is satisfied, then only it returns a specific value from the table.
- You should use the **WHERE** clause to filter the records and fetching only the necessary records.
- The **WHERE** clause is not only used in the **SELECT** statement, but it is also used in the **UPDATE**, **DELETE** statement, etc.
- Syntax:  
*SELECT column1, column2, ..., columnN*  
*FROM table\_name*  
*WHERE [condition];*
- You can specify a condition using the comparison or logical operators like **>**, **<**, **=**, **LIKE**, **NOT**, etc.

# WHERE Clause – Example

- Consider the **CUSTOMERS** table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

- The following code is an example which would fetch the ID, Name, and Salary fields from the **CUSTOMERS** table, where the salary is greater than 2000:

```
SELECT ID, NAME, SALARY  
FROM CUSTOMERS  
WHERE SALARY > 2000;
```

- This would produce the following result:

ID	NAME	SALARY
4	Santosh	6500
5	Shreeram	8500
6	Krishna	4500
7	Sonam	10000

# WHERE Clause – Example

- The following query is an example, which would fetch the ID, Name, and Salary fields from the CUSTOMERS table for a customer with the name Shreeram.

```
SELECT ID, NAME, SALARY  
FROM CUSTOMERS  
WHERE NAME='Shreeram';
```

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

- Here, it is important to note that all the strings should be given inside single quotes ( ' '). Whereas, numeric values should be given without any quote as in the earlier example.

ID	NAME	SALARY
5	Shreeram	8500

# AND & OR Conjunctive Operators

- The SQL **AND** & **OR** operators are used to combine multiple conditions to narrow data in an SQL statement.
- These two operators are called as the **conjunctive operators**.
- These operators provide a means to make multiple comparisons with different operators in the same SQL statement.
- The **AND** operator allows the existence of multiple conditions in an SQL statement's **WHERE** clause.
- The **OR** operator is used to combine multiple conditions in an SQL statement's **WHERE** clause.

# The AND Operator

- *Syntax:*

*SELECT column1, column2, ..., columnN*

*FROM table\_name*

*WHERE [condition1] AND [condition2] ... AND [conditionN];*

- *You can combine N number of conditions using the AND operator.*
- *For an action to be taken by the SQL statement, whether it be a transaction or a query, all conditions separated by the AND must be TRUE.*

# The AND Operator – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- Following is an example, which would fetch the ID, Name, & Salary fields from the CUSTOMERS table, where the salary is greater than 2000 and the age is less than 25 years:

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 AND
AGE < 25;
```

- This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	SALARY
6	Krishna	4500
7	Sonam	10000

# The OR Operator

- *Syntax:*

*SELECT column1, column2, ..., columnN*

*FROM table\_name*

*WHERE [condition1] OR [condition2] ... OR [conditionN];*

- *You can combine N number of conditions using the OR operator.*
- *For an action to be taken by the SQL statement, whether it be a transaction or a query, the only any ONE of the conditions separated by the OR must be TRUE.*



# The OR Operator – EXAMPLE

- Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

- Following is an example, which would fetch the ID, Name, & Salary fields from the CUSTOMERS table, where the salary is greater than 2000 or the age is less than 25 years:

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY>2000 OR
AGE<25;
```

- This would produce the following result:

ID	NAME	SALARY
3	Prashant	2000
4	Santosh	6500
5	Shreeram	8500
6	Krishna	4500
7	Sonam	10000

# UPDATE Query

- The SQL **UPDATE** query is used to modify the existing records in a table.
- You can use the **WHERE** clause with the **UPDATE** query to update the selected rows, otherwise all the rows would be affected.
- Syntax:  
*UPDATE table\_name*  
*SET column1=value1, column2=value2, ..., columnN=valueN*  
*WHERE [condition];*
- You can combine *N* number of conditions using the **AND** or the **OR** operators.

# UPDATE Query – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- The following query will update the ADDRESS for a customer whose ID number is 6 in the table  
**UPDATE CUSTOMERS**  
**SET ADDRESS = 'Dang'**  
**WHERE ID = 6;**
- Now, the CUSTOMERS table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Dang	4500
7	Sonam	22	Bhojpur	10000

# UPDATE Query – EXAMPLE

- If you want to modify all the ADDRESS and the SALARY column values in the CUSTOMERS table, you do not need to use the WHERE clause as the UPDATE query would be enough as shown in the following query below:

- The following query will update the ADDRESS for a customer whose ID number is 6 in the table

**UPDATE CUSTOMERS**

**SET ADDRESS = 'Dang', SALARY = 1000.00;**

- Now, the CUSTOMERS table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Dang	1000
2	Sandace	25	Dang	1000
3	Prashant	23	Dang	1000
4	Santosh	25	Dang	1000
5	Shreeram	28	Dang	1000
6	Krishna	22	Dang	1000
7	Sonam	22	Dang	1000

# DELETE Query

- The SQL **DELETE** query is used to delete the existing records from a table.
- You can use the **WHERE** clause with a **DELETE** query to delete the selected rows, otherwise all the records would be deleted.
- Syntax:  
**DELETE FROM table\_name**  
**WHERE [condition];**
- You can combine N number of conditions using **AND** or **OR** operators.

# DELETE Query – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- The following code has a query, which will DELETE a customer, whose ID is 6  
**DELETE FROM CUSTOMERS  
WHERE ID = 6;**
- Now, the CUSTOMERS table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
7	Sonam	22	Bhojpur	10000

# DELETE Query – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- Now if you want to delete all the records from the CUSTOMERS table, you do not need to use the WHERE clause and the DELETE query would be as follows:  
**DELETE FROM CUSTOMERS**
- Now, the CUSTOMERS table would have no any record.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

# LIKE Clause

- The SQL **LIKE** clause is used to compare a value to similar values using wildcard operators.
- There are two wildcards used in conjunction with the LIKE operator.
  - The percent sign (%)
  - The underscore (\_)
- The **percent (%)** sign represents zero, one or multiple characters.
- The **underscore (\_)** sign represents a single number or character.
- These symbols can be used in combinations.



# LIKE Clause

- *Syntax:*

*SELECT FROM table\_name WHERE column LIKE 'XXXX%'*

*or*

*SELECT FROM table\_name WHERE column LIKE '%XXXX%'*

*or*

*SELECT FROM table\_name WHERE column LIKE 'XXXX\_'*

*or*

*SELECT FROM table\_name WHERE column LIKE '\_XXXX'*

*or*

*SELECT FROM table\_name WHERE column LIKE '\_XXXX\_'*

- *You can combine N number of conditions using AND or OR operators.*
- *Here, **XXXX** could be any numeric or string value.*

# LIKE Clause – Example

- The following table has a few examples showing the *WHERE* part having different *LIKE* clause with ‘%’ and ‘\_’ operators:

Sr. No.	Statement & Description
1	<i>WHERE SALARY LIKE ‘200%’</i> - Finds any values that start with 200.
2	<i>WHERE SALARY LIKE ‘%200%’</i> - Finds any values that have 200 in any position.
3	<i>WHERE SALARY LIKE ‘_00%’</i> - Finds any values that have 00 in the second and third positions.
4	<i>WHERE SALARY LIKE ‘2_ _%’</i> - Finds any values that start with 2 and are at least 3 characters in length.
5	<i>WHERE SALARY LIKE ‘%2’</i> - Finds any values that end with 2.
6	<i>WHERE SALARY LIKE ‘_2%3’</i> - Finds any values that have a 2 in the second position and end with a 3.
7	<i>WHERE SALARY LIKE ‘2_ _ _ 3’</i> - Finds any values in a five digit number that starts with 2 and end with 3.

# LIKE Clause – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- Following is an example, which would display all the records from the CUSTOMERS table, where the SALARY starts with 200.

**SELECT \* FROM CUSTOMERS  
WHERE SALARY LIKE '200%';**

- Now, the CUSTOMERS table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
3	Prashant	23	Chitwan	2000

# ORDER BY Clause

- The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns.
- Some databases sort the query results in an ascending order by default.
- Syntax:  
*SELECT column-list*  
*FROM table\_name*  
*WHERE [condition]*  
*ORDER BY [column1, column2, ..., columnN] [ASC | DESC];*
- You can use more than one column in the ORDER BY clause.
- Make sure whatever column you are using to sort that column should be in the column-list.

# ORDER BY Clause – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- The following code block has an example, which would sort the result in an ascending order by the NAME and the SALARY:

```
SELECT * FROM CUSTOMERS  
ORDER BY NAME;
```

- The following code block has an example, which would sort the result in the descending order by NAME:

```
SELECT * FROM CUSTOMERS  
ORDER BY NAME DESC;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
6	Krishna	22	Lamjung	4500
3	Prashant	23	Chitwan	2000
2	Sandace	25	Pokhara	1500
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
7	Sonam	22	Bhojpur	10000
5	Shreeram	28	Lumbini	8500
4	Santosh	25	Biratnagar	6500
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
6	Krishna	22	Lamjung	4500
1	Anil	32	Kathmandu	2000

# GROUP BY Clause

- The SQL **GROUP BY** clause is used in collaboration with the **SELECT** statement to arrange identical data into groups.
- This **GROUP BY** clause follows the **WHERE** clause in a **SELECT** statement and precedes the **ORDER BY** clause.
- Syntax:  
*SELECT column-list*  
*FROM table\_name*  
*WHERE [condition]*  
*GROUP BY column1, column2*  
*ORDER BY [column1, column2, ..., columnN] [ASC | DESC];*

- You can use more than one column in the **GROUP BY** clause.

# GROUP BY Clause – EXAMPLE

- Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

- If you want to know the total amount of the salary on each customer, then the GROUP BY query would be as follows:

```
SELECT  NAME, SUM(SALARY)
FROM CUSTOMERS
GROUP BY NAME;
```

- This would produce the following result.

NAME	SUM(SALARY)
Anil	2000
Santosh	6500
Sandace	1500
Prashant	2000
Shreeram	8500
Krishna	4500
Sonam	10000

# GROUP BY Clause – EXAMPLE

- Now, let us look at a table where the CUSTOMER table has the following records with duplicate names.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Anil	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Prashant	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

- Now again, if you want to know the total amount of SALARY on each customer, then the GROUP BY query would be as follows:

```
SELECT NAME, SUM(SALARY)
FROM CUSTOMER
GROUP BY NAME;
```

NAME	SUM(SALARY)
Anil	3500
Prashant	8500
Shreeram	8500
Krishna	4500
Sonam	10000

- This would produce the following result.



# ***DISTINCT** Keyword*

- *The SQL **DISTINCT** keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.*
- *There may be a situation when you have multiple duplicate records in a table.*
- *While fetching such records, it makes more sense to fetch only those unique records instead of fetching duplicate records.*
- *Syntax:*  
***SELECT DISTINCT** column-list*  
***FROM** table\_name*  
***WHERE** [condition];*

# DISTINCT – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- First let us see how the following SELECT query returns the duplicate salary records:

**SELECT SALARY  
FROM CUSTOMERS  
ORDER BY SALARY;**

- Now, let us use the DISTINCT keyword with the above SELECT query:

**SELECT DISTINCT SALARY  
FROM CUSTOMERS  
ORDER BY SALARY;**

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Anil	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Prashant	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

SALARY
1500
2000
2000
4500
6500
8500
10000

SALARY
1500
2000
4500
6500
8500
10000

# Using Joins

- *The SQL **Joins** clause is used to combine records from two or more tables in a database.*
- *A JOIN is a means for combining fields from two tables by using values common to each.*
- *Let us first create another table ORDERS using the following query:*  

```
CREATE TABLE ORDERS(  
  OID INT NOT NULL,  
  DATE_OF_ORDER DATE NOT NULL,  
  CUSTOMERS_ID INT NOT NULL,  
  AMOUNT INT NOT NULL,  
  PRIMARY KEY (OID),  
  FOREIGN KEY (CUSTOMERS_ID) REFERENCES CUSTOMERS(ID)  
);
```

# Using Joins

- Now, let us insert data to the *ORDERS* table using the following query:

```
INSERT INTO ORDERS
```

```
VALUES (102, TO_DATE('08 / October / 2009', 'DD / MON / YY'), 3, 3000);
```

```
INSERT INTO ORDERS
```

```
VALUES (100, TO_DATE('08 / October / 2009', 'DD / MON / YY'), 3, 1500);
```

```
INSERT INTO ORDERS
```

```
VALUES (101, TO_DATE('20 / November / 2009', 'DD / MON / YY'), 2, 1560);
```

```
INSERT INTO ORDERS
```

```
VALUES (103, TO_DATE('20 / May / 2008', 'DD / MON / YY'), 4, 2060);
```

# Using Joins

- Now, we have the following two tables – CUSTOMERS table and ORDERS table.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

OID	DATE_OF_ORDER	CUSTOMERS_ID	AMOUNT
102	08-OCT-09	3	3000
100	08-OCT-09	3	1500
101	20-NOV-09	2	1560
103	20-MAY-08	4	2060

# Using Joins

- Now, let us join these two tables in our *SELECT* statement using the following query:

```
SELECT ID, NAME, AGE, AMOUNT  
FROM CUSTOMERS, ORDERS  
WHERE CUSTOMERS.ID = ORDERS.CUSTOMERS_ID;
```

- This would produce the following result:

ID	NAME	AGE	AMOUNT
2	Sandace	25	1560
3	Prashant	23	3000
3	Prashant	23	1500
4	Santosh	25	2060

# Using Joins

- *Here, it is noticeable that the join is performed in the WHERE clause.*
- *Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, **BETWEEN**, **LIKE**, and **NOT**.*
- *They can all be used to join tables.*
- *However, the most common operator is the **equal to** symbol.*

# Different Types of Joins

JOIN	DESCRIPTION
INNER JOIN	Returns rows when there is a match in both tables.
LEFT JOIN	Returns all rows from the left table, even if there are no matches in the right table.
RIGHT JOIN	Returns all rows from the right table, even if there are no matches in the left table.
FULL JOIN	Returns rows when there is a match in one of the tables.
SELF JOIN	Is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
CARTESIAN JOIN	Returns the Cartesian product of the sets of records from the two or more joined tables.



# Aggregate Functions & Operations

- Aggregation function takes a collection of values and returns a single value as a result.

**avg**: average value

**min**: minimum value

**max**: maximum value

**sum**: sum of values

**count**: number of values

- Aggregate operation in relational algebra

$$G_1, G_2, \dots, G_n \text{ } \cup \text{ } F_1(A_1), F_2(A_2, \dots, F_n(A_n)) (E)$$

$E$  is any relational-algebra expression

- $G_1, G_2, \dots, G_n$  is a list of attributes on which to group (can be empty)
- Each  $F_i$  is an aggregate function
- Each  $A_i$  is an attribute name

# Aggregate Operation – Example

- Let us suppose a relation  $r$

A	B	C
$\alpha$	$\alpha$	7
$\alpha$	$\beta$	7
$\beta$	$\beta$	3
$\beta$	$\beta$	10

$r$

- Now,  $g_{sum(c)}(r)$  gives the following result.

Sum(c)
27

# Aggregate Operation – Example

- Relation *account* grouped by *branch-name*

<i>branch_name</i>	<i>account_number</i>	<i>balance</i>
<i>Perryridge</i>	<i>A – 102</i>	<i>400</i>
<i>Perryridge</i>	<i>A – 201</i>	<i>900</i>
<i>Brighton</i>	<i>A – 217</i>	<i>750</i>
<i>Brighton</i>	<i>A – 215</i>	<i>750</i>
<i>Redwood</i>	<i>A – 222</i>	<i>700</i>

- Now, *branch\_name*  $g_{sum(balance)}(account)$  gives the following result.

<i>branch_name</i>	<i>sum(balance)</i>
<i>Perryridge</i>	<i>1300</i>
<i>Brighton</i>	<i>1500</i>
<i>Redwood</i>	<i>700</i>

# Aggregate Operation – Example

- Find the average salary in each department for the following relation *instructor*.

ID	name	dept_name	salary
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

- Now,  
*dept\_name g<sub>avg</sub>(salary)(instructor)*  
gives the following result.

dept_name	avg_salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

# Aggregate Functions (Contd.)

- *Result of aggregation does not have a name.*
- *It can use rename operation to give it a name.*
- *For convenience, we permit renaming as part of aggregate operation*  
*branch\_name g<sub>sum(balance)</sub> as sum\_balance(account)*

***END OF UNIT TWO***

# Syllabus

## UNIT 3: Advanced Server Side Scripting

- **Object Oriented Programming in PHP**
  - *Classes and Objects; Defining and Using Properties & Methods; Constructors and Destructors; Method Overriding; Encapsulation; Inheritance; Polymorphism; Static Members; Exception Handling*
- **AJAX (Asynchronous JavaScript and XML)**
  - *Using PHP; Using PHP+MySQL*
- **jQuery**
  - *Playing with Elements; Hiding and Unhiding Images, jQuery UI*
- **JOOMLA**
  - *Introduction to CMS, Installation, Handling Joomla Back End, Customization in Joomla, Introduction to Extensions, Installation and Uses of Extensions in Joomla, Template Development in Joomla, IDE, Module Development in Joomla, Component Development in Joomla, Introduction to MVC (Model, View, and Controller)*
- **WordPress Administrator Level**
  - *Theme Integration, Creating Pages, Managing Posts, Managing Widgets*

# UNIT 3

## *ADVANCED SERVER SIDE SCRIPTING*



# OOP

- *OOP stands for Object-Oriented Programming.*
- *Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.*
- *Object-oriented programming has several advantages over procedural programming:*
  - *OOP is faster and easier to execute.*
  - *OOP provides a clear structure for the programs.*
  - *OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug.*
  - *OOP makes it possible to create full reusable applications with less code and shorter development time.*
- *Note: The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.*

# Classes and Objects

- *Classes and objects are the two main aspects of object-oriented programming.*
- *Look at the following table to see the difference between class and object:*

Class	Objects
Fruit	Apple
	Banana
	Mango
Car	Volvo
	Audi
	Toyota

- *So, a class is a template for objects, and an object is an instance of a class.*
- *When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.*

# Define a Class

- A class is defined by using the *class* keyword, followed by the name of the class and a pair of curly braces (*{}*).
- All its properties and methods go inside the braces:

```
<?php  
class Fruit  
{  
    // code goes here...  
}  
?>
```

# Define a Class

- Below we declare a class named *Fruit* consisting of two properties (*\$name* and *\$color*) and two methods *set\_name()* and *get\_name()* for setting and getting the *\$name* property:

```
<?php
class Fruit
{
    // Properties
    public $name;
    public $color;
    // Methods
    function set_name($name)
    {
        $this->name = $name;
    }
    function get_name()
    {
        return $this->name;
    }
}
?>
```

# Define Objects

- *Classes are nothing without objects.*
- *We can create multiple objects from a class.*
- *Each object has all the properties and methods defined in the class, but they will have different property values.*
- *Objects of a class is created using the new keyword.*

# Define Objects

- In the example below, `$apple` and `$banana` are instances of the class `Fruit`:

```
<?php
class Fruit
{
    // Properties
    public $name;
    public $color;
    // Methods
    function set_name($name)
    {
        $this->name = $name;
    }
    function get_name()
    {
        return $this->name;
    }
}
$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');
echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

# Define Objects

- In the example below, we add two more methods to class Fruit, for setting and getting the \$color property:

```
<?php
class Fruit
{
    // Properties
    public $name;
    public $color;
    // Methods
    function set_name($name)
    {
        $this->name = $name;
    }
    function get_name()
    {
        return $this->name;
    }
    function set_color($color)
    {
        $this->color = $color;
    }
    function get_color()
    {
        return $this->color;
    }
}
$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>
```

# PHP – The *\$this* Keyword

- The *\$this* keyword refers to the current object, and is only available inside methods.

- Look at the following example:

```
<?php  
class Fruit  
{  
    public $name;  
}  
$apple = new Fruit();  
?>
```

- So, where can we change the value of the *\$name* property?



# PHP – The \$this Keyword

- *There are two ways:*
  - *Inside the class (by adding a `set_name()` method and use `$this`):*

```
<?php
class Fruit
{
    public $name;
    function set_name($name)
    {
        $this->name = $name;
    }
}
$apple = new Fruit();
$apple->set_name("Apple");
?>
```

- *Outside the class (by directly changing the property value):*

```
<?php
class Fruit
{
    public $name;
}
$apple = new Fruit();
$apple->name = "Apple";
?>
```

# PHP – instanceof Keyword

- You can use the *instanceof* keyword to check if an object belongs to a specific class:

```
<?php  
$apple = new Fruit();  
var_dump($apple instanceof Fruit);  
?>
```

# PHP – instanceof Keyword

```
<?php
class Fruit
{
    // Properties
    public $name;
    public $color;
    // Methods
    function set_name($name)
    {
        $this->name = $name;
    }
    function get_name()
    {
        return $this->name;
    }
}
$apple = new Fruit();
var_dump($apple instanceof Fruit);
?>
```

# PHP OOP – Constructor

- *A constructor allows you to initialize an object's properties upon creation of the object.*
- *If you create a `__construct()` function, PHP will automatically call this function when you create an object from a class.*
- *Notice that the construct function starts with two underscores (`__`).*

# PHP OOP – Constructor

- We see in the example below, that using a constructor saves us from calling the `set_name()` method which reduces the amount of code:

```
<?php
class Fruit
{
    public $name;
    public $color;
    function __construct($name)
    {
        $this->name = $name;
    }
    function get_name()
    {
        return $this->name;
    }
}
$apple = new Fruit("Apple");
echo $apple->get_name();
?>
```

# PHP OOP – Constructor

```
<?php
class Fruit
{
    public $name;
    public $color;
    function __construct($name, $color)
    {
        $this->name = $name;
        $this->color = $color;
    }
    function get_name()
    {
        return $this->name;
    }
    function get_color()
    {
        return $this->color;
    }
}
$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>
```

# PHP OOP – Destructor

- *A destructor is called when the object is destructed or the script is stopped or exited.*
- *If you create a `__destruct()` function, PHP will automatically call this function at the end of the script.*
- *Notice that the destruct function starts with two underscores (`__`).*

# PHP OOP – Destructor

- The example below has a `__construct()` function that is automatically called when you create an object from a class, and a `__destruct()` function that is automatically called at the end of the script:

```
<?php
class Fruit
{
    public $name;
    public $color;
    function __construct($name)
    {
        $this->name = $name;
        echo "This is constructor. Fruit name is {$this->name}. <br>";
    }
    function __destruct()
    {
        echo "This is destructor. Fruit name is {$this->name}.";
    }
}
$apples = new Fruit("Apple");
?>
```



# PHP OOP – Destructor

```
<?php
class Fruit
{
    // Properties
    var $name;
    var $color;
    // Methods
    function __construct($name, $color)
    {
        $this->name = $name;
        $this->color = $color;
    }
    function __destruct()
    {
        echo "The fruit is {$this->name} and the color is {$this->color}.";
    }
}
$apple = new Fruit("Apple", "red");
?>
```

# PHP OOP – Access Modifiers

- *Properties and methods can have access modifiers which control where they can be accessed.*
- *There are three access modifiers:*
  - **Public**
    - *The property or method can be accessed from everywhere. This is default.*
  - **Protected**
    - *The property or method can be accessed within the class and by classes derived from that class.*
  - **Private**
    - *The property or method can ONLY be accessed within the class*

# PHP OOP – Access Modifiers

```
<?php
class Fruit
{
    public $name;
    protected $color;
    private $weight;
}
$mango = new Fruit();
$mango->name = 'Mango'; // OK
echo "This is {$mango->name}.";
$mango->color = 'Yellow'; // ERROR
echo "This is {$mango->color}.";
$mango->weight = '300'; // ERROR
echo "This is {$mango->weight}.";
?>
```

# PHP OOP – Inheritance

- *Inheritance in OOP means when a class derives from another class.*
- *The child class will inherit all the public and protected properties and methods from the parent class.*
- *In addition, it can have its own properties and methods.*
- *An inherited class is defined by using the extends keyword.*

# PHP OOP – Inheritance

```
<?php
class Fruit
{
    public $name;
    public $color;
    public function __construct($name, $color)
    {
        $this->name = $name;
        $this->color = $color;
    }
    public function intro()
    {
        echo "The fruit is {$this->name} and the color is {$this->color}.";
    }
}

// Strawberry is inherited from Fruit
class Strawberry extends Fruit
{
    public function message()
    {
        echo "Am I a fruit or a berry? <br>";
    }
}

$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
?>
```

# PHP – Overriding Inherited Methods

- Inherited methods can be overridden by redefining the methods (use the same name) in the child class.
- Look at the example below. The `__construct()` and `intro()` methods in the child class (Strawberry) will override the `__construct()` and `intro()` methods in the parent class (Fruit):

```
<?php
class Fruit
{
    public $name;
    public $color;
    public function __construct($name, $color)
    {
        $this->name = $name;
        $this->color = $color;
    }
    public function intro()
    {
        echo "The fruit is {$this->name} and the color is
        {$this->color}.";
    }
}
```

```
class Strawberry extends Fruit
{
    public $weight;
    public function __construct($name, $color,
    $weight)
    {
        $this->name = $name;
        $this->color = $color;
        $this->weight = $weight;
    }
    public function intro()
    {
        echo "The fruit is {$this->name}, the color is
        {$this->color}, and the weight is {$this-
        >weight} gram.";
    }
}
$strawberry = new Strawberry("Strawberry",
"red", 50);
$strawberry->intro();
?>
```

# PHP OOP – Polymorphism

- *Polymorphism means the ability to have many forms.*
- *Let's consider this with an example.*
- *Suppose a person is a teacher, and he is also studying, so he is a student as well.*
- *It means that a person has more than one roles, means, more than one form.*
- *Some times he is acting as a teacher, whereas he is acting as a student at some points.*
- *The inheritance is one way to implement polymorphism in PHP.*
- *We can create a Person class which will have some common attributes like name, age and CNIC of the person.*
- *Then we will create a Teacher class and a Student class which will have their unique attributes.*

# PHP OOP – Polymorphism

- The following piece of code will create these three classes:

```
<?php
class Person
{
    $name;
    $age;
    $cnic;
}
class Teacher extends Person
{
    $qualification;
    $roomNumber;
}
class Student extends Person
{
    $programName;
    $totalProgramFee;
}
?>
```



# PHP OOP – Polymorphism

- To make two roles or two forms of a person, we can simply create the objects of the respective roles.
- See the code below for understanding:  

```
$personAsaTeacher = new Teacher();  
$personAsaStudent = new Student();
```
- The “*personAsaTeacher*” will have the details of the person along with the details of its specific role, which is a teacher.
- Same like the teacher, the “*personAsaStudent*” will have the details of the same person but with the different information of the student.

# PHP OOP – Static Members (Static Methods)

- *Static methods can be called directly – without creating an instance of the class first.*
- *Static methods are declared with the **static** keyword:*

```
<?php
class ClassName
{
    public static function staticMethod()
    {
        echo "Hello World!";
    }
}
?>
```

- *To access a static method use the class name, double colon (::), and the method name:*

```
ClassName::staticMethod();
```

# PHP OOP – Static Members (Static Methods)

```
<?php  
class greeting  
{  
    public static function welcome()  
    {  
        echo "HelloWorld!";  
    }  
}  
  
// Call static method  
greeting::welcome();  
?>
```

# PHP OOP – Static Members

## (Static Properties)

- *Static properties can be called directly – without creating an instance of the class.*
- *Static properties are declared with the **static** keyword:*

```
<?php  
class ClassName  
{  
    public static $staticProperty = "Texas International College";  
}  
?>
```

- *To access a static property use the class name, double colon (::), and the property name:*

```
ClassName::staticProperty;
```

# PHP OOP – Static Members (Static Properties)

```
<?php  
class pi  
{  
    public static $value = 3.14159;  
}  
// Get static property  
echo pi::$value;  
?>
```

# PHP OOP – Exception Handling

- *Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs.*
- *This condition is called an **exception**.*
- *This is what normally happens when an exception is triggered:*
  - *The current code state is saved.*
  - *The code execution will switch to a predefined (custom) exception handler function.*
  - *Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code.*
- **Note:** *Exceptions should only be used with error conditions, and should not be used to jump to another place in the code at a specified point.*

# Basic Use of Exceptions

- When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.
- If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

```
<?php  
function checkNum($number)      // create function with an exception  
{  
    if($number > 1)  
    {  
        throw new Exception("Value must be 1 or below");  
    }  
    return true;  
}  
checkNum(2);      // trigger exception  
?>
```

- The code above will get an error like this:

*Fatal error: Uncaught Exception: Value must be 1 or below in C:\xampp\htdocs\OOP in PHP\Exception.php:7 Stack trace: #0 C:\xampp\htdocs\OOP in PHP\Exception.php(12): checkNum(2) #1 {main} thrown in C:\xampp\htdocs\OOP in PHP\Exception.php on line 7*

# Try, Throw, & Catch

- *To avoid the error from the example earlier, we need to create the proper code to handle an exception.*
- *Proper exception code should include:*
  - **try**
    - *A function using an exception should be in a "try" block.*
    - *If the exception does not trigger, the code will continue as normal.*
    - *However if the exception triggers, an exception is "thrown"*
  - **throw**
    - *This is how you trigger an exception.*
    - *Each "throw" must have at least one "catch"*
  - **catch**
    - *A "catch" block retrieves an exception and creates an object containing the exception information.*



# Try, Throw, & Catch

```
<?php
function checkNum($number)      // create function with an exception
{
    if($number > 1)
    {
        throw new Exception("Value must be 1 or below");
    }
    return true;
}

try      // trigger exception in a "try" block
{
    checkNum(2);
    echo 'If you see this, the number is 1 or below'; // If the exception is thrown, this text will not be shown
}

catch(Exception $e)      // catch exception
{
    echo 'Message: ' . $e->getMessage();
}

?>
```

# Multiple Exceptions

- *It is possible for a script to use multiple exceptions to check for multiple conditions.*
- *It is possible to use several if...else blocks, a switch, or nest multiple exceptions.*
- *These exceptions can use different exception classes and return different error messages.*

# Multiple Exceptions

```
<?php
class OddNumberException extends Exception { }
//user-defined function with an exception
function testEven($num)
{
//trigger an exception in a "try" block
try
{
if($num%2 == 1)
{
//throw an exception
throw new OddNumberException;
echo "After throw this statement will not
execute";
}
echo '</br> <b> If you see this text, the passed
value is an EVEN Number </b>';
}

//catch exception
catch (OddNumberException $ex)
{
echo '</br> <b> Exception Message: ODD
Number' . '</b>';
}
//catch exception
catch (Exception $e)
{
echo '</br> <b> Exception Message: ' . $e-
>getMessage() . '</b>';
}
}
echo 'Output for EVEN Number';
testEven(28);
echo '</br> </br>';
echo 'Output for ODD Number';
testEven(17);
?>
```

# Custom Exceptions

```
<?php
class DivideByZeroException extends Exception { }
class DivideByNegativeNoException extends Exception { }
function checkdivisor($dividend, $divisor)
{
    // Throw exception if divisor is zero
    try
    {
        if ($divisor == 0)
        {
            throw new DivideByZeroException;
        }
        else if ($divisor < 0)
        {
            throw new DivideByNegativeNoException;
        }
        else
        {
            $result = $dividend / $divisor;
            echo "Result of division = $result </br>";
        }
    }
}
```

```
catch (DivideByZeroException $dze)
{
    echo "Divide by Zero Exception! </br>";
}
catch (DivideByNegativeNoException $dnne)
{
    echo "Divide by Negative Number Exception
    </br>";
}
catch (Exception $ex)
{
    echo "Unknown Exception";
}
}
checkdivisor(18, 3);
checkdivisor(34, -6);
checkdivisor(27, 0);
?>
```

# Re-throwing Exceptions

- *Sometimes, when an exception is thrown, you may wish to handle it differently than the standard way.*
- *It is possible to throw an exception a second time within a "catch" block.*
- *A script should hide system errors from users.*
- *System errors may be important for the coder, but are of no interest to the user.*
- *To make things easier for the user you can re-throw the exception with a user friendly message.*

# Re-throwing Exceptions

```
<?php
class customException extends Exception
{
    public function errorMessage()
    {
        //error message
        $errorMsg = $this->getMessage().' is not a valid E-Mail address.';
        return $errorMsg;
    }
}

$email = "someone@example.com";

try
{
    try
    {
        //check for "example" in mail address
        if(strpos($email, "example") !== FALSE)
        {
            //throw exception if email is not valid
            throw new Exception($email);
        }
    }
}
```

```
catch(Exception $e)
{
    //re-throw exception
    throw new customException($email);
}

catch (customException $e)
{
    //display custom message
    echo $e->errorMessage();
}

?>
```

# Rules for Exceptions

- *Code may be surrounded in a try block, to help catch potential exceptions.*
- *Each try block or “throw” must have at least one corresponding catch block.*
- *Multiple catch blocks can be used to catch different classes of exceptions.*
- *Exceptions can be thrown (or re-thrown) in a catch block within a try block.*

## ***A SIMPLE RULE***

***IF YOU THROW SOMETHING, YOU HAVE TO CATCH IT.***

# PHP + MySQL

- *With PHP, you can connect to and manipulate databases.*
- *MySQL is the most popular database system used with PHP.*
- *PHP combined with MySQL are cross-platform.*
- *PHP5 and later can work with a MySQL database using:*
  - *MySQLi extension (“i” stands for improved)*
  - *PDO (PHP Data Objects)*
- *Earlier versions of PHP used the MySQL extension.*
- *However, this extension was deprecated in 2012.*



# MySQLi or PDO?

- *Both MySQLi and PDO have their advantages.*
- *PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL database.*
- *So, if you have to switch your project to use another database, PDO makes the process easy.*
- *You only have to change the connection string and a few queries.*
- *With MySQLi, you will need to rewrite the entire code — queries includes.*
- *Both are object-oriented, but MySQLi also offers a procedural API.*
- *Both supports prepared statements.*
- *Prepared statements protect from SQL injection, and are very important for web application security.*

# Open/Close - Connection to MySQL

```
<?php
```

```
// Create connection
```

```
$conn = mysqli_connect("localhost", "root", " ");
```

```
// Check connection
```

```
if (!$conn)
```

```
{
```

```
    die("Connection failed: " . mysqli_connect_error());
```

```
}
```

```
echo "Connected successfully";
```

```
// Close connection
```

```
mysqli_close($conn);
```

```
?>
```

# PHP Create a MySQL Database

```
<?php
// Create connection
$conn = mysqli_connect("localhost", "root", " ");
// Check connection
if (!$conn)
{
    die("Connection failed: " . mysqli_connect_error());
}
// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql))
{
    echo "Database created successfully";
}
else
{
    echo "Error creating database: " . mysqli_error($conn);
}
// Close connection
mysqli_close($conn);
?>
```

# PHP MySQL Create Table

```
<?php
// Create connection
$conn = mysqli_connect("localhost", "root", "", "myDB");
// Check connection
if (!$conn)
{
    die("Connection failed: " . mysqli_connect_error());
}
// sql to create table
$sql = "CREATE TABLE MyGuests
(
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    firstname VARCHAR(30) NOT NULL,
    lastname VARCHAR(30) NOT NULL,
    email VARCHAR(50),
    reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";
if (mysqli_query($conn, $sql))
{
    echo "Table MyGuests created successfully";
}
else
{
    echo "Error creating table: " . mysqli_error($conn);
}
// Close connection
mysqli_close($conn);
?>
```

# PHP MySQL Insert Data

```
<?php
// Create connection
$conn = mysqli_connect("localhost", "root", "", "myDB");
// Check connection
if (!$conn)
{
    die("Connection failed: " . mysqli_connect_error());
}
// sql to insert data into table
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";
if (mysqli_query($conn, $sql))
{
    echo "New record created successfully";
}
else
{
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
// Close connection
mysqli_close($conn);
?>
```

# PHP MySQL Insert Multiple Data

```
<?php
// Create connection
$conn = mysqli_connect("localhost", "root", "", "myDB");
// Check connection
if (!$conn)
{
    die("Connection failed: " . mysqli_connect_error());
}
// sql to insert multiple data into table
$sql = "INSERT INTO MyGuests (firstname, lastname, email)VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)VALUES ('Julie', 'Dooley', 'julie@example.com')";
if (mysqli_multi_query($conn, $sql))
{
    echo "New records created successfully";
}
else
{
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
// Close connection
mysqli_close($conn);
?>
```

# PHP MySQL Select Data

```
<?php
// Create connection
$conn = mysqli_connect("localhost", "root", "", "myDB");
// Check connection
if (!$conn)
{
    die("Connection failed: " . mysqli_connect_error());
}
// sql to select data
$sql = "SELECT id,firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);
if (mysqli_num_rows($result) > 0)
{
    // output data of each row
    while($row = mysqli_fetch_assoc($result))
    {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
    }
}
else
{
    echo "0 results";
}
// Close connection
mysqli_close($conn);
?>
```

# PHP MySQL Update Data

```
<?php
// Create connection
$conn = mysqli_connect("localhost", "root", "", "myDB");
// Check connection
if (!$conn)
{
    die("Connection failed: " . mysqli_connect_error());
}
// sql to update data
$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";
if (mysqli_query($conn, $sql))
{
    echo "Record updated successfully";
}
else
{
    echo "Error updating record: " . mysqli_error($conn);
}
// Close connection
mysqli_close($conn);
?>
```



# PHP MySQL Delete Data

```
<?php
// Create connection
$conn = mysqli_connect("localhost", "root", " ", "myDB");
// Check connection
if (!$conn)
{
    die("Connection failed: " . mysqli_connect_error());
}
// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";
if (mysqli_query($conn, $sql))
{
    echo "Record deleted successfully";
}
else
{
    echo "Error deleting record: " . mysqli_error($conn);
}
// Close connection
mysqli_close($conn);
?>
```

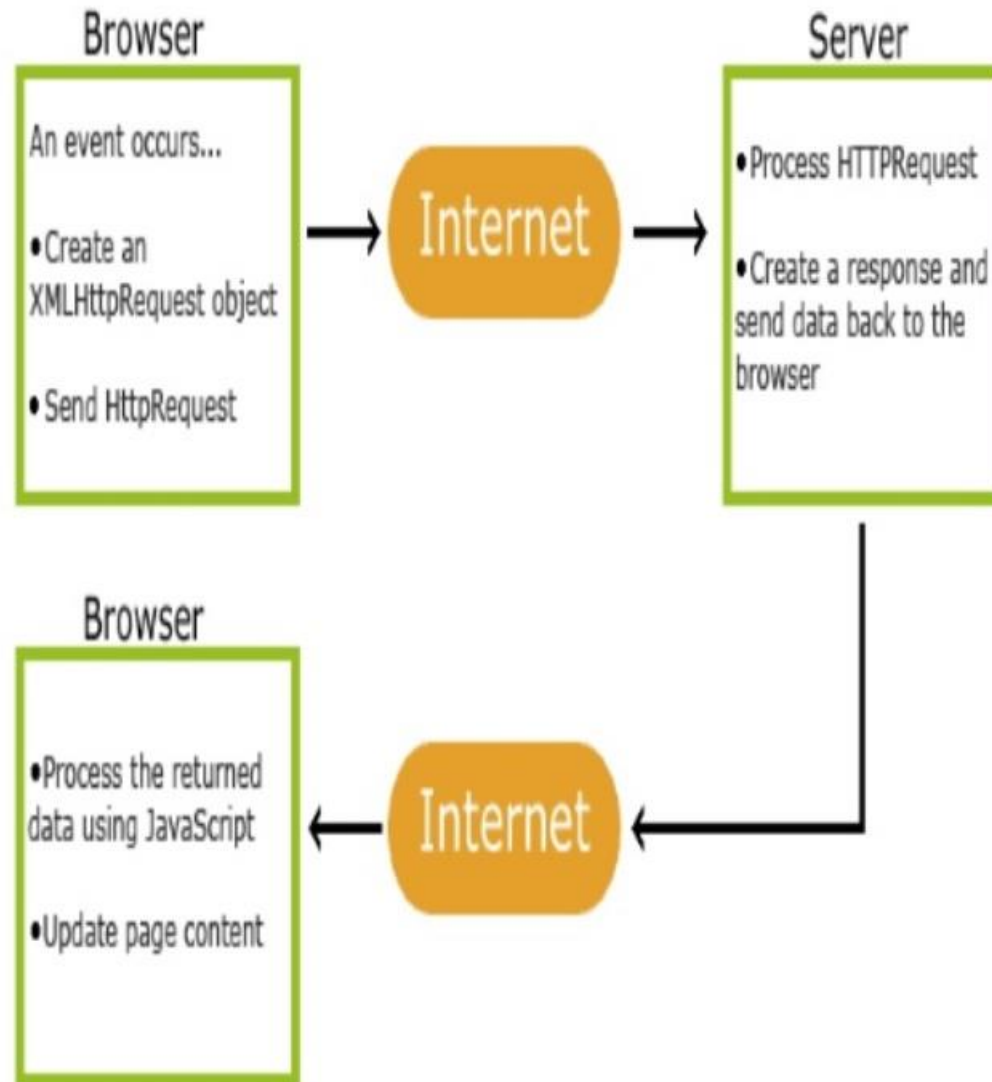
# AJAX

- *Asynchronous JavaScript And XML*
- *Not a stand-alone language or technology*
- *Combines a set of known technologies in order to create faster and more user friendly web pages*
- *Client side technology*
- *Makes web pages more responsive by exchanging small amounts of data*
- *Allows the web page to change its content without refreshing the whole page*
- *Web browser technology independent of web server software*

# The AJAX Technique

- *HTML or XHTML and CSS for presentation*
- *The Document Object Model for dynamic display and interaction with data*
- *XML for the interchange of data, and XSLT for its manipulation*
- *The XMLHttpRequest object for asynchronous communication*
- *JavaScript to bring these technologies together*

# How AJAX work?



# Problems with Conventional Web Applications

- *Interruption of user operation*
  - *Users cannot perform any operation while waiting for a response.*
- *Loss of operational context during refresh*
  - *Loss of information on the screen*
  - *Loss of scrolled position*
- *No instant feedback's to user activities*
  - *A user has to wait for the next page*
- *Constrained by HTML*
  - *Lack of useful widgets*

# Advantages of AJAX

- *Page can be refreshed dynamically*
- *Response of the interface to user is faster*
- *Load much faster because the payload is much smaller*
- *Reduces the demand for bandwidth*
- *Allows the web applications to be much more efficient*
- *Operate more like an application rather than a standalone program*

# Conclusion for AJAX

- *AJAX provides functionality to create a robust web application.*
- *If an AJAX web application is coded properly it will run faster than and as secure as a non-AJAX program.*
- *AJAX also allows websites to reduce their overall bandwidth usage and server load by reducing the amount of full page loads.*
- *Several applications using AJAX includes the following:*
  - *Google Map*
  - *Gmail*
  - *YouTube*
  - *Facebook*

# jQuery

- *jQuery is a powerful and widely used JavaScript library to simplify common web scripting task.*
- *It is based on the principle “write less, do more”.*
- *The purpose of jQuery is to make it much easier to use JavaScript.*
- *The jQuery library contains the following features:*
  - *HTML manipulation*
  - *CSS manipulation*
  - *HTML event methods*
  - *Effects and Animations*
  - *AJAX*
  - *Utilities*



# What you can do with jQuery?

- *Easily select elements to perform manipulations.*
- *Easily create effect like show or hide elements, sliding transition, and so on.*
- *Easily create complex CSS animation with fewer lines of code.*
- *Easily manipulate DOM elements and their attributes.*
- *Easily implement Ajax to enable asynchronous data exchange between client and server.*
- *Easily traverse all around the DOM tree to locate any element.*
- *Easily perform multiple actions on an element with a single line of code.*
- *Easily get or set dimensions of the HTML elements.*

# Advantages of jQuery?

- *Save lots of time*
- *Simplify common JavaScript tasks*
- *Easy to use*
- *Compatible with browsers*
- *Absolutely free*

# jQuery Syntax

- A jQuery statement starts with the dollar sign (\$) and ends with a semicolon (;).
- The dollar sign (\$) is just an alias for jQuery.

- Basic syntax for jQuery is: **\$(selector).action();**

```
<script>
```

```
$(document).ready(function(){
```

```
// Some code to be executed...
```

```
alert("Hello World!"); });
```

```
</script>
```

- **<script>** element — since jQuery is just a JavaScript library, the jQuery code can be placed inside the **<script>** element. However, if you want to place it in an external JavaScript file, which is preferred, you just remove this part.
- **\$(document).ready(handler);** — this statement is known as ready event where the **handler** is basically a function that is passed to the **ready()** method to be executed safely as soon as the document is ready to be manipulated.

# jQuery Selectors

- *jQuery selectors allow us to select and manipulate HTML elements.*
- *Current Element Selector:*
  - *`$(this).hide()` – hides the current element.*
- *Element Selector:*
  - *`$("p").hide()` – hides all `<p>` elements.*
- *.class Selector:*
  - *`$(".test").hide()` – hides the elements with `class = "test"`.*
- *#id Selector:*
  - *`$("#test").hide()` – hides the element with `id = "test"`.*

# jQuery Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $(this).hide();
    // $("p").hide();
    // $("#test").hide();
    // $(".test").hide();
  });
});
</script>
</head>
<body>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<p id="test">This is another paragraph with id.</p>
<p class="test">This is another paragraph with class.</p>
<button>Click me to hide</button>
</body>
</html>
```

# jQuery Selectors

Syntax	Description
<code>\$("*")</code>	Selects all elements
<code>\$(this)</code>	Selects the current HTML element
<code>\$("p.intro")</code>	Selects all <code>&lt;p&gt;</code> elements with <code>class="intro"</code>
<code>\$("p:first")</code>	Selects the first <code>&lt;p&gt;</code> element
<code>\$("ul li:first")</code>	Selects the first <code>&lt;li&gt;</code> element of the first <code>&lt;ul&gt;</code>
<code>\$("ul li:first-child")</code>	Selects the first <code>&lt;li&gt;</code> element of every <code>&lt;ul&gt;</code>
<code>\$("[href]")</code>	Selects all elements with an <code>href</code> attribute
<code>\$("a[target='_blank']")</code>	Selects all <code>&lt;a&gt;</code> elements with a <code>target</code> attribute value equal to <code>"_blank"</code>
<code>\$("a[target!='_blank']")</code>	Selects all <code>&lt;a&gt;</code> elements with a <code>target</code> attribute value NOT equal to <code>"_blank"</code>
<code>\$(":button")</code>	Selects all <code>&lt;button&gt;</code> elements and <code>&lt;input&gt;</code> elements of <code>type="button"</code>
<code>\$("tr:even")</code>	Selects all even <code>&lt;tr&gt;</code> elements
<code>\$("tr:odd")</code>	Selects all odd <code>&lt;tr&gt;</code> elements

# *jQuery Events*

- *Events are often triggered by the user's interaction with the web page, such as:*
  - *when a link or button is clicked*
  - *text is entered into an input box or textarea*
  - *selection is made in a select box*
  - *key is pressed on the keyboard*
  - *the mouse pointer is moved, etc.*
- *In some cases, the browser itself triggers the event, such as the page load and unload events.*

<i>Mouse Events</i>	<i>Keyboard Events</i>	<i>Form Events</i>	<i>Window Events</i>
<i>click</i>	<i>keypress</i>	<i>submit</i>	<i>load</i>
<i>dblclick</i>	<i>keydown</i>	<i>change</i>	<i>resize</i>
<i>mouseenter</i>	<i>keyup</i>	<i>focus</i>	<i>scroll</i>
<i>mouseleave</i>		<i>blur</i>	<i>unload</i>

# *jQuery Mouse Events Example*

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<style>
  p{ padding: 20px; font: 20px sans-serif; background: aqua; }
</style>
<script>
$(document).ready(function(){
  $("p").click(function(){
    $(this).slideUp();
  });
});
</script>
</head>
<body>
  <p>Click on me and I'll disappear.</p>
  <p>Click on me and I'll disappear.</p>
  <p>Click on me and I'll disappear.</p>
</body>
</html>
```



# jQuery Keyboard Events Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<style>
  p{ padding: 10px; background: lightgreen; display: none; }
  div{ margin: 20px 0; }
</style>
<script>
$(document).ready(function(){
  var i = 0;
  $('input[type="text"]').keypress(function(){
    $("span").text(i += 1);
    $("p").show().fadeOut();
  });
});
</script>
</head>
<body>
  <input type="text">
  <div>Keypress: <span>0</span></div>
  <div><strong>Note:</strong> Enter something inside the input box and see the result.</div>
  <p>Keypress is triggered.</p>
</body>
</html>
```

# jQuery Form Events Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<script>
$(document).ready(function(){
    $("select").change(function(){
        var selectedOption = $(this).find(":selected").val();
        alert("You have selected - " + selectedOption);
    });
});
</script>
</head>
<body>
    <form>
        <label>City:</label>
        <select>
            <option>Kathmandu</option>
            <option>Pokhara</option>
            <option>Chitwan</option>
        </select>
    </form>
<p><strong>Note:</strong> Select any value from the dropdown select and see the result.</p>
</body>
</html>
```

# jQuery Form Events Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<style>
  label{ display: block; margin: 5px 0; }
  label span{ display: none; }
</style>
<script>
$(document).ready(function(){
  $("input").focus(function(){
    $(this).next("span").show().fadeOut();
  });
});
</script>
</head>
<body>
  <form>
    <label>Email: <input type="text"> <span>focus fire</span></label>
    <label>Password: <input type="password"> <span>focus fire</span></label>
    <label><input type="submit" value="Sign In"> <span>focus fire</span></label>
  </form>
  <p><strong>Note:</strong> Click on the form control or press the "Tab" key to set focus.</p>
</body>
</html>
```

# jQuery Window Events Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<style>
  p{ padding: 20px;font: 20px sans-serif;background: #f0e68c; }
</style>
<script>
$(document).ready(function(){
  $(window).resize(function() {
    $(window).bind("resize",function(){
      $("p").text("Window width: " + $(window).width() + ", " + "Window height: " + $(window).height());
    });
  });
});
</script>
</head>
<body>
  <p>Open the output in a new tab and resize the browser window by dragging the corners.</p>
</body>
</html>
```

# jQuery Window Events Example

```
<html>
<head>
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
<style>
  p{ width: 100%; padding: 50px 0; text-align: center; font: bold 34px sans-serif; background:
    #f0e68c; position: fixed; top: 50px; display: none; }
  .dummy-content{ height: 600px; font: 34px sans-serif; text-align: center; }
</style>
<script>
$(document).ready(function(){
  $(window).scroll(function() {
    $("p").show().fadeOut("slow");
  });
});
</script>
</head>
<body>
  <p>Scroll Happened!</p>
  <div class="dummy-content">Scroll the viewport.</div>
  <div class="dummy-content">Scroll the viewport.</div>
  <div class="dummy-content">Scroll the viewport.</div>
  <div class="dummy-content">Scroll the viewport.</div>
  <div class="dummy-content">Scroll the viewport.</div>
</body>
</html>
```

# Introduction to JSON

- *JSON is a syntax for storing and exchanging data.*
- *JSON stands for JavaScript Object Notation.*
- *It is a lightweight text based data-interchange format.*
- *When exchanging data between a browser and a server, the data can only be text.*
- *It is easy to understand, manipulate, and generate.*
- *It is based on a subset of the JavaScript Programming Language.*

# Why use JSON?

- *Straightforward syntax*
- *Easy to create and manipulate*
- *Supported by all major JavaScript frameworks*
- *Supported by most backend technologies*

# JSON Object Syntax

- *Unordered sets of name / value pairs*
- *Begins with { (left brace)*
- *Ends with } (right brace)*
- *Each name is followed by : (colon)*
- *Name / Value pairs are separated by , (comma)*



# JSON Example

```
<html>
<body>
<p>Access a JavaScript object:</p>
<p id="demo"></p>
<script>
var myObj, x;
myObj = { name: "John", age: 30, city: "NewYork" };
//myObj.name = "Gilbert";
x = myObj.name;
//person["name"] = "Gilbert";
//x = myObj["name"];
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

<https://www.joomla.org/>

*Joomla*

[\*https://wordpress.com/\*](https://wordpress.com/)

*WordPress*

***END OF UNIT THREE***