PHP Introduction

- PHP is a recursive acronym for “PHP: Hypertext Preprocessor” -- It is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.

PHP Introduction

- The PHP code is enclosed in special start and end processing instructions **<?php** and **?>** that allow you to jump into and out of "PHP mode."

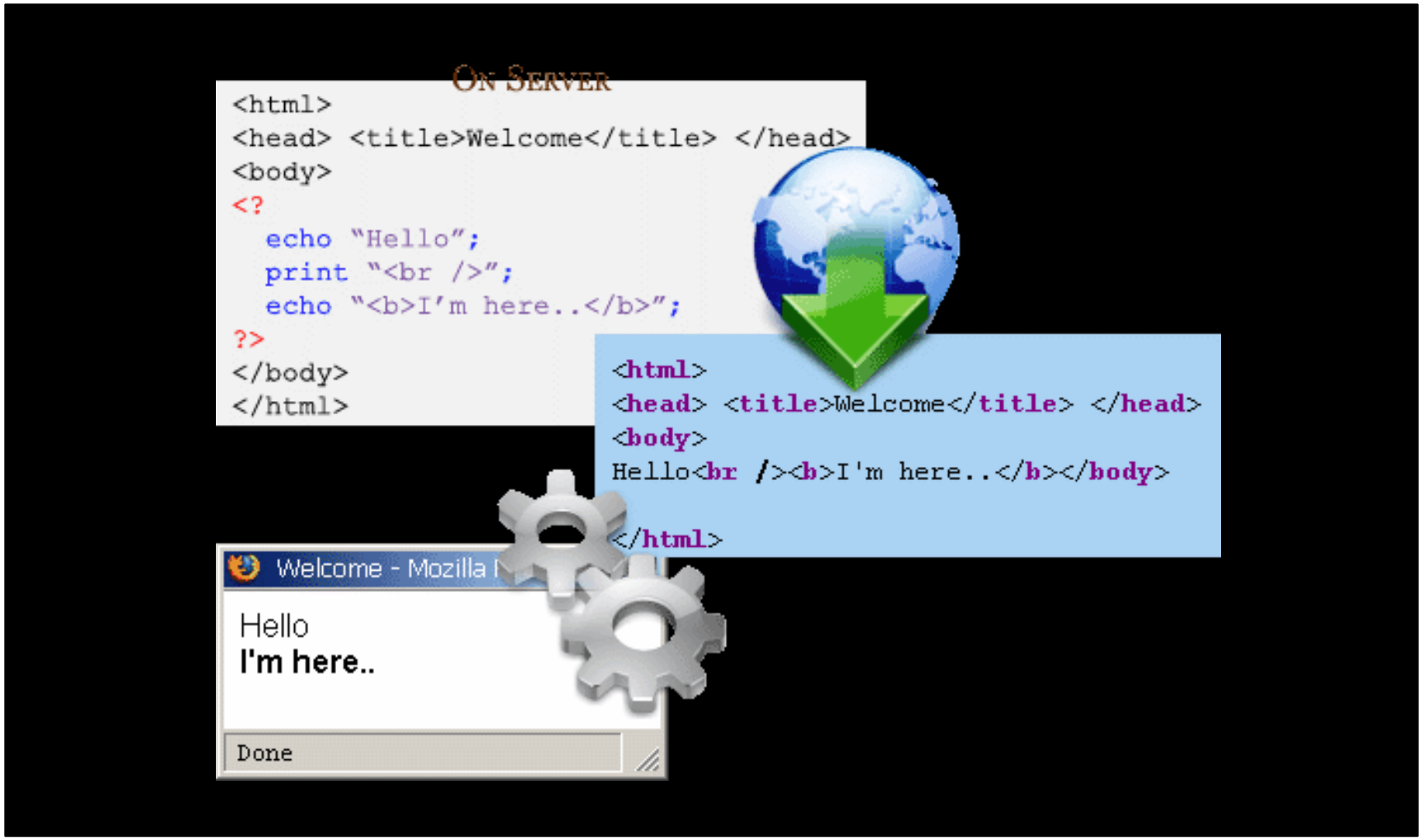
PHP Introduction

```
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <?php
            echo "Hi, I'm a PHP script!";
        ?>

    </body>
</html>
```

PHP Introduction

- PHP code is executed on the server, generating HTML which is then sent to the client. The client would receive the results of running that script, but would not know what the underlying code was.
- **echo()** statement all it does is display
- A visual, if you please...

PHP Introduction



PHP Variables

•> Variables are used for storing values, like text strings, numbers or arrays.

•> When a variable is declared, it can be used over and over again in your script.

•> All variables in PHP start with a $ sign symbol.

•> The correct way of declaring a variable in PHP:

```
$var_name = value;
```

PHP Variables

```
<?php
$txt="Hello World!";
$x=16;
?>
```

- > ***In PHP, a variable does not need to be declared before adding a value to it.***
- > In the example above, you see that you do not have to tell PHP which data type the variable is.
- > PHP automatically converts the variable to the correct data type, depending on its value.

PHP Concatenation

- > The concatenation operator (.) is used to put two string values together.
- > To concatenate two string variables together, use the concatenation operator:

```
<?php
$txt1="Hello World!";
$txt2="What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

PHP Operators

•Operators are used to operate on values. There are four classifications of operators:

- > Arithmetic
- > Assignment
- > Comparison
- > Logical

PHP Operators

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

PHP Operators

Assignment Operators

Operator	Example	Is The 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

PHP Operators

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
<>	is not equal	5<>8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

PHP Operators

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

PHP Conditional Statements

- **> if** statement - use this statement to execute some code only if a specified condition is true
- **> if...else** statement - use this statement to execute some code if a condition is true and another code if the condition is false
- **> if...elseif....else** statement - use this statement to select one of several blocks of code to be executed
- **> switch** statement - use this statement to select one of many blocks of code to be executed

PHP Conditional Statements

- Use the **if....else** statement to execute some code if a condition is true and another code if a condition is false.

```
<html>
<body>

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

</body>
</html>
```

```
<html>
<body>

<?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!";
?>

</body>
</html>
```

PHP Loops

- **while** - loops through a block of code while a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

PHP Loops

```
<html>
<body>

<?php
$i=1;
while($i<=5)
  {
  echo "The number is " . $i . "<br />";
  $i++;
  }
?>

</body>
</html>
```

```
<html>
<body>

<?php
for ($i=1; $i<=5; $i++)
  {
  echo "The number is " . $i . "<br />";
  }
?>

</body>
</html>
```

PHP Numeric Arrays

- In the following example the index is automatically assigned (the index starts at 0):

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

- In the following example we assign the index manually:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

PHP Loops - Foreach

```
foreach ($array as $value)
  {
  code to be executed;
  }
```

- For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.

PHP Loops - Foreach

- The following example demonstrates a loop that will print the values of the given array:

```
<html>
<body>

<?php
$x=array("one","two","three");
foreach ($x as $value)
  {
  echo $value . "<br />";
  }
?>

</body>
</html>
```

PHP Functions

- A function will be executed by
- a call to the function.

```
function functionName()
{
code to be executed;
}
```

```
<html>
<body>

<?php
function writeName()
{
echo "Kai Jim Refsnes";
}

echo "My name is ";
writeName();
?>

</body>
</html>
```

- > Give the function a name that reflects what the function does
- > The function name can start with a letter or underscore (not a number)

PHP Forms - $_GET Function

- > The built-in $_GET function is used to collect values from a form sent with method="get".
- > Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send (max. 100 characters).

PHP Forms - $_GET Function

```
<form action="welcome.php" method="get">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

Welcome.php

```
Welcome <?php echo $_GET["fname"]; ?>.<br />
You are <?php echo $_GET["age"]; ?> years old!
```

PHP Forms - $_POST Function

- > The built-in $_POST function is used to collect values from a form sent with method="post".
- > Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.
- > Note: However, there is an 8 Mb max size for the POST method, by default (can be changed by setting the post_max_size in the php.ini file).

PHP Forms - $_POST Function

```
<form action="action.php" method="post">
 <p>Your name: <input type="text" name="name" /></p>
 <p>Your age: <input type="text" name="age" /></p>
 <p><input type="submit" /></p>
</form>
```

And here is what the code of action.php might look like:

```
Hi <?php echo htmlspecialchars($_POST['name']); ?>.
You are <?php echo (int)$_POST['age']; ?> years old.
```

PHP Forms - $_POST Function

- Apart from **htmlspecialchars()** and **(int)**, it should be obvious what this does. **htmlspecialchars()** makes sure any characters that are special in html are properly encoded so people can't inject HTML tags or Javascript into your page.

- For the age field, since we know it is a number, we can just convert it to an integer which will automatically get rid of any stray characters. The **$_POST['name']** and **$_POST['age']** variables are automatically set for you by PHP.