Web Technology II - BIM 4th SEMESTER

Unit 5: Functions

Date Time in PHP

The PHP date() function is used to format a time and/or date. The PHP date() function formats a timestamp to a more readable date and time. A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

Syntax:

date(format,timestamp)

Parameter	Description
format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time

PHP Date() - Format the Date

The required format parameter in the date() function specifies how to format the date/time. Here are some characters that can be used:

•d - Represents the day of the month (01 to 31)

•m - Represents a month (01 to 12)

•Y - Represents a year (in four digits)

A list of all the characters that can be used in the format parameter, can be found in our PHP Date reference. Other characters, like"/", ".", or "-" can also be inserted between the letters to add additional formatting:

```
<?php
echo date("Y/m/d") . "<br />";
echo date("Y.m.d") . "<br />";
echo date("Y-m-d");
?>
```

The output of the code above could be something like this:

```
2009/05/11
2009.05.11
```

```
2009-05-11
```

PHP Date() - Adding a Timestamp

The optional timestamp parameter in the date() function specifies a timestamp. If you do not specify a timestamp, the current date and time will be used. The mktime() function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

Syntax for mktime()

The mktime(hour,minute,second,month,day,year,is_dst) To go one day in the future we simply add one to the day argument of mktime():

```
<?php

$tomorrow = mktime(0,0,0,date("m"),date("d")+1,date("Y"));

echo "Tomorrow is ".date("Y/m/d", $tomorrow);

?>

The output of the code above could be something like this:

Tomorrow is 2009/05/12
```

Function in PHP (User Defined Function with Arguments and return Values)

The real power of PHP comes from its functions. In PHP, there are more than 700 built-in functions. To keep the script from being executed when the page loads, you can put it into a function. A function will be executed by a call to the function. You may call a function from anywhere within a page.

Create a PHP Function

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

```
Syntax

function functionName()

{

code to be executed;

}
```

PHP function guidelines:

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

Example 1: User Defined Function with no Arguments or no Parameters

```
<html>
<body>
<?php
function writeName()
{
echo "Kai Jim Refsnes";
}
echo "My name is ";
writeName();
?>
</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes
```

PHP Functions - Adding parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable. Parameters are specified after the function name, inside the parentheses.

Example 2: User Defined Function with Arguments

The following example will write different first names, but equal last name:

```
<html>
<body>
<?php
function writeName($fname)
{
```

```
echo $fname . " Refsnes.<br />";
}
echo "My name is ";
writeName("Kai Jim");
echo "My sister's name is ";
writeName("Hege");
echo "My brother's name is ";
writeName("Stale");
?>
</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes.
My brother's name is Stale Refsnes.
```

Example 3: User Defined with two Parameters or Arguments

```
<html><body>
<?php
function writeName($fname,$punctuation)
{
echo $fname . " Refsnes" . $punctuation . "<br />";
}
echo "My name is ";
writeName("Kai Jim",".");
echo "My sister's name is ";
writeName("Hege","!");
echo "My brother's name is ";
writeName("Ståle","?");
```

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

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes!
My brother's name is Ståle Refsnes?
```

PHP User Defined Functions - Return values

To let a function return a value, use the return statement.

Example 4:

```
<html>
<body>
<?php
function add($x,$y)
{
$total=$x+$y;
return $total;
}
echo "1 + 16 = " . add(1,16);
?>
</body>
</html>
```

Output:

```
1 + 16 = 17
```

Passing Arguments by Reference

In PHP, arguments are usually passed by value, which means that a copy of the value is used in the function and the variable that was passed into the function cannot be changed. When a function argument is passed by reference, changes to the argument also change the variable that was passed in. To turn a function argument into a reference, the & operator is used:

Example

Use a pass-by-reference argument to update a variable:

```php
<?php
function add_five(&$value) {
  $value += 5;
}

$num = 2;
add_five($num);
echo $num;
?>
```

PHP Functions - Returning values with strict_types declaration

To specify strict we need to set declare(strict_types=1);. This must be on the very first line of the PHP file.

In the following example we try to send both a number and a string to the function, but here we have added the strict declaration:

To let a function return a value, use the return statement:

Example

```php
<?php

declare(strict_types=1); // strict requirement
function sum(int $x, int $y) {
  $z = $x + $y;
  return $z;
}

echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

Understanding Variable Scope in php

Variable Scope in PHP PHP has an easy-to-understand and beginner-friendly approach to variable scope. Basically, **a variable you define somewhere in a PHP file will be visible or available almost everywhere after it is defined**. You will also be able to use it within other files added to the program using include () and require ().

Variables are used to store and access or manipulate information later, but you cannot just access their data from any place you like. Some variables might become inaccessible in certain places, depending on the programming language you are using.

In simple terms, the scope of a variable determines its availability in different sections of a program.

In PHP, variables can be declared anywhere in the script. The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

1. **local**
2. **global**
3. **static**

However, when script has a user defined function, any variable inside has a local scope. As a result, variable defined inside a funcion can't be accessed outside. Variable defined outside (above) the function has a global scope.

Example

```
<?php
$x = 5; // global scope

function myTest() {
  // using x inside this function will generate an error
  echo "<p>Variable x inside function is: $x</p>";
$localvariable=30; // Local Scope
echo "Local variable scope can be access only in this section is: $localvariable";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>
```

Output

```
Notice: Undefined variable: x in C:\xampp\htdocs\SMS\variablescope.php on line 16
Local variable scope can be access only in this section is: 30

Variable x inside function is:

Variable x outside function is: 5
```

global scope

Access to global variable inside local scope should be explicitly enabled by using global keyword. The PHP script is as follows −

Example

```
<?php
$a=10;
$b=20;
echo "before function call a = $a b = $b" . "\n";
function myfunction(){
   global $a, $b;
   $c=($a+$b)/2;
   echo "inside function a = $a b = $b c = $c" . "\n";
   $a=$a+10;
}
myfunction();
echo "after function a = $a b = $b c = $c";
?>
```

Output

This will produce following result −

```
before function call a = 10 b = 20
inside function a = 10 b = 20 c = 15
PHP Notice: Undefined variable: c in line 13
after function a = 20 b = 20 c =
```

Global variable can now be processed inside function. Moreover, changes made to global variable inside function will be reflected in global namespace

static variable

A variable defined with static keyword is not initialized at every call to the function. Moreover, it retains its value of previous call

Example

```
<?php
function myfunction(){
  static $x=0;
  echo "x = $x" . "\n";
  $x++;
}
for ($i=1; $i<=3; $i++){
  echo "call to function :$i : ";
  myfunction();
}
?>
```

Output

This will produce following result

```
call to function :1 : x = 0
call to function :2 : x = 1
call to function :3 : x = 2
```