**Date Functions**

**PHP Date & Time Functions**

There are nearly fifty date and time functions, so for the purpose of this tutorial we will narrow them down to three.

|  |  |
| --- | --- |
| **Function** | **Description** |
| [time()](http://www.php.net/manual/en/function.time.php) | Returns the Current Time as a Unix Timestamp |
| [date()](http://www.php.net/manual/en/function.date.php) | Formats a Local Time/Date |
| [strtotime()](http://www.php.net/manual/en/function.strtotime.php) | Parses an English Textual Date or Time Into a Unix Timestamp |

**The PHP Time() Function**

The time() function returns the current timestamp. The timestamp that you see in the previous paragraph is generated by a very small line of code: <?php echo time(); ?>

The time() function can also return a modified timestamp. You can add or subtract any number of seconds to the function in order to return the timestamp of a previous or upcoming date and time.

For example, to return a timestamp for next week, or to return a timestamp from last week, I can add or subtract 7 days by determining how many seconds are involved (7 days \* 24 hours per day \* 60 minutes in an hour \* 60 seconds in an hour = number of second in 7 days).

<?php  
  $last\_week = time() - (7 \* 24 \* 60 \* 60);  
  $next\_week = time() + (7 \* 24 \* 60 \* 60);  
  $next\_month = time() + (30 \* 24 \* 60 \* 60);  
  echo "Last Week: " . $last\_week . "<br />";  
  echo "Next Week: " . $next\_week . "<br />";  
  echo "Next Month: " . $next\_month . "<br />";  
?>

The result will be:

Last Week: 1399978968  
Next Week: 1401188568  
Next Month: 1403175768

**The PHP Strtotime() Function**

The strtotime() function accepts an English datetime description and turns it into a timestamp. It is a simple way to determine "next week" or "last monday" without using the time() function and a bunch of math.

Some examples are:

<?php  
  echo strtotime("now") . "<br />";  
  echo strtotime("tomorrow") . "<br />";  
  echo strtotime("yesterday") . "<br />";  
  echo strtotime("10 September 2000") . "<br />";  
  echo strtotime("+1 day") . "<br />";  
  echo strtotime("+1 week") . "<br />";  
  echo strtotime("+1 week 2 days 4 hours 2 seconds") . "<br />";  
  echo strtotime("next Thursday") . "<br />";  
  echo strtotime("last Monday") . "<br />";  
  echo strtotime("4pm + 2 Hours") . "<br />";  
  echo strtotime("now + 2 fortnights") . "<br />";  
  echo strtotime("last Monday") . "<br />";  
  echo strtotime("2pm yesterday") . "<br />";  
  echo strtotime("7am 12 days ago") . "<br />";  
?>

**The PHP Date() Function**

The date() function formats a timestamp so that it actually makes sense, such as 4:02 AM Tuesday, May 20, 2014.

The date() function accepts two arguments, according to the following syntax: date(format, timestamp);. The first argument is the format that you want the timestamp in. The second argument (optional) is the timestamp that you want formatted. If no timestamp is supplied, the current timestamp will be used.

PHP provides over thirty-five case-sensitive characters that are used to format the date and time. These characters are:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Character** | **Description** | **Example** |
| Day | j | Day of the Month, No Leading Zeros | 1 - 31 |
| Day | d | Day of the Month, 2 Digits, Leading Zeros | 01 - 31 |
| Day | D | Day of the Week, First 3 Letters | Mon - Sun |
| Day | l (lowercase 'L') | Day of the Week | Sunday - Saturday |
| Day | N | Numeric Day of the Week | 1 (Monday) - 7 (Sunday) |
| Day | w | Numeric Day of the Week | 0 (Sunday) - 6 (Saturday) |
| Day | S | English Suffix For Day of the Month | st, nd, rd or th |
| Day | z | Day of the Year | 0 - 365 |
| Week | W | Numeric Week of the Year (Weeks Start on Mon.) | 1 - 52 |
| Month | M | Textual Representation of a Month, Three Letters | Jan - Dec |
| Month | F | Full Textual Representation of a Month | January - December |
| Month | m | Numeric Month, With Leading Zeros | 01 - 12 |
| Month | n | Numeric Month, Without Leading Zeros | 1 - 12 |
| Month | t | Number of Days in the Given Month | 28 - 31 |
| Year | L | Whether It's a Leap Year | Leap Year: 1, Otherwise: 0 |
| Year | Y | Numeric Representation of a Year, 4 Digits | 1999, 2003, etc. |
| Year | y | 2 Digit Representation of a Year | 99, 03, etc. |
| Time | a | Lowercase Ante Meridiem & Post Meridiem | am or pm |
| Time | A | Uppercase Ante Meridiem & Post Meridiem | AM or PM |
| Time | B | Swatch Internet Time | 000 - 999 |
| Time | g | 12-Hour Format Without Leading Zeros | 1 - 12 |
| Time | G | 24-Hour Format Without Leading Zeros | 0 - 23 |
| Time | h | 12-Hour Format With Leading Zeros | 01 - 12 |
| Time | H | 24-Hour Format With Leading Zeros | 00 - 23 |
| Time | i | Minutes With Leading Zeros | 00 - 59 |
| Time | s | Seconds With Leading Zeros | 00 - 59 |
| Timezone | e | Timezone Identifier | Example: UTC, Atlantic |
| Timezone | I (capital i) | Whether Date Is In Daylight Saving Time | 1 if DST, otherwise 0 |
| Timezone | O | Difference to Greenwich Time In Hours | Example: +0200 |
| Timezone | P | Difference to Greenwich Time, With Colon | Example: +02:00 |
| Timezone | T | Timezone Abbreviation | Examples: EST, MDT ... |
| Timezone | Z | Timezone Offset In Seconds | -43200 through 50400 |

Using a combination of these characters and commas, periods, dashes, semicolons and backslashes, you can now format dates and times.

<?php  
  // Will Echo: 4:02 AM Tuesday, May 20, 2014  
  echo date("g:i A l, F d, Y");  
  
  // Will Echo: 2014-05-19  
  $yesterday = strtotime("yesterday");  
  echo date("Y-m-d", $yesterday);  
?>

**Calculating Past & Future Dates**

It is very simple to compare two dates in php, to verify if one is past, present or future.

Since all dates can be converted back into timestamps, all it takes is a simple comparison of the two timestamps, which are essentially numbers. If the date in question is a bigger number than the current date, than you know that your date is in the future, and if the date is smaller than the current date, you know that the date is already past.

<?php  
  $my\_date = strtotime("10 April 2005");  
  if(strtotime($my\_date) > time()) {  
    echo "Future Date!";  
  } if(strtotime($my\_date) < time()) {  
    echo "Past Date!";  
  } if(strtotime($my\_date) == time()) {  
    echo "Current Date!";  
  }  
?>

# PHP Date - The Timestamp

The date function always formats a timestamp, whether you supply one or not. What's a timestamp? Good question!

* **Timestamp**: A timestamp is the number of seconds from January 1, 1970 at 00:00. Otherwise known as the Unix Timestamp, this measurement is a widely used standard that PHP has chosen to utilize.

## PHP Code:

<?php

echo date("m/d/y");

?>

If the 2010 Winter Olympics were just finishing up, you would see something like:

## Display:

02/27/10

Be sure to test this out on your own PHP enabled server, it's really great to see the instant results available with PHP date!

# Supplying a Timestamp

As our first example shows, the first argument of the *date* function tells PHP how you would like your date and time displayed. The second argument allows for a timestamp and is optional.

This example uses the *mktime* function to create a timestamp for tomorrow. To go one day in the future we simply add one to the day argument of *mktime*. For your future reference, we have the arguments of *mktime*.

**Note**: These arguments are all optional. If you do not supply any arguments the current time will be used to create the timestamp.

* mktime(hour, minute, second, month, day, year, daylight savings time)

## PHP Code:

<?php

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

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

?>

Notice that we used one letter at a time with the function *date* to get the month, day and year. For example the *date("m")* will return the month's number 01-12.

If we were to run our new script just after the 2010 Winter Olympics our display would look like:

## Display:

Tomorrow is 02/28/10

**PHP Session**

# Starting a PHP Session

Before you can begin storing user information in your PHP session, you must first start the session. When you start a session, it must be at the very beginning of your code, before any HTML or text is sent.

Below is a simple script that you should place at the beginning of your PHP code to start up a PHP session.

## PHP Code:

<?php

session\_start(); // start up your PHP session!

?>

This tiny piece of code will register the user's session with the server, allow you to start saving user information and assign a UID (unique identification number) for that user's session.

# Storing a Session Variable

When you want to store user data in a session use the $\_SESSION [associative array](http://www.tizag.com/phpT/arrays.php). This is where you both store and retrieve session data. In previous versions of PHP there were other ways to perform this store operation, but it has been updated and this is the correct way to do it.

## PHP Code:

<?php

session\_start();

$\_SESSION['views'] = 1; // store session data

echo "Pageviews = ". $\_SESSION['views']; //retrieve data

?>

## Display:

Pageviews = 1

In this example we learned how to store a variable to the session associative array $\_SESSION and also how to retrieve data from that same array.

# PHP Sessions: Using PHP's *isset* Function

Now that you are able to store and retrieve data from the $\_SESSION array, we can explore some of the real functionality of sessions. When you create a variable and store it in a session, you probably want to use it in the future. However, before you use a session variable it is necessary that you check to see if it exists already!

This is where PHP's *isset* function comes in handy. *isset* is a function that takes any variable you want to use and checks to see if it has been **set**. That is, it has already been assigned a value.

With our previous example, we can create a very simple pageview counter by using *isset* to check if the pageview variable has already been created. If it has we can increment our counter. If it doesn't exist we can create a pageview counter and set it to one. Here is the code to get this job done:

## PHP Code:

<?php

session\_start();

if(isset($\_SESSION['views']))

$\_SESSION['views'] = $\_SESSION['views']+ 1;

else

$\_SESSION['views'] = 1;

echo "views = ". $\_SESSION['views'];

?>

The first time you run this script on a **freshly opened browser** the *if statement* will fail because no session variable *views* would have been stored yet. However, if you were to refresh the page the *if statement* would be true and the counter would increment by one. Each time you reran this script you would see an increase in *view* by one.

# Cleaning and Destroying your Session

Although a session's data is temporary and does not require that you explicitly clean after yourself, you may wish to delete some data for your various tasks.

Imagine that you were running an online business and a user used your website to buy your goods. The user has just completed a transaction on your website and you now want to remove everything from their shopping cart.

## PHP Code:

<?php

session\_start();

if(isset($\_SESSION['cart']))

unset($\_SESSION['cart']);

?>

You can also completely destroy the session entirely by calling the *session\_destroy* function.

## PHP Code:

<?php

session\_start();

session\_destroy();

?>

Destroy will reset your session, so don't call that function unless you are entirely comfortable losing all your stored session data!

**phpCookies**

# Creating Your First PHP Cookie

When you create a cookie, using the function *setcookie*, you must specify three arguments. These arguments are ***setcookie****(name, value, expiration)*:

1. **name**: The name of your cookie. You will use this name to later retrieve your cookie, so don't forget it!
2. **value**: The value that is stored in your cookie. Common values are username(string) and last visit(date).
3. **expiration**: The date when the cookie will expire and be deleted. If you do not set this expiration date, then it will be treated as a session cookie and be removed when the browser is restarted.

In this example we will be creating a cookie that stores the user's last visit to measure how often people return to visit our webpage. We want to ignore people that take longer than two months to return to the site, so we will set the cookie's expiration date to two months in the future!

## PHP Code:

<?php

//Calculate 60 days in the future

//seconds \* minutes \* hours \* days + current time

$inTwoMonths = 60 \* 60 \* 24 \* 60 + time();

setcookie('lastVisit', date("G:i - m/d/y"), $inTwoMonths);

?>

Don't worry if you can't follow the somewhat involved date calculations in this example. The important part is that you know how to set a cookie, by specifying the three important arguments: name, value and expiration date.

# Retrieving Your Fresh Cookie

If your cookie hasn't expired yet, let's retrieve it from the user's PC using the aptly named *$\_COOKIE* associative array. The name of your stored cookie is the key and will let you retrieve your stored cookie value!

## PHP Code:

<?php

if(isset($\_COOKIE['lastVisit']))

$visit = $\_COOKIE['lastVisit'];

else

echo "You've got some stale cookies!";

echo "Your last visit was - ". $visit;

?>

This handy script first uses the *isset* function to be sure that our "lastVisit" cookie still exists on the user's PC, if it does, then the user's last visit is displayed. If the user visited our site on February 28, 2008 it might look something like this:

## Display:

Your last visit was - 11:48 - 02/28/08

# PHP - String Capitalization Functions

If you've ever wanted to manipulate the capitalization of your PHP strings, then this lesson will be quite helpful to you. PHP has three primary capitalization related functions: strtoupper, strtolower and ucwords. The function names are pretty self-explanatory, but why they are useful in programming might be new to you.

# Converting a String to Upper Case - strtoupper

The *strtoupper* function takes one argument, the string you want converted to upper case and returns the converted string. Only letters of the alphabet are changed, numbers will remain the same.

## PHP Code:

$originalString = "String Capitalization 1234";

$upperCase = strtoupper($originalString);

echo "Old string - $originalString <br />";

echo "New String - $upperCase";

## Display:

Old string - String Capitalization 1234  
New String - STRING CAPITALIZATION 1234

One might use this function to increase emphasis of a important point or in a title. Another time it might be used with a font that looks very nice with all caps to fit the style of the web page design.

A more technical reason would be to convert two strings you are comparing to see if they are equal. By converting them to the same capitalization you remove the possibility that they won't match simply because of different capitalizations.

# Converting a String to Lower Case - strtolower

The *strtolower* function also has one argument: the string that will be converted to lower case.

## PHP Code:

$originalString = "String Capitalization 1234";

$lowerCase = strtolower($originalString);

echo "Old string - $originalString <br />";

echo "New String - $lowerCase";

## Display:

Old string - String Capitalization 1234  
New String - string capitalization 1234

# Capitalizing the First Letter - ucwords

Titles of various media types often capitalize the first letter of each word and PHP has a time-saving function that will do just this.

## PHP Code:

$titleString = "a title that could use some hELP";

$ucTitleString = ucwords($titleString);

echo "Old title - $titleString <br />";

echo "New title - $ucTitleString";

## Display:

Old title - a title that could use some hELP  
New title - A Title That Could Use Some HELP

Notice that the last word "hELP" did not have the capitalization changed on the letters that weren't first, they remained capitalized. If you want to ensure that **only** the first letter is capitalized in each word of your title, first use the *strtolower* function and then the *ucwords* function.

## PHP Code:

$titleString = "a title that could use some hELP";

$lowercaseTitle = strtolower($titleString);

$ucTitleString = ucwords($lowercaseTitle);

echo "Old title - $titleString <br />";

echo "New title - $ucTitleString";

## Display:

Old title - a title that could use some hELP  
New title - A Title That Could Use Some Help