Welcome to PHP

PHP: Hypertext Preprocessor (**PHP**) is a free, highly popular, open source scripting language. PHP scripts are executed on the **server**.

Just a short list of what PHP is capable of:
- Generating dynamic page content
- Creating, opening, reading, writing, deleting, and closing files on the server
- Collecting form data
- Adding, deleting, and modifying information stored in your database
- controlling user-access
- encrypting data
- and much more!

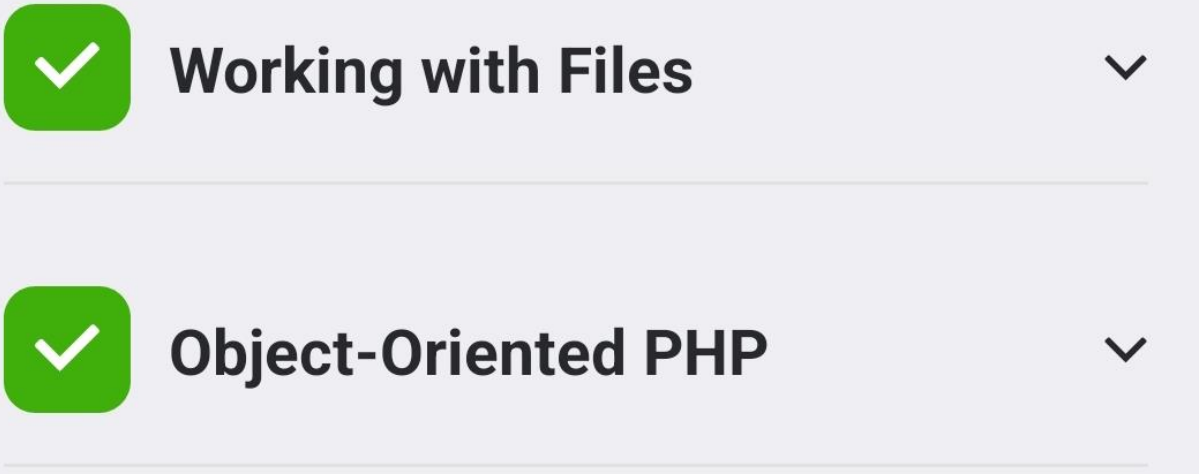
Before starting this tutorial, you should have a basic understanding of **HTML**.

Why PHP

PHP **runs** on numerous, varying platforms, including Windows, Linux, Unix, Mac OS X, and so on.
PHP is **compatible** with almost any modern server, such as Apache, IIS, and more.
PHP **supports** a wide range of databases.
PHP is **free**!

PHP is easy to learn and runs efficiently on the server side.

- [x] Basic Syntax
- [x] Variables
- [x] Operators
- [x] Arrays
- [x] Control Structures
- [x] Functions
- [x] Predefined Variables


Working with Files
Object-Oriented PHP

PHP Syntax

A PHP script starts with **<?php** and ends with **?>**:

```php
<?php
  // PHP code goes here
?>
```

Here is an example of a simple PHP file. The PHP script uses a built in function called "**echo**" to output the text "Hello World!" to a web page.

CODE PLAYGROUND **PHP**

```php
<html>
  <head>
    <title>My First PHP Page</title>
  </head>
  <body>
  <?php
    echo "Hello World!";
  ?>
  </body>
</html>
```

Tap to edit

PHP statements end with **semicolons (;)**.

PHP Syntax

Alternatively, we can include PHP in the HTML **<script>** tag.

```HTML
<html>
  <head>
    <title>My First PHP Page</title>
  </head>
  <body>
  <script language="php">
    echo "Hello World!";
  </script>
  </body>
</html>
```

However, the latest version of PHP removes support for <script language="php"> tags. As such, we recommend using **<?php ?>** exclusively.

PHP Syntax

You can also use the shorthand PHP tags, <? ?>, as long as they're supported by the server.

CODE PLAYGROUND **PHP**

```php
<?
  echo "Hello World!";
?>

Tap to edit
```

However, **<?php ?>**, as the official standard, is the recommended way of defining PHP scripts.

Echo

PHP has a built-in "**echo**" function, which is used to output text.
In actuality, it's not a function; it's a **language construct**. As such, it does not require parentheses.

Let's output a text.

CODE PLAYGROUND **PHP**

```php
<?php
  echo "I love PHP!";
?>
```

Tap to edit

PHP Statements

Each PHP statement must end with a **semicolon**.

CODE PLAYGROUND **PHP**

```php
<?php
  echo "A";
  echo "B";
  echo "C";
?>

Tap to edit |
```

Forgetting to add a semicolon at the end of a statement results in an **error**.

Echo

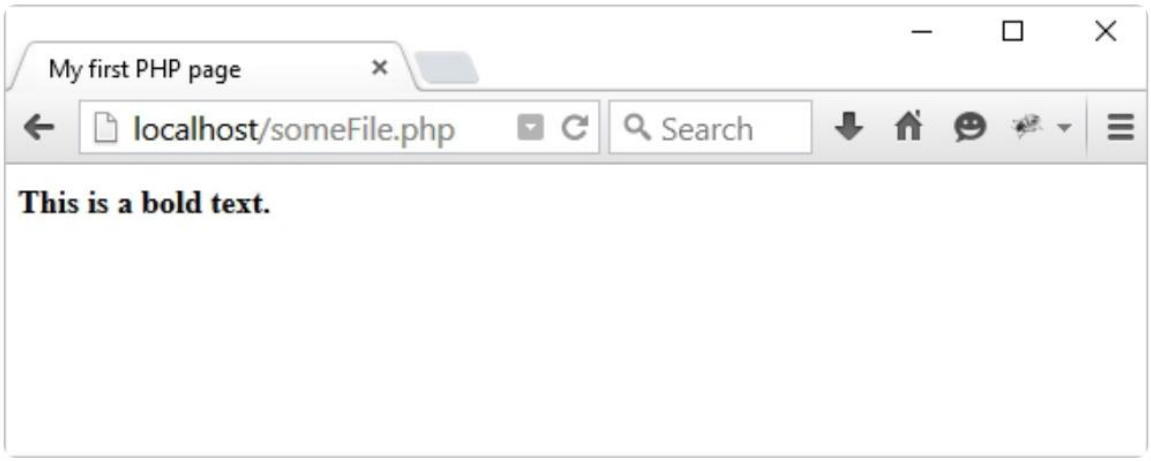
HTML markup can be added to the text in the **echo** statement.

CODE PLAYGROUND PHP

```php
<?php
   echo "<strong>This is a bold text.</
strong>";
?>

Tap to edit |
```

Result:



Run the code and see how it works!

Comments

In PHP code, a **comment** is a line that is not executed as part of the program. You can use comments to communicate to others so they understand what you're doing, or as a reminder to yourself of what you did.

A **single-line** comment starts with //:

CODE PLAYGROUND **PHP**

```
<?php
  echo "<p>Hello World!</p>";
  // This is a single-line comment
  echo "<p>I am learning PHP!</p>";
  echo "<p>This is my first program!</
p>";
?>
```

Multi-Line Comments

Multi-line comments are used for composing comments that take more than a single line.
A multi-line comment begins with /* and ends with */.

CODE PLAYGROUND PHP

```
<?php
  echo "<p>Hello World!</p>";
   /*
   This is a multi-line comment block
   that spans over
   multiple lines
   */
   echo "<p>I am learning PHP!</p>";
   echo "<p>This is my first program!</
p>";
?>
```

Variables

Lesson

Variables

Lesson

Constants

Lesson

Data Types

Lesson

Variable Scope

Lesson

Variable Variables

Variables

Variables are used as "containers" in which we store information.
A PHP variable starts with a dollar sign ($), which is followed by the name of the variable.

```php
$variable_name = value;
```

Rules for PHP variables:
- A variable name must start with a letter or an underscore
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($name and $NAME would be two different variables)

For example:

CODE PLAYGROUND PHP

```php
<?php
    $name = 'John';
    $age = 25;
    echo $name;

  // Outputs 'John'
?>
```

Tap to edit |

In the example above, notice that we did 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.

Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Constants

Constants are similar to variables except that they cannot be changed or undefined after they've been defined.
Begin the name of your constant with a letter or an underscore.
To create a constant, use the **define()** function:

```php
define(name, value, case-insensitive)
```

Parameters:
name: Specifies the name of the constant;
value: Specifies the value of the constant;
case-insensitive: Specifies whether the constant name should be case-insensitive. Default is **false;**

The example below creates a constant with a **case-sensitive** name:

CODE PLAYGROUND **PHP**

```php
<?php
  define("MSG", "Hi SoloLearners!");
  echo MSG;

   // Outputs "Hi SoloLearners!"
?>
```

Tap to edit |

The example below creates a constant with a **case-insensitive** name:

CODE PLAYGROUND **PHP**

```php
<?php
  define("MSG", " Hi SoloLearners!",
true);
  echo msg;

  // Outputs "Hi SoloLearners!"
?>
```

Tap to edit |

No dollar sign ($) is necessary before the constant name.

Data Types

Variables can store a variety of data types.
Data types supported by PHP: **String**, **Integer**, **Float**, **Boolean**, **Array**, **Object**, NULL, Resource.

[h2]PHP String[/h2]
A **string** is a sequence of characters, like "Hello world!"
A string can be any text within a set of single or double **quotes**.

```php
<?php
  $string1 = "Hello world!"; //double quotes
  $string2 = 'Hello world!';  //single quotes
?>
```

You can join two strings together using the dot (.) concatenation operator.
For example: **echo $s1 . $s2**

[h2]PHP Integer[/h2]
An **integer** is a whole number (without decimals) that must fit the following criteria:
- It cannot contain commas or blanks
- It must not have a decimal point
- It can be either positive or negative

```php
<?php
  $int1 = 42; // positive number
  $int2 = -42; // negative number
?>
```

Variables can store a variety of data types.

PHP Float

A **float**, or floating point number, is a number that includes a decimal point.

```php
<?php
  $x = 42.168;
?>
```

[h2]PHP Boolean[/h2]

A **Boolean** represents two possible states: TRUE or FALSE.

```php
<?php
  $x = true; $y = false;
?>
```

Booleans are often used in conditional testing, which will be covered later on in the course.

Most of the data types can be used in combination with one another. In this example, **string** and **integer** are put together to determine the sum of two numbers.

CODE PLAYGROUND **PHP**

```php
<?php
  $str = "10";
  $int = 20;
  $sum = $str + $int;
  echo ($sum);

  // Outputs 30
?>

Tap to edit |
```

PHP automatically converts each variable to the correct data type, according to its value. This is why the variable **$str** is treated as a number in the addition.

Variables Scope

PHP variables can be declared anywhere in the script.
The **scope** of a variable is the part of the script in which the variable can be referenced or used.

PHP's most used variable scopes are **local**, **global**.
A variable declared outside a function has a **global scope**.
A variable declared within a function has a **local scope**, and can only be accessed within that function.

Consider the following example.

CODE PLAYGROUND PHP

```php
<?php
  $name = 'David';
  function getName() {
    echo $name;
  }
  getName();

  // Error: Undefined variable: name
?>
```

Tap to edit

This script will produce an error, as the **$name** variable has a **global** scope, and is not accessible within the **getName()** function. Tap continue to see how functions can access global variables.

Functions will be discussed in the coming lessons.

The global Keyword

The **global** keyword is used to access a global variable from within a function.
To do this, use the **global** keyword within the function, prior to the variables.

CODE PLAYGROUND PHP

```php
<?php
  $name = 'David';
  function getName() {
    global $name;
    echo $name;
  }
  getName();

  //Outputs 'David'
?>
```

Variable Variables

With PHP, you can use one variable to specify another variable's name.
So, a **variable variable** treats the value of another variable as its name.

For example:

CODE PLAYGROUND PHP

```php
<?php
  $a = 'hello';
  $hello = "Hi!";
  echo $$a;

  // Outputs 'Hi!'
?>
```

$$a is a variable that is using the value of another variable, **$a**, as its name.
The value of **$a** is equal to "hello".
The resulting variable is **$hello**, which holds the value "Hi!".

Operators

Lesson

Arithmetic Operators

Lesson

Assignment Operators

Lesson

Comparison Operators

Lesson

Logical Operators

Operators

Operators carry out operations on variables and values.

[h2]Arithmetic Operators[/h2]

Arithmetic operators work with numeric values to perform common arithmetical operations.

Operator	Name	Example
+	**Addition**	$x + $y
-	**Subtraction**	$x - $y
*	**Multiplication**	$x * $y
/	**Division**	$x / $y
%	**Modulus**	$x % $y

Example:

CODE PLAYGROUND PHP

```php
<?php
  $num1 = 8;
  $num2 = 6;

  //Addition
  echo $num1 + $num2; //14

  //Subtraction
  echo $num1 - $num2; //2

  //Multiplication
  echo $num1 * $num2; //48

  //Division
   echo $num1 / $num2; //1.33333333333
?>
```

Tap to edit |

Modulus

The **modulus** operator, represented by the **%** sign, returns the remainder of the division of the first operand by the second operand:

CODE PLAYGROUND PHP

```php
<?php
  $x = 14;
  $y = 3;
  echo $x % $y; // 2
?>

Tap to edit |
```

If you use floating point numbers with the modulus operator, they will be converted to **integers** before the operation.

Increment & Decrement

The **increment** operators are used to increment a variable's value.
The **decrement** operators are used to decrement a variable's value.

```php
$x++; // equivalent to $x = $x+1;
$x--; // equivalent to $x = $x-1;
```

Increment and decrement operators either precede or follow a variable.

```php
$x++; // post-increment
$x--; // post-decrement
++$x; // pre-increment
--$x; // pre-decrement
```

The difference is that the post-increment returns the original value **before** it changes the variable, while the pre-increment changes the variable first and then returns the value.

Example:

```php
$a  = 2; $b = $a++; // $a=3,  $b=2
$a  = 2; $b = ++$a; // $a=3,  $b=3
```

The increment operators are used to increment a variable's value.

Assignment Operators

Assignment operators are used to write values to variables.

```php
$num1 = 5;
$num2 = $num1;
```

$num1 and **$num2** now contain the value of 5.

Assignments can also be used in conjunction with arithmetic operators.

Assignment	Same as...	Description
x+=y	x = x + y	**Addition**
x-=y	x = x - y	**Subtraction**
x*=y	x = x * y	**Multiplication**
x/=y	x = x / y	**Division**
x%=y	x = x % y	**Modulus**

Example:

CODE PLAYGROUND **PHP**

```php
<?php
 $x = 50;
 $x += 100;
 echo $x;

 // Outputs: 150
?>

Tap to edit |
```

Comparison Operators

Comparison operators compare two values (numbers or strings).
Comparison operators are used inside conditional statements, and evaluate to either **TRUE** or **FALSE**.

Operator	Name	Example	Result
==	Equal	**$x == $y**	Returns true if **$x is equal to $y**
===	Identical	**$x === $y**	Returns true if **$x is equal to $y**, and they are of the same type
!=	Not equal	**$x != $y**	Returns true if **$x is not equal to $y**
<>	Not equal	**$x <> $y**	Returns true if **$x is not equal to $y**
!==	Not identical	**$x !== $y**	Returns true if **$x is not equal to $y,** or they are not of the same type

Be careful using == and === ; the first one doesn't check the type of data.

Comparison Operators

Additional comparison operators:

Operator	Name	Example	Result
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

The PHP comparison operators are used to compare two values (number or string).

Logical Operators

Logical operators are used to combine conditional statements.

Operator	Name	Example	Result
and	And	**$x and $y**	True if **both $x and $y are true**
or	Or	**$x or $y**	True if **either $x or $y is true**
xor	Xor	**$x xor $y**	True if **either $x or $y is true, but not both**
&&	And	**$x && $y**	True if **both $x and $y are true**
||	Or	**$x || $y**	True if **either $x or $y is true**
!	Not	**!$x**	True if **$x is not true**

You can combine as many terms as you want. Use parentheses () for precedence.

Arrays

Lesson

Numeric Arrays

Lesson

Associative Arrays

Lesson

Multi-Dimensional Arrays

Lesson

Module 4 Quiz

Arrays

An **array** is a special variable, which can hold more than one value at a time.
If you have a list of items (a list of names, for example), storing them in single variables would look like this:

```php
$name1 = "David";
$name2 = "Amy";
$name3 = "John";
```

But what if you have 100 names on your list? The solution: Create an **array**!

[h2]Numeric Arrays[/h2]
Numeric or indexed arrays associate a numeric index with their values.
The index can be assigned automatically (index always starts at **0**), like this:

```php
$names = array("David", "Amy", "John");
```

As an alternative, you can assign your index manually.

```php
$names[0] = "David";
$names[1] = "Amy";
$names[2] = "John";
```

We defined an array called **$names** that stores three values.
You can access the array elements through their indices.

CODE PLAYGROUND PHP

```php
echo $names[1]; // Outputs "Amy"

Tap to edit |
```

Remember that the first element in an array has the index of **0**, not 1.

Numeric Arrays

You can have integers, strings, and other data types together in one array.

Example:

CODE PLAYGROUND PHP

```php
<?php
$myArray[0] = "John";
$myArray[1] = "<strong>PHP</strong>";
$myArray[2] = 21;

echo "$myArray[0] is $myArray[2] and
knows $myArray[1]";

// Outputs "John is 21 and knows PHP"
?>
```

Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.
There are two ways to create an associative array:

```php
$people = array("David"=>"27",
"Amy"=>"21", "John"=>"42");
// or
$people['David'] = "27";
$people['Amy'] = "21";
$people['John'] = "42";
```

In the first example, note the use of the **=>** signs in assigning values to the named keys.

Associative Arrays

Use the named keys to access the array's members.

CODE PLAYGROUND **PHP**

```php
$people = array("David"=>"27",
"Amy"=>"21", "John"=>"42");

echo $people['Amy']; // Outputs 21"

Tap to edit
```

Run the code and see how it works!

Multi-Dimensional Arrays

A **multi-dimensional** array contains one or more arrays.

The dimension of an array indicates the number of indices you would need to select an element.
- For a **two-dimensional** array, you need two indices to select an element
- For a **three-dimensional** array, you need three indices to select an element

Arrays more than three levels deep are difficult to manage.

Multi-Dimensional Arrays

Let's create a two-dimensional array that contains 3 arrays:

```php
$people = array(
   'online'=>array('David', 'Amy'),
   'offline'=>array('John', 'Rob',
'Jack'),
   'away'=>array('Arthur', 'Daniel')
);
```

Now the two-dimensional **$people** array contains 3 arrays, and it has two indices: **row** and **column**.
To access the elements of the $people array, we must point to the two indices.

CODE PLAYGROUND PHP

```php
echo $people['online'][0]; //Outputs
"David"

echo $people['away'][1]; //Outputs
"Daniel"

Tap to edit |
```

The arrays in the multi-dimensional array can be both numeric and associative.

Fill in the blanks to declare a two-dimensional array.

```
<?php

$cars = array (

  'BMW'    Type

 array('X5', 'red', '2013'),

  'AUDI' =>    Type

('A4', 'white', '2008')

);

?>
```

Control Structures

Lesson

The If Else Statement

Lesson

The Elseif Statement

Lesson

The while Loop

Lesson

The Do While Loop

Lesson

The For Loop

Lesson

The Foreach Loop

Lesson

The Switch Statement

Lesson

The Break Statement

Lesson

The Continue Statement

Lesson

Include & Require

Lesson

Module 5 Quiz

Conditional Statements

Conditional statements perform different actions for different decisions.
The **if else** statement is used to execute a certain code if a condition is **true**, and another code if the condition is **false**.

Syntax:

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

code if a condition is **true**, and another code if the condition is **false**.

Syntax:

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

You can also use the **if** statement without the **else** statement, if you do not need to do anything, in case the condition is **false**.

Fill in the blanks to output "Welcome", if the variable 'age' is greater than 18.

```
Type ($age Type 18)
{
    Type "Welcome";
}
```

If Else

The example below will output the greatest number of the two.

CODE PLAYGROUND **PHP**

```php
<?php
$x = 10;
$y = 20;
if ($x >= $y) {
   echo $x;
} else {
   echo $y;
}

// Outputs "20"
?>
```

Tap to edit |

The Elseif Statement

Use the **if...elseif...else** statement to specify a **new** condition to test, if the first condition is **false**.

Syntax:

```php
if (condition) {
  code to be executed if condition is
true;
} elseif (condition) {
  code to be executed if condition is
true;
} else {
   code to be executed if condition is
false;
}
```

You can add as many **elseif** statements as you want. Just note, that the **elseif** statement must begin with an **if** statement.

How many times can an elseif statement be used?

- `As many as you want`
- `None`
- `One`

The Elseif Statement

For example:

CODE PLAYGROUND PHP

```php
<?php
$age = 21;

if ($age<=13) {
   echo "Child.";
} elseif ($age>13 && $age<19) {
   echo "Teenager";
} else {
   echo "Adult";
}

//Outputs "Adult"
?>
```

We used the **logical AND** (&&) operator to combine the two conditions and check to determine whether $age is between 13 and 19.

The curly braces can be omitted if there only one statement after the **if/ elseif/else**.

For example:

```
if($age<=13)
  echo "Child";
else
  echo "Adult";
```

Fill in the blanks to print "Male" if the variable "gender" is equal to 0, "Female" if it is equal to 1, and "Undefined" in all other cases.

```
if($gender ==  [ Type ] )
   echo "Male";
[ Type ] ($gender [ Type ] 1)
   echo "Female";
[ Type ]
   echo "Undefined";
```

Loops

When writing code, you may want the same block of code to run over and over again. Instead of adding several almost equal code-lines in a script, we can use **loops** to perform a task like this.

[h2]The while Loop[/h2]

The **while** loop executes a block of code as long as the specified condition is **true**.

Syntax:

```php
while (condition is true) {
   code to be executed;
}
```

If the condition never becomes **false**, the statement will continue to execute indefinitely.

What is the syntax for the while loop?

```
while { } :
```

```
while () => { }
```

```
while ( ) { }
```

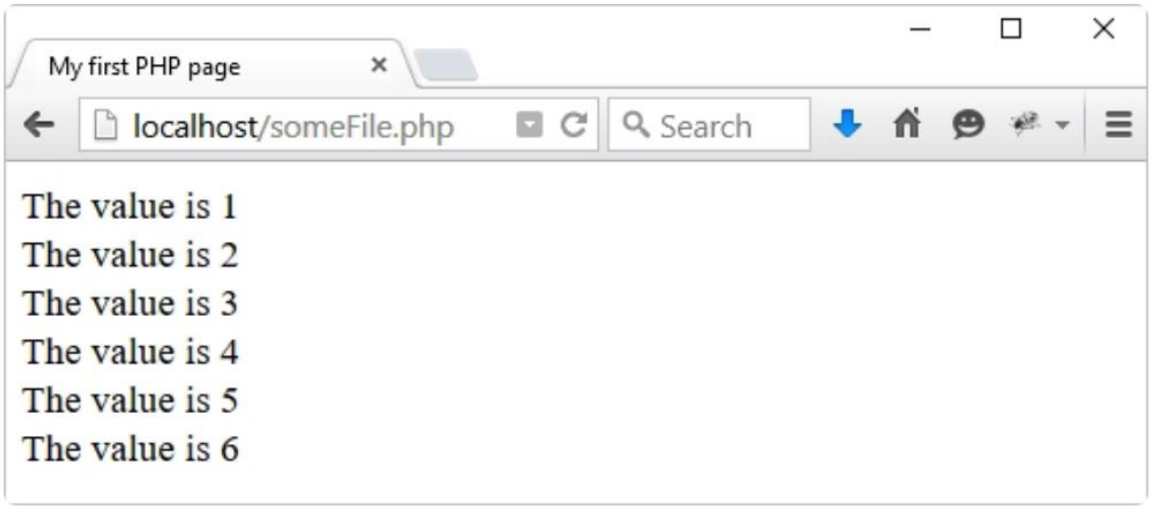
The while Loop

The example below first sets a variable $i to one ($i = 1). Then, the **while** loop runs as long as $i is less than seven ($i < 7). $i will increase by one each time the loop runs ($i++):

CODE PLAYGROUND PHP

```php
$i = 1;
while ($i < 7) {
  echo "The value is $i <br />";
  $i++;
}
```

Tap to edit


My first PHP page
localhost/someFile.php
Search
The value is 1
The value is 2
The value is 3
The value is 4
The value is 5
The value is 6

Fill in the blanks to print the numbers 1 to 15 to the screen.

```
$i = 1;

while($i <=    Type    )   {

    echo $i;

    $i   Type   ;

}
```

The do...while Loop

The **do...while** loop will always execute the block of code once, check the condition, and repeat the loop as long as the specified condition is true.

Syntax:

```php
do {
  code to be executed;
} while (condition is true);
```

Regardless of whether the condition is **true** or **false**, the code will be executed at least **once**, which could be needed in some situations.

How many times will the following code print "Hello"?

```
$i = 1;
do {  echo "Hello"; }
while($i < 0);
```

None

1

Infinite

The do...while Loop

The example below will write some output, and then increment the variable $i by one. Then the condition is checked, and the loop continues to run, as long as $i is less than or equal to 7.

CODE PLAYGROUND PHP

```php
$i = 5;
do {
   echo "The number is " . $i . "<br/>";
   $i++;
} while($i <= 7);

//Output
//The number is 5
//The number is 6
//The number is 7
```

Note that in a **do while** loop, the condition is tested AFTER executing the statements within the loop. This means that the **do while** loop would execute its statements at least once, even if the condition is false the first time.

Rearrange the blocks to form a correct do while loop.

```
$i = 0;
```

```
do {
```

```
} while ($i < 10);
```

```
$i++;
```

The for Loop

The **for** loop is used when you know in advance how many times the script should run.

```php
for (init; test; increment) {
   code to be executed;
}
```

Parameters:
init: Initialize the loop counter value
test: Evaluates each time the loop is iterated, continuing if evaluates to **true**, and ending if it evaluates to **false**
increment: Increases the loop counter value

Each of the parameter expressions can be empty or contain multiple expressions that are separated with **commas**.
In the **for** statement, the parameters are separated with **semicolons**.

Which is the correct syntax for the classic for loop?

```
for (expr1)
```

```
for (expr1; expr2; expr3)
```

```
for (expr1; expr2)
```

The for Loop

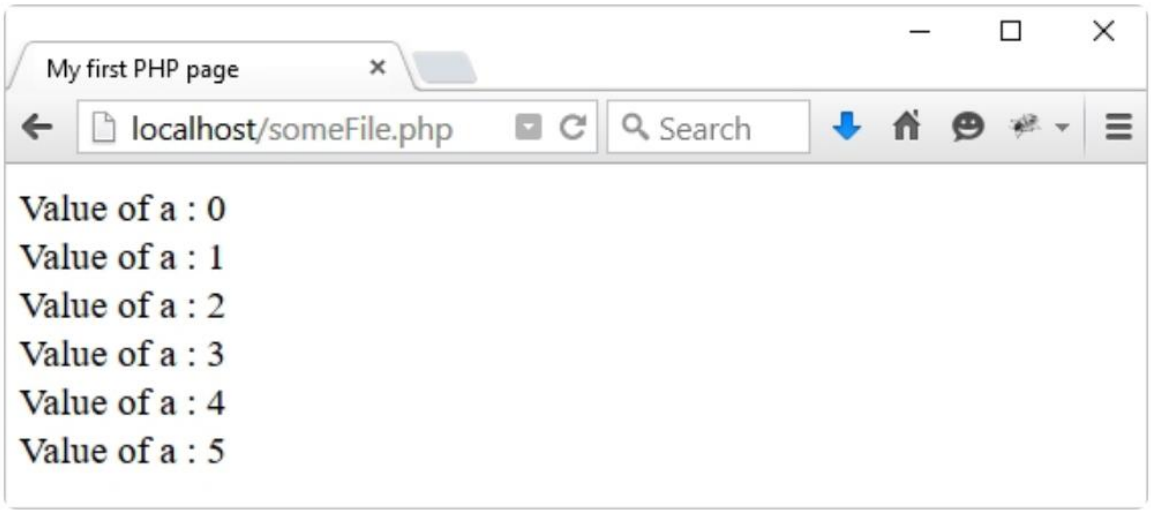
The example below displays the numbers from 0 to 5:

CODE PLAYGROUND PHP

```php
for ($a = 0; $a < 6; $a++) {
   echo "Value of a : ". $a . "<br />";
}

Tap to edit |
```

Result:



The **for** loop in the example above first sets the variable **$a** to 0, then checks for the condition ($a < 6). If the condition is true, it runs the code. After that, it increments $a ($a++).

Drag and drop from the options below to print the variable "name" 10 times.

```
$name = "Bruce";

for(      ;      ;      )   {

   echo $name;

}
```

$i=0 $i<10 $i++

The foreach Loop

The **foreach** loop works only on arrays, and is used to loop through each key/value pair in an array.

There are two syntaxes:

```php
foreach (array as $value) {
  code to be executed;
}
//or
foreach (array as $key => $value) {
   code to be executed;
}
```

The first form loops over the array. On each iteration, the value of the current element is assigned to **$value**, and the array pointer is moved by one, until it reaches the last array element. The second form will additionally assign the current element's key to the **$key** variable on each iteration.

The following example demonstrates a loop that outputs the values of the **$names** array.

CODE PLAYGROUND **PHP**

```php
$names = array("John", "David", "Amy");
foreach ($names as $name) {
    echo $name.'<br />';
}

// John
// David
// Amy
```

Fill in the blanks to print the elements of the array using the foreach loop.

```
$items = array("one", "two", "three");

Type ($items Type $item) {

   echo $ Type ."<br />";

}
```

The switch Statement

The **switch** statement is an alternative to the **if-elseif-else** statement.
Use the **switch** statement to select one of a number of blocks of code to be executed.

Syntax:

```php
switch (n) {
  case value1:
    //code to be executed if n=value1
    break;
  case value2:
     //code to be executed if n=value2
     break;
  ...
  default:
    // code to be executed if n is
```

First, our single expression, **n** (most often a variable), is evaluated once. Next, the value of the expression is compared with the value of each case in the structure. If there is a match, the block of code associated with that case is executed.

Using **nested if else statements** results in similar behavior, but switch offers a more elegant and optimal solution.

The Switch statement is a replacement for...

`while loop`

`do while loop`

`if elseif else statement`

Switch

Consider the following example, which displays the appropriate message for each day.

PHP | OUTPUT

```php
<?php
    $today = 'Tue';

    switch ($today) {
        case "Mon":
            echo "Today is Monday.";
            break;
        case "Tue":
            echo "Today is Tuesday.";
            break;
        case "Wed":
            echo "Today is Wednesday.";
            break;
        case "Thu":
            echo "Today is Thursday.";
            break;
        case "Fri":
             echo "Today is Friday.";
             break;
        case "Sat":
             echo "Today is Saturday.";
             break;
        case "Sun":
             echo "Today is Sunday.";
             break;
        default:
             echo "Invalid day.";
    }
?>
```

PHP | OUTPUT

Today is Tuesday.

The **break** keyword that follows each case is used to keep the code from automatically running into the next case. If you forget the **break;** statement, PHP will automatically continue through the next case statements, even when the case doesn't match.

Fill in the blanks.

```
$i = 1;

switch($i) {

    Type    "1":

    echo 'One';

        Type   ;

  case "2":

    echo 'Two';

    break;

}
```

default

The **default** statement is used if no match is found.

CODE PLAYGROUND PHP

```php
$x=5;
switch ($x) {
  case 1:
    echo "One";
    break;
  case 2:
    echo "Two";
    break;
  default:
    echo "No match";
}

//Outputs "No match"
```

The **default** statement is optional, so it can be omitted.

What keyword is used to handle cases that are not defined with the case keyword?

Type

Switch

Failing to specify the **break** statement causes PHP to continue executing the statements that follow the **case**, until it finds a **break**. You can use this behavior if you need to arrive at the same output for more than one case.

CODE PLAYGROUND PHP

```php
$day = 'Wed';

switch ($day) {
  case 'Mon':
    echo 'First day of the week';
    break;
  case 'Tue':
  case 'Wed':
  case 'Thu':
    echo 'Working day';
    break;
  case 'Fri':
    echo 'Friday!';
    break;
  default:
    echo 'Weekend!';
}

//Outputs "Working day"

Tap to edit
```

The example above will have the same output if **$day** equals 'Tue', 'Wed', or 'Thu'.

What output results from the following code?

```
$mo = "December";
switch ($mo) {
case "July":
   echo "Summer";
break;
case "January":
case "February":
   echo "Winter";
}
?>
```

Nothing

Winter

Summer

The break Statement

As discussed in the previous lesson, the **break** statement is used to break out of the **switch** when a case is matched.
If the break is absent, the code keeps running.

For example:

CODE PLAYGROUND PHP

```php
$x=1;
switch ($x) {
  case 1:
    echo "One";
  case 2:
    echo "Two";
  case 3:
    echo "Three";
  default:
    echo "No match";
}

//Outputs "OneTwoThreeNo match"
```

Tap to edit

Break can also be used to halt the execution of **for**, **foreach**, **while**, **do-while** structures.

The **break** statement ends the current **for**, **foreach**, **while**, **do-while** or **switch** and continues to run the program on the line coming up after the loop.
A **break** statement in the outer part of a program (e.g., not in a control loop) will stop the script.

Fill in the blanks to break out of the loop after the number 5 is printed to the screen.

```
for ($i=0;$i<=50;$i++)  {
  echo $i;

  if ($i==  Type  ) {

      Type  ;

  }
}
```

The continue Statement

When used within a looping structure, the **continue** statement allows for skipping over what remains of the current loop iteration. It then continues the execution at the condition evaluation and moves on to the beginning of the next iteration.

The following example skips the even numbers in the **for** loop:

CODE PLAYGROUND PHP

```php
for ($i=0; $i<10; $i++) {
  if ($i%2==0) {
    continue;
  }
  echo $i . ' ';
}

//Output: 1 3 5 7 9
```

You can use the **continue** statement with all looping structures.

Fill in the blanks to print all the numbers from 0 to 15, except for the numbers 10 and 14.

```
for ($i=0; $i<=15;$i++) {

  if ($i==10 || $i  Type  14) {

      Type  ;

  }

  echo $i."<br />";

}
```

include

The **include** and **require** statements allow for the insertion of the content of one PHP file into another PHP file, before the server executes it. Including files saves quite a bit of work. You can create a standard header, footer, or menu file for all of your web pages. Then, when the header is requiring updating, you can update the header include file only.

Assume that we have a standard header file called **header.php**.

```php
<?php
  echo '<h1>Welcome</h1>';
?>
```

Use the **include** statement to include the header file in a page.

```php
<html>
  <body>

  <?php include 'header.php'; ?>

  <p>Some text.</p>
  <p>Some text.</p>
  </body>
</html>
```

The include and require statements allow for the insertion of the content of one PHP file into another PHP file, before the server executes it.

Which is the correct syntax for the include statement in PHP?

```
include 'file.php';
```

```
include-file.php;
```

```
#include <file.php>
```

include

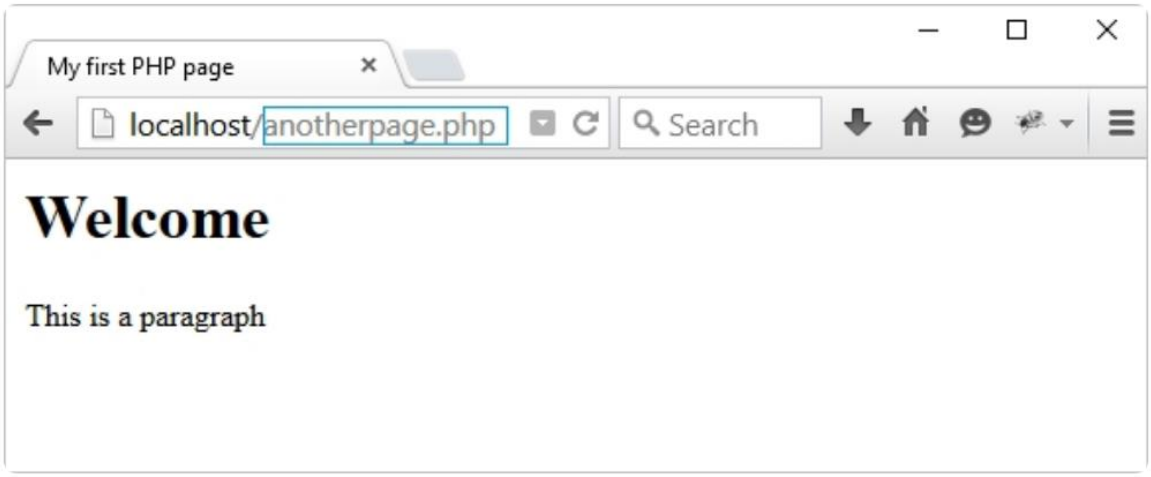
Using this approach, we have the ability to include the same **header.php** file into multiple pages.

```PHP
<html>
  <body>

  <?php include 'header.php'; ?>

  <p>This is a paragraph</p>
  </body>
</html>
```

Result:



Files are included based on the file **path**.
You can use an **absolute** or a **relative** path to specify which file should be included.

Fill in the blank to include the file header.php

Type

include vs require

The **require** statement is identical to include, the exception being that, upon failure, it produces a fatal error.
When a file is included using the **include** statement, but PHP is unable to find it, the script continues to execute.
In the case of **require**, the script will cease execution and produce an error.

Use **require** when the file is required for the application to run.
Use **include** when the file is not required. The application should continue, even when the file is not found.

What is the difference between include and require?

- The way they handle errors
- There is no difference
- The syntax is different

Using the for loop, print only the even numbers between 0 and 10.

```
for ($i=0; $i<=10;$i++) {
        {
    }
}
```

`echo $i."<br/>";` `continue;`

`if ($i%2 != 0)`

Using the for loop, print only the even numbers between 0 and 10.

```
for ($i=0; $i<=10;$i++) {
        {
    }
}
```

```
echo $i."<br/>";
```

```
continue;
```

```
if ($i%2 != 0)
```

What output results from the following code?

```
$x = 0;
while($x<=7) {
  $x++;
}
echo $x;
```

Type

The PHP Else If statement must be preceded by an If statement before it can be used.

True

False

Functions

Lesson

User-Defined Functions

Lesson

Function Parameters

Lesson

The Return Statement

Lesson

Module 6 Quiz

Functions

A **function** is a block of statements that can be used repeatedly in a program.
A function will not execute immediately when a page loads. It will be executed by a call to the function.
A user defined function declaration starts with the word **function**:

```php
function functionName() {
    //code to be executed
}
```

A function name can start with a letter or an underscore, but not with a number or a special symbol.

Function names are NOT case-sensitive.

Which of the following is a correct name for a function?

```
11_hello()
```

```
_hello()
```

```
#hello()
```

```
123()
```

Functions

In the example below, we create the function **sayHello()**. The opening curly brace ({) indicates that this is the beginning of the function code, while the closing curly brace (}) indicates that this is the end.
To call the function, just write its name:

CODE PLAYGROUND PHP

```php
function sayHello() {
  echo "Hello!";
}

sayHello(); //call the function

//Outputs "Hello!"
```

Tap to edit

Drag and drop from the options below to create a function called myFunc and then call it:

```
          () {
    echo "PHP is awesome!";
}

```

myFunc(); myFunc function

Function Parameters

Information can be passed to functions through **arguments**, which are like variables.
Arguments are specified after the function name, and within the parentheses.
Here, our function takes a number, multiplies it by two, and prints the result:

CODE PLAYGROUND PHP

```php
function multiplyByTwo($number) {
  $answer = $number * 2;
  echo $answer;
}
multiplyByTwo(3);
//Outputs 6

Tap to edit |
```

You can add as many arguments as you want, as long as they are separated with **commas**.

CODE PLAYGROUND **PHP**

```php
function multiply($num1, $num2) {
  echo $num1 * $num2;
}
multiply(3, 6);
//Outputs 18

Tap to edit |
```

When you define a function, the variables that represent the values that will be passed to it for processing are called **parameters**. However, when you use a function, the value you pass to it is called an **argument**.

What output results from the following code?

```
function test($num) {
    echo $num/2;
}
test(14);
```

Type

Default Arguments

Default arguments can be defined for the function arguments.
In the example below, we're calling the function **setCounter()**. There are no arguments, so it will take on the default values that have been defined.

CODE PLAYGROUND **PHP**

```php
function setCounter($num=10) {
   echo "Counter is ".$num;
}
setCounter(42);  //Counter is 42
setCounter();  //Counter is 10

Tap to edit
```

When using default arguments, any defaults should be on the right side of any non-default arguments; otherwise, things will not work as expected.

Fill in the blanks to define the function "myMul", which takes two parameters and prints the product of their multiplication.

```
function myMul($num1,  Type  num2)  {

 echo $num1  Type  $num2;

}
```

Return

A function can return a value using the **return** statement.
Return stops the function's execution, and sends the value back to the calling code.

For example:

CODE PLAYGROUND PHP

```php
function mult($num1, $num2) {
  $res = $num1 * $num2;
  return $res;
}

echo mult(8, 3);
// Outputs 24
```

Leaving out the return results in a **NULL** value being returned.
A function cannot return multiple values, but returning an **array** will produce similar results.

Rearrange the code to create a function which returns the sum of its parameters, and call it for 5 and 6 as its parameters.

```
echo sum(5, 6);
```

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

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

```
return $sum; }
```

What output results from the following code?

```
function func($arg)  {
  $result = 0;
  for($i=0; $i<$arg; $i++) {
    $result = $result + $i;
  }
  return $result;
}
echo func(5);
```

Type

Fill in the blanks to output "Welcome Robert".

```
function greet($name)   {

  return "Welcome ".  $   Type   ;

}

Type    greet('Robert');
```

Fill in the blanks to declare a function myFunction, taking two parameters and printing the product of their multiplication to the screen.

```
Type    myFunction($a, $b) {

  echo $a    Type    $b;

}
```

Predefined Variables

Lesson

$_SERVER Variables: Script Name

Lesson

$_SERVER Variables: Host Name

Lesson

PHP Forms

Lesson

GET and POST

Lesson

$_SESSION

Lesson

$_COOKIE

Predefined Variables

A **superglobal** is a predefined variable that is always accessible, regardless of scope. You can access the PHP superglobals through any function, class, or file.

PHP's superglobal variables are $_SERVER, $GLOBALS, $_REQUEST, $_POST, $_GET, $_FILES, $_ENV, $_COOKIE, $_SESSION.

[h2]$_SERVER[/h2]

$_SERVER is an array that includes information such as headers, paths, and script locations. The entries in this array are created by the web server. **$_SERVER['SCRIPT_NAME']** returns the path of the current script:

Our example was written in a file called **somefile.php**, which is located in the root of the web server.

Fill in the blank to print the current script's file path to the screen.

```
<?php

  $addr =  $ Type
SERVER['SCRIPT_NAME'];

  echo $addr;

?>
```

$_SERVER

$_SERVER['HTTP_HOST'] returns the Host header from the current request.

```php
<?php
echo $_SERVER['HTTP_HOST'];
//Outputs "localhost"
?>
```

This method can be useful when you have a lot of images on your server and need to transfer the website to another host. Instead of changing the path for each image, you can do the following: Create a **config.php** file, that holds the path to your images:

```php
<?php
$host = $_SERVER['HTTP_HOST'];
$image_path = $host.'/images/';
?>
```

Use the **config.php** file in your scripts:

```php
<?php
require 'config.php';
echo '<img src="'.
$image_path.'header.png" />';
?>
```

The path to your images is now dynamic. It will change automatically, based on the Host header.

This graphic shows the main elements of $_SERVER.

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

$_SERVER['HTTP_HOST'] returns the Host header from the current request.

What is the $_SERVER variable?

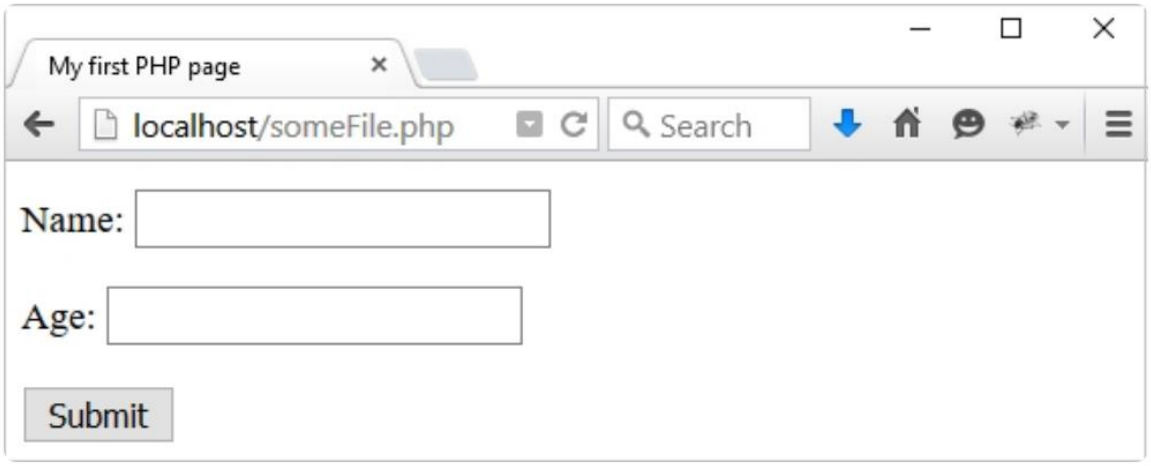
- An integer
- A function
- An array
- A string

Forms

The purpose of the PHP superglobals **$_GET** and **$_POST** is to collect data that has been entered into a form.
The example below shows a simple HTML form that includes two input fields and a submit button:

```HTML
<form action="first.php" method="post">
  <p>Name: <input type="text"
name="name" /></p>
  <p>Age: <input type="text" name="age" /
></p>
  <p><input type="submit" name="submit"
value="Submit" /></p>
</form>
```

Result:



The purpose of the PHP superglobals $_GET and $_POST is to collect data that has been entered into a form.

Which HTML element is needed to collect user input from a web page?

- `hr`
- `div`
- `table`
- `form`

Forms

The **action** attribute specifies that when the form is submitted, the data is sent to a PHP file named **first.php**.
HTML form elements have **names**, which will be used when accessing the data with PHP.

The **method** attribute will be discussed in the next lesson. For now, we'll set the value to "**post**".

Which form attribute indicates the page to which the form is submitted?

- `Action`
- `Method`
- `Input`
- `Submit`

Forms

Now, when we have an HTML form with the **action** attribute set to our PHP file, we can access the posted form data using the **$_POST** associative array.

In the first.php file:

```php
<html>
<body>

Welcome <?php echo $_POST["name"]; ?
><br />
Your age: <?php echo $_POST["age"]; ?>

</body>
</html>
```

The **$_POST** superglobal array holds key/value pairs. In the pairs, keys are the **names** of the form controls and values are the **input data** entered by the user.

We used the **$_POST** array, as the **method="post**" was specified in the form.
To learn more about the form methods, press **Continue**!

Fill in the blanks to submit the form to a page called my.php using the POST method.

```
<form action="  Type  .php  "

    Type  ="post">

 </form>
```

POST

The two methods for submitting forms are **GET** and **POST**.
Information sent from a form via the **POST** method is invisible to others, since all names and/or values are embedded within the body of the HTTP request. Also, there are no limits on the amount of information to be sent.
Moreover, POST supports advanced functionality such as support for multi-part binary input while uploading files to the server.
However, it is not possible to bookmark the page, as the submitted values are not visible.

POST is the preferred method for sending form data.

Fill in the blanks to print the value of a text box named "email", which was submitted using POST.

```
<?php

   echo   $  [ Type ]  POST["  [ Type ]  "];

?>
```

GET

Information sent via a form using the **GET** method is visible to everyone (all variable names and values are displayed in the **URL**). **GET** also sets limits on the amount of information that can be sent - about 2000 characters.
However, because the variables are displayed in the URL, it is possible to bookmark the page, which can be useful in some situations.

For example:

```HTML
<form action="actionGet.php"
method="get">
  Name: <input type="text" name="name" /
><br /><br />
  Age: <input type="text" name="age" /
><br /><br />
  <input type="submit" name="submit"
value="Submit" />
</form>
```

actionGet.php

```php
<?php
echo "Hi ".$_GET['name'].". ";
echo "You are ".$_GET['age']." years
old.";
?>
```

Now, the form is submitted to the **actionGet.php**, and you can see the submitted data in the **URL**:

Sessions

Using a **session**, you can store information in variables, to be used across multiple pages.
Information is not stored on the user's computer, as it is with **cookies**.
By default, session variables last until the user closes the browser.

[h2]Start a PHP Session[/h2]

A session is started using the **session_start()** function.
Use the PHP global **$_SESSION** to set session variables.

```php
<?php
// Start the session
session_start();

$_SESSION['color'] = "red";
$_SESSION['name'] = "John";
?>
```

Now, the **color** and **name** session variables are accessible on multiple pages, throughout the entire session.

The **session_start()** function must be the very first thing in your document. Before any HTML tags.

Type in the function that must be called before working with the session variables.

Type

Session Variables

Another page can be created that can access the session variables we set in the previous page:

```php
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
echo "Your name is " .
$_SESSION['name'];
// Outputs "Your name is John"
?>
</body>
</html>
```

Your session variables remain available in the **$_SESSION** superglobal until you close your session.
All global session variables can be removed manually by using **session_unset()**. You can also destroy the session with **session_destroy()**.

Rearrange the code to declare the variable name, add it to the session, and then print it to the screen.

```
echo $_SESSION['name'];
```

```
$_SESSION['name'] = $name;
```

```
$name = "Alex";
```

```
session_start();
```

Cookies

Cookies are often used to identify the user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page through a browser, it will send the cookie, too. With PHP, you can both create and retrieve cookie values.

Create cookies using the setcookie() function:

```php
setcookie(name, value, expire, path,
domain, secure, httponly);
```

name: Specifies the cookie's name
value: Specifies the cookie's value
expire: Specifies (in seconds) when the cookie is to expire. The value: time()+86400*30, will set the cookie to expire in 30 days. If this parameter is omitted or set to 0, the cookie will expire at the end of the session (when the browser closes). Default is 0.
path: Specifies the server path of the cookie. If set to "/", the cookie will be available within the entire domain. If set to "/php/", the cookie will only be available within the php directory and all sub-directories of php. The default value is the current directory in which the cookie is being set.

domain: Specifies the cookie's domain name. To make the cookie available on all subdomains of example.com, set the domain to "example.com".
secure: Specifies whether or not the cookie should only be transmitted over a secure, HTTPS connection. TRUE indicates that the cookie will only be set if a secure connection exists. Default is FALSE.
httponly: If set to TRUE, the cookie will be accessible only through the HTTP protocol (the cookie will not be accessible to scripting languages). Using httponly helps reduce identity theft using XSS attacks. Default is FALSE.

The **name** parameter is the only one that's required. All of the other parameters are optional.

Where are cookies stored?

On the server

On the user's computer

They are not stored

Cookies

The following example creates a cookie named "user" with the value "John". The cookie will expire after 30 days, which is written as 86,400 * 30, in which 86,400 seconds = one day. The '/' means that the cookie is available throughout the entire website.

We then retrieve the value of the cookie "user" (using the global variable **$_COOKIE**). We also use the **isset()** function to find out if the cookie is set:

```php
<?php
$value = "John";
setcookie("user", $value, time() + (86400
* 30), '/');

if(isset($_COOKIE['user'])) {
  echo "Value is: ". $_COOKIE['user'];
}
//Outputs "Value is: John"
?>
```

The **setcookie()** function must appear BEFORE the <html> tag.
The value of the cookie is automatically encoded when the cookie is sent, and is automatically decoded when it's received.
Nevertheless, **NEVER** store sensitive information in cookies.

Fill in the blanks to set a cookie named 'logged' with the value 1 and an expiration time of 1 hour.

```
setcookie( ' Type ' , Type ,
time()+ Type );
```

Which of the form submit methods should be chosen for a login page?

GET

POST

Should you store sensitive information in cookies?

No

Yes

Fill in the blanks:

```
<form      ="      ">
   <input type="text" name="name" />
   <input type="submit" />
</      >
```

form method GET