PHP (PHP Hypertext Preprocessor)

* allows web developers to create dynamic content that interacts with databases
* Used for developing web based software applications

Syntax

* Starts with **<?php** and ends with **?>**
* You can omit the end tag if it was a pure PHP file.

<?php  
echo "Hello world";  
  
// ... more code  
  
echo "Last statement";  
  
// the script ends here with no PHP closing tag

Comments

* Supports C, C++ and Unix shell style comments

<?php  
    echo 'This is a test'; // This is a one-line c++ style comment  
    /\* This is a multi line comment  
       yet another line of comment \*/  
    echo 'This is yet another test';  
    echo 'One Final Test'; # This is a one-line shell-style comment  
?>

The "one-line" comment styles only comment to the end of the line or the current block of PHP code, whichever comes first. This means that HTML code after // ... ?> or # ... ?> WILL be printed: ?> breaks out of PHP mode and returns to HTML mode, and // or # cannot influence that.

TYPES

PHP supports nine primitive types.

* Four scalar types:
  + [boolean](http://tw2.php.net/manual/en/language.types.boolean.php)
    - This is the simplest type. A [boolean](http://tw2.php.net/manual/en/language.types.boolean.php) expresses a truth value. It can be either TRUE or FALSE.

<?php  
$foo = True; // assign the value TRUE to $foo  
?>

* + [integer](http://tw2.php.net/manual/en/language.types.integer.php)
    - An [integer](http://tw2.php.net/manual/en/language.types.integer.php) is a number of the set ℤ = {..., -2, -1, 0, 1, 2, ...}.

<?php  
$a = 1234; // decimal number  
$a = -123; // a negative number  
$a = 0123; // octal number (equivalent to 83 decimal)  
$a = 0x1A; // hexadecimal number (equivalent to 26 decimal)  
$a = 0b11111111; // binary number (equivalent to 255 decimal)  
?>

* + [float](http://tw2.php.net/manual/en/language.types.float.php) (floating-point number, aka [double](http://tw2.php.net/manual/en/language.types.float.php))
    - Floating point numbers (also known as "floats", "doubles", or "real numbers") can be specified using any of the following syntaxes:

<?php  
$a = 1.234;   
$b = 1.2e3;   
$c = 7E-10;  
?>

* + [string](http://tw2.php.net/manual/en/language.types.string.php)
    - A [string](http://tw2.php.net/manual/en/language.types.string.php) is series of characters, where a character is the same as a byte. This means that PHP only supports a 256-character set, and hence does not offer native Unicode support.
* Four compound types:
  + [array](http://tw2.php.net/manual/en/language.types.array.php)
    - An [array](http://tw2.php.net/manual/en/language.types.array.php) in PHP is actually an ordered map. A map is a type that associates values to keys. This type is optimized for several different uses; it can be treated as an array, list (vector), hash table (an implementation of a map), dictionary, collection, stack, queue, and probably more. As [array](http://tw2.php.net/manual/en/language.types.array.php) values can be other [array](http://tw2.php.net/manual/en/language.types.array.php)s, trees and multidimensional [array](http://tw2.php.net/manual/en/language.types.array.php)s are also possible.

<?php  
$array = array(  
    "foo" => "bar",  
    "bar" => "foo",  
);  
  
// as of PHP 5.4  
$array = [  
    "foo" => "bar",  
    "bar" => "foo",  
];  
?>

* + [object](http://tw2.php.net/manual/en/language.types.object.php)
    - To create a new [object](http://tw2.php.net/manual/en/language.types.object.php), use the new statement to instantiate a class:

<?php  
class foo  
{  
    function do\_foo()  
    {  
        echo "Doing foo.";   
    }  
}  
  
$bar = new foo;  
$bar->do\_foo();  
?>

* + [callable](http://tw2.php.net/manual/en/language.types.callable.php)
  + Iterable
    - It accepts any [array](http://tw2.php.net/manual/en/language.types.array.php) or object implementing the [Traversable](http://tw2.php.net/manual/en/class.traversable.php)interface. Both of these types are iterable using [foreach](http://tw2.php.net/manual/en/control-structures.foreach.php) and can be used with yield from within a [generator](http://tw2.php.net/manual/en/language.generators.php).
    - can be used as a parameter type to indicate that a function requires a set of values, but does not care about the form of the value set since it will be used with [foreach](http://tw2.php.net/manual/en/control-structures.foreach.php). If a value is not an array or instance of [Traversable](http://tw2.php.net/manual/en/class.traversable.php), a [TypeError](http://tw2.php.net/manual/en/class.typeerror.php) will be thrown.

<?php  
  
function foo(iterable $iterable) {  
    foreach ($iterable as $value) {  
        // ...  
    }   
}  
  
?>

* And finally two special types:
  + [resource](http://tw2.php.net/manual/en/language.types.resource.php)
    - A [resource](http://tw2.php.net/manual/en/language.types.resource.php) is a special variable, holding a reference to an external resource. Resources are created and used by special functions.
  + [NULL](http://tw2.php.net/manual/en/language.types.null.php)
    - The special NULL value represents a variable with no value. NULL is the only possible value of type [null](http://tw2.php.net/manual/en/language.types.null.php).
      * A variable is considered to be [null](http://tw2.php.net/manual/en/language.types.null.php) if:
      * it has been assigned the constant NULL.
      * it has not been set to any value yet.
      * it has been [unset()](http://tw2.php.net/manual/en/function.unset.php).

<?php  
$var = NULL;         
?>

* [pseudo-types](http://tw2.php.net/manual/en/language.pseudo-types.php)
  + [mixed](http://tw2.php.net/manual/en/language.pseudo-types.php#language.types.mixed)
    - mixed indicates that a parameter may accept multiple (but not necessarily all) types.
  + [number](http://tw2.php.net/manual/en/language.pseudo-types.php#language.types.number)
    - indicates that a parameter can be either [integer](http://tw2.php.net/manual/en/language.types.integer.php) or [float](http://tw2.php.net/manual/en/language.types.float.php).
  + [callback](http://tw2.php.net/manual/en/language.pseudo-types.php#language.types.callback) (aka [callable](http://tw2.php.net/manual/en/language.types.callable.php))
    - [callback](http://tw2.php.net/manual/en/language.pseudo-types.php#language.types.callback) pseudo-types was used in this documentation before [callable](http://tw2.php.net/manual/en/language.types.callable.php) type hint was introduced by PHP 5.4. It means exactly the same.
  + array|object
    - array|object indicates that a parameter can be either [array](http://tw2.php.net/manual/en/language.types.array.php) or [object](http://tw2.php.net/manual/en/language.types.object.php).
  + void
    - as a return type means that the return value is useless. void in a parameter list means that the function doesn't accept any parameters.

**$** - pseudo variable

* in function prototypes means and so on. This variable name is used when a function can take an endless number of arguments.

VARIABLES

* Variables in PHP are represented by a dollar sign followed by the name of the variable. The variable name is case-sensitive.

<?php  
$var = 'Bob';  
$Var = 'Joe';  
echo "$var, $Var";      // outputs "Bob, Joe"  
  
$4site = 'not yet';     // invalid; starts with a number  
$\_4site = 'not yet';    // valid; starts with an underscore  
$täyte = 'mansikka';    // valid; 'ä' is (Extended) ASCII 228.  
?>

CONSTANT

* A constant is an identifier (name) for a simple value. As the name suggests, that value cannot change during the execution of the script. A constant is case-sensitive by default. By convention, constant identifiers are always uppercase.

<?php  
  
// Valid constant names  
define("FOO",     "something");  
define("FOO2",    "something else");  
define("FOO\_BAR", "something more");  
  
// Invalid constant names  
define("2FOO",    "something");  
  
// This is valid, but should be avoided:  
// PHP may one day provide a magical constant  
// that will break your script  
define("\_\_FOO\_\_", "something");   
?>

EXPRESSIONS

* the most important building blocks of PHP. In PHP, almost anything you write is an expression. The simplest yet most accurate way to define an expression is "anything that has a value".

<?php  
function foo ()  
{  
    return 5;  
}  
?>

OPERATOR

* something that takes one or more values (or expressions, in programming jargon) and yields another value (so that the construction itself becomes an expression).
* can be grouped according to the number of values they take. Unary operators take only one value, for example ! (the [logical not operator](http://tw2.php.net/manual/en/language.operators.logical.php)) or ++ (the [increment operator](http://tw2.php.net/manual/en/language.operators.increment.php)). Binary operators take two values, such as the familiar [arithmetical operators](http://tw2.php.net/manual/en/language.operators.arithmetic.php) + (plus) and - (minus), and the majority of PHP operators fall into this category. Finally, there is a single [ternary operator](http://tw2.php.net/manual/en/language.operators.comparison.php#language.operators.comparison.ternary), ? :, which takes three values; this is usually referred to simply as "the ternary operator" (although it could perhaps more properly be called the conditional operator).

FUNCTIONS

* User-defined functions

<?php  
function foo($arg\_1, $arg\_2, /\* ..., \*/ $arg\_n)  
{  
    echo "Example function.\n";  
    return $retval;  
}  
?>

* Function Arguments

<?php  
function takes\_array($input)  
{  
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];  
}  
?>

* Returning Values

<?php  
function square($num)  
{  
    return $num \* $num;  
}  
echo square(4);   // outputs '16'.  
?>

* Variable functions
  + PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it. Among other things, this can be used to implement callbacks, function tables, and so forth.

<?php  
function foo() {  
    echo "In foo()<br />\n";  
}  
  
function bar($arg = '')  
{  
    echo "In bar(); argument was '$arg'.<br />\n";  
}  
  
// This is a wrapper function around echo  
function echoit($string)  
{  
    echo $string;  
}  
  
$func = 'foo';  
$func();        // This calls foo()  
  
$func = 'bar';  
$func('test');  // This calls bar()  
  
$func = 'echoit';  
$func('test');  // This calls echoit()  
?>