PHP – Quick reference

Basic syntax

PHP section

```
<p>HTML 1</p>
<?php echo 'HTML-output from PHP.'; ?>
<p>HTML 2</p>
```

Variables

```
$name = "Anthony";
$pointerName = &$name;
$a = "bc";
$$a = "a"; // Variable named bc with value a
Usage with arrays: ${$a[1]} or ${$a}[1]
Constant: define("constantA", "value");
```

Predefined variables

$GLOBALS: all variables
$_SERVER: Server and execution environment
$_GET: HTTP GET-variables
$_POST: HTTP POST-variables
$_FILES: HTTP File upload variables
$_REQUEST: HTTP request variables
$argc: Number of passed arguments
$argv: Array of passed arguments
$_SESSION: Session variables
$_ENV: Environment variables
$_COOKIE: HTTP cookies
$php_errormsg: Last error message
$HTTP_RAW_POST_DATA: Raw POST data
$HTTP_response_header: HTTP response header

Functions

Check data types:

is_array, is_binary, is_bool, is_buffer, is_callable, is_double, is_float, is_int, is_integer, is_long, is_null, is_numeric, is_object, is_real, is_resource, is_scalar, is_string, is_unicode

get_ resource_ type, gettype

Check: empty, isset

Cast: floatval, intval, strval, settype

Output: var_ dump, var_ export, print_ r

Delete: unset

Serialization: serialize, unserialize

Escape codes

Symbol	Meaning	ASCII
\n	New line	LF or 0x0A (10)
\r	Carriage return	CR or 0x0D (13)
\t	Horizontal tab	HT or 0x09 (9)
\v	Vertical tab	VT or 0x0B (11)
\f	Form feed	FF or 0x0C (12)
\\	Backslash	
\$	Dollar symbol	
\"	Double quotes	

Array

Declaration

Creation using the array()-construct

```
array array ([ mixed $... ] )
array(  key =>  value,    , ...   )
or
$arr[key] = value;
$arr[] = value;
```

Types

Numeric array

```
// Use of array()
$farben = array("red", "green");
// Explicit setting of values
// zero based index
$farben[] = "red";
$farben[] = "green";
// Access
echo $colors[1]; // "green"
```

Assoziatives Array

```
// Use of array()
$sales =
  array("Jan" => 100,
          "Feb" => 120);
// Explicit setting of values
$sales["Jan"] = 100;
$sales["Feb"] = 120;
// Access
echo $umsaetze["Jan"];
```

Functions

array_flip: Inverts key-value-structure
array_intersect: Creates intersection of elements
array_pop: Retrieves last element
array_push: Adds element to the end
array_rand: Retrieves elements at random
array_reverse: Reverses order of elements
array_search: Searches an array and retrieves keys
array_shift: Retrieves first element
array_slice: Extracts a subset
count: Counts elements
in_array: Checks occurrence of a value
shuffle: Randomizes elements

Control structures

Conditional statement

Conditional statements check a boolean expression and execute statements depending on the true/false output. The expressions can be combined into complex tests by the && (and) or || (or) operators.

```
$expr = true;
if (expr)
    echo "singleStatementInNextRow";

$number1 = 5;
$number2 = 2;
if  ($number1 = 4){
   // statements
}
elseif ($number1 >= 4 && $ number2 < 3){
  // statements
}
else {
 // statements
}

switch ($animal) {

  case "Dog":
  // statements
  break;
  case "Cat":
  // statements
  break;
  default:
  // default statements

}
```

Loops

Loops execute statements repeatedly.
Executes statements until the condition is true.

```
$number = 1;
while ($number <= 5){
 echo $number."<br/>";
 $number++;
}

for ($i = 0; $i < 10; $i++){
 echo $i."<br/>";
}
```

Executes statements once and checks condition afterwards. If the condition is true, it repeats the execution of the statements until the condition becomes false.

```
$number = 10;
do {
 echo $number."<br/>";
 $number ++;
}
while ($number <=5);
```

Traverses the array elements and executes statements for each iteration. The array index can be assigned to a temporary variable.

```
$celBodies = array("sun" => "yellow",
                   "moon" => "pale",
                   "stars" => "white");
foreach ($celBodies as $name => $celBody){
  echo $name.":  ".$celBody."<br/>";
}
```

Exceptions

An exception is a typed error message encapsulated in an object. Base class for specialized exceptions is typically Exception. An exception is thrown (triggered) by throw. A block in which an exception is thrown and handled is surrounded by try {}. The exception is handled in the catch {}-block.

```
// Throwing an exception
throw new Exception("Msg text", 123);

// Method call and exception handling
try {
 // Statements which may cause exception
}
catch (MyException $e){
  echo $e->getMessage()." (".$e->getCode().")";
}
catch (Exception $e){
  echo $e->getMessage()." (".$e->getCode().")";
}

// Declaring a specialized exception
class MyException extends Exception {
  public function __construct($message = "Msg text",
                              $code = 123){
     $this->message = $message;
     $this->code = $code;
  }
  // Additional methods
  ...
}
```

Additional statements

- break stops the execution of a loop.
- continue skips the execution of statements and starts the next iteration.

```
for ($i = 0; $i < 10; $i++){
 if ($i == 2) continue;
 if ($i % 2 == 0 || $i == 5) {
  echo $i."<br/>";
 }
 if ($i > 8) break;
}
```

Script Inclusion

The statements require("file.php") and include("file.php") include PHP files for execution. Upon error, require generates a warning and include a Fatal Error. The statements require_once and include_once perform the inclusion of the same file only once.

Comelio

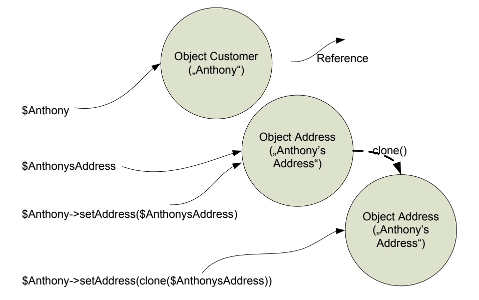
Classes and their members

Classes

```
class Person {
  // Properties
  private $firstName;
  private $surname;
  // Constructor
  public function __construct($f, $s){
    $this->setFirstName($f);
    $this->setSurname($s);
  }
 // Getter and Setter
 public setFirstName($f){
  $this->firstName = $f;
 }
 public getFirstName(){
  return $this->firstName;
 }
 public setSurname($s){
  $this->surname = $s;
 }
 public getSurname(){
  return $this->surname;
 }
}
```

Objects

```
$p = new Person("Anthony",
"Ross");
$p->setFirstName("Tina");
$p->setSurname("Bullock");
print_r($p);
```



Object Customer („Anthony“)
Reference
$Anthony
Object Address („Anthony's Address“)
clone()
$AnthonysAddress
$Anthony->setAddress($AnthonysAddress)
Object Address („Anthony's Address“)
$Anthony->setAddress(clone($AnthonysAddress))

$anthony is an object variable which references an object (instance) of the class Person. Through the assignment of an object to a variable, this variable points to the memory address of this object. The clone()-function instantiates a new object which can then be assigned to a variable.

Constants

Constants store important values, e.g. for tests in conditional statements or parameters, and provide an alternative to “magic values”. They can be accessed via ClassName::Constant and self::Constant (only within class scope).

```
class Customer {
  // Properties
  public $name;

  // Constants
  const CUSTOMER_XML  = "xml";
  const CUSTOMER_TXT  = "txt";

  // Method using constants (2 versions)
  public function getDescription($format){
    switch ($format){
     case Customer::CUSTOMER_XML:
      return "<Customer>".$this-
>name."</Customer>";
      break;
     case self::CUSTOMER_TXT:
      return $this->name."\n";
      break;
    }
  }
}
```

Sichtbarkeit

public : Other classes can access members of the class and class members can access others as well.

private: The access is restricted so that only members can access private members. Other classes and child classes cannot access private members.

protected: Protected members are only accessible from the parent class and its child vclasses. Other classes cannot access them.

Static members

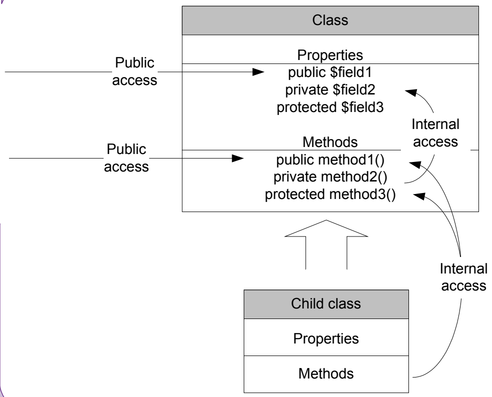
Class members can be accessed through an object (object members) or the class itself (static or class members). Static members are accessed through Class::Member or self::Member and do not need any object instantiation. However, object members are only accessible if an object has been instantiated ($object->member or $this->member).

```
class Customer {
  // Private object properties
  private $name;
  private $nr;
  // Private class property (static)
  private static $counter;
  // Constructor
  public function __construct($name){
    // Set object properties
    $this->setName($name);
    // Increment class property $counter
    self::incrementCounter();
    // Set value of $nr
    $this->setNr(self::$counter);
  }

  // Class methods (static)
  public static function getCounter(){
    return self::$counter;
  }

  private static function incrementCoun-
ter(){
    self::$counter++;
  }

  // Object methods
  // Getter and setter
  ...
}
```

Class hierarchies

Inheritance

Class hierarchies are built using inheritance. Child (sub) classes inherit public and protected members from their parent (super, base) class and can add new members and override inherited ones.
Final public function name() prohibits overriding this method.



Interfaces

An interface defines a set of methods (public API) which need to be implemented by conforming classes. No instantiation is possible. Interface can be derived from parent interfaces using the extends keyword. A class can implement multiple interfaces provided that they do not share method names. The interface type can be used for class hints.

```
// Interface containing public API
interface Exportable {
  public function getXML();
  public function getCSV($separator);
}

// Class implementing interface
class Customer extends Person implements
Exportable {
  ...
  // Implemented methods
  public function getXML(){
    // Implementation
  }
  public function getCSV($separator){
    // Implementation
  }
}
```

Magic methods

__construct(): Constructor
__destruct() : Destructor
__call(string name, array args): triggered by call of an unknown object method
__callStatic(string name, array args): triggered by call of an unknown static method
__get(string name): triggered by read access of unknown property
__set(string name, mixed value): triggered by write access of unknown property
__isset(string name): triggered by isset()
__unset(string name): triggered by unset()
__sleep(): triggered by serialize()
__wakeup():triggered by unserialize()
__toString():triggered by echo
__clone(): triggered during cloning

Abstract classes

An abstract class is a set of both common and abstract/unimplemented methods as well as other class members which cannot be instantiated and is only used in class hierarchies. The class type can be used for class hints.

```
// Abstract parent class
abstract class Person {
  protected $name;
  public function __construct($name){
    $this->setName($name);
  }
  // Abstract method
  abstract function getDescription();
  // Common methods
  ...
}
// Child class
class Customer extends Person {
  private $nr;
  public function __construct($name, $nr){
    parent::__construct($name);
    $this->setNr($nr);
  }
  // Common and implemented methods
  function getDescription() {
   // Implementation
  }
  ...
}
```

Compiled and edited by Marco Skulschus

Berlin, Germany
Miami, FL
info@comelio.com