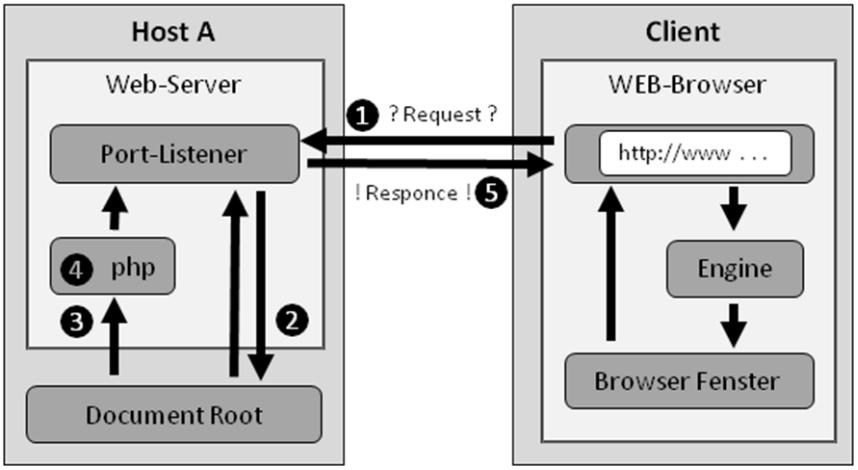
1. Zugriffsschema eines php-Documents

Die serverseitige Scriptsprache php (Kurzform von php Hypertext Preprocessor oder Personal Home Page) ist speziell für den Einsatz im Internet entwickelt. php ist für alle „gängigen“ Betriebssysteme als CGI-Version oder als Modulversion eines WEB-Servers verfügbar.



①	Client-Browser fordert am Server eine Ressource (HTML oder php-Sites) an (SERVER-REQUEST)
②	im WEB-Server wird die Ressource aus dem „Document-root“ geladen (und bei HTML-Sites sofort ausgeliefert)
③	php-Sites werden an den php-Processor übergeben
④	php-Quellcode wird verarbeitet (und aus dem Dokument entfernt oder durch HTML-Ergebnisse substituiert)
⑤	somit liefert der Server die Ressource (reines HTML-Sites) an den Client (RESPONCE) aus

Ablaufschema

1.1. Einbindung von php-Code in HTML-Sites

```
① <script language="php">
②       echo "Ausgabe ";
③ </script>
```

Server Side Scripting

```
① <?php
② echo "Ausgabe " ;
③ ?>
```

SGML-Schreibweise

Damit die serverseitigen php-Scripte korrekt verarbeitet werden können, muss:

- der Dateiextender der HTML-Site *.php lauten
- die Script-Anweisung(en) im Dokument gekennzeichnet sein.
- die Ressource über einen WEB-Server aufgerufen werden.
- die Ressource im DOCUMENT-ROOT des WEB-Servers liegen.

Der php-Quellcode bleibt für den Client grundsätzlich unsichtbar, da nur das Ergebnis der Scriptverarbeitung ausgegeben wird. Theoretisch kann die Scripteinbindung an jeder Stelle (auch mehrmals) innerhalb des Dokumentes erfolgen. Wenn das Script selber Ausgaben erzeugt (z.B. durch `echo "Ausgabe ";`), sollten diese Teile innerhalb des HTML-Bodys (`<body> ... </body>`) stehen.

```
① phpinfo();  // einzeiliger Kommentar
② phpinfo();  /*
③       mehrzeiliger Kommentar
④             */ echo "ist zu ende";
```

Syntax Kommentar

1.2. Kommentare in php

können auch genutzt werden um Abschnitte des Quellcodes zu Testzwecken auszublenden, da dieser nicht geparst wird.

2. Datentypen und Variablen

php unterstützt die Datentypen: Boolean, Integer, Fließkomma-Zahlen und String sowie die zusammengesetzten Typen: Array und Object, und arbeitet mit einer **impliziten Variablendeklaration**. Das bedeutet, die Zuweisung eines Wertes an einen Bezeichner (der in php immer mit einem $-Zeichen beginnt) deklariert und initialisiert eine Variable (des zugewiesenen Typs). Da es bei dieser (Scriptsprachen typischen) Art zu einigen Fehler kommen kann, gibt es eine Reihe von vordefinierten Funktionen im Umgang mit Variablen. Eine explizite Typumwandlung wird durch die Angabe des gewünschten Datentyps (in Klammern) erreicht.

①	`$wert_vor = 3.14;`
②	`$wert_nach = (int)$wert_vor;`
③	`echo "Erg = ".$wert_nach;`

Typenkonvertierung

Folgende Umwandlungen sind möglich:

- (int), (integer) - nach integer
- (bool), (boolean) - nach boolean
- (float), (double), (real) - nach float
- (string) - nach string

2.1. Wichtige Funktionen im Umgang mit Variablen

Funktion	Rückgabe	Beschreibung
`gettype($var);`	"int", "string", "array" . . . "unknow type",	ermittelt den Typ einer Variable
`isset($var);`	TRUE ➞ $var angelegt / FALSE ➞ sonst	prüft, ob eine Variable existiert
`unset($var);`	keine Rückgabe ($var wird entfernt)	löscht eine Variable
`empty($var);`	TRUE ➞ $var ist 0 , leer oder nicht dekl. / FALSE ➞ sonst	prüft, ob eine Variable leer ist
`var_dump($var)`	keine Rückgabe sonder Ausgabe	gibt alle Informationen einer V. aus
`is_int($var)`	TRUE ➞ $var ist Integer FALSE ➞ sonst	prüft, ob Variable vom Typ Integer ist

Vordefinierte Variablen
werden in php über die REGISTER-GLOBALS übergeben und werden (abhängig von der Konfigurations-Datei php.ini) meist als Assoziativ-Array zur Verfügung gestellt.

Bezeichner	Beschreibung → Variablen die ...
`$_GET[]`	über HTTP GET geliefert werden.
`$_POST[]`	über HTTP POST geliefert werden.
`$_FILES[]`	über HTTP Post Datei-Uploads geliefert werden.
`$_COOKIE[]`	über HTTP Cookies geliefert werden.
`$_SESSION[]`	aktuell in der Session registriert sind.

Vordefinierte Konstanten
die zur Laufzeit zur Verfügung stehen

`__FILE__`	der vollständige Pfad- und Dateiname dieser Datei
`__DIR__`	der Verzeichnisname, in dem sich die Datei befindet.

2.2. Ausgabe von Daten (Strings) und Variablen

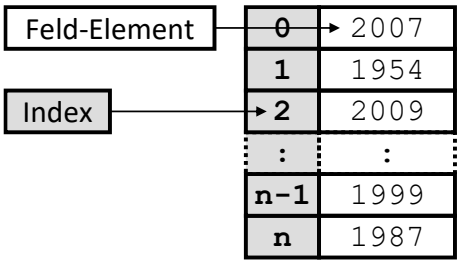
Zeichenketten können zur Ausgabe oder Zuweisung einer String-Variable mit doppelten ("") oder einfachen ('') Anführungszeichen begrenzt werden. Dabei werden **nur** bei doppelten Hochkommas Variablen innerhalb einer Zeichnenkette durch ihren Wert substituiert. Der Punkt-Operator ist Verkettungsoperator.
Zur Maskierung von Zeichen wird Backslash verwendet (z.B. \n ➞ New Line oder \t ➞ Tabulator bzw. \").

Beispiel : Hyperlink `<a href="http://www.web.de">http://www.web.de</a>`

	`$URL = 'http://www.web.de';`	Zuweisung der URL an eine (String)Variable
①	`echo '<a href="'.$URL.'" >'.$URL.'</a>';`	
	Ausgabe in mehreren (fünf) Teilen mit Verkettungsoperator	
②	`echo "<a href=\"$URL\" >$URL</a>";`	
	Ausgabe in einem String mit substituierter Variable und quotiertem Hochkomma der Attributs-Angabe von href=""	

3. Arrays (Felder)

Durch Arrays (Felder) ist es möglich, mehrere Daten (gleichen oder unterschiedlichen Typs) in einer Struktur zu speichern. Über den Index besteht ein wahlfreier Zugriff auf die einzelnen Daten (Array-Elemente oder Feld-Elemente).
Sie werden nach der Index-Art unterschieden.
(numerisch → Integer / assoziativ → String).



3.1. Erzeugen von Arrays

Arrays werden (wie alle php-Variblen) nicht explizit sondern implizit (durch den 1. Aufruf) deklariert. Dabei kann wahlweise der Befehl **`array`** oder eine einfache Wertzuweisung verwendet werden.

```
① $Array_Bezeichner = array([IndexA =>]WertA , [IndexB =>]WertB , ... );
                                                          Array-Deklaration
```

Diese Angaben sind kombinierbar, was zu verschiedenen Effekten (siehe Beispiele) führen kann.

	Arraydeklaration $A1 bis $A4
①	`$A1=array('A','B','C','D');`
②	`$A2=array(2=>'E' ,4=>'F',7=>'G','H');`
③	`$A3=array('IA'=>'A','IB'=>6,'IC'=>'C','ID'=>3);`
④	`$A4=array('IE'=>'E',6,'IF'=>'F',3);`

$A1	
0	A
1	B
2	C
3	D

$A2	
2	E
4	F
7	G
8	H

$A3	
IA	A
IB	6
IC	C
ID	3

$A4	
IE	E
0	6
IF	F
1	3

3.2. Zugriff auf Array-Inhalte

Auf einzelne Array-Elemente kann wie auf primitive Variablen zugegriffen werden. Zusätzlich zum Bezeichner muss nur der Index in eckigen Klammern (und ggf mit Hochkommas) angegeben werden.

```
① $name[Index] = Wert;    /* schreiben */
② $var = $name[Index];    /* lesen */
③ echo $name[Index];      /* ausgeben */
```

`$name` : Arraybezeichnung
`Index` : Selektion einzelner Zellen (numerisch oder assoziativ)
`Wert` : Zelleninhalt

3.3. Mehrdimensionale Arrays

Jedes Array-Element kann (entweder primitive Datentypen) oder wiederum Arrays enthalten. Daraus ergeben sich mehrdimensionale Arrays (in der Praxis reicht meist der Umgang mit zweidimensionalen Arrays).

```
① $A1=array('A','B','C','D')
② $A2=array('E','F','G','H')
③ $A3=array('I','J','K','L')
④ $A=array($A1,$A2,$A3);
```

Der Zugriff auf die einzelnen Array-Elemente erfolgt durch die Angabe beider Indizes (in Reihenfolge), so dass
`echo $A[1][2];` G ausgeben würde.

$A	**0**	**1**	**2**	**3**
0	A	B	C	D
1	E	F	G	H
2	I	J	K	L

Auch hier kann numerische und assoziative Indizierung kombiniert werden (z.B.: `$B[7]['ID']`).

Hinweis : In php ist es möglich, auf Stringvariablen auch als numerisches Array zuzugreifen. Dies stellt eine elegante und einfache Weise der Stringverarbeitung und Stringmanipulation dar.

```
$Name = 'Rimba';
$Name[0] → 'R'     $Name[1] → 'i'  usw.
```
auch Zuweisungen wie : `$Name[0] = 'B'`
sind syntaktisch zulässig.

4. Operatoren

Die Rangfolge der Operatoren entspricht den allgemeinen Regeln (z.B. Punkt vor Strich etc.).

Arithmetisch	`$a + $b`	Summe von $a und $b.
	`$a - $b`	Differenz von $a und $b.
	`$a * $b`	Produkt von $a und $b.
	`$a / $b`	Quotient von $a und $b.
	`$a % $b`	Rest von $a durch $b (ganz).

Logische	UND	`$a and $b`
	ODER	`$a or $b`
	ausschließlich ODER	`$a xor $b`
	NICHT	`! $a`
	UND	`$a && $b`

Vergleich	`$a == $b`	gleich
	`$a === $b`	identisch
	`$a != $b`	ungleich
	`$a !== $b`	nicht identisch
	`$a < $b`	kleiner als (bzw. >)
	`$a <= $b`	kleiner gleich als (bzw. >=)

Bit-Operatoren	UND	`$a & $b`
	ODER	`$a | $b`
	ausschließlich ODER (XOR)	`$a ^ $b`
	NICHT	`~ $a`
	nach links verschieben	`$a << $b`
	nach rechts verschieben	`$a >> $b`

5. Kontrollstrukturen

php kennt für die meisten Kontrollstrukturen (funktional-identische) Syntax-Varianten um den Umsteigern aus anderen Sprachen den Einstieg in php zu erleichtern.

①	`if (Bedingung)`
②	`{ Anweisung(en);}`
③	`[else { Anweisung(en);}]`
	ein [zwei]-seitige Verzweigung

①	`while (Bedingung)`
②	`{ Anweisung(en);}`
	Kopfgesteuerte bedingte Schleife

①	`do { Anweisung(en); }`
②	`while (Bedingung)`
	Fussgesteuerte bedingte Schleife

①	`for(Init;Bedingung;Inkrement)`
②	`{ Anweisung(en);}`
	Zählergesteuerte Schleife

①	`foreach($A as [$key=>] $val)`
②	`{ Anweisung(en); }`
	Array Schleife

6. Unterprogramme und Modularisierung

6.1. Externe php-Code Dateien einbinden

Zur Modularisierung und Mehrfachverwendung von php-Code kann dieser auch in externe Dateien ausgelagert werden. An der entsprechenden Stelle im Quellcode des Hauptprogramms wird mittels einer Anweisung der Quelltext der externen Datei (an genau dieser Stelle) eingebunden.

Zu beachten ist, dass der Parsermodus für die Einbindung dieser Dateien auf `html` steht. Somit muss der php-Code in dieser Datei wieder mit **`<?php`** bzw. **`?>`** gekennzeichnet sein (und die Datei muss die Erweiterung *.php haben). Dabei werden vier Varianten des Einbindens unterschieden:

	Syntax	Bedeutung
①	**`include_once`**`('Datei.php');`	Bindet noch nicht eingebundene Dateien, **wenn notwendig** ein.
②	**`require_once`**`('Datei.php');`	Bindet noch nicht eingebundene Dateien **auch wenn nicht notwendig** ein.
③	**`include`**`('Datei.php');`	Bindet nur **wenn notwendig** Datei ein
④	**`require`**`('Datei.php');`	Bindet Datei **immer** ein

6.2. Unterprogramme (Funktionen) verwenden

Ein wesentlicher Grund für die rasante Verbreitung und den Einsatz von php ist die große Anzahl von vordefinierten Funktionen, die dem Programmierer zur Verfügung gestellt werden. Eine gute Übersicht bietet beispielsweise die Seite „http://www.selfphp.de“. Für den Programmierer stellt sich eine vordefinierte Funktion wie eine „Blackbox“ dar, von der nur Funktionsname, Übergabe- und Rückgabeparameter bekannt sein müssen. Daraus leitet sich eine typische Syntaxbeschreibung ab:

MachWas liefert die in `$Wort` übergebene Zeichenkette zurück, bei der ab unter `$Pos` angegebenen Position ein Zeichen in Großbuchstaben umgewandelt wird. Optional kann mit `$Anz` die Zahl der umzuwandelnden Zeichenanzahl größer als eins bestimmt werden.

```
string MachWas(string $Wort , int $Pos [,int $Anz] )
①      ②       ③      ④      ③     ④     ③     ⑤
```

⑤ Eckige Klammern kennzeichnen immer optionale Argumente.

② Funktionsname

④ (meist sinnhafter) Bezeichner des Übergabearguments

① Datentyp des Rückgabewertes

③ Datentyp des Übergabearguments

Mit dieser Darstellung und der dazugehörigen Beschreibung sind alle notwendigen Informationen gegeben.

6.3. Funktionsaufruf

Funktionen werden durch ihren Bezeichner (Name) und die in Klammern angegebenen den Argumenten aufgerufen. Dabei ist zu beachten:

- Anzahl, Typ und Reihenfolge der Argumente muss mit der Funktionsdefinition übereinstimmen.
- Argumente können als Wert oder Variable (die einen entsprechenden Wert enthält) übergeben werden.
- bei Funktionen ohne Argumente muss eine leere Klammer geschrieben werden.
- in php muss der Rückgabewert nicht zwingend verarbeitet (gespeichert oder ausgegeben) werden.

Ein korrekter Funktionsaufruf stellt (bei Rückgabe) einen Wert entsprechenden Datentyps dar und kann selbst wieder als Argument übergeben werden (Iteration). Entsprechend der Syntaxbeschreibung von „MachWas“ sind einige Beispiele nachfolgend angegeben.

	Syntax	return	Beschreibung → Aufruf von MachWas ...
①	`$Wert = MachWas('rimba',2);`	`riMba`	... mit zwei Argumenten und Rückgabespeicherung
②	`echo MachWas('rimba',2,4);`	`riMBA`	... mit optionalem (dritten) Argument und Ausgabe
③	`echo MachWas(MachWas('rimba',0),4);`	`RimbA`	... geschachtelt mit Ausgabe

6.4. Eigene Funktionen deklarieren

Abschließend wird die unter 6.2 beschriebene Funktion deklariert.

```
①  function MachWas($Wort, $Pos, $Anz=1)
②      {
③      for($i=$Pos ; $i<$Pos+$Anz ; $i++)
④          {
⑤          $Wort[$i]=strtoupper($Wort[$i]);
⑥          }
⑦      return $Wort;
⑧      }
```

Funktions-Deklaration

In ① wird mit dem Bezeichner **MachWas** eine Funktion deklariert, die zwei Pflichtargumente (`$Wort`, `$Pos`) erwartet. Optional kann ein drittes Argument `$Anz` übergeben werden(default=1). Die geschweiften Klammern ② und ⑧ schließen den Funktionsrumpf ein. Mit dem Schlüsselwort **`return`** in ⑦ wird der in `$Wort` bearbeitete Wert zurückgeliefert.

- Soll die Funktion keine Rückgabe liefern, entfällt das Schlüsselwort **`return`**.
- Soll die Funktion keine Übergabe haben, muss die Klammer leer bleiben.

7. Dateizugriff

Aus php-Scripten kann ein Dateizugriff auf die Dateiablage des Servers erfolgen. Grundsätzlich entscheidet über die Zugriffsberechtigung das Server-BS. Ein (zeigerorientierter) Zugriff erfolgt immer nach dem Schema:

	Vorgang	Syntax ausgewählter Datei-Funktionen	Anzahl
①	Datei öffnen	`resource fopen ( string $file , string $mode)`	einmalig

②	Datei bearbeiten (Lesen / Schreiben)	`string fgets ( resource $handle )`
		`int fputs ( resource $handle, string $str )`
		`bool feof ( resource $handle )`
			auch mehrmalig

③	Datei schließen	`bool fclose( resource $handle )`	einmalig

Da in einer Serverumgebung ein php-Script(Instanz) gleichzeitig mehrfach ausgeführt werden kann (Multiuser-Zugriff) ist es notwendig, die Zugriffsart einer Datei beim Öffnen zu bestimmen.

Nachstehend ist die Wirkungsweise des **mode**-Arguments in der Funktion **fopen()** beschrieben.

	öffnet zum Schreiben	öffnet zum Lesen	löscht ggf. den Inhalt	erzeugt die Datei ggf.	Dateizeiger auf Anfang	Dateizeiger auf Ende
a	✓			✓		✓
a+	✓	✓		✓		✓
r		✓			✓	
r+	✓	✓			✓	
w	✓		✓	✓	✓	
w+	✓	✓		✓	✓	

8. Parameter - Übergabe an php-Scripte

Damit der Anwender über eine HTML-Seite mit dem Server interagieren kann, ist eine Datenübertragung vom Client (WEB-Browser) zum Server (z.B.: php-Script) notwendig. Dazu stehen zwei mögliche Übertragungsmethoden (**GET** und **POST**) zur Auswahl. Diese unterscheiden sich u.a. in :

GET
Daten werden durch Anhängen an die URL übermittelt
Daten sind in der Adresszeile des Browsers sichtbar und somit änderbar
Größe ist (auf meist 2048 Zeichen) begrenzt

POST
Daten werden im HEADER des Server-Request's gespeichert
Daten werden clientseitig nicht gespeichert (nicht sichtbar im Verlauf oder in Favoriten)
keine Beschränkung der Datengröße

8.1. Linkaufruf mit einer GET-Übertragung (URL in HTML)

Die zu übertragenden Daten werden als Bezeichner-Wert- Kombination in der URL notiert.

①	`<a href="[URL...]target.php?Bez1=Wert1&Bez2=Wert2">GET-Daten</a>`
	Link in der HTML-Datei

8.2. Der Zugriff auf Übergabewerte . .

erfolgt über das assoziative $_GET-Array welches für jeden Bezeichner (assoziativ) einen Eintrag mit dem entsprechenden Wert enthält. Diese müssen im Script explizit gelesen werden.

①	`$Var1=$_GET['Bez1'];`
②	`$Var2=$_GET['Bez2'];`
③	`echo "Werte : $Var1 $Var2";`
	GET-Parameter

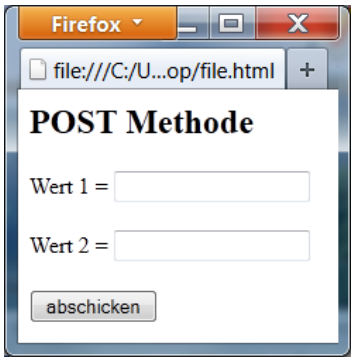
TIPP: Bevor Anwenderdaten in einem php-Script verarbeitet werden, sollten sie aus Sicherheitsgründen unbedingt überprüft werden. In php stehen dazu eine Reihe von Funktionen bereit.

8.3. POST-Übertragung (aus Formularen)

erfolgt meist über ein HTML-Formular mit der eingestellten Methode. Dabei werden die Bezeichner-Wert-Kombinationen aus den name-Attributen der Formular-Elemente sowie den Anwendereingaben (oder Aktionen) gebildet. Der serverseitige Zugriff erfolgt wiederum über ein GLOBAL-ARRAY namens $_POST (assoziativ).

```
① <form action="script.php" method="POST">
② Wert1 = <input type="Text" name="BezA" >
③ Wert2 = <input type="Text" name="BezB" >
④ <input type="Submit" value="schicken" >
⑤ </form>
```

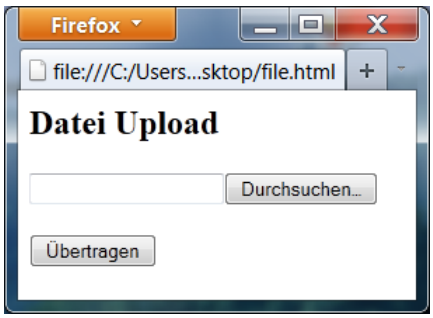
POST-Parameter



Standard (und Fall back) Übertragungsmethode in HTML-Formularen ist GET.

8.4. Dateiupload auf den WEB-Server

Für den Dateiupload (Dateiübertragung vom Client zum Server) ist eine erweiterte HTML-Formulardeklaration notwendig (siehe HTML-Dokumentation). Nach Dateiauswahl-Dialog (Durchsuchen) und Dateiübertragung steht auf dem Server das zweidimensionale Array **`$_FILE['']['']`** mit den übertragenen Daten zur Verfügung




```
① <form action="script.php" method="POST"
②         enctype = "multipart/form-data" >
③    <input type="file" name="Datei" >
④    <input type="submit" >
⑤ </form>
```

FILE-Parameter

Mit der ersten Dimension wird der Name des HTML-File-Steuerelements angesprochen. Des Weiteren stehen folgende Daten zur Verfügung:

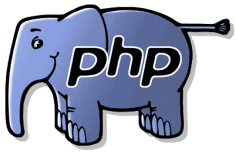
$_FILE Array		
	`$_FILES['Datei']['name']`	Der ursprüngliche Dateiname auf der Client Maschine.
	`$_FILES['Datei']['type']`	Der Mime-Type der Datei (z.B.:*"image/gif"*).
	`$_FILES['Datei']['size']`	Die Größe der hochgeladenen Datei in Bytes.
	`$_FILES['Datei']['error']`	Der Fehlercode im Zusammenhang mit dem Hochladen.
	`$_FILES['Datei']['tmp_name']`	Der Name unter dem die Datei auf dem Server gespeichert ist.

Nach ausführlicher Kontrolle der übertragenen Datei lässt sich diese auf dem Serverdateisystem ablegen.

```
① bool move_uploaded_file(string $filename, string $destination)
② move_uploaded_file($_FILES['Datei']['tmp_name'],'tmp/Bild.jpg')
```

move_file_upload

Häufige Fehlerquellen sind die (selbsterklärenden) unpassenden Parametereinstellungen von **`memory_limit`**, **`upload_max_filesize`** und **`max_execute_time`**, in der Datei php.ini.



A1 nützliche String-Funktionen

	Syntax /Beispiel	Erklärung
1	string **addcslashes**(string, charlist) **addcslashes**(“rz.uni.ka.de”,”.”)= rz\.uni\.ka\.de	Stellt jedem Zeichen aus string das in charlist enthalten ist ein\ voran.
2	string **chr**(int) / int **ord**(string) **chr**(65)=”A” / **ord**(“A”)=65	Wandelt Buchstaben in ASCII –Wert bzw. ASCII in Buchstaben um.
4	mixed **count_chars**(string [,mode]) [mode = 0 bis 4]	Gibt (entsprechend mode) die Auftritts-häufigkeit der Buchstaben in string zurück.
5	int **crc32**(string)	Berechnet polynomischen CRC32-Wert.
6	string **md5**(string)	Verschlüsselt nach der MD5-Methode.
7	string **crypt**(string [,string salt])	Verschlüsselt string (One-Way-Verfahren).
8	array **explode**(separator, string [,int]) string **implode**(string, array)	Teilt string an Hand von separator. Fügt ein Array zu einem string zusammen.
10	string **htmlentities**(string[, int quote_style])	Wandelt Sonderzeichen in HTML-Code.
11	string **ltrim**(string) / **rtrim**(str) / **trim**(str)	Löscht linke/rechte/alle Leerzeichen.
14	string **str_repeat**(string, int)	Verlängert string int mal.
15	string **str_replace**(needle, string, haystack) **str_replace**("@","#","rim@de")=”rim#de”	Sucht in haystack alle Zeichen aus needle und ersetzt sie durch Zeichen aus string.
16	string **strchr**(haystack, needle) string **strrchr**(haystack, needle) **strchr**(“rimba@www@de”,"@")=”@www@de”	Sucht in haystack nach dem 1. (**strchr**) bzw. letzten (**strrchr**)Auftreten von needle und gibt den Rest zurück.
18	int **strpos**(haystack, needle) int **strrpos**(haystack, needle) **strpos**(“rimba@www@de”,"@")= 5	Sucht in haystack nach dem 1. (**strpos**) bzw. letzten (**strrpos**) Auftreten von needle und gibt die Position zurück.
20	int **strlen**(string)	Gibt die Länge von string an.
21	string **strrev**(string)	Dreht string um.
22	string **strtok**(string, needle)	Zerlegt string an der Pos. needle (auch Rek.)
23	string **strtolower**(string) string **strtoupper**(string)	Wandelt string in Kleinbuchstaben (lower) bzw. Grossbuchstaben (upper) um.
24	string **ucfirst**(string) string **ucwords**(string)	Wandelt den ersten Buchstaben (des ersten, oder aller Wörter) in Grossbuchstaben um.
25	string **substr**(string, start [,length])	Gibt Teilstring von start len Zeichen zurück.
26	int **substr_count**(haystack, needle)	Gibt an, wie oft needle in haystack vorkommt.