PHP Web Development with MySQL
A Hands On Approach to Application Programming

by Kenneth E. Marks
# Table of Contents

## Introduction

1. The Life and Times of a PHP Script
   - Static Vs. Dynamic Websites  
   - The Browser and the Server  
   - The Server and PHP  

2. Writing Your First PHP Script
   - Setting Up a Development Environment  
   - Hello World!  
   - Exercises  

3. Why Variables Matter
   - Variables in PHP  
   - Types of Variables  
   - Constants  
   - Exercises  

4. Basic String Interpretation
   - Concatenation  
   - Interpolation  
   - Escaping  
   - Heredoc  
   - Exercises  

5. Operators, Expressions, and Basic Arithmetic
   - Operators and Expressions  
   - Math Functions  

---

**Sample Data:**

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td></td>
<td>XVII</td>
</tr>
<tr>
<td>1. The Life and Times of a PHP Script</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Static Vs. Dynamic Websites</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>The Browser and the Server</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>The Server and PHP</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>2. Writing Your First PHP Script</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Setting Up a Development Environment</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>Hello World!</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>Exercises</td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>3. Why Variables Matter</td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>Variables in PHP</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>Types of Variables</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>Constants</td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>Exercises</td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>4. Basic String Interpretation</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>Concatenation</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>Interpolation</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>Escaping</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>Heredoc</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>Exercises</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>5. Operators, Expressions, and Basic Arithmetic</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>Operators and Expressions</td>
<td></td>
<td>38</td>
</tr>
<tr>
<td>Math Functions</td>
<td></td>
<td>41</td>
</tr>
</tbody>
</table>
## 6. Arrays

- Simple Arrays in PHP 44
- Associative Arrays 45
- Adding Values 46
- Explicit Versus Short Array Syntax 46
- Useful Array Functions 46
- Multidimensional Arrays 52
- Exercises 54

## 7. Truth, Comparisons, Conditions, and Compound Conditions 55

- Comparison Operators 56
- Conditional Logic 59
- Compound Conditional Logic Using Logical Operators 62
- Exercises 64

## 8. Verifying Variables and Type Checking 65

- Verifying Variables 66
- Verifying and Checking Variable Types 68
- Exercises 71

## 9. Looping 73

- Counting Loops 74
- Sentinel Loops 76
- Exiting and Continuing a Loop 78
- Exercises 79

## 10. Functions 81

- Simple Function 82
- Function Parameters/Arguments 83
- Returning Values from a Function 84
- Further Advice On Writing Good Functions 85
- Exercises 88
11. Working with HTML Forms
   A Simple Form 90
   Processing Our Form and Outputting Back to the Web Page 91
   Cleaning It Up Using a Self Referencing Page 93
   Exercise: Badlibs, Part 1 95

12. Inserting Data Into a MySQL Database
   Using the MySQL CLI 98
   Create a PHP Application to Insert Data 108
   Exercises 116

13. Returning Data from a MySQL Database
   Returning Database Rows in a PHP Application 118
   Exercise: Badlibs, Part 2 121

14. Validating Form Data and Creating Sticky Fields
   Modifying FullName Behavior Based On Validation 124
   Adding Field Validation 125
   Making the First and Last Name Fields Sticky 127
   Testing Our Script with Sticky Fields 130
   Exercise: Contact Form 131

15. Displaying a List of Item Details
   Designing the Database 134
   Creating the Database 135
   Adding Movie Data 139
   Creating the Main Movie Listing Page 141
   Creating the Movie Details Page 149
   Exercises 155

16. Adding Data Using the Web Application
   Creating a Page to Add Movies 158
   Complete Code Listing 169
   Link to the “Add a Movie” Page from the Listing Page 174
   Exercises 174
17. Removing Data Using the Web Application
   Adding Deletion Links to Movie Listings 176
   Creating a “Remove a Movie Page” 178
   Complete Code Listing 186
   Exercises 189

18. Editing Data Using the Web Application
   Linking Movie Details to the Edit Page 192
   Create the Editing Page 193
   Complete Code Listing 209
   Exercises 213

19. Working With Files and Feature Additions to Existing Code
   Add a Field for File Information 217
   Create a Folder for Uploaded Movie Image Files 219
   Adding File Upload Capability 219
   Displaying Thumbnail Images of Movies on Main Page 248
   Displaying Movie Image on Details Page 252
   Add Image File Uploads to the Editing Page 256
   Displaying Image on Deletion Page 269
   Exercises 276

20. Basic HTTP Authentication
   Password Protection with HTTP Authentication 278
   How Does HTTP Authentication Work? 279
   Create authorizeaccess.php 280
   Adding Authorization to Pages 283
   Exercises 284

21. Persistence
   Cookies 286
   Session Variables 294
   Cookies and Session Variables 299
   The Database 300
   Best Practices in Solving the Persistence Problem 300
   Exercises 300
### 22. Creating Secure Web Applications

- Secure Password Protection for Authenticating
- Guarding Against SQL Injection
- Leaking Information to Hackers
- Preventing Cross-Site Scripting Attacks
- File Uploads
- Securing Your Session
Program 319
- Final Thoughts
Program 322
- Exercises
Program 322

### 23. Adding User Logins

- Create a user Table
Program 325
- Create a Signup.php script
Program 326
- Create a login.php Script
Program 341
- Create a logout.php Script
Program 351
- Allow Users with Administrative Access
Program 352
- Exercises
Program 356

### 24. Adding a Navigation Menu

- Create Navbar Logic
Program 358
- Add the Navigation Bar
Program 362
- Add Navigation Bar to Details Page
Program 363
- Add Login Link to Navigation Bar
Program 365
- Add Logout Link to Navigation Bar
Program 367
- Add Sign Up Link to Navigation Bar
Program 368
- Add Navigation Bar to addmovie.php
Program 369
- Add Navigation Bar to Unauthorizedaccess.php
Program 372
- Add Navigation Bar to editmovie.php
Program 372
- Add Navigation Bar to removemovie.php
Program 375
- Complete Code Listings
Program 376
- Exercises
Program 378
Chapter 3

Why Variables Matter

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.”

– Martin Golding
3. Why Variables Matter

Variables in PHP

PHP supports several kinds of variables. Predefined variables are defined already by the PHP language, while user-defined variables are defined by you, the developer, in your code. Then there are form variables defined by the name attributes in an HTML form and become keys in a predefined variable.

Valid Variable Names

- PHP variable names must begin with a dollar sign ($).
- A variable name must be at least one character in length.
- The first character after the dollar sign $ can be a letter or an underscore _, and characters after that can be a letter, an underscore, or a number.
- Spaces and special characters other than _ and $ are not allowed in any part of a variable name.

Here are a few examples of valid variable names:

```
$name1
$price_tag
$_abc
$Abc_22
$A23
```

Here are a few examples of invalid variable names:

```
$10names
box.front
$name#last
A-23
$5
```

Recommendations for Naming Your Variables

PHP does have a set of coding standards, which this book follows. You can find them on the PHP-FIG website at PSR-1: Basic Coding Standard[1] and PSR-2: Coding Style Guide[2]. However, the coding standards intentionally give little guidance on how to name your variables. The standards recommend using camelCase for naming your methods, StudlyCaps for class names, and ALL_CAPS separated by underscores for naming constants[3].

I like to use the following conventions when creating variables, functions, methods, constants, and classes in PHP, which I will be using throughout this book:

<table>
<thead>
<tr>
<th>Property</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Names</td>
<td>$snake_case</td>
</tr>
<tr>
<td>Function/Method Names</td>
<td>function camelCase()</td>
</tr>
<tr>
<td>Classes</td>
<td>class StudlyCaps</td>
</tr>
<tr>
<td>Constants</td>
<td>const ALL_CAPS</td>
</tr>
</tbody>
</table>

Regarding naming your variables, a recommended practice is to choose good descriptive names for your variables (e.g., $temperature_fahrenheit). Also, most predefined PHP variables start with a $_ (i.e. $_POST[]). I recommend that you do not create any variables starting with an underscore (_) as this might be confusing to other PHP developers that have to maintain your code.

**Types of Variables**

All data is eventually represented to a computer using 1s and 0s. However, a programming language interpreter or compiler must know the data type representation before correctly converting the data into a format the computer can use. Like several other programming languages (e.g., JavaScript), PHP is a dynamically typed language (as opposed to a statically typed language like Java). A variable will dynamically change its type implicitly based on the data type of the value assigned to it or the context in which it's used.

**Scalar Data Types**

PHP supports the following scalar data types:

- Boolean
- integer
- float
- string
3. **Why Variables Matter**

*Booleans*

A Boolean data type contains a logical value that is either `TRUE` or `FALSE`. Boolean values are typically used in conditional logic statements:

```php
$passed_drivers_license_exam = TRUE;
if ($passed_drivers_license_exam == TRUE)
{
    echo "Award driver's license.<br/>";
}
```

The online PHP documentation has more information about Boolean data types[^4].

*Integers*

An integer data type contains a whole number that can be negative, zero, or positive. They are typically represented in the base-10 number system but can be represented using base 2, 8, 10, or 16.

For more information, see the PHP docs about integer data types[^5].

*Floats*

A floating-point data type contains real numbers which can be expressed either using decimals and/or scientific notation:

```php
// Pascal to Pound per square inch
$pa_to_psi = 0.000145037738;

// Pascal to Pound per square inch
$pa_to_psi = 145037738e-12;

// Pascal to Pound per square inch
$pa_to_psi = 1.45037738e-4;
```

See the PHP documentation for more information about floating point data types[^6].

[^4]: Boolean data types: [http://php.net/language.types.boolean](http://php.net/language.types.boolean)
[^5]: Integer data types: [http://php.net/language.types.integer](http://php.net/language.types.integer)
[^6]: Floating point data types: [http://php.net/language.types.float](http://php.net/language.types.float)
Types of Variables

**Strings**

A string is a group of characters enclosed in either single (‘) or double (“) quotes. The type of opening quote must match the closing quote:

```php
echo "This is a string";
echo 'This is also a string';
echo "This is a string with 'singe-quotes' embedded";
echo 'This is a string with "double-quotes" embedded';
```

If you have a string surrounded by double-quotes (") you can contain a double-quote in your string by escaping it with the back-slash (\). Likewise, you can embed single-quotes by escaping them if they are inside of a string surrounded by single-quotes:

```php
echo "This is a string surrounded by \"";
echo 'I don\'t like using contractions';
```

See the PHP manual for more information about string data types[7].

**Compound Data Types**

PHP defines several “compound” data types which allow you to contain or aggregate multiple pieces of data of the same data type under a single entity. PHP supports the following compound data types:

- array
- object
- callable
- iterable

**Array**

Arrays in PHP are ordered maps, which are a way to associate a key with its corresponding value. Therefore, arrays in PHP are known as “associative arrays.”

An array is created using the `array()` language construct. Here’s how to create an empty array:

```php
$fahrenheit_temperatures = array();
```

3. Why Variables Matter

PHP also supports short array syntax, which lets you define an array like this:

```php
$temperatures = [];
```

To add (or push) values onto the end of an array, use the [] syntax immediately following the variable name. Without specifying a key when adding values to an array, the key will be the next integer value:

```php
$fahrenheit_temperatures[] = 32; // 32 is associated with key 0
$fahrenheit_temperatures[] = 100; // 100 is associated with key 1
```

Keys can be specified using either strings or integers. Associative arrays often use strings as keys to give meaning to the values they associate with in the array. To initialize an array with named keys, use the rocket (=>) operator:

```php
$us_state_captials = array(  
    "Wisconsin" => "Madison",
    "California" => "Sacramento"
);
```

To add a named key to the end of the array, specify it in between []s:

```php
$us_state_captials["Florida"] = "Tallahassee";
```

Note that arrays in keys are unique, so if you specify a key that already exists, you will be replacing its value.

A useful function for viewing the contents of an array is `print_r()`[^8]. Embed `print()` in a set of `<pre>` tags as shown in Listing 3.1.

Listing 3.1.

```
<pre>
<?php
$us_state_captials = array(
    "Wisconsin" => "Madison",
    "California" => "Sacramento"
);
print_r($us_state_captials);
?>
</pre>
```

[^8]: `print_r()`: [http://php.net/print_r](http://php.net/print_r)
This function call produces the following output:

```
Array
(
    [Wisconsin] => Madison
    [California] => Sacramento
)
```

You can find more information on array data types[^9] online.

**Object**

PHP is an “Object-Oriented” programming language, and it allows you to create objects. Objects are created from “class” definitions. Class definitions are like complex types that allow you to group your program data (what your program knows) and your program functions (what your program does) in one place to represent modular components in software better. We will cover object-oriented programming in more detail later in the book.

To create an object, you “instantiate” it from a class definition using the `new` keyword as in Listing 3.2.

**Listing 3.2.**

```
1. <?php
2. 
3. class Radio
4. {
5.     function turnOnRadio()
6.     {
7.         echo "Turning radio on";
8.     }
9. }
10. 
11. $car_radio = new Radio();
12. $car_radio->turnOnRadio();
```

The PHP manual has more information on objects[^10].

[^9]: array data types: [http://php.net/language.types.array](http://php.net/language.types.array)
[^10]: objects: [http://php.net/language.types.object](http://php.net/language.types.object)
Callable

“Callables” can be created in PHP by naming a function to call as a string and invoking it with the `call_user_func()` function\[11\]. You can do this with simple functions, static class methods, and instantiated class methods. The following is a simple example of using a callback.

```php
function exampleCallbackFunction()
{
    echo "Hello world!";
}
call_user_func('exampleCallbackFunction');
```

For more information on callables\[12\], check the online manual.

Iterable

An `iterable`\[13\] is a pseudo-type. It enforces arguments to functions or return values from functions are traversable like arrays. You may see this typehint when looking at the API for PHP functions. It mainly means that you can loop through the variable using a `foreach`.

Special Data Types

PHP defines a couple of special data types as well. These are:

- resource
- NULL

Resource

A “resource” is a special variable containing a reference to an external resource. Resources are typically used for working with files and databases:

```php
$db_connection = mysqli_connect(
    'localhost', 'db_user', 'db_password', 'db_to_use'
);
$file_handle = fopen('file.txt' 'r');
```

For more information on resources\[14\] and their usages, see the online documentation.

\[11\] call_user_func() function: http://php.net/call_user_func
\[12\] callables: http://php.net/language.types.callable
\[13\] iterable: http://php.net/language.types.iterable
\[14\] resources: http://php.net/language.types.resource
NULL

A “NULL” value\textsuperscript{[15]} is a special variable that does not contain a value. A variable is NULL if:

- it is assigned the constant NULL,
- it has not been assigned any value,
- or it has been unset().

Constants

Constants are values that do not change. Named constants are created in PHP using the \texttt{define()}\textsuperscript{[16]} function:

\begin{Verbatim}
\texttt{define("BOILING\_TEMP\_IN\_CELCIUS", 100);}  \\
\texttt{echo BOILING\_TEMP\_IN\_CELCIUS; \// outputs 100}
\end{Verbatim}

Exercises

Create a script \texttt{variables.php} and do the following:

1. Assign numbers to two variables and echo their values.
2. Create a variable to hold a name, echo the string "Hello NAME" where \texttt{NAME} is the value of your variable.
3. Define a constant that represents the acceleration due to gravity (9.81 m/s). Echo the value of this constant.

\textsuperscript{[15]} “NULL” value: \url{http://php.net/language.types.null}
\textsuperscript{[16]} \texttt{define}(): \url{http://php.net/function.define}
Index

A
access credentials, 282
modifiers, 452, 454, 473
privileges, 324–25, 327, 353, 400–401, 403
adminer, 12–14, 98, 112, 136–37, 139, 155, 217, 325, 380, 397
algorithm, 304, 325
password hashing, 304–5
algorithms, encryption, 306
apache, 9–10, 12, 219, 223, 302
apt install, 9
restart, 12
API, 30, 315, 474
external, 88
unsafe JavaScript, 315
application security risk (ASR), 302, 306, 315
associative, 27–28, 45, 53–54, 77, 91, 94, 119, 477
empty, 27, 44, 295
functions, 46–47, 49, 51
indexed, 44
multidimensional, 52–53
numeric, 44–45, 479
short syntax, 28, 44–46
superglobal, 161, 178, 180, 182, 184–85, 192, 202, 205, 271
attack, 306, 315–16, 319
collision, 302
directory path traversal, 319
man-in-the-middle, 321
authentication, 277–80, 282–84, 352
basic, 278–79
headers, 281, 283, 352
B
Bash, 98–101, 106
Bootstrap, 141–42, 149, 160, 164, 178, 183
card, 159, 178
Client-Side Validation, 164, 203, 291, 328, 343
callables, 30, 69
Canonical, 6
CAPTCHAs, 326
characters
special, 19, 24
string escape, 35
class
base, 465–68
constants, 461, 481
definitions, 29, 453
design, 453–54
instantiating, 458
name, 24
naming, 453
CLI (command-line interface), 13, 98, 106
code
conditional, 127, 160
legacy, 109
procedural-based, 313
vulnerable, 317
concatenate, 34, 36, 305, 474
catch-all, 410, 424, 438
elseif, 225, 344
ternary, 366, 372
conditional logic, 56, 59, 93, 288, 313, 330, 476
conditional statements, 59–62, 281
compound, 62–64
constants, 25, 31, 85–86, 142, 223, 225, 229, 461, 475
global, 85
referencing, 461
constructors, 452, 455, 469–73, 475
child, 472
default, 459, 469
cookies, 184, 286–92, 295–300, 314, 459
referencing, 296
saved, 288, 290, 292
values, 288
CrackStation, 303–4
data

- persisting, 300
- untrusted, 315
- user-supplied, 314–15, 322


code, 235, 261

connection, 109–10, 333, 337

error querying, 77, 111, 119–20, 127, 129
errors, 333, 476

northwind, 99–100, 102

queries, 118, 146, 316, 332–33, 337, 473


user account, 13

data type, 25, 27, 57, 102, 333, 476

- Boolean, 26, 68
- complex, 458
- compound, 27
- floating-point, 26
- integers, 26, 68
- special, 30
- string, 27, 68–69, 332, 337

date, 3, 18–19, 286, 302, 325, 449

debugging, 6, 61, 85, 89
destructors, 472
directory

- current, 90
downloads, 13

traversal attacks, 318

Docblocks, 224, 332

e-mail, 19, 318, 470–72, 475–78, 480, 482

email address, 318, 474–75

error

- codes, 222, 332
- connection, 114
- fatal, 467
- last, 333, 337
- output, 113
- error conditions, 225, 237–38
- error handlers, custom, 109
- exception, 358, 452, 475–77, 479, 482, 484
  handling, 476

file

- executable, 222, 320

field, 218, 248–49, 252, 259, 262, 270–71, 381

php.ini, 4, 11

uploading, 319–20

uploads, 146, 222–23, 228, 258, 319

form

- login, 318, 341
- simple, 90

submitted, 93


- anonymous, 205
- call, 29
- header, 332–33
- scoping operators, 86

signature, 86, 333


accessor methods, 454

hash, 302–6, 310, 325, 336, 338–39, 346, 348–49

salted, 305

unique, 304–5

IDEs (integrated development environment), 112, 333

immutability, 454

inheritance, 452, 465, 467–68

integers, 25–26, 28, 41, 44, 68–70, 333, 458

integrated development environment. See IDEs

Java, 25, 323, 456–57

JavaScript, 165, 173, 204, 212, 293, 298, 315–16, 321, 340, 343, 350

- code, 164, 205, 317
- function, 291, 330, 343
L
LastPass, 304
login
  individual, 324
script, 336, 341–45, 347, 349, 353, 365
loop, 73–79, 86, 119–20, 212, 216, 247, 249, 293, 299, 340, 343, 350
  foreach, 75, 78–79, 203
  sentinel, 74, 76–77
M
md5, 303–4
modulus, 39–42
  operator, 41
MySQL
  database, 76–77, 97–120, 309, 475
  documentation, 99
  northwind.sql, 101
  server, 98, 109, 111, 119–20, 151, 180–81, 194–95, 311–13,
  server and client, 10
MySQL CLI, 14, 98–107, 112, 118, 135
N
Northwind SQL file, 77, 99–102
O
Object-Oriented Programming. See OOP
objects, 27, 29, 38–39, 69, 71, 453–56, 458–59, 467, 469–71,
  476, 479
  database results, 333, 337
  immutable, 455
OOP (Object-Oriented Programming), 29, 82, 451–86
Open Web Application Security Project[1]. See OWASP
operator, 28, 34, 37–42, 45, 57
  arithmetic, 39–41
  assignment, 38
  comparison, 56–57
  logical, 61–63
  precedence, 39–40, 62
  rocket, 45
  scope resolution, 461
  scoping, 86–87
  ternary, 60, 202–3, 361
OWASP (Open Web Application Security Project[1]), 302, 315,
  319, 322
OWASP, Cheat Sheet Series, 302
OWASP, Top ten, 302, 306, 315
P
paamayim-nekudotayim, 461
parameters, 68, 82–84, 86–87, 110, 224, 332–34, 337, 398–99,
  470, 475–76, 479–80, 484
  default, 318
  hidden form, 408
  incoming, 476, 483
  malicious, 186, 208
  named, 476
password
  fields, 331, 345
  hash, 303
  hashes, 304
  salted, 304–5
  securing user, 302
  valid, 328, 339, 342, 350
PDO (PHP Data Objects), 108,
  452, 473–76, 478–79, 481–82, 484
PHP Data Objects. See PDO
phpinfo, 4, 20
php.ini
  configuration, 223
  directives, 4
POST
  form variables, 237
  request, 164, 184, 223
  precedence, 39, 62
properties, 25, 46, 452–56,
  458–59, 470, 473, 479–81, 485
  class instance, 472
  object’s, 469
  parent, 465
Q
query parameters, 145–47,
  150–51, 176–78, 180, 182, 192,
  196, 205, 207, 313–14, 318,
  332–33, 337
  sending, 146
query string, 110, 115, 118–19,
  143, 186, 206, 208, 332–33, 337
R
readability, 62, 82, 91
  high, 85
security, 4, 12, 115, 158, 176, 192, 222, 278, 286, 291
risks, 299, 302, 306
web site, 322
security risks, top application, 302
browser, 286–87
cookies, 299, 321
current, 295–99
ID, 299–300, 314, 321
securing, 321
variables, 286, 293–300, 314, 344, 351, 358, 366, 414
session variables
accessing, 294–95
creating, 342
saved, 296, 298
SQL (Standard Query Language), 98–99, 110, 118, 241, 309, 385
SQL commands, 98, 102, 106–7, 312
injection, 115, 306–13, 473
query, 199, 306–7, 312, 332–33, 337
UPDATE query, 206
Standard Query Language. See SQL
superglobal, 93–94, 150, 281, 287, 294–95, 297
COOKIE, 290
POST, 92, 260, 331, 345
SESSION, 294
superuser, 7–10, 13, 15
ternary operation, 60, 420
type, floating point, 26, 68–69
Ubuntu, 6, 8, 10–11, 16
LTS release, 99
Unix, 6–7, 101
UNIX, timestamp, 287
url, 2, 93, 146, 150–51, 314, 348–49, 351
current web page, 93
destination, 185
user logins, 134, 158, 176, 192, 325
individual, 324, 356
adding, 131
client-side, 164
errors, 160, 165
boolean, 125, 165, 330
cookies and session, 286, 299
creating, 25
global, 85
local, 83, 91–92, 457
names, 19, 24–25, 28, 44–45, 75, 453
scope, 85–86
superglobal, 145, 150, 195–96, 205, 222, 225, 259, 280
web server, 2–4, 6, 94, 219, 221–22, 229–31, 319–21
XSS, 315–16, 318, 321
attacks, 315–16, 318