

תכנות PHP 5 Power

◆ פיתוח אפליקציות Java על Linux קארל אלבינג ומייקל שוורץ ◆ תכנות GUI ב-C++ עם Qt 3 ג'סמין בלאנשט, מארק סאמרפילד ◆ ניהול מערכות Linux עם Webmin: ניהול מערכת ופיתוח מודולים ג'יימי קמרון ◆ הבנת מנהל הזיכרון הווירטואלי של Linux מל גורמן ◆ הטמעת CIFS: מערכת קבצי האינטרנט המשותפת כריסטופר הרטל ◆ פיתוח תוכנה למערכות משובצות (Embedded) עם eCos אנתוני מאסה ◆ פיתוח אפליקציות מהיר (RAD) עם Mozilla נייג'ל מקפרלן ◆ פלטפורמת הפיתוח של Linux: הגדרה, שימוש ותחזוקה של סביבת תכנות מלאה רפיק אור רחמן, כריסטופר פול ◆ זיהוי חדירות עם SNORT: טכניקות IDS מתקדמות באמצעות PHP, MySQL, Apache, SNORT ו-ACID רפיק אור רחמן ◆ המדריך הרשמי ל-Samba 3: HOWTO ומדריך ייחוס ג'ון ה. טרפסטר, ילמר ר. ורנואי, עורכים ◆ Samba-3 לפי דוגמה: תרגילים מעשיים לפריסה מוצלחת ג'ון ה. טרפסטר

המחברים והמוציא לאור נקטו משנה זהירות בהכנת ספר זה, אך אינם מספקים אחריות מפורשת או משתמעת מכל סוג שהוא ואינם נושאים באחריות לשגיאות או השמטות. לא תהיה כל חבות לנזקים נלווים או תוצאתיים בקשר לשימוש במידע או בתוכניות הכלולים בספר זה או הנובעים משימוש זה.

מוציא לאור: ג'ון וייט

עורך ראשי: דון או'הגן

עורך רכישות: מארק ל. טאוב

עוזרת מערכת: נורין רג'ינה

עורכת פיתוח: ג'אנט ואלאד

מנהלת שיווק: רובין אובריאן

מעצבת עטיפה: נינה סקודרי

עורכת מנהלת: ג'ינה קנאוס

עורכת פרויקט בכירה: קריסטי הארט

עריכת לשון: Specialized Composition

מפתחת אינדקס: ליסה סטומפף

סדרנית בכירה: גלוריה שוריק

קניין ייצור: דן אוריג

המוציא לאור מציע הנחות מצוינות על ספר זה בהזמנה כמותית לרכישות מרוכזות או מכירות מיוחדות, אשר עשויות לכלול גרסאות אלקטרוניות ו/או עטיפות ותכנים מותאמים אישית לעסק שלך, ליעדי ההכשרה, למיקוד השיווקי ולאינטרסים המיתוגיים שלך. למידע נוסף, אנא צרו קשר עם:

מכירות למגזר העסקי והממשלתי בארה"ב

(800) 382-3419

corpsales@pearsontechgroup.com

למכירות מחוץ לארה"ב, אנא צרו קשר עם:

מכירות בינלאומיות

international@pearsoned.com

בקרו אותנו באינטרנט: www.phptr.com

נתוני קיטלוג בספריית הקונגרס:

2004107331

זכויות יוצרים © 2005 Pearson Education, Inc.

ניתן להפיץ חומר זה רק בכפוף לתנאים ולהתניות המפורטים ברישיון הפרסום הפתוח (Open Publication License), גרסה 1.0 ומעלה (הגרסה העדכנית ביותר זמינה כעת בכתובת <http://www.opencontent.org/openpub>).

Pearson Education, Inc.

One Lake Street

Upper Saddle River, NJ 07458

נעשה כל מאמץ ליצור קשר ולתת קרדיט לכל בעלי זכויות היוצרים. שימוש בחומר ללא קרדיט מתאים אינו מכוון.

ISBN 0-131-47149-X

הטקסט הודפס בארצות הברית על נייר ממוחזר ב-Phoenix שבהאגרטאון, מרילנד.

הדפסה ראשונה, [אוקטובר 2004]

ליפעת, אשתי וחברתי הטובה ביותר, שסבלה בסבלנות את המעורבות שלי ב-PHP מההתחלה, ועודדה ותמכה בי בכל צעד ושעל.

אנדי גוטמנס

למריאן, על הסבלנות והעידוד.

סטיג סאתר באקן

להורי, שדואגים לי גם כשאני לא בסביבה;

ול-42, התשובה לחיים,

ליקום ולכל השאר.

דריק רתאנס

תוכן עניינים

הקדמה מאת זאב סורסקי

פתח דבר: מבוא ורקע

- פרק 1: מה חדש ב-PHP 5?
- פרק 2: שפת הליבה של PHP 5
- פרק 3: תכנות מונחה עצמים (OO) ב-PHP 5
- פרק 4: תכנות מונחה עצמים מתקדם ותבניות עיצוב (Design Patterns) ב-PHP 5
- פרק 5: כיצד לכתוב אפליקציית אינטרנט עם PHP
- פרק 6: בסיסי נתונים עם PHP 5
- פרק 7: טיפול בשגיאות
- פרק 8: XML עם PHP 5
- פרק 9: הרחבות נפוצות (Mainstream Extensions)
- פרק 10: שימוש ב-PEAR
- פרק 11: חבילות PEAR חשובות
- פרק 12: בניית רכיבי PEAR
- פרק 13: ביצוע המעבר (לגרסה 5)
- פרק 14: ביצועים
- פרק 15: מבוא לכתיבת הרחבות ל-PHP
- פרק 16: כתיבת סקריפטים למעטפת (Shell Scripting) ב-PHP
- א. אינדקס חבילות PEAR ו-PECL
- ב. רפרנס פורמט phpDocumentor
- ג. מדריך התחלה מהירה ל-Zend Studio
- אינדקס (מפתח עניינים)

פירוט הפרקים

1. מה חדש ב-PHP 5?
- 1.1 מבוא
- 1.2 מאפייני שפה
- 1.2.1 מודל מונחה עצמים חדש
- 1.2.2 מאפיינים מונחי עצמים חדשים

- 1.2.3 מאפייני שפה חדשים נוספים
- 1.3 שינויים כלליים ב-PHP
 - 1.3.1 XML ושירותי רשת (Web Services)
- 1.4 מאפיינים חדשים נוספים ב-PHP 5
 - 1.4.1 מנהל זיכרון חדש
 - 1.4.2 הפסקת תמיכה ב-Windows 95
- 1.5 סיכום
- 2. שפת הליבה של PHP 5
 - 2.1 מבוא
 - 2.2 הטמעה בתוך HTML
 - 2.3 הערות (Comments)
 - 2.4 משתנים
 - 2.4.1 הפניות עקיפות למשתנים
 - 2.4.2 ניהול משתנים
 - 2.4.3 סופר-גלובליים (Superglobals)
 - 2.5 סוגי נתונים בסיסיים
 - 2.5.1 מספרים שלמים (Integers)
 - 2.5.2 מספרים בנקודה צפה (Floating-Point)
 - 2.5.3 מחרוזות (Strings)
 - 2.5.4 בוליאנים (Booleans)
 - 2.5.5 Null
 - 2.5.6 משאבים (Resources)
 - 2.5.7 מערכים (Arrays)
 - 2.5.8 קבועים (Constants)
 - 2.6 אופרטורים
 - 2.6.1 אופרטורים בינאריים

2.6.2 אופרטורי השמה

2.6.3 אופרטורי השוואה

2.6.4 אופרטורים לוגיים

2.6.5 אופרטורים על סיביות (Bitwise)

2.6.6 אופרטורים אונאריים

2.6.7 אופרטורי שלילה

2.6.8 אופרטורי קידום/הפחתה (Increment/Decrement)

2.6.9 אופרטורי המרה (Cast)

2.6.10 אופרטור ההשתקה (Silence Operator)

2.6.11 האופרטור הטרנארי היחיד במינו

2.7 מבני בקרה

2.7.1 מבני בקרה מותנים (Conditional)

2.7.2 מבני בקרה של לולאות

2.7.3 מבני בקרה להכללת קוד

2.8 פונקציות

2.8.1 פונקציות המוגדרות על ידי המשתמש

2.8.2 טווח הכרזה של פונקציות (Scope)

2.8.3 החזרת ערכים לפי ערך (By Value)

2.8.4 החזרת ערכים לפי הפניה (By Reference)

2.8.5 הכרזה על פרמטרים של פונקציה

2.8.6 משתנים סטטיים

2.9 סיכום

3. תכנות מונחה עצמים (OO) ב-PHP 5

3.1 מבוא

3.2 אובייקטים

3.3 הכרזה על מחלקה (Class)

3.4 מילת המפתח new ובנאים (Constructors)

3.5 מפרקים (Destructors)

3.6 גישה למתודות ומאפיינים באמצעות המשתנה this\$

3.6.1 מאפייני public, protected ו-private

3.6.2 מתודות public, protected ו-private

3.6.3 מאפיינים סטטיים

3.6.4 מתודות סטטיות

3.7 קבועי מחלקה

3.8 שכפול אובייקטים (Cloning)

3.9 פולימורפיזם (רב-צורתיות)

3.10 parent:: ו-self::

3.11 אופרטור instanceof

3.12 מתודות ומחלקות מופשטות (Abstract)

3.13 ממשקים (Interfaces)

3.14 הורשה של ממשקים

3.15 מתודות final

3.16 מחלקות final

3.17 המתודה toString__()

3.18 טיפול בחריגות (Exception Handling)

3.19 autoload__()

3.20 רמזי סוג (Type Hints) בפרמטרים של פונקציות

3.21 סיכום

4. תכנות מונחה עצמים מתקדם ותבניות עיצוב

4.1 מבוא

4.2 יכולות העמסה (Overloading)

4.2.1 העמסת מאפיינים ומתודות

4.2.2 העמסת תחביר גישה למערך

4.3 איטרטורים (Iterators)

4.4 תבניות עיצוב (Design Patterns)

4.4.1 תבנית האסטרטגיה (Strategy Pattern)

4.4.2 תבנית הסינגלטון (Singleton Pattern)

4.4.3 תבנית המפעל (Factory Pattern)

4.4.4 תבנית הצופה (Observer Pattern)

4.5 רפלקציה (Reflection)

4.5.1 מבוא

4.5.2 ממשק תכנות היישומים (API) של הרפלקציה

4.5.3 דוגמאות לרפלקציה

4.5.4 מימוש תבנית ההאצלה (Delegation) באמצעות רפלקציה

4.6 סיכום

5. כיצד לכתוב אפליקציית אינטרנט עם PHP

5.1 מבוא

5.2 הטמעה בתוך HTML

5.3 קלט משתמש

5.4 טיפול בטוח בקלט משתמש

5.4.1 טעויות נפוצות

5.5 טכניקות להפיכת סקריפטים ל"בטוחים"

5.5.1 אימות קלט (Validation)

5.5.2 אימות HMAC

5.5.3 הספריה PEAR::Crypt_HMAC

5.5.4 מסנן קלט

5.5.5 עבודה עם סיסמאות

5.5.6 טיפול בשגיאות

5.6 עוגיות (Cookies)

5.7 סשנים (Sessions)

5.8 העלאת קבצים

5.8.1 טיפול בקובץ המועלה הנכנס

5.9 ארכיטקטורה

5.9.1 סקריפט אחד משרת את כולם

5.9.2 סקריפט אחד לכל פונקציה

5.9.3 הפרדת לוגיקה מעיצוב (Layout)

5.10 סיכום

6. בסיסי נתונים עם PHP 5

6.1 מבוא

MySQL 6.2

6.2.1 נקודות חוזק וחולשה של MySQL

6.2.2 ממשק ה-PHP

6.2.3 נתוני דוגמה

6.2.4 חיבורים

6.2.5 שאלות מאוחסנות (Buffered) לעומת לא מאוחסנות

6.2.6 שאלות

6.2.7 הצהרות מרובות (Multi Statements)

6.2.8 מצבי שליפה (Fetching Modes)

6.2.9 הצהרות מוכנות (Prepared Statements)

6.2.10 טיפול ב-BLOB

SQLite 6.3

6.3.1 נקודות חוזק וחולשה של SQLite

6.3.2 תחומי השימוש הטובים ביותר

6.3.3 ממשק ה-PHP

PEAR DB 6.4

PEAR DB השגת 6.4.1

6.4.2 יתרונו וחסרונות של הפשטת בסיס נתונים (Abstraction)

6.4.3 אילו מאפיינים מופשטים?

6.4.4 חיבורי בסיס נתונים

6.4.5 ביצוע שאילתות

6.4.6 שליפת תוצאות

6.4.7 סדרות (Sequences)

6.4.8 מאפייני ניידות (Portability)

6.4.9 שגיאות מופשטות

6.4.10 מתודות נוחות

6.5 סיכום

7. טיפול בשגיאות

7.1 מבוא

7.2 סוגי שגיאות

7.2.1 שגיאות תכנות

7.2.2 סמלים לא מוגדרים (Undefined Symbols)

7.2.3 שגיאות ניידות

7.2.4 שגיאות זמן ריצה (Runtime)

7.2.5 שגיאות PHP

7.3 שגיאות PEAR

7.3.1 המחלקה PEAR_Error

7.3.2 טיפול בשגיאות PEAR

7.3.3 מצבי שגיאה של PEAR

7.3.4 טיפול אדיב בשגיאות (Graceful Handling)

7.4 חריגות (Exceptions)

7.4.1 מהן חריגות?

throw-try, catch 7.4.2

7.5 סיכום

הקדמה

במהלך השנים האחרונות, PHP צמחה והפכה לפלטפורמת הרשת הנפוצה ביותר בעולם, כשהיא פעילה ביותר משליש משרתי האינטרנט ברחבי הגלובוס. הצמיחה של PHP אינה רק כמותית אלא גם איכותית. יותר ויותר חברות, כולל חברות מרשימת ה-Fortune, מסתמכות על PHP כדי להריץ את האפליקציות הקריטיות לעסקיהן, דבר היוצר מקומות עבודה חדשים ומגדיל את הדרישה למפתחי PHP. גרסה 5, הצפויה להשתחרר בעתיד הקרוב מאוד, נושאת הבטחה גדולה אף יותר.

בעוד שהמורכבות של תחילת העבודה עם PHP נותרה ללא שינוי ונמוכה מאוד, המאפיינים שמציעה PHP כיום מאפשרים למפתחים להגיע הרבה מעבר לאפליקציות HTML פשוטות. מודל העצמים המעודכן מאפשר כתיבת פרויקטים בקנה מידה גדול בצורה יעילה, תוך שימוש במתודולוגיות סטנדרטיות של תכנות מונחה עצמים. תמיכת ה-XML החדשה הופכת את PHP לשפה הטובה ביותר הקיימת לעיבוד XML, ויחד עם התמיכה החדשה ב-SOAP, היא מהווה פלטפורמה אידיאלית ליצירה ושימוש בשירותי רשת (Web Services).

ספר זה, שנכתב על ידי עמיתי, אנדי גוטמנס, ושני מפתחי PHP בולטים מאוד, סטיג באקן ודריק רתאנס, מחזיק במפתח לפתיחת העושר של PHP 5. הוא מכסה ביסודיות את כל המאפיינים של הגרסה החדשה, והוא פריט חובה לכל מפתחי PHP המעוניינים לחקור את היכולות המתקדמות של PHP 5.

זאב סורסקי

פתח דבר

"הביטחון הטוב ביותר נגד מהפכה טמון בתיקון מתמיד של עוולות ובהכנסת שיפורים נחוצים. ההזנחה של תיקון בזמן היא שהופכת בנייה מחדש לנחוצה." — ריצ'רד וייטלי

בראשית

זה היה לפני שמונה שנים, כשראסמוס לרדורף החל לראשונה לפתח את PHP/FI. הוא לא יכול היה לדמיין שיצירתו תוביל בסופו של דבר לפיתוח של PHP כפי שאנו מכירים אותה כיום, הנמצאת בשימוש של מיליוני אנשים. הגרסה הראשונה של "PHP/FI", שנקראה Personal Homepage Tools/Form Interpreter, הייתה אוסף של סקריפטים ב-Perl בשנת 1995. אחד המאפיינים הבסיסיים היה שפה דמוית Perl לטיפול בשליחת טפסים, אך חסרו בה מאפייני שפה נפוצים ושימושיים רבים, כמו לולאות for.

PHP/FI 2

שכתוב הגיע עם PHP/FI 2 בשנת 1997, אך באותה עת הפיתוח נוהל כמעט בלעדית על ידי ראסמוס. לאחר שחרורה בנובמבר של אותה שנה, אנדי גוטמנס וזאב סורסקי נתקלו ב-PHP/FI בזמן שחיפשו שפה לפיתוח פתרון מסחר אלקטרוני כפרויקט אוניברסיטאי. הם גילו ש-PHP/FI לא הייתה חזקה כפי שנראתה, ושפתה חסרה מאפיינים נפוצים רבים. אחד ההיבטים המעניינים ביותר כלל את הדרך שבה מומשו לולאות while. הסורק הלקסיקלי (lexical scanner) שנבנה בעבודת יד היה עובר על הסקריפט, וכשהיה נתקל במילת המפתח while, היה זוכר את מיקומה בקובץ. בסוף הלולאה, מצביע הקובץ היה חוזר למיקום שנשמר, וכל הלולאה נקראה ובוצעה מחדש.

PHP 3

זאב ואנדי החליטו לשכתב לחלוטין את שפת הסקריפטים. לאחר מכן הם חברו לראסמוס כדי לשחרר את PHP 3, ואיתה הגיע גם שם חדש: PHP: Hypertext Preprocessor, כדי להדגיש ש-PHP היא מוצר שונה ולא מתאימה רק לשימוש אישי. זאב ואנדי גם תכננו ומימשו API חדש להרחבות. API זה אפשר לתמוך בקלות בהרחבות נוספות לביצוע משימות כגון גישה לבסיסי נתונים, בודקי איות וטכנולוגיות אחרות, דבר שמשך מפתחים רבים שלא היו חלק מקבוצת ה"ליבה" להצטרף ולתרום לפרויקט PHP. בזמן שחרור PHP 3 ביוני 1998, בסיס ההתקנות המוערך של PHP עמד על כ-50,000 דומיינים. PHP 3 הציתה את תחילת הפריצה האמיתית של PHP, והייתה הגרסה הראשונה שהגיעה לבסיס התקנות של למעלה ממיליון דומיינים.

PHP 4

בשלהי 1998, זאב ואנדי הביטו לאחור על עבודתם ב-PHP 3 והרגישו שהם יכלו לכתוב את שפת הסקריפטים אפילו טוב יותר, ולכן החלו בשכתוב נוסף. בעוד ש-PHP 3 עדיין פירשה (parsed) את הסקריפטים ברצף תוך כדי ביצועם, PHP 4 הגיעה עם פרדיגמה חדשה של "קודם הידור, אחר כך ביצוע". שלב ההידור (compilation) אינו מהדר סקריפטים של PHP לקוד מכונה; במקום זאת, הוא מהדר אותם ל"קוד בייט" (byte code), שמבוצע לאחר מכן על ידי מנוע ה-Zend (Zend הוא ראשי תיבות של Zeev & Andi), הלב החדש של PHP 4. בשל דרך ביצוע חדשה זו של סקריפטים, הביצועים של PHP 4 היו טובים בהרבה מאלו של PHP 3, עם פגיעה מזערית בלבד בתאימות לאחור. בין השיפורים האחרים היה API משופר להרחבות לביצועי זמן-ריצה טובים יותר, שכבת הפשטה לשרת אינטרנט המאפשרת ל-PHP 4 לרוץ על רוב שרתי האינטרנט הפופולריים, ועוד הרבה יותר. PHP 4 שוחררה רשמית ב-22 במאי 2000, וכיום בסיס ההתקנות שלה עבר את רף ה-15 מיליון דומיינים.

ב-PHP 3, מספר הגרסה המשני (הספרה האמצעית) מעולם לא היה בשימוש, וכל הגרסאות מוספרו כ-3.0.x. זה השתנה ב-PHP 4, ומספר הגרסה המשני שימש לציון שינויים חשובים בשפה. השינוי החשוב הראשון הגיע ב-PHP 4.1.0, שהציגה את ה"סופר-גלובליים" (superglobals) כגון `$_GET` ו-`$_POST`. ניתן לגשת לסופר-גלובליים מתוך פונקציות מבלי להשתמש במילת המפתח `global`. תכונה זו נוספה כדי לאפשר את כיבוי אפשרות ה-INI שנקראת `register_globals`.

`register_globals` היא תכונה ב-PHP הממירה אוטומטית משתני קלט כמו "foo=bar" בכתובת <http://php.net/?foo=bar> למשתנה PHP הנקרא `$foo`. מכיוון שאנשים רבים אינם בודקים משתני קלט כראוי, לאפליקציות פותחו חורי אבטחה, מה שהפך את עקיפת קוד האבטחה והאימות לקלה למדי.

עם הסופר-גלובליים החדשים במקומם, ב-22 באפריל 2002, שוחררה גרסה 4.2.0 של PHP כש-`register_globals` כבוי כברירת מחדל. PHP 4.3.0, גרסת ה-PHP 4 המשמעותית האחרונה, שוחררה ב-27 בדצמבר 2002. גרסה זו הציגה את ממשק שורת הפקודה (CLI), שכבת קלט/פלט (I/O) מחודשת של קבצים ורשת (הנקראת streams), וספריית GD מובנית. למרות שלרוב התוספות הללו אין השפעה ממשית על משתמשי הקצה, מספר הגרסה הראשי הועלה בשל השינויים הגדולים בליבת ה-PHP.

PHP 5

זמן קצר לאחר מכן, הביקוש לתכונות מונחות-עצמים נפוצות יותר גדל מאוד, ואנדי הגה את הרעיון לשכתב את החלק מונחה-העצמים של מנוע ה-Zend. זאב ואנדי כתבו את המסמך "Zend Engine II: סקירת תכונות ועיצוב" והציתו דיונים סוערים על עתידה של PHP. למרות ששפת הבסיס נשארה זהה, תכונות רבות נוספו, הושמטו ושונה עד ש-PHP 5 הבשילה. לדוגמה, מרחבי שמות (namespaces) והורשה מרובה, שהוזכרו במסמך המקורי, מעולם לא נכנסו ל-PHP 5. הורשה מרובה הושמטה לטובת ממשקים (interfaces), ומרחבי שמות הושמטו לחלוטין. תוכלו למצוא רשימה מלאה של תכונות חדשות בפרק "מה חדש ב-PHP 5".

PHP 5 צפויה לשמור ואף להגדיל את ההובלה של PHP בשוק פיתוח הרשת. היא לא רק מחוללת מהפכה בתמיכה מונחית-העצמים של PHP, אלא היא מכילה גם תכונות חדשות רבות שהופכות אותה לפלטפורמת פיתוח הרשת האולטימטיבית. פונקציונליות ה-XML המשותפת ב-PHP 5 מעמידה אותה בשורה אחת עם טכנולוגיות רשת אחרות בתחומים מסוימים ועוקפת אותן באחרים, במיוחד הודות להרחבת ה-SimpleXML החדשה שהופכת את המניפולציה של מסמכי XML לקלה באופן מגוון. בנוסף, ה-SOAP החדש, MySQLi ומגוון הרחבות אחרות הם אבני דרך משמעותיות בתמיכה של PHP בטכנולוגיות נוספות.

קהל היעד

ספר זה הוא מבוא לתכונות המתקדמות החדשות ב-PHP 5. הוא נכתב עבור מתכנתי PHP שעושים את המעבר ל-PHP 5. למרות שפרק 2, "שפת הליבה של PHP 5", מכיל מבוא לתחביר PHP 5, הוא מיועד כרענון למתכנתי PHP ולא כמדריך למתכנתים חדשים. עם זאת, מפתחי רשת עם ניסיון בתכנות בשפות אחרות ברמה גבוהה עשויים אכן למצוא שמדריך זה הוא כל מה שהם צריכים כדי להתחיל לעבוד ביעילות עם PHP 5.

סקירת פרקים

פרק 1, "מה חדש ב-PHP 5?": דן בתכונות החדשות ב-PHP 5. רוב התכונות הללו עוסקות ביכולות מונחות-עצמים חדשות, כולל דוגמאות קטנות לכל תכונה. הוא גם נותן סקירה של ההרחבות החדשות ב-PHP 5. רוב הנושאים המוזכרים בפרק זה מוסברים בפירוט רב יותר בפרקים מאוחרים יותר.

פרק 2, "שפת הליבה של PHP 5": מציג את תחביר ה-PHP לקוראים שאינם מכירים את PHP. כל מבני השפה הבסיסיים וסוגי המשתנים מוסברים יחד עם דוגמאות פשוטות כדי לתת לקורא את אבני הבניין הדרושות לבניית סקריפטים אמיתיים.

פרק 3, "תכנות מונחה עצמים ב-PHP 5": ממשיך לחקור את התחביר של PHP 5, תוך התמקדות בפונקציונליות מונחית-העצמים שלו. פרק זה מכסה יסודות, כגון מאפיינים ומתודות, ומתקדם לנושאים מסובכים יותר, כגון פולימורפיזם, ממשקים, חריגות (exceptions) ועוד הרבה יותר.

פרק 4, "תכנות מונחה עצמים מתקדם ותבניות עיצוב ב-PHP 5": משתמש בפרק הקודם כבסיס ומכסה כמה מהתכונות המתקדמות ביותר של מודל העצמים של PHP 5. לאחר לימוד תכונות אלו, כולל ארבע תבניות עיצוב נפוצות ויכולות הרפלקציה של PHP, תהפכו במהרה לאשפי OO.

פרק 5, "כיצד לכתוב אפליקציית אינטרנט עם PHP 5": כעת כשאתם מכירים את התחביר ותכונות השפה, פרק זה מכניס אתכם לעולם של כתיבת אפליקציות רשת. המחברים מראים לכם יסודות, כמו טיפול בקלט דרך משתני טפסים וטכניקות בטיחות, אך הפרק כולל גם נושאים מתקדמים יותר, כגון ניהול סשנים (sessions) עם עוגיות והרחבת ה-session של PHP. תמצאו גם כמה טיפים על פריסת קוד המקור עבור אפליקציות הרשת שלכם.

פרק 6, "בסיסי נתונים עם PHP 5": מציג שימוש ב-MySQL, SQLite ו-Oracle מתוך PHP, אך מתמקד בעיקר בפרטים הספציפיים ל-PHP 5 של גישה לבסיסי נתונים. עבור כל בסיס נתונים, תלמדו על כמה מנקודות החוזק והחולשה שלו, כמו גם על סוגי האפליקציות שבהן כל אחד מצטיין. וכמובן, תלמדו כיצד להתממשק איתם באמצעות הפונקציות המובנות של PHP או באמצעות PEAR DB.

פרק 7, "טיפול בשגיאות": כל הסקריפטים יכולים להעלות שגיאות, אך כמובן שאינכם רוצים שהן יופיעו באתר שלכם ברגע שהאפליקציה עברה את שלב הפיתוח. פרק זה עוסק בסוגים שונים של שגיאות קיימות, כיצד לטפל בשגיאות אלו עם PHP וכיצד לטפל בשגיאות עם PEAR.

פרק 8, "XML עם PHP 5": מכיוון שאחת התכונות החדשות והחשובות ב-PHP 5 היא התמיכה המחדשת ב-XML, לא ניתן היה לוותר על פרק זה. הוא עוסק באסטרטגיות שונות של ניתוח (parsing) והמרת XML לפורמטים אחרים עם XSLT. מוצגים XML-RPC ו-SOAP כדי להראות לכם כיצד להטמיע שירותי רשת בשתי הטכניקות.

פרק 9, "הרחבות נפוצות": למרות שאינן ספציפיות ל-PHP 5, חמש ההרחבות המרכזיות שפרק זה מכסה הן חשובות מספיק כדי לזכות במקום בספר זה. החלק הראשון, "קבצים וזרמים" (Files and Streams), מסביר על טיפול בקבצים וזרמי רשת. זרם (stream) אינו אלא דרך לגשת לנתונים חיצוניים. החלק השני, "ביטויים רגולריים", מסביר את התחביר של מנוע הביטויים הרגולריים (PCRE). ב"טיפול בתאריכים", אנו מסבירים את הפונקציות השונות המשמשות לניתוח ועיצוב מחרוזות תאריך ושעה. ב"מניפולציית גרפיקה עם GD", אנו מראים לכם דרך שני תרחישים מהחיים האמיתיים את הפונקציות הבסיסיות ליצירה ועריכת גרפיקה. החלק האחרון, "מחרוזות רב-בייטיות ומערכי תווים", מסביר את מערכי התווים השונים והפונקציות לטיפול בהם.

פרק 10, "שימוש ב-PEAR": מציג את PEAR, מאגר ההרחבות והאפליקציות של PHP. הפרק מראה כיצד להשתמש ב-PEAR ולתחזק את החבילות המותקנות מקומית.

פרק 11, "חבילות PEAR חשובות": נותן סקירה של חבילות ה-PEAR החשובות ביותר, יחד עם דוגמאות. החבילות המכוסות כוללות מערכות תבניות (Template Systems), חבילת Auth לביצוע אימות, טיפול בטפסים וחבילה לפישוט זיכרון מטמון (caching).

פרק 12, "בניית רכיבי PEAR": מסביר כיצד ליצור חבילת PEAR משלכם. תקן הקידוד של PEAR ופורמט הגדרת החבילה package.xml יובילו אתכם בדרך להשלמת חבילת ה-PEAR הראשונה שלכם.

פרק 13, "ביצוע המעבר": עוסק בכמה שינויים שאינם תואמים לאחור שהוצגו בין PHP 4 ל-PHP 5. פרק זה אומר לכם באילו דברים עליכם לטפל כשאתם גורמים לאפליקציה שלכם לעבוד על PHP 5.

פרק 14, "ביצועים": מראה לכם כיצד לגרום לסקריפטים שלכם לעבוד טוב יותר. הפרק מציע טיפים על שימוש סטנדרטי ב-PHP, שימוש בכלי עזר חיצוניים (APD ו-Xdebug) למציאת בעיות, ומאצי PHP.

פרק 15, "מבוא לכתיבת הרחבות ל-PHP": מסביר כיצד לכתוב הרחבת PHP מותאמת אישית משלכם. אנו משתמשים בדוגמה פשוטה כדי להסביר את הדברים החשובים ביותר כמו ניתוח פרמטרים וניהול משאבים.

פרק 16, "כתיבת סקריפטים למעטפת (Shell Scripting)": מראה לכם כיצד לכתוב סקריפטים למעטפת ב-PHP, מכיוון ש-PHP שימושית ליותר מאשר רק אפליקציות רשת. אנו מסבירים את ההבדלים בין קבצי ההרצה של CGI ו-CLI.

הערה על סגנונות קידוד

ישנם כמעט כמספר סגנונות הקידוד כמספר המתכנתים. דוגמאות ה-PHP בספר זה עוקבות אחר תקן הקידוד של PEAR, כאשר הסוגריים המסולסלים הפותחים נמצאים בשורה שמתחת לשם הפונקציה.

אודות התוכנה

מצורף בגב הספר קישור מיוחד <https://www.google.com/search?q=%D7%9C-Zend.com>, שם תוכלו להוריד גרסת ניסיון מלאה ל-90 יום של Zend Studio IDE. סביבת הפיתוח של Zend (ZDE) היא כלי נוח המשלב עורך, דיבאגר ומנהל פרויקטים כדי לעזור לכם לפתח, לנהל ולנפות באגים בקוד שלכם.

עדכונים, טעויות והורדות

עדכונים, תיקוני טעויות (errata) והעתקים של תוכניות הדוגמה המשמשות בספר זה ניתן למצוא בכתובת:
<http://php5powerprogramming.com>

תודות

ספר זה לא היה יכול להיכתב ללא המשוב של הסוקרים הטכניים שלנו; לפיכך, ברצוננו להודות למרכוס בורגר, סטף פוקס, מרטין יאנסן ורוב ריצ'רדס על הערותיהם והמשוב המצוין שלהם. מלבד ארבעת הסוקרים הללו, ישנם עוד כמה אנשים שעזרו להשיב על מספר שאלות במהלך כתיבת הספר, ובאופן ספציפי יותר: כריסטיאן סטוקר על העזרה בפרק ה-XML, ווז פורלונג ושרה גולמן על המענה לשאלות בנושא שכבת הזרמים (streams layer), פייר-אלן ג'וי על שסיפק תובנות לגבי העבודה הפנימית של ספריית ה-GD, ובאופן כללי יותר קהילת PEAR על תמיכתם ומסירותם למאגר נהדר של רכיבי PEAR שימושיים. חלק מהסעיפים בספר זה נכתבו על ידי מחברים-שותפים; גאורג ריכטר תרם את סעיף ה-MySQLi בפרק בסיסי הנתונים, וזאב סורסקי הוסיף את הסעיף על חבילת הביצועים של Zend (Zend's Performance Suite).

ברצוננו להודות גם למארק ל. טאוב ולצוות המערכת של Pearson PTR על הדברים שהם טובים בהם: ארגון, תכנון ושיווק של ספר זה, ודאגה לכך שהכל ישתלב יחד. תודה לג'אנט ואלאד על תמיכת עריכת הפיתוח המועילה, ולעורכת הפרויקט שלנו קריסטי הארט, שעזרה לנו לסיים את הספר תחת לחץ והוסיפה לו את הנגיעות האחרונות.

תהנו!

אנדי, סטיג ודריק

פרק 1

מה חדש ב-PHP 5?

"הדרך הטובה ביותר להיות מוכן לעתיד היא להמציא אותו." — ג'ון סקאלי

1.1 מבוא

רק הזמן יגיד אם השחרור של PHP 5 יהיה מוצלח כמו שני קודמיו (PHP 3 ו-PHP 4). התכונות והשינויים החדשים שואפים להיפטר מכל חולשה שהייתה ל-PHP ולהבטיח שהיא תישאר בראש כשפת תכנות הרשת הטובה בעולם.

ספר זה מפרט על PHP 5 ועל תכונותיה החדשות. עם זאת, אם אתם מכירים את PHP 4 ומשתוקקים לדעת מה חדש ב-PHP 5, הפרק הזה הוא בשבילכם. בסיום קריאת פרק זה, תלמדו על:

- תכונות השפה החדשות
- חדשות בנוגע להרחבות PHP
- שינויים ראויים לציון אחרים בגרסה האחרונה של PHP

1.2 תכונות שפה

1.2.1 מודל מונחה-עצמים חדש

כשזאב סורסקי הוסיף את התחביר מונחה-העצמים בימי PHP 3, הוא נוסף כ"סוכר תחבירי לגישה לאוספים". מודל ה-(OO) Object Oriented תמך גם בהורשה ואפשר למחלקה (ולאובייקט) לאגד מתודות ומאפיינים, אך לא הרבה מעבר לכך. כשזאב ואנדי גוטמנס שכתבו את מנוע הסקריפטים עבור PHP 4, זה היה מנוע

חדש לחלוטין; הוא רץ הרבה יותר מהר, היה יציב יותר והציע יותר תכונות. עם זאת, במודל ה-OO שהוצג לראשונה ב-PHP 3 כמעט ולא נגעו.

למרות שלמודל האובייקטים היו מגבלות רציניות, נעשה בו שימוש נרחב ברחבי העולם, לעיתים קרובות באפליקציות PHP גדולות. השימוש המרשים הזה בפרדיגמת ה-OOP עם PHP 4, למרות חולשותיה, הוביל לכך שהיא הפכה למוקד העיקרי של שחרור PHP 5.

אז מה היו חלק מהמגבלות ב-PHP 3 ו-4? המגבלה הגדולה ביותר (שהובילה למגבלות נוספות) הייתה העובדה שסמנטיקת ההעתקה של אובייקטים הייתה זהה לזו של סוגים פרימיטיביים (native types). כיצד זה השפיע על מפתח ה-PHP? בעת השמת משתנה (המצביע על אובייקט) למשתנה אחר, נוצר עותק של האובייקט. לא רק שזה השפיע על הביצועים, אלא שזה בדרך כלל הוביל להתנהגות מעורפלת ובאגים באפליקציות PHP 4, מכיוון שמפתחים רבים חשבו ששני המשתנים יצביעו על אותו אובייקט, מה שלא היה המקרה. המשתנים הצביעו במקום זאת על עותקים נפרדים של אותו אובייקט. שינוי של אחד לא שינה את השני.

לדוגמה, הקוד הבא ב-PHP 4 היה מדפיס "Andi" במקום "Stig", מכיוון שהאובייקט \$person מועבר לפונקציה changeName() לפי ערך (by-value), ולכן הפונקציה עובדת על עותק.

ב-PHP 5, התשתית של מודל האובייקטים שוכתבה לעבודה עם **ידיית אובייקטים** (object handles). אלא אם תשכילו אובייקט במפורש באמצעות מילת המפתח clone, לעולם לא תיצרו עותקים "מאחורי הקלעים".

1.2.2 תכונות מונחות-עצמים חדשות

התכונות החדשות רבות מכדי לפרטן כאן (ראו פרק 3), אך הנה העיקריות שבהן:

- **שינויי גישה (Access Modifiers):** הוספת public, private ו-protected לשליטה על הגישה למתודות ומאפיינים.
- **שם בנאי אחיד:** שימוש ב-construct__() במקום שם המחלקה.
- **מפרק (Destructor):** תמיכה במתודה destruct__() שרצה בעת השמדת האובייקט.
- **ממשקים (Interfaces):** היכולת של מחלקה לממש חוזים מרובים.
- **אופרטור instanceof:** תמיכה ברמת השפה לבדיקת סוג האובייקט.
- **מתודות ומחלקות final:** מניעת דריסה של מתודות או הורשה של מחלקות.
- **שכפול אובייקטים מפורש:** שימוש במילת המפתח clone.
- **קבועי מחלקה:** הגדרת קבועים בתוך מחלקה באמצעות const.
- **מתודות ומאפיינים סטטיים:** גישה דרך המחלקה עצמה ללא צורך ביצירת מופע (instance), שימושי מאוד בתבניות כמו Singleton.
- **מחלקות ומתודות מופשטות (Abstract):** מחלקות שלא ניתן ליצור מהן מופע ומשמשות כבסיס להורשה בלבד.
- **רמזי סוג (Class Type Hints):** אפשרות להגדיר שפרמטר של פונקציה חייב להיות מסוג מחלקה מסוימת.
- **Dereferencing של אובייקטים:** היכולת לקרוא למתודה ישירות מאובייקט המוחזר ממתודה אחרת: obj->method()->method2\$().
- **איטרטורים (Iterators):** PHP 5 מאפשרת הן למחלקות PHP והן למחלקות של הרחבות PHP לממש ממשק Iterator. לאחר מימוש ממשק זה, ניתן לעבור על מופעים של המחלקה באמצעות מבנה השפה foreach():
- PHP

```
;$obj = new MyIteratorImplementation()  
} (foreach ($obj as $value
```

```
;"print "$value
{
```

- לדוגמה מלאה יותר, ראו פרק 4, "תכנות מונחה עצמים מתקדם ותבניות עיצוב ב-PHP 5".
- `:autoload__()` מפתחים רבים הכותבים אפליקציות מונחות-עצמים יוצרים קובץ מקור PHP אחד לכל הגדרת מחלקה. אחת המטרות המציקות ביותר היא הצורך לכתוב רשימה ארוכה של הכללות (inclusions) בתחילת כל סקריפט. ב-PHP 5 זה כבר לא נחוץ. ניתן להגדיר פונקציית `:autoload__()` שנקראת אוטומטית במקרה שבו מנסים להשתמש במחלקה שטרם הוגדרה, מה שמאפשר להטעין את הקובץ המתאים רגע לפני שהמנוע קורס עם שגיאה.

1.2.3 תכנות שפה חדשות נוספות

- טיפול בחריגות (Exception Handling): PHP 5 מוסיפה את היכולת להשתמש בפרדיגמה המוכרת של `try/throw/catch`. ניתן לזרוק רק אובייקטים היורשים ממחלקת ה-`Exception`.
- `foreach` עם הפניות (References): ב-PHP 4 לא ניתן היה לשנות ערכי מערך תוך כדי איטרציה עליו. PHP 5 תומכת בכך על ידי הוספת סימן ה-`&` (הפניה) בלולאת ה-`foreach`, כך שכל שינוי בערך משפיע ישירות על המערך המקורי.
- ערכי ברירת מחדל לפרמטרים המועברים בהפניה: כעת ניתן להגדיר ערכי ברירת מחדל גם לפרמטרים שמוגדרים עם `&`.

1.3 שינויים כלליים ב-PHP

1.3.1 XML ושירותי רשת (Web Services)

עדכוני ה-XML ב-PHP 5 הם אולי המשמעותיים והמרגשים ביותר.

- **התשתית:** ב-PHP 4 התמיכה ב-XML התבססה על ספריות שונות (`Expat`, `Sablotron`, `libxml2`), מה שהוביל לתחזוקה לקויה וחוסר תאימות. ב-PHP 5, כל הרחבות ה-XML שוכתבו לשימוש בערכת הכלים המעולה `libxml2`, המציעה מימוש יעיל ועשיר בתכונות לתקני ה-XML.
- **SAX ו-DOM:** מימוש ה-DOM עבר "מתיחת פנים" יסודית והוא כעת תואם לתקן W3C (למשל, שמות פונקציות בשיטת `studlyCaps`).
- **SimpleXML:** הרחבה זו מחוללת מהפכה בעבודה עם XML ב-PHP על ידי ייצוג קובץ ה-XML כאובייקט PHP טבעי. הגישה לאלמנטים ותכונות הופכת לפשוטה ביותר.
- **SOAP:** ב-PHP 5 נוספה תמיכה רשמית ב-SOAP כהרחבת C מובנית, מה שמשפר משמעותית את הביצועים לעומת מימושי PHP קודמים (כמו ב-PEAR).

1.3.1.7 הרחבת MySQLi החדשה

עבור PHP 5, חברת MySQL AB כתבה הרחבה חדשה המאפשרת לנצל את כל היכולות של MySQL 4.1 ומעלה, כולל ממשק מונחה-עצמים, הצהרות מוכנות (`prepared statements`), חיבורי SSL ועוד.

1.3.1.8 הרחבת SQLite

ספריית SQL מובנית שאינה דורשת שרת, מתאימה לאפליקציות קטנות או לסביבות ללא שרת SQL. ב-PHP 5 היא מציעה ממשק OO ותמיכה באיטרטורים.

1.4 מאפיינים חדשים נוספים ב-PHP 5

- **מנהל זיכרון חדש:** מנוע ה-Zend כולל מנהל זיכרון חדש התומך טוב יותר בסביבות מרובות-תהליכים (multi-threaded) ומשחרר זיכרון ביעילות רבה יותר בסיום כל בקשה.
- **הפסקת תמיכה ב-Windows 95:** עקב חוסר בתמיכה טכנית של מיקרוסופט ומגבלות פונקציונליות, PHP 5 אינה תומכת עוד במערכת הפעלה זו.

1.5 סיכום

כמות השיפורים ב-PHP 5 מרשימה ביותר, והפרקים הבאים יכסו אותם לעומק.

פרק 2

שפת הליבה של PHP 5

2.1 מבוא

PHP שואלת חלק מהתחביר שלה משפות כמו C, Shell, Perl ואפילו Java. זוהי שפה היברידית הלוקחת את התכונות הטובות ביותר ליצירת שפת סקריפטים עוצמתית וקלה לשימוש. בפרק זה תלמדו על מבנה השפה, הטמעה ב-HTML, הערות, ניהול משתנים, בסיסי נתונים, מבני בקרה ופונקציות.

אם אתם מפתחי PHP 4 מנוסים, ייתכן שתצטרכו לדלג ישירות לפרק הבא העוסק בשינויים המשמעותיים במודל מונחה-העצמים.

2.2 הטמעה בתוך HTML

הדבר הראשון שעליכם ללמוד על PHP הוא כיצד היא מוטמעת בתוך HTML:

```
HTML

<html>
<head>
<title>סקריפט PHP לדוגמה</title>
</head>
<body>
הקוד הבא מדפיס "Hello, World":
php?>
    print "Hello, World";
<?
</body>
</html>
```

בדוגמה זו, ניתן לראות שקוד ה-PHP שלכם מוטמע בתוך ה-HTML. בכל פעם שמפרש ה-PHP מגיע לתג פתיחה של (<?php), הוא מתחיל לבצע את הקוד עד שהוא נתקל בתג הסגירה (<?). לאחר מכן PHP

מחליפה את קוד ה-PHP בפלט שלו (אם קיים), בעוד שכל טקסט שאינו PHP (כגון HTML) מועבר כפי שהוא לדפדפן. לכן, הרצת הסקריפט הנ"ל תוביל לפלט הבא:

HTML

```
<html>
<head>
<title>סקריפט PHP לדוגמה</title>
</head>
<body>
הקוד הבא מדפיס "Hello, World":
Hello, World
</body>
</html>
```

טיפ: ניתן להשתמש גם בתג פתיחה מקוצר `<?>`. עם זאת, שימוש זה אינו מומלץ מכיוון שלא בכל השרתים אפשרות זו מופעלת (ניתן להפעילה דרך הגדרת ה-`short_open_tag` בקובץ `.php.ini`).

2.3 הערות

כתיבת הערות בקוד חשובה מאוד כדי להסביר מה הקוד עושה. PHP תומכת במספר סגנונות של הערות:

- **הערות שורה אחת:** סגנון ++C (`//`) או סגנון Shell (`#`).
- **הערות מרובות שורות:** סגנון C (`/* ... */`).

דוגמה:

PHP

```
php?>
// זוהי הערה בשורה אחת
# גם זו הערה בשורה אחת
/* זוהי הערה
   המשתרעת על פני מספר שורות */
<?>
```

2.4 משתנים

ב-PHP, כל שמות המשתנים מתחילים בסימן דולר (\$) . שם המשתנה (לאחר ה-\$) חייב להתחיל באות או בקו תחתון (_), ויכול להכיל אותיות, מספרים או קווים תחתונים. PHP היא שפה **Case-Sensitive** (רגישה לרישיות), כלומר המשתנה `$foo` ו-`$Foo` הם משתנים שונים.

2.4.2 ניהול משתנים

ישנם שלושה מבני שפה (Language Constructs) בסיסיים לניהול משתנים: `unset()` ו-`isset()`.

2.4.2.1 isset() ו-empty()

הפונקציה `isset()` בודקת אם משתנה הוגדר והוא אינו `NULL`. הפונקציה `empty()` בודקת אם הערך של המשתנה נחשב ל"ריק" (כמו מחרוזת ריקה, `0`, `false`, או `NULL`).

ניתן להשתמש ב-`isset()` גם על איברי מערך או מאפייני אובייקט:

PHP

```
    } elseif (isset($arr["offset"]  
    ...  
    {  
    } elseif (isset($obj->property)  
    ...  
    {
```

שימו לב שבשתי הדוגמאות לא היינו צריכים לבדוק אם `$arr` או `$obj` מוגדרים בעצמם; `isset()` תחזיר `false` אוטומטית אם הם אינם קיימים.

`isset()` היא היחידה שמקבלת מספר בלתי מוגבל של פרמטרים:

```
isset($var1, $var2, $var3, ...);
```

היא תחזיר `true` רק אם כל המשתנים הוגדרו. זה שימושי מאוד לבדיקת קלט שנשלח מהלקוח.

2.4.2.2 unset()

הפונקציה `unset()` "מבטלת" הגדרה של משתנה שהוגדר בעבר, ומשחררת את הזיכרון ששימש אותו (אם אין משתנים אחרים שמצביעים על ערכו). קריאה ל-`isset()` על משתנה שעבר `unset()` תחזיר `false`.

PHP

```
    $name = "John Doe";  
    unset($name);  
    if (isset($name))  
        print '$name is set';  
    // זה לא יודפס  
    {
```

2.4.2.3 empty()

ניתן להשתמש ב-`empty()` כדי לבדוק אם משתנה לא הוצהר או שערכו הוא `false`. מבנה שפה זה משמש בדרך כלל לבדיקה אם משתנה טופס לא נשלח או אינו מכיל נתונים. בעת בדיקת ערך האמת של משתנה,

ערכו מומר תחילה לבוליאני (Boolean) לפי הכללים המפורטים בסעיף הבא, ולאחר מכן הוא נבדק אם הוא אמת או שקר (true/false).

לדוגמה:

```
PHP
    } ((if (empty($name
    ;'print 'Error: Forgot to specify a value for $name
    {
קוד זה ידפיס הודעת שגיאה אם $name אינו מכיל ערך שמתפרש כ-true.
```

2.4.3 סופר-גלובליים (Superglobals)

ככלל, PHP אינה תומכת במשתנים גלובליים (משתנים שניתן לגשת אליהם אוטומטית מכל טווח הכרזה). עם זאת, משתנים פנימיים מיוחדים מסוימים מתנהגים כמו משתנים גלובליים בדומה לשפות אחרות. משתנים אלו נקראים **סופר-גלובליים** והם מוגדרים מראש על ידי PHP לשימושכם. כמה דוגמאות לסופר-גלובליים הן:

- `$_GET`: מערך הכולל את כל משתני ה-GET ש-PHP קיבלה מדפדפן הלקוח.
- `$_POST`: מערך הכולל את כל משתני ה-POST ש-PHP קיבלה מדפדפן הלקוח.
- `$_COOKIE`: מערך הכולל את כל העוגיות (cookies) ש-PHP קיבלה מדפדפן הלקוח.
- `$_ENV`: מערך המכיל את משתני הסביבה.
- `$_SERVER`: מערך המכיל את ערכי המשתנים של שרת האינטרנט.

ניתן לגשת למשתנים אלו בכל מקום בסקריפט – בין אם בתוך פונקציה, מתודה או בטווח הגלובלי – ללא צורך להשתמש במילת המפתח `global`.

2.5 סוגי נתונים בסיסיים

ב-PHP קיימים שמונה סוגי נתונים שונים: חמישה מהם סקלאריים (scalar) ולכל אחד משלושת הנתונים ייחודיות משלו. המשתנים יכולים להכיל ערכים מכל סוגי הנתונים הללו מבלי להצהיר במפורש על סוגם. המשתנה "מתנהג" בהתאם לסוג הנתונים שהוא מכיל.

2.5.1 מספרים שלמים (Integers)

מספרים שלמים הם מספרים שלמים ללא שבר. ברוב המכונות הנפוצות (כמו Intel Pentium), מדובר במספר שלם של 32 סיביות עם טווח שבין -2,147,483,648 ל-2,147,483,647.

ניתן לכתוב מספרים שלמים בשיטה עשרונית, הקסדצימלית (עם הקידומת 0x) או אוקטלית (עם הקידומת 0).

2.5.2 מספרים בנקודה צפה (Floating-Point Numbers)

אלו הם מספרים ממשיים הכוללים נקודה עשרונית. ברוב הפלטפורמות, גודל הטיפוס הוא 8 בתים והטווח הוא בערך בין 2.2×10^{-308} ל- 1.8×10^{308} .

2.5.3 מחרוזות (Strings)

מחרוזות ב-PHP הן רצף של תווים המסתיימים פנימית בתו `null`. בניגוד לשפת C, PHP זוכרת את אורך המחרוזת פנימית, מה שמאפשר טיפול קל בנתונים בינאריים. אורך המחרוזת המקסימלי תלוי בפלטפורמה, אך ניתן לצפות לתמיכה של לפחות 2GB.

ניתן להגדיר מחרוזות בשלוש דרכים:

1. **מירכאות כפולות** (""): תומכות בתווי מילוט (כמו \n לשורה חדשה) ובשילוב משתנים ישירות בתוך המחרוזת (למשל: "The result is \$result").
2. **מירכאות יחידות** ('): פשוטות יותר; אינן מבצעות החלפת משתנים ותומכות רק בתווי מילוט בסיסיים ביותר (כמו \" או '\').
3. **Here-Docs**: מאפשרים להטמיע קטעי טקסט גדולים ללא צורך במילוט מירכאות. הם מתחילים ב-<<< ולאחריו מזהה (identifier) ומסתיימים באותו מזהה בתחילת שורה.

גישה לתווי מחרוזת

ניתן לגשת לתווים בודדים במחרוזת באמצעות התחביר {offset}\$str.

לדוגמה:

PHP

```
;"str" = "A$";  
;"str{2}" = "d$";  
;"str{1}" = "n$";  
;"str" = $str; // שרשור  
print $str; // ידפיס Andi
```

הערה: ב-PHP 4 השתמשו בסוגריים מרובעים [] לגישה למחרוזות. ב-PHP 5 מומלץ להשתמש בסוגריים מסולסלים {} כדי להבדיל בין מחרוזת למערך.

2.5.4 בוליאנים (Booleans)

ערך בוליאני יכול להיות true (אמת) או false (שקר). PHP ממירה סוגי נתונים לבוליאנים באופן אוטומטי בתוך תנאים (כמו if). למשל, המספר 0 יומר ל-false, בעוד שכל מספר אחר יומר בדרך כלל ל-true.

טבלת ערכי אמת (Boolean conversion)

להלן האופן שבו PHP ממירה סוגי נתונים שונים לערך בוליאני:

סוג נתון	ערכי שקר (False)	ערכי אמת (True)
מספר שלם (Integer)	0	כל ערך שאינו אפס
נקודה צפה (Float)	0.0	כל ערך שאינו אפס

מחרוזות (Strings)	מחרוזת ריקה (""), המחרוזת אפס ("0")	כל שאר המחרוזות
Null	תמיד שקר	אף פעם לא אמת
מערך (Array)	אם הוא אינו מכיל איברים	אם הוא מכיל לפחות איבר אחד
אובייקט (Object)	אף פעם לא שקר	תמיד אמת
משאב (Resource)	אף פעם לא שקר	תמיד אמת

2.5.7 מערכים (Arrays)

מערכים ב-PHP הם למעשה מפות מסודרות (Ordered Maps) שמקשרות בין מפתחות (keys) לערכים. המפתח יכול להיות מספר שלם או מחרוזת.

2.5.7.1 יצירת מערכים

ניתן ליצור מערך באמצעות המבנה `array()`. המפתח הוא אופציונלי; אם הוא לא מצוין, PHP מקצה אוטומטית מפתח מספרי הגדול ב-1 מהמפתח הגבוה ביותר שהיה עד כה (מתחיל ב-0).

דוגמאות:

- `array(1, 2, 3)` – זהה ל-`array(0 => 1, 1 => 2, 2 => 3)`.
- `array("name" => "John", "age" => 28)` – מערך אסוציאטיבי עם מפתחות מחרוזת.
- `array(1 => "ONE", "TWO", "THREE")` – יהיה שווה ל-`array(1 => "ONE", 2 => "TWO", 3 => "THREE")`.

2.5.7.2 גישה לאיברי מערך

הגישה נעשית באמצעות התחביר `$arr[key]`. כשמשתמשים במחרוזת קבועה כמפתח, חובה להשתמש במירכאות: `$arr["key"]`.

2.5.7.3 שינוי ויצירת איברים

ניתן להוסיף איברים למערך קיים או ליצור מערך חדש שלב אחר שלב:

PHP

```
$arr2[] = 1$ // מוסיף איבר במפתח הפנוי הבא (0)
$arr2[] = 2$ // מוסיף איבר במפתח 1
```

2.5.7.6 מעבר על מערכים (Traversing) באמצעות foreach

זוהי הדרך האלגנטית ביותר לעבור על מערך. התחביר הוא:

```
foreach($array as [$key =>] [&] $value
```

השימוש ב-& לפני ה-\$value מאפשר לשנות את ערכי המערך המקורי תוך כדי הלולאה (תכונה חדשה ב-PHP 5).

דוגמה לשימוש ב-& לשינוי ערכים:

PHP

```
} (foreach ($people as &$person
    } (if ($person["age"] >= 35
        ;"person["age group"] = "Old$
    } else {
        ;"person["age group"] = "Young$
    }
}
```

בדוגמה זו, המערך המקורי \$people יתעדכן ויתווסף לו המפתח "age group" לכל אדם.

2.5.7.7 שימוש ב-list() ו-each()

שיטה ישנה יותר (פחות מומלצת מ-foreach) למעבר על מערך היא שילוב של each() ו-list().

- **reset()**: מאפס את המצביע הפנימי של המערך לתחילתו.
- **each()**: מחזיר את צמד המפתח/ערך הנוכחי ומקדם את המצביע.
- **list()**: מבנה המאפשר להציב מספר ערכים ממערך למספר משתנים בפקודה אחת:
list(\$name, \$age) = \$person_data

2.5.8 קבועים (Constants)

קבועים הם שמות לערכים פשוטים שלא משתנים במהלך הרצת הסקריפט. בניגוד למשתנים, הם אינם מתחילים בסימן דולר (\$) והם נגישים מכל מקום בתוכנית (Global Scope).

הגדרת קבוע:

```
;(define("CONSTANT_NAME", value
```

- נהוג לכתוב קבועים באותיות גדולות (UPPERCASE).
- קבועים יכולים להכיל רק ערכים סקלאריים (לא מערכים או אובייקטים).

דוגמה:

PHP

```
;(define("MY_OK", 0
;(define("MY_ERROR", 1
```

```
} (if ($error_code == MY_ERROR  
;("print("There was an error\n  
{
```

2.6 אופרטורים (Operators)

PHP מכילה שלושה סוגי אופרטורים: אופרטורים אונאריים (על אופרנד אחד), בינאריים (על שניים) ואופרטור טרנארי אחד.

PHP יכולה לבצע פעולות בינאריות רק על שני אופרנדים מאותו סוג. אם הם מסוגים שונים, PHP מבצעת **המרה אוטומטית** לפי הכללים הבאים:

- **בוליאנים, Null ומשאבים:** מתנהגים כשלמים (False=0, True=1, Null=0).
- **שלם ונקודה צפה:** השלם מומר לנקודה צפה.
- **שלם ומחרוזת:** המחרוזת מומרת למספר. אם המחרוזת מייצגת מספר ממשי, גם השלם יומר לממשי.

2.6.1 אופרטורים בינאריים

1. **אופרטורים חשבוניים:** + (חיבור), - (חיסור), * (כפל), / (חילוק) ו-% (מודולו - שארית החילוק). כולם עובדים על ערכים מספריים.
2. **אופרטור השרשור (.):** מחבר שתי מחרוזות לאחת. אם אחד האופרנדים אינו מחרוזת, הוא יומר לאחת.
דוגמה: 2000. print "The year is " ידפיס "The year is 2000".

2.6.2 אופרטורי השמה (Assignment)

האופרטור הבסיסי הוא =. בנוסף, קיימים אופרטורים משולבים המבצעים פעולה והשמה בצעד אחד:

2\$ counter += זהה ל-counter = \$counter + 2\$.

אופרטורים נפוצים: +=, -=, *=, /=, =.

השמה בהפניה (&): יוצרת כינוי (Alias) למשתנה אחר. שינוי באחד ישפיע על השני.

PHP

```
;"name = "Judy$  
;name_alias =& $name$  
;"name_alias = "Jonathan$  
Jonathan $name // ידפיס Jonathan
```

2.6.3 אופרטורי השוואה

מאפשרים לקבוע את היחס בין שני אופרנדים ומחזירים ערך בוליאני.

- `==` (שווה ל-): בודק שוויון לאחר המרת סוגים במידת הצורך.
- `===` (זהה ל-): בודק שוויון כולל סוג הנתונים. ללא המרה אוטומטית.
- `!=` (לא שווה), `!$=>$` (לא זהה), `>` (קטן מ-), `<` (גדול מ-), `>=`, `<=`.

2.6.4 אופרטורים לוגיים

משמשים לחיבור תנאים: `&&` או `and` (וגם), `||` או `or` (או), `xor` (או מוציא).

הערכה בקיצור דרך (Short-Circuit Evaluation): PHP חכמה מספיק כדי לעצור הערכה אם התוצאה כבר ידועה. למשל בתנאי `1 && 0`, המנוע יראה את ה-0 (שקר) ויעצור מיד כי התוצאה חייבת להיות שקר, מבלי לבדוק את הצד הימני.

2.6.5 אופרטורים ברמת הביט (Bitwise)

פועלים על הייצוג הבינארי של המספרים: `&` (NOT), `^` (XOR), `|` (OR), `&` (AND), `<<` (הזזה ימינה), `>>` (הזזה שמאלה).

2.6.8 אופרטורי קידום והפחתה (Increment/Decrement)

אופרטורים אלו ייחודיים כי הם משנים את ערך המשתנה עצמו.

- `$Pre-increment (++var)`: מקדם ב-1 ואז מחזיר את הערך החדש.
- `$Post-increment (++$var)`: מחזיר את הערך הנוכחי ואז מקדם ב-1.

דוגמה להמחשה:

```
PHP
$num = 5;
echo $num++; // ידפיס 5, אבל הערך של num יהפוך ל-6
echo ++$num; // הערך יהפוך ל-7 וידפיס 7
```

2.6.8.1 קידום מחרוזות

מחרוזות שאינן נומריות (מספריות) מקודמות בדומה לשפת Perl: אם האות האחרונה היא אלפא-נומרית, היא מקודמת ב-1. אם היא הייתה 'z', היא תהפוך ל-'a' והתו שלפניה יקודם גם הוא.

- **הערה:** לא ניתן לבצע פעולת הפחתה (`--`) על מחרוזות שאינן נומריות.

2.6.9 אופרטורי המרה (Cast Operators)

ניתן לכפות המרת סוג נתונים (Casting) בדומה לשפת C. ההמרה משנה את סוג הערך באותו רגע, אך לא את סוג המשתנה המקורי.

- `(int)`, `(integer)` — המרה למספר שלם.
- `(string)` — המרה למחרוזת.
- `(bool)`, `(boolean)` — המרה לבוליאני.
- `(array)` — המרה למערך.

2.6.10 אופרטור השתקה (Silence Operator)

הסימן @ לפני ביטוי משתיק הודעות שגיאה שעלולות להיווצר במהלכו.

2.6.11 האופרטור הטרינארי (Ternary Operator)

אופרטור אלגנטי המבצע בדיקה והחזרת ערך בשורה אחת: `truth_expr ? expr1 : expr2`.

אם התנאי אמת, יוחזר `expr1`, אחרת יוחזר `expr2`.

PHP

```
;$message = isset($a) ? '$a is set' : '$a is not set'
```

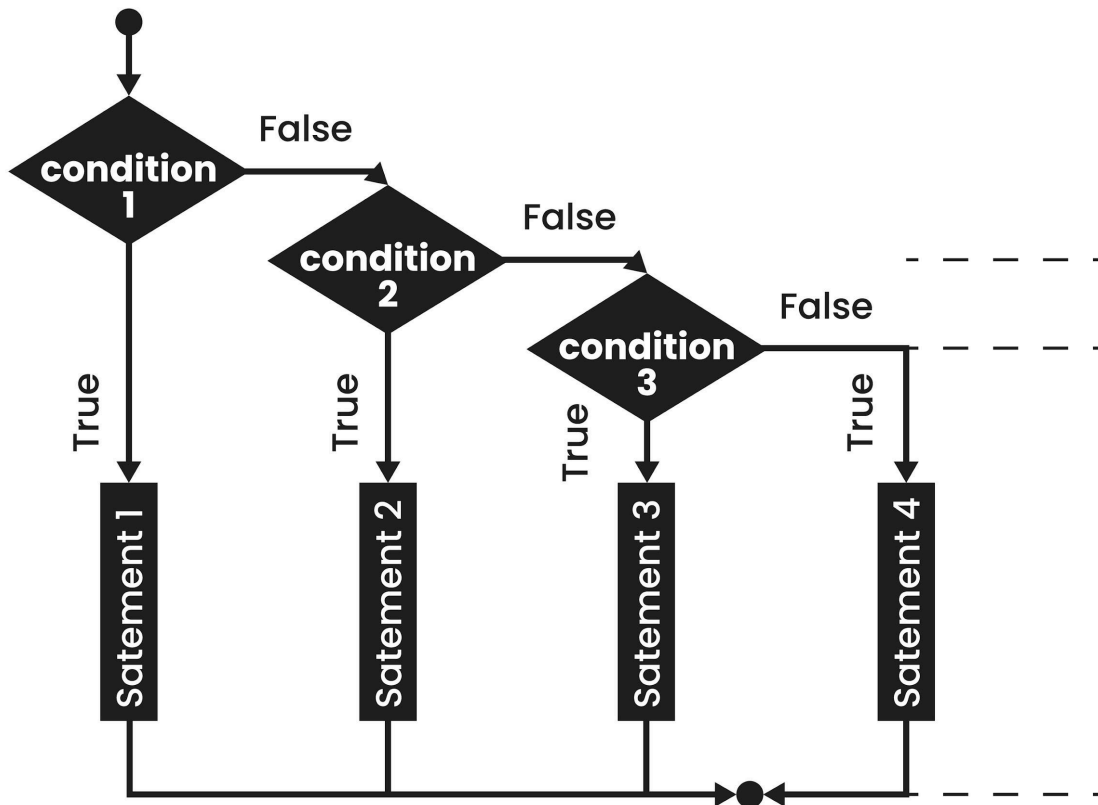
2.7 מבני בקרה (Control Structures)

מבני הבקרה מחולקים לשתי קבוצות: מבני תנאי (משפיעים על כיוון התוכנית) ומבני לולאה (חוזרים על קוד).

2.7.1 מבני תנאי

1. **if Statements**: המבנה הנפוץ ביותר. אם "ביטוי האמת" הוא `true`, הקוד יבוצע. ניתן להוסיף `else` למקרה שהתנאי לא מתקיים, ו-`elseif` לבדיקות נוספות.

IF-ELSE-IF STATEMENT



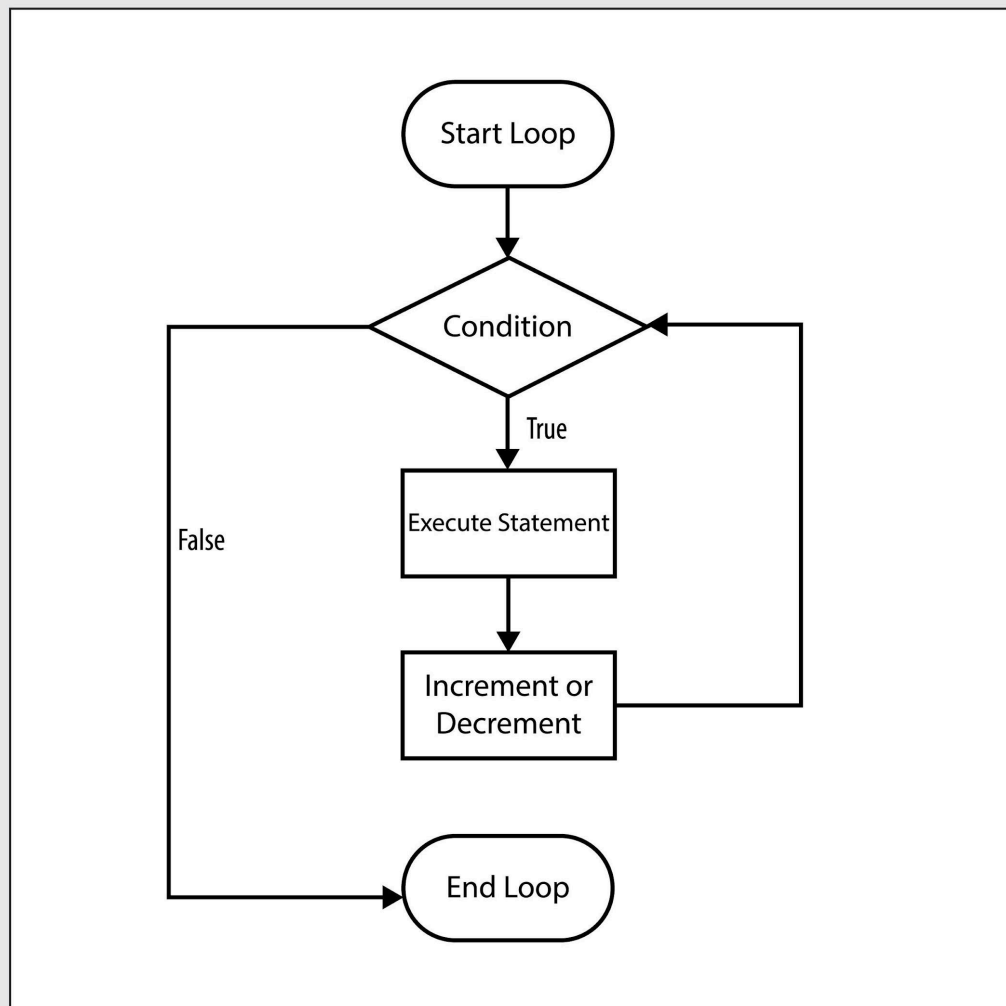
.2

Shutterstock .3

.2 **switch Statements**: מחליף סדרת if/elseif ארוכה בצורה אלגנטית. הוא משווה ביטוי מול מספר מקרים (case). חשוב להשתמש ב-break כדי לעצור את הביצוע לאחר שנמצאה התאמה, אחרת PHP תמשיך לבצע את המקרים הבאים. המקרה default יבוצע אם לא נמצאה אף התאמה.

2.7.2 מבני לולאה

.1 **while loops**: הלולאה הפשוטה ביותר. התנאי נבדק **בתחילת** כל איטרציה. אם הוא אמת, הלולאה ממשיכה.



.2

.3 Shutterstock

.4 Explore

.2 **break ו-continue:**

- **break** עוצר את הלולאה לחלוטין ויוצא ממנה.
- **continue** עוצר את האיטרציה הנוכחית ועובר מיד לבדיקת התנאי של האיטרציה הבאה.
- ניתן להוסיף מספר (למשל **break 2**) כדי לצאת ממספר לולאות מקוננות.
- .3 **do...while Loops:** דומה ל-**while**, אך התנאי נבדק **בסוף** האיטרציה. המשמעות היא שהלולאה תרוץ לפחות פעם אחת תמיד.
- .4 **for Loops:** לולאות בסגנון שפת C המקבלות שלושה ארגומנטים:
 (for (start_expressions; truth_expressions; increment_expressions)
 - **ביטוי התחלה:** מבוצע פעם אחת בלבד בכניסה ללולאה.
 - **ביטוי אמת:** נבדק לפני כל סיבוב.
 - **ביטוי קידום:** מבוצע בסוף כל סיבוב.

דוגמה ללולאת for:

PHP

```
} (++for ($i = 0; $i < 10; $i  
;"print "The square of $i is " . $i*$i . "\n"  
{
```

כמו בשפת C, ניתן לספק יותר מביטוי אחד לכל אחד משלושת הארגומנטים על ידי הפרדתם בפסיקים. הערך של כל ארגומנט יהיה הערך של הביטוי הימני ביותר. ניתן גם להשאיר ארגומנטים ריקים; ארגומנט ריק ייחשב כ-true. למשל, for(;;) יוצר לולאה אינסופית.

טיפ ביצועים: PHP אינה מבצעת אופטימיזציה לערכים קבועים בתוך הלולאה. למשל, בביטוי `count($array); $i <= $i; 0 = i$)for++`, הפונקציה `count` תרוץ בכל סיבוב. עדיף לחשב את האורך מראש במשתנה מחוץ ללולאה.

2.7.3 מבני בקרה להכללת קוד (Inclusion)

מבנים אלו חיוניים לארגון הקוד וליצירת יחידות בנייה שניתן לעשות בהן שימוש חוזר.

2.7.3.1 include, require ודומיהם

פקודת ה-`include` מאפשרת לפצל את קוד המקור למספר קבצים. בעת הביצוע, PHP קוראת את הקובץ, מקמפלת ומריצה אותו בתוך אותו "טווח משתנים" (scope) של הסקריפט הקורא.

- **include:** אם הקובץ לא נמצא, תונפק אזהרה (Warning) אך הסקריפט ימשיך לרוץ.
- **require:** אם הקובץ לא נמצא, תיווצר שגיאה קריטית (Fatal Error) והסקריפט יעצר.
- **include_once / require_once:** הגרסאות המומלצות ביותר. הן מוודאות שהקובץ ייכלל **פעם אחת בלבד**, גם אם נקראו שוב בהמשך. זה מונע הגדרות כפולות של פונקציות ומחלקות.

ניתן להשתמש בנתיבים יחסיים או מוחלטים. מפתחים רבים משתמשים בנתיב מוחלט דרך `SERVER["DOCUMENT_ROOT_"]`.

2.7.3.2 eval()

פונקציה זו דומה ל-`include`, אך במקום לקבל קוד מקובץ, היא מקבלת קוד PHP בתוך **מחרוזת** ומריצה אותו.

אזהרה: לעולם אל תעבירו קלט משתמש ישירות ל-`eval()`, שכן הדבר מהווה פרצת אבטחה חמורה המאפשרת הרצת קוד זדוני.

2.8 פונקציות (Functions)

פונקציה ב-PHP יכולה להיות מובנית (של השפה) או מוגדרת על ידי המשתמש. הקריאה לשתייה זהה: `func(...args)`.

2.8.1 פונקציות מוגדרות משתמש

הגדרה בסיסית:

```
PHP
function function_name($arg1, $arg2) {
    // קוד
    return $value; // החזרת ערך ויציאה מהפונקציה
}
```

2.8.2 טווח משתנים בפונקציה (Function Scope)

לכל פונקציה יש "ארגז חול" משלה. משתנים שהוגדרו מחוץ לפונקציה אינם נגישים בתוכה כברירת מחדל, ומשתנים שנוצרים בתוכה "מתים" כשהיא מסתיימת.

- **גישה למשתנים גלובליים:** ניתן להשתמש במערך המובנה `GLOBALS["var_name$"]` כדי לגשת למשתנה מחוץ לפונקציה.
- **מילת המפתח `global`:** קיימת דרך להצהיר על משתנה כגלובלי בתוך הפונקציה (`global $var`), אך המחברים אינם ממליצים להשתמש בה עקב בעיות תחזוקה והתנהגות לא צפויה עם הפניות.

2.8.3 החזרת ערכים לפי ערך (By Value)

כברירת מחדל, פקודת ה-`return` מחזירה עותק של הערך. אם תשנו את הערך המוחזר מחוץ לפונקציה, המשתנה המקורי שנמצא בתוך הפונקציה (או במערך הגלובלי שהוחזר) לא ישתנה.

2.8.4 החזרת ערכים לפי הפניה (Returning by Reference)

PHP מאפשרת להחזיר משתנים לפי הפניה. המשמעות היא שאינכם מחזירים עותק של הערך, אלא את "כתובת" המשתנה עצמו. הדבר מאפשר לשנות את המשתנה המקורי מתוך הטווח שקרא לפונקציה.

כדי להשתמש בזה, יש להוסיף את הסימן `&` לפני שם הפונקציה בהגדרה, וגם בעת הקריאה לה (בזמן ההשמה למשתנה).

```
PHP
function &get_global_variable($name) {
    return $GLOBALS[$name];
}

$num = 10$
// השמה לפי הפניה
$value =& get_global_variable("num$");
$value = 20$
print $num // ידפיס 20
```

זהירות: השתמשו בטכניקה זו רק כשצריך. שכחה של סימן ה-`&` בזמן ההשמה תגרום ל-PHP להעתיק את הערך במקום להפנות אליו, מה שעלול ליצור באגים קשים לאיתור.

2.8.5 הצהרה על פרמטרים של פונקציה

קיימות שתי דרכים להעביר ארגומנטים (פרמטרים) לפונקציה:

1. **העברה לפי ערך (By-Value):** זוהי הדרך הנפוצה ביותר. הפונקציה מקבלת עותק של הערך, וכל שינוי בו בתוך הפונקציה אינו משפיע על המשתנה המקורי בחוץ.
2. **העברה לפי הפניה (By-Reference):** הפונקציה מקבלת "קיצור דרך" למשתנה המקורי. כל שינוי בפרמטר בתוך הפונקציה ישנה את המשתנה שנשלח אליה בטווח החיצוני.
דוגמה: `function square($n) { $n = $n * $n; }` תגרום למשתנה שישלח אליה להפוך לערך הריבועי שלו.

2.8.5.3 פרמטרים כברירת מחדל (Default Parameters)

בדומה ל-C++, ניתן לקבוע ערך קבוע לפרמטר אם הוא לא נשלח בעת הקריאה לפונקציה. הערך חייב להיות קבוע (מספר, מחרוזת או מערך קבוע).

PHP

```
(function increment(&$num, $increment = 1) {  
    $num += $increment;  
})
```

- **כלל חשוב:** אם החלטתם להשמיט ארגומנט עם ערך ברירת מחדל בעת הקריאה, עליכם להשמיט גם את כל הארגומנטים שאחריו. בהתאם, בהגדרת הפונקציה, כל הפרמטרים שאחרי פרמטר עם ברירת מחדל חייבים להיות גם הם בעלי ברירת מחדל.

2.8.6 משתנים סטטיים (Static Variables)

PHP תומכת במשתנים מקומיים מסוג `static`. אלו משתנים ששומרים על ערכם בין קריאות שונות לפונקציה, אך הם נגישים רק מתוך הפונקציה שבה הם הוגדרו. האתחול שלהם מתבצע רק בפעם הראשונה שבה מגיעים להצהרה עליהם.

זהו כלי מעולה להרצת קוד הגדרות ראשוני (Initialization) שצריך לקרות רק פעם אחת:

PHP

```
function do_something() {  
    static $first_time = true;  
    if ($first_time) {  
        // קוד שירוצ רק בקריאה הראשונה לפונקציה  
        $first_time = false;  
    }  
    // הלוגיקה העיקרית שתרוץ בכל קריאה  
}
```

2.9 סיכום

פרק זה כיסה את תכונות היסוד של PHP: משתנים, מבני בקרה ופונקציות. ידע זה מספק לכם את כל הכלים הדרושים כדי להיות פרודוקטיביים בכתיבת קוד פונקציונלי.

הפרק הבא יעסוק במהפכה של PHP 5: המודל מונחה העצמים (Object-Oriented).

פרק 3

3.1 מבוא

ב-PHP 5, מודל האובייקטים עוצב מחדש לחלוטין. בניגוד לגרסאות קודמות, PHP 5 מציעה סט עשיר של תכונות המאפשרות בניית אפליקציות מורכבות ומסודרות. הפרק יכסה את יסודות המודל, יצירת אובייקטים, בנאים (Constructors), הרשאות גישה (public, private, protected) וירושה.

3.2 אובייקטים (Objects)

ההבדל העיקרי בין תכנות מונחה עצמים (OOP) לתכנות פונקציונלי הוא שנתונים וקוד ארוזים יחד לישות אחת הנקראת אובייקט.

- **מאפיינים (Properties):** המשתנים השייכים לאובייקט (הנתונים שלו).
- **מתודות (Methods):** הפונקציות שהאובייקט תומך בהן.
- **מחלקה (Class):** ה"תבנית" (Template) שמגדירה אילו מתודות ומאפיינים יהיו לאובייקטים מסוג זה.
- **מופע (Instance):** אובייקט ספציפי שנוצר מתוך המחלקה.

דוגמה למחלקה Person:

אם יש לנו שני אנשים באפליקציה, "ג'ודי" ו"ג'ו", ניצור שני מופעים נפרדים של המחלקה Person. לכל אחד מהם יהיה משתנה \$name משלו.

PHP

```
class Person
{
    private $name;

    function setName($name)
    {
        $this->name = $name;
    }

    function getName()
    {
        return $this->name;
    }
}

$judy = new Person();
$judy->setName("Judy");
```



```
;)joe = new Person$  
;"joe->setName("Joe$
```

```
print $judy->getName(); // Judy
```

3.4 מילת המפתח new ובנאים (Constructors)

כדי ליצור אובייקט חדש, משתמשים במילת המפתח new.

מיד לאחר יצירת האובייקט, PHP קוראת אוטומטית למתודה מיוחדת שנקראת הבנאי (construct__). הבנאי משמש בדרך כלל לאתחול מאפיינים.

PHP

```
} class Person  
;private $name  
  
(function __construct($name  
;this->name = $name$  
{  
}  
;"judy = new Person("Judy$
```

3.5 מפרקים (Destructors)

הפוך מהבנאי, המפרק (destruct__) נקרא כאשר האובייקט נהרס (למשל כשאין יותר הפניות אליו או כשהסקריפט מסתיים). זהו מקום טוב לסגור חיבורים למסד נתונים או לכתוב ללוג.

3.6 גישה באמצעות המשתנה this\$

בתוך מתודה של אובייקט, המשתנה המיוחד this\$ מייצג את האובייקט הנוכחי. דרכו ניתן לגשת למאפיינים ולמתודות של אותו מופע (למשל: \$this->name).

3.6.1 הרשאות גישה: Public, Protected, Private

זהו אחד העקרונות החשובים ב-OOP (אנקפסולציה):

1. **public**: ניתן לגשת למאפיין/מתודה מכל מקום (גם מחוץ למחלקה).
2. **protected**: הגישה מותרת רק מתוך המחלקה עצמה או ממחלקות שירשו ממנה.
3. **private**: הגישה מותרת רק מתוך המחלקה שבה הוגדר המאפיין.

3.6.3 מאפיינים סטטיים (Static Properties)

בעוד שמאפיין רגיל שייך למופע ספציפי (לאובייקט), מאפיין סטטי שייך למחלקה עצמה. יש רק עותק אחד שלו לכל האובייקטים מאותו סוג.

ניגשים אליו באמצעות השם של המחלקה והסימן :: בתוך המחלקה, ניתן להשתמש ב-`::self`.

דוגמה לשימוש (מונה מזהה ייחודי):

PHP

```
class MyUniqueIdClass
{
    static $idCounter = 0;
    public $uniqueId;

    function __construct()
    {
        ++self::$idCounter;
        $this->uniqueId = self::$idCounter$;
    }
}
```

בכל פעם שניצור אובייקט חדש, ה-`idCounter` הכללי יעלה ב-1, וכל אובייקט יקבל מספר רץ ייחודי.

3.6.4 מתודות סטטיות (Static Methods)

מתודות סטטיות שייכות למחלקה עצמה ולא למופע (אובייקט) ספציפי.

- איך קוראים להן? באמצעות `ClassName::methodName()`.
- גישה לנתונים: מכיוון שהן לא קשורות לאובייקט, המשתנה `this` לא זמין בתוכן. כדי לגשת למאפיינים או מתודות סטטיות אחרות מאותה מחלקה, משתמשים ב-`::self`.
- שימוש: מתאים לפונקציות עזר (Utility) שאינן דורשות מידע השמור באובייקט ספציפי.

3.7 קבועים במחלקה (Class Constants)

ב-PHP 5 ניתן להגדיר קבועים בתוך מחלקה באמצעות מילת המפתח `const`.

- בדומה למאפיינים סטטיים, הם שייכים למחלקה.
- הם תמיד רגישים לרישיות (Case-sensitive).
- לא ניתן לשנות אותם לאחר ההגדרה. נהוג לכתוב אותם באותיות גדולות (למשל: `const RED = "Red";`).

3.8 שכפול אובייקטים (Cloning)

ב-PHP 5, משתנה אובייקט הוא למעשה "מזהה" (Handle) לאובייקט בזיכרון.

- העתקה רגילה: אם תכתבו `obj2 = $obj1`, שניהם יצביעו על אותו אובייקט. שינוי באחד ישפיע על השני.
- שכפול אמיתי: כדי ליצור עותק נפרד, משתמשים במילת המפתח `clone`.
- מתודת `__clone()`: אם תגדירו מתודה כזו במחלקה, היא תופעל אוטומטית על העותק החדש שנוצר. זה שימושי לביצוע "העתקה עמוקה" (למשל, לפתוח קובץ חדש עבור העותק במקום לחלוק את אותו קובץ עם המקור).

3.9 פולימורפיזם (Polymorphism) וירושה

פולימורפיזם (ריבוי צורות) הוא המושג החשוב ביותר ב-OOP. הוא מאפשר לנו להתייחס לאובייקטים מסוגים שונים דרך ממשק משותף.

- **ירושה (extends):** מחלקה "בן" יורשת את כל המאפיינים והמתודות של מחלקת ה"אב".
- **הקשר:** נוצר קשר של "הוא-סוג-של" (is-a). לדוגמה, Dog הוא סוג של Animal.

במקום לכתוב המון תנאי if כדי לבדוק איזה חיה קיבלנו, אנחנו פשוט קוראים למתודה makeSound(). כל חיה תבצע את הגרסה שלה למתודה (נביחה או יללה), אך הקוד שקורא להן נשאר זהה ופשוט.

3.10 ::self ו-::parent

- **::self** – מתייחס למחלקה הנוכחית (בעיקר עבור סטטיים וקבועים).
- **::parent** – מתייחס למחלקת האב. משמש בעיקר כדי לקרוא לבנאי של האב או למתודות שהאב מגדיר ושדרסנו בבן.

3.11 instanceof האופרטור

משמש לבדיקה האם אובייקט שייך למחלקה מסוימת או יורש ממנה. מחזיר true או false.

3.12 מחלקות ומתודות מופשטות (Abstract)

לפעמים נרצה להגדיר מחלקה כללית מדי מכדי ליצור ממנה אובייקטים (למשל "צורה").

- **מחלקה מופשטת (abstract class):** לא ניתן ליצור ממנה מופע (instance) עם new. היא קיימת רק כדי שאחרים יירשו ממנה.
- **מתודה מופשטת (abstract function):** הגדרה של מתודה ללא קוד (ללא סוגריים מסולסלים). היא "מכריחה" את כל הילדים שיורשים מהמחלקה לממש את המתודה הזו בעצמם.

3.13 ממשקים (Interfaces)

ממשק הוא "חוזה" שהמחלקה מתחייבת לקיים. הוא מכיל רק חתימות של פונקציות (ללא קוד).

- מכיוון ש-PHP לא תומכת בירושה מרובה (מחלקה לא יכולה לרשת משני אבות), ממשקים הם הפתרון: מחלקה יכולה לממש (implements) כמה ממשקים שהיא רוצה.
- **דוגמה:** מחלקת Person יכולה לממש ממשק Loggable, שמחייב אותה להגדיר פונקציה שמחזירה מחרוזת ללוג.

3.15 ו-3.16 final

- **מתודות final:** לא ניתן לדרוס אותה במחלקות יורשות.
- **מחלקת final:** לא ניתן לרשת ממנה בכלל. זהו כלי להגנה על הקוד מפני שינויים לא רצויים בעתיד.

3.17 מתודת __toString()

מתודה מיוחדת שמאפשרת לקבוע מה יודפס כשננסה להדפיס את האובייקט (למשל עם `echo $obj`). היא חייבת להחזיר מחרוזת.

3.18 טיפול בחריגות (Exception Handling)

במקום לבדוק שגיאות עם המון תנאי `if`, משתמשים במבנה `try / catch`:

1. `try`: הקוד שעלול להיכשל.
 2. `throw`: זריקת אובייקט שגיאה (Exception) כשמשו משתבש.
 3. `catch`: תפיסת השגיאה וטיפול בה.
- זה מפריד את הלוגיקה העסקית מהטיפול בשגיאות והופך את הקוד לנקי הרבה יותר.

3.19 הטענה אוטומטית (`autoload`)

כשיש פרויקט עם המון קבצי מחלקות, קשה לנהל את כל ה-`include`. פונקציית `autoload($class_name)` נקראת אוטומטית על ידי PHP ברגע שאתם מנסים להשתמש במחלקה שהיא עוד לא מכירה. בתוכה תוכלו לכתוב קוד שטוען את הקובץ המתאים "בדיוק בזמן".

3.20 רמיזות טיפוס למחלקות בפרמטרים של פונקציות (Class Type Hints)

למרות ש-PHP אינה שפה בעלת "טיפוסיות חזקה" (Strictly Typed), בה נדרש להצהיר על סוג המשתנה בכל פעם, היא מאפשרת לכם לציין במפורש איזו מחלקה אתם מצפים לקבל בפרמטרים של פונקציה או מתודה.

בעבר, כדי להבטיח שפונקציה תקבל אובייקט מסוג מסוים, היה עליכם לכתוב קוד בדיקה ידני:

```
PHP
function onlyWantMyClassObjects($obj)
{
    if (!$obj instanceof MyClass)
    {
        die("ניתן לשלוח לפונקציה זו רק אובייקטים מסוג MyClass");
    }
    // ... המשך הקוד
}
```

כדי לחסוך עבודה זו, PHP מאפשרת להוסיף את שם המחלקה ישירות לפני שם הפרמטר בהגדרת הפונקציה:

```
PHP
function onlyWantMyClassObjects(MyClass $obj)
{
    // ...
}
```

דגשים חשובים:

- **בדיקה אוטומטית:** בעת הקריאה לפונקציה, PHP מבצעת אוטומטית בדיקת `instanceof`. אם האובייקט שנשלח אינו שייך למחלקה המבוקשת (או יורש ממנה), התוכנית תעצור עם הודעת שגיאה.
- **תמיכה בירושה:** ניתן לשלוח כל אובייקט שמקיים קשר של "הוא-סוג-של" (`is-a`) עם המחלקה המוגדרת.
- **יתרון בפיתוח:** תכונה זו עוזרת לוודא שאינכם מעבירים בטעות אובייקטים לפונקציות שלא תוכננו לטפל בהם, מה שמונע באגים קשים לאיתור בזמן ריצה.

3.21 סיכום הפרק

פרק זה סקר את מודל האובייקטים החדש של PHP 5. למדנו על:

- ההבדל בין מחלקות (התבנית) לאובייקטים (המופעים).
- פולימורפיזם וירושה המאפשרים שימוש חוזר בקוד.
- מנגנוני הגנה כמו `private` ו-`protected`.
- טיפול מתקדם בשגיאות באמצעות חריגות (Exceptions).

אם הגעתם משפות כמו Java או C++, ודאי תזהו ש-PHP 5 הפכה לשפה מודרנית ובשלה הרבה יותר. אם כתבתם ב-PHP 4, התכונות הללו הן "אוויר לנשימה" עבור פיתוח פרויקטים גדולים ומורכבים.

פרק 4

4.2 יכולות "העמסה" (Overloading)

ב-PHP, המושג "Overloading" שונה משפות כמו Java או C++. כאן הכוונה היא ליכולת ליירט (Intercept) גישה למאפיינים או למתודות שאינם קיימים במחלקה.

4.2.1 העמסת מאפיינים ומתודות

באמצעות מימוש "מתודות קסם" (Magic Methods), ניתן לשלוט במה שקורה כשמישהו מנסה לגשת למשתנה או פונקציה שלא הוגדרו:

- `get($property__)`: מופעלת כשמנסים לקרוא מאפיין לא קיים.
- `set($property, $value)`: מופעלת כשמנסים לכתוב למאפיין לא קיים.
- `call($method, $args)`: מופעלת כשמנסים לקרוא למתודה לא קיימת.

דוגמה לשימוש (Delegation): ניתן להשתמש ב-`__call` כדי להעביר קריאות לאובייקט אחר באופן אוטומטי (כמו עוזר אישי שמעביר שיחות למנהל).

4.2.2 העמסת גישה כמערך (Array Access)

לפעמים נרצה שאובייקט יתנהג כמו מערך (למשל: `PHP 5` [John]\$userMap). מאפשרת זאת על ידי מימוש הממשק `ArrayAccess`.

זה שימושי מאוד כשרוצים לגשת למסד נתונים עצום: במקום לטעון את כל המידע למערך, האובייקט יבצע שאילתה למסד הנתונים רק ברגע שנבקש מפתח ספציפי באמצעות סוגריים מרובעים.

4.3 איטרטורים (Iterators)

כברירת מחדל, לולאת foreach רצה על כל המאפיינים הציבוריים של אובייקט. אבל מה אם האובייקט מייצג קובץ הגדרות או תוצאת שאילתה?

PHP מאפשרת להגדיר התנהגות מותאמת אישית ללולאה על ידי מימוש הממשקים Iterator או IteratorAggregate.

המתודות שיש לממש:

1. rewind(): חזרה לתחילת המידע.
2. current(): הערך הנוכחי.
3. key(): המפתח הנוכחי.
4. next(): מעבר לאיבר הבא.
5. valid(): האם נשאר עוד מידע?

4.4 תבניות עיצוב (Design Patterns)

תבניות עיצוב הן פתרונות מקובלים לבעיות תכנות שחוזרות על עצמן. הן מספקות שפה משותפת למפתחים.

4.4.1 תבנית האסטרטגיה (Strategy Pattern)

משמשת כשרוצים להחליף אלגוריתם אחד באחר בקלות.

דוגמה: שרת הורדות שבוחר אם לתת לינק לקובץ zip. (למשתמשי חלונות) או tar.gz. (למשתמשי לינוקס). הקוד הראשי לא משתנה, רק "האסטרטגיה" של יצירת הלינק מוחלפת בזמן ריצה.

4.4.2 תבנית הסינגלטון (Singleton Pattern)

מבטיחה שקיים רק מופע אחד ויחיד של מחלקה בכל האפליקציה (למשל: אובייקט לניהול לוגים או חיבור למסד נתונים).

- איך עושים זאת? הופכים את הבנאי (construct__) ל-private ויוצרים מתודה סטטית בשם getInstance() שמנהלת את יצירת המופע.

4.4.3 תבנית המפעל (Factory Pattern)

במקום ליצור אובייקטים ישירות עם new, משתמשים ב"מפעל" (מתודה סטטית) שמקבל קלט ומחליט איזה סוג של אובייקט לייצר עבורנו.

דוגמה: ("UserFactory::Create("Andi") יחזיר אובייקט מסוג AdminUser או GuestUser בהתאם להרשאות שלו במסד הנתונים.

4.4.4 תבנית הצופה (Observer Pattern)

מאפשרת לאובייקטים "להירשם" לאירועים. כשמידע משתנה באובייקט אחד, כל האובייקטים שנרשמו אליו מקבלים הודעה אוטומטית.

דוגמה: כששער החליפין של הדולר מתעדכן, כל מוצרי החנות (הצופים) מעדכנים את המחיר המוצג שלהם באופן אוטומטי.

4.5 רפלקציה (Reflection / Introspection)

4.5.1 מבוא

יכולות הרפלקציה מאפשרות לכם לבצע אינטרוספקציה (בחינה פנימית) של הסקריפט בזמן ריצה. אתם יכולים לבחון פונקציות, מחלקות, מתודות ופרמטרים, ואף לגשת אליהם או להפעיל אותם דרך המידע שנמצא (Metadata). בעוד ש-PHP תמיד אפשרה קריאה עקיפה לפונקציות (כמו `((...$func$))`), ה-API של הרפלקציה הרבה יותר עשיר ומובנה.

4.5.2 ה-API Reflection

ה-API מורכב מסדרה של מחלקות ייעודיות, שכל אחת מהן מטפלת בישות אחרת בשפה.

הפריטים העיקריים ב-API:

- **ReflectionClass**: מאפשרת לקבל מידע על מחלקה (מתודות, מאפיינים, קבועים, האם היא מופשטת וכו').
- **ReflectionMethod**: בוחנת מתודות בתוך מחלקה (האם היא ציבורית, סטטית, סופית וכו').
- **ReflectionFunction**: בוחנת פונקציות רגילות.
- **ReflectionProperty**: בוחנת משתנים (מאפיינים) של מחלקה.
- **ReflectionParameter**: בוחנת פרמטרים שנשלחים לפונקציה או מתודה (שם הפרמטר, האם הוא מאפשר null).

4.5.3 דוגמאות לשימוש ברפלקציה

4.5.3.1 חילוץ מידע אוטומטי

ניתן להשתמש במתודה הסטטית `ReflectionClass::export()` כדי להדפיס את כל המבנה של מחלקה מסוימת (כולל מחלקות פנימיות של PHP). זהו כלי מעולה לדיבאגינג או ליצירת תיעוד אוטומטי של הקוד שלכם.

4.5.4 מימוש תבנית האצלה (Delegation) באמצעות רפלקציה

לעיתים נרצה שמחלקה אחת תבצע את כל מה שמחלקה אחרת עושה, אך מבלי להשתמש בירושה (כי הקשר אינו קשר של "הוא-סוג-של" או כי המחלקה כבר יורשת ממשהו אחר). במקרה כזה נשתמש בהאצלה (Delegation).

איך זה עובד בדוגמה מהספר?

1. יוצרים מחלקה בשם `ClassOneDelegator`.
2. משתמשים במתודת הקסם `call__()` כדי ליירט כל קריאה למתודה שאינה קיימת.
3. בתוך `call__()`, עוברים על רשימת אובייקטים רשומים ומשתמשים ב-`ReflectionClass` כדי לבדוק: "האם לאובייקט הזה יש מתודה בשם שביקשנו? האם היא ציבורית? האם היא לא מופשטת?".
4. אם התשובה חיובית, מפעילים אותה באמצעות `invoke()`.

4.6 סיכום

פרק זה כיסה את התכונות המתקדמות ביותר של PHP 5. בזכות הכלים הללו (העמסה, איטרטורים, תבניות עיצוב ורפלקציה), PHP הפכה לשפה המסוגלת להריץ אפליקציות ברמת Enterprise בקנה מידה גדול.

המלצה: הספר ממליץ להעמיק בתבניות עיצוב דרך המקורות הקלאסיים, כמו הספר של "כנופיית הארבעה" (Gang of Four), כדי לשדרג את רמת התכנון שלכם.

פרק 5

כיצד לכתוב אפליקציית אינטרנט ב-PHP

"האבטחה האולטימטיבית היא ההבנה שלך את המציאות." — ה. סטנלי ג'אד

5.1 מבוא

השימוש הנפוץ ביותר ב-PHP הוא בניית אתרי אינטרנט. PHP הופכת אפליקציות אינטרנט לדינמיות ומאפשרת למשתמשים ליצור אינטראקציה עם האתר. האפליקציה אוספת מידע מהמשתמש באמצעות טפסי HTML ומעבדת אותו. חלק מהמידע שנאסף רגיש, מה שהופך את האבטחה לנושא מרכזי. PHP מספקת תכונות המאפשרות לאסוף מידע ולאבטח אותו.

לאחר קריאת פרק זה, תלמדו:

- כיצד להטמיע PHP בתוך קבצי HTML.
- כיצד לאסוף מידע ממבקרים באמצעות טפסי HTML.
- טכניקות נפוצות לתקיפת אתרים ואיך להתגונן מפניהן.
- כיצד לטפל בשגיאות בקלט של משתמשים.
- שימוש ב-Cookies וב-Sessions לשמירת נתונים לאורך האפליקציה.
- העלאת קבצים מהמשתמש.
- ארגון נכון של אפליקציית האינטרנט.

5.2 הטמעה (Embedding) בתוך HTML

ניתן ליצור קובץ PHP ללא HTML בכלל, אך בבניית אפליקציות אינטרנט לרוב משלבים בין השניים. PHP פותחה במקור כ"שפת תבניות" (Templating language) המוטמעת ב-HTML. קבצים אלו יקבלו לרוב סיומת `.php`.

כאשר הטקסט להדפסה פשוט, ניתן להשתמש ב-`echo`. אך כאשר הטקסט מכיל גרשיים (בודדים או כפולים), הקוד הופך למסורבל. דוגמה לכתיבה לא נקייה:

PHP

```
<? } ('php if ($user == 'Jerry'?>
שלום ג'רי סיינפלד!
<? } php } else?>
```


בוקר טוב!
<? { php?>

כתיבה זו מפרה את העיקרון: "הפרד לוגיקה מתוכן". פתרון טוב יותר הוא שמירת התוכן במשתנה והדפסתו בסוף.

PHP תומכת גם בתגית מקוצרת: `<? variable$ =?>` שהיא קיצור ל-`<? php echo $variable?>`. עם זאת, מומלץ להשתמש בתגיות המלאות `<? ... php?>` כדי להבטיח תאימות לכל השרתים, שכן התג המקוצר עשוי להיות כבוי בהגדרות השרת.

5.3 קלט משתמש (User Input)

כדי לאסוף נתונים (כמו במערכת רישום), נשתמש בטופס HTML.

שיטות שליחה (Methods):

- **GET**: מקודדת את הנתונים בתוך ה-URL. הם נראים בשורת הכתובת וניתן לשמור אותם בסימניות (Bookmarks). לא מתאים לנתונים רגישים כמו סיסמאות.
- **POST**: שולחת את הנתונים בגוף הבקשה (HTTP Body). הנתונים לא מופיעים בכתובת ה-URL. זו השיטה המועדפת למידע רגיש.

גישה לנתונים ב-PHP:

PHP משתמשת במערכים גלובליים מיוחדים ("סופר-גלובליים"):

- `$_GET` – לנתונים שנשלחו ב-GET.
- `$_POST` – לנתונים שנשלחו ב-POST.
- `$_REQUEST` – מכיל את שניהם (וגם Cookies).

ניתן לבדוק אם טופס נשלח על ידי בדיקה אם כפתור השליחה (Submit) קיים במערך:

```
{ ... } (['if (isset($_POST['register
```

5.4 טיפול בטוח בקלט משתמש

כלל ברזל: אל תסמוך על אף אחד, בטח לא על המשתמשים. משתמשים עשויים להזין נתונים לא צפויים בטעות או בזדון.

5.4.1 טעויות נפוצות

1. **משתנים גלובליים (register_globals)**: בעבר PHP הפכה אוטומטית כל פרמטר ב-URL למשתנה גלובלי. זהו פרצת אבטחה חמורה. בגרסאות מודרניות זה כבוי כברירת מחדל, ויש להשתמש רק במערכים המפורשים (`$_POST` וכדומה).
2. **הזרקת קוד (Cross-Site Scripting - XSS)**: תוקף עשוי להזין קוד JavaScript לתוך שדה טקסט. אם תדפיסו את הטקסט הזה ישירות ל-HTML ללא סינון, הקוד ירוץ בדפדפן של משתמשים אחרים ויכול לגנוב עוגיות.
 - פתרון: שימוש בפונקציות כמו `strip_tags()` או `htmlspecialchars()`.

3. **הזרקת SQL (SQL Injection):** תוקף מזין תווים מיוחדים (כמו גרש ') כדי לשנות את שאלת מסד הנתונים שלכם ולעקוף את מערכת האימות.
- פתרון: שימוש ב-`addslashes()` או בטכניקות מתקדמות יותר כמו Prepared Statements.

5.5 טכניקות להפיכת סקריפטים ל"בטוחים"

5.5.1 אימות קלט (Input Validation)

עליכם לבדוק כל פיסת מידע שמגיעה מהמשתמש. אם אתם מצפים למספר (ID), וודאו שהוא אכן מספר:

```
['prod_id = (int) $_GET['prod_id$
```

כל ערך שאינו מספר יהפוך ל-0.

5.5.2 אימות HMAC

כשאתם מעבירים פרמטרים רגישים ב-URL, תוקף יכול לשנות אותם. כדי למנוע זאת, ניתן לצרף "חתימה" (Hash). שיטת ה-HMAC משתמשת במפתח סודי כדי ליצור חתימה שאי אפשר לזייף ללא המפתח.

5.5.5 עבודה עם סיסמאות

לעולם אל תשמרו סיסמאות כטקסט פשוט (Plain Text) במסד הנתונים. אם האקר יפרוץ למסד הנתונים, כל הסיסמאות ייחשפו.

יש להשתמש בפונקציות גיבוב (Hashing) כמו `sha1()` או `md5()` (היום מומלץ להשתמש ב-`password_hash()` המודרני). חתימות אלו אינן הפיכות – אי אפשר לשחזר את הסיסמה המקורית מהחתימה.

5.6 עוגיות (Cookies)

עוגיות מאפשרות לשמור נתונים בין דפים שונים, שכן האינטרנט כשלעצמו הוא "חסר מצב" (Stateless).

שליחת עוגיה מתבצעת באמצעות:

```
;(setcookie('name', 'value', expiry_time, path
```

חשוב: עוגיות (וכותרות HTTP אחרות) חייבות להישלח לפני שנשלח תוכן HTML כלשהו לדפדפן. אם תנסו לשלוח עוגיה אחרי שכבר הדפסתם טקסט, תקבלו שגיאת "Headers already sent". כדי לעקוף זאת, ניתן להשתמש ב-`ob_start()` (Output Buffering) שאוגר את כל הפלט בזיכרון ושולח אותו רק בסוף.

5.7 ששנים (Sessions)

סשן (Session) ב-PHP מאפשר לאפליקציה לאחסן מידע עבור ה"מפגש" הנוכחי, שניתן להגדירו כמשתמש אחד המחובר לאפליקציה שלכם. סשן מזוהה על ידי מזוהה ייחודי (PHP Session ID). יוצרת מזוהה סשן

שהוא מחרוזת הקסדצימלית המורכבת מגיבוב (MD5 Hash) של כתובת ה-IP המרוחקת, הזמן הנוכחי ומידה נוספת של אקראיות. ניתן להעביר את מזהה הסשן הזה בעוגייה (Cookie) או להוסיפו לכל הכתובות (URLs) בעת הניווט באפליקציה. מטעמי אבטחה, עדיף לאלץ את המשתמש להפעיל עוגיות מאשר להעביר את מזהה הסשן בכתובת ה-URL (מה שבדרך כלל נעשה ידנית על ידי הוספת `PHP_SESSID=` או על ידי הפעלת `session.use_trans_sid` בקובץ `php.ini`), שם הוא עלול להופיע ביומני שרת האינטרנט כ-`HTTP_REFERER` או להימצא על ידי גורם עוין המנטר את התעבורה שלכם. אותו גורם עוין עדיין יכול לראות את נתוני עוגיית הסשן, כמובן, לכן מומלץ להשתמש בשרת המאובטח ב-SSL כדי להיות בטוחים באמת.

כדי להמשיך בדיון על סשנים, נכתוב מחדש את דוגמת העוגיות הקודמת באמצעות סשנים. ניצור קובץ בשם `session.inc` המגדיר כמה ערכי סשן, כפי שמוצג בדוגמה הבאה, ונכלול (Include) קובץ זה בתחילת כל סקריפט המהווה חלק מהסשן:

בשורה הראשונה, פרמטר ההגדרה `session.use_cookies` מוגדר ל-1, מה שאומר שעוגיות יישמשו להעברת מזהה הסשן. בשורה השנייה, `session.use_only_cookies` מוגדר ל-1, מה שאומר שמזהה סשן שיועבר בכתובת ה-URL לסקריפט יידחה. הגדרה זו מחייבת שלמשתמשים יהיו עוגיות מופעלות כדי להשתמש בסשנים. אם אינכם יכולים להסתמך על כך שלאנשים יש עוגיות מופעלות, תוכלו להסיר שורה זו או לשנות את הערך ל-0.

טיפ: ניתן להגדיר את המיקום שבו PHP תאחסן את קבצי הסשן באמצעות הגדרת `session.save_path`.

הפונקציה `session_start()` חייבת להופיע לאחר שכל ההגדרות הקשורות לסשן בוצעו עם `ini_set()`. פונקציה זו מאתחלת את מודול הסשן, מגדירה כותרות (Headers) מסוימות (כמו עוגיית מזהה הסשן וכותרות למניעת זיכרון מטמון), ולכן היא חייבת להופיע לפני שנשלח פלט כלשהו לדפדפן. אם אין מזהה סשן זמין בזמן הקריאה ל-`session_start()`, נוצר מזהה סשן חדש והסשן מאתחל עם מערך `$_SESSION` ריק. הוספת איברים למערך זה היא פשוטה, כפי שמוצג בדוגמה הבאה.

טיפ: ניתן לקרוא ל-`session_name('NAME')` לפני הקריאה ל-`session_start()` כדי לשנות את שם ברירת המחדל של עוגיית הסשן מ-`PHP_SESSID`.

תחילה אנו כוללים את הקובץ `session.inc`. הוספת משתנה הסשן `'uid'` לסשן מתבצעת בקלות על ידי הגדרת איבר ה-`uid` במערך הסופר-גלובלי `$_SESSION` לערך של `uid$`. ביטול הגדרה של משתנה סשן יכול להתבצע עם `unset($_SESSION['uid'])`.

טיפ: אם עליכם לעבד כמות גדולה של נתונים לאחר שינוי משתני הסשן, מומלץ לקרוא ל-`session_write_close()`, דבר המבוצע בדרך כלל אוטומטית בסוף הסקריפט. פעולה זו כותבת את קובץ הסשן לדיסק ומשחררת את הנעילה שלו ממערכת ההפעלה כך שסקריפטים אחרים יוכלו להשתמש בו.

התנתקות (Logout) זהה להשמדת הסשן והנתונים הקשורים אליו: אנו עדיין צריכים לאתחל את הסשן עם `session_start()`, ולאחר מכן נוכל לנקות אותו על ידי הגדרת מערך `$_SESSION` למערך ריק. לבסוף, אנו משמידים את הסשן על ידי קריאה ל-`session_destroy()`.

הגישה למשתני סשן נעשית דרך המערך הסופר-גלובלי `$_SESSION`. בסקריפט `index.php` שלנו, העברנו את הצהרת ה-`if` שבודקת אם המשתמש מחובר לפונקציה מיוחדת בתוך הקובץ `session.inc`:

PHP

```
} function check_login
{ if (!isset($_SESSION['uid']) || !$_SESSION['uid'])
/* אם אין UID בעוגייה, אנו מפנים לדף ההתחברות */
```

```
;'header('Location: http://kossu/session/login.php
{
{
```

אנו קוראים לפונקציה `check_login()` בכל דף שבו אנו דורשים מהמשתמש להיות מחובר. עלינו לוודא שקובץ `session.inc` נכלל לפני שנוצר פלט כלשהו, מכיוון שייתכן שיהיה עליו לשלוח כותרות (Headers) לדפדפן.

5.8 העלאת קבצים (File Uploads)

טרם כיסינו סוג אחד של קלט – העלאת קבצים. ניתן להשתמש בתכונת העלאת הקבצים של PHP כדי להעלות תמונות או חומרים רלוונטיים. מכיוון שהדפדפן צריך לעשות קצת יותר מאשר רק לשלוח בקשת POST עם הנתונים, עליכם להשתמש בטופס שנבנה במיוחד להעלאת קבצים.

ההבדל בין טפסים להעלאת קבצים לטפסים רגילים הוא קריטי. ראשית, אטריביוט ה-`enctype` הכלול בתגית ה-`form` מורה לדפדפן לשלוח סוג אחר של בקשת POST. זהו POST רגיל, אלא שהגוף המכיל את הקבצים המקודדים שונה לחלוטין. במקום תחביר פשוט של `field=var&field2=var2`, נשלח גוף הדומה למייל המכיל טקסט ו-HTML, כאשר כל חלק הוא שדה בטופס.

שדה העלאת הקובץ עצמו הוא מטיפוס `file`, המציג שדה קלט וכפתור "עיון" (Browse) המאפשר למשתמש לחפש קובץ במערכת הקבצים. שדה הקלט הנסתר שולח `MAX_FILE_SIZE` לדפדפן, המגדיר את הגודל המקסימלי המותר לקובץ המועלה, אך רוב הדפדפנים מתעלמים ממנו, ולכן עליכם לבדוק זאת בסקריפט הטיפול.

5.8.1 טיפול בקובץ המועלה

המערך `FILES_$` מכיל מידע על כל קובץ שהועלה. סקריפט הטיפול ניגש למידע באמצעות שם שדה הקובץ כמפתח. המשתנה `FILES['book_image_$]` מכיל את המידע הבא:

מפתח	ערך	תיאור
name	מחרוזת	השם המקורי של הקובץ במחשב של המשתמש.
type	מחרוזת	סוג ה-MIME של הקובץ (למשל image/jpeg).
tmp_name	מחרוזת	שם הקובץ הזמני במערכת הקבצים של השרת.
error	מספר	קוד השגיאה (0 אם הצליח).

size	מספר	גודל הקובץ בבתים.
------	------	-------------------

כמה שגיאות אפשריות במהלך העלאת קובץ קשורות לרוב לגודלו. לכל קוד שגיאה יש קבוע תואם:

#	קבוע	תיאור
0	UPLOAD_ERR_OK	הקובץ הועלה בהצלחה.
1	UPLOAD_ERR_INI_SIZE	הקובץ חרג מהגודל המוגדר ב-php.ini.
2	UPLOAD_ERR_FORM_SIZE	הקובץ חרג מהגודל המוגדר בשדה ה-MAX_FILE_SIZE בטופס.
3	UPLOAD_ERR_PARTIAL	הקובץ הועלה באופן חלקי בלבד.
4	UPLOAD_ERR_NO_FILE	לא נבחר קובץ להעלאה.

בסקריפט העלאת הקובץ, אנו בודקים אם הגודל מקובל (למשל, לא יותר מ-50KB) ואם הקובץ מהסוג הנכון (למשל JPEG או PNG). מכיוון שלא ניתן להסתמך על ה-MAX_FILE_SIZE שמספק המשתמש, עלינו תמיד לבדוק את הגודל בעצמנו באמצעות `FILES['book_image']['size']`. אנו משתמשים בפונקציה `move_uploaded_file()` כדי להעביר את הקובץ ליעדו הסופי. פונקציה זו בודקת אם הקובץ הוא באמת קובץ שהועלה ולא ניסיון הונאה לגשת לקבצים רגישים בשרת.

5.9 ארכיטקטורה

בחלק זה נדון בכמה דרכים לארגון הקוד באפליקציית האינטרנט שלכם.

5.9.1 סקריפט אחד משרת את כולם (One Script Serves All)

הקונספט הוא שסקריפט אחד, בדרך כלל `index.php`, מטפל בכל הבקשות עבור הדפים השונים. תוכן שונה מועבר כפרמטרים לכתובת ה-URL, למשל `page=register?`. מומלץ לטעון מודולים באופן דינמי מספרייה ייעודית.

5.9.2 סקריפט אחד לכל פונקציה

אלטרנטיבה היא אחסון כל פונקציה בסקריפט נפרד הנגיש דרך ה-URL שלו (למשל `about.php`). בשיטה זו אין סקריפט "מנהל", ויש לכלול את הבסיס (כמו טיפול בסשן) בכל קובץ בנפרד.

5.9.3 הפרדת לוגיקה מעיצוב (Layout)

בכל גישה שתבחרו, תמיד עליכם לשאוף להפריד את הלוגיקה מהעיצוב של הדפים. ניתן לעשות זאת באמצעות מנועי תבניות (Templates). קובץ התבנית יהיה החלק ה"סטטי" המכיל HTML עם הצהרות PHP פשוטות להדפסת משתנים הממלאים על ידי הלוגיקה בסקריפט הראשי.

5.10 סיכום

PHP מוטמעת בקלות בתוך קובצי HTML, ומציגה טפסים האוספים נתונים שהוזנו על ידי משתמשים וקבצים שהועלו על ידם. איסוף מידע ממשתמשים מציב סוגיות אבטחה עבור אתר האינטרנט ועבור כל מידע של משתמש המאוחסן באתר. לצרכי אבטחה, יש להגדיר ב-PHP את `register_globals` למצב Off. כדי לתקוף את האתר שלכם או לגנוב את הנתונים שלכם, ה"רעים" משתמשים בטכניקות כמו Cross-site scripting (הרצת קטעי סקריפט בצד הלקוח באתר שלכם) ו-SQL injection (הזרקת קוד זדוני לתוך שאילתות המורצות על מסד הנתונים שלכם). כדי להתגונן מפני התקפות, עליכם להטיל ספק בכל הנתונים שמקורם במשתמשים. עליכם לאמת בקפידה את כל הנתונים שאתם מקבלים ולבדוק אותם ביסודיות כדי לוודא שהם בטוחים ואינם מסוכנים לאתר.

ניתן להגן על האתר בעת העלאת קבצים על ידי בדיקת גודל הקובץ וסוג הקובץ המועלה. בנוסף, ניתן להגן על המידע הגלוי בחלון הכתובת של הדפדפן – מידע המועבר ב-URL – על ידי גיבובו (Hashing) באמצעות אחת מתוך מספר שיטות, כולל מחלקת PEAR בשם `Crypt_HMAC`. גיבוב שימושי גם להגנה על סיסמאות המאוחסנות לצורך אימות משתמשים. אמצעי שימושי נוסף להגנה על האתר מפני טעויות משתמש או התקפות הוא פיתוח מטפל שגיאות (Error handler) משלכם, שיזהה מתי משהו אינו כשורה ויטפל בבעיה.

כדי שאפליקציית אינטרנט תהיה שימושית, נתוני האפליקציה חייבים להיות זמינים לכל דפי האינטרנט באפליקציה במהלך ששן (Session) של משתמש. דרך אחת להעביר נתונים מדף אחד למשנהו היא באמצעות עוגיות (Cookies). כאשר המשתמש ניגש לדף, מוצג דף התחברות, והחשבון והסיסמה שהוזנו נבדקים מול הנתונים המאוחסנים. אם המשתמש מאומת, מוגדרת עוגייה. המידע בעוגייה מועבר אוטומטית עם כל דף מבוקש. שיטה שנייה לשמירה על המשכיות הנתונים היא שימוש בתכונות הסשן של PHP. ברגע שמתחילים סשן, ניתן לאחסן משתנים הזמינים לסקריפטים אחרים באותו סשן.

לאחר הכרת כל החלקים הדרושים לאפליקציה, עליכם לארגן אותם למכלול שימושי. שיטה נפוצה אחת לארגון נקראת "סקריפט אחד משרת את כולם" (one script serves all), שמשמעותה היא ש-`index.php` מטפל בכל הבקשות לדפים השונים. ארגון נפוץ אחר הוא "סקריפט אחד לכל פונקציה". עיקרון כללי הוא להפריד בין העיצוב (Layout) ללוגיקה. לאחר ארגון החלקים לאפליקציה מקיפה, אתם מוכנים לצאת לדרך.

פרק 6: מסדי נתונים עם PHP 5

חלק בלתי נפרד מכל ספר PHP הוא נושא מסדי הנתונים והממשק בינם לבין PHP. ספר זה אינו שונה, פשוט מכיוון שרוב האנשים שכותבים אפליקציות PHP מעוניינים להשתמש במסד נתונים.

פרק זה מציג את השימוש ב-MYSQL וב-SQLite מתוך PHP, אך מתמקד בעיקר בפרטים הספציפיים ל-PHP 5 בכל הנוגע לממשקי מסדי נתונים. בסוף פרק זה תלמדו:

- נקודות חוזק וחולשה של MYSQL ו-SQLite.
- עבודה עם MYSQL באמצעות הרחבת mysqli החדשה.
- שימוש בהרחבת sqlite המובנית של PHP 5.
- שימוש ב-PEAR DB לכתובת קוד מסד נתונים נייד (Portable) יותר.

הערה לגבי גרסאות: פרק זה מתמקד בתכונות הקישוריות החדשות של PHP 5, ובפרט בהרחבות mysqli ו-sqlite. כדי ליהנות מהפונקציונליות המתוארת, עליכם להשתמש בגרסאות עדכניות: MYSQL 4.1.2 ומעלה, SQLite כפי שמצורף ל-PHP 5.0.0 ומעלה, ו-PEAR DB 1.6 ומעלה.

MYSQL 6.2

MYSQL ו-PHP הפכו ל"לחם והחמאה" של בוני אפליקציות אינטרנט. זהו השילוב שסביר להניח שתפגשו כיום ובשנים הבאות. פרק זה מתמקד בהרחבת mysqli (או MYSQL Improved) המצורפת ל-PHP 5, הדורשת לפחות את גרסה 4.1.2 של שרת ה-MYSQL.

6.2.1 נקודות חוזק וחולשה של MYSQL

- חוזק: חדירה עמוקה לשוק. ל-MYSQL נתח השוק הגדול ביותר מבין מסדי הנתונים בקוד פתוח. כמעט כל חברת אחסון אתרים מספקת גישה ל-MYSQL.
- חוזק: קלות התחלה. ניהול מסד הנתונים הוא פשוט וישנם כלים כמו phpMyAdmin לניהול נוח.
- חוזק: רישיון קוד פתוח לרוב המשתמשים. ניתן להשתמש ב-MYSQL תחת רישיון GPL כל עוד אינכם מפיצים אותה מסחרית.
- חוזק: מהירות. MYSQL תמיד הייתה מהירה יחסית בשל פשטותה, ועם השנים הוסיפה תכונות של "Enterprise Class" מבלי להתפשר על ביצועים.
- חוזק: סקילביליות (יכולת צמיחה) סבירה. MYSQL התפתחה כך שהיא כמעט משתווה למערכות מסחריות גדולות כמו Oracle מבחינת אמינות, אך עדיין ניתנת להגדרה לשימוש "קל".
- חולשה: רישיון מסחרי להפצה מסחרית. אם אתם מאגדים את MYSQL עם מוצר מסחרי בקוד סגור, עליכם לרכוש רישיון.

6.2.2 ממשק PHP

הרחבת mysqli נכתבה מהיסוד כדי לתמוך בתכונות החדשות של MYSQL 4.1 ו-PHP 5.0. השיפורים לעומת הרחבת mysql הישנה כוללים:

- פונקציונליות Native bind/prepare/execute.
- תמיכה ב-Cursors.
- קודי שגיאה בתקן SQLSTATE.
- הרצת מספר הצהרות (Statements) בשאילתה אחת.

לכל פונקציית mysql יש מקבילה כמתודה או כמאפיין (Property) בגישה מונחית עצמים.

6.2.4 חיבורים (Connections)

טבלה 6.1 מציגה את פונקציות ומתודות החיבור.

טבלה 6.1: פונקציות ומתודות חיבור ב-mysql

- `new mysql / mysql_connect(...)`: פותח חיבור חדש לשרת MySQL.
- `$mysql->close / mysql_close(...)`: סוגר חיבור פתוח.
- `mysql_connect_errno()`: מחזיר את קוד השגיאה של ניסיון החיבור האחרון שנכשל.

6.2.5 שאילתות נאגרות מול שאילתות לא נאגרות (Buffered vs. Unbuffered)

ללקוח MySQL יש שני סוגי שאילתות:

1. שאילתות נאגרות (Buffered): התוצאות נשלפות ומאוחסנות בזיכרון של צד הלקוח.
 - יתרון: ניתן לנווט בתוך התוצאות בחופשיות (Seeking).
 - חיסרון: דורש זיכרון נוסף ועלול לעכב את החזרת הפלט עד שכל התוצאות נשלפו.
2. שאילתות לא נאגרות (Unbuffered): מגבילות אתכם לגישה סדרתית בלבד לתוצאות, אך אינן דורשות זיכרון נוסף לאחסון כל סט התוצאות.
 - יתרון: חסכון בזיכרון בסטים גדולים של נתונים; ניתן להתחיל לעבד שורות מיד כשהשרת מתחיל להחזירן.

הבחירה בסוג השאילתה תלויה במצב: שאילתות לא נאגרות חוסכות זיכרון כשהתוצאה גדולה מאוד, בעוד שאילתות נאגרות נוחות יותר בזכות יכולת הניווט בתוצאות.

6.2.6 שאילתות (Queries)

חלק זה מתאר פונקציות ומתודות לביצוע שאילתות (ראו טבלה 6.3).

טבלה 6.3: פונקציות שאילתה ב-mysql

שם הפונקציה	תיאור
<code>mysqli_query(...)</code>	שולחת שאילתה לבסיס הנתונים ומחזירה אובייקט תוצאה. פרמטרים: חיבור (בפונקציה בלבד), שאילתה (מחרוזת), מצב (נאגר או לא נאגר).
<code>mysqli_multi_query(...)</code>	שולחת ומעבדת מספר שאילתות בבת אחת. פרמטרים: אובייקט חיבור (בפונקציה בלבד), שאילתה (מחרוזת).

הפונקציה `mysqli_query()` מחזירה אובייקט של סט תוצאות. במקרה של כישלון, השתמשו בפונקציה `mysqli_error()` או במאפיין `conn->error` כדי לקבוע את סיבת הכישלון:

PHP

```

$result = $conn->query("SELECT Name FROM City$
    } while ($row = $result->fetch_row
        ;print $row[0] . "<br>\n
    {
        ;$result->free$
        ;$conn->close$

```

לאחר ביצוע השאילתה, מוקצה זיכרון בצד הלקוח כדי לאחזר את סט התוצאות המלא. כדי להשתמש בסט תוצאות שאינו נאגר (Unbuffered), עליכם לציין את הפרמטר האופציונלי `MYSQLI_USE_RESULT`:

PHP

```

$result = $conn->query("SELECT Name FROM City", MYSQLI_USE_RESULT$
    } while ($row = $result->fetch_row
        ;print $row[0] . "<br>\n
    {
        ;$result->free$
        ;$conn->close$

```

6.2.7 הצהרות מרובות (Multi Statements)

הרחבת `mysqli` מאפשרת לכם לשלוח מספר הצהרות SQL בקריאה אחת לפונקציה באמצעות `mysqli_multi_query`. מחרוזת השאילתה מכילה הצהרת SQL אחת או יותר המופרדות בנקודה-פסיק בסוף כל הצהרה. אחזור סטים של תוצאות מהצהרות מרובות הוא מעט מורכב, כפי שמדגימה הדוגמה הבאה:

PHP

```

} ((if ($mysqli->multi_query($query
    } do

```

```

} ((if ($result = $mysqli->store_result
} ((while ($row = $result->fetch_row
;([printf("Col: %s\n", $row[0
{
;())result->close$
{
;())while ($conn->next_result {
{
;())conn->close$

```

6.2.8 מצבי שליפה (Fetching Modes)

קיימות שלוש דרכים לשלוף שורות של תוצאות, בדומה להרחבת mysql הישנה: כמערך ממוספר, כמערך אסוציאטיבי או כאובייקט (ראו טבלה 6.4).

טבלה 6.4: פונקציות שליפה ב-mysqli

שם הפונקציה	תיאור
<code>mysqli_fetch_row(...)</code>	שולפת שורה כמערך ממוספר. הפרמטר הוא אובייקט התוצאה (בפונקציה בלבד).
<code>mysqli_fetch_assoc(...)</code>	שולפת שורה כמערך אסוציאטיבי. הפרמטר הוא אובייקט התוצאה (בפונקציה בלבד).
<code>mysqli_fetch_object(...)</code>	שולפת שורה לתוך אובייקט. הפרמטר הוא אובייקט התוצאה (בפונקציה בלבד).

6.2.9 הצהרות מוכנות (Prepared Statements)

אחד היתרונות המרכזיים של הרחבת mysqli בהשוואה ל-mysql הוא "הצהרות מוכנות". אלו מספקות למפתחים את היכולת ליצור שאילתות מאובטחות יותר, בעלות ביצועים טובים יותר ונוחות יותר לכתיבה.

ישנם שני סוגים של הצהרות מוכנות: כאלו המבצעות מניפולציית נתונים (Data Manipulation) וכאלו המבצעות שליפת נתונים (Data Retrieval). הצהרות אלו מאפשרות לקשור (Bind) משתני PHP ישירות לקלט ולפלט.

תהליך היצירה פשוט: תבנית שאילתה נוצרת ונשלחת לשרת ה-MYSQL. השרת מתקף אותה, מנתח (Parses) אותה ומאחסן אותה בבאפר מיוחד. לאחר מכן הוא מחזיר "ידיית" (Handle) מיוחדת שניתן להשתמש בה מאוחר יותר כדי להתייחס להצהרה המוכנה.

6.2.9.1 קישור משתנים (Binding Variables)

ישנם שני סוגי משתנים קשורים: משתני קלט (Input) הנקשרים להצהרה, ומשתני פלט (Output) הנקשרים לסט התוצאות. עבור משתני קלט, עליכם לציין סימן שאלה כמציין מקום (Placeholder) בהצהרת ה-SQL שלכם:

`?=SELECT Id, Country FROM City WHERE City`

משתני קלט חייבים להיקשר **לפני** ביצוע ההצהרה, בעוד משתני פלט נקשרים **לאחר** הביצוע.

תהליך משתני הקלט:

1. הכנה (ניתוח) של ההצהרה.
2. קישור משתני הקלט.
3. הקצאת ערכים למשתנים הקשורים.
4. ביצוע (Execute) ההצהרה המוכנה.

תהליך משתני הפלט:

1. הכנה של ההצהרה.
2. ביצוע ההצהרה המוכנה.
3. קישור משתני הפלט.
4. שליפת (Fetch) הנתונים לתוך משתני הפלט.

טבלה 6.5: פונקציות להצהרות מוכנות ב-mysql

שם הפונקציה	תיאור
<code>mysql_prepare(...)</code>	מכינה הצהרת SQL לביצוע.
<code>mysql_stmt_bind_param(...)</code>	קושרת משתנים להצהרה (קלט). מציינת טיפוסים (i=s, מחזורות וכו').
<code>mysql_stmt_execute(...)</code>	מבצעת את ההצהרה המוכנה.
<code>mysql_stmt_bind_result(...)</code>	קושרת משתנים לסט התוצאות (פלט).
<code>mysql_stmt_fetch(...)</code>	שולפת את הנתונים לתוך המשתנים הקשורים.

דוגמה לשליפת נתונים עם קישור פלט:

```

$stmt = $mysqli->prepare("SELECT year, model, accel FROM alfas ORDER BY year$
;($stmt->execute$
;($stmt->bind_result($year, $model, $accel$
} (($while ($stmt->fetch
;($printf("%d %s %.1f sec\n", $year, $model, $accel
{

```

6.2.10 טיפול ב-BLOB

BLOB ראשי תיבות של Binary Large Object ומתייחס לנתונים בינאריים, כגון תמונות JPEG המאוחסנות בבסיס הנתונים.

6.2.10.1 הזנת נתוני BLOB

בעבר, נתוני BLOB הוכנסו ישירות כחלק מהשאלתה. ב-mysqli, כאשר מכניסים קבצים גדולים, יעיל יותר להשתמש במתודה `send_long_data()`. שיטה זו מאפשרת לשלוח את הקובץ בחלקים (למשל 1KB בכל פעם) מבלי ש-PHP תצטרך לאגור את כל הקובץ בזיכרון בבת אחת.

6.3 SQLITE

PHP 5 הציגה מנוע בסיס נתונים מובנה הזמין כברירת מחדל בשם SQLite.

6.3.1 חוזקות וחולשות של SQLite

- **חוזקה: ללא שרת (Serverless) - SQLite** אינה משתמשת במודל לקוח/שרת. היא מוטמעת באפליקציה ודורשת רק גישה לקבצי בסיס הנתונים.
- **חוזקה: קלות התחלה** - יצירת בסיס נתונים חדש אינה דורשת התערבות של מנהל מערכת.
- **חוזקה: קלות משקל ומהירה** - עבור רוב השאלות, הביצועים שלה משתווים או עולים על אלו של MySQL.
- **חולשה: בעיות הרחבה (Scaling)** - היעדר שרת מוביל לקשיים בנעילת קבצים ובגישה בו-זמנית של משתמשים רבים.
- **חולשה: טרנזקציות נועלות את כל בסיס הנתונים - SQLite** נועלת את כל הקובץ בזמן כתיבה, מה שעלול לעכב קריאות בו-זמניות.

6.3.3 PHP ממשק

SQLite ב-PHP 5 מציעה גם ממשק פרוצדורלי וגם ממשק מונחה עצמים (OO). יצירת בסיס נתונים היא פשוט פתיחת קובץ; אם הוא לא קיים, PHP תיצור אותו.

טבלה 6.6: פתיחה וסגירה של בסיס נתונים

שם הפונקציה	תיאור
<code>sqlite_open(...)</code>	מחבר את הסקריפט לבסיס נתונים SQLite או יוצר אחד חדש.

מנתק את החיבור.	sqlite_close(...)
-----------------	-------------------

6.3.3.2 שאליות פשוטות

ב-SQLite ישנם רק שני טיפוסים פנימיים עיקריים: `INTEGER` (למספרים) ו-"כל השאר" (הדומה ל-`VARCHAR` ויכול לאחסן טקסט ארוך מאוד). בניגוד להרחבות אחרות, SQLite מאפשרת לבצע מספר שאליות `CREATE TABLE` בקריאה אחת ל-`query()`.

6.3.3.4 טרנזקציות

כברירת מחדל, SQLite כותבת כל שאליתה לדיסק מיד, מה שעלול להפוך הזנה של נתונים רבים לאיטית. כדי להאיץ את התהליך, ניתן להשתמש בטרנזקציות (`BEGIN` ו-`COMMIT`). בדוגמה של ייבוא 638 אימיילים, השימוש בטרנזקציה הוריד את זמן הייבוא מכ-60 דקות לפחות מ-2 דקות.

6.3.3.6 פונקציות מוגדרות משתמש (UDFs)

אחת התכונות המרשימות של SQLite היא היכולת לרשום פונקציות PHP כפונקציות SQL לכל דבר. ניתן להשתמש ב-`createFunction()` כדי לקשור פונקציית PHP לשאלת SQL, ובכך לבצע עיבוד נתונים מורכב (כמו אינדוקס מילים) ישירות מתוך ה-SQL.

טבלה 6.10: פונקציות שאלתה ב-SQLite

שם הפונקציה	תיאור
sqlite_query()	מבצעת שאלתה פשוטה.
sqlite_unbuffered_query()	מבצעת שאלתה ללא אגירה בלקוח (חוסך זיכרון).
sqlite_single_query()	מבצעת שאלתה ומחזירה רק את העמודה הראשונה מהרשומה הראשונה.

6.3.3.9 איטרטורים (Iterators)

ב-PHP 5 ניתן לעבור על סט תוצאות של SQLite גם באמצעות איטרטור (לולאת `foreach`), דבר המהיר יותר מאשר שימוש בפונקציות שליפה (`fetch`) מסורתיות.

6.3.3.10 איטרציה עצמית (Homegrown Iteration)

כדי לראות בצורה ברורה יותר כיצד האיטרטור עובד פנימית, ניתן לבצע זאת גם ידנית (ללא הקסם של `foreach`), כפי שמוצג כאן בחלקו השני של הסקריפט:

PHP

```

" = details_query$
(SELECT document_id, substr(doc.body, position - 20, 100
FROM dictionary d, lookup l, document doc
WHERE d.id = l.word_id
('AND word in ('$words
(AND document_id IN ($doc_ids
AND document_id = doc.id
GROUP BY document_id, doc.body
;";
;(result = $db->unbufferedQuery($details_query, SQLITE_NUM$
} (())while ($result->valid
;())record = $result->current$
;[list[$record[0]] = $record[1$
;())result->next$
{

```

כברירת מחדל, ה-`result` מצביע על השורה הראשונה בעת תחילת האיטרציה, והמתודה `current` מחזירה את הרשומה הנוכחית (באינדקס לפי הסוג שצוין בפרמטר השני של `unbufferedQuery`). באמצעות המתודה `next`, ניתן להתקדם לרשומה הבאה בסט התוצאות. קיימות עוד מספר מתודות שניתן להשתמש בהן; הטבלה הבאה מציגה אילו מהן קיימות, וגם מונה את הפונקציות הפרוצדורליות המקבילות להן. הפרמטר הראשון בפונקציות הממשק הפרוצדורלי הוא תמיד ה"ידיית" (`Handle`) של התוצאה, והוא אינו מופיע בטבלה 6.12.

טבלה 6.12: פונקציות ומתודות לניווט בסט תוצאות

שם המתודה	תיאור
<code>result->seek()</code> /\$ <code>(sqlite_seek)</code>	מדלג לשורה מסוימת בסט התוצאות. הפרמטר היחיד הוא מספר הרשומה (מבוסס 0). ניתן לשימוש רק בשאילתות נאגרות (<code>Buffered</code>).
<code>result->rewind()</code> /\$ <code>(sqlite_rewind)</code>	מחזיר את מצביע התוצאה לרשומה הראשונה. ניתן לשימוש רק בשאילתות נאגרות.
<code>result->next()</code> /\$ <code>(sqlite_next)</code>	מתקדם לרשומה הבאה בסט התוצאות.
<code>result->prev()</code> /\$ <code>(sqlite_prev)</code>	מחזיר את מצביע התוצאה לרשומה הקודמת. ניתן לשימוש רק בשאילתות נאגרות.

מחזיר האם קיימות רשומות נוספות בסט התוצאות.	<code>result->valid()</code> /\$ <code>(sqlite_valid</code>
מחזיר האם קיימת רשומה קודמת. לא ניתן לשימוש בשאלות לא נאגרות.	<code>result->hasPrev()</code> /\$ <code>(sqlite_has_prev</code>

כעת נותר רק החלק האחרון של סקריפט החיפוש שלנו – החלק שבו אנו מוציאים את התוצאות בפועל:

PHP

```
} (foreach ($rank as $record
, "...echo $record[0], "\n===\n
;list[$record[0]], "... \n-----\n$
{
<?
```

כאן, אנו פשוט עוברים שוב על תוצאת השאילתה הראשונה ומשתמשים ב-ID של ההודעה כמפתח לסט התוצאות כדי להציג את החלקים הרלוונטיים מהאימיילים שנמצאו.

6.3.3.11 פונקציות נוספות הקשורות לסט התוצאות

ניתן להשתמש בכמה פונקציות ומתודות נוספות על סטים של תוצאות. המתודה `numFields()` (או `sqlite_num_fields()`) מחזירה את מספר השדות בסט התוצאות, והמתודה `fieldName()` (או `sqlite_field_name()`) מחזירה את שם השדה. הפרמטר היחיד למתודה זו הוא האינדקס של השדה בתוך סט התוצאות (מבוסס 0). אם אתם מבצעים Join בין מספר טבלאות, שימו לב שפונקציה זו מחזירה את שם השדה "כפי שהוא" מהשאילתה; לדוגמה, אם השאילתה מכילה "SELECT a.field1 FROM address a", שם השדה שיוחזר יהיה "a.field1".

מוזרות נוספת עם שמות עמודות, התקפה גם לגבי מפתחות במערכים מוחזרים כאשר האופציה `SQLITE_ASSOC` מוגדרת, היא שהם תמיד מוחזרים באותו Case (אותיות גדולות/קטנות) שבו הם נוצרו בהצהרת ה-"CREATE TABLE". על ידי הגדרת האופציה `sqlite.assoc_case` ב-`php.ini` ל-1, אתם מאלצים את הרחבת SQLite להחזיר שמות עמודות באותיות גדולות (Uppercase). הגדרה ל-2 תאלץ אותיות קטנות (Lowercase). הגדרה ל-0 (ברירת המחדל) לא תיגע ב-Case של שמות העמודות כלל.

המתודה `numRows()` (או `sqlite_num_rows()`) מחזירה את מספר הרשומות בסט התוצאות, אך עובדת רק עבור שאילתות נאגרות (Buffered).

6.3.3.12 פונקציות אגרגציה מוגדרות משתמש (Aggregate UDFs)

מלבד UDFs רגילים בדומה לאלו שהשתמשנו בהם כדי ליצור את האינדקס שלנו מתוך Trigger, ניתן גם להגדיר UDF עבור פונקציות אגרגציה (צבירה). בדוגמה הבאה, אנו מחשבים את אורך המילים הממוצע במילון שלנו:

PHP

```
)db->createAggregate$
```

```

        'average_length',
        'average_length_step', 'average_length_finalize'
    );

```

המתודה `createAggregate()` יוצרת את פונקציית האגרגציה שלנו. הפרמטר הראשון הוא שם הפונקציה לשימוש בשאליות SQL; השני הוא הפונקציה שמתבצעת עבור כל רשומה (הנקראת גם step); והפרמטר השלישי הוא שם הפונקציה שרצה כאשר כל הרשומות נבחרו.

PHP

```

    $avg = $db->singleQuery(
        "SELECT average_length(word) FROM dictionary"
    );
    echo $avg;
}

```

כאן, אנו פשוט מריצים את השאלית תוך שימוש בפונקציה החדשה שהגדרנו ומדפיסים את התוצאה, שאמורה להיראות בערך כך:

Average over 28089 words is 10.038 chars

6.3.3.13 קידוד תווים (Character Encoding)

ל-SQLite יש תמיכה בשני סטים של תווים: ISO-8859-1, שהוא ברירת המחדל ומשמש לרוב השפות המערב-אירופאיות, ו-UTF-8. כדי להפעיל מצב UTF-8, עליכם להורות לפקודת ה-`configure()` של PHP לעשות זאת. המתג לשימוש במצב UTF-8 של SQLite הוא `enable-sqlite-utf8`. אופציה זו משפיעה רק על מיון התוצאות.

6.3.3.14 כונון ביצועים (Tuning)

כבר ראינו שניתן להאיץ כמויות גדולות של הזנות נתונים (Inserts) על ידי עטיפת השאליות בתוך Transaction. אך ישנם עוד כמה טריקים. בדרך כלל, בעת הזנת נתונים רבים, איננו מעוניינים לדעת כמה שינויים היו בסט התוצאות. SQLite מאפשרת לכבות את ספירת השינויים, מה שמספר את המהירות בזמן ההזנה:

```
PRAGMA count_changes = 0
```

טריק נוסף הוא לשנות את הדרך שבה SQLite מרוקנת (Flushes) נתונים לדיסק. עם ה-`synchronous=0`, ניתן לעבור בין המצבים הבאים, כפי שמוצג בטבלה 6.13.

טבלה 6.13: אופציות "PRAGMA Synchronous"

מצב	תיאור

SQLite לא תבצע ריקון לדיסק כלל; הטיפול בכך מוטל על מערכת ההפעלה.	OFF
במצב זה, SQLite תוודא שהנתונים נכתבו לדיסק על ידי ביצוע קריאת המערכת fsync() מדי פעם.	ON/NORMAL (ברירת מחדל)
SQLite תבצע קריאות fsync() נוספות כדי להפחית את הסיכון לשחיתות נתונים במקרה של הפסקת חשמל.	FULL

במצבים שבהם יש הרבה קריאות ממסד הנתונים, כדאי להגדיל את גודל המטמון (Cache). בעוד שבברירת המחדל היא 2,000 דפים (דף הוא 1,536 בתים), ניתן להגדיל זאת כך:

```
;PRAGMA cache_size=5000
```

הגדרה זו תקפה רק לסשן הנוכחי. אם ברצונכם שהיא תישמר, עליכם להשתמש ב-default_cache_size במקום.

6.3.3.15 טריקים נוספים

נותרו עוד כמה דברים לגבי SQLite – למשל, מהי המתודה לתשאל מבנה מסד הנתונים? התשובה קלה – באמצעות השאילתה הבאה:

```
SELECT * FROM sqlite_master
```

שאילתה זו מחזירה איבר אחד לכל אובייקט במסד הנתונים (טבלה, אינדקס וטריגר) עם המידע הבא: סוג האובייקט, שמו, הטבלה אליה הוא מקושר, ID, ושאילתת ה-SQL DDL ליצירת האובייקט.

6.3.3.16 דברי חכמה

לסיום, הנה כמה דברי חכמה מאת מחבר מנוע ה-SQLite, בהם הוא משתמש במקום הודעת זכויות יוצרים:

- מי ייתן ותעשה טוב ולא רע.
 - מי ייתן ותמצא מחילה לעצמך ותסלח לאחרים.
 - מי ייתן ותשתף בחופשיות, ולעולם לא תיקח יותר ממה שאתה נותן.
- ד. ריצ'רד היפ

PEAR DB 6.4

חבילת ה-PEAR DB הנפוצה ביותר לגישה למסדי נתונים היא PEAR DB. זוהי שכבת הפשטה (Abstraction Layer) המספקת API אחיד לתשאל רוב מסדי הנתונים הנתמכים על ידי PHP, כמו גם דברים ספציפיים למסדי נתונים בדרך ניידת (Portable), כגון רצפים (Sequences) וטיפול בשגיאות.

6.4.2 יתרונות וחסרונות של הפשטת מסדי נתונים

שני היתרונות המרכזיים בשימוש בשכבת הפשטה כמו PEAR DB הם:

1. **API אחיד שקל לזכור.** אתם פרודוקטיביים יותר כשאתם מבזבזים פחות זמן בחיפוש בתייעוד.
2. **API אחיד המאפשר לרכיבים אחרים להשתמש ב-DB API** לגישה גנרית ל-DBMS, מבלי לדאוג לפרטים הספציפיים של ה-Back-end.

מכיוון ש-DB ממומש ב-PHP, יתרונות אלו מגיעים עם מחיר:

- שכבה הכתובה ב-PHP איטית יותר משימוש בפונקציות PHP מובנות.
- שכבת הקוד הנוספת מוסיפה מורכבות ומקורות פוטנציאליים לשגיאות.

6.4.4 חיבורי מסד נתונים

PEAR DB שואל את המונח (Data Source Name (DSN מ-ODBC כדי לתאר כיצד פונים למסד נתונים. DSNs משתמשים בפורמט URI. זהו דוגמה ל-DSN המתייחס למסד נתונים מסוג mysql ב-localhost בשם "world":

mysql://user:password@host/world

הפורמט המלא של DSN הוא הרבה יותר מפורט, ורוב השדות הם אופציונליים. למעשה, רק שם הרחבת מסד הנתונים הוא חובה עבור כל הדרייברים. הדרייברים המצורפים ל-DB הם: dbase, fbsql, ibase, ifx, mysql, mssql, mysql, oci8, odbc, pgsql, sqlite.

האם תרצה שאמשיך עם תרגום קטעי הקוד של PEAR DB או הסברים על פונקציות הקישור שלו?

הנה תרגום הפרק לעברית (טקסט מתורגם בלבד):

6.4.4.2 יצירת חיבורים

הנה דוגמה כיצד ליצור חיבור למסד נתונים באמצעות PEAR DB:

PHP

```
require_once "DB.php";
$dsn = "mysql://test@localhost/test";
$dbh = DB::connect($dsn);
if (DB::isError($dbh)) {
    print "Error connecting: " . $dbh->getMessage() . "\n";
    print "Error details: " . $dbh->getUserInfo() . "\n";
    exit(1);
}
print "Connect ok!\n";
```

סקריפט זה מתחבר למסד הנתונים "test" באמצעות הרחבת mysql. שרת מסד הנתונים רץ על localhost, והחיבור ייפתח כמשתמש "test" ללא סיסמה.

DB.php הוא הקובץ היחיד שעליכם לכלול כדי להשתמש ב-PEAR DB. המתודה DB::connect() היא מתודת "מפעל" (Factory method) שכוללת אוטומטית את הקובץ המתאים עבור הדרייבר שלכם. היא יוצרת אובייקט דרייבר, מאתחלת אותו וקוראת לפונקציה המובנית ליצירת החיבור בפועל. במקרה של כישלון, היא תעורר שגיאת PEAR.

עבור מסדי נתונים של **SQLite**, כל שעליכם לציין הוא את הרחבת ה-PHP ואת קובץ מסד הנתונים: `sqlite:///test.db`. במקרה זה, הקובץ ייפתח מהתיקיה הנוכחית. כדי לציין נתיב מלא, יש להוסיף לוכסן נוסף לפני שם הקובץ: `sqlite:///var/lib/sqlite/test.db`.

6.4.4.3 אפשרויות הגדרה (Configuration Options)

ניתן להגדיר חלק מהתנהגות ה-DB לכל חיבור באמצעות המתודה `setOption()`. האפשרויות הנתמכות הן:

- **persistent**: (בוליאני) האם להשתמש בחיבור קבוע (Persistent connection).
- **ssl**: (בוליאני) האם להשתמש בחיבורי SSL מאובטחים.
- **seqname_format**: (מחרוזת) פורמט השם עבור רצפים (Sequences) מדומים. ברירת המחדל היא `%s_seq`.
- **autofree**: (בוליאני) האם לשחרר אוטומטית את סטי התוצאות לאחר סיום השאילתות.
- **portability**: (מספר שלם) קובע אילו תכונות ה-DB צריך לדמות עבור ניידות בין מסדי נתונים שונים.

6.4.5 ביצוע שאילתות

ישנן ארבע דרכים להרצת שאילתות עם PEAR DB, כולן מבוצעות על ידי קריאה למתודות באובייקט החיבור:

1. `query($query, $params = array)`: הדרך הסטנדרטית. אם יש תוצאות, היא מחזירה אובייקט תוצאה; אחרת, מחזירה ערך בוליאני המעיד על הצלחה.
2. `limitQuery($query, $from, $count, $params = array)`: זהה ל-`query()`, אך מאפשר להגביל את מספר התוצאות ולהגדיר היסט (Offset).
3. `prepare()` ו-`execute()`: משמשות להכנת שאילתה וביצועה עם פרמטרים. זה חוסך זמן ניתוח (Parsing) של השאילתה כאשר מריצים אותה פעמים רבות עם נתונים שונים (למשל בלולאת INSERT).
4. `simpleQuery($query)`: מיועדת לשאילתות מניפולציית נתונים שאינן מחזירות תוצאות. היא בעלת תקורה (Overhead) נמוכה במיוחד.

6.4.6 שליפת תוצאות

הכיתה `DB_result` מציעה מספר מצבי שליפה (Fetch Modes):

- **DB_FETCHMODE_ORDERED**: מחזיר מערך עם אינדקס מספרי (ברירת מחדל).
- **DB_FETCHMODE_ASSOC**: מחזיר מערך אסוציאטיבי שבו שמות העמודות הם המפתחות.
- **DB_FETCHMODE_OBJECT**: מחזיר אובייקט שבו שמות העמודות הם המאפיינים (Properties).

ניתן להגדיר את מצב השליפה לכל החיבור באמצעות `setFetchMode()` או לשנות אותו נקודתית בכל קריאה ל-`fetchRow()` או `fetchInto()`.

6.4.7 רצפים (Sequences)

רצפים בבסיסי נתונים הם עניין מורכב לניידות, כיוון שכל מסד נתונים (כמו Oracle מול MySQL) מטפל בהם אחרת. PEAR DB מציע API אחיד ליצירת מספרי זיהוי (ID) ייחודיים באמצעות המתודות:

- `nextId($seqname)`: מחזירה את המספר הבא ברצף.
- `createSequence($seqname)`: יוצרת רצף חדש.
- `dropSequence($seqname)`: מוחקת רצף.

6.4.8 תכונות ניידות (Portability)

החל מגרסה 1.6, ניתן להפעיל או לכבות תכונות ניידות ספציפיות כדי לאזן בין ביצועים ליכולת מעבר בין מסדי נתונים:

- **DB_PORTABILITY_LOWERCASE**: הופך את כל שמות העמודות לאותיות קטנות.
- **DB_PORTABILITY_RTRIM**: מסיר רווחים מיותרים מסוף מחרוזות.
- **DB_PORTABILITY_DELETE_COUNT**: מאפשר לדעת כמה שורות נחקו גם במסדי נתונים שאינם תומכים בכך כברירת מחדל.
- **DB_PORTABILITY_ERRORS**: מתקן מיפוי שגיאות כך שיהיה עקבי בין כל הדרייברים.

6.4.9 שגיאות מופשטות (Abstracted Errors)

במקום להסתמך על קודי שגיאה שונים מכל שרת (MySQL, Oracle וכו'), PEAR DB משתמש בסט קבועים משלו, כגון:

- **DB_ERROR_ACCESS_VIOLATION** (בעיית הרשאות)
- **DB_ERROR_NOSUCHTABLE** (טבלה לא קיימת)
- **DB_ERROR_SYNTAX** (שגיאת תחביר SQL)

6.4.10 מתודות נוחות (Convenience Methods)

PEAR DB כולל מתודות לקיצור תהליכי שליפה נפוצים:

- **getOne()**: מחזירה רק את הערך הראשון מהשורה הראשונה.
- **getRow()**: מחזירה את כל השורה הראשונה כמערך.
- **getCol()**: מחזירה עמודה ספציפית מכל השורות כמערך.
- **getAssoc()**: מחזירה מערך אסוציאטיבי שבו העמודה הראשונה היא המפתח.
- **getAll()**: מחזירה את כל הנתונים מכל השורות כמערך של מערכים.

6.5 סיכום

פרק זה הציג שתי הרחבות מסדי נתונים חדשות ב-mysqli ו-PHP 5 ו-sqlite, וכן את PEAR DB כשכבת הפשטה למדעם:

- את היתרונות והחסרונות של MySQL מול SQLite.
- מתי כדאי להשתמש בשכבת הפשטה למסדי נתונים.
- כיצד לבצע שאילתות, לשלוף תוצאות ולטפל בשגיאות בכל אחת מהשיטות.
- את ההבדל בין שאילתות נאגרות (Buffered) ללא נאגרות.
- כיצד להשתמש בטריגרים ובפונקציות מוגדרות משתמש (UDF) ב-SQLite.

האם תרצה שאסביר בהרחבה על אחת מהמתודות של PEAR DB או אולי אראה דוגמה מורכבת יותר לשימוש ב-prepare/execute?

פרק 7

טיפול בשגיאות
7.1 מבוא

ניתן לצמצם את מספר השגיאות באפליקציה על ידי שימוש בפרקטיקות תכנות טובות; עם זאת, גורמים רבים הגורמים לשגיאות נמצאים מעבר לשליטתנו בסקריפט. הפסקות רשת, דיסקים קשיחים מלאים, כשל בחומרה, באגים ברכיבי PHP אחרים או בתוכנות שהאפליקציה מתממשת איתן – כל אלה יכולים לגרום לשגיאות שאינן נובעות מכל אשמה בקוד ה-PHP שלכם.

אם לא תעשו דבר כדי לטפל בשגיאות כאלו, התנהגות ברירת המחדל של PHP היא להציג את הודעת השגיאה למשתמש, יחד עם קישור לדף במדריך המתאר את הפונקציה שנכשלה, וכן את שם הקובץ והשורה בקוד שעוררו את השגיאה. עבור רוב השגיאות, PHP ממשיכה לרוץ לאחר הצגת הודעה זו. ראו איור 7.1.

איור 7.1 הודעת שגיאה של PHP.

הודעת שגיאה זו מיועדת למעשה לכם, המפתחים, ולא למשתמשי האתר. המשתמשים יעריכו דף המסביר במונחים פשוטים מה השתבש, ואין להם עניין בקישורים לתיעוד או במיקום הקוד שלכם בשרת.

PHP מספקת מספר אפשרויות לטיפול בשגיאות כאלה בצורה טובה יותר. לאחר שתסיימו לקרוא פרק זה, תלמדו:

- את סוגי השגיאות השונים שבהם המשתמשים שלכם עלולים להיתקל.
- אילו אפשרויות עומדות בפניכם כמפתחים בתוך PHP לטיפול בהן.
- כיצד לכתוב מטפלי שגיאות (Error handlers) משלכם.
- המרה בין מנגנוני דיווח שגיאות שונים.

7.2 סוגי שגיאות

7.2.1 שגיאות תכנות

לפעמים שגיאות מתרחשות עקב טעויות בקוד שלנו. במובנים מסוימים, אלו השגיאות שהכי קל לטפל בהן מכיוון שניתן לחשוף אותן לרוב על ידי בדיקות ישירות, פשוט על ידי ניסוי כל הפעולות שהאפליקציה מספקת. הטיפול בהן הוא פשוט עניין של תיקון הקוד.

7.2.1.1 שגיאות תחביר/ניתוח (Syntax/Parse Errors)

שגיאות תחביר ושגיאות ניתוח אחרות נתפסות כאשר הקובץ עובר קומפילציה (הידור), עוד לפני ש-PHP מתחילה להריץ אותו בכלל.

PHP

```
php?>
;"print "Hello!\n
<invalid_xml_tag>
<?
```

דוגמה זו מכילה תגית XML במקום שבו PHP מצפה למצוא קוד. הרצת קוד זה תגרום לשגיאה:

Parse error: parse error in test.php on line 4

כפי שניתן לראות, הסקריפט אפילו לא הדפיס "Hello!" לפני הצגת הודעת השגיאה, כיוון ששגיאת התחביר התגלתה בזמן הקומפילציה.

Eval 7.2.1.2

כל שגיאות התחביר נתפסות בזמן הקומפילציה, למעט שגיאות בקוד המורץ באמצעות eval(). במקרה של eval, הקוד עובר קומפילציה במהלך הרצת הסקריפט.

PHP

```
php?>
;"print "Hello!\n
;"<eval("<invalid_xml_tag
<?
```

הפעם הפלט שונה:

!Hello

Parse error: parse error in /home/ssb/test.php(4) : eval()'d code on line 1

הפעם השגיאה הוצגה בזמן הריצה, כיוון שקוד המורץ ב-eval() אינו עובר קומפילציה עד שפעולת ה-eval() עצמה מתבצעת.

Include / Require 7.2.1.3

אם הסקריפט שלכם כולל קובץ אחר שיש בו שגיאת ניתוח (Parse error), הקומפילציה תיעצר בשגיאה. קוד והצהרות שקדמו לשגיאה יעברו קומפילציה, ואילו אלו שאחריה ייזרקו. זה אומר שתקבלו קובץ שחציו עבר קומפילציה.

7.2.2 סמלים לא מוגדרים (Undefined Symbols)

בזמן הריצה, PHP עשויה להיתקל בשמות של משתנים או פונקציות שהיא אינה מכירה. בניגוד לשגיאות תחביר, שגיאות לגבי סמלים לא מוגדרים מתרחשות בזמן שהקוד רץ.

7.2.2.1 משתנים וקבועים

משתנים וקבועים לא מוגדרים אינם גורמים לקריסה דרמטית, אלא רק להתראה (Notice):

PHP

```
php?>
;(var_dump($undefined_variable
;(var_dump(UNDEFINED_CONSTANT
;"print "Still alive!\n
<?
```

הפלט יהיה התראה על משתנה לא מוגדר שיקבל ערך NULL, והתראה על קבוע לא מוגדר ש-PHP תניח שהתכוונתם למחרוזת בשם הקבוע. שימוש במשתנים לא מוגדרים אינו שגיאה טכנית ב-PHP, אלא פרקטיקת תכנות רשלנית. אנו ממליצים להשאיר את הדיווח על התראות (Notices) פעיל ולתקן את

המשתנים. כאשר לקבועים לא מוגדרים – אלו הם באגים; אל תסתמכו על כך ש-PHP הופכת אותם למחרוזות, והקפידו לשים מחרוזות במירכאות.

7.2.2.2 אינדקסים במערך

אם ניגשים לאינדקס במערך שאינו קיים (למשל `$_GET['name']` כשלא נשלח פרמטר), PHP תציג Notice: Undefined index.

7.2.2.3 פונקציות ומחלקות

בעוד ש-PHP ממשיכה לרוץ אחרי היתקלות במשתנה לא מוגדר, היא מפסיקה מיד (Abort) כאשר היא נתקלת בפונקציה או מחלקה לא מוגדרת. זוהי שגיאה קריטית (Fatal error).

יוצא דופן אחד למחלקות הוא הפונקציה `__autoload`, הנקראת כאשר PHP נתקלת במחלקה לא מוכרת, ומאפשרת לטעון אותה בזמן אמת.

7.2.3 שגיאות ניידות (Portability Errors)

7.2.3.1 הבדלים במערכות הפעלה

למרות ש-PHP רצה על פלטפורמות רבות, זה לא הופך את הקוד לאוטומטי לבלתי תלוי בפלטפורמה. יש להתחשב בתווים המפרידים בין נתיבי קבצים (סלאש מול בקסלאש), פונקציות שזמינות רק בלינוקס או רק בוינדוס, ושירותים חיצוניים.

7.2.3.2 הבדלים בהגדרות PHP

קל להסתבך כשמניחים הנחות לגבי הגדרות ב-`php.ini`. דוגמה נפוצה היא `magic_quotes_gpc`; אם היא פעילה, PHP מוסיפה לוכסנים אוטומטית לנתונים חיצוניים. הדרך הנכונה היא לבדוק את ההגדרות בזמן ריצה באמצעות `ini_get()` ולהתאים את הקוד.

רשימת הגדרות חשובות לניידות:

- **register_globals**: קובע אם משתני GET/POST יוכנסו כמשתנים גלובליים. מומלץ להימנע מכך ולהשתמש ב-`$_GET` וחבריו.
- **magic_quotes_gpc**: מוסיף לוכסנים אוטומטית. מומלץ לכבות זאת, אך קוד נייד חייב לדעת לטפל בכך בעזרת `stripslashes()`.
- **allow_url_fopen**: מאפשר לפונקציות קבצים לקרוא כתובות URL. אם זה כבוי, פעולות אלו ייכשלו.

האם תרצה שאפרט על פונקציות ספציפיות ב-Table 7.1 שמיועדות לשרתי Apache לעומת ממשקי ?CLI

טבלה 7.1: פונקציות ספציפיות ל-SAPI

פונקציה	שכבות SAPI שמגדירות אותה

apache_hooks	ApacheRequest (מחלקה)
apache, apache_hooks, apache2filter	apache_lookup_uri
apache, apache_hooks, apache2filter	apache_request_headers
apache, apache_hooks, apache2filter	apache_response_headers
apache, apache_hooks, apache2filter	apache_note
apache, apache_hooks, apache2filter	apache_setenv
apache, apache_hooks	apache_getenv
apache, apache_hooks	apachelog
apache, apache_hooks	apache_child_terminate
apache, apache_hooks	apache_exec_uri
aolserver, apache, apache_hooks, apache2filter	getallheaders
militer	smfi_setflags
militer	smfi_settimeout

militer	smfi_getsymval
militer	smfi_setreply
militer	smfi_addheader
militer	smfi_chgheader
militer	smfi_addrcpt
militer	smfi_delrcpt
militer	smfi_replacebody
apache, apache_hooks, apache2filter	virtual

7.2.3.4 התמודדות עם ניידות

שגיאות ניידות יכולות להיות קשות לאיתור מכיוון שהן דורשות בדיקה יסודית של הקוד בתצורות שונות ועל מערכות שונות. עם זאת, בדיקות נאותות וסקירות קוד הן הדרכים הטובות ביותר למציאת בעיות ניידות.

כמובן, אם אתם כותבים ומפיצים את כל הקוד שלכם על אותה פלטפורמה עם תצורה אחידה, ייתכן שלעולם לא תיתקלו בבעיות ניידות. מודעות לנושאי ניידות היא דבר חיובי בכל מקרה; היא מאפשרת לכם לכתוב קוד טוב יותר, עמיד יותר וניתן לשימוש חוזר.

תיקון שגיאות ניידות עשוי להיות פשוט, כמו בדיקת הגדרות ה-ini בדוגמה הקודמת של `magic_quotes_gpc`, אך הוא יכול להיות גם מורכב יותר. ייתכן שתצטרכו לנתח פלט של פקודה בצורה שונה עבור מערכות הפעלה שונות, או לספק מימוש חלופי ב-PHP עבור רכיב שזמין רק בחלק מהפלטפורמות.

במקרים מסוימים, מה שאתם מנסים לעשות פשוט אינו אפשרי בדרך ניידת. באופן כללי, הגישה הטובה ביותר לבעיות ניידות היא הסתרת פרטי מערכת ההפעלה או ה-SAPI בתוך שכבת קוד המפשיטה (Abstracting) את הבעיה. דוגמה אחת להפשטה כזו היא מחלקת ה-**System** מתוך **PEAR**, המספקת מימשי PHP לכמה פקודות UNIX נפוצות ופעולות אחרות שהן ספציפיות למערכת ההפעלה.

7.2.3.5 כלי ניידות

- מחקלת PEAR: System מחלקה זו זמינה כחלק מהתקנת הבסיס של PEAR.
- מחקלת PEAR: OS_Guess מחלקה זו משתמשת בפונקציה php_uname() כדי לקבוע על איזו מערכת הפעלה היא רצה. היא גם מספקת דרכים להכללה והשוואה של חתימות מערכות הפעלה:

PHP

```
php?>
;require_once "OS/Guess.php";
os = new OS_Guess$
;print "OS signature: " . $os->getSignature() . "\n"
} ("if ($os->matchSignature("linux-*-i386
;print "Linux running on an Intel x86 CPU\n"
{
<?
```

פלט לדוגמה:

OS signature: linux-2.4-i386-glibc2.1

Linux running on an Intel x86 CPU

7.2.4 שגיאות זמן ריצה (Runtime Errors)

ברגע שהקוד פועל, שגיאות זמן ריצה שאינן קריטיות (Non-fatal) הן סוג השגיאה הנפוץ ביותר ב-PHP. המונח "זמן ריצה" מתייחס לשגיאות המתרחשות במהלך ביצוע הקוד, שלרוב אינן טעויות תכנות אלא נגרמות מגורמים מחוץ ל-PHP עצמו, כגון פעולות דיסק, רשת או קריאות למסד נתונים.

ל-PHP יש מנגנון דיווח שגיאות המשמש לכל השגיאות שמתעוררות בתוך PHP, בין אם בזמן הידור הסקריפט או בעת ביצוע פונקציה מובנית. ניתן להשתמש במנגנון זה גם מתוך סקריפט, אם כי קיימות דרכים חזקות יותר לדיווח על שגיאות (כגון חריגות - Exceptions).

דוגמאות לשגיאות זמן ריצה מתרחשות כאשר `fopen()` נכשל כי קובץ חסר, כאשר `mysql_connect()` נכשל בגלל שם משתמש שגוי, או אם ניסיתם להזין שורה לטבלה מבלי לספק ערך לעמודה שמוגדרת כ-Not-null.

7.2.5 שגיאות PHP

מנגנון השגיאות ב-PHP משמש את כל הפונקציות המובנות. כברירת מחדל, מנגנון פשוט זה מדפיס הודעת שגיאה עם שם הקובץ ומספר השורה, ויוצא מהתוכנית.

7.2.5.1 רמות שגיאה (Error Levels)

שגיאות PHP מחולקות לרמות, החל מהתראות (Notices) ועד שגיאות קריטיות (Fatal errors). רמת השגיאה מעידה על חומרתה.

- **E_ERROR**: שגיאה קריטית שלא ניתן להתאושש ממנה (למשל חריגת זיכרון).
- **E_WARNING**: הסוג הנפוץ ביותר. מעיד שמשהו שניסיתם לעשות השתבש (למשל חסר פרמטר בפונקציה או חלוקה באפס).
- **E_PARSE**: שגיאת ניתוח בזמן ההידור, הגורמת ל-PHP לעצור לפני הריצה.
- **E_STRICT**: נועד להקל על המעבר מ-PHP 4 ל-PHP 5 על ידי הצעות לשיפור הקוד.
- **E_NOTICE**: מציין שהקוד עושה משהו שעלול להיות לא מכוון (כמו קריאת משתנה לא מוגדר).
- **E_CORE_ERROR / E_COMPILE_ERROR**: שגיאות פנימיות בזמן העלייה או ההידור של PHP.
- **E_USER_ERROR / WARNING / NOTICE**: שגיאות המוגדרות על ידי המשתמש/הסקריפט ולא על ידי PHP עצמו.

7.2.5.2 דיווח שגיאות (Error Reporting)

מספר הגדרות ב-php.ini שולטות אילו שגיאות יוצגו וכיצד:

- **error_reporting**: קובע אילו רמות שגיאה ידווחו (למשל E_ALL).
- **display_errors**: שולט האם להציג שגיאות כחלק מהפלט בדפדפן.
- **log_errors**: שולט האם לתעד שגיאות בקובץ לוג. הכתובת נקבעת ב-error_log.
- **html_errors**: קובע האם להוסיף פורמט HTML להודעות השגיאה.

7.2.5.3 מטפלי שגיאות מותאמים אישית

במקום ש-PHP תדפיס את השגיאה, ניתן לרשום פונקציה שתופעל בכל שגיאה. כך תוכלו לתעד שגיאות למסד נתונים או לשלוח התראות בדוא"ל.

7.2.5.4 השתקת שגיאות (Silencing Errors)

ניתן להשתיק שגיאות באמצעות התו @ לפני ביטוי. זה יגרום לכך שהודעת השגיאה לא תוצג, אך המנגנון המובנה עדיין יופעל. יש להיזהר בשימוש בזה עם מטפלי שגיאות מותאמים אישית.

7.3 שגיאות PEAR

ל-PEAR מנגנון דיווח משלו המבוסס על העיקרון ששגיאות הן אובייקטים שניתן להעביר כערכים.

PHP

```
php?>
require_once "DB.php";
$dbh = DB::connect("mysql://test@localhost/world$");
if (PEAR::isError($dbh)) {
    die("Connection failed (" . $dbh->getMessage() . ")");
}
print "DB::connect ok!\n";
<?>
```

בדוגמה זו, אם החיבור נכשל, המתודה מחזירה אובייקט שגיאת PEAR. אנו בודקים זאת בעזרת `PEAR::isError()` ושולפים את ההודעה בעזרת `getMessage()`.

PEAR::isError 7.3.0.2())

מתודה סטטית המחזירה אמת אם המשתנה שהועבר לה הוא אובייקט מסוג PEAR_Error או תת-מחלקה שלו.

7.3.0.3 העלאת שגיאות (Raising Errors)

בטרמינולוגיה של PEAR, שגיאות "מועלות" (Raised), אם כי הדרך הקלה ביותר להעלות שגיאת PEAR היא להחזיר את ערך החזרה ממתודה שנקראת `throwError`. זאת פשוט מפני ש-`throwError` היא גרסה מפושטת של המתודה המקורית PEAR `raiseError`. משתמשת במונח "העלאה" כדי למנוע בלבול עם חריגות (Exceptions) של PHP, אשר אותן "זורקים" (Thrown).

העלות היחסית של העלאת שגיאת PEAR בהשוואה לעירור שגיאת PHP היא גבוהה, כיוון שהיא כרוכה ביצירת אובייקט ומספר קריאות לפונקציות. המשמעות היא שעליכם להשתמש בשגיאות PEAR בזירות – שמרו אותן לכשלים שלא אמורים לקרות בדרך כלל. העדיפו להשתמש בערך חזרה בוליאני פשוט למקרים הרגילים. עצה זו תקפה גם לגבי שימוש בחריגות ב-PHP, כמו גם ב-C++ ,Java או שפות אחרות.

כאשר אתם משתמשים בחבילות PEAR בקוד שלכם, עליכם להתמודד עם שגיאות שהועלו על ידי החבילה. ניתן לעשות זאת באחת משתי דרכים: כתלות בשאלה האם אתם נמצאים בהקשר של אובייקט (Object context), והאם המחלקה הנוכחית שלכם יורשת ממחלקת ה-PEAR.

אם הקוד שלכם אינו רץ בהקשר של אובייקט, למשל מתוך הטווח הגלובלי (Global scope), בתוך פונקציה רגילה או במתודה סטטית, עליכם לקרוא למתודה הסטטית `PEAR::throwError()`:

```
PHP
php?>

require_once 'PEAR.php';

if (PEAR::isError($e = lucky)) {
    die($e->getMessage()). "\n";
}

print "You were lucky, this time.\n";

function lucky()
{
    if (rand(0, 1) == 0)
    {
        return PEAR::throwError('tough luck');
    }
}
```

כאשר שגיאות מועלות באמצעות קריאות למתודה סטטית, מוחלות הגדרות ברירת המחדל שנקבעו עם `PEAR::setErrorHandler()`. הדרך השנייה להעלאת שגיאות היא כאשר המחלקה שלכם יורשת מ-PEAR, והקוד שלכם מבוצע בהקשר של אובייקט:

PHP

`php?>`

```
;'require_once 'PEAR.php
```

```
class Luck extends PEAR
```

```
}
```

```
} ()function testLuck
```

```
} (if (rand(0, 1) == 0
```

```
;) (!return $this->throwError('tough luck
```

```
{
```

```
;"!return "lucky
```

```
{
```

```
{
```

```
;"luck = new Luck$
```

```
;"()test = $luck->testLuck$
```

```
} ((if (PEAR::isError($test
```

```
;"die($test->getMessage()) . "\n
```

```
{
```

```
;"print "$test\n
```

`<?`

כאשר `throwError()` נקראת בהקשר של אובייקט, הגדרות ברירת המחדל שנקבעו באותו אובייקט עם `(object->setErrorHandler()` מוחלות תחילה. אם לא נקבעו ברירות מחדל לאובייקט, יחולו ברירות המחדל הגלובליות.

PEAR::throwError() 7.3.0.4

`[object PEAR::throwError([string message], [int code], [string userinfo]`

מתודה זו מעלה שגיאת PEAR תוך החלת הגדרות ברירת המחדל לטיפול בשגיאות. אילו ברירות מחדל מוחלות בפועל תלוי באופן הקריאה למתודה: בסטטיות (גלובלי) או מתוך אובייקט (מקומי לאובייקט).

אם אינכם בקיאים בסמנטיקה של `this$` ב-PHP, אתם עשויים להיתקל בהפתעות. אם תקראו למתודה סטטית מתוך אובייקט (שבו ל-`this$` יש ערך), הערך של `this$` יהיה מוגדר גם בתוך המתודה שנקראה סטטית. המשמעות היא שאם תקראו ל-`PEAR::throwError()` מתוך אובייקט, `this$` יתייחס לאותו אובייקט והגדרותיו המקומיות יוחלו לפני הגלובליות.

PEAR::raiseError() 7.3.0.5

`object PEAR::raiseError([string message], [int code], [int mode], [mixed options],
([string userinfo], [string error_class], [bool skipmsg]`

מתודה זו שקולה ל-`throwError()` אך עם פרמטרים נוספים. בדרך כלל לא תזדקקו לכל האפשרויות הללו, אך הן עשויות להועיל אם אתם בונים מערכת שגיאות משלכם המבוססת על PEAR.

- **error_class**: המחלקה שתשמש לאובייקט השגיאה (ברירת מחדל "PEAR_Error").
- **skipmsg**: פרמטר המורה להתעלם מהודעת השגיאה, שימושי למנגנונים המבוססים על קודי שגיאה בלבד.

PEAR_Error מחלקת 7.3.1

מחלקה זו היא בסיס דיווח השגיאות של PEAR. ניתן להרחיב אותה לצרכים ספציפיים, ו-`PEAR::isError()` עדיין תזהה אותה.

7.3.1.1 הקונסטרקטור של PEAR_Error

בדרך כלל לא יוצרים שגיאות PEAR עם `new`, אלא דרך `throwError()` או `raiseError()`. הפרמטרים כוללים:

- **message**: הודעת השגיאה.
- **code**: קוד שגיאה (מספר שלם). מומלץ תמיד לציין קוד שגיאה לטיפול נקי יותר.
- **mode**: מצב השגיאה (למשל: RETURN, PRINT, DIE, TRIGGER, CALLBACK).
- **options**: פרמטר המשתנה לפי המצב (למשל, פורמט `printf` או שם פונקציית CALLBACK).
- **userinfo**: מידע נוסף (כמו שאילתת ה-SQL שנכשלה).

מתודות נבחרות ב-PEAR_Error:

- **getMessage() / getCode()**: החזרת ההודעה או הקוד.
- **getUserInfo()**: החזרת המידע הנוסף.
- **getBacktrace()**: החזרת היסטוריית הקריאות (Stack trace) מרגע יצירת השגיאה.

7.3.2 טיפול בשגיאות PEAR

כברירת מחדל, שגיאות PEAR רק מחזירות אובייקט. ניתן לקבוע "מצב שגיאה" שיחול על כל השגיאות הבאות:

```

php?>
require_once 'DB.php';
PEAR::setErrorHandler(PEAR_ERROR_DIE, "Aborting: %s\n");
$dbh = DB::connect('mysql://test@localhost/test$');
print "DB::connect ok!\n";
<?

```

בדוגמה זו השתמשנו ב-`PEAR_ERROR_DIE`, מה שגורם לתוכנית להדפיס את השגיאה ולהיעצר מיד במקרה של כישלון בחיבור.

PEAR::setErrorHandler() 7.3.2.1

קובע את ברירות המחדל לטיפול בשגיאות. קריאה סטטית קובעת ברירות מחדל גלובליות, וקריאה מתוך אובייקט (כמו `$dbh->setErrorHandler()`) קובעת אותן עבור אותו אובייקט בלבד.

7.3.3 מצבי שגיאה (Error Modes) של PEAR

1. `PEAR_ERROR_RETURN`: לא עושה דבר מלבד החזרת אובייקט השגיאה (ברירת המחדל).
2. `PEAR_ERROR_PRINT`: מדפיס את ההודעה אוטומטית.
3. `PEAR_ERROR_DIE`: מדפיס את ההודעה ויוצא מהתוכנית.
4. `PEAR_ERROR_TRIGGER`: מעביר את ההודעה לפונקציית `trigger_error()` המובנית של PHP.
5. `PEAR_ERROR_CALLBACK`: מפעיל פונקציה חיצונית שהוגדרה מראש.

7.3.4 טיפול אלגנטי (Graceful Handling)

PEAR::pushErrorHandler() 7.3.4.1

דוחף מצב טיפול בשגיאות חדש לראש "מחסנית" המטפלים. המצב הזה יהיה פעיל עד שתיקרא המתודה `popErrorHandler()`. זה שימושי לשינוי זמני של התנהגות השגיאות (למשל, מכיבוי התוכנית להחזרת ערך בלבד) בזמן ביצוע פעולה ספציפית.

PEAR::expectError() 7.3.4.3

מאפשר להגדיר קודי שגיאה או הודעות ספציפיות שאנו "מצפים" להם. עבור שגיאות אלו, מצב הטיפול ייכפה ל-`PEAR_ERROR_RETURN` באופן זמני, מה שמאפשר לטפל בהן ידנית בקוד מבלי להפעיל את המטפל הגלובלי (כמו `DIE`).

7.4 חריגות (EXCEPTIONS)

7.4.1 מהן חריגות?

חריגות הן מנגנון שגיאות מובנה ברמה גבוהה שהוצג ב-PHP 5. חריגות הן אובייקטים שניתן "לזרוק" (Throw). אם משהו מוכן "לתפוס" (Catch) את החריגה, היא מטופלת; אם לא, PHP עוצרת עם שגיאה קריטית ומציגה Stack trace.

7.4.2 throw, catch, try

שלושה מבני שפה משמשים בחריגות:

1. **try**: בלוק קוד שבו PHP "מחפשת" חריגות.
2. **catch**: בלוק קוד המבוצע אם נזרקה חריגה מסוג מסוים בתוך ה-try.
3. **throw**: הצהרה המשמשת ליצירת וזריקת חריגה חדשה.

PHP

} try

```
;$fp = @fopen($filename, "r$
```

```
} ((if (!is_resource($fp
```

```
;$throw new FileException("could not read '$filename
```

```
{
```

```
} (catch (FileException $e {
```

```
;die($e->getMessage
```

```
{
```

ניתן להגדיר תת-מחלקות של Exception כדי להפריד בין סוגי שגיאות שונים (למשל IO_Exception לעומת XML_Exception). אם חריגה אינה נתפסת באף בלוק catch, היא עוברת לפונקציה הקוראת. אם היא לא נתפסת כלל, ניתן להגדיר מטפל סופי באמצעות set_exception_handler().

7.5 סיכום

בפרק זה למדתם על סוגי השגיאות השונים ש-PHP ו-PEAR יכולות לחולל ולנהל. למדתם כיצד להתאים אישית את הטיפול בשגיאות באמצעות הקובץ php.ini, לכתוב מטפלי שגיאות משלכם ולהמיר שגיאות PHP לשגיאות PEAR או לחריגות (Exceptions).

למדתם על הבעיות שעלולות להיגרם כתוצאה מהבדלים בין ממשקי השרת (מודולי SAPI) ומערכות הפעלה שונות, ועל כמה דרכים להתמודדות עם סוגיית הניידות (Portability).

לבסוף, למדתם כיצד להשתמש בצורה הטובה ביותר בחריגות ב-PHP ואת הפרטים הספציפיים לגבי שימוש בחריגות עם PEAR.

בזמן כתיבת שורות אלו, קהילת PEAR עדיין מגבשת את הדרך הטובה ביותר להציג ולהשתמש בחריגות בתוך PEAR, ולכן השימוש בחריגות עם PEAR הושמט במכוון ממהדורה זו של הספר. עקבו אחר עדכונים באתר האינטרנט של הספר בכתובת <http://php5powerprogramming.com>!

פרק 8

XML עם PHP 5

8.1 מבוא

XML צוברת תאוצה כשפה אוניברסלית לתקשורת בין פלטפורמות; יש המכנים אותה "מהפכת האינטרנט החדשה". לעיתים משתמשים ב-XML כמסד נתונים לאחסון מסמכים, אך אחסון נתונים מעולם לא היה מטרתה העיקרית. היא פותחה כדי להעביר מידע ממערכת אחת לאחרת בפורמט משותף.

XML היא שפת תגיות (Tagged language). הנתונים עצמם מוכלים בתוך אלמנטים מובנים ומתוייגים במסמך. יש לנתח (Parse) את מסמך ה-XML כדי לחלץ את המידע. לעיתים קרובות, יש להמיר את המידע לפורמט אחר. בפרק זה נתמקד בשימוש ב-PHP לקריאה ושינוי (Transformation) של מסמכי XML, ובשימוש ב-XML כפרוטוקול תקשורת מול שירותים מרוחקים.

לאחר סיום פרק זה, תלמדו:

- את המבנה של מסמך XML.
- את הטרימינולוגיה הדרושה לעבודה עם מסמכי XML.
- כיצד לנתח קובץ XML באמצעות שתי השיטות המרכזיות: SAX ו-DOM.
- כיצד לנתח קובץ XML פשוט בדרך קלה יותר: הרחבת SimpleXML של PHP.
- כיצד להשתמש בחבילות PEAR שימושיות עבור XML.
- כיצד להמיר מסמך XML לפורמט אחר באמצעות XSLT.
- כיצד לשתף מידע בין מערכות באמצעות XML.

8.2 אוצר מילים

בעבודה עם מסמכי XML, תיתקלו במספר מונחים שעשויים להיות לא מוכרים. הדוגמה הבאה מציגה מסמך XML שהוא מסמך XHTML:

```
XML
<? "xml version="1.0" encoding="ISO-8859-1"?>
DOCTYPE html!>
"PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN
<"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
<"html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title>XML Example</title>
    </head>
    <"body background="bg.png"
    <p>
        .<Moved to <a href="http://example.org/">example.org</a>
    </ br>
```

```
foo & bar
<p/>
<body/>
<html/>
```

- **הצהרת XML:** השורה הראשונה מציינת את גרסת ה-XML וקידוד הקובץ. שימו לב שהיא מתחילה ב-`<?>`. זה עלול ליצור בעיה אם הגדרת ה-`short_open_tag` ב-PHP פעילה, שכן PHP תחשוב שמדובר בתחילת קוד PHP. מומלץ לכבות הגדרה זו ב-`php.ini`.
- **הצהרת DOCTYPE:** מציינת את אלמנט השורש (html) ומפנה לקובץ DTD (הגדרת סוג מסמך). קובץ DTD מתאר את מבנה המסמך.
- **Well-formed vs. Valid:** מסמך **Well-formed** (בנוי כהלכה) הוא כזה העומד בחוקי התחביר של XML. מסמך **Valid** (תקף) הוא כזה שגם תואם את הכללים הספציפיים שהוגדרו ב-DTD שלו.
- **אלמנט שורש (Root Node):** האלמנט העוטף את כל המסמך (בדוגמה שלנו: `<html>`).
- **מרחב שמות (Namespace):** מוגדר ע"י `xmlns`. מאפשר לערבב סוגי מסמכים שונים (כמו MathML בתוך XHTML) ללא התנגשויות.
- **תכונות (Attributes):** מידע נוסף בתוך התגית (כמו `background="bg.png"`). ב-XHTML חובה שלכל תכונה יהיה ערך והוא חייב להיות בתוך מירכאות.
- **ישויות (Entities):** תווים מיוחדים כמו `>` או `&` חייבים להיכתב כ-`<lt>` ו-`&`; בהתאמה, כדי שהמנתח לא יחשוב שמדובר בתגיות.

8.3 ניתוח XML (Parsing)

קיימות שתי טכניקות עיקריות לניתוח XML ב-PHP:

1. **SAX (Simple API for XML):** המנתח עובר על המסמך ויורה "אירועים" (Events) עבור כל תגית התחלה, סוף או תוכן. אתם מחליטים איך לטפל בכל אירוע. חסכוני בזיכרון.
2. **DOM (Document Object Model):** הקובץ כולו נטען לזיכרון כעץ של אובייקטים שניתן לנווט ביניהם. קל יותר לשימוש אך צורך זיכרון רב.

SAX 8.3.1

ב-SAX, עלינו להגדיר "מטפלים" (Handlers) לפונקציות שלנו.

- `xml_parser_create`: יצירת אובייקט המנתח.
- `xml_set_element_handler`: הגדרת הפונקציות שירוצו בתחילת ובסוף תגית.
- `xml_set_character_data_handler`: הגדרת הפונקציה שתטפל בטקסט שבתוך התגיות.

בדוגמה שבספר, המנתח מציג את מבנה התגיות עם הזחה (Indentation) לפי עומק התגית. מכיוון ש-SAX עשוי לקרוא את הטקסט בחלקים, נהוג להשתמש ב"באפר" (Buffer) כדי לאסוף את כל הטקסט ורק אז להציגו בצורה נקייה.

DOM 8.3.2

ניתוח קובץ פשוט עם SAX דורש עבודה רבה. שיטת ה-DOM קלה בהרבה, אך המחיר הוא בצריכת הזיכרון. בעוד שזה לא מורגש בקבצים קטנים, ניתוח קובץ של 20MB בשיטת DOM עשוי להיות כבד מאוד. DOM יוצר עץ בזיכרון המייצג את הקובץ.

כפי שניתן לראות באיור 8.1, עץ ה-DOM מייצג את המבנה ההיררכי של הקובץ. אנו יכולים להציג את כל התוכן ללא התגיות על ידי מעבר על עץ האובייקטים. בדוגמה זו אנו עושים זאת על ידי מעבר רקורסיבי על כל הילדים של הצמתים:

PHP

php?>

```
;(dom = new DomDocument$
;('dom->load('test2.xml$
;root = $dom->documentElement$

;(process_children($root

(function process_children($node
    }

;children = $node->childNodes$

} (foreach ($children as $elem
} (if ($elem->nodeType == XML_TEXT_NODE
    } (((if (strlen(trim($elem->nodeValue
        ;"echo trim($elem->nodeValue)."n
    {
} (else if ($elem->nodeType == XML_ELEMENT_NODE {
    ;(process_children($elem
    {
    {
    {
<?
```

הפלט יהיה:

XML Example

Moved to

example.org

.

foo & bar

דוגמה זו מציגה עיבוד DOM בסיסי מאוד. בשורה 4 אנו שולפים את אלמנט השורש של המסמך. עבור כל אלמנט שאנו פוגשים, אנו קוראים ל-`process_children()` אשר עוברת על רשימת צמתי הבן. אם הצומת הוא צומת טקסט, אנו מדפיסים את ערכו, ואם הוא אלמנט, אנו קוראים לפונקציה באופן רקורסיבי. הרחבת ה-DOM חזקה בהרבה ממה שמוצג כאן ומממשת כמעט את כל הפונקציונליות של מפרט DOM2.

הדוגמה הבאה משתמשת במתודה `getAttributeNode()` כדי להחזיר את התכונה `background` של תגית `body`:

PHP

`php?>`

```
;$dom = new DomDocument$
```

```
;$dom->load('test2.xml'$
```

```
;$root = $dom->documentElement$
```

```
;(process_children($root
```

```
(function process_children($node
```

```
}
```

```
;$children = $node->childNodes$
```

```
} (foreach ($children as $elem
```

```
} (if ($elem->nodeType == XML_ELEMENT_NODE
```

```
} ('if ($elem->nodeName == 'body
```

```
;"echo $elem->getAttributeNode('background')->value . "\n
```

```
{
```

```
;(process_children($elem
{
{
{
<?
```

אנו עדיין צריכים לחפש רקורסיבית בעץ, אך כיוון שאנו מכירים את מבנה המסמך, ניתן לפשט זאת:

PHP

php?>

```
;(dom = new DomDocument$
;('dom->load('test2.xml$
;(body = $dom->documentElement->getElementsByTagName('body')->item(0$
;"echo $body->getAttributeNode('background')->value . "\n
<?
```

שורה 4 היא העיקרית: אנו מבקשים את כל אלמנטי הבן עם שם התגית `body`, לוקחים את הפריט הראשון ברשימה, ובשורה 5 שולפים ומציגים את ערך התכונה `background`.

8.3.2.1 שימוש ב-XPath

באמצעות XPath נוכל לפשט את הדוגמה עוד יותר. XPath היא שפת שאילתות למסמכי XML (בדומה ל-SQL למסדי נתונים):

PHP

php?>

```
;(dom = new DomDocument$
;('dom->load('test2.xml$
;(xpath = new DomXPath($dom$
;(nodes = $xpath->query("[local-name()='body']", $dom->documentElement$
;"echo $nodes->item(0)->getAttributeNode('background')->value . "\n
<?
```

8.3.2.2 יצירת עץ DOM

הרחבת ה-DOM יכולה גם ליצור מסמך XML מאפס על ידי בניית עץ אובייקטים ושמירתו:

PHP

`<?php`

```
;$dom = new DomDocument$
;$html = $dom->createElement('html'$
;$html->setAttribute('xmlns', 'http://www.w3.org/1999/xhtml'$
;$html->setAttribute('xml:lang', 'en'$
;$html->setAttribute('lang', 'en'$
;($dom->appendChild($html)$

;$head = $dom->createElement('head'$
;($html->appendChild($head)$
;$title = $dom->createElement('title'$
;($title->appendChild($dom->createTextNode('XML Example'$
;($head->appendChild($title)$

/* יצירת גוף המסמך ותוכן נוסף באותה שיטה... */

;($dom->saveXML
```

`<?>`

SimpleXML 8.4

הרחבת SimpleXML היא הדרך הקלה ביותר לעבוד עם XML ב-PHP 5. אין צורך לזכור API מסובך; פשוט ניגשים לנתונים כאל מבנה נתונים. ארבעת הכללים הפשוטים הם:

1. **מאפיינים (Properties)** מייצגים איטרטורים של אלמנטים.
2. **אינדקסים מספריים** מייצגים אלמנטים.

3. אינדקסים שאינם מספריים מייצגים תכונות (Attributes).
4. המרה למחרוזת מאפשרת גישה לנתוני טקסט (TEXT).

8.4.1 יצירת אובייקט SimpleXML

ניתן ליצור אובייקט בשלוש דרכים: מקובץ (simplexml_load_file), ממחרוזת (simplexml_load_string), או מאובייקט DOM (simplexml_load_dom).

8.4.2 דפדוף באובייקט SimpleXML

- לולאה על כל תגיות ה-p בתוך ה-p (\$sx->body->p as \$p) (body: foreach).
- גישה לתגית ה-p השנייה: \$sx->body->p[1].
- גישה לתכונה "background": \$sx->body["background"].
- הצגת הטקסט של תגית: [echo \$sx->body->p[1]].

כדי לכלול צמתי בן בטקסט, ניתן להשתמש ב-asXML() בשילוב עם strip_tags(). למעבר על כל ילדי צומת משתמשים ב-children(), ולמעבר על תכונות משתמשים ב-attributes().

PEAR 8.5

PEAR מכילה מחלקות לטיפול ב-XML כאשר הרחבות ה-DOM אינן זמינות או למשימות ספציפיות:

- **XML_Tree**: שימושי לבניית מסמכי XML בצורה מהירה ופשוטה יותר מ-DOM.
- **XML_RSS**: מחלקה ייעודית לניתוח קבצי RSS (עדכוני חדשות).

XML_RSS 8.5.2

RSS הוא אוצר מילים של XML לתיאור פריטי חדשות. מחלקת XML_RSS מאפשרת לקחת קובץ RSS (כמו זה של php.net), לנתח אותו ולשלוף בקלות את הכותרות, הקישורים והתיאורים של הכתבות באמצעות המתודות getItems() ו-getChannelInfo().

8.6 המרת XML

ייתכן שתמצאו להמיר מסמך XML למשהו אחר, כגון מסמך HTML, קובץ טקסט או קובץ XML בפורמט שונה. השיטה הסטנדרטית להמרת מסמך XML לפורמט אחר היא באמצעות **XSLT** (שפת גיליונות סגנון מורחבת להתמרות). XSLT היא מורכבת, ולכן לא נעבור על כל פרטי אוצר המילים של ה-XSLT. אם תרצו ללמוד עוד על XSLT, תוכלו למצוא את המפרט המלא בכתובת <http://www.w3.org/TR/xslt>.

אם XSLT לא עושה את מה שאתם רוצים, ייתכן שתצטרכו לפנות לפתרונות אחרים. מחלקת ה-PEAR XML_Transformer היא פתרון אפשרי אחד. באמצעות XML_Transformer תוכלו לבצע התמרות XML עם PHP ללא צורך ב-XSLT או בספריות חיצוניות.

XSLT 8.6.1

כדי להשתמש בפונקציות ה-XSLT ב-PHP, עליכם להתקין את הגרסה העדכנית ביותר של ספריית libxslt, המממשת את הפונקציות הדרושות להתמרות.

אם אתם משתמשים בווינדוס, תוכלו להעתיק את הקובץ libxslt.dll מספריית ה-dlls של הפצת ה-PHP למיקום בנתיב המערכת שלכם. הפעלת ההרחבה ביוניקס (UNIX) מתבצעת על ידי הוספת --with-xsl לשורת ה-configure והידור מחדש. משתמשי ווינדוס יכולים להסיר את ה-comment מהשורה extension=php_xsl.dll בקובץ ה-php.ini.

כפי שהוסבר קודם, ניתן להשתמש ב-XSLT כדי להתמיר את מסמכי ה-XML שלכם לפורמט אחר. אנחנו נתמיר קובץ הדומה לקובץ ה-RSS שלנו לקובץ X(HT)ML על ידי החלת גיליונות סגנון על מסמך ה-XML.

גיליונות סגנון (Stylesheets) משמשים לכל ההתמרות המבוצעות עם XSLT כדי למפות את האלמנטים בקובץ ה-XML המקורי לתבנית (template) עבור כל אלמנט. החלק הראשון של גיליון הסגנון (XSL stylesheet) מכיל אפשרויות לקלט ופלט. אנחנו רוצים להוציא את התוצאה כמסמך HTML עם mime-type של 'text/html' בקידוד ISO-8859-1. מרחב השמות (namespace) להצהרת ה-XSL מוגדר כ-xsl, מה שאומר שלכל אלמנט הקשור ל-XSL יש את הקידומת: xsl: לפני שם התגית (למשל, xsl:output).

התבניות עוקבות אחר חלק המבוא שהוצג קודם. תכונת ה-match של אלמנט ה-xsl:template משמשת לבחירת אלמנטים במסמך. בתבנית הראשונה, כל אלמנטי ה-"rdf" במסמך יותאמו. מכיוון שזהו אלמנט השורש של המסמך שלנו, התבנית מוחלת פעם אחת בלבד. כאשר אלמנט מותאם על ידי תבנית, תוכן ה-xsl:template מועתק למסמך הפלט, למעט אלמנטים השייכים למרחב השמות XSL שיש להם משמעות מיוחדת.

התגית `<xsl:value-of>` "מחזירה" את הערך של אלמנט או תכונה המצוינים בתכונת ה-select. בתבנית המוצגת כאן, התוכן של הילד title של אלמנט ה-channel מוכנס לתוך תגית ה-title של ה-HTML.

כעת, כשיש לנו גם את גיליון הסגנון וגם את קובץ המקור של ה-XML, נוכל להשתמש ב-PHP כדי להחיל את גיליון הסגנון על מסמך ה-XML. נשתמש בפונקציות ה-XSLT עם הקבצים `php.net.xsl` ו-`php.net-stripped.rss`, ונדפיס את הפלט למסך:

```
PHP
<?php?>

$dom = new domDocument();
$dom->load("php.net.xsl");
$proc = new xsltprocessor();
$xml = $proc->importStylesheet($dom);
$xml = new domDocument();
$xml->load('php.net-stripped.rss');
$string = $proc->transformToXml($xml);
echo $string;
```

טיפ: ניתן להשתמש באותו גיליון סגנון XSLT שנטען מ-`()dom->load$` להתמרה של מספר מסמכי XML. הדבר חוסך את התקורה הכרוכה בניתוח (parsing) של גיליון הסגנון.

בנוסף למתודה `transformToXml()`, זמינות שתי פונקציות עיבוד XSLT נוספות להמרת מסמכים: `transformToDoc()` ו-`transformToUri()`. הראשונה מוציאה אובייקט `DomDocument` שניתן להמשיך לעבד אותו, והשנייה מרנדרת לכתובת URI המועברת כפרמטר השני.

8.7 תקשורת באמצעות XML

יישומים מתקשרים כיום דרך האינטרנט במספר דרכים. TCP/IP ו-UDP/IP משמשים כפרוטוקולי תחבורה ברמה נמוכה. תקשורת בין מערכות היא קשה מכיוון שמערכות מאחסנות נתונים בזיכרון בשיטות שונות (למשל סדר בתים Little Endian לעומת Big Endian). פתרון אחד הוא **RPC** (קריאה לפרוצדורה מרוחקת), אך הוא אינו קל לשימוש. XML הוא לעיתים קרובות הפתרון הטוב ביותר, שכן הוא מאפשר ליישומים במערכות שונות לתקשר באמצעות פורמט סטנדרטי מבוסס **XML-RPC**. ASCII ו-**SOAP** הם שניהם פרוטוקולים מבוססי XML.

8.7.1 XML-RPC

נתחיל בדרך הפשוטה ביותר לתקשורת: XML-RPC.

8.7.1.1 הודעות: XML-RPC הוא פרוטוקול של בקשה-תגובה (request-response). עבור כל בקשה לשרת, מוחזרת תגובה. הבקשה והתגובה מקודדות כ-XML. המפרט מגדיר מספר סוגים סקלאריים (ראו טבלה 8.1).

טבלה 8.1: סוגי נתונים ב-XML-RPC

| סוג XML-RPC | תיאור |

| ---: | ---: |

| `<int>` או `<i4>` | מספר שלם חתום (4 בתים) |

| `<0>` | `<boolean>` (שקר) או 1 (אמת) |

| `<string>` | מחרוזת ASCII |

| `<double>` | מספר נקודה צפה בדיוק כפול |

| `<dateTime.iso8601>` | תאריך/שעה |

| `<base64>` | נתונים בינאריים בקידוד Base64 |

זמינים שני סוגי נתונים מורכבים: `<array>` למערכים שאינם אסוציאטיביים, ו-`<struct>` למערכים אסוציאטיביים.

8.7.1.2 בקשה (Request): בקשות ב-XML-RPC הן בקשות POST רגילות לשרת HTTP עם כמה תוספות מיוחדות. ה-`Content-Type` הוא תמיד `text/xml`. גוף ההודעה מורכב ממסמך XML הכולל תגית `<methodCall>` המכילה את שם הפונקציה (`<methodName>`) ופרמטרים (`<params>`).

8.7.1.3 תגובה (Response): כאשר קריאת הפונקציה מצליחה, מוחזרת תגובה מקודדת ב-XML תחת התגית `<methodResponse>`. היא תמיד תכיל אלמנט `<params>` עם ערך חזרה יחיד (שיכול להיות מערך).

8.7.1.4 שגיאה (Fault): כאשר משהו לא עובד כמצופה, מוחזר אלמנט `<fault>` במקום `<params>`. הוא מכיל מבנה עם `faultCode` (קוד שגיאה) ו-`faultString` (תיאור השגיאה).

8.7.1.5 הלקוח (The Client)

לכתיבת לקוח נשתמש במחלקת ה-XML_RPC_PEAR. אנו בונים אובייקט XML_RPC_Message עם שם הפונקציה והפרמטרים, ושולחים אותו באמצעות אובייקט XML_RPC_Client.

ניתן להשתמש ב-XML_RPC_encode() כדי להמיר אוטומטית משתנה PHP לסוג ה-XML-RPC המתאים, וב-XML_RPC_decode() לפעולה ההפוכה.

8.7.1.6 רטרופקטיבה (Retrospection): זוהי היכולת לשאול את השרת אילו פונקציות הוא מייצא. הפונקציה `system.listMethods` מחזירה רשימה של כל הפונקציות הזמינות. ניתן להשתמש גם ב-`system.methodHelp` לתיאור הפונקציה וב-`system.methodSignature` כדי לדעת אילו סוגי פרמטרים היא מקבלת.

8.7.1.7 השרת (The Server)

כתיבת השרת דומה לכתיבת הלקוח. אנו מממשים את הפונקציות ב-PHP (למשל `hello` או `add`), מגדירים את ה"חתימה" שלהן (הסוגים שהן מקבלות ומחזירות) בתוך מערך `$methods`, ומפעילים את השרת על ידי יצירת אובייקט `new XML_RPC_Server($methods)`. הקונסטרקטור של המחלקה מטפל בניתוח הבקשה ובקריאה לפונקציות המתאימות באופן אוטומטי.

בזה נסיים את נושא ה-XML-RPC.

8.7.2 SOAP

סעיף זה ידריך אתכם בשימוש ב-SOAP כלקוח עבור ה-Google Web API ובהטמעת שרת SOAP משלכם. מכיוון ש-SOAP מורכב אף יותר מ-XML-RPC, למרבה הצער לא נוכל לכלול את הכל.

8.7.2.1 SOAP::PEAR

גוגל הוא מנוע חיפוש נחמד ומהיר. האם זה לא היה נהדר לו היה לכם מנוע חיפוש משלכם בשורת הפקודה הכתוב ב-PHP? סעיף זה מסביר כיצד לעשות זאת.

Google: כדי לעשות שימוש ב-API SOAP שגוגל מייצאת, אתם זקוקים לחשבון, אותו תוכלו ליצור בכתובת `http://www.google.com/apis`. בעת ההרשמה, תקבלו מפתח (Key) בדואר אלקטרוני, בו תשתמשו כאשר תקראו למתודות ה-SOAP.

כדי שהדוגמה הבאה תעבוד כראוי, עליכם להתקין את מחלקת ה-SOAP של PEAR באמצעות הפקודה `pear install SOAP`. לאחר ההתקנה, נוכל להתחיל עם הסקריפט הפשוט הבא:

PHP

```
usr/local/bin/php/!#
php?>
;require_once 'SOAP/Client.php'
... הגדרת פרמטרים ומפתח ...
$params = array(
    'key' => $key
```

```

        ,q' => $query'
        ,start' => $start'
        ... פרמטרים נוספים ... //
    );
    ;('client = new SOAP_Client('http://api.google.com/search/beta2$
        )response = $client->call$
        ,doGoogleSearch'
        ,params$
        ('array('namespace' => 'urn:GoogleSearch
    );

```

בדוגמה זו, אנו מניחים שקריאת החיפוש החזירה משהו מועיל. ה-API של גוגל מחזיר טקסט עם ישויות XML מומרות וכמה תגיות `
` מוכנסות. אנו ממירים את הישויות חזרה לתווים רגילים באמצעות `html_entity_decode()` ומסירים את כל התגיות בעזרת `strip_tags()`.

שרת SOAP: ראשית נכלול את מחלקת `SOAP_Server` של PEAR. לאחר מכן נגדיר מחלקה (Example) עם שתי הפונקציות שאנו רוצים לייצא. במתודה `hello()` נשתמש בהמרה משתמעת מסוגי PHP לסוגי SOAP; במתודה `add()` נגדיר במפורש את סוג ה-SOAP (float):

```

PHP
php?>
;require_once 'SOAP/Server.php'
} class Example
} (function hello($name
;return "Hi $name
{
} (function add($a, $b
;return new SOAP_Value('return', 'float', $a + $b
{
{
;server = new SOAP_Server$
;()soapclass = new Example$
;('server->addObjectMap($soapclass, 'urn:Example$
;(server->service($HTTP_RAW_POST_DATA$
<?

```

לקוח SOAP: הלקוח דומה מאוד ללקוח של גוגל, פרט לכך שהשתמשנו בטיפוס מפורש (explicit typing) עבור הפרמטרים בקריאה למתודה `add()` באמצעות `SOAP_Value`.

8.7.2.2 הרחבת ה-SOAP של PHP

PHP 5 מגיע גם עם הרחבת SOAP מובנית (`ext/soap`), שהיא בעלת תכונות רבות יותר מ-`PEAR::SOAP`, וכתובה ב-C (מה שהופך אותה למהירה יותר).

הרחבת ה-SOAP תומכת גם ב-WSDL (נהגה "וויז-דל"), אוצר מילים של XML המשמש לתיאור שירותי רשת (Web Services). באמצעות קובץ WSDL זה, ההרחבה יודעת פרטים כמו נקודת הקצה (`endpoint`), הפרוצדורות וסוגי ההודעות.

כאשר משווים את השימוש ב-WSDL לשיטה הקודמת, רואים שקריאה למתודת SOAP היא הרבה יותר קלה – שתי שורות בלבד!

שרת SOAP: פיתוח שרת SOAP וקובץ ה-WSDL הנלווה אליו אינו קשה במיוחד; הבעיה הגדולה ביותר היא יצירת קובץ תיאור ה-WSDL עצמו. קוד השרת מחבר את המחלקה המספקת את המתודות לשרת ה-SOAP בעזרת קובץ ה-WSDL, והמתודה `handle()` דואגת לעיבוד המידע כאשר לקוח מבקש קריאה למתודה.

לקוח SOAP: הלקוח משתמש ב-`SoapClient` וביכולות הטיפול בשגיאות של `try-catch` (PHP 5). אם ננסה לקרוא למתודה ללא פרמטרים נדרשים, תיזרק חריגת `SoapFault`.

8.8 סיכום

XML תוכן בעיקר לשימוש בחילופי מידע בין מערכות. ל-XML טרמינולוגיה משלו המתארת את מבנה המסמכים, כאשר המידע עטוף בתגיות המזהות אותו בצורה מובנית. כדי להשתמש במידע, עליכם לנתח (Parse) את המסמכים.

PHP מספקת שני מנתחים עיקריים:

1. **SAX** (Simple API for XML) – מנתח כל אלמנט במסמך תוך כדי תנועה.
2. **DOM** (Document Object Model) – יוצר עץ היררכי בזיכרון של כל מבנה המסמך.

PHP 5 מספקת גם את הרחבת SimpleXML הקלה יותר לניתוח מסמכים פשוטים.

לצורך המרת מסמכי XML לפורמטים אחרים (כמו HTML), השיטה הסטנדרטית היא XSLT, המשתמשת בגיליונות סגנון ובתבניות.

לתקשורת בין יישומים במערכות שונות, PHP מציעה שני פתרונות מבוססי XML:

- **XML-RPC** – מאפשר ללקוח להריץ מתודות על שרת.
- **SOAP** – מגדיר פורמט להחלפת נתונים רחב ומורכב יותר, עם פוטנציאל רב ליישומים עתידיים.

פרק 9

הרחבות מרכזיות (Mainstream Extensions)

"הדבר החשוב הוא לא להפסיק לשאול." – אלברט איינשטיין

9.1 מבוא

הפרקים הקודמים כיסו את ההרחבות הנפוצות ביותר. פרק זה מציג הרחבות מרכזיות וערכיות נוספות. הסעיף הראשון מתאר קבוצת פונקציות שהן חלק מליבת ה-PHP ולא הרחבה נפרדת. שאר הסעיפים דנים במספר הרחבות פופולריות ושימושיות שאינן חלק מהליבה.

לאחר סיום פרק זה, תלמדו:

- לפתוח, לקרוא ולכתוב קבצים מקומיים ומרוחקים.
- לתקשר עם תהליכים ותוכניות.

- לעבוד עם תזרימים (Streams).
- לבצע התאמת טקסט, אימות קלט, החלפת טקסט, פיצול טקסט ומניפולציות טקסט אחרות באמצעות ביטויים רגולריים (Regular Expressions).
- לטפל בביתוח ופרמוט של תאריכים ושעות, כולל סוגיות של שעון קיץ.
- ליצור תמונות עם הרחבת ה-GD.
- לחלץ מידע מטא (Meta Information) מתמונות דיגיטליות בעזרת הרחבת ה-Exif.
- להמיר בין סטים של תווים בבית-בודד (Single-byte) ומרובי-בתים (Multi-byte).

9.2 קבצים ותזרימים (Files and Streams)

הגישה לקבצים השתנתה באופן דרסטי. לפני גרסת PHP 4.3.0, לכל סוג קובץ (מקומי, דחוס, מרוחק) היה מימוש שונה. עם זאת, מאז הצגת התזרימים, כל אינטראקציה עם קובץ עושה שימוש בשכבת ה-**Streams**, שכבה המפשיטה את הגישה לפרטי המימוש של סוג ה"קובץ" הספציפי. שכבת התזרימים מאפשרת ליצור אובייקט תמונה ב-GD ממקור HTTP באמצעות תזרים URL, לעבוד עם קבצים דחוסים, או להעתיק קובץ ממקום למקום. ניתן להחיל המרות משלכם במהלך ההעתקה על ידי מימוש תזרים-משתמש (User-stream) או מסנן (Filter).

9.2.1 גישה לקבצים

נתחיל בפונקציות הבסיסיות לגישה לקבצים. במקור, פונקציות אלו עבדו רק על קבצים רגילים, ולכן שמן מתחיל באות "f", אך PHP הרחיבה זאת כמעט לכל דבר. הפונקציות הנפוצות ביותר הן:

- `fopen()` – פותחת ידית (Handle) לקובץ מקומי או לקובץ מכתובת URL.
- `fread()` – קוראת בלוק של נתונים מקובץ.
- `fgets()` – קוראת שורה בודדת מקובץ.
- `fwrite()` / `fputs()` – כותבת בלוק נתונים לקובץ.
- `fclose()` – סוגרת את ידית הקובץ הפתוחה.
- `feof()` – מחזירה אמת (True) כאשר מגיעים לסוף הקובץ.

העבודה עם קבצים היא פשוטה, כפי שמראה הדוגמה הבאה:

PHP

`<?php`

```
;$fp = fopen("counter.dat", "r");
```

```
;$count = (int) fgets($fp, 20);
```

```
;$count++;
```

```
rewind($fp);
```

```
fwrite($fp, $count);
```

```
fclose($fp);
```

`>`

בשורה 3, ידית קובץ (fp\$) משויכת לתזרים, והתזרים משויך לקובץ counter.dat שעל הדיסק. הפרמטר השני ב-fopen() הוא ה-Mode (מצב), המציין האם התזרים נפתח לקריאה, כתיבה, שניהם או הוספה (Append):

- r: קריאה בלבד. המצביע בתחילת הקובץ.
- r+: קריאה וכתיבה. המצביע בתחילת הקובץ.
- w: כתיבה בלבד. תוכן הקובץ נמחק והמצביע בתחילה. אם הקובץ לא קיים, המערכת תנסה ליצור אותו.
- w+: קריאה וכתיבה. הקובץ נמחק והמצביע בתחילה.
- a: כתיבה בלבד (Append). המצביע בסוף הקובץ. אם הקובץ לא קיים, המערכת תנסה ליצור אותו.
- a+: קריאה וכתיבה. המצביע בסוף הקובץ.

ניתן להוסיף את המגדיר b כדי לציין שהקובץ הוא בינארי. מערכות Windows מבחינות בין קבצי טקסט לבינאריים; ללא שימוש ב-b, הקובץ עלול להישחת. לכן, כדאי להשתמש ב-b תמיד כדי להבטיח ניידות (Portability).

9.2.2 קלט/פלט של תוכניות

בדומה לפרדיגמה של UNIX שבה "כל קלט/פלט הוא קובץ", ב-PHP הפרדיגמה היא "כל קלט/פלט הוא תזרים". כשרוצים לעבוד עם קלט ופלט של תוכנית חיצונית, פותחים תזרים לאותה תוכנית באמצעות fopen() או proc_open().

9.2.2.1 fopen()

פונקציה פשוטה המספקת קלט/פלט חד-כיווני בלבד (רק r או w). פתיחת תזרים כזו נקראת Pipe (צינור). ניתן להשתמש בפונקציות קבצים רגילות כדי לקרוא ממנו.

9.2.2.2 proc_open()

מאפשרת משימות אינטראקטיביות יותר. ניתן לקשר את כל ערוצי הקלט והפלט של התהליך לצינורות (Pipes) או לקבצים. הפונקציה דורשת שלושה פרמטרים: פקודה להרצה, מערך הגדרות (descriptorspec), ומערך צינורות.

9.2.2.3 מתארי קבצים (File Descriptors)

מתאר קובץ הוא מספר המייצג ערוץ תקשורת. כברירת מחדל, 0 הוא קלט סטנדרטי (stdin), 1 הוא פלט סטנדרטי (stdout), ו-2 הוא שגיאה סטנדרטית (stderr). בעזרת proc_open() ניתן לנתב ערוצים אלו לקבצים או למשתנים בתוך הסקריפט, מה שמאפשר שליטה מלאה בתהליכי מערכת חיצוניים.

9.2.2.4 צינורות (Pipes)

במקום להשתמש בידית קובץ עבור קלט ופלט לתהליך ה-PHP, כפי שהוצג בסעיף הקודם, ניתן לפתוח צינורות (pipes) לתהליך ה-PHP המאפשרים לכם לשלוט בתהליך שנוצר מתוך הסקריפט שלכם. הסקריפט הבא שולח קוד PHP מתוך הסקריפט עצמו למפרש ה-PHP שנוצר. הסקריפט כותב את הפלט של פקודת ה-echo לפלט הסטנדרטי, תוך הפעלת urlencode על מחרוזת הפלט "Hello you!".

PHP

php?>

```
;$descs = array(0 => array('pipe', 'r'), 1 => array('pipe', 'w');
```

```

;(res = proc_open("php", $descs, $pipes$
    } ((if (is_resource($res
;('<? ";"fputs($pipes[0], '<?php echo "Hello you!\n
    ;(fclose($pipes[0
    } (([while (!feof($pipes[1
;([line = fgets($pipes[1$
; (echo urlencode($line
    {
        ;(proc_close($res
    {
<?

```

הפלט הוא: Hello+you%21%0A

9.2.2.5 קבצים

ניתן להעביר קובץ כמעבד עבור מתארי הקבצים (file descriptors) של התהליך שלכם, כפי שמוצג בדוגמה הבאה:

PHP

```

php?>
)descs = array$
,('array('pipe', 'r' <= 0
,('array('file', 'output', 'w' <= 1
('array('file', 'errors', 'w' <= 2
;
;(res = proc_open("php", $descs, $pipes$
    } ((if (is_resource($res
;('<? ";"fputs($pipes[0], '<?php echo "Hello you
    ;(fclose($pipes[0
    ;(proc_close($res
    {
<?

```

קובץ הפלט מכיל כעת Hello you! וקובץ ה-'errors' ריק.

בנוסף לצינור הקלט [pipe[0 וצינור הפלט [pipe[1 שהוצגו, ניתן להשתמש בצינורות אחרים כדי לנתב מחדש את כל מתארי הקבצים של תהליך הבן. בדוגמה לעיל, ניתבנו את כל הודעות השגיאה (מתאר 2) לצינור [pipe[2, הקובץ errors. האינדקס במערך \$descs אינו מוגבל ל-0-2, כך שניתן לשלוח בכל מתאר קובץ ראות עיניכם (אם כי ב-Windows אינדקסים מעל 2 טרם נתמכים ב-PHP).

9.2.3 תזרימי קלט/פלט (I/O Streams)

ב-PHP ניתן להשתמש ב-stdout, stderr ו-stdin כקבצים. "קבצים" אלו, המקושרים לתזרימי התהליך, נגישים באמצעות מציין הפרוטוקול //:php בקריאה ל-fopen(). תכונה זו שימושית במיוחד בעבודה עם ממשק שורת הפקודה (CLI).

קיימים שני תזרימי קלט/פלט נוספים: `php://input` ו-`php://output`. בעזרת `php://input` ניתן לקרוא נתוני POST גולמיים (Raw Data), דבר השימושי בעיבוד בקשות XML-RPC, WebDAV או SOAP.

כפי שניתן לראות באיור 9.1, לעיתים נתוני ה-POST הגולמיים ב-`php://input` מכילים מידע רב יותר מאשר מערך ה-`POST_$_` האוטומטי, במיוחד כאשר ישנם שדות בעלי שמות זהים שאינם מוגדרים כמערך. התזרים `php://output` מאפשר כתיבה ישירה למאגרי הפלט של PHP (בדומה ל-`echo`).

9.2.4 תזרימי דחיסה

PHP מספקת עטיפות (wrappers) עבור פונקציות דחיסה. קריאה וכתיבה מקובץ דחוס ב-`gzip` או `bzip` עובדת בדיוק כמו בקובץ רגיל, באמצעות המציינים `compress.zlib` ו-`compress.bzip2`.

תזרימי Gzip תומכים במדדים נוספים כמו רמת דחיסה (1-9) ושיטות דחיסה (f למסונן, h להופמן בלבד). ניתן אפילו לשרשר עטיפות, למשל פתיחת קובץ דחוס ישירות מכתובת URL:

```
compress.zlib://http://www.example.com/foobar.gz
```

9.2.5 תזרימי משתמש (User Streams)

שכבת התזרימים ב-PHP 5 מאפשרת להגדיר "תזרימי משתמש" – עטיפות תזרים הממומשות בקוד PHP באמצעות מחלקה. עבור כל פעולת קובץ (פתיחה, קריאה וכדומה), יש לממש מתודה תואמת:

- `stream_open`: נקראת בעת `fopen`.
- `stream_read`: מחזירה נתונים בעת `fread` או `fgets`.
- `stream_write`: נקראת בעת כתיבה לתזרים.
- `stream_seek` / `stream_tell`: לניווט ודיווח על המיקום בתזרים.
- `stream_eof`: בודקת האם הגענו לסוף התזרים.

9.2.6 URL תזרימי

אלו תזרימים שהנתיב שלהם מזכיר URL רגיל. התזרימים הבסיסיים הנתמכים הם:

- `http` ו-`https`: עבור קבצים בשרתי HTTP (דורש OpenSSL עבור SSL).
- `ftp` ו-`ftps`: עבור קבצים בשרתי FTP.

עבור אימות (Authentication), ניתן להוסיף את שם המשתמש והסיסמה בתוך ה-URL:

```
;'fp = fopen('ftp://username:password@ftp.example.com', 'wb$
```

PHP

`php?>`

```
;$context = stream_context_create(array('ftp' => array('overwrite' => true$
```

```
;'fp = fopen('ftp://secret@ftp.php.net', 'wb', false, $context$
```

`<?`

איור 9.2: phpsuck בפעולה.

הדוגמה הבאה מדגימה קריאת קובץ משרת HTTP ושמירתו כקובץ דחוס. דוגמה זו מציגה גם פרמטר רביעי לקריאה ל-fopen() המגדיר **Context** (הקשר) עבור התזרים. באמצעות פרמטר זה, ניתן להגדיר אפשרויות מיוחדות, כמו למשל **notifier** (מדווח). פונקציית ה-callback הזו תיקרא באירועים שונים במהלך התקשורת:

```
PHP
usr/local/bin/php/#!/#

php?>

/* מיפוי חומרת הודעות */
$severity_map = array(
    'info' => 0,
    'warning' => 1,
    'error' => 2,
);

/* פונקציית Callback עבור אירועי תזרים */
function notifier($code, $severity, $msg, $xcode, $sofar, $max)
{
    global $term_sol, $severity_map, $max_kb_sec, $size;

    /* אל תדפיס קידומת סטטוס כאשר מתקבל אירוע PROGRESS */
    if ($code != STREAM_NOTIFY_PROGRESS)
    {
        echo $severity_map[$severity] . " : " . $msg . "\n";
    }

    switch ($code)
    {
        case STREAM_NOTIFY_CONNECT:
            printf("Connected\n");
            GLOBALS['begin_time'] = time() - 0.001$
```

```

;break

:case STREAM_NOTIFY_AUTH_REQUIRED
;((printf("Authentication required: %s\n", trim($msg
;break

:case STREAM_NOTIFY_AUTH_RESULT
;((printf("Logged in: %s\n", trim($msg
;break

:case STREAM_NOTIFY_MIME_TYPE_IS
;((printf("Mime type: %s\n", $msg
;break

:case STREAM_NOTIFY_FILE_SIZE_IS
;((printf("Downloading %d kb\n", $max / 1024
;size = $max$
;break

:case STREAM_NOTIFY_REDIRECTED
;((printf("Redirecting to %s...\n", $msg
;break

:case STREAM_NOTIFY_PROGRESS
/* חישוב מד התקדמות */
} (if ($size
;(stars = str_repeat ('*', $sofar * 50 / $size$
} else {
;" = stars$
{
;(stripe = str_repeat ('-', 50 - strlen($stars$
/* חישוב מהירות הורדה */

```

```

;kb_sec = ($sofar / (time() - $GLOBALS['begin_time'])) / 1024$

/* השהיית הסקריפט אם המהירות גבוהה מהמקסימום שהוגדר */
} (while ($kb_sec > $max_kb_sec

;usleep(1

;kb_sec = ($sofar / (time() - $GLOBALS['begin_time'])) / 1024$

{

; (printf("${term_sol}[%s] %d kb %.1f kb/sec", $stars.$stripe, $sofar / 1024, $kb_sec

; break

: case STREAM_NOTIFY_FAILURE

; (printf("Failure: %s\n", $msg

; break

{

{

/* קביעת שם הקובץ לשמירה */

; ([url_data = parse_url($argv[1]$

; ([file = basename($url_data['path'$

{ ; if (empty($file)) { $file = "index.html

; ("printf ("Saving to $file.gz\n

; "fil = "compress.zlib://$file.gz$

/* notifier-הגדרת והקשר */

; ()context = stream_context_create$

; ("stream_context_set_params($context, array ("notification" => "notifier

```

```

/* פתיחת כתובת היעד */
;($fp = fopen($url, "rb", false, $context$
} ((if (is_resource($fp
;($fs = fopen($fil, "wb9", false, $context$
} ((if (is_resource($fs
} ((while (!feof($fp
;($data = fgets($fp, 1024$
;($fwrite($fs, $data
{
;($fclose($fs
{
;($fclose($fp
printf("{ $term_sol}{%s} Download time: %ds\n", str_repeat("*", 50), time() -
;([ $GLOBALS["begin_time
{
<?

```

ניתן ל¹ טפל באירועים שונים בתוך פונקציית ה-notify. להלן הרשימה המלאה:

- **STREAM_NOTIFY_CONNECT**: נורה כאשר נוצר חיבור עם המשאב.
- **STREAM_NOTIFY_AUTH_REQUIRED**: מופעל כאשר נדרשת הרשאה.
- **STREAM_NOTIFY_AUTH_RESULT**: מופעל בסיום תהליך האימות (הצלחה או כישלון).
- **STREAM_NOTIFY_MIME_TYPE_IS**: מופעל ב-HTTP כאשר כותרת ה-Content-Type זמינה.
- **STREAM_NOTIFY_FILE_SIZE_IS**: מופעל כאשר גודל הקובץ מזוהה (ב-FTP או דרך Content-Length ב-HTTP).
- **STREAM_NOTIFY_REDIRECTED**: מופעל בעת זיהוי הפניה (Redirect).
- **STREAM_NOTIFY_PROGRESS**: נורה ברגע שמגיעה חבילת נתונים (משמש למדי התקדמות).
- **STREAM_NOTIFY_FAILURE**: מופעל כאשר מתרחשת תקלה (כמו פרטי התחברות שגויים).

9.2.7 נעילה (Locking)

בעת כתיבה לקבצים שייכת ונקראים על ידי סקריפטים אחרים בו-זמנית, עלולות להיווצר בעיות (קריאת קובץ חלקי). הפתרון לכך נקרא נעילה (Locking).

PHP מאפשרת להגדיר נעילות באמצעות הפונקציה flock(). היא מונעת מסקריפט קורא לגשת לקובץ בזמן שסקריפט אחר כותב אליו, בתנאי ששני הסקריפטים מממשים את מנגנון הנעילה.

9.2.8 שינוי שם ומחיקת קבצים

PHP מספקת את הפונקציה unlink() למחיקת קובץ (ניתוק מהספרייה). במערכות UNIX, הקובץ יימחק פיזית מהדיסק רק לאחר שכל התוכניות המשתמשות בו יסגרו את הידיית שלהן (fclose()).

הזזת קובץ עם הפונקציה rename() היא פעולה אטומית (Atomic) אם הקובץ נשאר באותה מערכת קבצים. פעולה אטומית מבטיחה שהתהליך לא יופרע באמצע. אם עוברים בין מערכות קבצים שונות, בטוח יותר להעתיק את הקובץ לשם זמני ורק אז לשנות את שמו לשם הסופי באותה מערכת קבצים, כדי להבטיח שהקובץ לעולם לא יופיע שם במצב חלקי.

9.2.9 קבצים זמניים

במידה ואתם מעוניינים ליצור קובץ זמני, הדרך הטובה ביותר לעשות זאת היא באמצעות הפונקציה tmpfile(). פונקציה זו יוצרת קובץ זמני עם שם אקראי ייחודי בספרייה הנוכחית ופותחת אותו לכתיבה. קובץ זה ייסגר (ויימחק) אוטומטית כאשר תסגרו אותו עם fclose() או כשהסקריפט יסתיים.

אם אתם רוצים שליטה רבה יותר על המיקום שבו נוצר הקובץ הזמני ועל שמו, תוכלו להשתמש בפונקציה tempnam(). בניגוד ל-tmpfile(), קובץ זה לא יוסר אוטומטית מהדיסק:

הפרמטר הראשון לפונקציה מציין את הספרייה שבה ייווצר הקובץ, והשני הוא קידומת (prefix) שתתווסף לשם הקובץ האקראי.

9.3 ביטויים רגולריים (Regular Expressions)

למרות שביטויים רגולריים הם חזקים מאוד, הם קשים לשימוש, במיוחד למתחילים. לכן, לפני שנצלול לפונקציות ה-PHP, נכסה קודם את תחביר התבניות (Pattern Syntax). אם PCRE מופעל, תוכלו לראות זאת בפלט של phpinfo() (איור 9.3).

9.3.1 תחביר (Syntax)

פונקציות PCRE בודקות האם מחרוזת טקסט תואמת לתבנית מסוימת. תחביר התבנית תמיד נראה כך: מגדירים [מפריד] [תבנית] [מפריד].

המגדירים (modifiers) הם אופציונליים. המפריד (delimiter) מפריד בין התבנית למגדירים. PCRE משתמש בתו הראשון של הביטוי כמפריד. נהוג להשתמש ב-/, אך ניתן להשתמש גם ב-| או @.

הפונקציה preg_match() משמשת להתאמת ביטויים רגולריים. היא מחזירה TRUE אם נמצאה התאמה ו-FALSE אם לאו. ניתן להעביר פרמטר שלישי (אופציונלי), משתנה שאליו יוכנסו התוצאות שנמצאו.

9.3.1.2 תווים מיוחדים (Metacharacters)

התווים בטבלה 9.1 הם תווים מיוחדים המשמשים לבניית תבניות:

טבלה 9.1: תווים מיוחדים

| תו | תיאור |

| ---: | ---: |

| \ | תו מילוט (Escape). משמש לביטול המשמעות המיוחדת של תווים מיוחדים אחרים. |

| . | מתאים בדיוק לתו אחד (פרט לתו שורה חדשה). |

| ? | מסמן שהתו או תת-התבנית שקדמו לו הם אופציונליים (0 או 1 פעמים). |

| + | מתאים לתו הקודם פעם אחת או יותר. |

| * | מתאים לתו הקודם אפס פעמים או יותר. |

| {m,n} | התאמה בין m ל-n פעמים. |

| ^ | מסמן את תחילת המחרוזת. |

| \$ | מסמן את סוף המחרוזת. |

| [...] | מגדיר מחלקת תווים (למשל [0-9] עבור כל ספרה). |

| (...) | יוצר תת-תבנית (Sub-pattern) לקבוצה, שתופיע בנפרד במערך התוצאות. |

| (...)? | יוצר תת-תבנית ללא "לכידה" (לא תופיע במערך התוצאות בנפרד). |

9.3.1.3 דוגמה 1: כתובת MAC

התבנית `/^[0-9a-f]{2}:[0-9a-f]{2}:[0-9a-f]{2}:[0-9a-f]{2}:[0-9a-f]{2}:[0-9a-f]{2}$/` מתאימה לכתובת MAC בפורמט `00:04:23:7c:5d:01`. היא מחפשת 5 קבוצות של שתי ספרות הקסדצימליות ונקודתיים, ואחריהן קבוצה שישית ללא נקודתיים.

9.3.1.4 דוגמה 2: כתובת אימייל פשוטה

התבנית `/^[a-zA-Z0-9_!@#$%^&*~.-]+@[a-zA-Z0-9_!@#$%^&*~.-]+\.[a-zA-Z]{2,}$/` משמשת להתאמה של פורמט כמו `Derick Rethans <derick@php.net>`.

9.3.1.5 רצפי מילוט (Escape Sequences)

בנוסף לביטול סימנים מיוחדים, \ משמש לקיצורים (טבלה 9.2):

- `d\`: כל ספרה (כמו `[0-9]`).
- `D\`: כל תו שאינו ספרה.
- `s\`: כל תו "רווח" (רווח, טאב, שורה חדשה).
- `w\`: כל תו "מילה" (אותיות, ספרות וקו תחתון).
- `b\`: גבול מילה (הנקודה שבין תו מילה לתו שאינו מילה).

9.3.1.7 התאמה "עצלנית" (Lazy Matching)

כברירת מחדל, אופרטורים כמו `+` ו-`*` הם "חמדנים" (Greedy) – הם ינסו להתאים כמה שיותר טקסט. הוספת `?` אחריהם (למשל `.*?`) הופכת אותם ל"עצלנים", כך שיעצרו בנקודה הראשונה האפשרית.

9.3.2 פונקציות

PHP

php?>

```
;$str = "This is an example for preg_split$
```

```
;(ops = preg_split('/[s.]+/', $str$
```

```
;(print_r($ops
```

<?

ניתן להעביר פרמטרים נוספים כמו **limit** (הגבלת מספר האיברים המוחזרים) ודגלים (**flags**):

- **PREG_SPLIT_NO_EMPTY**: מונע החזרת איברים ריקים.
- **PREG_SPLIT_DELIM_CAPTURE**: מחזיר גם את הטקסט ששימש כמפריד (אם הוא מוקף בסוגריים בתבנית).
- **PREG_SPLIT_OFFSET_CAPTURE**: מחזיר מערך דו-ממדי הכולל גם את המיקום (offset) של כל איבר במחרוזת.

9.4 טיפול בתאריכים

PHP מציעה מגוון פונקציות לטיפול בתאריך ושעה. רובן עובדות עם **UNIX timestamp** – מספר השניות שחלפו מאז ה-1 בינואר 1970. בגלל מגבלות של 32-ביט, טווח התאריכים ברוב המערכות הוא בין 1970 ל-2038.

9.4.1 שליטת במידע על תאריך ושעה

- **time()**: מחזירה את חותמת הזמן הנוכחית בשניות.
- **microtime()**: מחזירה זמן בדיוק גבוה יותר (כולל מיקרו-שניות).
- **gettimeofday()**: מחזירה מערך עם פרטי הזמן הנוכחי ומיקרו-שניות.
- **localtime()** ו-**getdate()**: מקבלות חותמת זמן ומחזירות מערך מפורט (שניות, דקות, שעות, יום בשבוע וכו').

טבלה 9.5: רכיבים במערכי **localtime()** ו-**getdate()**

האלמנט **tm_isdst** ב-**localtime()** הוא הדרך היחידה ב-PHP לבדוק אם השרת נמצא כרגע בשעון קיץ (DST).

mktime() ו-**gmmktime()** יוצרות חותמת זמן מפרמטרים נפרדים (שעה, דקה, חודש וכו'). ההבדל הוא ש-**gmmktime()** מתייחסת לפרמטרים כזמן גריניץ' (GMT), בעוד **mktime()** מתייחסת אליהם כזמן מקומי לפי הגדרות השרת.

9.4.2 פרמטרי תאריך ושעה

לצורך הצגת התאריך בפורמט קריא, נשתמש ב-**date()** (זמן מקומי) או ב-**gmdate()** (זמן GMT). שתי הפונקציות משתמשות במחרוזת פורמט (Format String). הן תמיד מחזירות שמות באנגלית. כדי להציג תאריכים בשפה המוגדרת במערכת (Locale), יש להשתמש ב-**strftime()** ו-**gmstrftime()**.

דוגמה 1: מספרי שבועות לפי ISO 8601

לפי תקן ISO 8601, השבוע הראשון של השנה הוא השבוע שבו יש לפחות ארבעה ימים. לכן, ה-1 בינואר עשוי לפעמים להיות שייך רשמית לשבוע 53 של השנה הקודמת.

9.4.2.2 דוגמה 2: בעיות בשעון קיץ (DST)

בכל שנה בסביבות אוקטובר, מדווחים עשרות באגים שבהם יום מסוים מופיע פעמיים בתצוגה. למעשה, היום שמופיע פעמיים הוא התאריך שבו מסתיים שעון הקיץ.

הסיבה לכך היא שבמעבר משעון קיץ לחורף, יש למעשה 25 שעות בין חצות של ה-31 באוקטובר לחצות של ה-1 בנובמבר, ולא 24 שעות כפי שנוספו בלולאה פשוטה. ניתן לפתור זאת בשתי דרכים:

1. בחירת שעה אחרת ביום (למשל צהריים), כך שהסטייה של שעה אחת לא תחצה את גבול חצות.
2. "שימוש לרעה" מושכל בפונקציה `mktime()`: במקום להוסיף שניות (שמשתנות בגלל DST), מוסיפים את מספר הימים ישירות לפרמטר היום ב-`mktime()`. הפונקציה תדע לחשב נכון את המעבר לחודש הבא ולטפל בשעות ה-DST באופן אוטומטי.

9.4.2.3 דוגמה 3: הצגת זמן מקומי באזורי זמן אחרים

לפעמים תרצו להציג זמן בפורמט מקומי עבור אזורי זמן שונים. איור 9.4 מציג פלט של סקריפט המציג את ה-1 במרץ במדינות כמו ארה"ב, נורווגיה, הולנד וישראל.

הערה: לשם כך יש לוודא שה-Locales והגדרות אזורי הזמן מותקנים במערכת ההפעלה שלכם.

9.4.3 ניתוח (Parsing) של פורמטי תאריך

הפעולה ההפוכה לפרמוט היא ניתוח מחרוזת טקסטואלית והפיכתה לחותמת זמן (timestamp). הפונקציה `strtotime()` מטפלת במגוון רחב של פורמטים.

טבלה 9.7: פורמטי תאריך/זמן נתמכים (חלקי)

- **1970-09-17**: פורמט ISO 8601 מועדף.
- **next Thursday 7pm**: יום חמישי הבא בשעה שבע בערב.
- **year 2 days ago 1**: שנה ויומיים אחורה מהזמן הנוכחי.
- **yesterday / tomorrow**: אתמול / מחר.

9.5 מניפולציה של גרפיקה עם GD

במקום לעבור על כל הפונקציות, נדון בשני מקרים נפוצים של ספריית GD:

1. **טפסים מוגני-בוטים (CAPTCHA)**: יצירת תמונה עם קוד שקשה לקריאה ממוחשבת על ידי עיוותים וטשטוש.
2. **תרשימי עמודות (Bar Charts)**: יצירת גרף הכולל צירים, תוויות, רקע וטקסט TrueType.

כדי להשתמש ב-GD, יש לוודא שהיא מופעלת ב-PHP (איור 9.5).

9.5.1 מקרה 1: טפסים חסיני בוטים

השלבים ליצירת תמונה כזו כוללים:

- **יצירת קנבס:** שימוש ב-`imagecreatetruecolor()` ליצירת מרחב עבודה במיליוני צבעים.
- **הקצאת צבעים:** שימוש ב-`imagecolorallocate()` להגדרת צבעי רקע, גבול וטקסט (בפורמט RGB).
- **ציור טקסט:** שימוש ב-`imagefttext()` המאפשר לצייר אותיות בגדלים ובזוויות משתנות כדי להקשות על תוכנות זיהוי טקסט.
- **הוספת עיוותים:** ציור של מאות קווים קטנים אקראיים (`imageline()`) והפעלת Anti-aliasing להחלקת קצוות (איור 9.6).
- **פלט לדפדפן:** שליחת כותרת `Content-type: image/png` ולאחריה פקודת `imagepng()` כדי להציג את התמונה.

9.5.2 מקרה 2: תרשים עמודות

בגרפים מורכבים נשתמש בטכניקות מתקדמות יותר:

- **Alpha Blending:** שילוב ערוץ אלפא (שקיפות) כך שהעמודות יהיו שקופות למחצה וניתן יהיה לראות את הרקע דרכן.
- **טעינת תמונת רקע:** שימוש ב-`imagecreatefrompng()` לטעינת תמונה קיימת כבסיס.
- **חישוב תיבת טקסט (Bounding Box):** שימוש ב-`imageftbbox()` כדי לדעת בדיוק מהן המידות של מחרוזת טקסט לפני הציור, מה שמאפשר למרכז כותרות ותוויות בצורה מושלמת (איור 9.9).

Exif 9.5.3

Exif היא שיטה לשמירת מטא-דאטה (כמו זמן הצילום, מפתח צמצם ומהירות תריס) בתוך קובץ התמונה. PHP מאפשרת לקרוא נתונים אלו באמצעות פונקציות Exif (איור 9.11), מה שמאפשר להציג מידע טכני על התמונה באופן אוטומטי.

שימוש ב-PEAR

10.1 מבוא

ספר זה הזכיר את PEAR מספר פעמים בפרקים הקודמים. PEAR, קיצור של **PHP Extension and Application Repository**, היא מערכת חבילות עבור PHP. במהלך גרסה 4 של PHP, מספר המשתמשים זינק, ואיתו גם כמות קטעי הקוד שניתן היה להוריד מאתרים שונים. חלק מהאתרים הציעו קוד שהיה עליכם להעתיק ולהדביק לעורך שלכם, בעוד שאחרים אפשרו להוריד ארכיונים עם קבצי מקור. זה היה שימושי, אך נוצר צורך בדרך טובה יותר לשיתוף ושימוש חוזר בקוד PHP, בדומה ל-CPAN של Perl.

פרויקט PEAR הוקם כדי לפתור בעיה זו על ידי אספקת כלי התקנה ותחזוקה וסטנדרטים לניהול קוד וגרסאות. כיום, PEAR מספק:

- **PEAR Installer** (כלי לניהול חבילות)
- חבילות עם קוד ספריית PHP
- חבילות עם הרחבות PHP (הנקראות **PECL**)
- סטנדרטים של כתיבת קוד ב-PEAR, כולל סטנדרט גרסאות

ו-**PECL** (PHP Extension Community Library) הוא תוצר לוואי של פרויקט PEAR. בעבר הוא היה תת-קבוצה של PEAR, אך כיום הוא מנוהל בנפרד, עם אתר ורשימות תפוצה משלו. עם זאת, PEAR ו-PECL חולקים כלים ותשתית: שניהם משתמשים ב-PEAR Installer, באותו פורמט חבילה ובאותו

סטנדרט גרסאות. ההבדל הוא בסטנדרט הכתיבה: PECL עוקב אחרי הסטנדרט של PHP (לקוד C), בעוד ל-PEAR יש סטנדרט משלו.

בפרק זה נלמד על המושגים של PEAR וכיצד להשתמש ב-Installer כדי להתקין ולנהל חבילות.

10.2 מושגי PEAR

סעיף זה מסביר מושגים כמו חבילות (packages), גרסאות (releases) ושיטת מספור הגרסאות.

10.2.1 חבילות (Packages)

כשרוצים להתקין משהו מ-PEAR, מורידים ומתקינים גרסה (release) מסוימת של חבילה. לכל חבילה מצורף מידע:

- שם החבילה (למשל: HTML_QuickForm)
- תקציר, תיאור וכתובת אתר הבית
- מתחזק אחד או יותר
- מידע על רישיון
- מספר גרסאות שהופצו

חבילות PEAR דומות לפורמטים אחרים כמו RPM של לינוקס או חבילות דביאן. ההבדל העיקרי הוא שהן תוכננו להיות **בלתי תלויות בפלטפורמה**. ניתן להתקין אותן על כל פלטפורמה ש-PHP תומכת בה: UNIX, לינוקס, Windows ו-Mac OS X.

10.2.2 הפצות (Releases)

בדומה ל-PHP עצמה, הקוד מופץ בקובץ tar.gz או zip יחד עם הוראות התקנה הנקראות על ידי ה-Installer. כל הפצה כוללת:

- מספר גרסה
- רשימת קבצים והוראות התקנה
- מצב ההפצה (stable, beta, alpha, devel או snapshot)

כברירת מחדל, כשתתקינו חבילה, תקבלו את הגרסה היציבה (stable) האחרונה. למשל:

```
pear install XML_Parser $
```

הפקודה תוריד ותתקין את הגרסה היציבה האחרונה (לדוגמה 1.1.0).

10.2.3 מספרי גרסאות

סטנדרט הגרסאות של PEAR מגדיר כיצד לפרש מספר גרסה וכיצד להשוות בין שני מספרים. הוא מיושם באמצעות הפונקציה `version_compare()` של PHP.

10.2.3.1 פורמט מספר הגרסה

מספר גרסה יכול להיות פשוט כמו "1" או מורכב כמו "PEAR 8.1.1.2.9b2". מתמקדת לרוב בשלושה מספרים וחלק נוסף בסוף למקרים מיוחדים. התחביר הוא:

[[Major [. minor [. patch]] [dev | a | b | RC | pl [N

| גרסה | Major | Minor | Patch | מצב (State) |
|-------|-------|-------|-------|-------------|
| 1.2.1 | 1 | 2 | 1 | — |
| 1.0a1 | 1 | 0 | — | alpha 1 |

10.2.3.2 השוואת מספרי גרסאות

ההשוואה מתחילה במספר ה-Major. אם הוא זהה, עוברים ל-Minor, ואז ל-Patch. אם כולם זהים, בודקים את ה-State.

- מצבים כמו a, dev, (אלפא), b (בטא) ו-RC נחשבים **ישנים** יותר מגרסה ללא סיומת.
- מצב patch level (pl) נחשב **חדש** יותר.

Major לעומת Minor לעומת Patch:

- **Patch:** תמיד בטוח לשדרוג (תיקוני באגים בלבד).
- **Minor:** הוספת תכונות חדשות; ה-API עשוי להשתנות מעט. כדאי לקרוא את ה-changelog.
- **Major:** שינוי משמעותי. לא מובטחת תאימות לאחור. הקוד עשוי להיכתב מחדש לגמרי.

10.3 השגת PEAR

ניתן להתקין את PEAR כחלק מהפצת PHP או דרך האתר go-pear.org.

10.3.1 התקנה בלינוקס/UNIX

החל מגרסה 4.3.0, PEAR מותקן כברירת מחדל עם PHP. עם זאת, אם אתם מקמפלים את PHP מהמקור, שימו לב לא להשתמש ב--disable-pear או --disable-cli, שכן ה-Installer זקוק לממשק שורת הפקודה (CLI) ולתמיכה ב-XML.

10.3.1.1 Windows

התקינו את גרסת ה-Binary של PHP מהאתר הרשמי. כברירת מחדל, PHP תותקן ב-C:\PHP.

10.3.2 התקנה עם PHP Windows Installer

עליכם לוודא שה-include_path בקינפוג של PHP מוגדר נכון. ערכו את קובץ ה-php.ini שלכם ושנו את הנתיב ל-c:\php\pear כדי ש-PHP תוכל למצוא את החבילות המותקנות.

10.3.3 go-pear.org

זהו אתר עם סקריפט PHP יחיד שניתן להוריד ולהריץ כדי להתקין את הגרסה היציבה האחרונה של PEAR. הוא עובד על כל הפלטפורמות וניתן להריץ אותו משורת הפקודה או מהשרת.

דרישות מוקדמות:

מכיוון ש-go-pear כתוב ב-PHP, אתם זקוקים לגרסת CLI או CGI של PHP מותקנת. בדקו זאת על ידי הרצת `php -v`.

הפעלת go-pear:

ניתן להריץ את ההתקנה ישירות מהאינטרנט (למשל עם `lynx` או `curl`) ולהעביר את הפלט ל-PHP:

```
$ php | http://go-pear.org -source lynx
```

האם תרצה שאפרט על פקודות ספציפיות לניהול חבילות ב-PEAR, כגון חיפוש או הסרת חבילה?

איור 10.2 דוגמה לשינויים בקובץ `php.ini`.

כעת אתם מוכנים להשתמש ב-go-pear.

10.3.3 go-pear.org

go-pear.org הוא אתר אינטרנט המכיל סקריפט PHP יחיד שניתן להוריד ולהריץ כדי להתקין את הגרסה היציבה האחרונה של מתקין PEAR ושל מחלקות היסוד של PHP (PFC). ה-go-pear חוצה פלטפורמות וניתן להריץ אותו משורת הפקודה או משרת האינטרנט שלכם.

הפצות PHP מגיעות עם מהדורה מסוימת של מתקין PEAR; מנגד, go-pear מספק לכם את מהדורות PEAR היציבות החדשות ביותר. עם זאת, go-pear אמנם מכיר את מבנה הספריות שלכם, אך הוא מתאמץ מאוד כדי לפענח אותו וינסה להתאים את התקנת ה-PEAR שלכם למבנה זה.

בסעיף זה תלמדו כיצד להשתמש ב-go-pear משורת הפקודה ומשרת האינטרנט, הן ב-UNIX והן ב-Windows.

10.3.3.1 דרישות קדם

מכיוון ש-go-pear כתוב ב-PHP, דרושה לכם גרסת CGI או CLI של PHP כדי להריץ אותו מחוץ לשרת האינטרנט. כברירת מחדל, גרסת ה-CLI מותקנת יחד עם מודול ה-PHP של שרת האינטרנט. נסו להריץ `php -v` כדי לראות אם היא זמינה לכם:

Plaintext

```
PHP 5.0.0 (cli), Copyright (c) 1997-2004 The PHP Group
Zend Engine v2.0, Copyright (c) 1998-2004 Zend Technologies
```

כברירת מחדל, פקודת ה-`php` מותקנת בספרייה `usr/local/bin/` ב-UNIX, או ב-`c:\php\` ב-Windows. ב-Windows, גרסת ה-CLI של PHP עשויה להיקרא גם `php-cli`; במקרה כזה, עליכם להקליד `php-cli` בכל דוגמה שבה כתוב פשוט `php`.

10.3.3.2 מפעילים את PEAR

אם התקנת ה-PHP שלכם לא כללה את PEAR, תוכלו להשתמש ב-go-pear כ"מניע" (bootstrapper) אוניברסלי ל-PEAR. כל מה שאתם צריכים זה גרסת CLI או CGI של PHP מותקנת איפשהו.

ניתן להוריד את הסקריפט go-pear ולהריץ אותו, או להריץ הכל בפקודה אחת:

```
$ php | http://go-pear.org --source lynx
```

פקודה זו פשוט לוקחת את התוכן של <http://go-pear.org> ושולחת אותו ל-PHP לביצוע. אם אין לכם lynx, נסו דרך חלופית:

באמצעות wget:

```
$ php | http://go-pear.org -O -- wget
```

באמצעות fetch (ב-FreeBSD):

```
$ php | http://go-pear.org -o -- fetch
```

ב-Windows, אין כלי "משוך URL" מובנה, אך ייתכן שתוכלו להשתמש בזרמי ה-URL של PHP (וודאו ש-url_includes אינו מבוטל ב-php.ini):

```
"('C:\> php-cli -r "include('http://go-pear.org
```

אם שום דבר לא עובד, פתחו את הכתובת בדפדפן, שמרו את התוכן כ-go-pear.php והריצו:

```
C:\> php go-pear.php
```

הפלט יראה כך:

```
Plaintext
!Welcome to go-pear
Go-pear will install the 'pear' command and all the files needed by it
...
If you wish to abort, press Control-C now, or press Enter to continue
```

10.4 התקנת חבילות

סעיף זה עוסק בתחזוקת אוסף החבילות המותקנות שלכם.

למתקין PEAR יש ממשקים שונים (front-ends). ברירת המחדל היא ממשק שורת הפקודה (CLI).

10.4.1 שימוש בפקודת pear

פקודת ה-pear היא כלי ההתקנה הראשי. הפקודה הראשונה שכדאי להכיר היא help. הפקודה pear help subcommand תציג טקסט עזרה קצר.

אפשרויות (Options):

ניתן לציין אפשרויות (כמו -n) גם לפקודת ה-pear עצמה וגם לתת-הפקודה.

- -v: העלאת רמת הפירוט (Verbosity). רמה 0 היא שקטה, רמה 3 היא פלט לניפוי שגיאות (Debug).

- `-C / c`: ציון קובץ הגדרות (Configuration).
- `-D / d`: הגדרת פרמטר הגדרות עבור הרצה זו בלבד.
- `-S / s`: שמירת (Store) שינויי הגדרות שהתבצעו עם אפשרות `-d`.

10.5 פרמטרים של הגדרות (Configuration Parameters)

למתקין PEAR יש פרמטרים רבים. החשובים שבהם הם ספריות היעד להתקנה:

| פרמטר הגדרה | שם משתנה | ערך לדוגמה |
|-------------------|------------------------------|-----------------------------------|
| ספריית PEAR ראשית | <code>php_dir</code> | <code>usr/share/pear/</code> |
| ספריית קבצי הרצה | <code>bin_dir</code> | <code>usr/bin/</code> |
| ספריית תיעוד | <code>doc_dir</code> | <code>usr/share/pear/docs/</code> |
| מצב חבילה מועדף | <code>preferred_state</code> | <code>stable</code> |

שכבות הגדרה (Configuration Layers):

כל פרמטר יכול להיות מוגדר בשלוש שכבות:

1. **User (משתמש)**: הגדרות פרטיות של המשתמש.
2. **System (מערכת)**: הגדרות לכלל המערכת.
3. **Default (ברירת מחדל)**: ערכים מובנים במתקין.

המתקין בודק תחילה את שכבת המשתמש; אם לא נמצא ערך, הוא עובר למערכת, ולבסוף לברירת המחדל.

שינוי ההגדרות:

כדי לשנות פרמטר לצמיתות, השתמשו ב-`pear config-set <parameter> <value>`. כדי לראות ערך נוכחי, השתמשו ב-`pear config-get <parameter>`.

שימו לב: אם תשנו את `php_dir`, המתקין לא "יראה" יותר את החבילות שהותקנו בנתיב הישן, מכיוון שבספרייה זו נשמר בסיס הנתונים של החבילות המותקנות (תחת תיקיית `registry`).

10.6 פקודות PEAR

בסעיף זה תלמדו את כל פקודות מתקין PEAR עבור התקנה ותחזוקה של חבילות במערכת שלכם. עבור כל פקודה, יוצג הפלט של `pear help command` והסבר מפורט על כל אפשרות (option) שהפקודה מציעה. אם תבחינו בפקודות המופיעות בטקסט העזרה אך אינן מכוסות כאן, דעו כי אלו פקודות המשמשות מתחזקי חבילות PEAR במהלך הפיתוח. פקודות הפיתוח מכוסות בפרק 12.

pear install 10.6.1

פקודה זו לוקחת את התוכן של קובץ חבילה ומתקינה את הקבצים בספריות ה-PEAR שהגדרתם. ניתן לציין את החבילה להתקנה כקובץ מקומי, כשם חבילה בלבד או ככתובת HTTP מלאה. להלן טקסט העזרה עבור `pear install`:

```
pear help install $
```

```
... <pear install [options] <package
```

מתקין חבילת PEAR אחת או יותר. ניתן לציין חבילה להתקנה בארבע דרכים:

- **"Package-1.0.tgz"**: התקנה מקובץ מקומי.
- **"<http://example.com/Package-1.0.tgz>"**: התקנה מכל מקום ברשת.
- **"package.xml"**: התקנת החבילה המתוארת בקובץ `package.xml`. שימושי לבדיקות, או לעטיפת חבילת PEAR בתוך מנהל חבילות אחר כמו RPM.
- **"Package"**: תשאול השרת המוגדר שלכם (`pear.php.net`) והורדת החבילה החדשה ביותר באיכות/מצב המועדף (`stable`).

ניתן לציין יותר מחבילה אחת בו-זמנית, וניתן לערבב בין ארבע הדרכים הללו.

אפשרויות (Options):

- **--force, -f**: ידרוס חבילות מותקנות חדשות יותר. שימושי לתיקון התקנות פגומות או במהלך בדיקות.
- **--nodeps, -n**: התעלמות מתלויות (dependencies), התקנה בכל זאת. השתמשו בזה רק אם אתם מבינים את ההשלכות, שכן החבילה עלולה לא לעבוד כלל.
- **--register-only, -r**: רישום החבילה כמותקנת בלבד, ללא התקנת הקבצים בפועל. נועד לאפשר למנהלי חבילות חיצוניים (כמו RPM) לעדכן את מרשם ה-PEAR.
- **--soft, -s**: התקנה "רכה" – שדרוג אם החבילה קיימת, או התקנה אם לא.
- **--nobuild, -B**: אל תבנה הרחבות C. אם החבילה כוללת קוד C ואינכם מעוניינים לקמפל אותו, השתמשו באפשרות זו.
- **--nocompress, -Z**: בקשת קבצים לא דחוסים בעת ההורדה (שימושי אם אין תמיכה ב-zlib ב-PHP שלכם).
- **--installroot=DIR, -R DIR**: ספריית שורש שתשמש להתקנה (בדומה ל-INSTALL_ROOT של PHP).
- **--ignore-errors**: כפיית התקנה גם אם היו שגיאות. השתמשו בזהירות!
- **--alldeps, -a**: התקנת כל התלויות הנדרשות והאופציונליות באופן אוטומטי.
- **--onlyreqdeps, -o**: התקנת תלויות חובה בלבד, ללא תלויות אופציונליות.

דוגמאות לשימוש טיפוסי:

התקנה רגילה של חבילה ללא תלויות:

```
pear install Console_Table $
```


התקנה עם תלויות חובה בלבד (דילוג על אופציונליות):

```
pear install -o HTML_QuickForm $
```

התקנת חבילה וכל תלויותיה, בחיפוש אחר גרסאות במצב beta ומעלה:

```
pear -d preferred_state=beta install -a Services_Weather $
```

pear list 10.6.2

הפקודה `pear list` מציגה את רשימת החבילות המותקנות במערכת שלכם או את רשימת הקבצים בתוך חבילה ספציפית.

כדי לראות את כל החבילות המותקנות:

```
pear list $
```

כדי לבדוק אילו קבצים הותקנו עבור חבילה מסוימת (למשל `Net_Socket`):

```
pear list Net_Socket $
```

הפלט יראה את סוג הקובץ (`php`, `data`, `script`) ואת נתיב ההתקנה שלו.

pear info 10.6.3

הפקודה `pear info` מציגה מידע על חבילה מותקנת, קובץ חבילה (`tarball`) או קובץ הגדרת חבילה (`XML`). המידע כולל תיאור, מתחזקים, רישיון, גרסה והערות שחרור (`Release Notes`).

ניתן להריץ את הפקודה על חבילה מותקנת:

```
pear info XML_RPC $
```

או על קובץ שטרם הותקן:

```
pear info XML-RPC-1.1.0.tgz $
```

pear list-all 10.6.4

בעוד ש-`pear list` מציגה רק מה שמותקן אצלכם, `pear list-all` מציגה רשימה אלפביתית של כל החבילות הקיימות בשרת ה-PEAR, בצירוף הגרסה העדכנית ביותר והגרסה המותקנת אצלכם (אם יש כזו).

pear list-upgrades 10.6.5

הפקודה `pear list-upgrades` משווה בין הגרסאות המותקנות אצלכם לבין הגרסאות החדשות ביותר הזמינות בשרת (בהתאם להגדרת `preferred_state`). היא תציג טבלה עם הגרסה המקומית, הגרסה המרוחקת וגודל ההורדה.

pear upgrade 10.6.6

הפקודה pear upgrade מחליפה חבילה מותקנת אחת או יותר בגרסה חדשה יותר.

pear upgrade Log \$

לפקודה זו יש את אותן אפשרויות כמו לפקודת ה-install (למעט האפשרות -s). אם ברצונכם לשדרג בכל זאת לגרסה קיימת, השתמשו ב--force (f).

pear upgrade-all 10.6.7

לנוחיותכם, פקודה זו משלבת את list-upgrades ו-upgrade, ומשדרגת באופן אוטומטי את כל החבילות שיש להן גרסה חדשה יותר זמינה.

pear upgrade-all \$

pear uninstall 10.6.8

כדי למחוק חבילה, עליכם להסיר אותה מההתקנה (uninstall). להלן דוגמה:

pear uninstall Cache \$

'Warning: Package 'services_weather' optionally depends on 'Cache

uninstall ok: Cache

לפקודת ההסרה יש שלוש אפשרויות עיקריות:

... <pear uninstall [options] <package

ניתן להסיר חבילה אחת או יותר בו-זמנית.

אפשרויות (Options):

- **--nodeps-n**: התעלמות מתלוויות, הסרה בכל זאת.
- **--register-only-r**: אל תסיר את הקבצים בפועל, רק רשום את החבילה ככזו שאינה מותקנת.
- **--installroot=DIR-R DIR**: הגדרת ספריית שורש (בדומה ל-INSTALL_ROOT של PHP).
- **--ignore-errors--**: כפיית הפעולה גם אם התגלו שגיאות.

pear search 10.6.9

אם תרצו להתקין חבילה אך אינכם זוכרים את שמה המדויק, או אם אתם תוהים אם קיימת חבילה שמבצעת פעולה מסוימת (למשל X), תוכלו לחפש אותה. הפקודה מבצעת חיפוש של מחרוזת בתוך שמות החבילות.

לדוגמה, חיפוש חבילות הקשורות ל-XML:

pear search xml \$

הפלט מציג ארבע עמודות: שם החבילה, הגרסה העדכנית ביותר ברשת, הגרסה המותקנת מקומית (ריק אם אינה מותקנת), ותיאור קצר.

pear remote-list 10.6.10

פקודה זו מציגה רשימה של כל החבילות והגרסאות היציבות הזמינות במאגר החבילות המרוחק:

```
pear remote-list $
```

ההבדל מ-`list-all` הוא ש-`remote-list` מציגה רק את הגרסה האחרונה הזמינה, בעוד ש-`list-all` מציגה גם את מה שמוחקן אצלכם. פקודה זו מצייתת להגדרת ה-`preferred_state` שלכם (שכברירת מחדל היא `stable`).

ניתן לשנות זאת זמנית עבור פקודה אחת כדי לראות גם גרסאות אלפא:

```
pear -d preferred_state=alpha remote-list $
```

pear remote-info 10.6.11

כדי להציג מידע מפורט על חבילה שטרם התקנתם, השתמשו ב-`pear remote-info`. המידע (רישיון, קטגוריה, תיאור מלא) נלקח מהגרסה החדשה ביותר של החבילה בשרת.

pear download 10.6.12

פקודת ה-`install` אינה שומרת את קובץ החבילה המורד. אם אתם זקוקים לקובץ ה-`tarball` עצמו (להתקנה מאוחרת יותר או לכל מטרה אחרת), השתמשו ב-`download`:

```
pear download DB $
```

pear config-get / 10.6.14 pear config-set 10.6.13

כפי שראינו, פקודות אלו משמשות לצפייה ושינוי של פרמטרי ההגדרות.

- `config-get`: מציג ערך (ניתן לציין שכבה ספציפית: `user`, `system` או `default`).
- `config-set`: משנה ערך. כברירת מחדל השינוי מתבצע בשכבת המשתמש (`user`).

pear config-show 10.6.15

מציג את כל הגדרות התצורה הנוכחיות המיושמות במערכת.

10.6.16 קיצורי דרך (Shortcuts)

לכל פקודה ב-PEAR יש קיצור דרך כדי לחסוך בהקלדה. ניתן לראותם באמצעות `pear help shortcuts`.

לדוגמה:

- `i` במקום `install`
 - `up` במקום `upgrade`
 - `un` במקום `uninstall`
 - `cs` במקום `config-set`
-

10.7 ממשקי מתקין (Front-Ends)

10.7.1 ממשק שורת הפקודה (CLI)

זהו הממשק שבו השתמשנו עד כה, המציג פלט טקסטואלי בטרמינל.

10.7.2 ממשק Gtk (גרפי)

PEAR מציעה גם ממשק גרפי מבוסס Gtk (נפוץ בלינוקס, אך קיים גם ל-Windows). הוא דורש התקנה של `php-gtk` ואת החבילה `PEAR_Frontend_Gtk`.

שימוש במתקין הגרפי:

מפעילים אותו עם הפקודה: `pear -G`

הממשק הגרפי מאפשר ניווט נוח בין קטגוריות, סימון חבילות להתקנה או שדרוג באמצעות תיבות סימון (checkboxes), וצפייה בהגדרות התצורה בלשוניות (tabs) מעוצבות. הקוד המבצע את ההתקנה בפועל הוא אותו קוד של גרסת שורת הפקודה; רק התצוגה משתנה.

10.8 סיכום

מטרת פרק זה הייתה להציג את התשתית של PEAR ולהראות כיצד להתקין חבילות לשימושכם. בפרק הבא נלמד על חבילות חשובות ספציפיות וכיצד לשלבן בקוד שלכם.

פרק 11

חבילות PEAR חשובות

11.1 מבוא

בפרק זה נסקור דוגמאות לכמה מחבילות ה-PEAR הפופולריות ביותר. הספר אינו יכול להכיל דוגמאות לכל חבילה וחבילה, אך פרק זה יספק לכם לפחות מבוא בסיסי לנושא.

11.2 שאילתות מסדי נתונים

ראו פרק 6, "מסדי נתונים עם PHP 5", למבוא בנושא PEAR DB.

11.3 מערכות תבניות (Template Systems)

מערכות תבניות הן רכיבי PHP המאפשרים להפריד בין הלוגיקה האפליקטיבית (Application Logic) לבין לוגיקת התצוגה (Display Logic), ומציעות פורמט תבנית פשוט יותר מ-PHP עצמה.

זהו מצב אירוני ש-PHP, שהחלה את דרכה כשפת תבניות, משמשת כיום ליישום מערכות תבניות. אך ישנן סיבות טובות לכך מעבר להפרדת הקוד מהתצוגה, כגון מתן פורמט סימון פשוט יותר למעצבי אתרים ושיפור השליטה של המפתחים ביצירת הדפים. לדוגמה, מערכת תבניות יכולה לתרגם טקסטים באופן אוטומטי או למלא טפסים בערכי ברירת מחדל.

11.3.1 טרמינולוגיה של תבניות

לפני שנצלול למערכות השונות, כדאי להכיר את המושגים המקצועיים:

טבלה 11.1: מילון מונחים לתבניות

| מילה | משמעות |

| ---: | ---: |

| Template (תבנית) | תוכנית המתאר של הפלט; מכילה "ממלאי מקום" (Placeholders) ובלוקים. |

| Compile (הידור) | התמרת התבנית לקוד PHP. |

| Placeholder (ממלא מקום) | מחרוזת תחומה המוחלפת בנתונים בזמן הריצה. |

| Block (בלוק / מקטע) | חלק בתבנית שניתן לחזור עליו עם נתונים שונים. |

HTML_Template_IT 11.3.2

מערכת התבניות הראשונה של PEAR שנכיר היא HTML_Template_IT, או בקיצור IT. זוהי החבילה הפופולרית ביותר ב-PEAR, אך היא גם האיטית ביותר מאחר שהיא מנתחת (parses) את התבניות בכל בקשה מחדש ואינה מהדרת אותן לקוד PHP.

טיפ: חבילת HTML_Template_Sigma מספקת API תואם ל-IT, אך מהדרת את התבניות לקוד PHP לביצועים טובים יותר.

11.3.2.1 Placeholder תחביר

IT משתמשת בסוגריים מסולסלים כתווי תיחום: {PageTitle}.

11.3.2.2 דוגמה: תבנית IT בסיסית

PHP

```
php?>
```

```
require_once "HTML/Template/IT.php"
```

```
$tpl = new HTML_Template_IT('./templates$
```

```
$tpl->loadTemplateFile('hello.tpl$
```

```
$tpl->setVariable('title$ 'שלום עולם!');
```

```
$tpl->setVariable('body$ 'זוהי בדיקה של HTML_Template_IT');
```

```
$tpl->show$();
```

בקוד זה אנו טוענים קובץ תבנית, מגדירים משתנים (Variables) התואמים ל-Placeholders בקובץ ה-`tpl`, ולבסוף הפונקציה `show()` מבצעת את ההחלפות ומציגה את הפלט.

HTML_Template_Flexy 11.3.3

החבילה הבאה היא HTML_Template_Flexy (או פשוט Flexy). למרות שתבניות פשוטות של IT יעבדו בה, שתי החבילות שונות מאוד. Flexy עובדת על אובייקטים ומשתני מחלקה (Members) במקום על מערכים אסוציאטיביים.

11.3.3.1 דוגמה: תבנית Flexy בסיסית

Flexy דורשת הגדרת ספריית תבניות וספריית הידור (Compile Directory) שחייבת להיות בעלת הרשאות כתיבה לשרת.

PHP

```
$tpl = new HTML_Template_Flexy(array$
    , 'templateDir' => 'templates'
    , 'compileDir' => 'compiled'
);

$tpl->compile('hello.tpl$
    ;view = new stdClass$
    ;view->title = 'שלום עולם!$
    ;$tpl->outputObject($view$
```

11.3.3.3 פורמט הסימון של Flexy

Flexy תומכת בתגים מתקדמים המאפשרים לוגיקה בסיסית בתוך התבנית:

טבלה 11.2: תגי סימון ב-Flexy

- {variable} – הצגת משתנה (כולל קידוד HTML כברירת מחדל).
- {()method} – קריאה למתודה מהאובייקט.
- {if:variable} – הצהרת תנאי פשוטה.
- {foreach:arr,val} – לולאה על מערך (מקביל ל-foreach ב-PHP).

11.3.3.4 טיפול במאפייני HTML

אחת היכולות המעניינות של Flexy היא שימוש במאפיינים (Attributes) בתוך ה-HTML עצמו לשליטה בלוגיקה, למשל:

```
<li flexy:foreach="list_entries,text">{text}</li>
```

11.4 אימות זהות (Authentication)

חבילת PEAR Auth היא שכבת אימות מופשטת המשתמשת ב-"Containers" כדי להתממשק למערכות אימות שונות: קבצי סיסמה, מסדי נתונים, LDAP, IMAP ועוד.

11.4.1 סקירה כללית

החבילה משתמשת בבקשות POST להעברת שם משתמש וסיסמה. ברגע שהאימות מצליח, Auth משתמשת ב-Sessions כדי לעקוב אחר המשתמש. היתרון הוא שהבדיקה מול מסד הנתונים (שעשויה להיות "יקרה" במשאבים) מתבצעת פעם אחת בלבד בתחילת הסשן ולא בכל בקשת HTTP.

11.4.2 דוגמה: Auth עם קובץ סיסמאות

```
PHP

require_once 'Auth.php';

// הגדרת אימות מול קובץ httpasswd.

(auth = new Auth("File", ".httpasswd", "login_function$

(auth->start$

;())if (!$auth->getAuth

// אם לא מאומת, הפונקציה login_function תופעל והסקריפט יעצר

exit;

{

echo "אתה מחובר!";
```

הקריאה ל-(auth->start) מנהלת את הסשן באופן אוטומטי, בודקת משתני POST ומציגה את טופס ההתחברות במידת הצורך.

11.4.3 דוגמה: Auth עם מסד נתונים ונתוני משתמש

דוגמה זו מרחיבה את הקוד הקודם על ידי שימוש במסד נתונים לאחסון שמות משתמשים וסיסמאות, ואינה מספקת טופס התחברות מותאם אישית אלא משתמשת בטופס מובנה.

בנוסף, תלמדו כיצד לצרף מידע נוסף הרלוונטי למשתמש לסשן ההתחברות, וכיצד ליישם סשנים שפוקעים באופן אוטומטי. כדי להבין טוב יותר כיצד נתוני ההתחברות נשמרים בסשן, להלן דוגמה לנתוני סשן של Auth:

```
PHP

)SESSION["_authsession"] = array_$
    ,()data" => array"
    ,registered" => 1"
    ,"username" => "guest"
    ,timestamp" => 1075642673"
    ,idle" => 1075643017"
```

(

משתנה ה-PHP שמחזיק את ששן ה-Auth נקרא תמיד `$_SESSION`. המפתחות בתוך מערך זה מוצגים בטבלה 11.5.

הערה: הסיסמה אינה נשמרת בסשן. אין בכך צורך — המשתמש כבר אומת. הסשן מכיל רק מידע שנשלף לאחר אימות מוצלח, ומידע שמתעדכן כל הזמן (כמו `idle`).

טבלה 11.5: משתני ששן של Auth

| שם מפתח | תיאור |

| ---: | ---: |

| data | מקום לאחסון נתונים שסופקו על ידי המשתמש, או נתונים שנטענו ממסד הנתונים. |

| registered | מוגדר תמיד כ-TRUE כאשר המשתמש מחובר. |

| username | מכיל את שם המשתמש. |

| timestamp | מכיל את הזמן (time()) בו המשתמש התחבר. |

| idle | מכיל את זמן הפעילות האחרונה בסשן. |

ראיית המבנה הזה עוזרת להבין כיצד Auth עובד. לדוגמה, כדי להפסיק התחברות לאחר N שעות, Auth בודק את המשתנה `timestamp`. כדי להפסיק התחברות לאחר N דקות של חוסר פעילות, הוא בודק את המשתנה `idle`.

11.4.4 שיקולי אבטחה ב-Auth

סוגיית האבטחה הגדולה ביותר היא ההסתמכות על PHP Sessions. מזהה הסשן (Session ID) הוא סודי, אך הוא כל מה שמשתמש זדוני צריך כדי להשתלט על חשבון.

11.4.4.1 טיפ אבטחה 1: ביטול `session.trans_sid`

תכונה זו מוסיפה את מזהה הסשן לכל קישור בדף כפרמטר GET. זה שקול להצבת שם המשתמש והסיסמה בכתובת ה-URL. קיים סיכון שהמזהה ידלוף דרך כותרת ה-HTTP Referer לאתרים חיצוניים. יש לבטל אפשרות זו כדי למנוע דליפה.

11.4.4.2 טיפ אבטחה 2: שימוש ב-Auth_HTTP

אם ברצונכם לתמוך במשתמשים ללא עוגיות (Cookies), התקינו את Auth_HTTP. חבילה זו מחליפה את טופס ההתחברות בחלון קופץ סטנדרטי של דפדפן לאימות HTTP.

11.4.4.3 טיפ אבטחה 3: שימוש ב-HTTPS

שימוש ב-HTTPS מגן על שמות משתמש, סיסמאות ומזהי ששן מפני ציתות לחבילות מידע ברשת (Sniffers).

11.4.5 שיקולי סקלריות (Scalability)

מאחר ש-Auth משתמש בסשנים המאוחסנים כברירת מחדל בקבצים מקומיים, תיתקל בבעיה באתר המבוסס בין מספר שרתים (Load Balancing). אם משתמש התחבר בשרת 1, והבקשה הבאה שלו תגיע לשרת 2, שרת 2 לא ימצא את קובץ הסשן המקומי.

פתרונות אפשריים:

1. **Load-Balancing לפי Session ID**: הבטחה שכל הבקשות מאותו סשן יגיעו תמיד לאותו שרת.
2. **אחסון סשן משותף**: שימוש במסד נתונים או מערכת כמו `msession` לשיתוף נתוני הסשן בין כל שרתי האינטרנט.

11.5 טיפול בטפסים (Form Handling)

בניית טפסי HTML ידנית היא פשוטה, אך כשעולה הצורך באימות נתונים (Validation), טפסים מרובי דפים או שימוש בתבניות, עדיף להשתמש במחולל טפסים כמו **HTML_QuickForm**.

חבילה זו מייצגת כל אלמנט בטופס כאובייקט. ניתן להגדיר חוקי אימות שיבוצעו בצד הלקוח (JavaScript) או בצד השרת באופן אוטומטי.

דוגמה לקבלת נתונים:

המתודה `validate()` מחזירה `true` אם נשלחו נתונים והם עומדים בכל החוקים שהוגדרו:

PHP

```
} if ($form->validate()) {  
    // עיבוד הנתונים  
}
```

11.6 מטמון (Caching)

PEAR מציעה שתי חבילות: **Cache** ו-**Cache_Lite**. כפי ששמה מרמז, `Cache_Lite` מתוכננת להיות מהירה וקלה יותר על חשבון גמישות מסוימת.

Cache_Lite 11.6.1

החבילה מציעה מטמון מבוסס קבצים עבור:

- נתונים כלליים.
- פלט (Output) של PHP.
- ערכי חזרה של פונקציות.

היא בנויה כך שלא תטען מחלקות PEAR נוספות אלא אם יש בכך צורך, מה ששומר על זמן תגובה (Latency) נמוך במיוחד.

11.7 סיכום

פרק זה הציג כמה מהחבילות הנפוצות ביותר ב-PEAR. הכוונה הייתה להכניס אתכם לעניינים כדי שתוכלו להמשיך לחקור בעצמכם את התיעוד המקוון בכתובת: <http://pear.php.net/manual>.

פרק 12

12.1 מבוא

בפרקים 10 ו-11 למדתם כיצד להשתמש במתקין PEAR ובחבילות PEAR בקוד שלכם. בפרק זה תלמדו כיצד לבנות חבילות PEAR משלכם — בין אם לשימוש פנימי בארגון ובין אם לפרסום ברישיון קוד פתוח דרך שרת ההפצה של PEAR.

12.2 סטנדרטים של PEAR

תקן הקידוד של PEAR (או PCS) נועד בעיקר למפתחי חבילות, אך הוא שימושי לכל מי שמשתמש בהן כדי להבין את המבנה והשמות הצפויים.

12.2.1 מתן שמות לסמלים (Naming)

- **קבועים (Constants):** אותיות גדולות בלבד, עם שם החבילה כתחילית.
 - דוגמה: `PEAR_ERROR_DIE`, `AUTH_EXPIRED`.
 - ב-PHP 5 ניתן להשתמש ב-`class const`, למשל: `Auth::EXPIRED`.
- **משתנים גלובליים:** רצוי להימנע מהם ב-PHP 5 (לטובת משתני מחלקה סטטיים). אם חייבים, התבנית היא `Package_Name_variable_$`.
- **פונקציות:** שם החבילה כתחילית, ואחריו שם הפונקציה בשיטת `camelCase` (המתחילה באות קטנה).
 - דוגמה: `Package_Name_functionName()`.
 - פונקציה "פרטית" (פנימית לחבילה) תתחיל בקו תחתון: `Package_Name_privateFunction()`.
- **מחלקות (Classes):** שם החבילה משמש כתחילית (למשל `Package_Name`). חריג נפוץ הוא מחלקות של דרייברים (כמו `DB_mysql`), שם החלק המשתנה עשוי להיות באותיות קטנות.
- **מתודות (Methods):** אות קטנה בהתחלה וכל מילה נוספת באות גדולה (למשל `toHTML()`). ב-PHP 5 יש להשתמש במילות המפתח `private/public`, בעוד שבקוד תואם PHP 4 משתמשים בקו תחתון לפני שם המתודה כדי לסמן שהיא פרטית.

12.2.2 הזחה (Indentation)

PEAR משתמשת בהזחה של ארבעה רווחים (ללא שימוש ב-Tab!). הסיבה לכך היא שתווי Tab מרונדרים באופן שונה בעורכי טקסט שונים, מה שפוגע בעקביות הקוד עבור מפתחים אחרים.

12.3 ניהול גרסאות (Versioning)

- גרסה יציבה ראשונה חייבת להיות **1.0.0**.

- גרסאות לפני היציבות יהיו בתבנית **x.0** (גרסאות פיתוח).
- **גרסה N.1** חייבת להיות תואמת לגרסה **M.1** (כאשר $N > M$).
- שינוי בגרסה הראשית (Major), למשל מ-x.1 ל-x.2 מאפשר שבירת תאימות לאחור. במקרה כזה, לעיתים משנים את שם החבילה ל-Foo2.

12.4 סביבת שורת פקודה (CLI)

PEAR מאפשרת לכלול סקריפטים של שורת פקודה בחבילות. כדי לפתור בעיות של נתיבים למנוע ה-PHP, משתמשים במשתני סביבה כמו `PHP_PEAR_PHP_BIN`. ניתן להטמיע קוד PHP בתוך סקריפטים של Shell (לינוקס) או קבצי Bat (ווינדוס) בצורה חכמה שמתעלמת מקוד המעטפת ומריצה רק את ה-PHP.

12.5 יסודות

כאשר בונים חבילה, יש להקפיד על שלושה עקרונות בשילוב קבצים:

1. **שימוש ב-require_once בלבד:** כדי למנוע שגיאות של הגדרה כפולה של מחלקות.
2. **מתאם בין שם המחלקה לשם הקובץ:** PEAR דוגלת בעיקרון של "מחלקה אחת לקובץ". שם הקובץ נגזר משם המחלקה על ידי החלפת קווים תחתונים בלוחסנים.
 ○ `XML_Parser` יימצא בנתיב `XML/Parser.php`.
3. **אנקפסולציה של Includes:** כל קובץ צריך להצהיר על המחלקות שבהן הוא תלוי.

תרשים 12.1: תלויות משורשרות (Nested dependencies).

הסמל היחיד שמתייחסים אליו ישירות בתוך `A.php` הוא המחלקה `B` מתוך `B.php`. הקובץ `cll` אינו מתייחס למחלקה `C`. למעשה, עליכם להניח ש-`A.php` כלל אינו מודע לקיומו של `C.php`. על ידי הקפדה על עיקרון זה, אינכם מסתמכים על המבנה הפנימי של חבילה `B`, שעשוי להשתנות בעתיד. זה הופך את החבילה שלכם לעמידה יותר בפני שינויים בחבילות אחרות.

12.5.2 טיפול בשגיאות

קוד PEAR מדווח ותופס שגיאות באמצעות ה-API לטיפול בשגיאות של PEAR, המפורט בפרק 7.

12.6 בניית חבילות

בחלק זה נלמד כיצד לבנות חבילות משלכם. להלן דוגמה לחבילה המכילה מחלקת PHP, סקריפט שורת פקודה, בדיקת רגרסיה וקובץ תיאור חבילה (`package.xml`).

12.6.1 דוגמת PEAR: HelloWorld

זהו המקרה המינימלי — קובץ PHP בודד המממש מחלקה בשם `HelloWorld`:

PHP

`php?>`

`**/`

* קובץ `HelloWorld.php`

```

/*
} class HelloWorld

} (function display($name = null

} (if ($name

;"print "Hello, $name!\n

} else {

;"print "Hello, World!\n

{

{

{

<?

```

בנוסף, נכין סקריפט שורת פקודה וקובץ בדיקה בתבנית .phpt. קובץ .phpt מחולק למקטעים המתחילים ב--SECTION--.

טבלה 12.2: כותרות מקטעי בדיקה

מקטע	תיאור
---	---
*TEST	תיאור קצר של הבדיקה.
*FILE	קוד הבדיקה בפועל.
*EXPECT	הפלט המדויק שהקוד אמור להדפיס.
EXPECTF	פלט צפוי הכולל "ממלאי מקום" (Placeholders).
SKIPF	קוד שקובע אם לדלג על הבדיקה.

12.6.2 בניית קובץ ה-Tarball

לאחר הכנת הקבצים, ניצור את החבילה באמצעות הפקודה:

```
pear package $
```

התוצאה תהיה קובץ בשם HelloWorld-1.0.tgz. קובץ זה ניתן להתקנה בכל מכונה שיש בה מתקין PEAR.

12.6.3 אימות (Verification)

השתמשו בפקודה `pear package-validate` (או בקיצור `pear pv`) כדי לוודא שהחבילה תקינה. האימות ייכשל אם הגדרתם סמלים מחוץ ל-`Namespace` של החבילה, אם חסרים אלמנטים ב-`package.xml`, או אם רשימת הקבצים אינה תקינה.

12.7 מבנה קובץ `package.xml`

זהו קובץ התיאור של החבילה. להלן האלמנטים העיקריים:

- `<package>`: אלמנט השורש.
- `<name>`: שם החבילה (תלוי רישיות).
- `<summary>`: תיאור בשורה אחת.
- `<description>`: תיאור מלא של החבילה.
- `<maintainers>`: רשימת המתחזקים (כולל שם, אימייל ותפקיד כמו `lead` או `developer`).
- `<release>`: מידע על הגרסה הנוכחית (תאריך, מצב כמו `stable`, והערות שחרור).

12.7.2.6 רשימת קבצים (`<filelist>`)

מכיל אלמנטים מסוג `<dir>` (ספרייה) ו-`<file>` (קובץ).

לכל קובץ ניתן להגדיר תפקיד (`Role`):

- `php`: קובץ מקור.
- `test`: בדיקת רגרסיה.
- `doc`: תיעוד.
- `script`: סקריפט להרצה.

12.8 תלויות (Dependencies)

אחד היתרונות המרכזיים בשימוש ב-PEAR הוא שימוש חוזר בקוד. עם זאת, כאשר משתמשים בקוד מחבילה אחרת, נוצרות תלויות בין החבילות. יש לציין תלויות אלו בתיאור החבילה כדי ליידע את המשתמשים.

12.8.2 האלמנט `<dep>`

אלמנט זה מתאר תלות בודדת.

- **מאפיין `type`**: סוג התלות (למשל `php` עבור גרסת `PHP`, `pkg` עבור חבילת PEAR אחרת, או `ext` עבור הרחבת PHP ספציפית).
- **מאפיין `rel`**: מגדיר את היחס לגרסה (למשל `ge` עבור "גדול או שווה ל-", `lt` עבור "קטן מ-", או `eq` עבור "שווה ל-").
- **מאפיין `optional`**: מאפשר להגדיר שתלות מסוימת אינה חובה להתקנה, אלא רק מוסיפה פונקציונליות (ערכים: `yes` או `no`).

12.8.4 סיבות להימנע מתלויות

תלויות הן מנגנון הכרחי, אך הן עלולות להפוך ל"תלויות דוהרות" (`Run-away dependencies`) – מצב שבו חבילה אחת גוררת אחריה עשרות חבילות אחרות שלא לצורך.

ניהול תלויות מורכבות גוזל זמן פיתוח יקר. אם ההבדל בין הוספת תלות לבין כתיבת הקוד בעצמכם הוא רק כמה שורות, כדאי לשקול שוב האם התלות הכרחית. בתרשים 12.2 (בספר) מוצגות "תלויות שעירות", שבהן חבילה A תלויה ב-B ו-C, אך אלו תלויות בתוך B-F ו-D-G.

12.9 החלפות מחרוזות (String Substitutions)

ניתן להגדיר החלפות המבוצעות בקבצים בזמן ההתקנה. זה שימושי לסנכרון נתיבי ברירת מחדל עם הגדרות ה-PEAR של המשתמש.

- **האלמנט <replace>**: מגדיר החלפה. למשל, ניתן להחליף תבנית כמו `@php_bin@` בנתיב האמיתי למנוע ה-PHP במכונה של המשתמש באמצעות המאפיין `"type="pear-config"`.

12.10 הכללת קוד C

חבילת PEAR יכולה לכלול קוד C או ++C. מתקין PEAR יריץ את תהליך הבנייה (build) באופן אוטומטי אם יזהה קבצים עם התפקיד `"role="src"`. ניתן להגדיר אפשרויות הגדרה (`configureoptions`) שישאלו את המשתמש שאלות בזמן ההתקנה (למשל נתיב לספריות חיצוניות).

12.12 תהליך השחרור ב-PEAR

אם אתם מפרסמים חבילה דרך `pear.php.net`, עליכם לעבור תהליך קהילתי:

1. **הצעת חבילה (Proposal)**: הגשת הצעה באתר. הקהילה דנה בשם החבילה ובנחיצותה.
2. **הצבעה**: מפתחי PEAR מצביעים על ההצעה.
3. **יצירת החבילה**: לאחר האישור, מנהל מערכת מאשר את יצירת החבילה בשרת.
4. **אריזה (Rolling a tarball)**: יצירת קובץ ה-`tgz` בעזרת הפקודה `pear package`.
5. **בדיקות (QA)**: הרצת בדיקות רגרסיה.
6. **העלאה (Upload)**: פרסום המהדורה לעולם.

12.13 אריזה (Packaging)

ה-PEAR Packager מבצע מספר פעולות אוטומטיות:

- **ניתוח קוד המקור**: בודק תלויות ומזהה מחלקות ופונקציות כדי לוודא עמידה בתקני הקידוד.
- **יצירת MD5 Checksum**: מחשב חתימה דיגיטלית לכל קובץ כדי לוודא שהקבצים לא הושחתו בזמן ההורדה.
- **עדכון package.xml**: יוצר קובץ הגדרות סופי הכולל את כל המידע שנאסף.

12.14 העלאה וסיום

לאחר הבדיקות, מעלים את החבילה לדף ה-Upload ב-PEAR. לאחר אימות המידע, החבילה מפורסמת ומופיעה ברשימת "השחרורים האחרונים". הכרזה נשלחת אוטומטית לרשימת התפוצה של הקהילה.

12.15 סיכום

פרק זה סיפק לכם את הכלים להפוך מתורמים פסיביים למפתחים פעילים בקהילת ה-PEAR, או להקים תשתית הפצה פנימית בארגון שלכם.

פרק 13

13.1 מבוא

"קידמה פירושה לא לעמוד במקום כשכל השאר זז." — וודרו וילסון

עם כל כך הרבה תכונות חדשות, במיוחד בתחום ה-Object-Oriented (תכנות מונחה עצמים), כמעט בלתי אפשרי שכל סקריפט שנכתב ב-PHP 4 יעבוד ללא שינוי ב-PHP 5. צוות הפיתוח ניסה להפוך את המעבר לקל ככל האפשר, אך ישנן אי-תאימות קטנות. פרק זה מכסה את הדברים שעלולים להישר וכיצד לתקנם.

13.2 מודל האובייקטים

ב-PHP 5 יש מודל אובייקטים חדש המשנה את האופן שבו אובייקטים מנוהלים. בחלק מהמקרים, ניתן להורות ל-PHP 5 לחזור להתנהגות של PHP 4 באמצעות "מצב תאימות" (Compatibility Mode).

13.3 העברת אובייקטים לפונקציות

אחד השינויים הגדולים ביותר הוא שאובייקטים המועברים לפונקציה אינם מועתקים יותר כברירת מחדל. ב-PHP 4, העברת אובייקט יצרה עותק שלו. ב-PHP 5, מועבר "ידיית" (Handle) לאובייקט המקורי, כך ששינוי בתוך הפונקציה משפיע על האובייקט המקורי.

פתרון: אם ברצונך לשמור על התנהגות PHP 4, עליך להשתמש באופרטור clone:

```
;(display_quoted(clone $s
```

לחלופין, ניתן להפעיל ב-php.ini את ההגדרה: zend.ze1_compatibility_mode = 1.

13.4 מצב תאימות (Compatibility Mode)

הפעלת מצב זה משפיעה על מספר דברים מעבר להעברת אובייקטים:

1. **המרה (Casting):** ב-PHP 4, המרת אובייקט ריק ל-Integer החזירה 0. ב-PHP 5 זה תמיד מחזיר 1.
2. **השוואת אובייקטים:** ב-PHP 4, האופרטור `==` החזיר אמת אם לכל התכונות היו אותם ערכים. ב-PHP 5, הוא מחזיר אמת רק אם מדובר באותו אובייקט (אותה ידיית).

13.5 שינויים נוספים

ישנם שינויים ש"מצב תאימות" לא פותר:

- **השמה ל-\$this:** ב-PHP 5 לא ניתן לבצע השמה למשתנה `this$` (למשל: `this = new$` `OtherClass`;`()`). זה יגרום לשגיאה קטלנית. הפתרון הוא לעצב מחדש את המחלקה, למשל באמצעות Singleton.
- **הפונקציה `get_class()`:** ב-PHP 4 שם המחלקה תמיד הוחזר באותיות קטנות. ב-PHP 5, השם מוחזר כפי שהוגדר (Case-preserved). כדי לשמור על תאימות, מומלץ להשתמש ב-`((strtolower(get_class($obj`

13.6 שגיאות E_STRICT

PHP 5 הציגה רמת שגיאה חדשה שנועדה להתריע על שימוש בתכונות מיושנות (Deprecated). כדי לראות, יש להגדיר `error_reporting` לערך `E_ALL | E_STRICT` (או המספר 4095).

- **יצירת אובייקטים אוטומטית:** השמה למשתנה שלא הוגדר כאובייקט (למשל `person->name = $` "X";) תעבוד אך תעלה שגיאת `E_STRICT`.
- **Constructor:** ב-PHP 5 הוכנס המבנה האחיד `__construct()`. אם קיימת גם פונקציה בשם המחלקה (סגנון ישן) וגם `__construct()`, תעלה שגיאת `E_STRICT`.
- **ירוסה:** חתימת הפונקציה (הפרמטרים שהיא מקבלת) חייבת להיות זהה במחלקת הבה ובמחלקת האב.

13.7 בעיות תאימות אחרות

- **CLI (ממשק שורת פקודה):** בגרסת הווידוס, הקובץ שונה מ-`php.exe` ל-`php-cgi.exe`.
- **MySQL:** הספרייה של MySQL **אינה כלולה יותר כברירת מחדל** בתוך PHP 5. יש להשתמש בספרייה חיצונית (או בתוסף ה-Mysqli החדש).

13.8 שינויים בפונקציות

1. `array_merge()`: ב-PHP 5 פונקציה זו מקבלת **רק מערכים**. העברת משתנה רגיל (כמו מחזרת או מספר) תגרום לשגיאת אזהרה ותחזיר מערך ריק.
2. `strrpos()` ו-`stripos()`: ב-PHP 5, חיפוש המיקום האחרון של מחזרות בתוך מחזרות משתמש בכל ה-Needle (מחזרת החיפוש), בניגוד ל-PHP 4 שהשתמש רק בתו הראשון של מחזרת החיפוש.

13.9 סיכום

המעבר ל-PHP 5 מביא איתו עוצמה רבה אך דורש תשומת לב לפרטים קטנים במודל האובייקטים ובפונקציות הליבה. שימוש ב-`E_STRICT` הוא הדרך הטובה ביותר לוודא שהקוד שלכם מוכן לעתיד.

13.1 מבוא

"קידמה פירושה לא לעמוד במקום כשכל השאר זז." — וודרו וילסון

עם כל כך הרבה תכונות חדשות, במיוחד בתחום ה-Object-Oriented (תכנות מונחה עצמים), כמעט בלתי אפשרי שכל סקריפט שנכתב ב-PHP 4 יעבוד ללא שינוי ב-PHP 5. צוות הפיתוח ניסה להפוך את המעבר לקל ככל האפשר, אך ישנן אי-תאימות קטנות. פרק זה מכסה את הדברים שעלולים להישר וכיצד לתקנם.

13.2 מודל האובייקטים

ב-PHP 5 יש מודל אובייקטים חדש המשנה את האופן שבו אובייקטים מנוהלים. בחלק מהמקרים, ניתן להורות ל-PHP 5 לחזור להתנהגות של PHP 4 באמצעות "מצב תאימות" (Compatibility Mode).

13.3 העברת אובייקטים לפונקציות

אחד השינויים הגדולים ביותר הוא שאובייקטים המועברים לפונקציה **אינם מועתקים יותר כברירת מחדל**. ב-PHP 4, העברת אובייקט יצרה עותק שלו. ב-PHP 5, מועבר "ידיית" (Handle) לאובייקט המקורי, כך ששינוי בתוך הפונקציה משפיע על האובייקט המקורי.

פתרון: אם ברצונך לשמור על התנהגות PHP 4, עליך להשתמש באופרטור clone:

```
;(display_quoted(clone $s
```

לחלופין, ניתן להפעיל ב-php.ini את ההגדרה: zend.ze1_compatibility_mode = 1.

13.4 מצב תאימות (Compatibility Mode)

הפעלת מצב זה משפיעה על מספר דברים מעבר להעברת אובייקטים:

1. **המרה (Casting):** ב-PHP 4, המרת אובייקט ריק ל-Integer החזירה 0. ב-PHP 5 זה תמיד מחזיר 1.
2. **השוואת אובייקטים:** ב-PHP 4, האופרטור == החזיר אמת אם לכל התכונות היו אותם ערכים. ב-PHP 5, הוא מחזיר אמת רק אם מדובר באותו אובייקט (אותה ידית).

13.5 שינויים נוספים

ישנם שינויים ש"מצב תאימות" לא פותר:

- **השמה ל-\$this:** ב-PHP 5 לא ניתן לבצע השמה למשתנה `this$` (למשל: `this = new$` `OtherClass`); זה יגרום לשגיאה קטלנית. הפתרון הוא לעצב מחדש את המחלקה, למשל באמצעות Singleton.
- **הפונקציה `get_class()`:** ב-PHP 4 שם המחלקה תמיד הוחזר באותיות קטנות. ב-PHP 5, השם מוחזר כפי שהוגדר (Case-preserved). כדי לשמור על תאימות, מומלץ להשתמש ב-`strtolower(get_class($obj))`.

13.6 שגיאות E_STRICT

PHP 5 הציגה רמת שגיאה חדשה שנועדה להתריע על שימוש בתכונות מיושנות (Deprecated). כדי לראותן, יש להגדיר `error_reporting` לערך `E_ALL | E_STRICT` (או המספר 4095).

- **יצירת אובייקטים אוטומטית:** השמה למשתנה שלא הוגדר כאובייקט (למשל `person->name = $` `person->name = $`) תעבוד אך תעלה שגיאת E_STRICT.
- **Constructor:** ב-PHP 5 הוכנס המבנה האחיד `__construct()`. אם קיימת גם פונקציה בשם `__construct()`, תעלה שגיאת E_STRICT.
- **ירושה:** חתימת הפונקציה (הפרמטרים שהיא מקבלת) חייבת להיות זהה במחלקת הבה ובמחלקת האב.

13.7 בעיות תאימות אחרות

- **CLI (ממשק שורת פקודה):** בגרסת הווינדוס, הקובץ שונה מ-`php.exe` ל-`php-cgi.exe`.
- **MySQL:** הספרייה של MySQL אינה כלולה יותר כברירת מחדל בתוך PHP 5. יש להשתמש בספרייה חיצונית (או בתוסף ה-MySQLi החדש).

13.8 שינויים בפונקציות

1. `array_merge()`: ב-PHP 5 פונקציה זו מקבלת רק מערכים. העברת משתנה רגיל (כמו מחרוזת או מספר) תגרום לשגיאת אזהרה ותחזיר מערך ריק.
2. `strpos()` ו-`stripos()`: ב-PHP 5, חיפוש המיקום האחרון של מחרוזת בתוך מחרוזת משתמש בכל ה-Needle (מחרוזת החיפוש), בניגוד ל-PHP 4 שהשתמש רק בתו הראשון של מחרוזת החיפוש.

13.9 סיכום

המעבר ל-PHP 5 מביא איתו עוצמה רבה אך דורש תשומת לב לפרטים קטנים במודל האובייקטים ובפונקציות הליבה. שימוש ב-`E_STRICT` הוא הדרך הטובה ביותר לוודא שהקוד שלכם מוכן לעתיד.

פרק 14

14.1 מבוא

לכל אפליקציה יש יעדי ביצועים. תמיד יהיו מגבלות משאבים כמו מעבד (CPU), זיכרון, רוחב פס של הדיסק ועוד. אם האתר שלכם צפוי לתנועה משמעותית (מיליוני צפיות ביום), כדאי להקדיש זמן לכיוון ביצועים (Performance Tuning).

פרק זה ילמד אתכם כיצד:

- לעצב אפליקציות PHP בעלות ביצועים גבוהים.
- להשתמש בסוגים שונים של שיטות מטמון (Caching).
- לבצע ניתוח ביצועים (Profiling) לקוד PHP.
- לייעל את הקוד ואת מסד הנתונים.
- לבצע אופטימיזציה לשרת האינטרנט ולמערכת ההפעלה.

14.2 עיצוב לביצועים

התכנון לביצועים צריך להתחיל כבר בשלב העיצוב. עדיף להימנע מאופטימיזציה מאוחרת של הקוד, שעלולה להוביל לבאגים או לקוד שקשה לתחזק.

14.2.1 טיפ עיצוב #1: היזהרו ממצבים (State)

כדי לאפשר לאפליקציה שלכם לגדול (Scaling), עדיף להימנע ככל האפשר משמירת "מצב" בצד השרת בין בקשות. "מצב" הוא מידע שעובר מבקשה אחת לבאה אחריה (כמו שם משתמש או התקדמות בטופס).

ניהול Sessions: כברירת מחדל, PHP שומרת מידע על ה-Session בקבצים מקומיים. הבעיה מתחילה כשיש יותר משרת אחד.

כפי שמוצג בתרשים 14.1 (בספר), אם משתמש נשלח לשרת א' בבקשה הראשונה ולשרת ב' בשנייה, המידע שלו לא יימצא בשרת השני.

פתרון: בידוד הנתונים. ניתן לשמור את נתוני ה-Session במסד נתונים מרכזי או בשרת ייעודי. זה מאפשר להוסיף שרתי אינטרנט ללא חשש לאובדן מידע.

14.2.2 טיפ עיצוב #2: שימוש במטמון (!Cache)

שימוש במטמון הוא דרך מצוינת לקצר את זמן התגובה. כדאי לתכנן את האפליקציה בשכבות כדי שניתן יהיה להוסיף מטמון בקלות.

ניתן להוסיף מטמון בין כל שכבה ושכבה:

1. **מטמון שאילתות מסד נתונים:** שמירת תוצאות של שאילתות יקרות.
2. **מטמון קריאות לפונקציות:** שמירת ערך החזרה של פונקציה עבור פרמטרים מסוימים.
3. **תבניות מקומפלות (Compiled Templates):** רוב מערכות התבניות הופכות אותן לקוד PHP שניתן לשמור במטמון אופקוד (Opcode).
4. **מטמון פלט (Output Caching):** שמירת הפלט המודפס של סקריפט שלם או חלקים ממנו.

14.2.3 טיפ עיצוב #3: אל תגזימו בעיצוב (Over Design)

עם תכונות ה-OO החדשות של PHP 5, קל להתפתות לעיצובים מורכבים מדי.

- **עטיפת פונקציות מובנות:** אל תעטפו פונקציות מובנות של PHP במחלקות אלא אם כן זה מוסיף ערך אמיתי (כמו יצירת API אחיד למסדי נתונים שונים).
- **הכללה (Generalize) בזהירות:** אל תיצרו שכבות הפשטה סתם כך. זה מוסיף מורכבות ומאט את המערכת.
- **PHP היא לא Java:** ב-Java יצירת אובייקטים מהירה מאוד. ב-PHP עדיף להשתמש בטיפוסים המובנים (כמו מערכים) במקום לנסות לחקות מחלקות מורכבות מ-Java. השתמשו בחזקות של PHP.

14.3 בדיקות ביצועים (Benchmarking)

מה שחשוב בסופו של דבר הוא איך האתר שלכם מתפקד באופן כללי. דרך יעילה לבדיקת תכנונים ואיתור צווארי בקבוק היא להריץ בדיקות המדמות תעבורת אמת (Production).

14.3.1 שימוש ב-ApacheBench

כלי ה-ab (קיצור של Apache Benchmarking tool) מגיע בדרך כלל עם שרת ה-Apache. הוא פועל על ידי סימולציה של מספר לקוחות השולחים בקשות לשרת ב-URL ספציפי.

לדוגמה: `ab -n 10000 -c 10 http://localhost/test.php $`

פקודה זו תירה 10,000 שאילתות, 10 בכל פעם. בסיום, הכלי יציג נתונים כמו בקשות לשנייה (Throughput) וזמני תגובה באחוזונים (למשל, כמה זמן לקח ל-99% מהבקשות להסתיים).

14.3.2 שימוש ב-Siege

החולשה של **ab** היא שהוא לא מאפשר להגדיר התפלגות בקשות מציאותית (הוא "דופק" על אותו URL שוב ושוב). כלי ה-Siege מאפשר להגדיר קובץ עם רשימת כתובות URL מלאות ולבחור מהן באקראי.

14.3.3 בדיקה מול תעבורה אמיתית

הסכנה בבדיקות אלו היא שהן לא תמיד משקפות את העולם האמיתי (גולשים עם מודמים איטיים, בוטים של מנועי חיפוש וכדומה). מומלץ ליצור קובץ בקשות המבוסס על לוגים (Logs) אמיתיים של האתר.

14.4 ניתוח ביצועים עם ה-Profiler של Zend Studio

אופטימיזציה צריכה להתמקד רק בחלקים שגוזלים את רוב המשאבים. השקעת זמן בשיפור קוד שמהווה רק שבריר מהעומס היא בזבזת זמן, ולעיתים אף פוגעת בקריאות הקוד.

ה-Profiler של Zend Studio מספק מידע קריטי: אילו חלקים לוקחים הכי הרבה זמן, כמה פעמים כל פונקציה נקראה ואיך נראה ה-"Call Trace" (עץ הקריאות).

כפי שניתן לראות בתרשים 14.6 (בספר), אם קובץ מסוים אחראי ל-53% מהעומס, עדיף להתרכז בו מאשר בקבצים אחרים.

14.5 ניתוח ביצועים עם APD

APD (Advanced PHP Debugger) הוא תוסף Zend שאוסף סטטיסטיקות בזמן ריצה ושומר אותן לקובץ. לאחר מכן ניתן לנתח את הקובץ עם הכלי pprof.

14.5.1 התקנה והרצה

מתקינים דרך PECL (pear install apd) ומגדירים ב-php.ini. כדי להתחיל איסוף נתונים בסקריפט, קוראים לפונקציה apd_set_pprof_trace().

- **טיפ:** ניתן להפעיל זאת סלקטיבית רק כשמוסיפים פרמטר ל-URL (כמו ?profile=apd_), אך יש להיזהר מחשיפת אפשרות זו בשרתי ייצור.

14.5.2 ניתוח נתוני המעקב (Trace)

בניתוח של אתרים כמו pear.php.net, התגלה למשל ש-require_once צורך 50% מהזמן. זהו סימן מובהק לכך ששימוש ב-Opcode Cache יחתוך את זמן הריצה בחצי.

ב-PHP 5, ניתן לייעל קבועים על ידי הפיכתם ל-const בתוך מחלקה במקום להשתמש ב-define(), כיוון שקבועי מחלקה נשמרים במטמון האופקוד ואינם מחושבים מחדש בכל בקשה.

14.6 ניתוח ביצועים עם Xdebug

Xdebug הוא תוסף נוסף לאיסוף נתונים. בעוד APD מתמקד בעיקר בביצועים, Xdebug מתמקד גם בניפוי שגיאות (Debugging). הוא מאפשר לייצר נתונים בפורמט Cachegrind, שאותם ניתן להציג בצורה גרפית מרשימה בעזרת תוכנות כמו KCachegrind.

14.6.1 התקנת Xdebug

בדומה ל-APD, ניתן להתקין את Xdebug מ-PECL על ידי הרצת הפקודה pear install xdebug. לאחר ההתקנה, עליכם לטעון את Xdebug לתוך Zend ולהגדיר אותו בהתאם למשימה. להלן דוגמה להגדרה ב-php.ini לטעינת Xdebug:

```
zend_extension = "/usr/lib/php/extensions/20040412/xdebug.so"
```

או עבור שרתי אינטרנט מרובי-תהליכונים (Apache על Windows או IIS):

```
zend_extension_ts = "c:/php5/extensions/xdebug.dll"
```

הגדרת Xdebug תלויה במטרה אותה תרצו להשיג.

14.6.2 מעקב אחר הרצת סקריפט (Tracing)

מעקב אחר קריאות לפונקציות במהלך הרצת סקריפט מאפשר לכם לבחון אילו פונקציות נקראו לפי הסדר, כולל פרמטרים אופציונליים וערכי חזרה. קובץ המעקב (Trace) כולל לא רק את הקריאות עצמן, אלא גם מידע על תזמון ושימוש בזיכרון. הגדרות התצורה האופטימליות ליצירת קבצי מעקב מוצגות בטבלה 14.2.

טבלה 14.2: הגדרות תצורה אופטימליות למעקב הרצה

הגדרה	תיאור
<code>xdebug.extended_info = 0</code>	כאשר מופעל, טביעת הרגל של הזיכרון גדלה בכ-33% והביצועים מואטים.
<code>xdebug.auto_trace = 1</code>	הפעלת מעקב אוטומטי אחר סקריפטים.
<code>xdebug.trace_output_dir = /tmp/xdebug</code>	הגדרת ספריית הפלט לקבצי המעקב.
<code>xdebug.collect_includes = 1</code>	הכללת שמות הקבצים בקריאות <code>include/require</code> .
<code>xdebug.show_mem_delta = 1</code>	הצגת ההבדל בשימוש בזיכרון בין כל קריאה לפונקציה.
<code>xdebug.collect_return = 1</code>	איסוף ערכי חזרה של פונקציות.
<code>xdebug.collect_params = 1</code>	איסוף פרמטרים המועברים לפונקציות.

הערה: קבצי מעקב עלולים להפוך לעצומים (מעל 100MB) בסקריפטים מורכבים. ודאו שיש לכם מספיק שטח דיסק פנוי.

כאשר כל ההגדרות מוכנות, Xdebug מייצר קובץ בספרייה שהוגדרה בשם כגון `trace.480204079.xt`. איור 14.9 מציג קובץ מעקב כזה.

כל שורה מתחילה במדד זמן, כמות הזיכרון שבשימוש, והשינוי בזיכרון לעומת השורה הקודמת. ההזחה (Indentation) מראה את הקשר בין הקריאות, ואחריהן מופיעים שם הפונקציה, הפרמטרים, שם הקובץ ומספר השורה.

14.6.3 שימוש ב-KCachegrind

בעוד שמעקב (Trace) שימושי לניתוח פשוט, הוא נועד יותר ככלי לניפוי שגיאות (Debugging). עבור ניתוח ביצועים טהור (Xdebug), Profiling מציע פונקציונליות ייעודית שאת תוצאותיה ניתן לנתח בתוכנת KCachegrind.

לאחר טעינת הקובץ ל-KCachegrind, יוצג מסך הדומה לאיור 14.10:

החלונית השמאלית מציגה את כל הפונקציות ממוינות לפי הזמן שהושקע בהן. בבחירת פונקציה, ניתן לראות בחלונית הימנית-עליונה מי קרא לה, ובחלונית הימנית-תחתונה אילו פונקציות היא קראה.

כרטיסיית ה-**Call Map** מציגה דיאגרמה ויזואלית של הזמן המושקע בפונקציות. ככל שהשטח של פונקציה גדול יותר, כך הושקע בה זמן רב יותר.

14.7 שימוש ב-APC (Advanced PHP Cache)

אחת מבעיות הביצועים הגדולות ב-PHP היא שהזמן הנדרש לעיבוד הקוד גדל ככל שהקוד מורכב יותר. הפתרון לכך הוא **מטמון אופקוד (Opcode Cache)**. הוא שומר את הפלט של מהדר ה-Zend בזיכרון משותף, כך שבקשות עוקבות אינן צריכות לנתח את אותו הקוד שוב ושוב.

APC הוא מטמון פופולרי בקוד פתוח הזמין דרך PECL:

```
shell$ pear install apc
```

כדי להשתמש בו, הוסיפו ל-`php.ini`:

```
Ini, TOML
```

```
apc.enable = yes
```

```
apc.shm_size = 4
```

לאחר הפעלה מחדש של שרת האינטרנט, תבחינו שקריאות ל-`require/include` כמעט נעלמות מרשימת צרכני המעבד בביצוע ניתוח ביצועים (Profiling).

14.8 שימוש ב-ZPS (Zend Performance Suite)

ZPS הוא מוצר מסחרי מבית Zend.com המספק כלים ל:

- **אופטימיזציה אוטומטית:** שיפור של כ-20% בביצועים ללא שינוי קוד.
- **מטמון קוד מקומפל (Compiled-code Caching):** שיפור של 50% עד 300% בביצועים.
- **מטמון תוכן (Content-Caching):** האצה של עד פי 100 (10,000%) על ידי ביטול תקורה של ביצוע הקוד.
- **דחיסת תוכן:** צמצום נפח ה-HTML בערך ב-90% לשיפור מהירות הטעינה אצל המשתמש.

14.8.1 אופטימיזציה אוטומטית

PHP פועלת בשני שלבים: הידור הקוד לקוד ביניים (Intermediate code) וביצועו.

לדוגמה, הקוד `++i$`; מומר לפעולות פשוטות. לפעמים נוצרות פעולות מיותרות (כמו שמירת ערך זמני שאינו בשימוש). ה-Zend Optimizer מנתח את קוד הביניים ומחליף תבניות לא יעילות ביעילות יותר (למשל, הפיכת post-increment ל-pre-increment במידת האפשר). התהליך קורה בזיכרון ואינו משנה את קוד המקור.

14.8.2 מטמון קוד מקומפל

במצב רגיל, בסיום הרצת סקריפט, קוד הביניים מושמד. בביקור הבא, PHP תהדר אותו מחדש. באתרים פופולריים זהו צוואר בקבוק.

מודול ה-Acceleration של ZPS שומר את קוד הביניים בזיכרון משותף (Shared Memory). בפעם הבאה שהקובץ יידרש, המנוע ידלג על שלב ההידור ויעבור ישירות לביצוע. זה יעיל במיוחד באפליקציות OO (מונחות עצמים) הכוללות קבצים רבים.

14.8.3 מטמון תוכן דינמי

זו השיטה היעילה ביותר: שמירת תוצאת ה-HTML הסופית ושליחתה כפי שהיא לבקשות הבאות.

- **TTL (זמן חיים):** הגדרת משך הזמן שהעותק במטמון תקף.
- **תלויות (Dependencies):** המטמון חייב לדעת להבחין בין פרמטרים שונים (למשל `id=7` לעומת `id=5`).
- **מטמון חלקי/בלעדי:** ניתן להחריג אזורים אישיים בדף (כמו שם המשתמש) או להגיש מטמון רק למשתמשים שאינם מחוברים (Exclusive caching).

14.9 אופטימיזציה של הקוד

14.9.1 מיקרו-בנצ'מרק (Micro-Benchmarks)

לפעמים עולה השאלה מה מהיר יותר: למשל `str_replace()` או `preg_replace()`. ניתן לכתוב סקריפט קטן המודד מחזורי מעבד בעזרת פונקציית `getrusage()`.

חשוב: יש להריץ בדיקות אלו פעמים רבות כדי לקבל ממוצע אמין, שכן רזולוציית המדידה תלויה במערכת ההפעלה.

14.9.2 פרוצדורלי לעומת מונחה עצמים (OO)

בדיקות מראות שקריאות לפונקציות רגילות (Procedural) מהירות בכ-11-12% מקריאות למתודות בתוך אובייקטים. זה הופך לגורם משמעותי רק אם הקוד כולל המון פונקציות קטנות מאוד, שבהן תקופת ה"קריאה" (Call overhead) מהווה חלק ניכר מזמן הריצה הכולל.

14.10 סיכום

עיצוב אפליקציות לביצועים גבוהים הוא נושא מורכב. המפתח הוא לשאוף לעיצוב רזה, יעיל ואלגנטי, תוך ביצוע בדיקות ביצועים (Profiling & Benchmarking) ללא הרף כדי למצוא את צווארי הבקבוק האמיתיים.

פרק 15

15.1 מבוא

אחת הסיבות העיקריות להצלחתה של PHP היא הכמות העצומה של ההרחבות (Extensions) הזמינות עבורה. לא משנה מה מפתח אינטרנט עשוי להזדקק לו, רוב הסיכויים שהוא ימצא זאת בהפצה של PHP: תמיכה במסדי נתונים שונים, פורמטים גרפיים, דחיסה, טכנולוגיות XML ועוד.

הפריצה הגדולה של PHP התרחשה בגרסה 3 עם הצגת ה-API להרחבות, שאיפשר לקהילת המפתחים להרחיב את השפה בקלות. הרעיון היה להסתייג ככל האפשר את הקרביים של PHP ושל מנוע התסריט (Scripting Engine) מכותב ההרחבה, ולדרוש ממנו רק מיומנות ב-API עצמו.

ישנן שתי סיבות עיקריות לכתיבת הרחבה משלך:

1. **תמיכה בטכנולוגיה חדשה:** למשל, אם יוצא לשוק מסד נתונים חדש בשם FooBase, יהיה עליך ליצור הרחבת PHP שעוטפת את ספריית ה-C של FooBase כדי לאפשר ממשק אליה מתוך PHP.
2. **ביצועים או פונקציונליות:** כתיבת לוגיקה עסקית ב-C כדי להשיג מהירות גבוהה יותר.

פרק זה ילמד אתכם כיצד ליצור הרחבות פשוטות באמצעות סקריפט UNIX בשם `ext_skel` (או `ext_skel_win32.php` לחלונות), שיוצר שלד של הרחבה מתוך קובץ הגדרות.

15.2 מדריך מהיר (Quickstart)

במקום להסביר לאט את כל אבני הבניין, נצלול ישר לקוד.

תארו לעצמכם שאתם זקוקים לפונקציה שמשכפלת מחרוזת \$n\$ פעמים. ב-PHP זה נראה כך:

```
PHP
function self_concat($string, $n)
{
    $result = "";
    for ($i = 0; $i < $n; $i++)
    {
        $result .= $string;
    }
    return $result;
}
```


אם עליכם לקרוא לפונקציה זו לעיתים קרובות עם מחרוזות ארוכות מאוד וערכי \$n\$ גדולים, הדבר יגרום להרבה הקצאות זיכרון ויאט את המערכת. ב-C נוכל להקצות מראש את כל הזיכרון הנדרש ולבצע זאת הרבה יותר מהר.

שלב 1: הגדרת הפונקציה

ניצור קובץ בשם myfunctions.def עם השורה הבאה:

```
(string self_concat(string str, int n
```

שלב 2: יצירת השלד

נריץ את הסקריפט:

```
ext_skel --extname=myfunctions --proto=myfunctions.def/.
```

זה ייצור ספרייה בשם myfunctions/ תחת ספריית ext/.

שלב 3: קימפול

כדי ש-PHP יכיר בהרחבה, יש לערוך את הקובץ config.m4 ולהסיר את סימני ההערה משורות ה-PHP_ARG_ENABLE. לאחר מכן יש להריץ ./configure --enable-myfunctions/, buildconf, ולבסוף make.

שלב 4: כתיבת הלוגיקה ב-C

הסקריפט יצר עבורנו קוד התחלתי ב-C. הפונקציה המרכזית להבנת הקלט היא zend_parse_parameters().

טבלה 15.1: מזהי טיפוסים (Type Specifiers)

מזהה	טיפוס C תואם	תיאור
l	long	מספר שלם (Integer).
d	double	מספר עשרוני (Floating-point).
s	char *, int	מחרוזת בינארית (כולל אורך).
b	zend_bool	ערך בוליאני (1 או 0).

a	* zval	מערך אסוציאטיבי.
---	--------	------------------

הסבר על ה-zval:

ה-zval הוא כלי קיבול הערכים של מנוע ה-Zend. בין אם הערך הוא בוליאני, מחרוזת או אובייקט, המידע שלו נשמר בתוך מבנה ה-zval.

בדוגמה שלנו, כדי לקבל מחרוזת ומספר שלם, נשתמש ב-"sl":

```
C
(if (zend_parse_parameters(argc TSRMLS_CC, "sl", &str, &str_len, &n) == FAILURE
;return
```

שימו לב ש-s דורש שני ארגומנטים ב-C: מצביע למחרוזת (*char) ומשתנה לאורך שלה (int). תמיד עדיף להשתמש באורך המחרוזת כדי שהקוד יהיה "Binary Safe" (בטוח לשימוש עם נתונים בינאריים שיכולים להכיל תווים ריקים).

15.2.1 ניהול זיכרון

ה-API של PHP להקצאת זיכרון מהערימה (Heap) כמעט זהה ל-API הסטנדרטי של שפת C. בעת כתיבת הרחבות, השתמשו בפונקציות ה-API הבאות התואמות למקבילותיהן ב-C:

- ;(emalloc(size_t size
- ;(efree(void *ptr
- ;(ecalloc(size_t nmemb, size_t size
- ;(erealloc(void *ptr, size_t size
- ;(estrdup(const char *s
- ;(estrndup(const char *s, unsigned int length

הפונקציה (estrndup) היא היחידה המיוחדת ל-PHP. היא מתנהגת כמו (estrdup), אך מאפשרת לציין את אורך המחרוזת לשכפול (ללא ה-null המסיים), ולכן היא **Binary Safe** (בטוחה לשימוש בינארי). מומלץ להשתמש בה על פני (estrdup).

יתרונות השימוש בפונקציות אלו:

1. **מניעת דליפות זיכרון:** כל זיכרון שהוקצה דרכן ולא שוחרר בטעות, ישוחרר אוטומטית בסיום הבקשה (Request).
2. **יציבות:** PHP עלולה להתרסק אם תחזירו למנוע התסריט ערכים שלא הוקצו עם פונקציות אלו.
3. **ביצועים:** שיפור בביצועים בסביבות מרובות-תהליכונים (Multi-threaded) וזיהוי השחתת זיכרון במצב Debug.
4. **בטיחות:** אין צורך לבדוק אם חזר NULL; במקרה של כשל בהקצאה, הפונקציות יפסיקו את הריצה עם שגיאת E_ERROR.

15.2.2 החזרת ערכים מפונקציות PHP

ה-API כולל אוסף עשיר של מאקרואים (Macros) להחזרת ערכים, המגיעים בשני סגנונות:

1. `RETVAL_type()`: קובע את ערך החזרה אך ממשיך בביצוע קוד ה-C (שימושי לניקוי משאבים לפני סיום). לסיים יש להשתמש ב-`return`;
2. `RETURN_type()`: קובע את הערך וגם מחזיר מיד את השליטה ל-PHP.

טבלה 15.2: מאקרואים לערכי חזרה

סוג ערך	מאקרו (קביעה וסיום)	הערות
Long	<code>(RETURN_LONG(l</code>	מספר שלם.
Boolean	<code>(RETURN_BOOL(b</code>	ערך אמת/שקר.
String	<code>(RETURN_STRING(s, dup</code>	מחרוזת. אם <code>dup</code> הוא 1, המנוע ישכפל את המחרוזת.
String (אורך)	<code>(RETURN_STRINGL(s, l, dup</code>	מחרוזת באורך <code>l</code> . מהיר יותר אם האורך ידוע מראש.
True / False	<code>RETURN_TRUE</code> / <code>RETURN_FALSE</code>	ללא סוגריים.
Resource	<code>(RETURN_RESOURCE(r</code>	מזהה משאב (כמו חיבור למסד נתונים).

15.2.3 השלמת הפונקציה `self_concat()`

להלן הקוד המלא של הפונקציה המממשת שכפול מחרוזת ב-C:

```
C
(PHP_FUNCTION(self_concat
}
;char *str = NULL
;()int argc = ZEND_NUM_ARGS
;int str_len
```

```

;long n
/* מצביע למחרוזת התוצאה */ ;char *result
/* מצביע למיקום הבא להעתקה */ ;char *ptr
/* אורך התוצאה */ ;int result_length

(if (zend_parse_parameters(argc TSRMLS_CC, "sl", &str, &str_len, &n) == FAILURE
;return

;(result_length = (str_len * n
/* הקצאת זיכרון */ ;(result = (char *) emalloc(result_length + 1
;ptr = result

} (--while (n
/* העתקת המחרוזת */ ;(memcpy(ptr, str, str_len
;ptr += str_len
{

/* תמיד סיימו מחרוזת ב-null */ ;ptr = '\0'

/* החזרה ל-PHP ללא שכפול נוסף */ ;(RETURN_STRINGL(result, result_length, 0
{

```

15.2.5 עטיפת ספריות צד-שלישי

המוטיבציה הנפוצה ביותר לכתיבת הרחבות היא עטיפת ספריות C קיימות (כמו MySQL, ImageMagick או GD). בהמשך נדגים זאת על ידי עטיפת פונקציות הקבצים הסטנדרטיות של C (כמו `fopen`).

ההרחבה תשתמש בטיפוס נתונים מופשט שנקרא **Resource** כדי לייצג את הקובץ הפתוח (`FILE`). מנוע ה-PHP אינו מבין מצביעי C ישירות, ולכן "משאבים" הם הדרך לקשר ביניהם.

הממשק שתכננו להרחבה החדשה (`myfile.def`):

- (resource file_open(string filename, string mode)
- bool file_close(resource filehandle)¹
- string file_read(resource filehandle, int size)²
- bool file_write(resource filehandle, string buffer)³
- bool file_eof(resource filehandle)⁴

לאחר הרצת `ext_skel`, תקבלו קוד ראשוני. ייתכן שתראו שגי⁵ אות קומפילציה בשורות הכוללות את המאקרו `FETCH_RESOURCE()`, שכן הסקריפט אינו יודע להשלים אותן לבדו—עלינו להגדיר לו כיצד לטפל במשאב החדש.

15.2.5.2 משאבים (Resources)

משאב הוא ערך מופשט שיכול להכיל כל סוג של מידע. כפי שצוין בעבר, מידע זה מורכב לעיתים קרובות מנתונים כגון מצביעי קבצים (file handles), מבני התקשרות למסדי נתונים וטיפוסים מורכבים אחרים.

הסיבה העיקרית לשימוש במשאבים היא שהם מנוהלים באמצעות רשימה מרכזית המשמדה אוטומטית את המשאב במקרה שמפתח ה-PHP לא עשה זאת במפורש בסקריפט שלו.

לדוגמה, חשבו על כתיבת סקריפט הפותח חיבור ל-MYSQL באמצעות הקריאה `mysql_connect()`, אך אינו קורא ל-`mysql_close()` כדי לסגור אותו ברגע שמשאב החיבור למסד הנתונים אינו בשימוש יותר. ב-PHP, מנגנון המשאבים מזהה מתי יש להשמיד משאב זה, וישמיד אותו (לכל המאוחר) בסוף הבקשה הנוכחית, ולעיתים קרובות הרבה לפני כן. הדבר מספק מנגנון חסין לביטול האפשרות לדליפות משאבים. ללא מנגנון כזה, לאחר מספר בקשות אינטרנט, שרת האינטרנט עלול לסבול מדליפה פוטנציאלית של משאבים רבים, מה שעלול להוביל לקריסת שרת או לתפקוד לקוי.

15.2.5.3 רישום טיפוסים משאבים

כיצד משתמשים במשאבים? מנוע ה-Zend הפך את העבודה עם משאבים לקלה יחסית. הדבר הראשון שעליכם לעשות הוא לרשום את טיפוס המשאב שלכם במנוע. פונקציית ה-API לשימוש היא:

```
int zend_register_list_destructors_ex(rsrc_dtor_func_t ld, rsrc_dtor_func_t pld, char
                                     (*type_name, int module_number
```

הפונקציה מחזירה מזהה (ID) של טיפוס המשאב, אותו ההרחבה צריכה לשמור במשתנה גלובלי והוא יועבר לקריאות API אחרות של משאבים בעת הצורך. `ld` היא פונקציית ה-destructor (המשמיד) שצריכה להיקרא עבור המשאב הזה. `pld` משמש עבור משאבים קבועים (persistent) שיכולים לשרוד בין בקשות ולא יכוסו בפרק זה. `type_name` הוא מחרוזת עם שם תיאורי לטיפוס. `module_number` משמש פנימית את המנוע, וכאשר נקרא לפונקציה זו, פשוט נעביר את המשתנה `module_number` שכבר הוגדר.

בחזרה לדוגמה שלנו: נוסיף את הקוד הבא לקובץ המקור `myfile.c`. הוא כולל את ההגדרה לפונקציית ה-destructor שמועברת לפונקציית הרישום `zend_register_list_destructors_ex()` (יש להוסיף אותה בשלב מוקדם בקובץ כדי שתהיה מוגדרת בזמן הקריאה לרישום):

```
C
static void myfile_dtor(zend_rsrc_list_entry *rsrc TSRMLS_DC
{
    FILE *fp = (FILE *) rsrc->ptr
    fclose(fp)
}
```

לאחר הוספת שורת הרישום לפונקציית ה-PHP_MINIT_FUNCTION שנוצרה אוטומטית, היא אמורה להיראות דומה לזה:

```
C
(PHP_MINIT_FUNCTION(myfile
```

```

    }

    /* אם יש לכם רשומות INI, הסירו את הערות מהשורות הללו
    ;(ZEND_INIT_MODULE_GLOBALS(myfile, php_myfile_init_globals, NULL
    ;())REGISTER_INI_ENTRIES

    /*

le_myfile = zend_register_list_destructors_ex(myfile_dtor, NULL, "standard-c-file",
                                              ;(module_number

                                              ;return SUCCESS

    {

```

*שימו לב ש-le_myfile הוא משתנה גלובלי שכבר הוגדר על ידי הסקריפט ext_skel.

`()PHP_MINIT_FUNCTION` היא פונקציית הסטארט-אפ של המודול (ההרחבה) שהיא חלק מ-API החשוף להרחבה שלכם. טבלה 15.3 נותנת סקירה קצרה של הפונקציות הזמינות וכיצד ניתן להשתמש בהן.

טבלה 15.3: מאקרואים להצגת פונקציות

מאקרו הצגת פונקציה	סמנטיקה
<code>()PHP_MINIT_FUNCTION</code>	פונקציית הסטארט-אפ של המודול נקראת על ידי המנוע כאשר PHP נטען ומאפשרת לבצע אתחולים חד-פעמיים נחוצים, כגון רישום טיפוסים משאבים, רישום ערכי INI ועוד.
<code>()PHP_MSHUTDOWN_FUNCTION</code>	פונקציית הכיבוי של המודול נקראת על ידי המנוע כאשר PHP נסגר לחלוטין ומשמשת בדרך כלל לביטול רישום של רשומות INI.
<code>()PHP_RINIT_FUNCTION</code>	פונקציית הסטארט-אפ לכל בקשה נקראת בתחילת כל בקשה המטופלת על ידי PHP, ומשמשת לניהול לוגיקה לכל בקשה.

פונקציית הכיבוי לכל בקשה נקראת בסוף כל בקשה המטופלת על ידי PHP, ומשמשת לרוב לניקוי הלוגיקה של פונקציית הסטארט-אפ לכל בקשה.	<code>()PHP_RSHUTDOWN_FUNCTION</code>
פונקציית המידע של המודול נקראת במהלך פונקציית <code>phpinfo()</code> ומדפיסה את המידע של המודולים הללו.	<code>()PHP_MININFO_FUNCTION</code>

15.2.5.4 יצירה ורישום של משאבים חדשים

אנו עומדים לממש את הפונקציה `file_open()`. לאחר שנפתח את הקובץ ונקבל `*FILE`, עלינו לרשום אותו במנגנון המשאבים. המאקרו העיקרי להשגת מטרה זו הוא:

```
;(ZEND_REGISTER_RESOURCE(rsrc_result, rsrc_pointer, rsrc_type
```

ראו טבלה 15.4 להסבר על הארגומנטים של המאקרו.

טבלה 15.4: ארגומנטים של המאקרו `ZEND_REGISTER_RESOURCE`

ארגומנט המאקרו	טיפוס פרמטר
<code>rsrc_result</code>	<code>*zval</code> , אשר אמור להתעדכן עם מידע המשאב הרשום.
<code>rsrc_pointer</code>	מצביע לנתוני המשאב שלנו.
<code>rsrc_type</code>	מזהה המשאב שהתקבל בעת רישום טיפול המשאב.

15.2.5.5 פונקציות קבצים

כעת כשאתם יודעים כיצד להשתמש במאקרו `(ZEND_REGISTER_RESOURCE)`, אתם כמעט מוכנים לכתוב את `file_open()`. נותר רק נושא אחד נוסף שעלינו לכסות.

מכיוון ש-PHP רץ גם תחת שרתים מרובי-תהליכונים (multi-threaded), לא ניתן להשתמש בפונקציות הגישה לקבצים הסטנדרטיות של C. זאת משום שסקריפט PHP הרץ בתהליכון אחד עלול לשנות את תיקיית העבודה הנוכחית (CWD), מה שיוביל לכך שקריאת `fopen()` המשתמשת בנתיב יחסי בתהליכון אחר תיכשל בפתיחת הקובץ המיועד. כדי למנוע בעיות כאלו, תשתית PHP מספקת מאקרואים של **VCWD** (Virtual Current Working Directory) שיש להשתמש בהם במקום כל פונקציית גישה לקבצים המסתמכת על

תיקיית העבודה הנוכחית. (טבלה 15.5 מפרטת את המקרואים הזמינים). המקרואים מתנהגים אותו הדבר כמו הפונקציות שהם מחליפים, והכל מטופל עבורכם באופן שקוף. פונקציות ספריית C סטנדרטיות שאינן זמינות בפלטפורמות מסוימות אינן נתמכות על ידי תשתית ה-VCWD. לדוגמה, `()chown`, שאינה קיימת ב-Win32, לא תכלול מאקר `()VCWD_CHOWN` תואם.

טבלה 15.5: רשימת מאקרואי VCWD

הערה	מאקרואי VCWD	ספריית C סטנדרטית
	<code>()VCWD_GETCWD</code>	<code>()getcwd</code>
	<code>()VCWD_FOPEN</code>	<code>()fopen</code>
משמש לגרסת שני הפרמטרים.	<code>()VCWD_OPEN</code>	<code>()open</code>
משמש לגרסת שלושת הפרמטרים של <code>()open</code> .	<code>()VCWD_OPEN_MODE</code>	<code>()open</code>
	<code>()VCWD_CREAT</code>	<code>()creat</code>
	<code>()VCWD_CHDIR</code>	<code>()chdir</code>
	<code>()VCWD_GETWD</code>	<code>()getwd</code>
	<code>VCWD_REALPATH()</code> ²	<code>realpath()</code> ¹
	<code>VCWD_RENAME()</code> ⁴	<code>rename()</code> ³

	VCWD_STAT() ⁶	stat() ⁵
	VCWD_LSTAT() ⁸	lstat() ⁷
	VCWD_UNLINK() ¹⁰	unlink() ⁹
	VCWD_MKDIR() ¹²	mkdir() ¹¹
	VCWD_RMDIR() ¹⁴	rmdir() ¹³
	VCWD_OPENDIR() ¹⁶	opendir() ¹⁵
	VCWD_POPEN() ¹⁸	popen() ¹⁷

טבלה 15.5 רשימת פקודות מאקרו של VCWD

טבלה 15.5: רשימת מאקרואי VCWD

מאקרו VCWD	ספריית C סטנדרטית
()VCWD_ACCESS	()access

<code>()VCWD_UTIME</code>	<code>()utime</code>
<code>()VCWD_CHMOD</code>	<code>()chmod</code>
<code>()VCWD_CHOWN</code>	<code>()chown</code>

15.2.5.6 כתיבת פונקציית ה-PHP הראשונה שלך מבוססת משאבים

מימוש `file_open()` אמור להיות קל כעת, והוא אמור להיראות כך:

```
C
(PHP_FUNCTION(file_open
}

;char *filename = NULL

;char *mode = NULL

;()int argc = ZEND_NUM_ARGS

;int filename_len

;int mode_len

;FILE *fp

,if (zend_parse_parameters(argc TSRMLS_CC, "ss", &filename
} (filename_len, &mode, &mode_len) == FAILURE&

;return

{

;fp = VCWD_FOPEN(filename, mode

} (if (fp == NULL

;RETURN_FALSE
```

{

```
;ZEND_REGISTER_RESOURCE(return_value, fp, le_myfile
```

{

ייתכן שתבחינו שהארגומנט הראשון למאקרו רישום המשאב הוא משתנה בשם `return_value`, שהופיע משום מקום. משתנה זה מוגדר אוטומטית על ידי תשתית ההרחבות והוא מסוג `zval *` המצביע לערך החזרה של הפונקציה. המאקרואים שדנו בהם קודם המשפיעים על ערך החזרה, כגון `RETURN_LONG()` ו-`RETVAL_BOOL()`, למעשה משנים את הערך של `return_value`. לכן, קל לנחש שהקוד רושם את מצביע הקובץ `fp` שקיבלנו ומגדיר את `return_value` למשאב הרשום.

15.2.5.7 גישה למשאב

כדי לגשת למשאב, עליכם להשתמש במאקרו הבא (ראו טבלה 15.6 להסבר על הארגומנטים שלו):

```
ZEND_FETCH_RESOURCE(rsrc, rsrc_type, passed_id, default_id, resource_type_name,  
;resource_type
```

טבלה 15.6: ארגומנטים של המאקרו `ZEND_FETCH_RESOURCE`

פרמטר	משמעות
rsrc	המשתנה שאליו יוקצה ערך המשאב. עליו להיות מאותו טיפול של המשאב.
rsrc_type	הטיפוס של <code>rsrc</code> שישמש להמרת המשאב פנימית לטיפוס הנכון.
passed_id	ערך המשאב לחיפוש (בתור <code>zval **</code>).
default_id	אם ערך זה אינו 1-0, מזהה זה נלקח. משמש למימוש ברירת מחדל למשאב.
resource_type_name	שם קצר לטיפוס המשאב המשמש בהודעות שגיאה.

מזהה טיפוס המשאב של המשאב הרשום.	resource_type
----------------------------------	---------------

בעזרת מאקרו זה, נוכל כעת לממש את `file_eof()`:¹

```
C
(PHP_FUNCTION(file_eof)
{
    int argc = ZEND_NUM_ARGS();
    zval *filehandle = NULL;
    FILE *fp;

    if (zend_parse_parameters(argc TSRMLS_CC, "r", &filehandle) == FAILURE)
        return;

    if (ZEND_FETCH_RESOURCE(fp, FILE *, &filehandle, -1, "standard-c file", le_myfile)
        || (fp == NULL)
        || RETURN_FALSE
        || (feof(fp) <= 0)
        /* החזר eof גם אם הייתה שגיאה */
        || RETURN_TRUE
        || RETURN_FALSE
    )
    {

```

15.2.5.8 הסרת משאב

כדי להסיר משאב, בדרך כלל תרצו להשתמש במקרו הבא:

```
(int zend_list_delete(int id
```

טבלה 15.5: רשימת מאקרוי VCWD

מאקרו VCWD	ספריית C סטנדרטית
(VCWD_ACCESS)	(access)
(VCWD_UTIME)	(utime)
(VCWD_CHMOD)	(chmod)
(VCWD_CHOWN)	(chown)

15.2.5.6 כתיבת פונקציית ה-PHP הראשונה שלך מבוססת משאבים

מימוש (file_open) אמור להיות פשוט כעת, והוא אמור להיראות כך:

```
C
(PHP_FUNCTION(file_open
}
```

```
;char *filename = NULL
```

```
;char *mode = NULL
```

```
;int argc = ZEND_NUM_ARGS
```

```
;int filename_len
```

```
;int mode_len
```

```

;FILE *fp

,if (zend_parse_parameters(argc TSRMLS_CC, "ss", &filename
    } (filename_len, &mode, &mode_len) == FAILURE&

;return

{

;fp = VCWD_FOPEN(filename, mode

} (if (fp == NULL

;RETURN_FALSE

{

;(ZEND_REGISTER_RESOURCE(return_value, fp, le_myfile

{

```

ייתכן שתבחינו שהארגומנט הראשון למאקרו רישום המשאב הוא משתנה בשם `return_value`, שהופיע "משום מקום". משתנה זה מוגדר אוטומטית על ידי תשתית ההרחבות והוא מסוג `zval *` המצביע לערך החזרה של הפונקציה. המאקרואים שדנו בהם קודם המשפיעים על ערך החזרה, כגון `RETURN_LONG()` ו-`RETVAL_BOOL()`, למעשה משנים את הערך של `return_value`. לכן, קל לנחש שהקוד רושם את מצביע הקובץ `fp` שקיבלנו ומגדיר את `return_value` למשאב הרשום.

15.2.5.7 גישה למשאב

כדי לגשת למשאב, עליכם להשתמש במאקרו הבא (ראו טבלה 15.6 להסבר על הארגומנטים שלו):

```

ZEND_FETCH_RESOURCE(rsrc, rsrc_type, passed_id, default_id, resource_type_name,
;resource_type

```

טבלה 15.6: ארגומנטים של המאקרו `ZEND_FETCH_RESOURCE`

פרמטר	משמעות

rsrc	המשתנה שאליו יוקצה ערך המשאב. עליו להיות מאותו טיפוס של המשאב.
rsrc_type	הטיפוס של rsrc שישמש להמרת המשאב פנימית לטיפוס הנכון.
passed_id	ערך המשאב לחיפוש (בתור zval **).
default_id	אם ערך זה אינו -1, מזהה זה נלקח. משמש למימוש ברירת מחדל למשאב.
resource_type_name	שם קצר לטיפוס המשאב המשמש בהודעות שגיאה.
resource_type	מזהה טיפוס המשאב של המשאב הרשום.

בעזרת מאקרו זה, נוכל כעת לממש את `file_eof()`¹:

```
C
(PHP_FUNCTION(file_eof)
{
    int argc = ZEND_NUM_ARGS();
    zval *filehandle = NULL;
    FILE *fp;

    if (zend_parse_parameters(argc TSRMLS_CC, "r", &filehandle) == FAILURE)
        return;

    {
        (ZEND_FETCH_RESOURCE(fp, FILE *, &filehandle, -1, "standard-c file", le_myfile
```

```

    } (if (fp == NULL
;RETURN_FALSE

{

    } (if (feof(fp) <= 0
/* החזר true גם אם הייתה שגיאה */
;RETURN_TRUE

{
;RETURN_FALSE

{

```

15.2.5.8 הסרת משאב

כדי להסיר משאב, בדרך כלל תרצו להשתמש במאקרו הבא:

```
(int zend_list_delete(int id
```

למאקרו מועבר המזהה (ID) של המשאב, והוא מחזיר SUCCESS או FAILURE. אם המשאב קיים, לפני הסרתו מרשימת המשאבים של Zend, המנוע יקרא ל-destructor הרשום עבור טיפוס משאב זה. לכן, בדוגמה שלנו, אין צורך להשיג את מצביע הקובץ ולקרוא ל-fclose() לפני הסרת המשאב; ניתן פשוט למחוק אותו.

שימוש במאקרו זה מאפשר לנו לממש את file_close():

```

C
(PHP_FUNCTION(file_close
}

```

```
;()int argc = ZEND_NUM_ARGS
```

```
;zval *filehandle = NULL
```

```
} (if (zend_parse_parameters(argc TSRMLS_CC, "r", &filehandle) == FAILURE
```

```
;return
```



```

    {

        } (if (zend_list_delete(Z_RESVAL_P(filehandle)) == FAILURE

                                ;RETURN_FALSE

                                {

                                    ;RETURN_TRUE

                                }

```

בוודאי שאלתם את עצמכם מה עושה `Z_RESVAL_P()`. כאשר אנו שולפים את המשאב מרשימת הארגומנטים באמצעות `zend_parse_parameters()`, אנו מקבלים אותו בצורת `zval`. כדי לגשת למזהה המשאב `(resource id)`, אנו משתמשים במאקרו `Z_RESVAL_P()`, ואז מעבירים אותו ל-`zend_list_delete()`.

משפחה שלמה של מאקרואים מסייעת בגישה לערכים המאוחסנים בתוך `zval` (ראו טבלה 15.7).

טבלה 15.7: מאקרואים לגישה ל-`zval`

מאקרו	ערך הגישה	טיפוס C
<code>Z_LVAL</code> , <code>Z_LVAL_P</code> , <code>Z_LVAL_PP</code>	ערך שלם (Integer)	Long
<code>Z_BVAL</code> , <code>Z_BVAL_P</code> , <code>Z_BVAL_PP</code>	ערך בוליאני	zend_bool
<code>Z_DVAL</code> , <code>Z_DVAL_P</code> , <code>Z_DVAL_PP</code>	ערך עשרוני	double
<code>Z_STRVAL</code> , <code>Z_STRVAL_P</code> , <code>Z_STRVAL_PP</code>	ערך מחרוזת	* char
<code>Z_STRLEN</code> , <code>Z_STRLEN_P</code> , <code>Z_STRLEN_PP</code>	אורך מחרוזת	int
<code>Z_RESVAL</code> , <code>Z_RESVAL_P</code> , <code>Z_RESVAL_PP</code>	ערך משאב	Long

* HashTable	מערך אסוציאטיבי	Z_ARRVAL, Z_ARRVAL_P, Z_ARRVAL_PP
Enumeration	טיפוס ה-zval	Z_TYPE, Z_TYPE_P, Z_TYPE_PP

15.2.5.9 מאקרואים לגישה לערכי zval

לכל המאקרואים יש שלוש צורות: אחת המקבלת zval, אחת עבור *zval, ואחת עבור **zval. ההבדל בשמות הוא שלראשונה אין סיומת, ל-zval * יש סיומת P_ (מציין pointer אחד), ול-zval ** יש סיומת PP_ (שני פוינטרים).

כעת יש לכם מספיק מידע כדי להשלים את הפונקציות file_read() ו-file_write() בעצמכם. להלן מימוש אפשרי:²

```

C
(PHP_FUNCTION(file_read

}

;()int argc = ZEND_NUM_ARGS

;long size

;zval *filehandle = NULL

;FILE *fp

;char *result

;size_t bytes_read

} if (zend_parse_parameters(argc TSRMLS_CC, "rl", &filehandle, &size) == FAILURE

;return

{

;(ZEND_FETCH_RESOURCE(fp, FILE *, &filehandle, -1, "standard-c file", le_myfile

;(result = (char *) emalloc(size + 1

```

```

;(bytes_read = fread(result, 1, size, fp

;result[bytes_read] = '\0

;(RETURN_STRING(result, 0

{

(PHP_FUNCTION(file_write

}

;char *buffer = NULL

;()int argc = ZEND_NUM_ARGS

;int buffer_len

;zval *filehandle = NULL

;FILE *fp

if (zend_parse_parameters(argc TSRMLS_CC, "rs", &filehandle, &buffer, &buffer_len) ==
    ) {FAILURE

;return

{

;(ZEND_FETCH_RESOURCE(fp, FILE *, &filehandle, -1, "standard-c file", le_myfile

} (if (fwrite(buffer, 1, buffer_len, fp) != buffer_len

;RETURN_FALSE

{

;RETURN_TRUE

{

```

15.2.6 משתנים גלובליים

ייתכן שתמצאו להשתמש במשתני C גלובליים בהרחבה שלכם. מכיוון ש-PHP תוכננה לרוץ בסביבות מרובות-תהליכים (multi-threaded), אין להגדיר משתנים גלובליים באופן רגיל. PHP מספקת מנגנון היוצר עותק של המשתנים לכל תהליך בנפרד.

הסקריפט `ext_skel` יצר את הקוד הנחוץ לכך בתוך `php_myfile.h`:

C

```
(ZEND_BEGIN_MODULE_GLOBALS(myfile
```

```
;int global_value
```

```
;char *global_string
```

```
(ZEND_END_MODULE_GLOBALS(myfile
```

כדי לגשת למשתנים אלו בקוד ה-C שלכם, עליכם להשתמש במאקרו `MYFILE_G`. המנגנון מוודא גישה בטוחה לכל תהליך ללא צורך בניהול נעילות (mutual exclusion) מצדכם.

בנוסף, עליכם להסיר את סימן ההערה מהשורה הבאה ב-`myfile.c`:

```
(ZEND_DECLARE_MODULE_GLOBALS(myfile
```

ניתן לאתחל את המשתנים הגלובליים בתחילת כל בקשה באמצעות המאקרו:

```
ZEND_INIT_MODULE_GLOBALS(module_name, globals_ctor, globals_dtor
```

15.2.7 הוספת הוראות INI מותאמות אישית

מימוש קובץ ה-`php.ini` (INI) מאפשר להרחבות PHP לרשום ולהאזין לרשומות INI משלהן. אם לרשומות אלו מוקצה ערך דרך `php.ini`, קובץ `htaccess` של Apache או שיטות הגדרה אחרות, המשתנה הרשום תמיד יעודכן בערך הנכון.

הוראות INI נרשמות באמצעות המאקרו `STD_PHP_INI_ENTRY` בין המאקרואים `PHP_INI_BEGIN` ו-`PHP_INI_END`. לדוגמה, ב-`myfile.c` תראו משהו כזה:

C

```
(PHP_INI_BEGIN
```

```
STD_PHP_INI_ENTRY("myfile.global_value", "42", PHP_INI_ALL, OnUpdateInt, global_value,  
                  (zend_myfile_globals, myfile_globals
```

```
STD_PHP_INI_ENTRY("myfile.global_string", "foobar", PHP_INI_ALL, OnUpdateString,  
                  (global_string, zend_myfile_globals, myfile_globals
```

```
(PHP_INI_END
```

קיימים מאקרואים נוספים, אך זהו הנפוץ ביותר. להלן הסבר על הפרמטרים שלו:

טבלה 15.9: פרמטרים של המאקרו STD_PHP_INI_ENTRY

פרמטר	משמעות
name	שם רשומת ה-INI.
default_value	ערך ברירת המחדל (תמיד כמחרוזת).
modifiable	היכן ניתן לשנות את הערך: PHP_INI_SYSTEM (ב-, php.ini), PHP_INI_PERDIR (ב-, PHP_INI_USER), htaccess (בסקריפטים) או PHP_INI_ALL (בכל מקום).
on_modify	פונקציית Callback המטפלת בשינוי (למשל: OnUpdateInt, OnUpdateString, OnUpdateBool).
property_name	שם המשתנה שיש לעדכן.
struct_type	סוג המבנה (Struct) בו נמצא המשתנה (למשל zend_myfile_globals).
struct_ptr	שם מבנה הגלובלים (למשל myfile_globals).

כדי שהמנגנון יעבוד, עליכם להסיר את סימני ההערה מהקריאה ל-REGISTER_INI_ENTRIES() בתוך PHP_MINIT_FUNCTION ומהקריאה ל-UNREGISTER_INI_ENTRIES() בתוך PHP_MSHUTDOWN_FUNCTION.

הגישה למשתנים מתבצעת פשוט על ידי כתיבת MYFILE_G(global_value) מכל מקום בהרחבה.

15.2.8 מאקרואים של מנהל משאבים בטוח לתהליכונים (TSRM)

בוודאי שמתם לב לשימוש במאקרואים המתחילים ב-TSRM (ראשי תיבות של Thread-Safe Resource Manager). אלו מאפשרים להרחבה להחזיק משתנים גלובליים משלה בסביבה מרובת-תהליכונים.

כדי להשתמש במאקרו גישה כמו `MYFILE_G()`, המידע על הקשר (context) ה-TSRM חייב להיות נוכח בפונקציה. מנוע ה-Zend מעביר את ההקשר הזה כפרמטר ככל האפשר מטעמי ביצועים. בתוך `PHP_FUNCTION()` הוא זמין אוטומטית. עם זאת, אם אתם קוראים לפונקציות C אחרות הזקוקות לגלובלים, עליכם להעביר להן את ההקשר או לשלוף אותו (מה שאיטי יותר).

כדי לשלוף את ההקשר ידנית:

```
C
void myfunc() {
    TSRMLS_FETCH();
    MYFILE_G(myglobal) = 2;
}
```

עדיף להעביר את ההקשר כפרמטר באמצעות `TSRMLS_C` (בקריאה) ו-`TSRMLS_D` (בהצהרה):

```
C
void myfunc(TSRMLS_D) {
    MYFILE_G(myglobal) = 2;
}
```

```
PHP_FUNCTION(my_php_function) {
    myfunc(TSRMLS_C);
}
```

(הערה: המאקרואים עם סיומת `CC` או `DC` כוללים פסיק לפני ההקשר, לשימוש כשיש ארגומנטים נוספים).

15.3 סיכום

בפרק זה למדתם את היסודות החשובים לכתיבת והבנת הרחבות PHP. ה-API שמספק מנוע ה-Zend עשיר מאוד ומאפשר גם כתיבת הרחבות מונחות עצמים. אין תחליף לבחינת קוד המקור של הרחבות הליבה המגיעות עם PHP כדי ללמוד טכניקות מתקדמות.

מידע נוסף ניתן למצוא במדריך PHP תחת הפרק "Extending PHP", וכן מומלץ לעקוב אחרי כלי חדש בשם `PECL_Gen`, המציע תכונות מתקדמות יותר מהסקריפט `ext_skel`.

פרק 16

16.1 מבוא

באופן מסורתי, PHP משמשת בסביבות אינטרנט להפקת קוד HTML שבו המשתמש צופה בדפדפן. האינטראקציה בין PHP לשרת האינטרנט (Apache, AOLserver, IIS וכדומה) מתבצעת דרך שכבה הנקראת **SAPI** (קיצור של Web Server API). נדרשת גרסת בנייה נפרדת של PHP כדי להתממשק עם כל סוג של שרת אינטרנט דרך ה-SAPI.

בפרק זה תחקרו את ה-**CLI** (קיצור של Command Line Interface), ממשק שורת הפקודה שהופך את PHP לשפת תסריט (Scripting) מסורתית. פרק זה מדגים שימוש ב-CLI לכתובת כלי שורת פקודה וכן יישומי שרת עצמאיים.

16.2 תסריטי של (Shell) ב-PHP CLI

גרסת ה-CLI של PHP נועדה לכתובת תסריטי של עצמאיים הרצים ללא תלות בשרת אינטרנט. החל מגרסה 4.3.0, גרסת ה-CLI מותקנת כברירת מחדל לצד ממשק שרת האינטרנט שבחרתם.

16.2.1 במה שונה CLI מ-CGI

גרסת ה-CLI דומה למדי לגרסת ה-CGI שעליה התבססה בעבר, אך ההבדל העיקרי טמון באינטגרציה עם שרת האינטרנט. ב-PHP, CLI מקוצצת ליסודות בלבד: היא אינה מייבאת משתני טפסים (GET או POST), אינה מפיקה כותרות MIME בפלט, ובאופן כללי אינה מבצעת את הפעולות "מאחורי הקלעים" שביצועי SAPI אחרים עושים.

16.2.1.1 פרמטרים כברירת מחדל

ל-CLI יש ערכי ברירת מחדל שונים עבור מספר אפשרויות שורת פקודה והגדרות php.ini.

טבלה 16.1: אפשרויות ברירת מחדל ב-CLI

הגדרה/אופציה	ברירת ב-CLI	מחדל	תיאור
אופציה -q	מאפשרת		מבטלת כותרות HTTP בפלט.
אופציה -C	מאפשרת		PHP לא משנה את תיקיית העבודה לזו של התסריט.

html_errors	מבוטלת	הודעות שגיאה יוצגו בטקסט פשוט ולא ב-HTML.
implicit_flush	מאפשרת	הפלט נשלח מיד ללא אגירה (Buffer).
register_argc_argv	מאפשרת	המשתנים <code>argv\$</code> ו- <code>argc\$</code> נרשמים תמיד.
max_execution_time	0	זמן ריצה מקסימלי; 0 פירושו ללא הגבלה.

16.2.1.2 אפשרויות נוספות

ישנן אפשרויות שורת פקודה הקיימות ב-CLI אך לא ב-CGI.

טבלה 16.2: אפשרויות CLI נוספות

אופציה	תיאור
<code>-r code</code>	הרצת קוד ישירות מהטרמינל (ללא תגי <code><?php></code>).

אם אתם כותבים תסריט שצריך לקרוא מקלט סטנדרטי (STDIN) ובזמן קבלת קלט משתמש מהטרמינל, עליכם להשתמש ב-`dev/tty/` לאינטראקציה עם המשתמש.

16.2.2 אורך חיי הביצוע

בשרת אינטרנט, תסריטי PHP מסיימים את עבודתם מהר. ב-CLI, התסריט שלכם עשוי לרוץ לנצח (למשל כ-Daemon).

השלכה מעשית אחת היא שסגנון קידוד מרושל, שהוא יחסית חסר נזק בבקשת אינטרנט קצרה, הופך לקריטי בתסריט שרץ זמן רב. אם פתחתם קובץ או חיבור למסד נתונים ולא סגרתם אותו במפורש, המשאב יישאר תפוס עד שהתסריט יסתיים. לכן: סגרו קבצים, התנתקו ממסדי נתונים ורוקנו מערכים גדולים כשסיימתם איתם.

16.2.3 Hash-Bang (!#): הפיכת התסריט לבר-הרצה

במערכות דמויות UNIX, אם שני התווים הראשונים בקובץ הם `#!`, שאר השורה נחשבת לשם התוכנית שמריצה את הקובץ.

לדוגמה, תסריט שמתחיל כך:

usr/bin/php -Cq/!#

י אפשר לך להריץ את הקובץ ישירות מהשל מבלי להקליד php לפניו.

16.2.5 ניתוח אפשרויות שורת פקודה (getopt)

כדי להפוך את כלי שורת הפקודה שלכם למקצועיים, עליכם לתמוך בפרמטרים (כמו v- לגרסה או h- לעזרה).

הספרייה Console_Getopt (מבית PEAR) מאפשרת לנהל זאת בקלות.

דוגמה לשימוש באופציות קצרות וארוכות:

הקוד משתמש במחרוזת הגדרות שבה נקודתיים (:) אחרי תו מציינות שהאופציה דורשת ערך (למשל c: עבור קובץ קונפיגורציה). נקודתיים כפולות (::) מציינות ערך אופציונלי.

PHP

// דוגמה לשימוש בסיסי ב-getopt

```
;$options = Console_Getopt::getopt($argv, "qvhc$");
```

הפונקציה usage() בדוגמה מדגימה כתיבה ל-stderr (ערוץ השגיאות הסטנדרטי), מה שנחשב לפרקטיקה נכונה בכתיבת כלי של, כך שהודעות העזרה לא יתערבבו עם הפלט הרגיל של התוכנית.

16.2.4 פרקטיקות טובות

בעת כתיבת תסריטי של (shell scripts), עליך לעקוב אחר מספר פרקטיקות טובות כדי להקל על חיך ועל חיהם של אחרים שישתמשו בתסריט שלך. לדוגמה, רוב משתמשי UNIX מצפים שהתוכניות שלהם יגיבו ל-foo -h או foo --help עם הודעת שימוש קצרה, או שהן ידפיסו שגיאות לערוץ השגיאות הסטנדרטי (standard error) במקום לפלט הסטנדרטי. סעיף זה מונה מספר פרקטיקות שהמחברים מחשיבים כ"טובות" (™Good).

16.2.4.1 הודעת שימוש (Usage Message)

הציגו הודעת שימוש ב-standard error וצאו עם קוד שאינו 0 אם התסריט הופעל ללא הפרמטרים המצופים, או אם הוא רץ עם אופציית h- (או help-- אם אתם משתמשים באופציות ארוכות). הודעת השימוש צריכה למנות את כל הפרמטרים הנדרשים והאופציונליים.

קיימת גם מוסכמה סטנדרטית לרישום אופציות ופרמטרים:

- [c-] – עשוי להכיל את האופציה c-.
- {c foo-} – חייב להכיל את האופציה c- עם פרמטר.
- [a | -b] – עשוי להכיל או את a- או את b-.
- ...file – חייב לקבל פרמטר אחד או יותר של קבצים.

16.2.4.2 קוד יציאה (Exit Code)

אם התסריט נכשל, צא עם קוד שאינו 0 (פרט ל-255, השמור ל-PHP עצמה עבור שגיאות הידור/פענוח). אם התסריט הצליח, צא עם קוד 0.

16.2.4.3 הודעות שגיאה

הוסיפו את שם התסריט לפני כל הודעת שגיאה, כדי שהמשתמש יוכל לראות מאיזה תסריט מקורה השגיאה. זה שימושי במיוחד אם התסריט מופעל מתוך תסריטים או תוכניות אחרות.

16.2.5 בקרת תהליכים (Process Control)

בעת הרצת תסריטי PHP ב-CLI, הרחבת ה-`pcntl` מספקת פונקציות לשליטה בתהליך ה-PHP.

16.2.5.1 תהליכים (Processes)

תהליך הוא קטע קוד המבוצע על ידי מערכת ההפעלה. ב-UNIX, תהליכים מורכבים מקוד בר-ביצוע, משתני סביבה, זיכרון מחסנית (stack), זיכרון ערימה (heap), מתארי קבצים (file descriptors) ומאפייני אבטחה.

16.2.5.2 פיצול (Forking)

"Forking" הוא מונח ב-UNIX ליצירת תהליך חדש על ידי שכפול של תהליך קיים. תהליך הבן (child) יורש את הקוד, הסביבה, הזיכרון ומתארי הקבצים של תהליך האב.

ההבדל הוא שבתוך תהליך האב, הקריאה ל-`fork` מחזירה את מזהה התהליך (PID) של הבן, בעוד שבתוך תהליך הבן הקריאה מחזירה 0.

16.2.5.3 Exec

כאשר תוכנית אחת מריצה תוכנית אחרת, מבוצע הליך דו-שלבי: ראשית התהליך הקורא מתפצל (fork), ומיד לאחר מכן מבצע קריאת `exec` כדי להחליף את הקוד והזיכרון באלו של התוכנית החדשה. הדבר משמש גם ליצירת "Daemons" (תהליכי רקע) על ידי ניתוק מהטרמינל.

16.2.5.4 אותות (Signals)

ב-UNIX, אותות הם מנגנון בסיסי להעברת הודעות בין תהליכים (כמו `SIGINT` כאשר לוחצים על Ctrl-c). ב-PHP ניתן להגדיר פונקציה שתטפל באותות אלו באמצעות `pcntl_signal()`.

16.3 טבלה: אותות נבחרים

- **SIGHUP**: ניתוק חיבור טרמינל.
- **SIGINT**: הפרעה (Ctrl-c).
- **SIGKILL**: סיום תהליך כפוי (לא ניתן לתפוסה).
- **SIGTERM**: בקשת סיום תהליך רגילה.
- **SIGCHLD**: תהליך בן מת או שינה סטטוס.

16.2.6 דוגמאות

הפרק מציג מספר כלים:

1. **PHP Filter Utility**: כלי לקריאת שורות מהקלט הסטנדרטי (STDIN) ועיבודן דרך פונקציית PHP (כמו `base64_encode`).
 2. **Chat Server**: שרת צ'אט שלם המשתמש ב-`socket_select()` כדי לנהל משתמשים מרובים בו-זמנית, תומך בחוצצי קלט/פלט ובפקודות כמו `quit/` ו-`who/`.
-

16.3 סיכום

בפרק זה למדתם להשתמש ב-PHP עבור תסריטי של (shell scripting) מעבר לסביבת הרשת:

- פענוח אופציות שורת פקודה עם `Console_Getopt`.
- התנהגות נכונה של תסריטי של וניהול קלט/פלט סטנדרטי.
- בקרת תהליכים (fork, exec, signals).
- כתיבת שרתי PHP עצמאיים.

נספח א

אינדקס חבילות PEAR ו-PECL

טקסט זה נוצר אוטומטית מקובצי ה-`package.xml` של PEAR הזמינים דרך שרת ה-CVS של PHP ומוצג בפורמט "כפי שהוא" (as-is).

A.1 אימות (Authentication)

A.1.1 Auth

מאגר: PEAR – רישיון: PHP License

יצירת מערכת אימות.

A.1.1.1 תיאור

חבילת `PEAR::Auth` מספקת שיטות ליצירת מערכת אימות באמצעות PHP. נכון לעכשיו היא תומכת במכלי האחסון הבאים לקריאה/כתיבה של נתוני התחברות:

- כל מסדי הנתונים הנתמכים על ידי שכבת מסד הנתונים של PEAR
- כל מסדי הנתונים הנתמכים על ידי שכבת מסד הנתונים MDB
- קובצי טקסט פשוט (Plaintext)
- שרתי LDAP
- שרתי POP3 ו-IMAP
- חשבונות vpopmail
- RADIUS
- קובצי סיסמאות של SAMBA
- SOAP

A.1.2 Auth_Enterprise

שירות אימות והרשאות ברמת הארגון (Enterprise). החבילה כוללת שכבת שירות המטפלת בבקשות אימות ולקוח PHP. תכונות עיקריות: מבוסס Web Service, מימוש מודל "ספק" (Provider) המסוגל להתממשק למאגרי נתונים שונים, ושימוש בסט הרשאות אחיד.

A.1.3 Auth_HTTP

מספקת שיטות ליצירת מערכת אימות HTTP בדומה לאימות מבוסס `htaccess` של Apache.

A.1.4 Auth_PrefManager

מחלקה לניהול העדפות משתמש באפליקציית ווב, המאחזרת ערכים לפי זיהוי משתמש ושם ההעדפה.

A.1.7 LiveUser

מסגרת עבודה (Framework) לניהול אימות משתמשים והרשאות. מורכבת משלושה אלמנטים: מחלקת LiveUser, מכלי אימות (Auth) ומכלי הרשאות (Perm). המערכת מאפשרת לבדוק משתמשים מול מספר מכלי נתונים במקביל (למשל, מסד נתונים מקומי ושרת ארגוני).

A.2 בדיקות ביצועים (Benchmarking)

A.2.1 Benchmark

מסגרת עבודה למדידת זמני ריצה של סקריפטים ב-PHP או קריאות לפונקציות.

A.3 מטמון (Caching)

(A.3.1 APC (Alternative PHP Cache

מספק מסגרת עבודה חופשית וחזקה לאופטימיזציה ואחסון במטמון של קוד הביניים (intermediate code) של PHP.

A.3.2 Cache

מסגרת עבודה לאחסון במטמון של נתונים שרירותיים, תוצאות פונקציות או פלט של סקריפט שלם.

A.3.3 Cache_Lite

מערכת מטמון קטנה ומהירה המותאמת לעבודה עם קבצים, הכוללת מנגנוני נעילת קבצים למניעת שגיאות.

A.4 הגדרות (Configuration)

A.4.1 Config

"אולר שווייצרי" לניהול הגדרות. תומך ביצירת הגדרות מאפס, ניתוח (Parsing) ופלט בפורמטים שונים (XML, PHP, INI, Apache), והמרת הגדרות בין פורמטים.

A.5 מוסף (Console)

A.5.1 Console_Color

מאפשרת שימוש קל בצבעי ANSI במסוף. למשל, הפיכת קודים כמו `%r` לצבע אדום.

A.5.2 Console_Getopt

מימוש PHP ל-"getopt", התומך באופציות קצרות וארוכות בשורת הפקודה.

A.5.3 Console_ProgressBar

ממשק להצגת פסי התקדמות במסוף (למשל עבור הורדות או משימות ממושכות).

A.5.5.2 System_Command

ממשק להרצת פקודות מערכת משורת הפקודה בצורה בטוחה, עם טיפול בשגיאות PEAR והפרדה בין פלט סטנדרטי (stdout) לפלט שגיאות (stderr).

A.6 מסדי נתונים (Database)

A.6.1 DB

שכבת הפשטה למסדי נתונים (Abstraction Layer) המספקת API מונחה עצמים, ניידות בין מערכות ניהול מסדי נתונים שונות (DBMS), טיפול אחיד בשגיאות ותמיכה בשאילתות מוכנות (prepare/execute). תומכת במגוון רחב של מסדי נתונים כמו MySQL, PostgreSQL, SQLite ועוד.

A.6.6 DB_DataObject

מבצע שתי משימות: בניית שאילתות SQL מבוססות אובייקטים, ותפקוד כמאגר נתונים עבור שורות בטבלה.

A.6.7 DB_DataObject_FormBuilder

יוצר אוטומטית אובייקטים של `HTML_QuickForm` (טפסים) המבוססים על הגדרות מסד הנתונים של `DB_DataObject`, מה שמאפשר פיתוח אבות-טיפוס.

A.6.10 DB_NestedSet

API לבנייה ותשאול של מבני עץ בעלי עומק אינסופי בתוך מסד נתונים יחסי.

A.6.16 MDB / MDB2

שכבת הפשטה מתקדמת המשלבת את PEAR DB ו-Metabase. שמה דגש רב על ניידות (Portability) ותומכת בתכונות מתקדמות כמו ניהול מבנה מסד הנתונים (Schema) באמצעות קובצי XML ללא תלות בסוג מסד הנתונים.

A.6.17 MDB2

מאגר: PEAR – **רישיון:** BSD – **מאת:** לוקאס קהווא סמית' (מוביל)

שכבת הפשטה למסדי נתונים.

A.6.17.1 תיאור

PEAR MDB2 הוא מיזוג של שכבות ההפשטה PEAR DB ו-Metabase. הוא מספק API אחיד לכל מערכות ה-RDBMS הנתמכות. ההבדל העיקרי בינו לבין חבילות הפשטה אחרות הוא ש-MDB2 הולך רחוק יותר כדי להבטיח ניידות (Portability).

בין התכונות של MDB2:

- API לשאילתות בסגנון מונחה עצמים (OO).
- פורמט (DSN (Data Source Name) או מערך להגדרת שרתי מסד נתונים.
- הפשטת סוגי נתונים והמרתם לפי דרישה.
- קודי שגיאה ניידים.
- שליפת שורות רציפה, לא רציפה או בקבוצות (Bulk).
- יכולת ביצוע שאילתות מאוחסנות (buffered) או לא מאוחסנות.
- אמולציה של הכנה/ביצוע (Prepare/Execute).
- ניהול מבנה מסד הנתונים (Schema) באמצעות XML ללא תלות ב-RDBMS.
- הנדסה לאחור (Reverse engineering) של סכימות XML ממסד נתונים קיים.

MySQL, PostgreSQL, Oracle, Frontbase, Interbase/Firebird, MSSQL, **מסדי נתונים נתמכים:** SQLite ועוד.

A.6.19 MDB_QueryTool

ממשק מונחה עצמים לאחזור ושינוי נתונים בקלות. זוהי הפשטה של שפת SQL המספקת שיטות כמו `setWhere`, `setOrder`, `setJoin` וכו' לבנייה קלה של שאילתות, ומתממשקת היטב עם טופסי HTML באמצעות מערכים.

A.6.20 oci8

מאגר: PECL. מעטפת (Wrapper) לממשק הקריאות של אורקל (OCI), המאפשרת גישה למסדי נתונים מסוג Oracle 7/8/9.

A.6.23 SQLite

מאגר: PECL. קישורים למסד הנתונים SQLite – ספריית C המממשת מנוע מסד נתונים SQL משובץ (Embedded), שאינו דורש תהליך שרת נפרד.

A.7 תאריך ושעה (Date and Time)

A.7.1 Calendar

חבילה לבניית מבני נתונים של לוח שנה (ללא תלות בפלט). מאפשרת בניית ממשק משתמש בקלות מעל מבנה הנתונים וחיבורו למאגרי מידע של "אירועים".

A.7.2 Date

מחלקות גנריות לייצוג וניהול תאריכים, זמנים ואזורי זמן מבלי להסתמך על Timestamps (חותמות זמן), מה שמאפשר עבודה עם תאריכים שלפני 1970 ואחרי 2038.

A.8 הצפנה (Encryption)

- **Crypt_CHAP:** יצירת חבילות CHAP (פרוטוקול אימות). תומך ב-MS-CHAPv1, CHAP-MD5 ו-MS-CHAPv2.
 - **Crypt_HMAC:** מחלקה לחישוב חתימות (Hashes) תואמות RFC 2104.
 - **Crypt_RC4:** מחלקה לביצוע הצפנת RC4.
 - **Crypt_Xtea:** מימוש של אלגוריתם ההצפנה TEA (גרסה חדשה). אינו תלוי ב-mcrypt.
-

A.9 פורמטי קבצים (File Formats)

- **(PECL) bz2:** הרחבה לניהול דחיסת Bzip2.
- **Contact_Vcard_Build/Parse:** יצירה ופענוח של קובצי vCard (אנשי קשר) בגרסאות 2.1 ו-3.0.
- **File_DICOM:** קריאה ושינוי של קובצי DICOM (סטנדרט להדמיה רפואית כמו צילומי רנטגן ו-CT).
- **Spreadsheet_Excel_Writer:** יצירת גיליונות אקסל (פורמט BIFF5) ללא צורך באובייקטי COM. תומך בנוסחאות, תמונות ועיצוב תאים.
- **(PECL) zip:** הרחבה לקריאת קובצי Zip.

A.10 מערכת קבצים (File System)

- **Archive_Tar**: ניהול קובצי Tar (יצירה, חילוץ והוספה), כולל תמיכה ב-Gzip ו-Bz2.
- **File_Find**: חיפוש במערכת הקבצים, כולל חיפוש רקורסיבי ושימוש ב-Globbing.
- **File_Passwd**: ניהול סוגים רבים של קובצי סיסמאות (htdigest, httpasswd, SMB, Unix ועוד).
- **VFS**: ממשק API למערכת קבצים וירטואלית, עם תמיכה ב-SQL, FTP ומערכות קבצים מקומיות.

A.12 HTML

- **HTML_BBCodeParser**: מנוע לניתוח והחלפת תגי UBB (כמו בפורומים) בקוד HTML תקין.
- **HTML_Crypt**: הצפנת טקסט (כמו כתובות מייל) המופענח בצד הלקוח באמצעות JavaScript, למניעת איסוף נתונים על ידי רובוטי ספאם.
- **HTML_CSS**: ממשק מונחה עצמים ליצירה, פענוח וניהול של הצהרות CSS וגיליונות סגנון.
- **HTML_Menu**: יצירה וניהול של מבני ניווט ותפריטים באתרים מתוך מערכי נתונים (Hashes).
- **HTML_Page**: ממשק ליצירת דפי XHTML תקינים, כולל תמיכה ב-Doctypes, META, וקישור סקריפטים וגיליונות סגנון.

A.12.9 html_parse

מאגר: PECL – **רישיון:** PHP License – **מאת:** הרטמוט הולצגראפה (מוביל)

הרחבה לניתוח (parsing) של HTML המבוססת על ספריית ekhtml.

A.12.10 HTML_Progress

מאגר: PEAR – **רישיון:** PHP License 3.0

דרך מהירה וקלה להטמעת פס טעינה (loading bar) במסמכי XHTML. חבילה זו מאפשרת הוספת פס טעינה הניתן להתאמה אישית מלאה. הדפדפן נדרש לתמוך ב-DHTML.

תכונות:

- יצירת פסים אופקיים, אנכיים וגם מעגלים, אליפסות ופוליגונים.
- תמיכה בגיליונות סגנון (CSS) ו-JavaScript חיצוניים.
- תאימות מלאה לתקני CSS/XHTML ואינטגרציה קלה עם מנועי תבניות.
- מימוש של תבנית העיצוב Observer (תצפיתן).
- אפשרות לביטול ההתקדמות על ידי המשתמש בכל עת.

A.12.11 HTML_QuickForm

מאגר: PEAR – **רישיון:** PHP License

חבילה המספקת שיטות ליצירה, אימות ועיבוד של טופסי HTML בצורה דינמית.

תכונות:

- מעל 20 אלמנטים מוכנים לשימוש.
- קוד תואם XHTML.
- חוקי אימות ניתנים להרחבה וסינון אוטומטי בצד השרת.
- יצירת קוד JavaScript לאימות בצד הלקוח לפי דרישה.
- תמיכה בהעלאת קבצים ובהתאמה אישית מלאה של תצוגת הטופס.

A.12.12 HTML_QuickForm_Controller

תוסף ל-HTML_QuickForm המאפשר בניית טפסים מרובי-דפים (multipage forms), תוך מימוש תבנית העיצוב PageController לניהול ערכי הטופס בין הבקשות השונות.

A.12.15 HTML_Table

מאגר: PEAR – רישיון: PHP License

הופכת את עיצוב טבלאות ה-HTML לקל, גמיש ויעיל. מאפשרת שינוי טבלאות בכל עת, טיפול ב-`colspan/rowspan` ושימוש חוזר בהגדרות טבלה.

A.12.16 HTML_Table_Matrix

הרחבה ל-HTML_Table המאפשרת מילוי אוטומטי של טבלה בנתונים בדרכים שונות (למעלה, למטה, באלכסון וכו') באמצעות מחלקות Filler.

A.12.18 HTML_Template_Flexy

מאגר: PEAR – רישיון: PHP License

מנוע תבניות עוצמתי במיוחד המבוסס על Tokenizer.

תכונות:

- API פשוט מאוד ומהיר ביותר (לפחות פי 4 מרוב המנועים האחרים).
- תמיכה בעריכה באמצעות עורכי WYSIWYG מבלי שהקוד יישבר.
- אבטחה משופרת מפני התקפות XSS (סינון אוטומטי של משתנים).
- תמיכה בריבוי שפות ובאלמנטים דינמיים.

A.12.24 Pager

מאגר: PEAR – רישיון: PHP License

חבילה לחלוקת נתונים לדפים (Paging). היא מקבלת מערך נתונים ובונה קישורים לדפדוף בסגנון "חלון קופץ" או "חלון מחליק".

A.13 HTTP

A.13.2 HTTP_Client

דרך קלה לביצוע בקשות HTTP מרובות ועיבוד התוצאות שלהן. מנהלת עוגיות (cookies) והפניות (redirections) באופן אוטומטי.

A.13.3 HTTP_Download

ממשק לשליחת הורדות קבצים ללקוח דרך HTTP, כולל תמיכה במטמון (caching), דחיסה והורדות חלקיות (resuming).

A.13.5 HTTP_Request

מספקת דרך קלה לביצוע בקשות HTTP (תומכת ב-GET, POST, HEAD, PUT, DELETE), אימות (authentication), פרוקסי ו-SSL.

A.13.8 HTTP_Upload

ניהול קל ומאובטח של העלאת קבצים מטופסי HTML. תומכת בהעלאת מספר קבצים, שינוי שמות בצורה בטוחה ואימות סיומות קבצים.

A.14 תמונות (Images)

A.14.2 Image_Barcode

יצירת ברקודים ממחרוזות נתונות באמצעות פונקציות GD.

A.14.5 Image_Graph

שרטוט גרפים מנתונים מספריים (תנועה, כסף וכו') במגוון פורמטים (קווים, עמודות וכו'), כולל תמיכה בערוצי אלפא ורשתות.

A.14.6 Image_GraphViz

ממשק לכלי ה-GraphViz של AT&T ליצירת גרפים מכוונים ולא מכוונים והצגתם הוויזואלית.

A.14.11 Image_Transform

מספקת ממשק סטנדרטי לביצוע טרנספורמציות בתמונות (שינוי גודל, סיבוב, הוספת טקסט) תוך שימוש בספריות שונות כמו GD, ImageMagick ו-NetPBM.

A.15 בינלאומיות (Internationalization)

A.15.1 fribidi

מאגר: PECL. ממשק לספריית FriBidi המממשת את אלגוריתם ה-Unicode BiDi, ומספקת כלים לטיפול בטקסט הנכתב מימין לשמאל (RTL).

A.15.2 I18N

מאגר: PEAR – **רישיון:** PHP

חבילת בינאום (Internationalization). חבילה זו מסייעת בלוקליזציה (התאמה לשפה ולתרבות) של אפליקציות. היא כוללת שיטות לתרגום, זיהוי שפת המשתמש (דרך הדפדפן) ועיצוב תואם-תרבות של מספרים, תאריכים, שעות ומטבעות.

A.15.3 I18Nv2

גרסה מתקדמת של חבילת הבינאום. היא שואפת לספק דרך בלתי תלויה במערכת ההפעלה לביצוע פקודת `setlocale()` ומציעה שמות של מדינות ושפות המתורגמים לשפות רבות. המפתחים קוראים לציבור לתרום מידע על מוסכמות עיצוב מקומיות מרחבי העולם.

A.15.5 Translation

מחלקה ליצירת אתרים רב-לשוניים המאפשרת אחסון ושליפת מחרוזות טקסט ממסד נתונים (באמצעות PEAR::DB). היא טוענת את כל המחרוזות הרלוונטיות לעמוד מסוים לתוך משתנה כדי להאיץ את הגישה ולמנוע עומס על השרת.

A.15.6 Translation2

מחלקה לניהול אפליקציות רב-לשוניות. היא מספקת דרך קלה לשליפת מחרוזות ממקורות נתונים שונים (DB, MDB, gettext) תוך שימוש במנגנון מטמון (Caching). קיימת מחלקת ניהול (Admin) להוספה והסרה קלה של שפות ומחרוזות.

A.16 רישום (Logging)

A.16.1 Log

מאגר: PEAR – רישיון: PHP License

מערכת רישום (Logging) מופשטת התומכת בכתיבת יומני ריצה למסוף, קבצים, syslog, SQL, SQLite, דואר אלקטרוני ועוד. היא מממשת את תבנית העיצוב Subject-Observer.

A.16.2 Log_Parser

מנתח (Parser) המיועד לקריאת כמעט כל סוג של קובץ יומן ריצה. ניתן להגדיר פורמט יומן אישי באמצעות קובץ הגדרות בסגנון XML ולסנן את הנתונים המתקבלים.

A.17 דואר אלקטרוני (Mail)

A.17.1 Mail

חבילה המגדירה ממשק אחיד ליישום שליחת דואר. היא תומכת בפונקציית mail() של PHP, ב-sendmail וב-SMTP. היא כוללת גם כלי לאימות רשימות כתובות דוא"ל לפי תקן RFC 822.

A.17.5 Mail_Mime

מספקת מחלקות ליצירה ופענוח של הודעות MIME (דואר עם HTML, קבצים מצורפים, תמונות מוטמעות וכו').

A.17.6 Mail_Queue

חבילה לניהול תור הודעות דואר. היא מאפשרת להכניס הודעות למכל זמני ולשלוח אותן מאוחר יותר ברקע (למשל באמצעות crontab), כדי לא לעכב את טעינת הדף.

A.18 מתמטיקה (Math)

A.18.2 Math_Complex

מחלקות המגדירות מספרים מרוכבים ופעולות עליהם (אריתמטיקה, פונקציות טריגונומטריות, לוגריתמים ועוד).

A.18.3 Math_Fibonacci

חבילה לחישוב מספרי פיבונאצ'י באמצעות נוסחת הנסיגה או נוסחת לוקאס (יחס הזהב).

A.18.6 Math_Matrix

מחלקה לייצוג מטריצות וביצוע פעולות עליהן, כולל פתרון מערכות של משוואות ליניאריות.

A.18.9 Math_RPN

המרה של ביטויים מתמטיים לכתיב פולני הפוך (Reverse Polish Notation) וחישובם.

A.19 רשת (Networking)

A.19.12 Net_DNS

ספריית Resolver לתקשורת עם שרתי DNS לביצוע שאילתות, העברת אזורים (zone transfers) ועדכונים דינמיים. היא עוקפת את ספריית המערכת ומתקשרת ישירות עם השרת.

A.19.14 Net_FTP

ממשק מונחה עצמים לפונקציות ה-FTP של PHP. מוסיף תכונות כמו העלאה והורדה רקורסיבית, יצירת ספריות ושימוש ב-Observer להצגת פסי התקדמות.

A.19.22 Net_IRC

מחלקה ליישומי לקוח או בונים של IRC. תומכת בחיבור למספר שרתים במקביל, סוקטים שאינם חוסמים (non-blocking) ומערכת Callback לטיפול בהודעות.

A.19.23 Net_LDAP

ממשק מונחה עצמים לחיפוש וניהול רשומות LDAP (מבוסס על הממשק של Perl). תומך ב-LDAP, TLS וניהול סכימות.

A.19.31 Net_SmartIRC

חבילה מתקדמת במיוחד לתקשורת ברשתות IRC (תואמת RFC 2812). היא כוללת ניהול אירועים מבוסס זמן, הגנה מפני הצפה (Flood protection), חיבור מחדש אוטומטי, סנכרון ערוצים ומשתמשים ומערכת רישום שגיאות מפורטת.

A.19.33 Net_Socket

ממשק לממשקי סוקט של TCP. מספק פעולות חוסמות ושאינן חוסמות ומצבי קריאה/כתיבה שונים (לפי בית, לפי שורה וכו').

A.19.36 Net_UserAgent_Detect

מזהה את סוג דפדפן האינטרנט, הגרסה ומערכת ההפעלה מתוך מחרוזת ה-HTTP User Agent.

A.19.37 Net_UserAgent_Mobile

מאגר: PEAR – **רישיון:** PHP – **מאת:** קובו אטסוהירו (מוביל)

מנתח (Parser) של מחרוזות User Agent למכשירים ניידים (בעיקר יפניים). שימושי לניתוב דפים לפי סוג המכשיר הנייד.

A.19.38 Net_Whois

כלי לביצוע שאילתות בשירותי Whois לאיתור פרטי שמות מתחם (Domains) ומספרי רשת מול מאגרי NIC שונים.

A.19.39 opendirectory

ממשק PHP ל-OpenDirectory Framework – ארכיטקטורת שירות ספריות המאפשרת לאפליקציות לשלוף מידע מאוחסן בצורה מרכזית.

A.19.41 tcpwrap

קישור (Binding) ל-tcpwrappers המאפשר ניהול של קובצי ההרשאות `etc/hosts.allow/` ו-`etc/hosts.deny/`.

A.19.42 uuid

פונקציות ליצירה וניתוח של מזהים ייחודיים אוניברסליים (UUIDs). תלוי בספרייה חיצונית (libuuid).

A.20 מספרים (Numbers)

A.20.1 Numbers_Roman

שיטות סטטיות להמרה בין מספרים רגילים לספרות רומיות (תומך במספרים עד 5,999,999).

A.20.2 Numbers_Words

מחלקה להמרת מספרים הכתובים בספרות למילים (איות המספר) בשפות רבות (כגון אנגלית, גרמנית, צרפתית, פולנית ועוד).

A.21 תשלומים (Payment)

- **Payment_Process**: שלד (Framework) אחיד לעיבוד כרטיסי אשראי והמחאות אלקטרוניות ללא תלות בשער התשלום (Gateway).
- **TCLink (PECL)**: מאפשר עיבוד כרטיסי אשראי דרך שער התשלום TrustCommerce.

A.22 תשתיות PEAR

- **PEAR**: מערכת הבסיס הכוללת את מחלקת האם, מנגנוני טיפול בשגיאות, מתקין החבילות (Installer) ומחלקות עזר למערכת ההפעלה.
- **PEAR_Frontend_Web/Gtk**: ממשקי משתמש (גרפיים או מבוססי דפדפן) לניהול והתקנת חבילות PEAR.
- **PEAR_Info**: יוצר דף מידע מפורט על התקנת ה-PEAR שלך, בדומה ל-`phpinfo()`.

A.23 שפת PHP

(A.23.2 bcompiler (PECL

מהדר Bytecode המאפשר לקודד תסריטי PHP כדי להגן על קוד המקור, ליצור ספריות קוד סגור או אפליקציות למערכות משובצות (Embedded).

(A.23.5 memcache (PECL

הרחבה לעבודה עם Memcached – שירות זיכרון מטמון שנועד להפחית עומס ממסדי נתונים על ידי שמירת אובייקטים בזיכרון ה-RAM.

A.23.9 PHPUnit

שלד לבדיקות נסיגה (Regression testing) ובדיקות יחידה (Unit tests), מבוסס על JUnit.

A.23.12 PHP_Fork

מעטפת לפונקציית `pcntl_fork()` המאפשרת ניהול תהליכים (Multiprocessing) ב-PHP עם API דומה לשפת Java, כולל שיתוף זיכרון בין תהליכים.

A.23.15 Validate

חבילה לאימות נתונים שונים: מספרי כרטיסי אשראי, כתובות אימייל, תאריכים, כתובות URL ונתונים פיננסיים (כמו IBAN).

A.23.16 Var_Dump

מעטפת לפונקציית `var_dump` המאפשרת להציג מידע מובנה על משתנים במגוון פורמטים ויזואליים (HTML, XML, טקסט פשוט ועוד).

(A.23.18 Xdebug (PECL

הרחבה רבת עוצמה לניפוי שגיאות (Debugging). מספקת מעקב אחר פונקציות (Stack trace), ניתוח ביצועים (Profiling) ויכולת לעצור את הרצת הקוד באופן אינטראקטיבי.

A.24 עיבוד (Processing)

A.24.1 FSM

חבילה המממשת "מכונת מצבים סופית" (Finite State Machine).

A.25 מדע (Science)

A.25.1 Science_Chemistry

מחלקות לייצוג אובייקטים כימיים: אטומים, מולקולות ומקרו-מולקולות. כולל מנתחי קבצים לפורמטים כימיים ומידע על הטבלה המחזורית.

A.26 Streams (זרמים)

מימושי זרמים וכלי עזר ל-PHP.

A.26.1 bz2_filter

מאגר: PECL – רישיון: PHP – מאת: שרה גולמן (מובילה)

מימוש מסנן זרם לכיוון/פריסה בפורמט bzip2. מאפשר כיוון "על המשתמש" (inline) בכל זרם קלט/פלט של PHP.

A.26.2 oggvorbis

עטיפת fopen לקובצי OGG/Vorbis. מאפשרת פריסה של נתוני OGG לאודיו PCM ולהיפך.

A.26.5 Stream_Var

מאגר: PEAR – רישיון: PHP License

מאפשר גישה מבוססת זרם לכל משתנה. ניתן להתייחס למערכים כאל ספריות, ובכך להחליף ספריות וקבצים זמניים במשתנים בתוך האפליקציה.

A.27 Structures (מבנים)

מבני נתונים וטיפוסי נתונים מתקדמים.

A.27.1 Games_Chess

מאגר: PEAR – רישיון: PHP License

בנייה ואימות של משחק שחמט לוגי. מטפל בלוח השחמט ובניתוח פורמטים סטנדרטיים כמו FEN ו-SAN (תיאור מהלכים).

A.27.3 Structures_DataGrid

חבילה ליצירת מבנה דמוי טבלה (Grid) המבוסס על סט נתונים. תומכת בהוצאת פלט במגוון פורמטים כגון טבלת HTML, מסמך XML, גיליון Excel ועוד, כולל יכולות דפדוף ומיון.

A.27.4 Structures_Graph

ספרייה ליצירה וניהול של מבני נתונים מסוג גרף (Graph). מאפשרת בניית גרפים מכוונים ולא מכוונים ושליפת מאפיינים מהטופולוגיה שלהם.

A.27.6 Tree

ניהול עצים גנרי. תומך בבסיסי נתונים וב-XML כמקורות מידע. מאפשר עבודה עם עצים בזיכרון או קריאת צמתים לפי צורך (שימושי לעצים ענקיים).

A.28 System (מערכת)

A.28.2 System_ProcWatch

ניטור תהליכים רצים על בסיס קובץ הגדרות (XML, INI או מערך) שבו מגדירים תנאים ופעולות לביצוע.

A.29 Text (טקסט)

A.29.1 enchant

מאגר: PECL. מקשר לספריית libenchant, התומכת כמעט בכל כלי איות השפה (כולל hspell לעברית ו-uspell ליידיש).

A.29.3 panda

ספריית PDF חופשית ליצירת מסמכי PDF.

A.29.6 Text_Password

יצירת סיסמאות הניתנות להגייה (pronounceable) או כאלה שאינן ניתנות להגייה.

A.29.8 xdiff

הרחבה ליצירה והחלה של טלאים (patches) על קובצי טקסט וקבצים בינאריים.

A.30 Tools and Utilities (כלים ועזרים)

A.30.2 fann

מימוש של ספרית רשתות עצביות מלאכותיות (Artificial Neural Networks).

A.30.4 PhpDocumentor

כלי ליצירת תיעוד אוטומטי ישירות מקוד המקור (דומה ל-JavaDoc). תומך בפרמטרים HTML, PDF, CHM ו-XML.

(A.30.5 SPL (Standard PHP Library

הרחבה המממשת ממשקים ומחלקות ליעילות בגישה לנתונים, כגון iterators (איטרטורים) מתקדמים ללולאות וטיפול בספריות.

A.31 Web Services (שירותי רשת)

A.31.2 Services_Weather

ממשק לשירותי מזג אוויר מקוונים שונים. מאפשר חיפוש מיקומי תחזית ושליפת נתונים נוכחיים.

A.31.3 SOAP

מימוש פרוקסי ושירותי SOAP (פרוטוקול תקשורת מבוסס XML).

A.31.6 XML_RPC

מימוש PHP לפרוטוקול XML-RPC, כולל תמיכה בשרתי פרוקסי ואימות.

A.32 XML

A.32.1 XML_Beautifier

עיצוב מסמכי XML על ידי הוספת הזחות (indentation) ושבירת שורות להקלה על הקריאה.

A.32.6 XML_HTMLSax

מנתח (Parser) מבוסס SAX עבור מסמכי HTML או מסמכי XML שאינם בנויים היטב (badly formed).

A.32.9 XML_Parser

מנתח XML המבוסס על הרחבת expat המובנית ב-PHP. תומך במצבי עבודה מבוססי אירועים (callbacks).

A.32.11 XML_RSS

מנתח עבור מסמכי RSS ו-RDF (סיכומי אתרים).

A.32.12 XML_SaxFilters

תיאור: חבילה זו מספקת תשתית לשימוש במסנני Sax ב-PHP. סינון Sax הוא גישה שהופכת את ניתוח מסמכי ה-XML למודולרי וקל לתחזוקה.

הדגם מבוסס על "הורה" (parent) ו"ילד" (child): הורה הוא המסנן שנמצא "במעלה הזרם" ומקבל את אירועי ה-XML ראשון, בעוד שילד הוא מסנן "במורד הזרם" שאליו מואצלים האירועים. הנתח (Parser) עצמו נמצא תמיד בראש "אילן היוחסין" ומאציל את כל האירועים למסננים שתחתיו.

A.32.13 XML_Serializer

מאגר: PEAR – **רישיון:** PHP License – **מאת:** סטפן שמידט (מוביל)

"אולר שוויצרי" לקריאה וכתובה של קובצי XML. החבילה מסוגלת להפוך מבני נתונים מורכבים (כמו מערכים ואובייקטים) למסמכי XML ולהיפך. היא יכולה לשמש כחלופה לפונקציות `serialize()` ו-`unserialize()` המובנות, ואף מסוגלת לנתח קובצי RSS ולהופכם למערך מובנה בקלות.

A.32.14 XML_sql2xml

מחלקה המקבלת תוצאות משילתת SQL (אובייקט PEAR::DB) ומחזירה ייצוג XML שלהן. היא מתבססת על הרחבת DOMXML של PHP.

A.32.15 XML_Statistics

חבילה לשליפת מידע סטטיסטי ממסמכי XML, כגון כמות תגיות, מאפיינים (attributes), ישויות (entities) והנחיות עיבוד.

A.32.16 XML_SVG

מספקת ממשק תכנותי (API) מונחה עצמים לבניית מסמכי גרפיקה וקטורית מסוג SVG.

A.32.17 XML_svg2image

המרת קובצי SVG לתמונות PNG או JPEG בעזרת כלי חיצוני (apache-batik). דורשת תמיכה ב-Java ב-PHP.

A.32.18 XML_Transformer

מאפשרת לקשור פונקציונליות של PHP ישירות לתגיות XML, ובכך להפוך עץ XML של קלט לעץ XML של פלט ללא צורך בשימוש ב-XSLT.

A.32.19 XML_Tree

ייצוג נתוני XML במבנה של עץ ללא צורך בהרחבות חיצוניות כמו DOMXML.

A.32.20 XML_Util

אוסף שיטות עזר נפוצות לעבודה עם XML: יצירת רשימות מאפיינים ממערכים, יצירת תגיות, אימות שמות תגיות ועוד.

A.32.21 XML_Wddx

כלי לסדרול (Serialization) של WDDX. משמש כחלופה להרחבת ה-WDDX המובנית ומייצר קבצים הניתנים לעריכה נוחה (עם הזחות ושימוש ב-CDATA).

A.32.22 XML_XPath

מחלקה למניפולציה ושאליות על מסמכי XML באמצעות XPath ו-DOM. היא מאפשרת ניווט קל בעץ ה-DOM ומחזירה תוצאות כאובייקטי XPath_Result. דורשת את הרחבת domxml.

A.32.23 XML_XSLT_Wrapper

מספקת ממשק אחיד לביצוע טרנספורמציות XSLT באמצעות ספריות ופקודות שונות (כמו Sablotron, xsltproc, MSXML). היא מפשטת את העבודה מול מנועי XSLT שונים.

תיאור: מחלקה לבניית אפליקציות XUL (שפת ממשק המשתמש של Mozilla). היא מספקת API דומה ל-DOM ליצירת רכיבי ממשק מורכבים כמו טבלאות, עצים ותיבות לשוניות (tabs) כאובייקטי PHP.

נספח ב

סימוכין לפורמט phpDocumentor

"תיעוד הוא כמו סקס: כשהוא טוב, הוא טוב מאוד; וכשהוא רע, הוא עדיף על כלום." — דיק ברנדון

B.1 מבוא

מלבד תקני כתיבת קוד, לפרויקט PEAR יש שיטה סטנדרטית לתיעוד מחלקות וחבילות. שיטה זו משתמשת בכלי **phpDocumentor** כדי לייצר תיעוד HTML הניתן לדפדוף מתוך הערות בקוד המקור. הכלי הרשמי לתיעוד מחלקות PEAR הוא phpDocumentor (כתובת: <http://phpdoc.org>), המסוגל לייצר לא רק HTML, אלא גם PDF ו-Docbook XML. הוא דומה מאוד ל-JavaDoc ומשתמש ב"שפת סימון" דומה לתיעוד אלמנטים. ניתן להתקין אותו באמצעות הפקודה:

```
pear install phpDocumentor $
```

נספח זה יציג לכם את הכלי הרשמי, יחד עם דוגמאות לשימוש בו וכיצד לתעד את המחלקות שלכם.

B.2 הערות תיעוד (Docblocks)

הכלי מייצר תיעוד לאלמנטים המוטמעים בקוד כהערות. הכלי מזהה תשעה סוגים שונים של מקטעים: משתנה גלובלי, include, קבוע, פונקציה, define, מחלקה, משתנה, מתודה (method) ועמוד.

כל קובץ בפרויקט ה-PHP שלכם צריך להתחיל ב-**Docblock ברמת העמוד**, המתעד היבטים כלליים (כמו מחבר, שם החבילה וכדומה). Docblock תמיד מתחיל ברצף התווים `/**`, בניגוד להערות "רגילות" שמתחילות בדרך כלל ב-`/*`:

```
PHP
```

```
*/
```

```
@Examples package *
```

```
/*
```

לאחר מכן, ניתן לתעד אלמנטים ספציפיים. הערה כוללת תיאור קצר (בשורה אחת), תיאור מפורט (שיכול לכלול תגיות HTML כמו `` או `<code>`) ולאחר מכן מקטע מילות מפתח (Tags) המתחילות בסימן `@`.

B.3 מילון תגיות (Tag Reference)

B.3.1 abstract

(זמין ב-PHP 4 בלבד; ב-PHP 5 המידע נשלף אוטומטית).

משמש לתיעוד מחלקה מופשטת או פונקציה שעל מחלקות יורשות לממש.

B.3.2 access

(זמין ב-PHP 4 בלבד).

קובע אם אלמנט הוא `public`, `protected` או `private`. אלמנטים פרטיים לא יופיעו בתיעוד המשתמש אלא אם תצוין פקודה מיוחדת. לפי תקני PEAR, פונקציות פרטיות צריכות להתחיל בקו תחתון (`_`).

B.3.3 author

מציין את שם המחבר וכתובת האימייל שלו: `<author Name <email@example.com>`.

B.3.4 category

משייך מחלקה לקטגוריה רחבה (כמו Database או XML). שימושי מאוד עבור PEAR.

B.3.5 copyright

תיעוד זכויות יוצרים: `copyright Copyright © 2026, Name@`.

B.3.6 deprecated

מסמן פונקציות מיושנות שאינן מומלצות לשימוש, עם הסבר מאיזו גרסה הן הופסקו.

B.3.7 example

מאפשר להוסיף דוגמאות קוד. ניתן להוסיף קוד בתוך ההערה או לקשר לקובץ דוגמה חיצוני באמצעות: `example_file.php@`.

B.3.8 filesource

גורם ל-phpDocumentor לייצר גרסה של הקוד עם הדגשת תחביר (Syntax Highlighting) ולקשר אליה מהתיעוד.

B.3.9 final

(PHP 4 בלבד). מציין שלא ניתן לדרוס (overload) את המחלקה או התכונה הזו.

B.3.10 global

מתעד שימוש במשתנה גלובלי בתוך פונקציה, או מגדיר משתנה כגלובלי ברמת התסריט כולו.

B.3.11 ignore

משמש להחרגת אלמנטים מהתיעוד הסופי (למשל כדי למנוע כפילויות במקרה של הגדרות מותנות).

B.3.13 internal

תיעוד המיועד לשימוש פנימי בלבד (בתוך החברה) ולא יופיע בתיעוד הציבורי.

B.3.14 licence

קישור לרישיון הקוד (למשל PHP License).

(B.3.16 link (inline

יצירת קישור בתוך טקסט התיעוד לכתובת URL חיצונית או לאלמנט אחר בקוד:

```
link http://www.example.com{@}
```

B.3.17 name

תחביר:

```
name@ <שם_יפה>
```

מילת מפתח זו מעניקה שם "ידידותי" למשתנה גלובלי. בדוגמה הבאה, בתיעוד שיוצר ייעשה שימוש ב-\$foo במקום ב-\$GLOBALS['foo']:

```
PHP
```

```
**/
```

* דוגמה לתיעוד משתנה גלובלי

```
* $foo@ name
```

```
* $GLOBALS['foo@' string 'global']
```

```
/*
```

```
$Foobar = "GLOBALS['foo']";
```

B.3.18 package

תחביר:

```
@package <שם_חבילה>
```

התגית @package משמשת לקיבוץ אלמנטים (וחבילות-משנה ב-phpDocumentor). זהו פריט הקיבוץ ברמה העליונה ובדרך כלל הוא משויך לחבילת PEAR. ראו את הדוגמה באיור B.1, המשתמשת בתגיות package ו-subpackage כדי לתעד פונקציות במבנה בעל שתי רמות.

איור B.1: מבנה חבילות.

```
PHP
```

```
**/
```

* ניהול מטמון (Cache)

```
package Cache@ *
```

```
/*
```

```
} ()function Cache
```

```
{
```

```
**/
```

```
* שמירה במטמון בבסיס נתונים
```

```
package Cache@ *
```

```
subpackage Cache_DB@ *
```

```
/*
```

```
} ()function Cache_DB
```

```
{
```

```
**/
```

```
* שמירה במטמון בבסיס נתונים MySQL
```

```
package Cache@ *
```

```
subpackage Cache_DB@ *
```

```
/*
```

```
} ()function Cache_DB_MySQL
```

```
{
```

B.3.19 param

תחביר:

```
<param (type | object_definition) <$variable@
```

פרמטרים של פונקציות מתועדים באמצעות התגית `param@`. להלן מספר דוגמאות:

PHP

```
**/
```

* פונקציה לחיבור מספרים והכפלתם בשתיים

* @param float \$a זהו האלמנט הראשון שייכלל בתוצאה

* @param int \$b וכאן יש לנו את הפרמטר השני

* @return mixed

/*

(function addNumbersAndMultiplyByTwo (\$a, \$b

}

;return (\$a + \$b) * 2

{

phpDocumentor מזהה את ערך ברירת המחדל של משתנה מתוך קוד המקור, וכולל אותו באופן אוטומטי בתיעוד שנוצר. להלן דוגמה מורכבת יותר:

PHP

*/

* החזרת שורות

*

* הרצת שאילתה על חיבור בסיס הנתונים והחזרת מספר השורות שצוין, אם צוין

* @private

* @param resource \$conn משאב החיבור לבסיס הנתונים

* @param string \$query השאילתה

* @param int \$limit הגבלה למספר זה של שורות מוחזרות

* @return array

/*

(function _runQuery (\$conn, \$query, \$limit = 0

}

// ... קוד הפונקציה ...

{

B.3.20 return

תחביר:

`<return (type | object_definition@`

השתמשו בתגית `return@` כדי לתעד את סוג הערך המוחזר מהפונקציה שלכם:

PHP

`**/`

`* @param string $filename שם הקובץ של התמונה`

`* @return resource משאב תמונה מסוג GD`

`/*`

`(function returnNiceGif ($filename`

`}`

`;(return imagecreatefromgif ($filename`

`{`

B.3.21 see

תחביר:

`<see@ >אלמנט`

באמצעות התגית `see@`, ניתן להוסיף קישורים לאלמנטים אחרים בתיעוד. כל סוג אלמנט של phpDocumentor נתמך כפרמטר לתגית `:see@`

PHP

`**/`

`* מחבר מספרים`

`* @add::string (see)`

`/*`

`(function addNumbers ($number1, $number2`

`}`

```
;return $number1 + $number2
```

```
{
```

```
**/
```

```
* מחלקה למניפולציה של מחרוזות
```

```
/*
```

```
} class string
```

```
**/
```

```
* מחבר מחרוזות
```

```
see addNumbers@ *
```

```
/*
```

```
(function add ($string1, $string2
```

```
}
```

```
;return $string1 . $string2
```

```
{
```

```
{
```

B.3.22 since

תחביר:

```
@since <גרסה>
```

תגית זו מתעדת מתי אלמנט נוסף ל-API. פורמט מחרוזת התיאור הוא חופשי. להלן דוגמה מתוך מחלקת PEAR בשם HTML_Common:

```
PHP
```

```
**/
```

```
* מחזירה את ה-tabOffset
```

```
*
```

```
@since 1.5 *
```

```

return void@ *
/*
function getTabOffset
}
;return $this->_tabOffset
{

```

B.3.23 static

זמין עבור PHP 4 בלבד.

תחביר:

```
static@
```

תגית זו מתעדת שניתן לקרוא למתודות באופן סטטי (לדוגמה: `::()`Foo::Bar

PHP

```
*/
```

* מחלקה foo המבצעת פעולת bar סטטית

```
/*
```

```
} class foo
```

```
*/
```

* ניתן לקרוא לפונקציה זו באופן סטטי

```
static@ *
```

```
/*
```

```
} () function bar
```

```
{
```

```
{
```

```
;()foo::bar
```

B.3.24 staticvar

זמין עבור PHP 4 בלבד.

תחביר:

`<staticvar (type | object_definition) <$variable@`

התגית `staticvar@` מתעדת משתנה סטטי בתוך פונקציה. משתנים סטטיים אינם נהרסים כאשר הפונקציה מסתיימת. הדוגמה הבאה תדפיס 123:

PHP

`**/`

* דוגמה למשתנה סטטי בתוך פונקציה

* `@integer $count staticvar` סופר את מספר הפעמים שנקראה הפונקציה.

`/*`

`} ()function foo`

`;static $count`

`;++count$`

`;"echo $count. "\n`

`{`

`foo(); // 1`

`foo(); // 2`

`foo(); // 3`

B.3.25 subpackage

תחביר:

`<שם_תת_חבילה> subpackage@`

ניתן להשתמש ב-`subpackage` כשכבת קיבוץ נוספת עבור אלמנטים בחבילה שלכם. ראו את התיאור של תגית ה-`package@` לקבלת דוגמה.

B.3.26 todo

תחביר:

`<תיאור> todo@`

באמצעות תגית ה-`todo@`, ניתן לתעד שינויים שעדיין יש לבצע באלמנט ספציפי. לדוגמה:

PHP

*/

* @todo לתעד פרמטרים

/*

} (function todo_example(\$a, \$b

{

B.3.27 uses

תחביר:

@uses <אלמנט>

תגית זו מבצעת פעולה דומה לתגית `@see`, פרט לכך שהיא יוצרת קישור דו-כיווני בין האלמנט ה"בשימוש" לבין האלמנט שבו נעשה שימוש בתגית `@uses`. כלי ה-phpDocumentor מבצע זאת על ידי הוספת תגית מדומה בשם `@usedby` לאלמנט שאליו מצביעה תגית ה-`@uses`.

B.3.28 var

תחביר:

@\$variable (type | object_definition) <תיאור>

התגית `var` מתעדת את הסוג של משתני מחלקה. הסוג צריך להיות סוג נתונים תקני של PHP, או "mixed" אם המשתנה יכול להכיל סוגים שונים:

PHP

*/

* מחלקה ה"מחקה" מבנה (struct) כמו בשפת C

/*

} class person

*/

* @string \$name שמו של האדם

/*

;var \$name

*/

I^* $\{$ [illegible]

X	X	X	X	X	X	X	X	X	Deprecat ed
X	X	X	X	X	X	X	X	X	Example
X	X	X	X	X	X	X	X	X	Ignore
X	X	X	X	X	X	X	X	X	internal
X	X	X	X	X	X	X	X	X	Link
X	X	X	X	X	X	X	X	X	see
X	X	X	X	X	X	X	X	X	since
X	X	X	X	X	X	X	X	X	version
X								X	name
	X				X			M	global
	X				X				param
	X				X				return
X			X						package

X	X								static
---	---	--	--	--	--	--	--	--	--------

B.5 שימוש בכלי phpDocumentor

כדי לייצר את התיעוד מקוד המקור, תזדקקו לכלי phpDocumentor המותקן דרך PEAR. הכלי כולל מספר פרמטרים חשובים:

טבלה B.2: פרמטרים של כלי ה-phpDocumentor

אפשרות	הערות	דוגמה
-f, --filename	רשימת קבצים לניתוח (מופרדים בפסיקים).	index.php,lib.php -f
-d, --directory	רשימת ספריות לניתוח.	core,libs -d
-t, --target	ספריית היעד עבור התיעוד שיווצר.	docs/ -t
-ti, --title	כותרת התיעוד שיווצר.	My Project" -ti"
-pp, --parseprivate	הכללת אלמנטים פרטיים ופנימיים בתיעוד.	on -pp
-o, --output	פורמט הפלט והתבנית לשימוש.	HTML:frames:default -o
-s, --sourcecode	הכללת קוד מקור עם הדגשת תחביר.	on -s

פקודת הרצה לדוגמה:

```
$ phpdoc -d directory -pp on -s on -o HTML:frames:default -t outputdir
```

דוגמה לתיעוד שנוצר

מתוך קובצי המקור של הפרויקט, phpDocumentor מייצר ממשק דפדפן המציג את היררכיית המחלקות והמתודות.

- **איור B.2 ו-B.3:** מציגים את התייעוד עבור המחלקה `SumNumberElements`. ניתן לראות בבירור את עץ הירושה (Inheritance) המראה שהיא יורשת מהמחלקה `Sum`.
- **איור B.4:** מציג את היחסים בין כל המחלקות בחבילה כעץ, ומראה אילו מחלקות הן תתי-מחלקות של `Sum`.
- **איור B.5:** מציג אינדקס של כל האלמנטים הזמינים בחבילה: מודולים, מחלקות, פונקציות, משתנים וקבועים.

נספח C: מדריך התחלה מהירה ל-Zend Studio

C.1 גרסה x.3.5

Zend Studio היא סביבת פיתוח משולבת (IDE) המיועדת למפתחי PHP מקצועיים. זהו הכלי היחיד המקיף את כל רכיבי הפיתוח הדרושים למחזור החיים המלא של אפליקציית PHP.

C.3 אודות Zend

Zend היא "חברת ה-PHP". מייסדיה, **אנדי גוטמנס** ו**זאב סורסקי**, הם היוצרים והמחדשים של שפת PHP ומנוע ה-Zend Engine (קוד פתוח). המשימה של Zend היא להביא את הדור הבא של המוצרים והשירותים לפיתוח, פריסה וניהול של אפליקציות PHP ברמה ארגונית (Enterprise).

Zend Studio Client C.4: סקירה כללית

Zend Studio מפשט את משימות הפיתוח הכרוכות ביצירת אפליקציות PHP:

- **פיתוח:** השלמת קוד מתקדמת, ניהול פרויקטים והדגשת תחביר.
- **ניפוי שגיאות (Debugging):** מנפה שגיאות מרחבי המאפשר עבודה ישירות מול השרת, ומנפה שגיאות פנימי למחשב המקומי.
- **ניהול:** כלי אבחון מתקדמים לניהול מחזור חיי התוכנה.

נספח ג

נספח C: מדריך התחלה מהירה ל-Zend Studio

C.1 גרסה x.3.5

המידע במסמך זה עשוי להשתנות ללא הודעה מוקדמת ואינו מהווה התחייבות מצד חברת Zend Technologies, Ltd. אין לשכפל או להעביר שום חלק ממדריך זה בכל צורה או אמצעי לכל מטרה שאינה לשימוש האישי של הרוכש, ללא אישור בכתב מהחברה. כל הסימנים המסחריים שייכים לבעליהם.

C.2 אודות המדריך

מדריך זה נועד לסייע לך להתחיל לעבוד עם התוכנה באופן מיידי. למידע מלא על התכונות הנתמכות, יש לעיין בעזרה המקוונת (Online Help) המצורפת לאפליקציה.

C.3 אודות Zend

בפשטות, Zend היא חברת ה-PHP. מייסדיה—אנדי גוטמנס וזאב סורסקי—הם היוצרים והמחדשים של שפת PHP ומנוע ה-Zend Engine בקוד פתוח. החברה שמה לה למטרה להביא את מוצרי הפיתוח והניהול

של PHP לרמה ארגונית (Enterprise). כיום, PHP היא שפת הסקריפט הפופולרית ביותר ברשת, ומנוע ה-Zend Engine מניע למעלה מ-15 מיליון אתרי אינטרנט.

C.4 Zend Studio Client: סקירה כללית

סביבת Zend Studio עוצבה עבור מפתח ה-PHP המקצועי. זוהי סביבת הפיתוח (IDE) היחידה המקיפה את כל הרכיבים הדרושים למחזור החיים המלא של אפליקציית PHP. היא מסייעת בהאצת תהליך הפיתוח ומניבה קוד חסין ובטוח.

התוכנה מפשטת משימות כגון:

- **פיתוח:** השלמת קוד (Code Completion) מתקדמת, בוחני קוד (Inspectors) ברמת הקובץ והפרויקט, וחיפוש רוחבי בין קבצים.
- **ניפוי שגיאות (Debugging):** מנפה שגיאות מרחוק (Remote Debugger) לעבודה מול שרת, ומנפה פנימי לעבודה מקומית.
- **ניהול:** כלי אבחון מתקדמים כגון ה-Profiler ו-Code Analyzer.
- **פריסה (Deployment):** פרסום האפליקציה לשרת אירוח באמצעות FTP/SFTP או אינטגרציית CVS עוצמתית.

C.4.1 רכיבי המערכת

המערכת מורכבת משני רכיבים עיקריים:

1. **Zend Studio Client:** ממשיק המשתמש המותקן במחשב המקומי. כולל את סרגל הכלים לדפדפן, מדריך ה-PHP ורכיבי ניפוי שגיאות ל-PHP 4 ו-5.
2. **Zend Studio Server:** מוסיף יכולות ניפוי שגיאות ו-Profiling לשרתים קיימים, ומאפשר הקמת שרת אינטרנט מבוסס Apache ו-PHP במידת הצורך.

C.5 עריכת קובץ

כדי לערוך קובץ, פשוט פתח את התוכנה והתחל לכתוב. התוכנה כוללת תכונות חסכות זמן כמו **השלמת קוד (Code Completion)** המציגה אפשרויות רלוונטיות ל-PHP או HTML באופן אוטומטי. למשל, הקלדת התו `<` תפתח רשימת תגיות HTML לבחירה.

C.6 עבודה עם פרויקטים

ניהול פרויקטים מאפשר לך לארגן את הקבצים שלך בקלות:

1. עבור אל **Project > New Project**.
2. הזן שם לפרויקט.
3. הוסף נתיבים (Paths) לקבצים ולספריות שירכיבו את הפרויקט.
4. הגדר את הגדרות ניפוי השגיאות (Debug Mode) – מקומי או מרחוק.

C.7 הרצת מנפה השגיאות (Debugger)

התוכנה תומכת בשני סוגי ניפוי:

- **Internal Debugger (פנימי):** לניפוי אפליקציות PHP עצמאיות (דורש התקנת לקוח בלבד).
- **Remote Debugger (מרוחק):** לניפוי קבצים הרצים על שרת אינטרנט מרוחק.

C.7.1 ניפוי שגיאות פנימי - שלבי עבודה:

1. פתח את הקובץ המבוקש (למשל `DebugDemo.php`).

2. לחץ על כפתור ה-**Debug** בסרגל הכלים. המערכת תעצור בנקודת העצירה (Breakpoint).
3. השתמש ב-**Step Over** כדי להתקדם שורה אחת שורה.
4. עמוד עם הסמן מעל משתנים כדי לראות את ערכם (ToolTip).
5. השתמש ב-**Step Into** כדי להיכנס לתוך פונקציות, וב-**Step Out** כדי לצאת מהן חזרה לקוד הראשי.
6. התוצאות יופיעו בחלון ה-**Output** והודעות מערכת יופיעו בחלון **Debug Messages**.

C.7.1 מנפה שגיאות פנימי (Internal Debugger)

השתמש בתיבת הדו-שיח "Tip of the Day" (טיפ היום) כדי לגשת לקוד לדוגמה ולהסבר קצר על ניפוי שגיאות:

1. הפעל את Zend Studio Client; לחלופין, בחר בתפריט **Help > Tip of the Day**.
2. מתוך תיבת הדו-שיח "Tip of the Day", לחץ על הכפתור המתאים. הקובץ `DebugDemo.php` ייפתח בחלון העריכה.
3. בסרגל הכלים של הלקוח, לחץ להפעלת מנפה השגיאות. האייקון יופיע בזמן שרת הניפוי של Zend רץ, ויישאר על המסך עד שמנפה השגיאות יזהה נקודת עצירה (breakpoint) בשורה 46.
4. לחץ על **Step Over** (דלג מעל) מספר פעמים עד שהסמן יגיע לשורה 51.
5. הנח והחזק את הסמן מעל המשתנים `worker_name`, `$worker_address` ו-`worker_phone$`. יופיע ToolTip (חלופית צפה) המציג את ערכי המשתנים.
6. לחץ על כפתור **Step Into** (כנס פנימה). מנפה השגיאות יתקדם לשורה 26.
7. בחלון הניפוי (Debug), לחץ על לשונית ה-**Stack** (מחסנית) ולחץ על הצומת מימין ל-`row_color`. עץ מחסנית הקריאות יתרחב ויציג את המשתנה `i`.
8. לחץ על כפתור **Step Out** (צא החוצה). הסמן יחזור לשורה 51.
9. לחץ על כפתור החץ ימינה. פלט יופיע בחלון ה-Output; הודעת **Notice** תופיע בחלון הודעות הניפוי (Debug Messages).
10. בחלון הודעות הניפוי, לחץ פעמיים על ה-Notice. הסמן יקפוץ לשורה 61 בחלון העריכה.
11. הנח את הסמן בחלון ה-Debug Output, לחץ על הלחצן הימני ובחר **Show in Browser** מהתפריט. ייפתח חלון דפדפן המציג את תוכן חלון הפלט.

C.7.2 מנפה שגיאות מרוחק (Remote Debugger)

מנפה השגיאות המרוחק דומה מאוד בתכונותיו למנפה הפנימי, פרט לכך שהקוד מבוצע על שרת אינטרנט מרוחק. אם ברצונך לנפות אפליקציית רשת טיפוסית מבוססת דפדפן, עיין בסעיף "Debug URL" בהמשך הנספח.

כדי להשתמש במנפה המרוחק, יש להגדיר תחילה את הלקוח (Client) ואת השרת.

להגדרת הלקוח:

1. מהתפריט הראשי, בחר **Tools > Preferences**. חלון ההעדפות יופיע.
 2. בחר בלשונית **Debug**.
 3. באזור ה-Debug Server Configuration, בחר במצב ניפוי (Server/Internal).
 4. לחץ על **OK**.
- כעת ניתן לנפות את הקובץ הנוכחי באמצעות המנפה המרוחק.
הערה: ניתן להפעיל/להשבית ניפוי מרוחק גם מחלון מאפייני הפרויקט (Project Properties).

C.7.3 ניפוי כתובת אתר (Debug URL)

אפשרות זו מאפשרת להריץ הליך ניפוי על דפים המותקנים כעת באתר האינטרנט. ניתן לאתחל את סשן הניפוי מהלקוח על ידי בחירה בתפריט 'Debug URL' או דרך סרגל הכלים של Zend בדפדפן.

שרת Zend Studio נותן עדיפות עליונה לקבצים שעליהם אתה עובד. לשם כך, השרת פועל לפי ההיררכיה הבאה בעת בקשת קבצים:

1. בודק אם הקובץ שנקרא פתוח כעת בלקוח; אם נמצא, הוא משתמש בו.
2. מחפש את הקובץ בנתיב של הפרויקט הפתוח.
3. מחפש את הקובץ בנתיב השרת.

היררכיה זו מאפשרת לך להימנע מהעלאת (Upload) הגרסאות האחרונות שלך בכל פעם. לדוגמה, אם אתה גולש באתר ומגלה דף פגום, תוכל להתחיל ניפוי ישירות מהדפדפן. לאחר התיקון, תוכל להריץ ניפוי חדש ולראות את התוצאה ללא צורך בהעלאת הקבצים ששוננו תחילה.

C.8 הגדרת שרת Studio עבור ניפוי ופרופילינג מרוחקים

מטעמי אבטחה, על המשתמש להיות מוגדר כמשתמש מורשה ב-Zend Server Center. רק כתובות IP מורשות יכולות לגשת לשירותים אלו.

להגדרת משתמש מורשה:

1. התחבר ל-Zend Server Center כמנהל (Administrator) מכתובת IP מורשית.
2. פתח את מסך הגדרות האבטחה (**Security Settings**).
3. בלשונית **Manage IP Permissions**, הוסף את כתובת ה-IP המבוקשת לרשימת המארחים המורשים (**Allowed Host List**).
4. וודא שכתובת ה-IP אינה מופיעה ברשימת המארחים החסומים (**Denied Host List**).
5. לחץ על **OK**.
6. אתחל מחדש את שרת האינטרנט.

C.9 הרצת הפרופילר (Profiler)

הפרופילר המשולב עוזר לאופטימיזציה של ביצועי האפליקציה על ידי איתור "צווארי בקבוק" (bottlenecks) – קטעי קוד הצורכים זמן טעינה מופרז.

הפרופילר מבצע את הפעולות הבאות:

- ניטור קריאות לפונקציות ומספר הפעמים שקטע קוד בוצע.
- חישוב זמן הביצוע הכולל והפקת דוחות מפורטים.
- תצוגה גרפית של חלוקת הזמן והשוואה סטטיסטית בין פונקציות.
- הצגת המבנה ההיררכי של הפונקציות שהשתתפו בהרצת הסקריפט.

להרצת הפרופילר:

1. מתפריט **Tools**, בחר **Profile URL**.
2. אשר את כתובת האתר המוגדרת כברירת מחדל או שנה אותה ולחץ על **OK**. הדפדפן יציג את הדף, ולאחר מספר שניות יופיע חלון מידע של הפרופילר עם שלוש לשוניות: **Profiler Information** (מידע כללי ותרשים עוגה), **Function Statistics** (רשימת קבצים וסטטיסטיקת פונקציות), ו-**Call Trace** (תצוגה היררכית של סדר הפעולות).

C.10 תמיכה במוצר

Zend מחויבת לספק שדרוגים ותמיכה. ברכישת מוצר, מקבלים 60 ימי תמיכה חינם להתקנה ולהגדרה. תמיכה משודרגת (Enhanced Product Support) זמינה במינוי שנתי וכוללת שדרוגים משמעותיים ומענה בעדיפות גבוהה. שעות הפעילות הן שני עד ששי (GMT+2).

ניתן לקבל תמיכה דרך ה-Helpdesk באתר החברה, דרך תפריט העזרה בתוכנה, או דרך הפורומים של Zend.

C.11 תכונות עיקריות

Zend Studio משלב את כל הכלים הדרושים בממשק מאוחד:

- ניפוי שגיאות (Debug) ופרופיילינג.
- אינטגרציית CVS להשוואת קבצים וניהול גרסאות.
- ניהול פרויקטים לחיפוש וניווט קל.
- השלמת קוד (Code Completion) ותבניות קוד (Code Templates) מתקדמות ל-PHP 5.
- הדגשת תחביר (Syntax highlighting) ל-PHP, HTML ו-JavaScript בו-זמנית.
- עריכה ופריסה חלקה מול שרתי FTP.
- **ניתוח קוד סטטי:** איתור בעיות באפליקציה עוד לפני הרצתה!
- ניפוי ופרופיילינג ישירות מהדפדפן, גם עבור טפסים מורכבים או אפליקציות מבוססות Session.

להלן תרגום האינדקס לעברית:

סמלים

- `>?, 113`
- `.` (אופרטור שירשור), 32
- `” “` (מרכאות כפולות), מחוזות, 19–20
- `==` (אופרטורים של שוויון), 42
- `!#` (hash-bangs), סקריפטים של PHP בממשק שורה (512–511, CLI)
- `?` (סימן שאלה), 39
- `&` (סימן reference - התייחסות), 8
- `@` (אופרטור השתקה), 39
- `’ ‘` (מרכאות יחידות), מחוזות, 20
- `configure, 486/.`
- `COOKIE, 115_$`
- `GET, 115_$`
- `POST, 115_$`
- `key, 26$`
- `this$`
- גישה למתודות, 59–62
- גישה למאפיינים (59–61, properties)
- מתודות סטטיות, 64–65
- מאפיינים סטטיים, 62–64
- ב-PHP 5, 437–440
- `type, 130$`
- `value, 26$`
- `autoload(), 7, 80, 82, 197`
- `call(), 87, 109`

- `construct` () (בנאי), 3, 57
- `destructor` () (מפרק), 3
- `toString` (), 76–77
- 118N (בינאום), 567
- 118Nv2, 567

A

- a, 366-
- **ab** (כלי Apache Benchmarking), 457–458
- **abstract** (מופשט), 616–615
- **abstract classes** (מחלקה מופשטת), 5, 72–73
- **מתודות מופשטות**, 6, 73–72
- **שגיאות מופשטות** (abstracted errors), PEAR DB, 186
 - קודי שגיאה, 187–186
 - טיפול בשגיאות, 187
- **Acceleration Mode** (ZPS - מצב האצה), 472
- **גישה** (accessing), 616–617
 - אלמנטים במערך, 24
 - קבצים, 261
 - פונקציות, 264–262
 - מתודות באמצעות `this`, 59, 61–62\$
 - מערכים מקוננים, 26
 - מאפיינים באמצעות `this`, 59–61\$
 - משאבים (resources), 497–498
 - מתודות סטטיות באמצעות `this`, 64–65\$
 - מאפיינים סטטיים באמצעות `this`, 62–64\$
 - ערכי `zval`, 500–501
- **הוספת הנחיות INI להרחבות**, 504–503
- **APEX** (מערכת חשיפה לצילום), 327
- **addslashes** (), 128
- **APC** (Advanced PHP Cache), 470, 530
- **APD** (Advanced PHP Debugger), 461, 588
 - ניתוח נתוני מעקב (trace data), 462–465
 - התקנה, 462–461
- **API**, 422
 - extension API (הרחבה), 490
 - reflection API (שיקוף), 105–103
 - עטיפת הרחבות צד-שלישי, 493
- **ארכיטקטורה (architecture)**
 - סקריפט אחד לכל פונקציה, 144
 - סקריפט אחד משרת את כולם, 143
 - הפרדת לוגיקה מעיצוב (layout), 144–146
- **Archive_Tar**, 548
- **Archive_Zip**, 548
- **ארגומנטים** (arguments), 487
- **מערכים** (arrays), 23

- שינוי/יצירת אלמנטים, 25
- אינדקסים במערך, 23, 196
- קריאת ערכים, 25
- מעבר על מערך (traversing), 30
- מערכים אסוציאטיביים, 88
- **ArrayAccess** (ממשק), 88
- **הקצאת this\$ ב-PHP 5**, 437, 439–440
- **אופרטורים של הקצאה**, 32–33
- **אטומי (atomic)**, 278
- **מאפיינים (attributes)**, 221 (בנושא XML/PEAR)
- **Auth** (אימות), 392, 398, 527
- DB ונתוני משתמשים, 394–396
- קבצי סיסמאות, 393–394
- אבטחה, 396
- **אימות (authentication)**, 527–530
- **author** (מחבר), 617
- **אופטימיזציה אוטומטית**, 470–472

B

- **B**, 365–
- **תאימות לאחור (backward compatibility)**, 408
- **תרשימי עמודות (bar charts)**, 320–325
- **Benchmark** (מבחני ביצועים), 457, 530
- **ApacheBench**, 457–458
- **ספריית binaries**, 353
- **BLOB** (אובייקט בינארי גדול), 158
- **אופרטורים ביטויזיים (bitwise)**, 35
- **בלוקים (blocks)**, 384–387
- **Booleans** (בוליאנים), 22
- **טפסי הגשה חסיני בוטים**, 315–320
- **צווארי בקבוק (bottlenecks)**, 459–460
- **break**, 43
- **בניית חבילות (building packages)**, 411–415

C

- **C / -c**, 356–
- **C (שפת תכנות)**
- הכללה בחבילות PEAR, 428
- כתיבת הערות, 14
- **++C**
- ירושה, 70
- כתיבת הערות, 15
- **Cache** (מטמון), 451–453, 531
- APC, 470, 530

- Cache_Lite, 399–401, 531
- מטמון שאליות מסד נתונים, 455–453
- מטמון פלט (output caching), 456
- **Calendar** (לוח שנה), 542
- רגישות לאותיות רישיות (30), **case sensitivity**
- אופרטורים של המרה (38), **cast operators**
- תפיסת שגיאות (207, 216–218), **catching errors**
- המרה בין סטים של תווים (330–335), **character set conversions**
- מחלקות (56, 422), **classes**
 - הצהרה, 57
 - מחלקות סופיות (4, 76), **final**
 - חברים סטטיים, 5
- **CLI** (ממשק שורת פקודה), 507
 - ניתוח אפשרויות (512–515), **parsing**
 - סביבת סקריפטים, 512–510
- שיבוט אובייקטים (4, 66–67), **cloning**
- הערות (14, 613–615), **comments**
- תאימות (433–445), **compatibility**
 - מצב תאימות (435), **compatibility mode**
- קבועים (30–31), **constants**
- בנאים (57–58, 442), **constructors**
- עוגיות (131–133), **cookies**
- **XSLT** (המרת XML), 239–243

D

- טיפול בנתונים (301), **data handling**
- תאריכים (301–305, 542), **dates**
- **DB** (שכבת הפשטת מסד נתונים), 176, 534–533
- ניפוי שגיאות (**debugging**)
 - Zend Studio Debugger, 648
- הצהרה (**declaring**) על מחלקות, 57
- תלות (**dependencies**) ב-PEAR, 423–426
- תבניות עיצוב (94–109), **design patterns**
- מפרקים (58–59), **destructors**
- מדריכים/ספריות (353, 359–360), **directories**
- **DOM** (מודל אובייקטי של מסמך), 9, 231–222
- **DSN** (שם מקור נתונים), 178

E

- **...E_ERROR, E_NOTICE, E_STRICT** (רמות שגיאה), 202–201, 441
- עריכת קבצים, 647
- דואר אלקטרוני (418), **email**
- הטמעה (**embedding**) של PHP ב-HTML, 14, 112–114
- הצפנה (124–127, 543–544), **encryption**

- שגיאות (errors), 191
 - שגיאות תחביר, 192
 - שגיאות זמן ריצה, 201
 - שגיאות לוגיות, 197
- חריגות (exceptions), 216–218

F

- תבנית פקטורי (factory pattern), 97–101
- מערכת קבצים (file systems), 548–549
- קבצים (files), 261
 - נעילת קבצים, 277–276
 - העלאת קבצים, 142–137
- פילטרים (filters), 127
- פונקציות (functions), 48, 293
 - פונקציות מובנות, 456
 - פונקציות מוגדרות משתמש, 49–48

G

- GD (ספריות גרפיקה), 325–314
- משתנים גלובליים, 502–501, 118–117
- גרפיקה, 329–314

H

- HTML, 550–560
 - ניהול מאפיינים, 390
 - ניהול אלמנטים, 391
- HTTP, 561–563
- HTTPS, 397

I

- iconv (הרחבה), 334
- תמונות (images), 563–566
- ירושה (inheritance), 68–75
- התקנה (installing)
 - PEAR, 350–354
 - Zend Studio, 645–647
- ממשקים (interfaces), 4, 55, 74–75
- בינאום (internationalization), 566–568
- איטרטורים (iterators), 89–94, 170–172

להלן תרגום האינדקס לעברית:

J-K

****JPEG****, בהשוואה ל-
PNG**, 320**
kadm5**, 574**
(KCachegrind (Zdebug****
470–468
****Keyed-Hashing**** לאימות הודעות. ראה ****HMAC****
**** (keywords)** מילות מפתח
abstract**, 615–616**
access**, 616–617**
author**, 617**
category**, 618**
copyright**, 618**
deprecated**, 618**
example**, 619**
filesource**, 620**
final**, 4, 620–621**
global**, 621**
ignore**, 622**
inheritdoc (inline)**, 622**
internal (inline)**, 622–623**
licence**, 623**
link**, 623**
link (inline)**, 623**
name**, 624**
new**, 57–58**
package**, 624, 626**
param**, 626–627**
return**, 627**
see**, 627**
since**, 628**
static**, 628**
staticvar**, 629**
subpackage**, 629**
todo**, 630**
uses**, 630–631**
var**, 631**
version**, 631**
Knowles, Alan**, 613**

L

**** (language features)** תכונות שפה

ערכות תווים, 329
 מודל מונחה עצמים, 5–1
 שכבות (layers), 360
 פריסה (layout), הפרדה מ-
 לוגיקה, 144–146
 התאמה עצלה (lazy matching), 288–289
 lead, 418
 ספריות (libraries)
 ספריות GD. ראה ספריות GD
 libxslt, 239
 ספריית libxslt, 239
 licence, 623
 אלמנט license, 418–419
 מגבלות של PHP 3 ו-4, 2
 limitQuery (PEAR DB), 180–181
 link, 623
 link (inline), 623
 הפצת Linux PHP, התקנת, 350 PEAR
 list (), 28–30
 LiveUser, 529
 איזון עומסים (load-balancing), לפי
 מזהה סשן (session id), 398
 נעילת קבצים, 276–277
 Log, 568
 log_errors, 130
 log_errors (בוליאני), 203
 log_errors_max_len (שלם), 203
 Log_Parser, 568
 רישום לוגים (logging), 568
 לוגיקה, הפרדה מ-
 פריסה (layout), 144–146
 שגיאות לוגיות, 197
 אופרטורים לוגיים, 34–35
 טפסי התחברות, 399
 מבני בקרת לולאה
 לולאות do.. while, 43–44
 לולאות for, 44–45
 לולאות while, 42
 לולאות, בקרה, 43
 הפיכה לאותיות קטנות (lowercasing) (PEAR DB), 186
 lzf, 599

M ##

מאקרואים (macros)
 מאקרואים להצהרת פונקציות, 495

פרמטרים של מאקרו STD_PHP_INI_ENTRY**, 503**
 TSRM**, 504–505**
 VCWD**, 496**
 ארגומנטים של מאקרו ZEND_FETCH_RESOURCE**, 498**
 פרמטרים של מאקרו ZEND_INIT_MODULE_GLOBALS**, 502**
 מאקרואים לגישה ל-zval**, 499–501**
 גרשי קסם (magic quotes)**, 198**
 magic_quotes_gpc**, 198**
 magic_quotes_runtime**, 198**
 Mail**, 569**
 Mail_IMAP**, 569**
 Mail_Mbox**, 569**
 Mail_Mime**, 570**
 Mail_Queue**, 570**
 mailparse**, 569**
 POP3**, 570**
 vpopmail**, 571**
 maintainers**, 418**
 גרסה ראשית לעומת משנית לעומת רמת תיקון**, 349
 ניהול משתנים
 empty**, 17()**
 isset**, 16–17()**
 unset**, 17()**
 מניפולציה של גרפיקה
 תרשימי עמודות, 320–325
 טפסי שליחה חסיני בוטים, 315–320
 exif**, 326–329**
 פורמט סימון (markup format),
 HTML_Template_Flexy**, 390**
 פונקציות התאמה (matching functions), 293, 295
 מתמטיקה (Math)
 Math_Baseex**, 571**
 Math_Complex**, 571**
 Math_Fibonacci**, 571**
 Math_Histogram**, 571**
 Math_Integer**, 572**
 Math_Matrix**, 572**
 Math_Numerical_RootFinding**, 572**
 Math_Quaternion**, 573**
 Math_RPN**, 573**
 Math_Stats**, 573**
 Math_TrigOp**, 573**
 Math_Vector**, 574**
 mb_convert_encoding**, 333()**
 mb_strlen**, 335–337**
 mb_strpos**, 340()**
 mb_substitute_character**, 332()**
 mb_substr**, 340()**

mbstring**, 330 הרחבת**
 mbstring**, 331** של **phpinfo()** פלט
 mbstring**, 333()**
 PEAR**, 430** של **MD5 checksum**
 md5sum**, 421 תכונת**
 MDB**, 538–539**
 MDB_QueryTool**, 541**
 MDB2**, 539–540**
 mdbtools**, 541**
 **, (member variables) מחלקה משתני
 PCS**), 406** שיום סמלים
 memcache**, 589**
 זיכרון
 מנהל זיכרון חדש, 11
 כתיבת הרחבות, 489
 Message**, 544**
 XML RPC**), 244–245 הודעות**
 metacharacters**), 280, 283, 285 תווים מיוחדים**
 (methods) מתודות
 toString**, 76–77__()**
 abstract**), 6, 72–73** מתודות מופשטות
 this\$, 59, 61–62** גישה באמצעות
 call**, 253()**
 connect**, 62**
 createAggregate**, 174()**
 createDbConnection**, 62()**
 draw**, 72()**
 final**), 75–76** מתודות סופיות
 getAll**, 189()**
 getAssoc**, 188–189()**
 getAttribute**, 228()**
 getCol**, 188()**
 getName**, 56()**
 getOne**, 188()**
 getRow**, 188()**
 hash**, 125()**
 E_STRICT**, 442–443** מתודות מורשות
 outputObject**, 388()**
 overloading**), 85–87** דריסת מתודות
 POST**, 115** מתודת
 raiseError**, 207**
 service**, 255()**
 setDbConnection**, 62()**
 setName**, 56()**
 singleQuery**, 168()**
 5 מתודות סטטיות
 this\$, 64–65** גישה באמצעות
 PCS**), 405** שיום סמלים

micro-benchmarks**, 477–479) **ביצועים**
microtime**, 302() **טעויות בהגנה על קלט משתמש****, 117
סקריפטים חוצי-אתרים (XSS**), 118–119
משתנים גלובליים, 118–117
הזרקת SQL**, 119–120
mktime**, 303() **משנים** (modifiers**), 289, 293
שינוי אלמנטים במערך**, 25
module_number**, 494
mono**, 590
move_uploaded_file**, 142() **העברת קבצים באמצעות פונקציית rename****, 278()
MP3_ID**, 547
mqseries**, 574
MySQL**, 155–156) **ריבוי הצהרות**
multi-byte strings**), 330) **מחרוזות מרובות-בתים**
מערכים רב-ממדיים. **ראה מערכים מקוננים**
MySQL**, 149
BLOB
BLOB**, 159 **הכנסת נתוני**
BLOB**, 159–160 **שליפת נתוני**
buffered), 153) **שאילתות מאוחסנות**
תאימות, 445
חיבורים, 153–151
נתוני דוגמה, 151
מצבי שליפה (fetching modes), 156
ריבוי הצהרות, 156–155
ממשק PHP**, 150–151
הצהרות מוכנות (prepared statements), 156–158
שאילתות, 155–154
סקלריות, 150
מהירות, 150
נקודות חוזק וחולשה, 150
שאילתות לא מאוחסנות (unbuffered), 154
MySQLi**, 10
פונקציות חיבור mysqli**, 152
פונקציות שליפה mysqli**, 156
פונקציות שאילתה mysqli**, 154
קבועי mysqli_options**, 153

N ##

name**, 624
תכונת name**, 421–423, 428
אלמנט name**, 417

מוסכמות שיום, סמלים ב-403**, PCS**
 מחלקות, 404
 קבועים, 403
 פונקציות, 404
 משתנים גלובליים, 404
 משתני מחלקה, 406
 מתודות, 405
 אופרטורים של שלילה, 36
 מערכים מקוננים, גישה, 26
 Net_CheckIP**, 575**
 Net_Curl**, 575**
 Net_Cyrus**, 575**
 Net_Dict**, 575**
 Net_Dig**, 575**
 Net_DIME**, 576**
 Net_DNS**, 576**
 Net_Finger**, 576**
 Net_FTP**, 576**
 Net_GameServerQuery**, 576**
 Net_Geo**, 577**
 Net_Gopher**, 577**
 Net_Ident**, 577**
 Net_IMAP**, 577**
 Net_IPv4**, 577**
 Net_IPv6**, 578**
 Net_IRC**, 578**
 Net_LDAP**, 578**
 Net_LMTP**, 579**
 Net_NNTP**, 579**
 Net_Ping**, 579**
 Net_POP3**, 579**
 Net_Portscan**, 579**
 Net_Server**, 580**
 Net_Sieve**, 580**
 Net_SmartIRC**, 580–581**
 Net_SMTP**, 582**
 Net_Socket**, 582**
 Net_Traceroute**, 582**
 Net_URL**, 582**
 Net_UserAgent_Detect**, 582**
 Net_UserAgent_Mobile**, 583**
 Net_Whois**, 583**
 netools**, 574**
 רשת (networking),
 cvsclient**, 574**
 cyrus**, 574**
 kadm5**, 574**
 mqseries**, 574**

* (רשימת ספריות Net דומה למופיע לעיל) *

opendirectory**, 583**
tcpwrap**, 584**
uuid**, 584**
yaz**, 584**
new**, 57–58 מילת מפתח
nextId**, 185()**
nobuild**, 365--**
nocompress**, 365--**
notes**, 420 אלמנט
null**, 23**
מספרים
מספרי נקודה צפה, 19
Number_Roman**, 584**
Number_Words**, 585**
אופרטורים נומריים**, 32

○ ##

o-**, 366**
Object Cloning)**, 4) שכפול עצמים
מודל אובייקטים** (433**), (PHP 5**
תכונות מונחות עצמים**, 3–7
אובייקטים (55), objects**
המרה (casting) (מצב תאימות), 435–436
מחלקות, 56
שכפול, 4, 66–67
השוואה (מצב תאימות), 436–437
יצירה אוטומטית, 441
איטרטורים, 89–94
העברה לפונקציות, 433–435
אובייקטי **SimpleXML**
דפדוף, 233–234
יצירה, 232–233
אחסון, 234
תבנית צופה (101–103), observer pattern**
השגת PEAR DB**, 176
oci8**, 541**
odbt**, 541**
oggvorbis**, 595**
OLE**, 596**
סקריפט אחד לכל פונקציה**, 144
סקריפט אחד לכולם**, 143
onlyreqdeps**, 366--**
מודל OO (מונחה עצמים)**, 1–2
תבניות עיצוב, 94

תבניות מפעל (**97–101, factory**),
 תבניות צופה (**101–103, observer**),
 תבניות סינגלטון (**97–98, singleton**),
 תבניות אסטרטגיה (**95–96, strategy**),
 לעומת קוד פרוצדורלי, 480–481
 **יישומי OO, 70
 **מעטפות OO לפונקציות מובנות, 456
 **תכנות מונחה עצמים (55, OOP)
 פולימורפיזם, 67–69
 **מטמון אופקוד (470, opcode cache)
 **openal, 595
 **opendirectory, 583
 **תגיות פתיחה, 221
 **OpenSSL, 272
 **תלות במערכת הפעלה (425, PEAR)
 **הבדלי מערכות הפעלה, שגיאות ניידות, 197
 **אופרטורים, 31
 אופרטורים של השמה, 32–33
 אופרטורים בינאריים, 31–32
 אופרטורים על סיביות (**35, bitwise)
 אופרטורים של המרה (**38, cast)
 אופרטורים של השוואה, 33–34
 אופרטורים של קידום/פיחות, 37–38
 instanceof, 4, 71
 אופרטורים לוגיים, 34–35
 אופרטורים של שלילה, 36
 אופרטור השתקה, 39
 אופרטור טרנרי, 39
 אופרטורים אונאריים, 36
 **אופטימיזציה, 459
 קוד, 477
 מיקרו-מבחני ביצועים, 477–479
 OO לעומת קוד פרוצדורלי, 480–481
 כתיבה מחדש ב-C, 479
 **תכונת optional, 424
 **תלויות אופציונליות, 366
 התקנת חבילות איתן, 366
 **PEAR, 426
 **אפשרויות
 **CLI לעומת CGI, 509
 הגדרת **short_open_tags ב-14, INI
 **מטמון פלט (456, output caching)
 **Cache_Lite, 400–401
 **מתודת () 388, outputObject
 **דריסה (85, overloading)
 תחביר גישה למערך, 88–89
 איטרציה, 90
 מתודות, 85–87

P ##

צינורות (pipes) , 266
 package , 624, 626
 אלמנט (package) , 417
 מידע על חבילה (package.xml) , 417–419
 package.xml , 416, 431
 מידע על חבילה, 417–419
 מידע על גרסת הפצה (**release** , 419–422)
 חבילות (packages)
 בנייה
 דוגמת PEAR (**HelloWorld** , 411–414)
 בדיקות רגרסיה (**PEAR** , 416)
 ארכיוני tar (**PEAR** , 414)
 אימות (**PEAR** , 414–415)
 Cache_Lite** , 399–401
 התקנה, 354
 עם תלויות אופציונליות, 366
 עם פקודת pear** , 354–357
 PEAR** , 346
 שחרור חבילות, 428
 תהליך השחרור של PEAR** , 430
 אריזה (packaging) , אורז החבילות של PEAR** , 430
 MD5 checksum** , 430
 package.xml** , 431
 ניתוח מקור, 430
 ארכיוני tar** , 431
 Pager** , 560
 Pager_Sliding** , 560
 panda** , 599
 Paradox** , 542
 התקנות PEAR מקבילות , 362
 param** , 626–627
 פרמטרים
 רמזי טיפוס מחלקה (**type hints**) בפרמטרי פונקציה, 82–83
 הצהרה על פרמטרי פונקציה, 52
 parent** , 70
 שגיאות ניתוח (parse errors) , 192–194
 ניתוח (parsing)
 אפשרויות שורת פקודה, 512–515
 פורמטי תאריך, 313–314
 XML** , 222
 DOM** , 226–229
 SAX** , 222–226

מטמון חלקי, 475
 העברה
 אובייקטים לפונקציות, 433–435
 לפי ייחוס (**), 52
 לפי ערך, 52
 קבצי סיסמה, 393–394, Auth**
 סיסמאות, הגנה על סקריפטים, 127–129
 תחביר תבנית, 280
 תבניות (279, patterns). ראה גם תבניות עיצוב
 תשלום (payment)
 cybercash**, 585**
 cybermut**, 585**
 Payment_Clieop**, 586**
 Payment_DTA**, 586**
 Payment_Process**, 586**
 splus**, 586**
 TCLink**, 586**
 פלט () phpinfo של 279, PCRE
 תקן הקידוד של PEAR (ר"ת 403, PCS)
 הזחה (406–407, indentation)
 שיום סמלים, 403
 PEAR (PHP Extension and Application Repository**, 200–201, 345, 587)**
 Auth**, 392, 398**
 DB ונתוני משתמש, 394–396
 קבצי סיסמה, 393–394
 שיקולי סקלביליות, 397–398
 שיקולי אבטחה, 396–397
 בניית חבילות, 411–414
 מטמון, 399–401
 סביבת 408–410, CLI**
 Crypt_HMAC**, 124–127**
 תלויות, 423–424
 טיפול בשגיאות, 206–207, 212, 411
 תגיות פתיחה, 113
 מנהל חבילות (354, Installer)
 פקודת 354–357, 363, pear
 PEAR DB**, 176**
 שגיאות מופשטות, 186
 חיבורים, 178–179
 מתודות נוחות, 188
 ביצוע שאילתות, 180
 שליפת תוצאות, 182
 נידות, 185
 רצפים (**), 184–185, sequences**
 PECL (PHP Extension Community Library)**, 345**
 PECL_Gen**, 601**
 ביצועים (449, 456, performance)
 APC**, 470**

benchmarking**), 457**), ביצועים
 caching**), 451–453**), מטמון
 APD**), 461**), עם (**profiling**), פרופילינג
 Zdebug**), 465**), פרופילינג עם
 Zend Studio**), 459–461**), של **Profiler**), פרופילינג עם
 state**), 450**), מצב
 ZPS**), 470**),
 PHP),
 apd**), 588**),
 bcompiler**), 589**),
 HTML**), 112–114**), הטמעת קוד ב-
 ffi**), 589**),
 Inline_C**), 589**),
 memcache**), 589**),
 mono**), 590**),
 perl**), 590**),
 PHP_CompatInfo**), 591**),
 PHP_Fork**), 591**),
 PHP_Parser**), 592**),
 PHPDoc**), 590**),
 PHPUnit**), 590**),
 python**), 592**),
 כלים),
 crack**), 600**),
 fann**), 601**),
 PECL_Gen**), 601**),
 PhpDocumentor**), 601**),
 SPL**), 602**),
 Valkyrie**), 602**),
 2, מגבלות, **PHP 3**),
 433, 2, **PHP 4**), מעבר ל- **PHP 5**), מצב תאימות,
 PHP 5**), 433**),
 PHP**), 201**), שגיאות
 204, מטפלי שגיאות מותאמים אישית,
 202–201, רמות שגיאה,
 204, 202, דיווח שגיאות,
 206–205, השתקת שגיאות,
 PHP ממשקי),
 MySQL**), 150–151**),
 SQLite**), 162**),
 PHP**), 134–137, 396 של (sessions), סשנים
 phpDocumentor**), 601, 613, 633–640, כלי
 polymorphism**), 67–69, הפולימורפיזם
 portability**), 200, ניידות
 200–197, שגיאות ניידות,
 201–200, כללי ניידות,
 prepared statements**), 156, שאילתות מוכנות
 OO**), 480–481, קוד פרוצדורלי לעומת

proc_open**, 265() אובייקט**
profiling)**, 459 פרופילינג**
120 ,**סקריפטים**
provides**, 422 אלמנט**
public**, **E_STRICT**, 441**

Q-R

שאילתות (queries)
MySQL**, 153–155**
PEAR DB)**, 180–182**
SQLite)**, 162–165**
39 ,**סימן שאלה (?)**
מירכאות
20–19 , מחרוזות, (" ")
20 , מחרוזות, (')
raiseError**, 207, 209**
25 ,**קריאת ערכי מערך**
reflection)**, 103 רפלקציה**
107–106 , דוגמאות
105–103 ,**API של רפלקציה**
regular expressions)**, 279 ביטויים רגולריים**
301–293 , פונקציות
280–279 , תחביר
PEAR)**, 428 שחרור חבילות**
278–277 ,**ניקוי/הסרת קבצים**
rename)**, 278 שיום מחדש**
204 , 202 ,**דיווח שגיאות**
XML RPC)**, 245–246 בקשות**
resources)**, 493–499 משאבים**
XML RPC)**, 246 תגובות**
return**, 627**
490 ,**החזרת ערכים מפונקציות**
51 , לפי ייחוס
50 , לפי ערך
RSS**, 235–236**

S

SAPI (Server API)**, 507**
SAX**, 9, 222–226**
סקלבליות (scalability)
Auth**, 397–398**
MySQL**, 150**

סקריפטים, הגנה, 120
 אבטחה (security)
 Auth**, 396–397**
 הגנה על קלט משתמש, 120–117
 see**, 627**
 self**, 70**
 הפרדת לוגיקה מפריסה, 146–144
 רצפים (sequences)
 PEAR DB**, 184–185**
 שרתים (XML RPC)**, 250–252
 סשנים (sessions), 134–137, 450
 תגיות קצרות (short tags)**, 113
 אופרטור השתקה, 39
 SimpleXML **, 9–10, 222, 231–234**
 SOAP**, 10, 252–259**
 SQLite**, 160–176**
 מצב (state)**, 450
 סטטי (static)
 חברים סטטיים, 5, 62
 מתודות סטטיות, 5, 64–65
 מאפיינים סטטיים, 62–64
 משתנים סטטיים, 53
 תזרימים (streams), 261–276
 מחרוזות (strings), 19–22, 38
 מבנים (structures), 596–597
 דריסת מתודות/מאפיינים (overloading), 85–87
 תחביר (syntax), 88, 279–280

T–Z

תגיות (tags), 113, 221, 615–632
 מערכות תבניות (template systems), 383–392
 בדיקות (testing), 416, 430, 459, 501
 טקסט (text), 325, 599–600
 זמן (time), 301–314
 אזורי זמן, 312
 פורמט ISO 8601**, 309**
 כלים (tools). ראה גם כלי שורת פקודה
 טרנזקציות (SQLite), 164–165
 עצים (trees), **DOM**, 229–231
 טריגרים (triggers), **SQLite**, 165
 טיפול בשגיאות (try/catch), 216–218
 TSRM**, 504–505**
 UDFs (פונקציות מוגדרות משתמש), 165–168
 משתנים (variables)
 גלובליים, 18, 117

סטטיים, 53
ניהול, 16–17
גרסאות (versioning), 347–349
ווב (Web), שירותי ווב, 243
XML, 219–241
Zend, 643–653
Zend Engine, 1
Zend Studio, 644–653
zval, 499–501